第一单元 ARM和Cortex-M3/M4

本单元参考资料：

- 《The Definitive Guide to Arm Cortex-M3 and Cortex-M4 Processors》, Joseph Yiu, 3rd Ed.
- 《ARM Cortex-M3/M4 权威指南》，吴常玉等译，清华出版社，2015
- Chapter 1,3-8

第1章 ARM和Cortex-M3/M4

1.1 嵌入式系统和ARM（ref. chapter 1）

1.2 ARM Cortex-M3/M4处理器概述（ref. chapter 3）

1.3 Cortex-M3/M4编程模型（ref. chapter 4）

*1.4 Cortex-M3/M4指令系统（ref. chapter 5）

1.5 存储系统（ref. chapter 6）

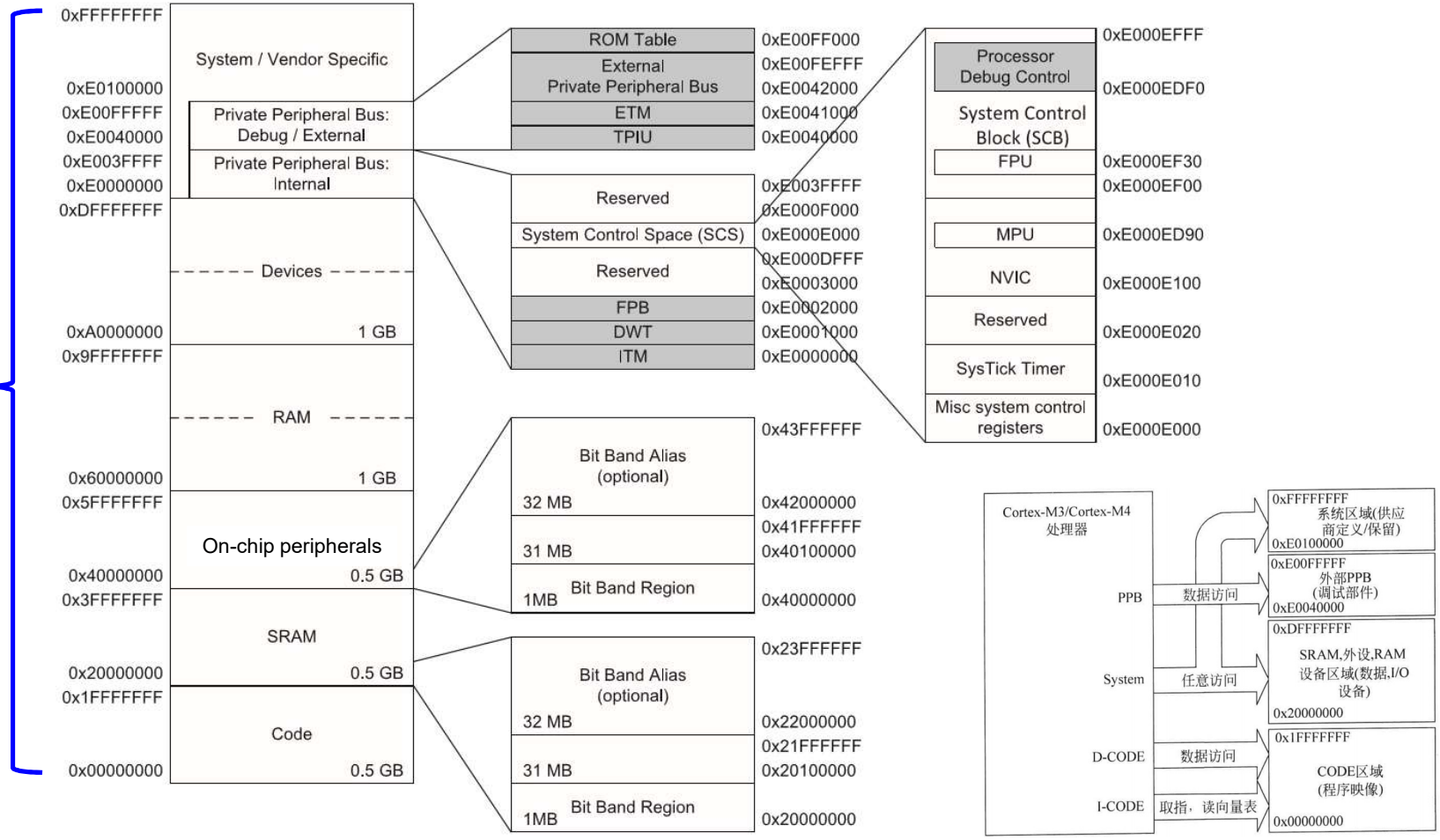
1.6 异常和中断（ref. chapter 7-8）

1.5 Cortex-M3/M4存储系统

- 存储系统功能概述
  - 存储器映射是预定义的，并且还规定好了哪个位置使用哪条总线
  - 支持位带（Bit-band）操作，提供对字数据单一比特的原子操作
  - 支持存储器系统的大小端配置
  - 支持非对齐访问和互斥访问
  - 支持四种存储器的访问属性
  - 可选的存储保护单元(MPU)，支持存储器的属性和访问权限设置

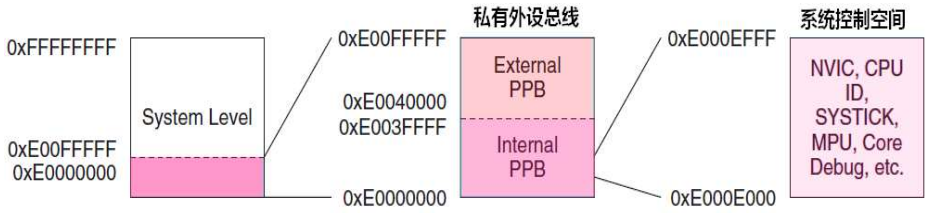
1.5.1 Cortex-M3/M4存储器映射

4GB的地址空间



预定义的存储器映射

- Cortex-M3/M4处理器4GB地址空间有一个固定的存储器映射
  - Code：512MB。代码存储区，包括中断向量表。通过I-Code、D-Code访问
  - SRAM：512MB。片上SRAM存储区，含32MB位带别名。通过系统总线访问，允许执行程序
  - 片上外设：512MB。由片上外设使用，含32M位带别名。通过系统总线访问
  - 外部 RAM：1 GB。用于扩展外部存储器，允许执行程序。通过系统总线访问
  - 外部设备：1 GB。用于扩展片外外设，不允许执行程序。通过系统总线访问
  - 系统级区域：0.5 GB。包括：
    - 内部私有外设总线 (PPB)区：系统控制组件
    - 外部私有外设总线 (PPB)区：可选调试组件
    - 供应商特定区：供应商指定组件



1.5.2 存储器的端模式

- 存储器的端模式
  - Cortex-M3/M4 支持小端和大端模式
  - 端模式在复位时被设置，当处理器退出复位后不能再被改变
  - 取指令总是在小端模式，配置系统控制空间(SCS)和外部私有总线(PPB)区的数据访问也永远使用小端模式
  - 通过使用指令可以方便地完成小端模式和大端模式的转换
  - 小端模式是ARM处理器的默认存储格式

存储器的端模式

存储器中的数据

Table 6.3 Little Endian Memory View

Address	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x0003 – 0x0000	Byte – 0x3	Byte – 0x2	Byte – 0x1	Byte – 0x0
…				
0x1003 – 0x1000	Byte – 0x1003	Byte – 0x1002	Byte – 0x1001	Byte – 0x1000
0x1007 – 0x1004	Byte – 0x1007	Byte – 0x1006	Byte – 0x1005	Byte – 0x1004
…				
…	Byte – 4xN+3	Byte – 4xN+2	Byte – 4xN+1	Byte – 4xN

Table 6.4 Big Endian Memory View

Address	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x0003 – 0x0000	Byte – 0x0	Byte – 0x1	Byte – 0x2	Byte – 0x3
…				
0x1003 – 0x1000	Byte – 0x1000	Byte – 0x1001	Byte – 0x1002	Byte – 0x1003
0x1007 – 0x1004	Byte – 0x1004	Byte – 0x1005	Byte – 0x1006	Byte – 0x1007
…				
…	Byte – 4xN	Byte – 4xN+1	Byte – 4xN+2	Byte – 4xN+3

AHB总线上的数据

Table 6.7 Little Endian – Data on the AHB Bus

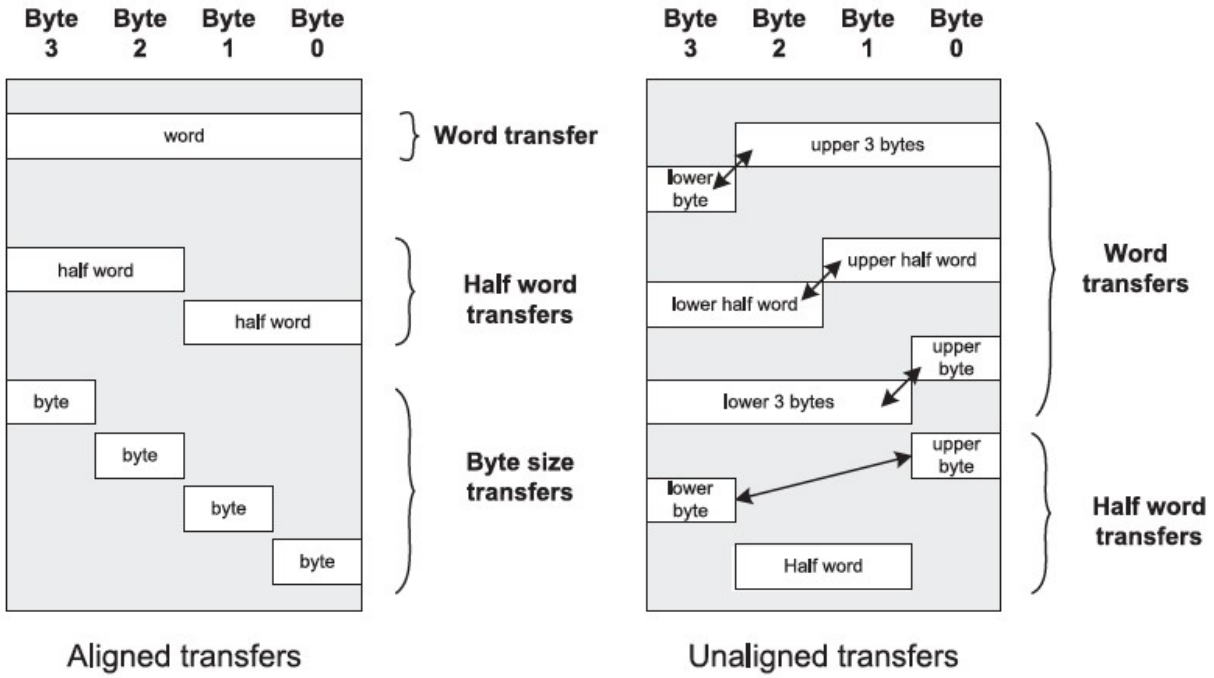
Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit [31:24]	Data bit [23:16]	Data bit [15:8]	Data bit [7:0]
0x1000, half word	-	-	Data bit [15:8]	Data bit [7:0]
0x1002, half word	Data bit [15:8]	Data bit [7:0]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

Table 6.5 The Cortex-M3 and Cortex-M4 (byte-invariant big-endian, BE-8) – Data on the AHB Bus

Address, Size	Bits 31 – 24	Bits 23 – 16	Bits 15 – 8	Bits 7 – 0
0x1000, word	Data bit[7:0]	Data bit [15:8]	Data bit [23:16]	Data bit [31:24]
0x1000, half word	-	-	Data bit [7:0]	Data bit [15:8]
0x1002, half word	Data bit [7:0]	Data bit [15:8]	-	-
0x1000, byte	-	-	-	Data bit [7:0]
0x1001, byte	-	-	Data bit [7:0]	-
0x1002, byte	-	Data bit [7:0]	-	-
0x1003, byte	Data bit [7:0]	-	-	-

1.5.3 对齐和非对齐数据传送

- 非对齐数据传送
  - Cortex-M3/M4 支持单次非对齐数据传送。数据存储器访问可以对齐也可以不对齐。



非对齐数据传送

- 非对齐数据传送的使用限制

  非对齐的数据传送只发生在常规的数据传送指令中，如 LDR/LDRH/LDRSH。 其它指令则不支持，包括：

  1. 多数据的加载/存储指令（LDM/STM）必须对齐
  2. 堆栈操作（PUSH/POP）必须对齐
  3. 互斥访问（LDREX/STREX）必须对齐
  4. 位带操作必须对齐

