第3章

Cortex-M0+处理器结构

本章介绍 ARM Cortex-M0＋处理器结构，内容包括 Cortex-M0＋处理器核和核心外设、Cortex-M0＋处理器的寄存器、Cortex-M0＋处理器空间结构、Cortex-M0＋的端及分配、Cortex-M0＋处理器异常及处理，以及 Cortex-M0＋存储器保护单元。

通过本章内容的学习，理解并掌握 Cortex-M0＋CPU 的结构及功能，为进一步学习该处理器的指令集打下坚实的基础。


视频讲解

3.1 Cortex-M0＋处理器核和核心外设

STM32G0 系列 MCU 内的 Cortex-M0＋处理器核和核心外设结构如图 3.1 所示。

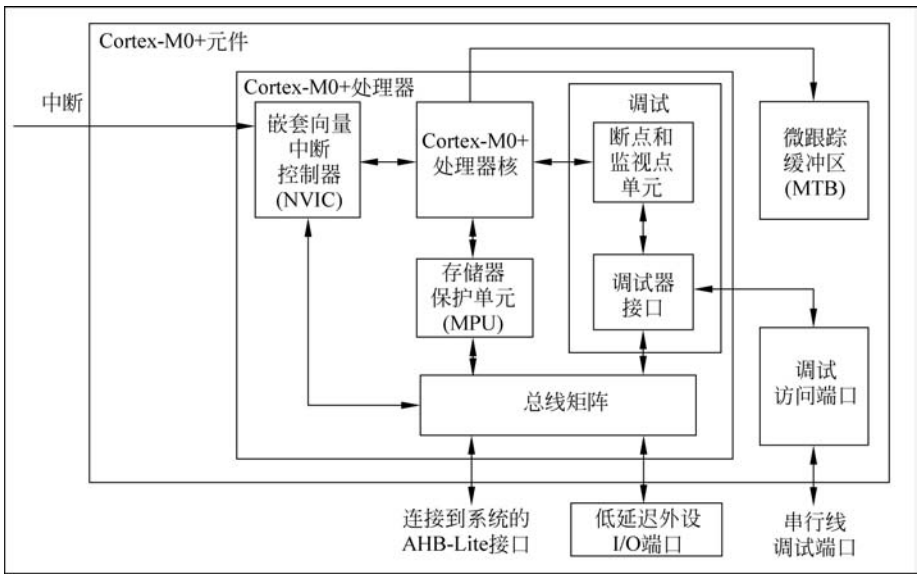

图 3.1 Cortex-M0＋处理器核和核心外设结构

该处理器是一款入门级的 32 位 ARM Cortex 处理器，专为广泛的嵌入式应用而设计。Cortex-M0＋处理器基于对面积和功耗进行充分优化的 32 位处理器内核构建，并具有两级流水线的冯·诺依曼架构。该处理器通过一个小的但是功能强大的指令集和广泛优化的设计来提供出色的能源效率，从而提供包括单周期乘法器在内的高端处理硬件。

Cortex-M0＋处理器实现了 ARMv6-M 架构，该架构基于 16 位的 Thumb 指令集并包含 Thumb-2 技术。这样，就提供了现代 32 位架构所期望的出色性能，并且具有比其他 8 位和 16

位微控制器更高的代码密度。

Cortex-M0＋处理器紧耦合集成了可配置的嵌套向量中断控制器(Nested Vectored Interrupt Controller,NVIC),以提供最好的中断性能。其特性包括:

(1) 包括不可屏蔽中断(Non-Maskable Interrupt,NMI);

(2) 提供零抖动中断选项;

(3) 提供 4 个中断优先级。

处理器内核和 NVIC 的紧密集成提供了快速执行中断服务程序(Interrupt Service Routine,ISR)的能力,从而显著减少了中断等待时间。这是通过寄存器的硬件堆栈以及放弃并重新启动多个加载和多个保存操作的能力来实现的。中断句柄不要求任何汇编程序封装代码,从而消除了 ISR 的代码开销。当从一个 ISR 切换到另一个 ISR 时,尾链优化还可以显著减少开销。