- 当非对齐传递被使用，事实上它们通过处理器的总线接口单元被转换成许多对齐传递，会需要更多的总线周期。

1.5.4 存储器的位带操作

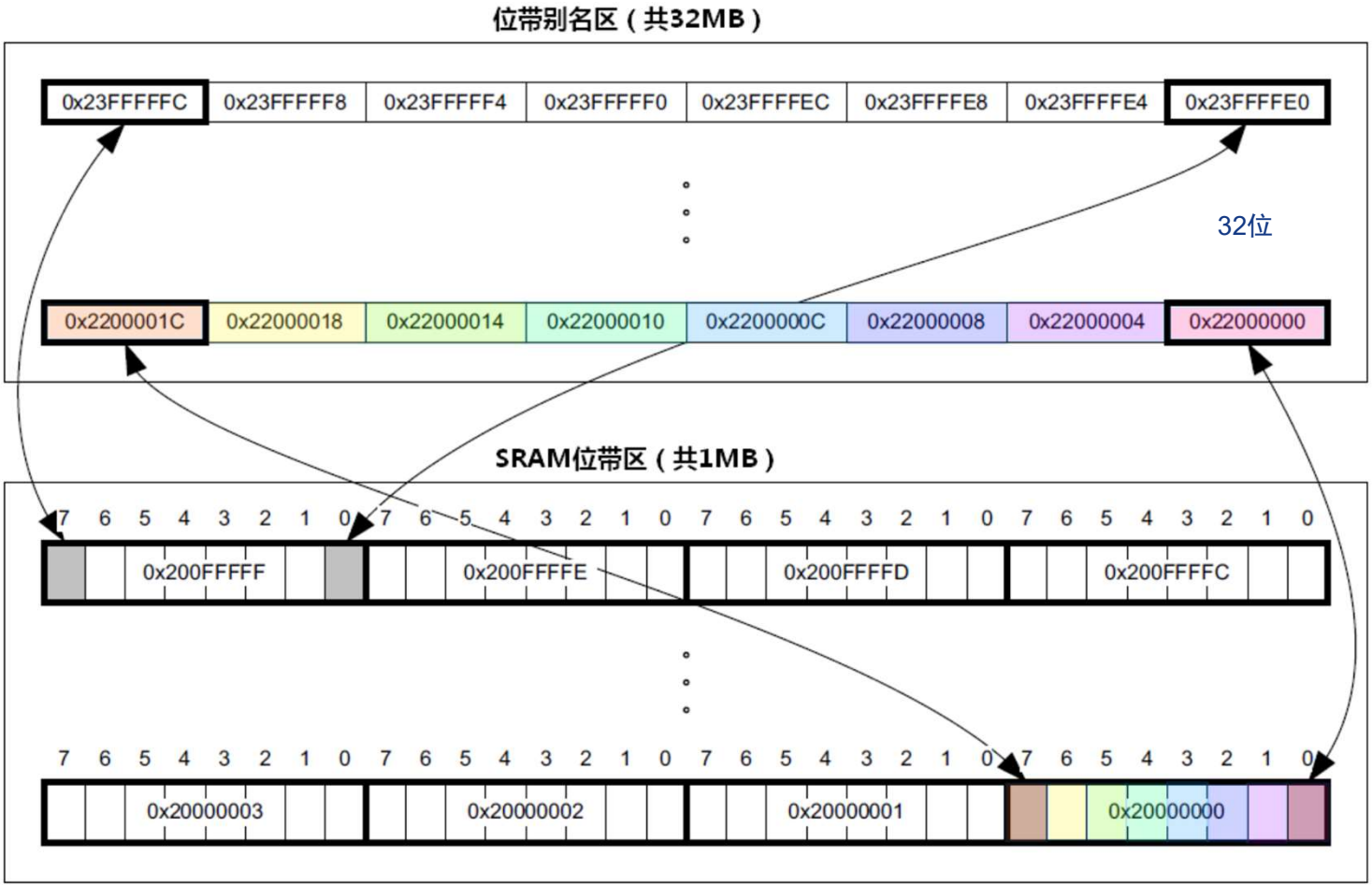
- Bit-Band（位带）操作

  Bit-band操作支持使用普通的Load/Store指令来对单一的位进行读/写访问

- CM3/M4有两个Bit-band区域：
  - SRAM区的第一个1MB
  - 片上外设区的第一个1 MB

它们除了可以像普通的RAM一样使用外，还可以通过一个单独的被称为位带别名区的存储区域来访问。位带别名区把每个比特膨胀成一个32位的字。

位带区与位带别名区的膨胀关系



Cortex-M3/M4位带操作

SRAM 区域的Bit-Band 地址重映射

Bit-Band 区域	别名等效
0x20000000 bit[0]	0x22000000 bit[0]
0x20000000 bit[1]	0x22000004 bit[0]
0x20000000 bit[2]	0x22000008 bit[0]
…	…
0x20000000 bit[31]	0x2200007C bit[0]
0x20000004 bit[0]	0x22000080 bit[0]
…	…
0x20000004 bit[31]	0x220000FC bit[0]
…	…
0x200FFFFC bit[31]	0x23FFFFFC bit[0]

片上外设存储区 Bit-Band 地址重映射

Bit-Band 区域	别名等效
0x40000000 bit[0]	0x42000000 bit[0]
0x40000000 bit[1]	0x42000004 bit[0]
0x40000000 bit[2]	0x42000008 bit[0]
…	…
0x40000000 bit[31]	0x4200007C bit[0]
0x40000004 bit[0]	0x42000080 bit[0]
…	…
0x40000004 bit[31]	0x420000FC bit[0]
…	…
0x400FFFFC bit[31]	0x43FFFFFC bit[0]

Cortex-M3/M4位带操作

- 位地址与位带别名地址的对应关系

1. 0x20000000~0x200FFFFF (SRAM, 1 MB)

   设SRAM位带区的某个比特所在字节地址为A，位序号n (0≤n≤7)，则对应的位带别名地址为：

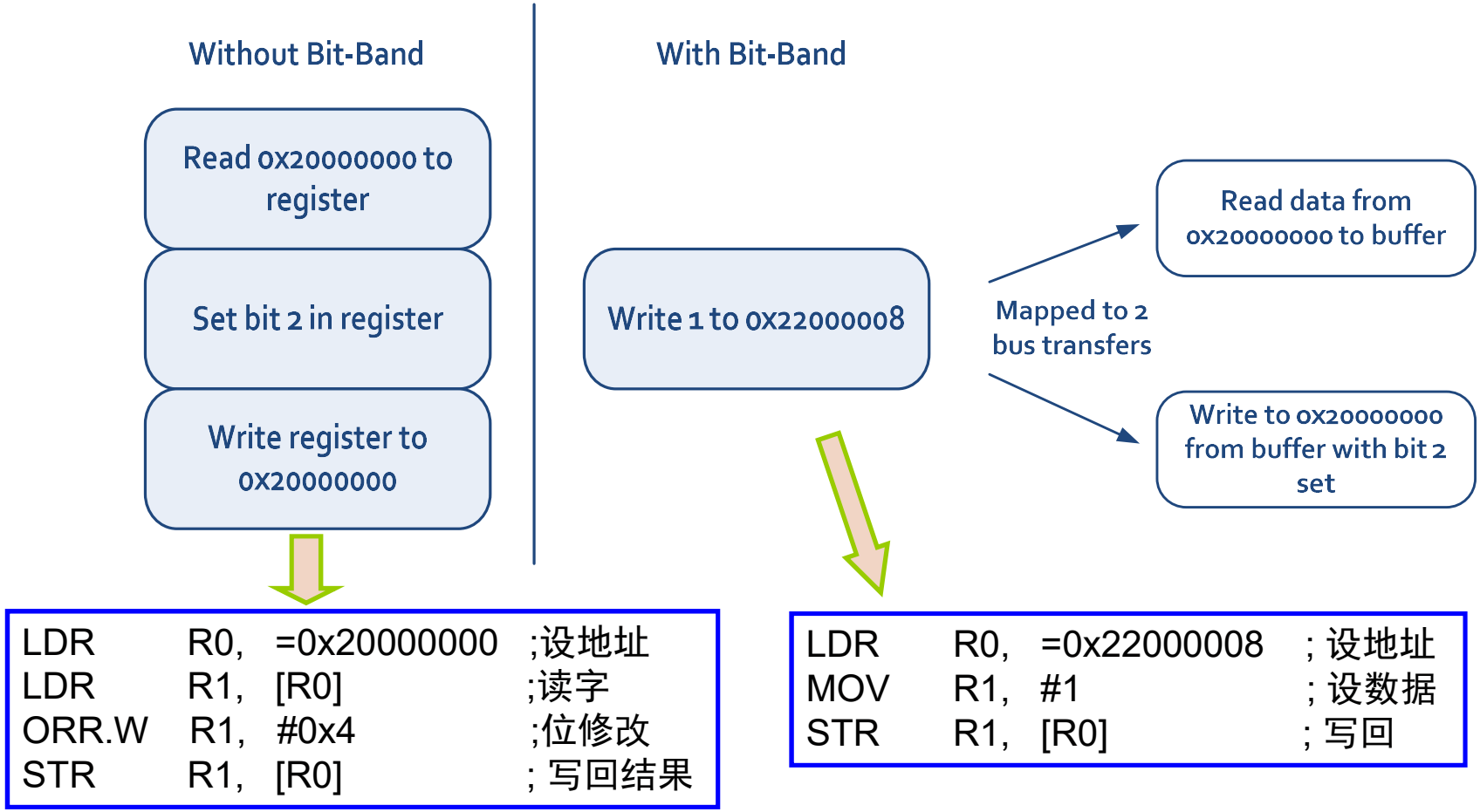
   AliasAddr = 0x2200.0000+(A-0x20000000)×32+n×4

2. 0x40000000~0x400FFFFF (片上外设, 1 MB)

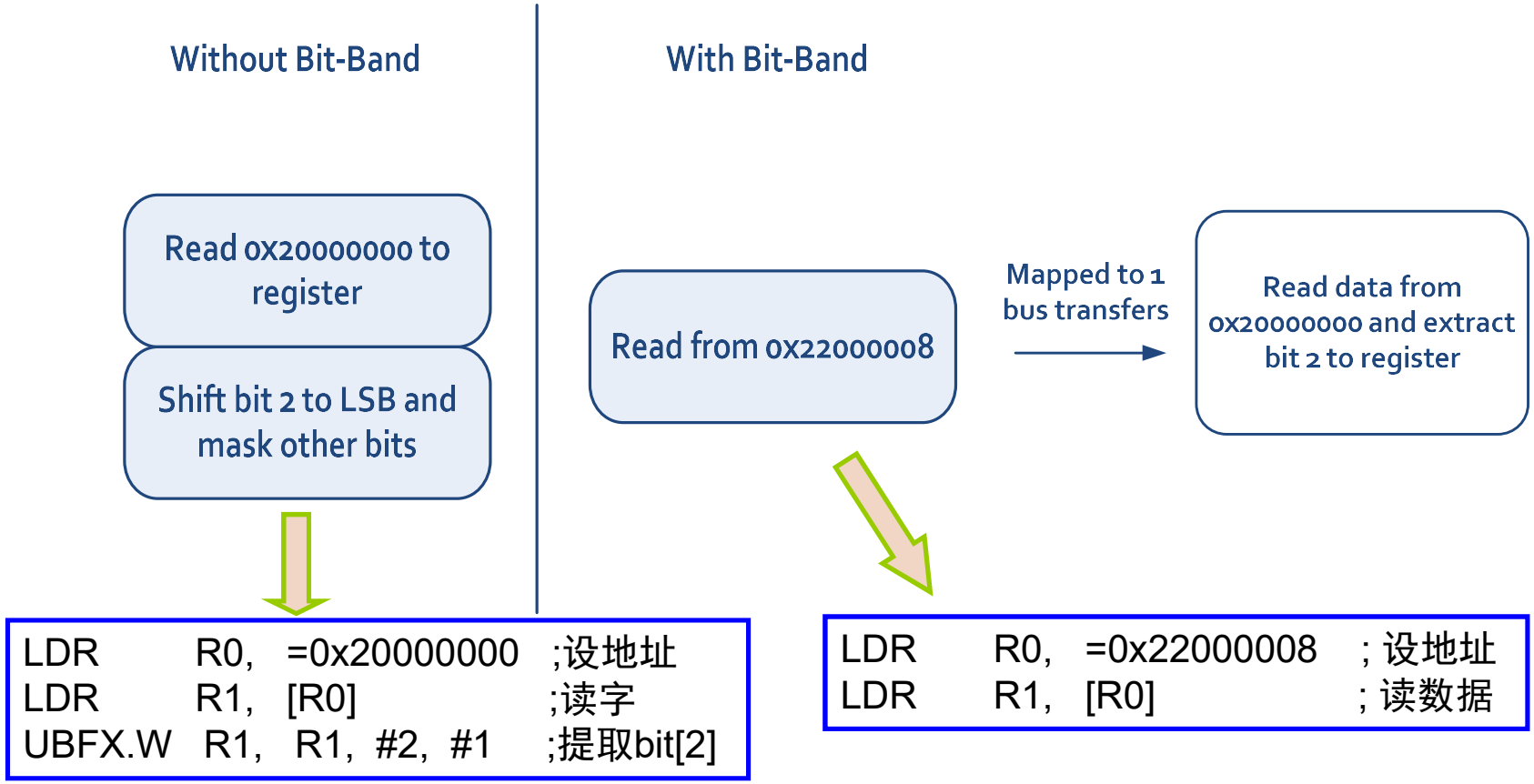
   设片上外设位带区的某个比特所在字节地址为A，位序号n (0≤n≤7)，则对应的位带别名地址为：

   AliasAddr = 0x4200.0000+(A-0x40000000)×32+n×4

- 例：设置地址0x20000000中bit2的操作：写操作



- 例：读取地址0x20000000中bit2的操作：读操作



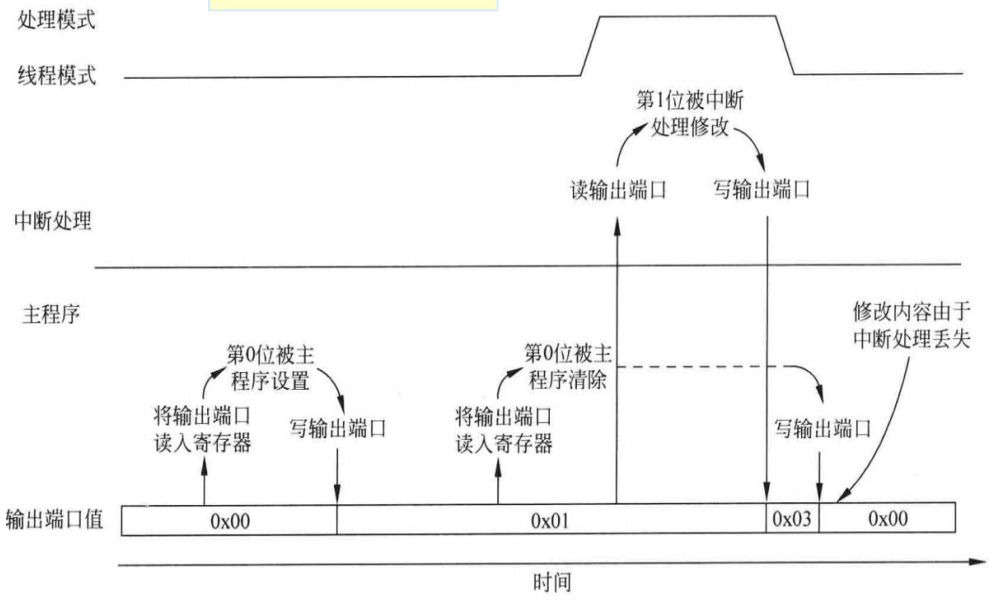
Cortex-M3/M4位带操作

- 在位带操作中，对于读操作，读取位带地址的一个字，再把所选择的位右移到LSB，并返回LSB。
- 对于写操作，所写的位被左移到对应的位序号处，然后执行一个原子的READ-MODIFY-WRITE 过程。
- 例如：假设地址 0x20000000 处的值为 0x3355AACC；
  - 读地址 0x22000008，这个读访问被重新映射到读访问 0x20000000，并提取位2，返回的值为1 (0x3355AACC的bit[2])；
  - 写 0x0 到地址 0x22000008，这个写访问被重新映射到对 0x20000000 的“读-改-写”原子操作，把位2清0。
  - 现在读地址 0x20000000，得到的返回值为 0x3355AAC8 (bit[2] 被清零)