为了优化低功耗设计,NVIC 与休眠模式集成在一起,该模式包括深度休眠功能,这样可使整个器件快速断电。

3.1.1 Cortex-M0＋处理器核

Cortex-M0＋处理器核是 Cortex-M0＋最核心的功能部件,负责处理数据,它包含内部寄存器、算术逻辑单元(ALU)、数据通路和控制逻辑。

1. 处理器核的主要功能

Cortex-M0＋处理器提供的主要功能包括:

(1) 使用 Thumb-2 技术的 Thumb 指令集;

(2) 用户模式和特权模式执行;

(3) 与 Cortex-M 系列处理器向上兼容的工具和二进制文件;

(4) 集成超低功耗休眠模式;

(5) 高效的代码执行可以降低处理器时钟或增加休眠时间;

(6) 用于对安全有严格要求的存储器保护单元(Memory Protection Unit,MPU);

(7) 低延迟、高速外设 I/O 端口;

(8) 向量表偏移寄存器;

(9) 丰富的调试功能。

2. 处理器核中的流水线

此外,ARM Cortex-M0＋内核内提供了两级流水线(Cortex-M0、M3 和 M4 具有三级流水线)。这个两级流水线减少了内核响应时间和功耗。第一级流水线完成获取指令(简称取指)和预译码,第二级流水线完成主译码和执行,如图 3.2 所示。

注:当以前的 Cortex-M 内核(具有三级流水线)执行条件分支时,下一条指令不再有效。这就意味着每次有分支的时候都必须刷新流水线。通过转移到两级流水线,可以最大限度地减少对 Flash 的访问并降低功耗。通常,Flash 存储器的功耗占据整个 MCU 功耗的绝大部分。因此,减少访问 Flash 的次数将对降低总功耗产生直接影响。

大多数 ARMv6-M 架构指令的长度是 16 位。只有 6 条 32 位指令,其中大多数是控制指令,很少使用。但是,用于调用子程序的分支和链接指令也是 32 位,以便支持该指令与指向要执行下一条指令的标号之间的较大偏移。

理想情况下,每两个 16 位指令只有一个 32 位访问,因此每条指令的访存次数更少。图 3.2 中,在第 2 个时钟周期没有发生取指操作。当第 N 条指令为加载/保存指令时,AHB-