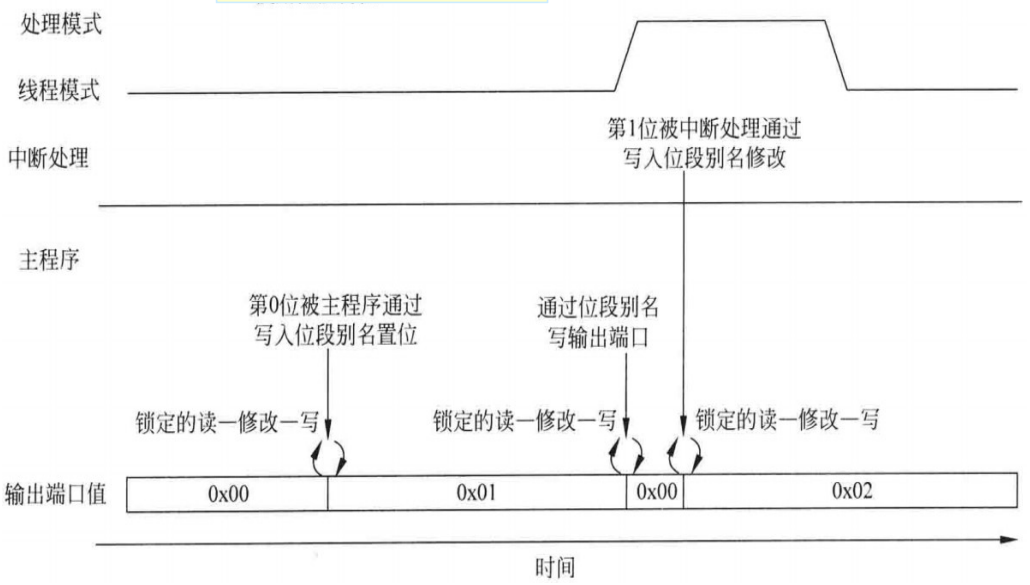
位带操作的优势

- ✓ 用更少的指令完成更快的位操作
- ✓ 互斥的读/写操作（由硬件完成）
  - ■ 防止中断程序修改同一个共享内存单元引起的数据丢失

无位带的操作



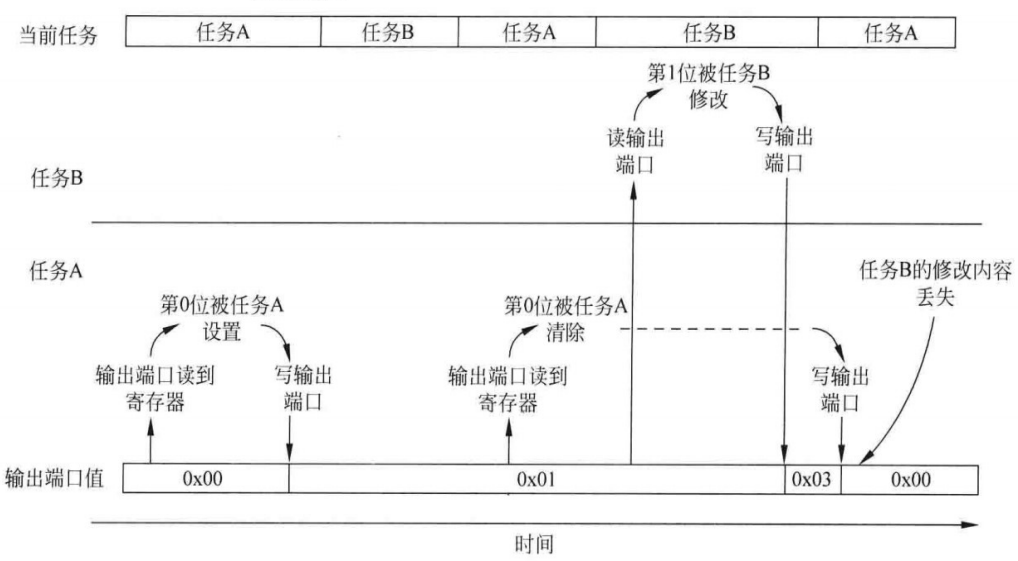
使用位带的操作



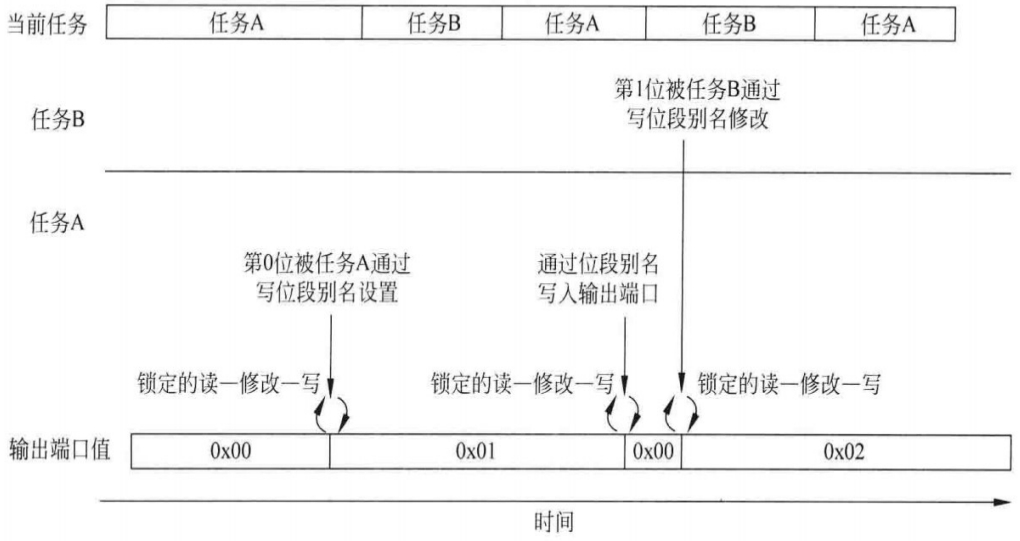
位带操作的优势

- ✓ 用更少的指令完成更快的位操作
- ✓ 互斥的读/写操作（由硬件完成）
  - ■ 防止不同的任务修改同一个共享内存单元引起的数据丢失

无位带的操作

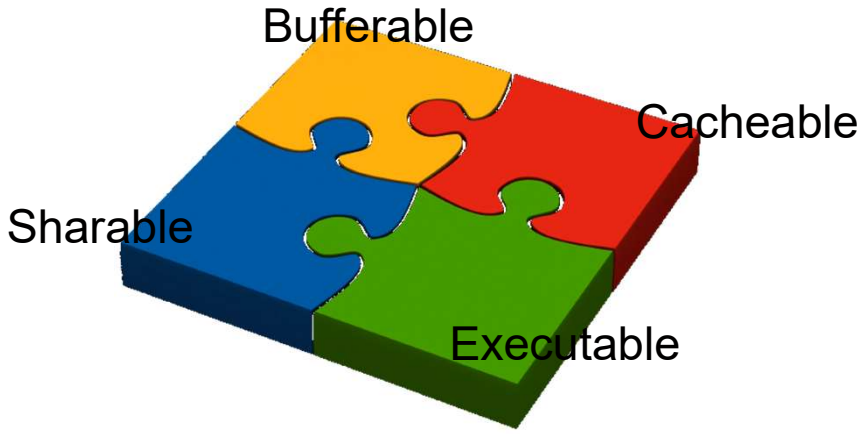


使用位带的操作



1.5.5 存储器访问属性

- 存储器访问属性包括：
  - 可否缓冲（Bufferable）
  - 可否缓存（Cacheable）
  - 可否执行（Executable）
  - 可否共享（Sharable）



存储器映射定义了一个缺省的存储器访问属性。（若MPU使能，则可以通过它重新配置不同的存储区属性，并覆盖缺省的访问属性）

缺省的存储器访问属性

区域	地址	用于	缓存属性	执行、缓冲属性
代码存储区	0x00000000– 0x1FFFFFFF		WT (写通)	可执行, 不可缓冲
SRAM 存储区域	0x20000000– 0x3FFFFFFF	片上RAM	WB-WA (写回, 写分配)	可执行, 可缓冲
片上外设区域	0x40000000– 0x5FFFFFFF	片上外设	不可缓存	不可执行, 可缓冲
外部 RAM 区域 (前半段)	0x60000000– 0x7FFFFFFF	片上或片外存储器	WB-WA (写回, 写分配)	可执行, 可缓冲
外部 RAM 区域 (后半段)	0x80000000– 0x9FFFFFFF	片上或片外存储器	WT (写通)	可执行, 不可缓冲
外部设备 (前半段)	0xA0000000– 0xBFFFFFFF	外部设备和/或共享内存	不可缓存	不可执行,可缓冲
外部设备 (后半段)	0xC0000000– 0xDFFFFFFF	外部设备和/或共享内存	不可缓存	不可执行,可缓冲
系统区域	0xE0000000– 0xFFFFFFFF	私有外设 供应商特定设备	不可缓存	不可执行,不可缓冲

1.5.6 存储器访问权限

- 存储器的缺省访问权限

Cortex-M3/M4 的内存映射具有一个缺省的存储器访问权限配置，能防止用户代码访问系统控制存储空间。

缺省的内存访问权限在下列两种情况之一生效：

- 1. 没有 MPU
- 2. MPU 存在但是被禁用

当一个用户访问被阻止时，会立即产生一个总线fault异常。

缺省存储器访问权限

存储器区域	地址范围	用户级的访问权限
Vendor specific	0xE0100000–0xFFFFFFFF	完全访问
ETM	0xE0041000–0xE0041FFF	阻止访问; 访问会引发总线错误
TPIU	0xE0040000–0xE0040FFF	阻止访问; 访问会引发总线错误
Internal PPB	0xE000F000–0xE003FFFF	阻止访问; 访问会引发总线错误
NVIC	0xE000E000–0xE000EFFF	阻止访问; 访问会引发总线错误
FPB	0xE0002000–0xE0003FFF	阻止访问; 访问会引发总线错误
DWT	0xE0001000–0xE0001FFF	阻止访问; 访问会引发总线错误
ITM	0xE0000000–0xE0000FFF	允许读取; 写忽略（除了用户访问允许时的stimulus端口）
External Ram	0x60000000–0x9FFFFFFF	完全访问
Peripheral	0x40000000–0x5FFFFFFF	完全访问
SRAM	0x20000000–0x3FFFFFFF	完全访问
Code	0x00000000–0x1FFFFFFF	完全访问

1.5.7 存储器的互斥访问

- 互斥访问
  - 在传统的ARM处理器中，SWP 指令是一条互锁传送指令，将一个内存单元的内容与一个寄存器的内容互换，即将内存的读取和写入作为一个原子操作，常用于信号量操作。
  - 在 ARM V7 架构中，存储器读和存储器写由相互独立的总线执行。由于互锁的读/写传送必须要位于同一条总线，因此SWP指令操作的原子性便无法保证了。为此，在Cortex-M中互锁传送被互斥访问所取代。

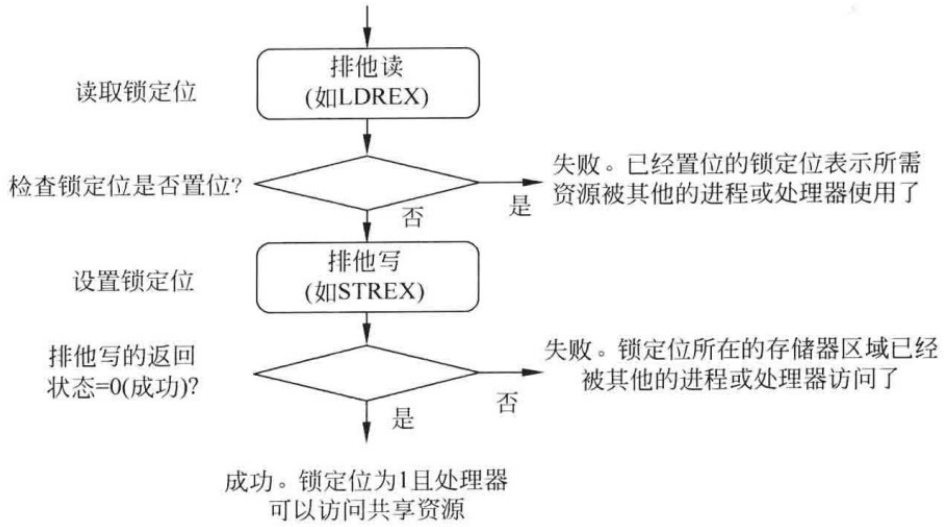
➢ 什么是信号量（Semaphore）？

信号量常用于共享资源的分配。 当一个资源正在被一个进程使用时，它的信号量被这个进程锁定，在该信号量被释放前其它进程不能不得再访问它。

互斥访问

- 互斥访问标记信号量状态，被锁定的互斥体地址只允许其他总线主设备进行读访问但不能进行写访问。互斥访问指令包括:

1. LDREX / STREX (字)
2. LDREXB / STREXB (字节)
3. LDREXH / STREXH (半字)

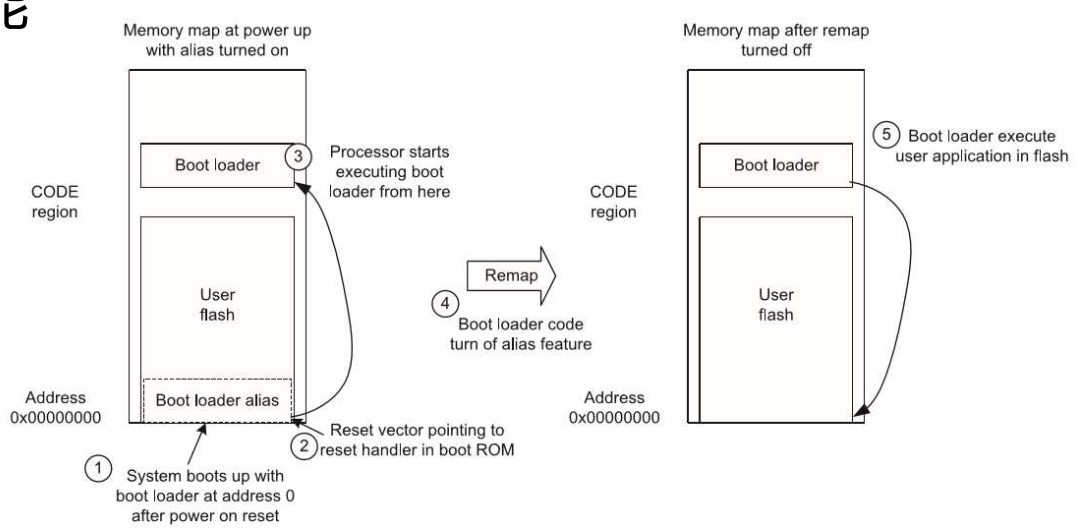


1.5.8 微控制器中的存储器系统

- 微控制器提供的存储器管理特性还包括：
  - Boot loader
  - 存储器重映射
  - 存储器别名
- 很多微控制器中，除了程序存储区（如FLASH）外，还会有一个独立的ROM区，里面包含引导加载程序（Boot loader），其作用是：
  - 提供Flash编程功能，一般通过UART接口进行
  - 提供通信协议栈等的额外的固件，供应用程序调用
  - 提供芯片内置的自检功能（BIST）