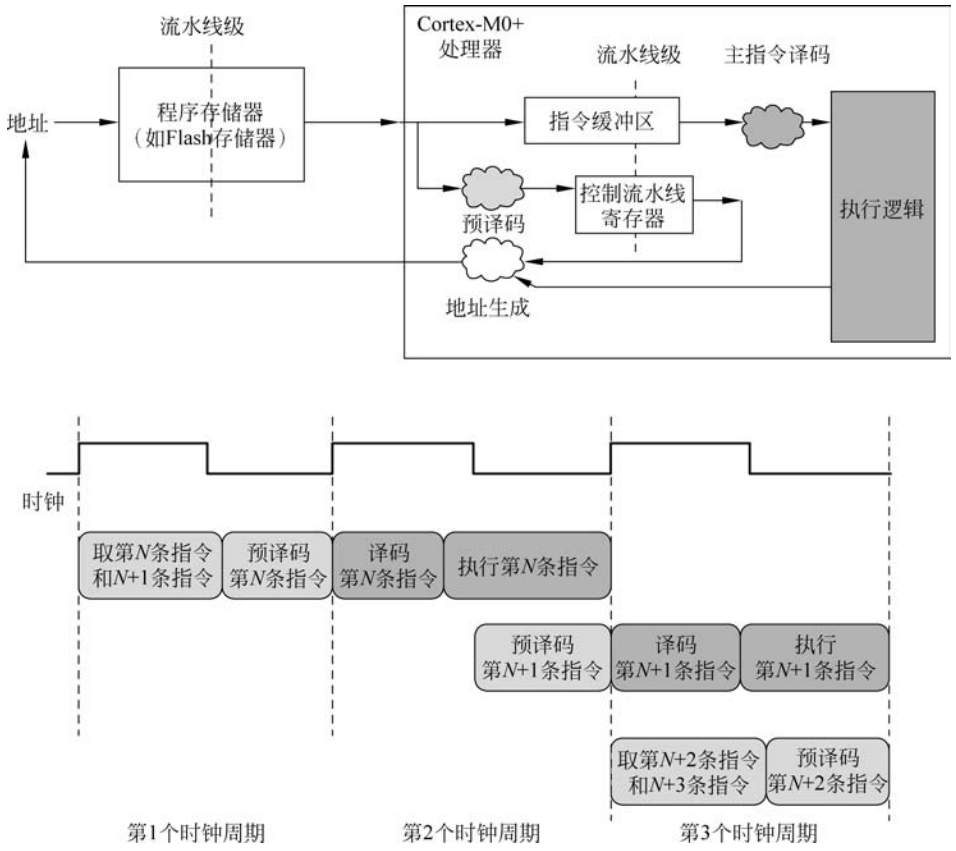

图 3.2　Cortex-M0＋处理器的两级流水线

Lite 端口可用于执行数据访问。

下面通过一个实例，说明 Cortex-M0＋处理器核采用两级流水线的优势。

代码清单 3-1　一段运行在 Cortex-M0＋上的代码

```
        第 0 条指令
        B    Label          ; 分支跳转到 Label
        第 1 条指令          ; 分支影子指令
        第 2 条指令          ; 分支影子指令
          ⋮
Label :  第 N 条指令
        第 N + 1 条指令
```

如图 3.3 所示，由于采用了两级流水线，使得浪费更少的预取指令。

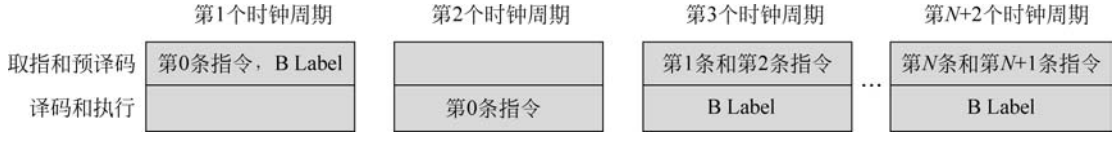

图 3.3　Cortex-M0＋处理器执行指令

（1）在第 1 个时钟周期，处理器加载第 0 条指令和一条无条件分支指令；

（2）在第 2 个时钟周期，处理器执行第 0 条指令；

(3) 在第 3 个时钟周期,处理器在取出第 1 条指令和第 2 条指令的同时,执行分支指令。

⋮

在第 $N+2$ 个时钟周期,处理器丢弃第 1 条指令和第 2 条指令,并取出第 N 条指令和第 $N+1$ 条指令。

前面提到,在 Cortex-M0、Cortex-M3 和 Cortex-M4 处理器中实现了三级流水线,即取指、译码和执行指令。分支影子指令的数量更多(最多达到 4 条 16 位指令)。

3. 处理器核的访问方式

如图 3.4 所示,Cortex-M0+既没有缓存,也没有内部 RAM。因此,任何取指交易都会指向 AHB-Lite 接口,并且任何数据访问都会指向 AHB-Lite 接口或单周期 I/O 端口。

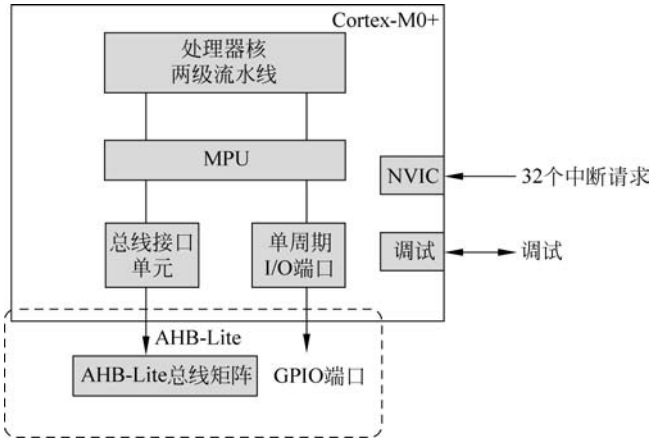

图 3.4　Cortex-M0+架构

注:STM32G0 在处理器外实现了片上系统级(System on Chip,SoC)的缓存。

AHB-Lite 主端口连接到总线矩阵,使得 CPU 可以访问存储器和外设。由于交易是在 AHB-Lite 上进行流水处理,因此最佳的吞吐量是每个时钟 32 位数据或指令,同时有最小的两个时钟延迟。

Cortex-M0+还具有单周期 I/O 端口,使 CPU 能够以一个时钟的延迟访问数据。

一个外部译码逻辑决定将数据访问指向这个端口的地址范围。在 STM32G0 系列 MCU 中,单周期 I/O 端口用于访问通用 I/O 端口(General-Purpose Input & Output, GPIO)寄存器,从而使这些端口能够以处理器频率工作。

当加载或保存指令的地址未落入单周期 I/O 端口地址范围内时,将在 AHB-Lite 端口上执行交易,从而防止 CPU 在同一时钟取指。

当加载或保存指令的地址落入单周期 I/O 端口地址范围内时,在该端口上执行交易,并可能与取指同时进行。

注:交易(transaction)是指执行某项任务的双方为了达成/实现最终的目的,而进行的一个协商或协调的过程,在有些书籍中将其翻译为“事务”。在本书中,使用“交易”这个词更能体现英文单词原本的含义。

3.1.2　系统级接口

Cortex-M0+处理器使用 AMBA 技术提供单个系统级接口,以提供高速、低延迟的存储器访问。

Cortex-M0＋处理器具有一个可选的 MPU，可以提供细粒度的存储器控制、使得应用程序可以使用多个特权级，并根据任务分割和保护代码、数据和堆栈。在许多嵌入式应用（例如汽车系统）中，此类要求变得非常重要。

3.1.3 可配置的调试

Cortex-M0＋处理器实现了完整的硬件调试解决方案，并具有广泛的硬件断点和观察点选项。通过一个具有两个引脚的串行线调试（Serial Wire Debug，SWD）端口，该系统可提供对处理器、存储器和外设的可视性，非常适合微控制器和其他小封装器件。

3.1.4 核心外设

核心外设是与 Cortex-M0＋处理器核紧密耦合的外部功能部件。

1. 嵌套向量中断控制器

NVIC 是一个嵌入的中断控制器，它提供了 32 个可屏蔽的中断通道和 4 个可编程的优先级控制，支持低延迟的异常和中断处理。此外，还提供了电源管理控制功能。

2. 系统控制块

系统控制块（System Control Block，SCB）是程序员与处理器的模型接口。它提供系统的实现信息和系统控制，包括配置、控制以及系统异常的报告。

3. 系统定时器

系统定时器 SysTick 是一个 24 位的向下计数器。将该定时器用作一个实时操作系统（Real Time Operating System，RTOS）滴答定时器或作为一个简单的计数器。

4. 存储器保护单元

存储器保护单元（Memory Protection Unit，MPU）通过定义不同存储器区域的存储属性来提高系统可靠性。它提供最多 8 个不同的区域以及一个可选的预定义背景区域。

5. I/O 端口

I/O 端口提供单周期加载，并保存到紧耦合的外设。

思考与练习 3-1：请说明 Cortex-M0＋处理器核的主要性能参数。

思考与练习 3-2：请说明 Cortex-M0＋采用的流水线结构。

思考与练习 3-3：Cortex-M0＋处理器由哪两部分组成？它们各自的主要功能是什么？

视频讲解

3.2 Cortex-M0＋处理器的寄存器

本节将详细介绍 Cortex-M0＋处理器的寄存器，对于处理器的内部寄存器来说，其特点主要包括：

（1）它们用于保存和处理处理器核内暂时使用的数据。

（2）这些寄存器在 Cortex-M0＋处理器核内，因此处理器访问这些寄存器速度较快。

（3）采用加载-保存结构，即：如果需要处理保存在存储器中的数据，则需要将保存在存储器中的数据加载到一个寄存器，然后在处理器内部进行处理。在处理完这些数据后，如果需要将其重新保存到存储器时，则将这些数据重新写回到存储器中。