带Boot loader的存储器重映射

- Bootloader必须位于地址0处，当系统上电启动时首先执行Bootoader
- 但是当应用程序写入Flash后，再次启动时，往往希望直接从Flash处运行程序，这时就需要进行存储器重映射
- 采用存储器别名，使Bootloader可以从两个不同的区域（ROM和FLASH）访问。重映射后关闭别名功能



1.5.9 存储保护单元（MPU）

- Cortex-M3/M4 内核包括了一个可选的存储器保护单元(MPU)，可以把存储器分成若干区域，分别予以保护。（MPU默认为禁止状态）
- MPU 可以提高系统的可靠性：

1. 阻止用户程序修改操作系统使用的数据；
2. 阻止一个任务访问其它任务的数据区，从而把任务隔开；
3. 允许将内存区域（如关键数据区）设置为只读；
4. 检测异常的内存访问；
5. 设置存储器区域的访问属性。

※ 详见参考教材chapter 11

第1章 ARM和Cortex-M3/M4

1.1 嵌入式系统和ARM（ref. chapter 1）

1.2 ARM Cortex-M3/M4处理器概述（ref. chapter 3）

1.3 Cortex-M3/M4编程模型（ref. chapter 4）

*1.4 Cortex-M3/M4指令系统（ref. chapter 5）

1.5 存储系统（ref. chapter 6）

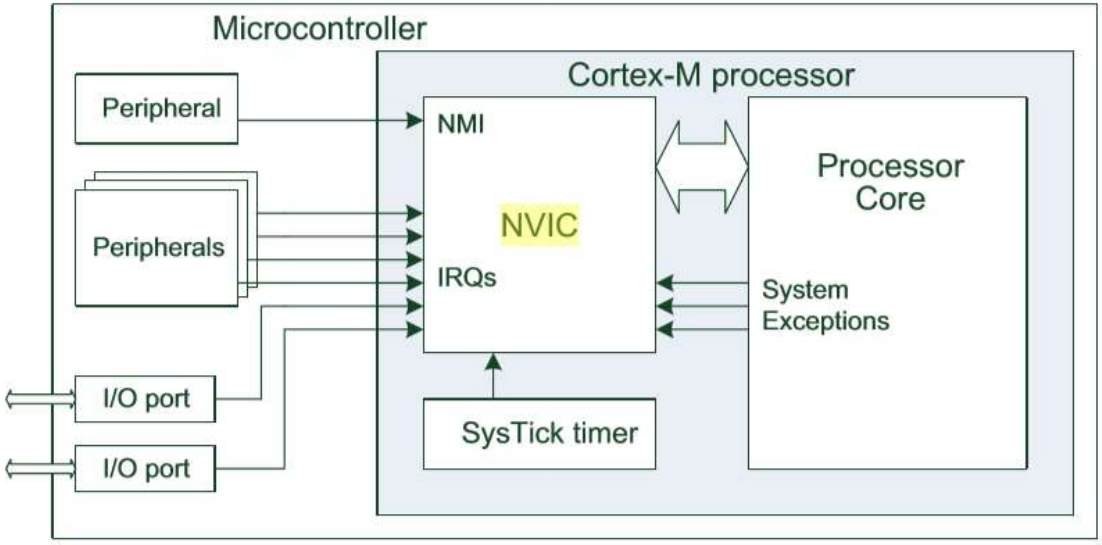
1.6 异常和中断（ref. chapter 7-8）

1.6 异常和中断

- 什么是异常
  - 异常是会改变程序流的事件，当其产生时，处理器暂停当前正在执行的任务，转而执行一段称为异常处理的程序。在异常处理完成后，处理器继续执行被暂停的任务。
- 在ARM架构中，由处理器内部故障，或者内核的SysTick、SVCall等引起的事件称系统异常，由随机的外部事件引发的异常称中断
- Cortex-M3/M4中由嵌套向量中断控制器（NVIC）处理异常

1.6.1 嵌套向量中断控制器 NVIC

- NVIC（Nested Vectored Interrupt Controller，嵌套向量中断控制器），是Cortex-M3/M4 处理器的一个重要组成部分
- NVIC 支持至多256个异常：一个不可屏蔽中断NMI，1 到240 外部中断请求IRQs，一个系统时钟中断SysTick和其他一些系统异常



1.6.2 异常类型

- 异常类型
  - 系统异常：编号为1到15的系统异常，是由内部故障，或者内核的SysTick、SVCall等引起的异常
  - 外部中断：编号大于等于16的至多240个外部中断IRQ，是由随机的外部事件引发的异常
  - 当前运行的异常编号由特殊寄存器IPSR和NVIC的中断控制状态寄存器表示

系统异常表

异常号	异常类型	优先级	描述
1	复位	-3 (最高)	复位
2	NMI	-2	不可屏蔽中断(外部NMI 输入)
3	硬故障	-1	各种故障情况
4	存储器管理故障	可编程	存储器管理故障，MPU 访问非法地址
5	总线故障	可编程	总线故障，比如预取终止
6	使用故障	可编程	由于程序故障或尝试访问协处理器导致的异常
7-10	保留	N/A	—
11	SVCall	可编程	执行SVC指令的系统服务调用
12	调试监视器	可编程	调试监视器，如断点等
13	保留	N/A	—
14	PendSV	可编程	可挂起的系统服务调用
15	SYSTICK	可编程	系统时钟定时器

*中断挂起：当一个异常发生时，如果它不能够被立即执行，则被挂起

外部中断表

异常号	异常类型	优先级	描述
16	IRQ#0	可编程	由外设产生的外部中断
17	IRQ#1	可编程	外部中断
…	…	…	…
255	IRQ#239	可编程	外部中断

1.6.3 优先级定义

- Cortex-M3/M4的优先级
  - 3个固定优先级：复位, NMI, 和硬故障，优先级最高
  - 256级可编程优先级：但大多数CM3/M4芯片会精简设计，通过减少优先级配置寄存器的一些低位来减少优先级数
  - 可编程优先级用一个8位无符号整数表示，数值越小优先级越高
  - 复位后所有中断都处于禁止状态，默认优先级为0
- Cortex-M3/M4支持中断嵌套，一个高优先级的异常可以抢占一个低优先级的异常
  - 128级抢占优先级

优先级配置

- 通过优先级配置寄存器设置可用优先级（该寄存器采用MSB对齐）
  - 最小使用宽度为3bit，即至少8级优先级
  - 最大使用宽度为8bit，即至多256级优先级

使用3位最多可配8个优先级（0x00,0x20,0x40,…, 0xE0）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
用于表达优先级			不使用, 读出值为0				

使用4位最多可配16个优先级（0x00, 0x10,0x20, …, 0xF0）

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
用于表达优先级				不使用, 读出值为0			

优先级配置

- 采用3-bit, 5-bit和8-bit 优先级配置寄存器情况下设备可使用的优先级:

优先级	异常类型	3位优先级寄存器的设备	5位优先级寄存器的设备	8位优先级寄存器的设备
-3 (Highest)	复位	-3	-3	-3
-2	NMI	-2	-2	-2
-1	硬故障	-1	-1	-1
0,	具有优先级并且可编程的异常	0x00	0x00	0x00
1,		0x20	0x08	0x01
2,		0x40	0x10	0x02
…		…	…	…
0xFF		0xE0	0xF8	0xFF

因为采用MSB对齐，位数变化不影响设备间优先级的相对关系

优先级分组

- 8位宽度优先级配置寄存器被进一步分为两个部分：
  - 高位部分对应**组优先级**（也称**抢占优先级**）
  - 低位部分对应**子优先级**
- 组优先级
  - 占0~7位，决定抢占行为，组优先级高的中断可以抢占正在处理的组优先级低的中断。最大128级抢占
- 子优先级
  - 占1～8位（至少1位），当组优先级相同的中断同时产生时，优先响应子优先级最高的中断。最大256级优先级（此时无抢占）

优先级分组

优先分组	组优先级字段	子优先级字段
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]

优先级设置的例子

设置方式	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	可用的优先级
优先级宽度：3位 优先分组：5 抢占优先级数：2^2=4 子优先级数(在每个组优先级内部)：2^1=2	组优先级 00 01 10 11		子优先级字段 0 1	子优先级字段（总是0）					-1, -2, -3, 0x00,0x20, 0x40,0x60, 0x80,0xA0, 0xC0,0xE0
优先级宽度：3位 优先分组：1 抢占优先级数：2^3=8 子优先级数(在每个组优先级内部)：无	组优先级字段 000 001 … 111			组优先级字段（总是0）			子优先级字段（总是0）		-1, -2, -3, 0x00, 0x20, 0x40, 0x60, 0x80, 0xA0, 0xC0, 0xE0

1.6.4 向量表

- 向量表
  - Cortex-M3/M4支持15个系统异常和最多240个外部中断IRQ（具体使用由芯片制造商决定）。异常类型号：1～255。
  - 向量表是一个字（32位整数）数组，每个数组元素对应一种异常，存放异常服务例程的入口地址。

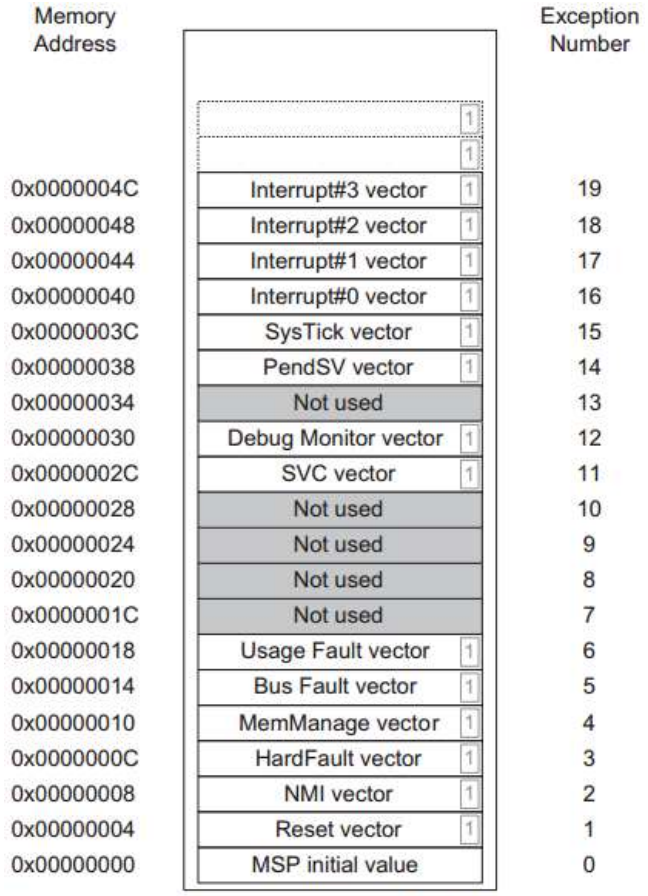
例如: 复位的异常类型号为1，复位向量的地址是 1×4 = 0x00000004

NMI 的异常类型号为2，NMI向量的地址是 2×4 = 0x00000008

0号类型（地址 0x00000000）用作复位后MSP的开始值

复位后的异常向量表

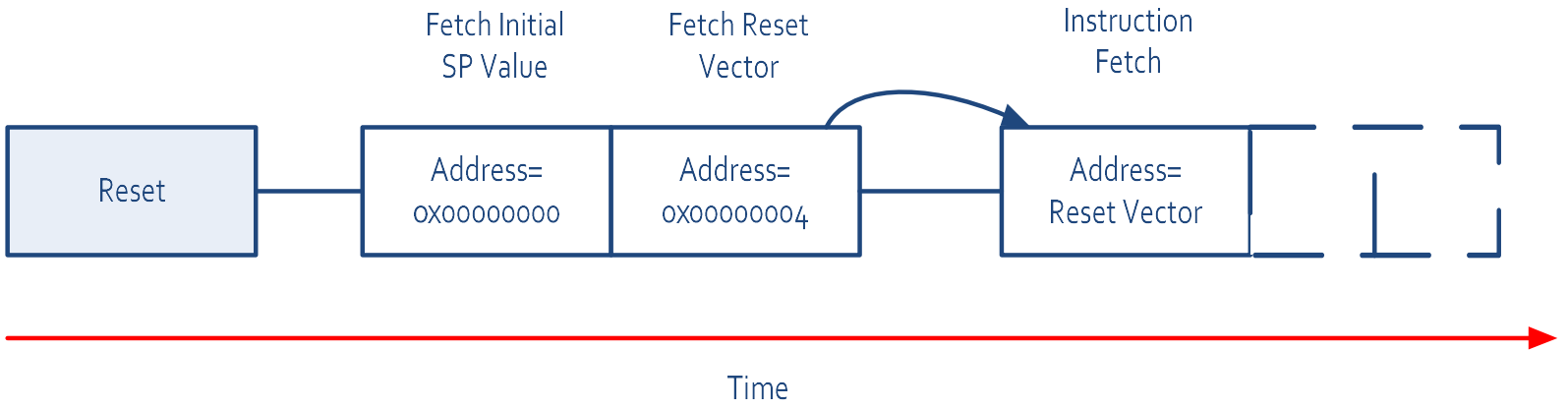
地址	异常类型号	异常向量 (32位整数值)
0x00000000	—	MSP 初始值
0x00000004	1	复位向量(程序计数器初始值)
0x00000008	2	NMI 服务例程启动地址
0x0000000C	3	硬故障服务例程入口地址
0x00000010	4	存储器管理故障服务例程入口地址
0x00000014	5	总线故障服务例程入口地址
0x00000018	6	用法故障服务例程入口地址
0x0000001C-0x28	7-10	保留
0x0000002C	11	SVC（系统调用）服务例程入口地址
0x00000030	12	调试监视服务例程入口地址
0x00000034	13	保留
0x00000038	14	PendSV服务例程入口地址
0x0000003C	15	SYSTICK服务例程入口地址