对于 Cortex-M0＋处理器的寄存器来说，包含寄存器组和特殊寄存器，如图 3.5 所示。下面将对这些寄存器的功能进行详细介绍。

3.2.1 通用寄存器

寄存器组提供了 16 个寄存器，这 16 个寄存器中的 R0～R12 寄存器可作为通用寄存器，

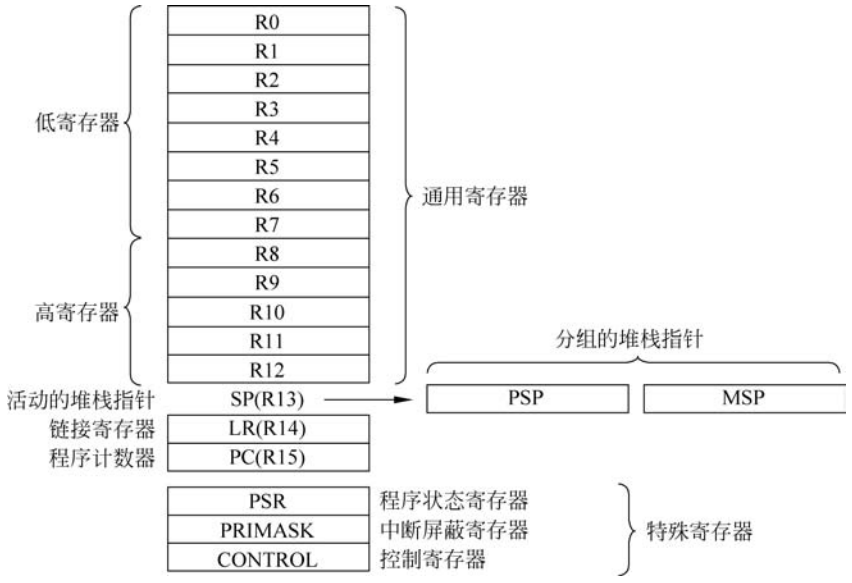

图 3.5　Cortex-M0＋处理器核内的寄存器

其中：

(1) 低寄存器，包括 R0～R7，所有指令均可访问这些寄存器。

(2) 高寄存器，包括 R8～R12，一些 Thumb 指令不可以访问这些寄存器。

3.2.2　堆栈指针

寄存器组中的 R13 寄存器可以用作堆栈指针(Stack Pointer，SP)。SP 的功能主要包括：

(1) 记录当前堆栈的地址。

(2) 当在不同的任务之间切换时，SP 可用于保存上下文(现场)。

(3) 在 Cortex-M0＋处理器核中，将 SP 进一步细分为：

① 主堆栈指针(Main Stack Pointer，MSP)。在应用程序中，需要特权访问时会使用 MSP，比如访问操作系统内核、异常句柄。

② 进程堆栈指针(Process Stack Pointer，PSP)。当没有运行一个异常句柄时，该指针可用于基本层次的应用程序代码中。

注：(1) 当复位时，处理器使用地址 0x00000000 中的值加载 MSP。

(2) 由 CONTROL 寄存器的 bit[1]来控制 SP 是用作 MSP 还是 PSP。

3.2.3　程序计数器

寄存器组中的 R15 寄存器可用作程序计数器(Program Counter，PC)，其功能主要包括：

(1) 用于记录当前指令代码的地址。

(2) 除了执行分支指令外，在其他情况下，对于 32 位指令代码来说，在每个操作时，PC 递增 4，即

$$(PC)+4\rightarrow(PC)$$

(3) 对于分支指令(如函数调用)，在将 PC 指向所指定地址的同时，将当前 PC 的值保存到链接寄存器(Link Register，LR)R14 中。