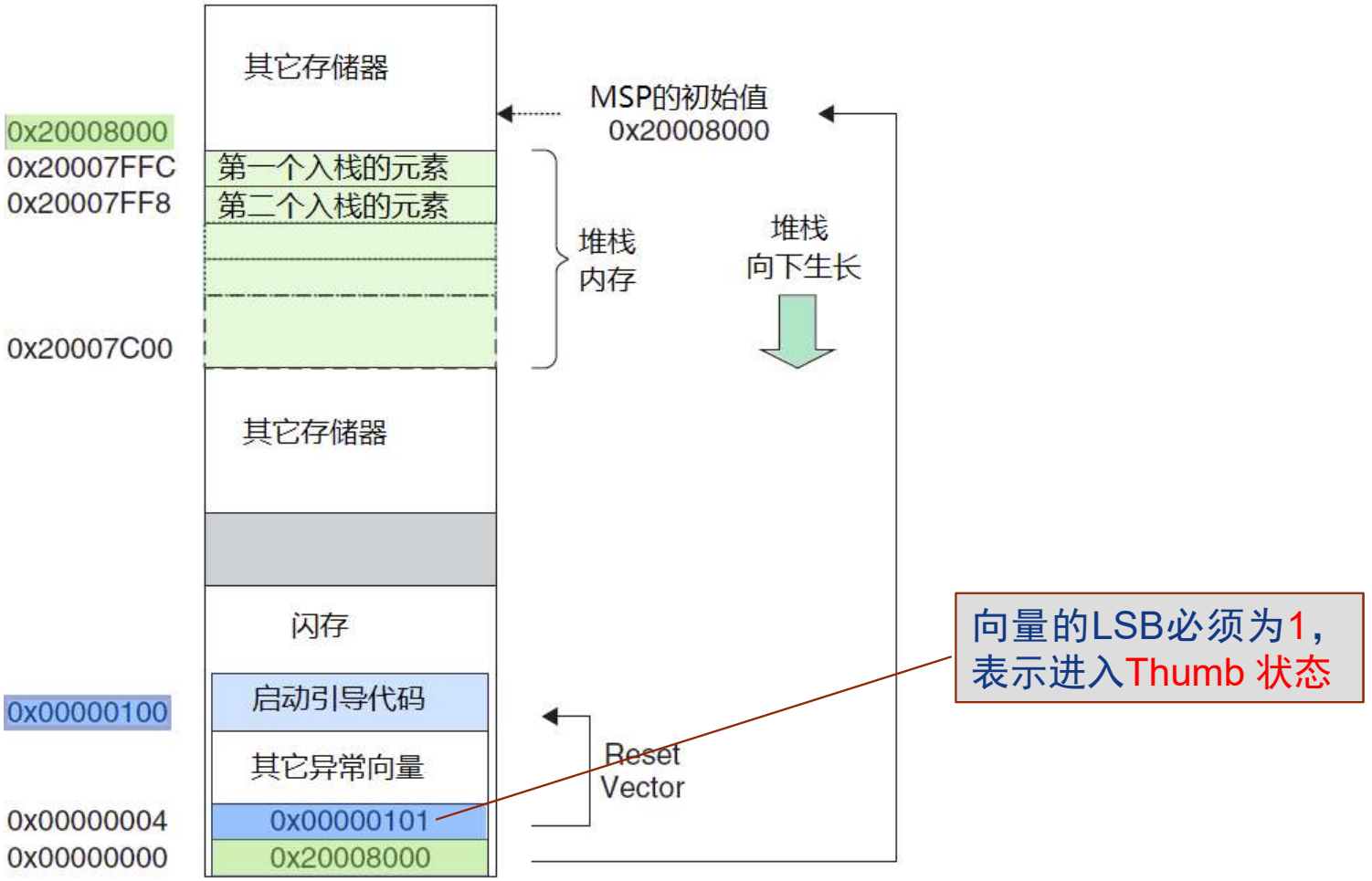
注：每个向量的LSB必须为1表示Thumb状态

Cortex-M3/M4的复位序列

- 在离开复位状态后，处理器做的第一件事就是读取下列两个32位整数的值:
  - 1. 地址 0x00000000：读取R13（主堆栈指针MSP）的初始值
  - 2. 地址 0x00000004：读取复位向量，即启动程序的起始地址⇒PC
    （复位向量的LSB必须为1，表示进入 Thumb 状态）



- 初始化堆栈指针值和初始化程序计数器 (PC) 值的例子



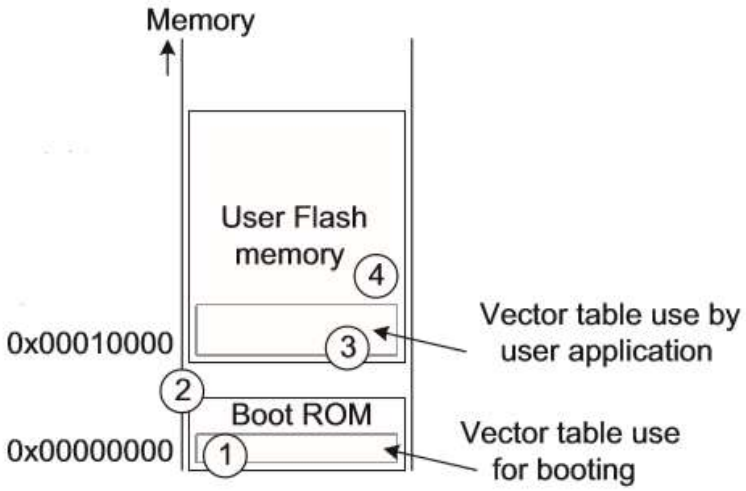
向量表重定位

- Cortex-M3/M4允许向量表重定位：
  - 向量表的存储位置可以通过设置NVIC中的向量表偏移寄存器（VTOR），将向量表重定位到其它的内存地址。复位后VTOR中的值为0。
  - 重定位的向量表基地址要与向量表大小（扩展到2的n次幂）对齐

例：若某微控制器共有32 个IRQ 中断，即总异常数 = 32+16 (系统异常) = 48，扩展到2的n次幂: 48→64，乘以4: 64×4=256，即向量表基地址必须与256对齐。因此重定位的向量表基地址可以被设置为0x0, 0x100, 0x200等等。

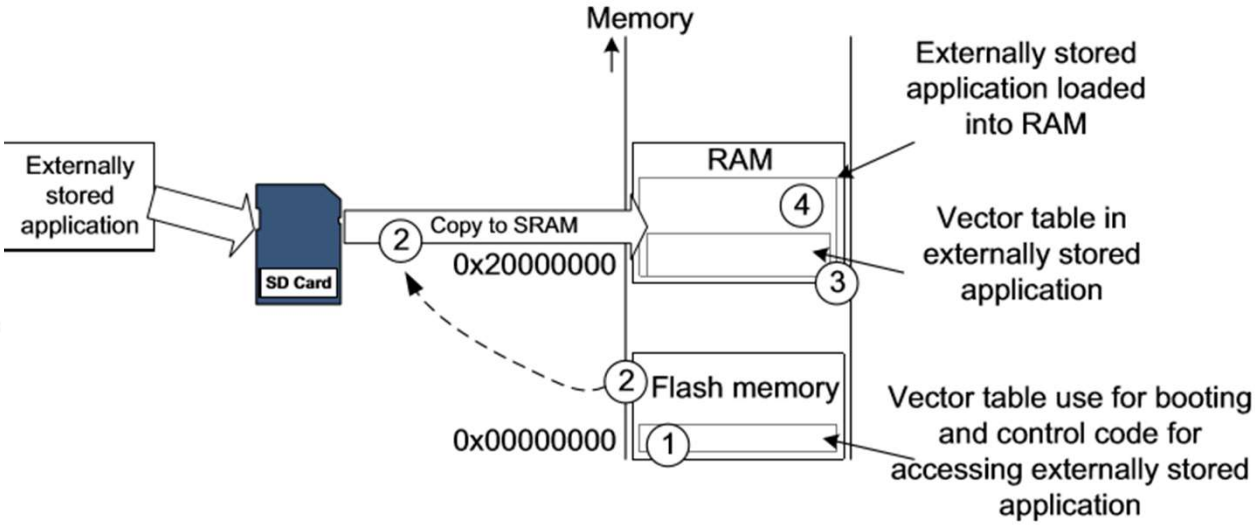
向量表重定位的应用情景

- 具有Bootloader的设备



(1) 利用启动ROM中的向量表启动
(2) 启动Boot loader任务
(3) 设置VTOR指向用户Flash中的向量表
(4) 跳转到用户Flash存储器的向量表中的复位向量

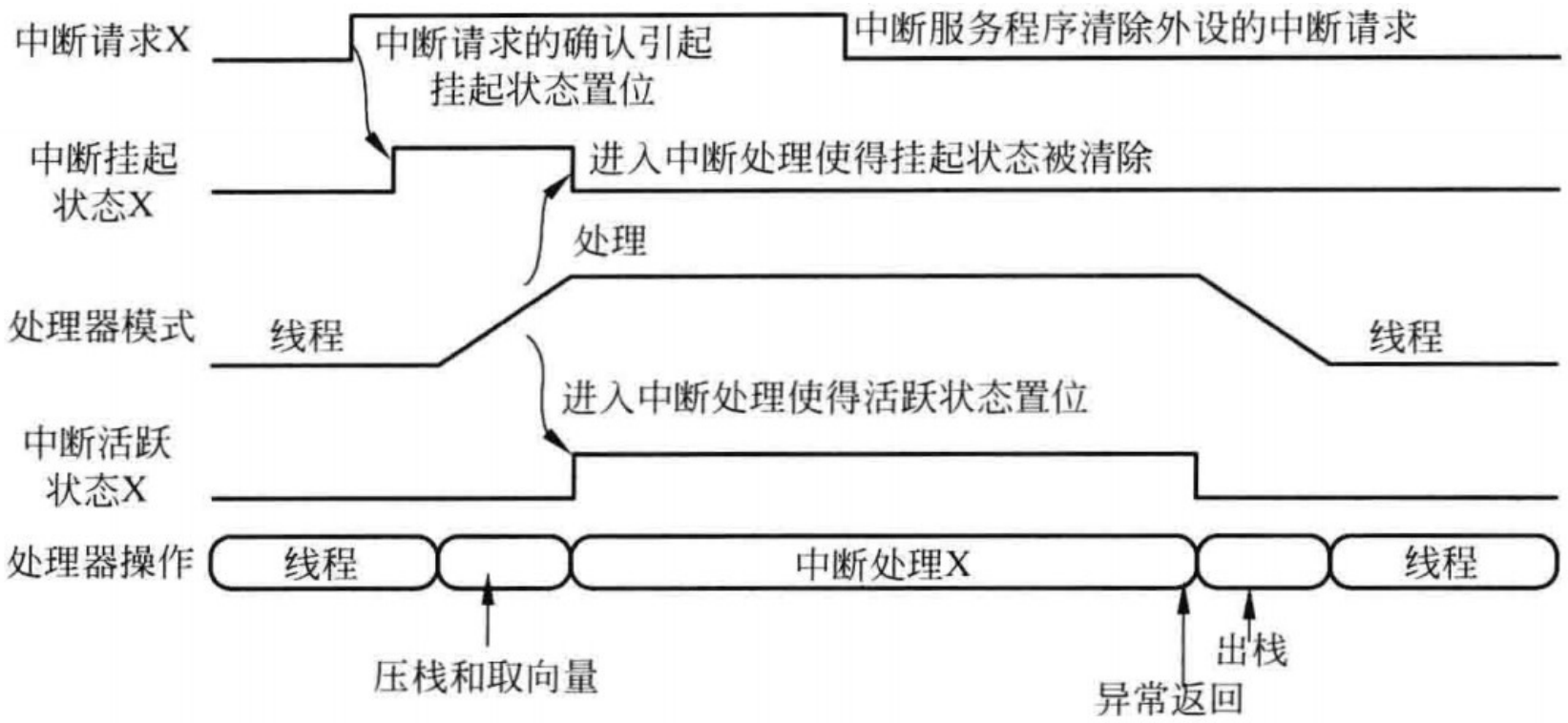
- 应用程序加载到RAM



(1) 利用Flash存储器中的向量表启动
(2) 初始化硬件并将存储在外部设备中的应用复制到RAM
(3) 设置VTOR指向SRAM中应用程序的向量表
(4) 跳转到SRAM中向量表的复位向量并启动应用