Cortex-M0＋处理器核的一个入栈和出栈操作过程，如图 3.6 所示。

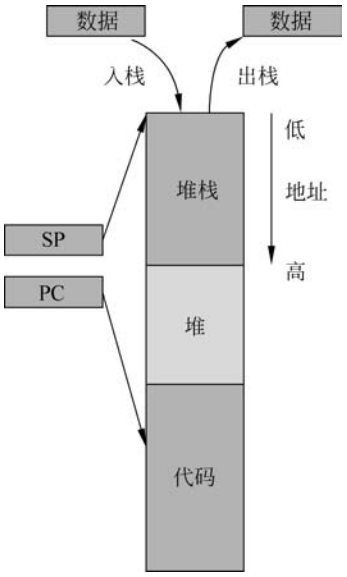

图 3.6　Cortex-M0＋堆栈操作过程

注：复位时，处理器将地址为 0x00000004 的复位向量的值加载到 PC。复位时，将该值的 bit[0]加载到 EPSR 寄存器的 T 比特位中，并且该值必须为 1。

3.2.4　链接寄存器

寄存器组中的 R14 寄存器可用作链接寄存器(Link Register，LR)，其功能主要包括：

(1) 该寄存器用于保存子程序、程序调用和异常的返回地址，如图 3.7(a)所示。

(2) 当程序调用结束后，Cortex-M0＋将 LR 中的值加载到 PC 中，如图 3.7(b)所示。

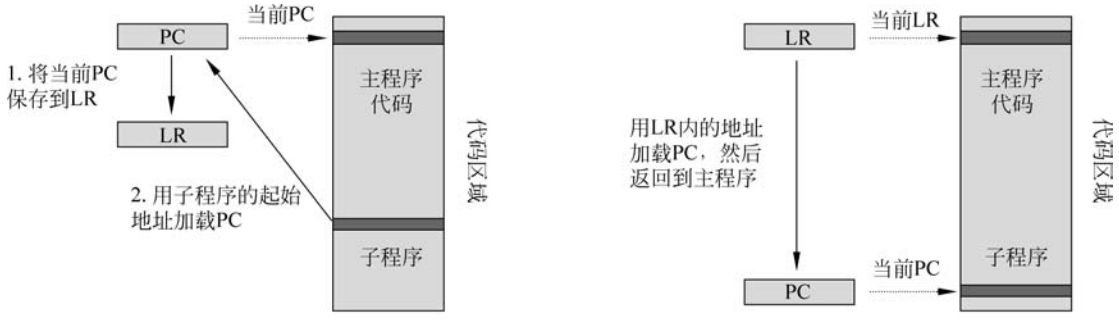

图 3.7　程序调用和返回

注：在复位时，LR 的值未知。

3.2.5　程序状态寄存器

程序状态寄存器(x Program Status Register，xPSR)用于提供执行程序的信息及 ALU 的标志位。它包含 3 个寄存器，如图 3.8 所示。

在图 3.8 中，有 3 个寄存器：

(1) 应用程序状态寄存器(Application Program Status Register，APSR)。

图 3.8　程序状态寄存器的格式

(2) 中断程序状态寄存器(Interrupt Program Status Register,IPSR)。

(3) 执行程序状态寄存器(Execution Program Status Register,EPSR)。

注：对这 3 个寄存器来说,它们可以作为一个寄存器 xPSR 来访问。比如,当发生中断的时候,xPSR 会被自动压入堆栈,从中断返回时,会自动恢复数据。在入栈和出栈时,将 xPSR 作为一个寄存器。

使用寄存器名字作为 MSR 或 MRS 指令的参数,可以单独访问这些寄存器,也可以将任意两个或所有 3 个寄存器组合起来以访问这些寄存器,如表 3.1 所示。例如：