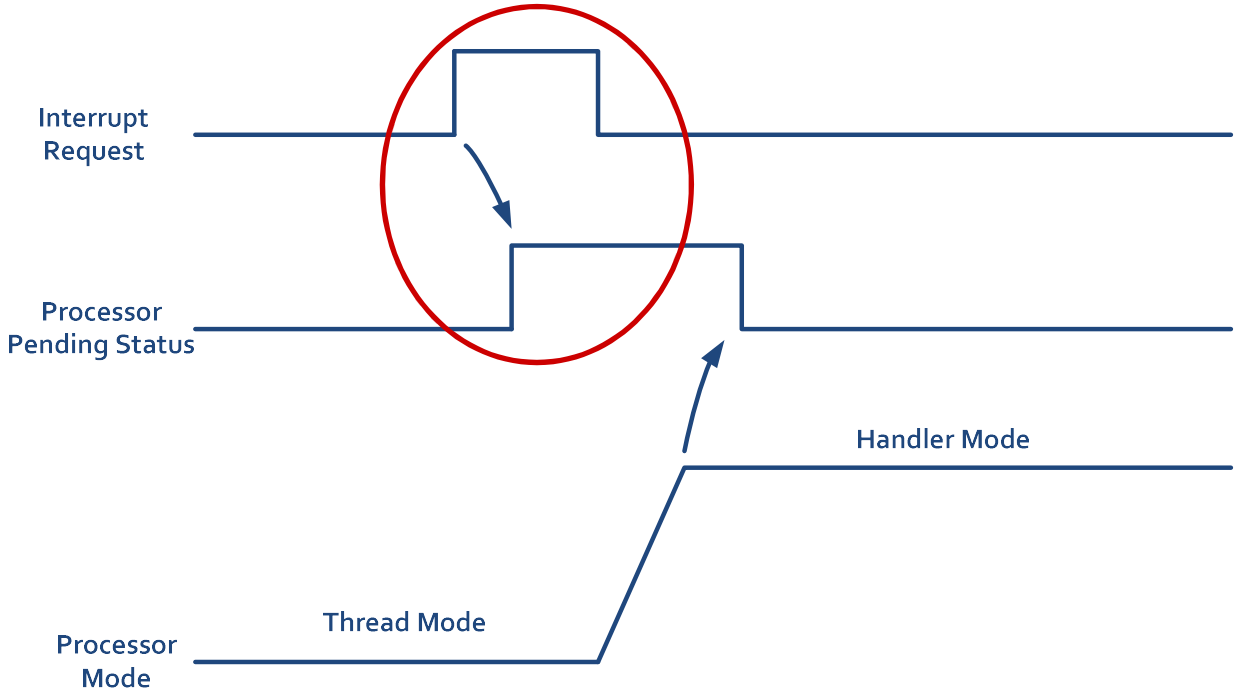
1.6.5 中断输入和挂起行为

- 中断挂起和激活过程：中断请求→中断挂起→中断激活→中断清除



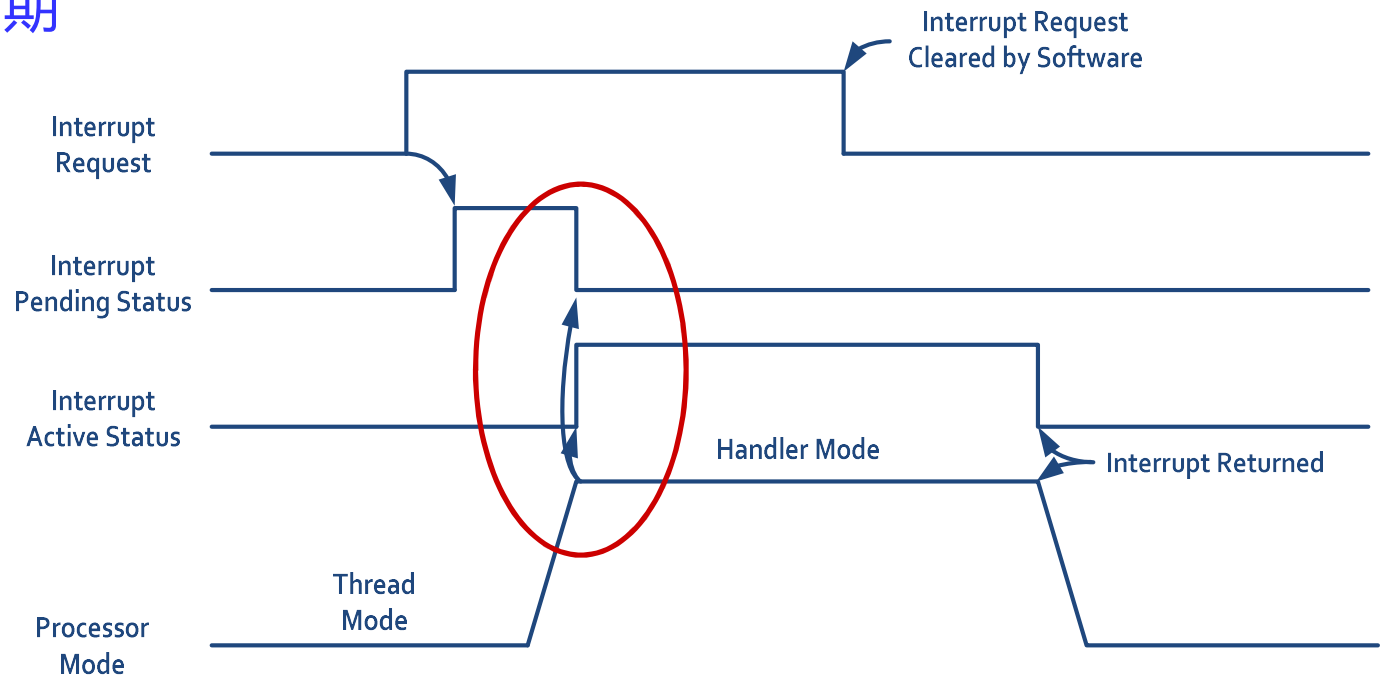
中断请求→中断挂起

- 当一个中断请求有效，该中断将被挂起（挂起状态被置位）。即使该中断源后来撤销了这个中断请求，挂起的中断状态仍会导致中断的响应。



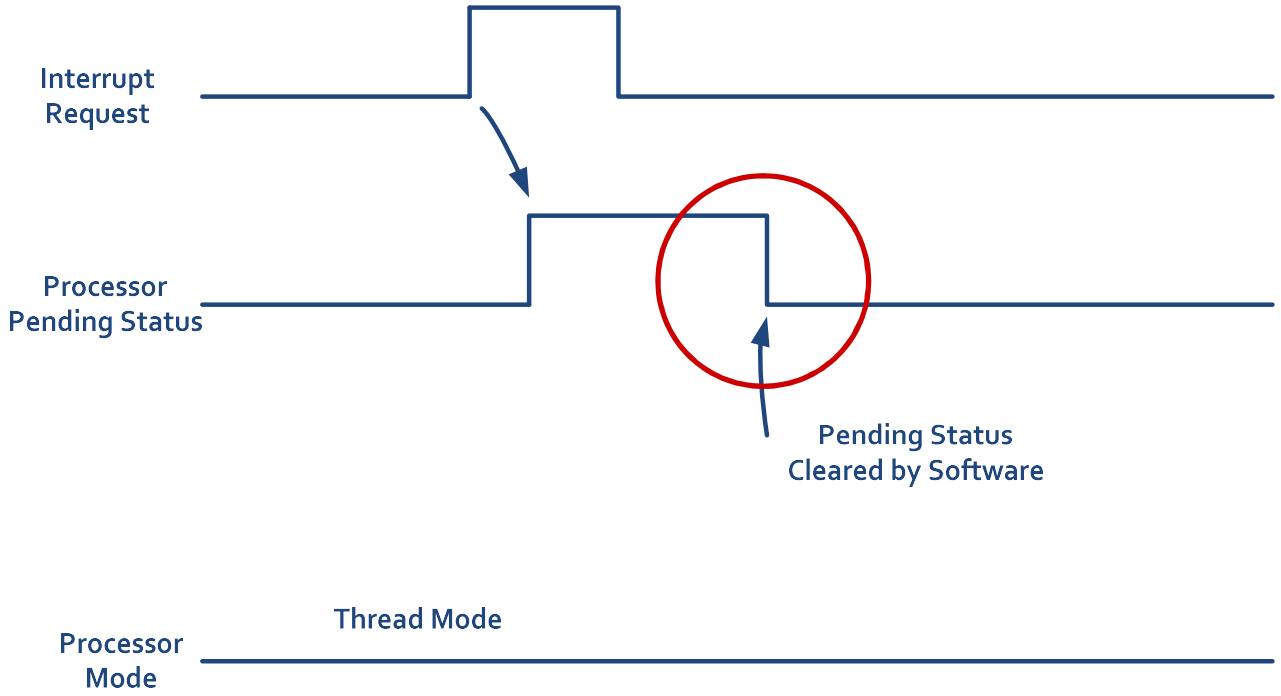
中断挂起→中断激活

- 当处理器开始执行一个中断，中断活动状态被置位，同时挂起状态位将被自动清除。从中断请求到中断被处理的时间称中断等待时间，至少需要12个时钟周期



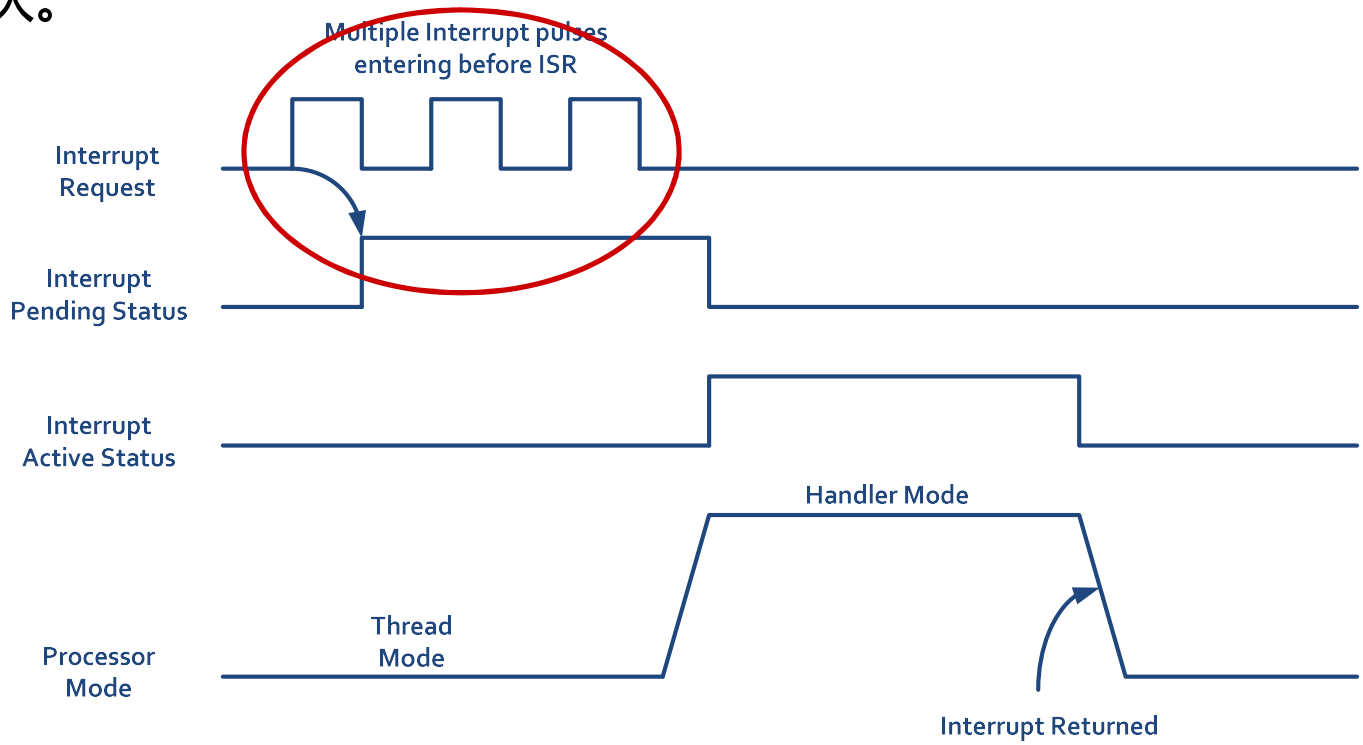
中断输入和挂起相关行为

- 如果挂起状态在处理器开始响应挂起中断之前被清除，则中断取消：



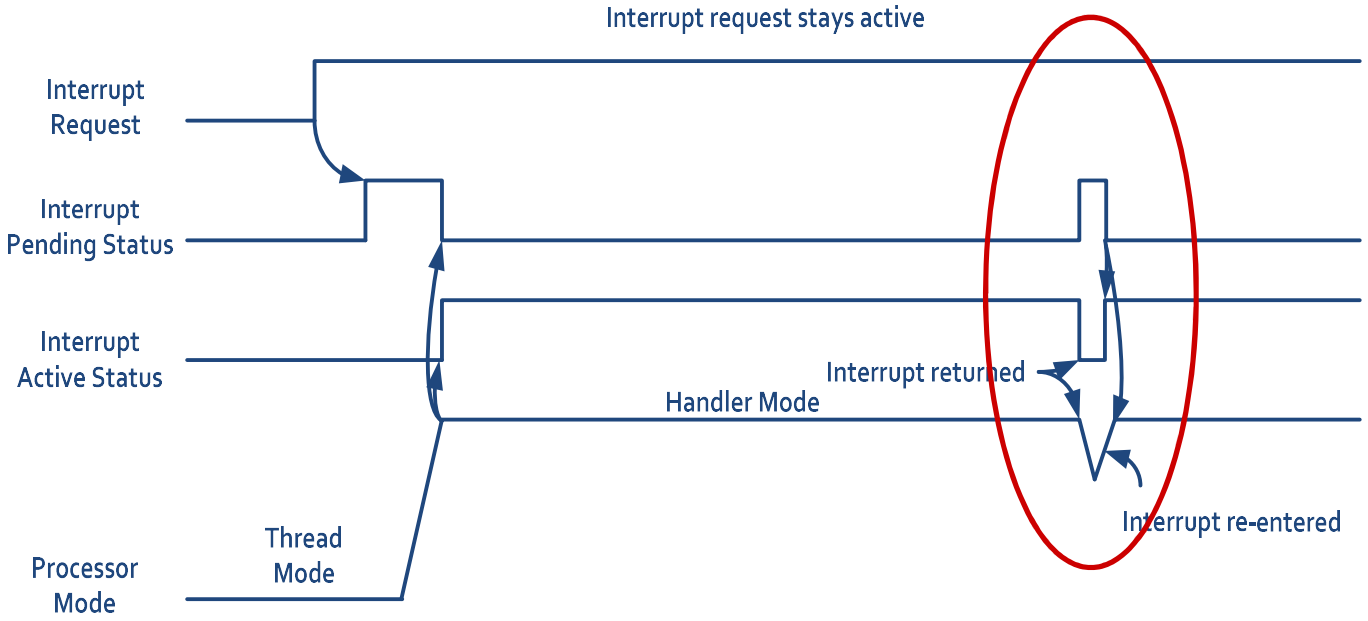
中断输入和挂起相关行为

- 如果一个中断在处理器执行前有多次脉冲，它将被视为一次中断请求，而不是多次。



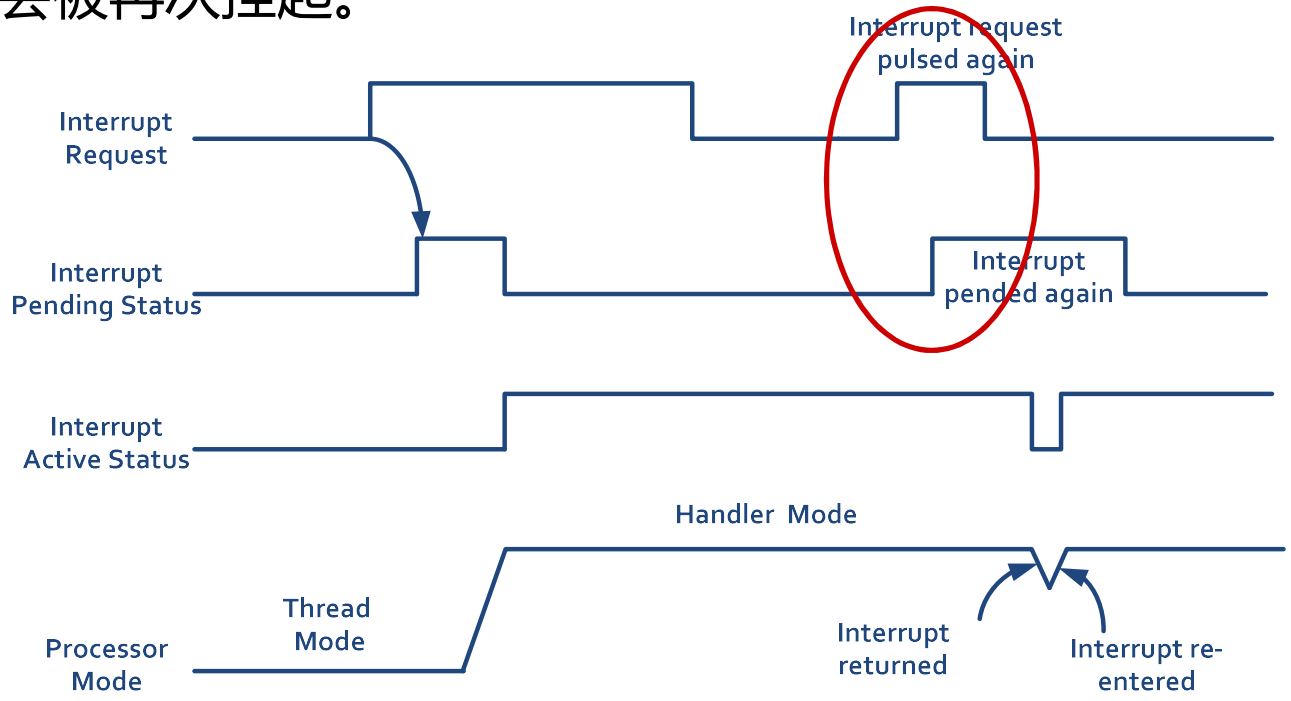
中断输入和挂起相关行为

- 如果一个中断源持续保持中断请求信号，在中断服务程序结束时，中断将被再次挂起。



中断输入和挂起相关行为

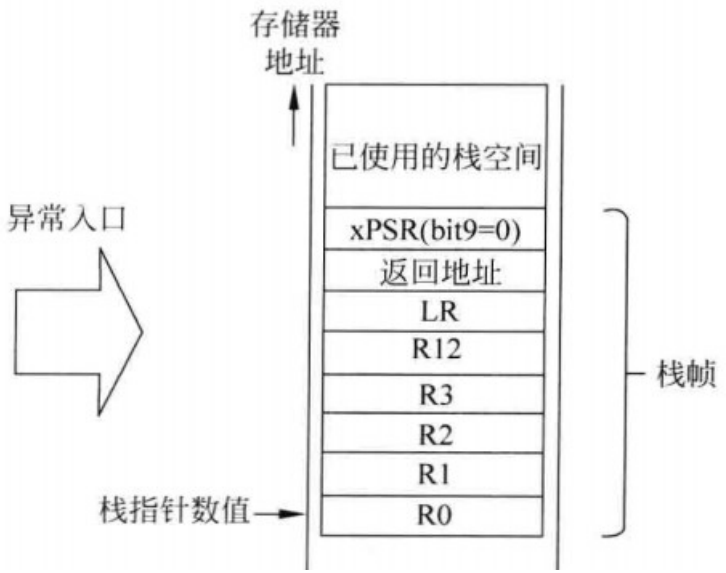
- 如果一个中断请求释放了，但是在中断服务结束前再次接收到脉冲，那么它将仍然会被再次挂起。



1.6.6 异常处理流程

- 异常进入和压栈

1. 入栈：依次把xPSR, PC, LR, R12,R3-R0自动压入适当的堆栈中组成栈帧，要求双字对齐。（不考虑浮点内容）
2. 取异常/中断向量：从向量表中找到异常向量
3. 更新寄存器：
   - SP：入栈时PSP或MSP被更新到新的位置。
     进入中断服务例程后，由MSP负责对堆栈的访问
   - PSR：IPSR位段会被更新为新响应的异常编号
   - PC：将指向中断服务例程的入口地址
   - LR：被更新成一种特殊的值，称为“EXC_RETURN”，用于异常返回



异常处理流程

- 异常返回和出栈

1. 出栈：恢复压入栈中寄存器
2. 更新NVIC寄存器：清异常活动状态位

- 3种途径可以触发异常返回：

返回指令	描　　述
BX<reg>	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
POP {PC}或 POP {…, PC}	在进入异常处理后，LR 的值通常会被压入栈中，可以使用操作一个寄存器或包括 PC 在内的多个寄存器的 POP 指令，将 EXC_RETURN 放到程序计数器中，这样处理器会执行中断返回
加载(LDR)或多加载(LDM)	可以利用 PC 为目的寄存器的 LDR 或 LDM 指令产生中断返回

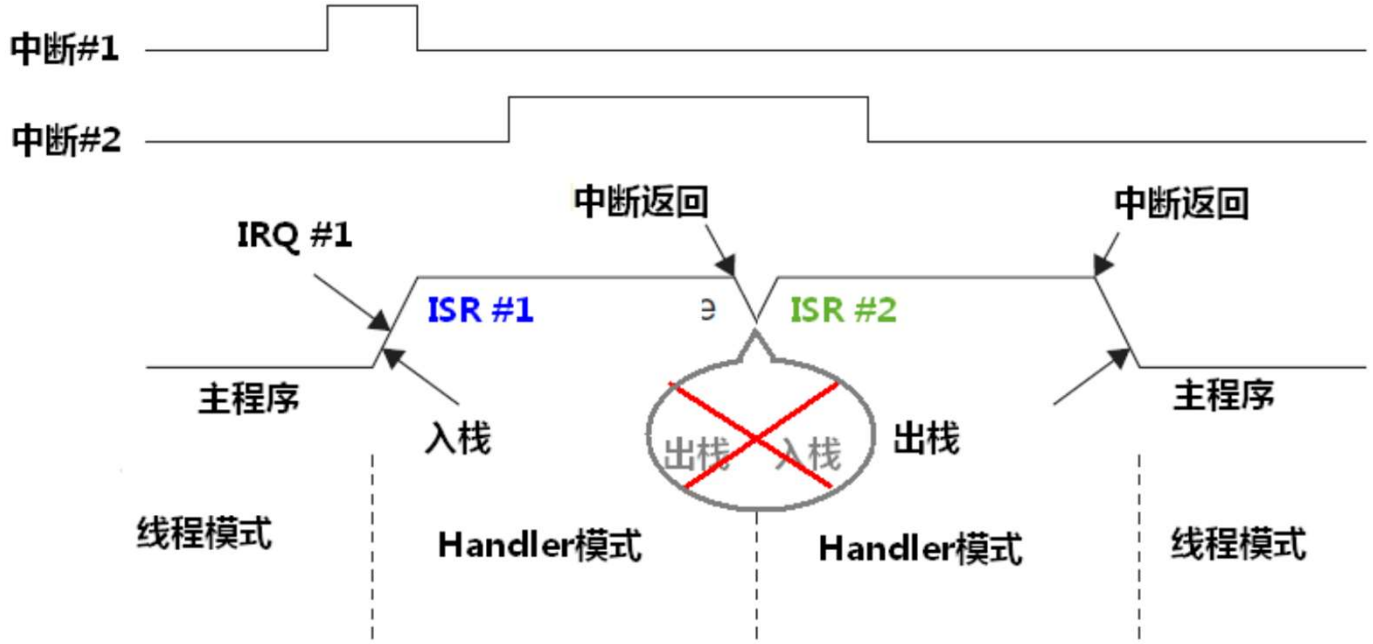
嵌套的中断

- NVIC根据优先级的设置控制抢占和嵌套；
- 自动入栈和出栈机制保证中断发生嵌套时寄存器数据的正确。
- 相同的异常不允许重入（如，SVC服务例程中，不得再使用SVC指令）

- 注意：必须正确设置主堆栈容量的最小安全值。每嵌套一级，至少需要8个字，即32字节的堆栈空间。堆栈溢出是很致命的

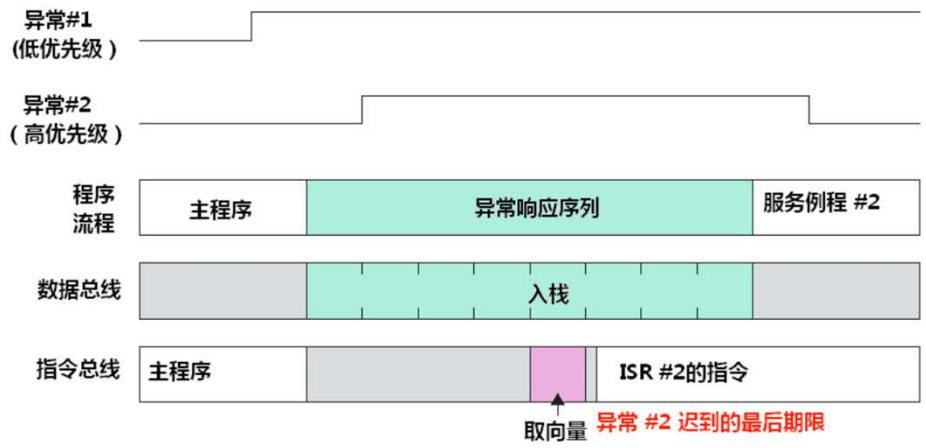
末尾连锁（Tail-Chaining）

- 若有异常因无法响应而被挂起，在当前的异常执行返回后，系统可以跳过出栈和入栈过程，直接进入挂起异常的服务例程。末尾连锁可以将中断等待时间减少至6个时钟周期

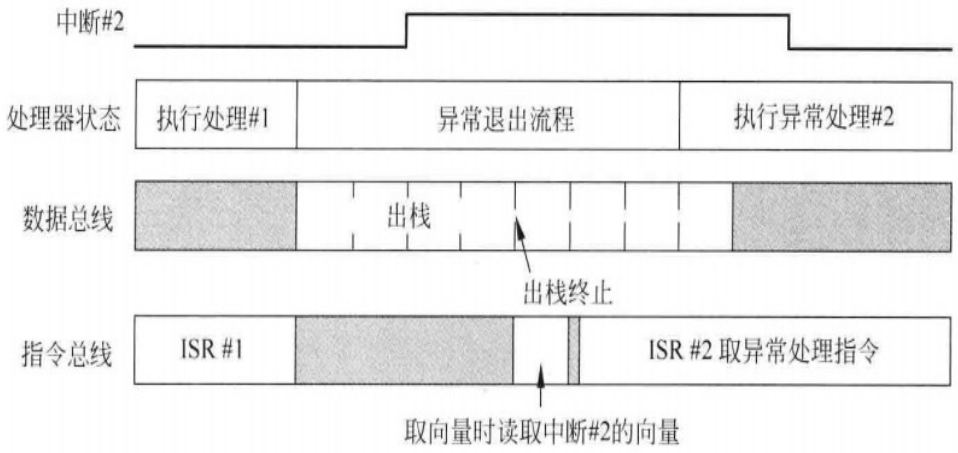


延迟到达和出栈抢占

- 延迟到达：当高优先级异常的请求在低优先级异常的入栈响应阶段到达，只要低优先级异常的服务例程尚未执行，则高优先级的后到异常的服务例程会优先执行。



- 出栈抢占：若一个异常的请求在另一个异常的出栈阶段到达，处理器会舍弃出栈操作而开始服务下一个异常。



1.6.7 Fault异常

- 总线Fault

总线故障是指在数据传输过程中AHB接口上产生的故障。

总线故障可能发生的情况包括：

1. 取指令（称为预取中止, Pre-fetch Abort）
2. 数据读/写（称为数据中止, Data Abort）
3. 在中断处理开始的压栈（称入栈故障）
4. 中断处理结束时的出栈（称出栈故障）
5. 当处理器启动中断服务序列时，读取中断向量地址（硬fault）

Fault异常

- 内存管理Fault

  常见的内存管理故障包括：

  1. 对MPU定义的内存区域之外的地址访问；
  2. 从不可执行内存区域执行代码；
  3. 向只读区域写；
  4. 在用户态下访问特权地址。

Fault异常

- 用法 Fault

  引起用法故障的有：

  1. 执行未定义指令
  2. 执行协处理器指令
  3. 尝试转换到ARM状态
  4. 无效的中断返回（链接寄存器包括无效/不正确的数值）
  5. 使用多重加载/存储指令访问时，地址不对齐

  此外，通过设置NVIC中的一些位， 可使下列行为产生用法故障：

  - 除以0、任何非对齐内存访问

Fault异常

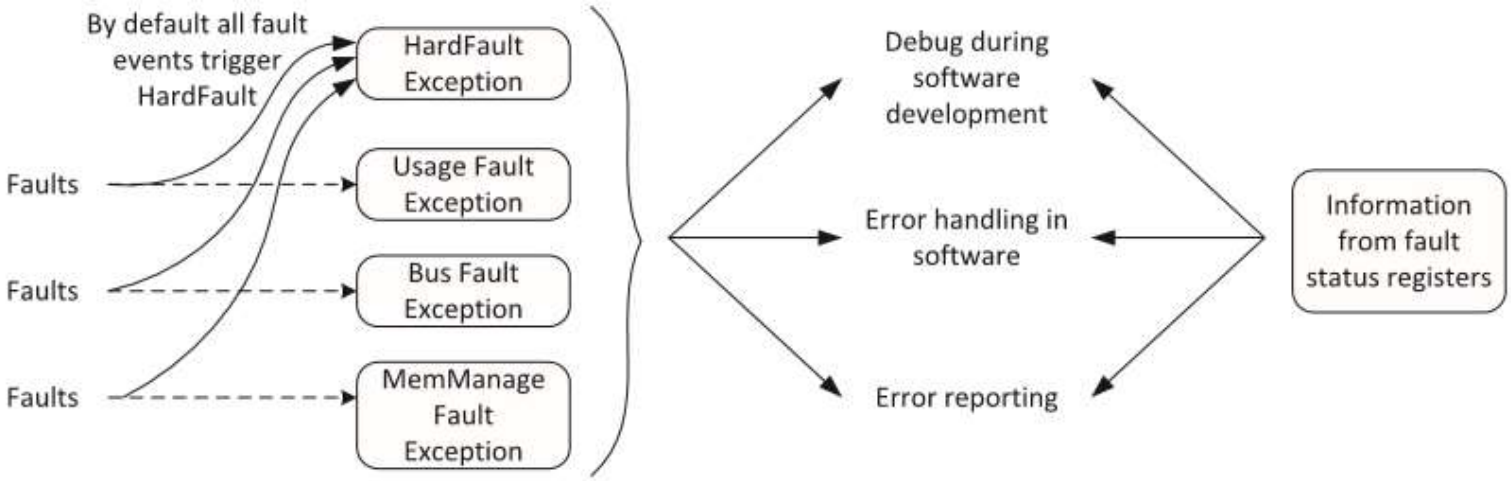
- 硬Fault

  可以引起硬故障的有：

  1. 用法故障、总线故障、内存管理故障的上访
  2. 取向量表时产生的总线故障
  3. 调试事件触发

Fault异常处理

- Fault异常处理
  - 缺省情况下，总线故障、用法故障、内存管理故障都除能的(disabled)，所有的故障都将触发硬Fault
  - 可以编程使能(enable)这三种故障



1.6.8 SysTick定时器

- SysTick是Cortex-M处理器内集成的一个系统节拍定时器，属于NVIC的一部分，异常类型号15 。
- 操作系统需要一个周期性的中断来定期触发OS内核，SysTick可用于任务管理和上下文切换等
- 在处理器内增加SysTick可提高软件的可移植性
- 若应用中不使用操作系统，SysTick可用作简单的定时器
- SysTick是一个24位向下计数器

1.6.9 SVC 和 PendSV

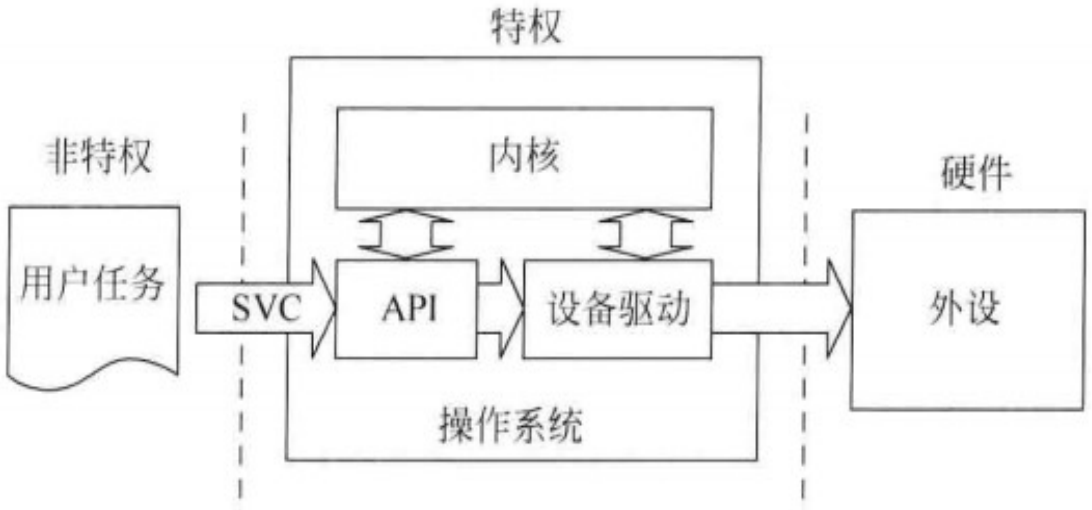
SVC 和 PendSV 是两个目标为软件和操作系统的异常。

- SVC（系统服务调用）
  - SVC 是用来产生系统功能调用请求，可用于实现用户应用任务访问系统资源的API
  - 操作系统通过SVC提供访问硬件的函数，而不是允许非特权级别的用户程序直接访问硬件。
  - SVC 可以提高软件的可移植性，因为用户程序无需知道硬件的编程细节

SVC 和 PendSV

- SVC可作为操作系统服务的入口
  - 使用SVC 指令产生SVC异常

例: SVC 0x3 ; Call SVC function 3



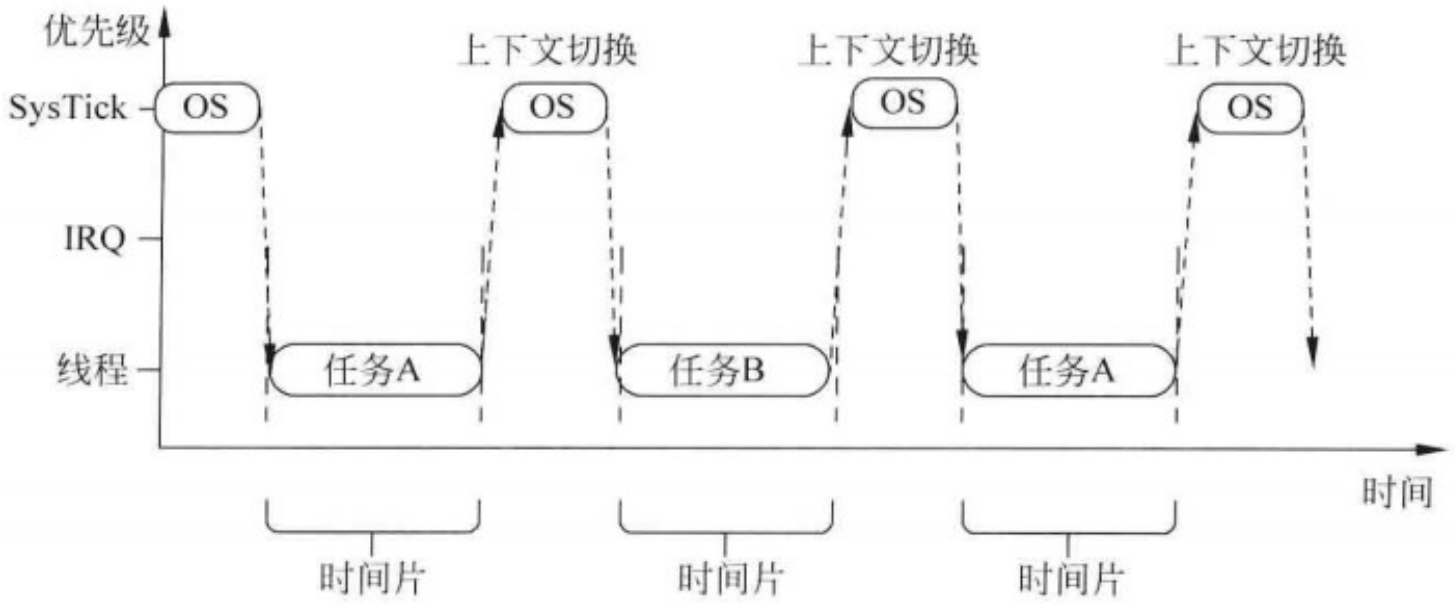
SVC 和 PendSV

- PendSV（挂起的系统调用）
  - PendSV 和 SVC 共同服务于 OS
    - SVC异常必须立即响应（否则将上访成硬故障）
    - PendSV 则可以被挂起，通常 OS 通过挂起一个异常来保证另一个更重要任务的优先完成
  - PendSV一个典型的用途是上下文切换
- 操作系统中上下文切换（多任务切换）一般通过以下两种途径实现：
  1. 系统定时器SysTick中断（轮转调度）
  2. 执行一个挂起系统调用PendSV

使用 SysTick 的上下文切换

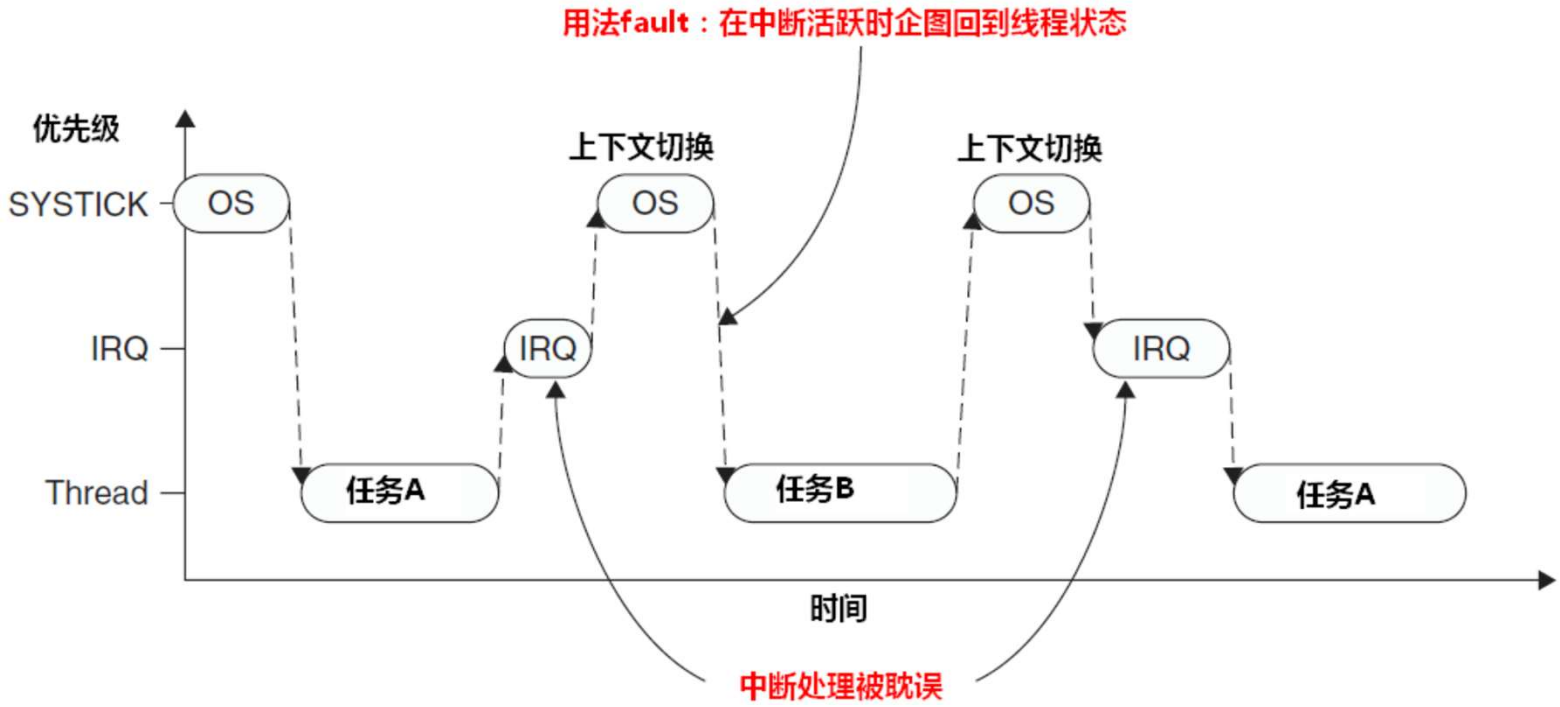
例1： 通过 SysTick 异常触发上下文切换

- 一个只有两个任务的系统，通过SysTick 异常触发上下文切换（分时系统）



使用 SysTick 的上下文切换

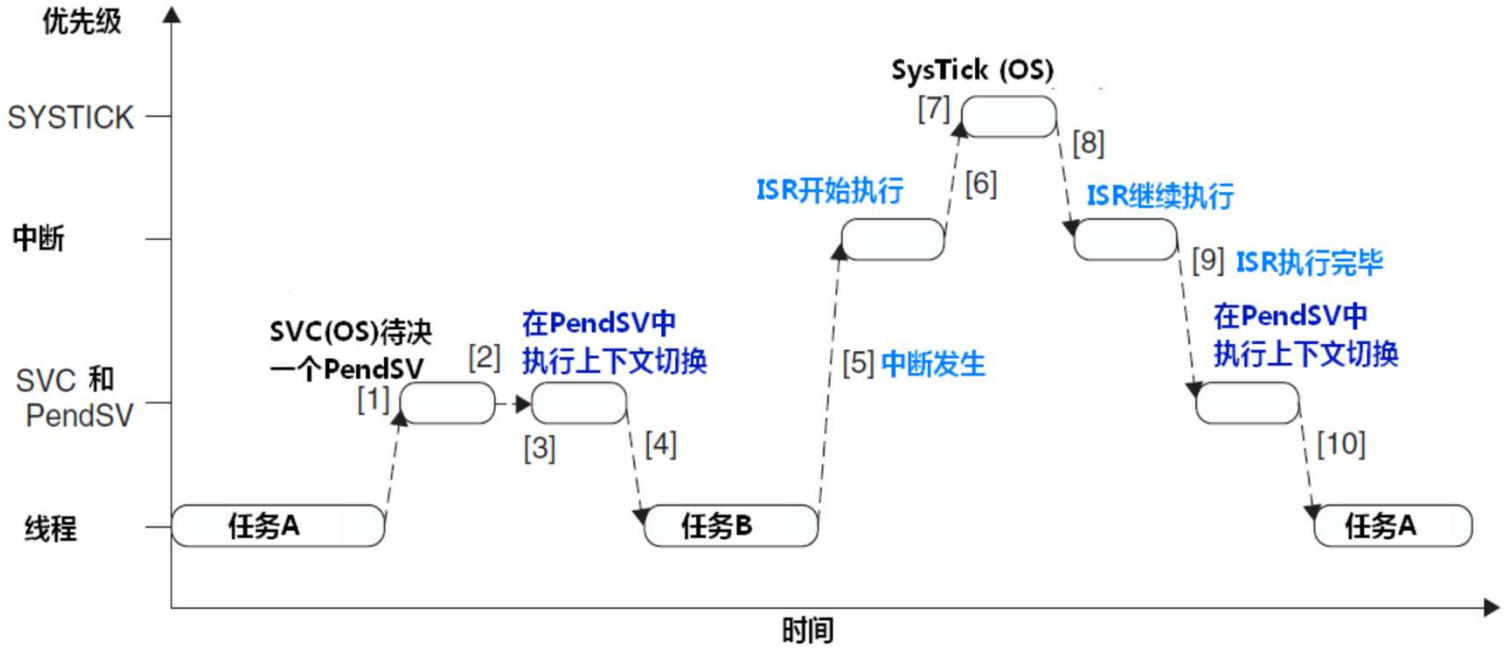
- 若在中断处理时发生SysTick异常，则SysTick异常会抢占IRQ中断服务。若此时执行任务切换，则会引发用法fault



使用 PendSV 的上下文切换

例2：通过PendSV异常实现上下文切换

- PendSV异常会自动延迟上下文切换的请求，直到其它的ISR都完成处理后才放行



- 第一单元完 -