(1) 在 MRS 指令中,使用 PSR 来读取所有寄存器。

(2) 在 MSR 指令中,使用 APSR 来写入 APSR。

表 3.1　PSR 寄存器的组合

寄　存　器	类　　型	组　　合
PSR	RW(1)(2)	APSR、EPSR 和 IPSR
IEPSR	RO	EPSR 和 IPSR
IAPSR	RW(1)	APSR 和 IPSR
EAPSR	RW(2)	APSR 和 EPSR

注：(1) 处理器忽略对 IPSR 位的写操作。

(2) 读取 EPSR 的位将返回 0,处理器忽略对这些位的写操作。

1. APSR

APSR 寄存器内保存着 ALU 操作后所产生的标志位,这些标志位包括：

(1) [31]：N,符号标志。

① 当 ALU 运算结果为负数时,将该位设置为 1。

② 当 ALU 运算结果为正数时,将该位设置为 0。

(2) [30]：Z,零标志。

① 当 ALU 运算结果等于 0 时,将该位设置为 1。

② 当 ALU 运算结果不等于 0 时,将该位设置为 0。

(3) [29]：C,进位或借位标志。

当操作产生进位时,将该标志设置为 1；否则,将该标志设置为 0。在下面的情况下,会产生进位标志,包括：

① 如果相加的结果大于或等于 2^{32}；

② 如果相减的结果为正或者零；

③ 作为移位或旋转指令的结果。

(4) [28]：V,溢出标志。

对于有符号加法和减法,如果发生了有符号溢出,则将该位设置为 1；否则,设置为 0。

例如：

① 如果两个负数相加得到一个正数。

② 如果两个正数相加得到一个负数。

③ 如果负数减去正数得到一个正数。

④ 如果正数减去负数得到一个负数。

注：除了丢掉结果外，比较操作 CMP 和 CMN 分别与减法和加法操作相同。

(5) [27:0]：Reserved，保留。

在 Cortex-M0＋中，大部分数据处理指令都会更新 APSR 中的条件标志。有些指令更新所有的标志，一些指令只更新其中一些标志。如果没有更新标志，则保留最初的值。

程序开发者可以根据另一条指令中设置的条件标志来执行条件转移指令：

(1) 在更新完标志的指令之后，立即执行。

(2) 在没有更新标志的任意数量的中间指令之后。

2. IPSR

该寄存器保存当前中断服务程序(Interrupt Service Routine，ISR)的异常号。在 Cortex-M0＋中每个异常中断都有一个特定的中断编号，用于表示中断类型。在调试时，它对于识别当前中断非常有用，并且在多个中断共享一个中断处理的情况下，可以识别出其中一个中断。IPSR 的位分配，如表 3.2 所示。

表 3.2 IPSR 的位分配

位	名 字	功 能
[31:6]	-	保留
[5:0]	Exception Number (异常编号)	当前异常的编号。 0＝线程模式 1＝保留 2＝NMI 3＝硬件故障 4～10＝保留 11＝SVCall 12～13＝保留 14＝PendSV 15＝SysTick|保留 16＝IRQ0 … 47＝IRQ31 48～63＝保留

3. EPSR

该寄存器只包含一个比特位 T，用于表示 Cortex-M0＋处理器核是否处于 Thumb 状态。当应用软件尝试使用 MRS 指令直接读取 EPSR 时，总是返回 0。当尝试使用 MSR 指令写入 EPSR 时，将忽略该操作。故障句柄用于检查堆栈 PSR 中的 EPSR 值，以确定故障原因。以下方法可以将 T 比特位清零，包括：

(1) 指令 BLX、BX 和 POP{PC}。

(2) 在从异常返回时，从堆栈的 xPSR 值恢复。

(3) 一个异常入口上向量值的 bit[0]。

当 T 比特位为 0 时,尝试执行指令将导致硬件故障或锁定。

3.2.6　可中断重启指令

可中断重启指令是 LDM 和 STM、PUSH、POP 和 MULS。当执行这些指令中的其中一条指令发生中断时,处理器将放弃执行该指令。当处理完中断后,处理器从头开始重新执行指令。

3.2.7　异常屏蔽寄存器

异常屏蔽寄存器禁止处理器对异常进行处理。禁止可能会影响时序关键任务或要求原子的代码序列的异常。

要禁止或重新使能异常,需要使用 MSR 和 MRS 指令或 CPS 指令来修改 PRIMASK 的值。

3.2.8　优先级屏蔽寄存器

优先级屏蔽寄存器(Priority Mask Register,PRIMASK)的位分配如图 3.9 所示。

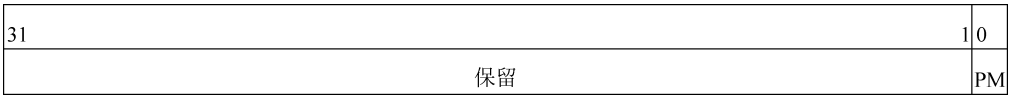

图 3.9　PRIMASK 寄存器的位分配

图中:

(1) [31:1]: Reserved(保留)。

(2) [0]: PM。可优先级排序的中断屏蔽。当:

① 0=没有影响。

② 1=阻止激活具有可配置优先级的所有异常。

3.2.9　控制寄存器

当处理器处于线程模式时,控制寄存器(CONTROL)控制使用的堆栈以及软件执行的特权级,如图 3.10 所示。

图 3.10　CONTROL 寄存器的位含义

在图 3.10 中:

(1) SPSEL 位定义当前的堆栈。当该位为 0 时,MSP 是当前的堆栈指针;当该位为 1 时,PSP 是当前的堆栈指针。

注:在句柄模式下,读取该位将返回零,忽略对该位的写操作。

(2) nPRIV 位定义线程模式特权级。当该位为 0 时,为特权级;当该位为 1 时,为非特权级。

句柄模式始终使用 MSP。因此，在句柄模式时，处理器将忽略对 CONTROL 寄存器活动堆栈指针位的显式写入操作。异常进入和返回机制会自动更新 CONTROL 寄存器。

在一个操作系统(Operating System，OS)环境中，建议以线程模式运行的线程使用线程堆栈，OS 内核和异常句柄使用主堆栈。

线程模式默认使用 MSP。要将线程模式下使用的堆栈指针切换到 PSP，使用 MSR 指令将活动的指针位设置为 1。

注：当改变堆栈指针时，软件必须在 MSR 指令后立即使用 ISB 指令。这样可以确保 ISB 之后的指令使用新的堆栈指针执行。

思考与练习 3-4：在 Cortex-M0＋处理器核中，通用寄存器的范围_____。

思考与练习 3-5：在 Cortex-M0＋处理器核中，实现堆栈指针功能的寄存器是_____。

思考与练习 3-6：在 Cortex-M0＋处理器核中，所提供的堆栈指针的类型包括_____和_____，它们各自实现的功能是_____和_____。

思考与练习 3-7：在 Cortex-M0＋处理器核中，用于实现程序计数器的寄存器是_____，程序计数器所实现的功能是_____。

思考与练习 3-8：在 Cortex-M0＋处理器核中，用于实现链接寄存器的寄存器是_____，链接寄存器的作用是_____。

思考与练习 3-9：在 Cortex-M0＋处理器核中，组合程序状态寄存器中所包含的寄存器是_____、_____和_____，它们各自的作用分别是_____、_____和_____。

思考与练习 3-10：在 Cortex-M0＋处理器核中，中断屏蔽寄存器所实现的功能是_____。

思考与练习 3-11：在 Cortex-M0＋处理器核中，控制寄存器所实现的功能是_____。

视频讲解

3.3 Cortex-M0＋存储器空间结构

本节介绍 Cortex-M0＋存储空间结构，内容包括存储空间映射架构、代码区域地址映射、SRAM 区域地址映射、外设区域地址映射、PPB 地址空间映射、SCS 地址空间映射以及系统控制和 ID 寄存器。

3.3.1 存储空间映射架构

Cortex-M0＋处理器采用了 ARMv6-M 架构，该架构支持预定义的 32 位地址空间，并细分为代码、数据和外设，以及片上和片外资源的区域，如图 3.11 所示。其中，片上是指紧耦合到处理器的资源。地址空间支持 8 个基本分区，每个分区是 0.5GB，包括：代码；SRAM；外设；两个 RAM 区域；两个设备区域；系统。

该架构分配用于系统控制、配置以及用作事件入口点或向量的物理地址。该架构定义相对表基地址的向量，该基地址在 ARMv6-M 中固定为地址 0x00000000。

地址空间 0xE0000000 到 0xFFFFFFFF 保留供系统级使用。

注：尽管默认规定了这些区域的使用方法，但是程序设计人员可以根据具体要求灵活地定义存储器映射空间，比如，访问内部私有外设总线。

ARMv6-M 地址映射关系，如表 3.3 所示。

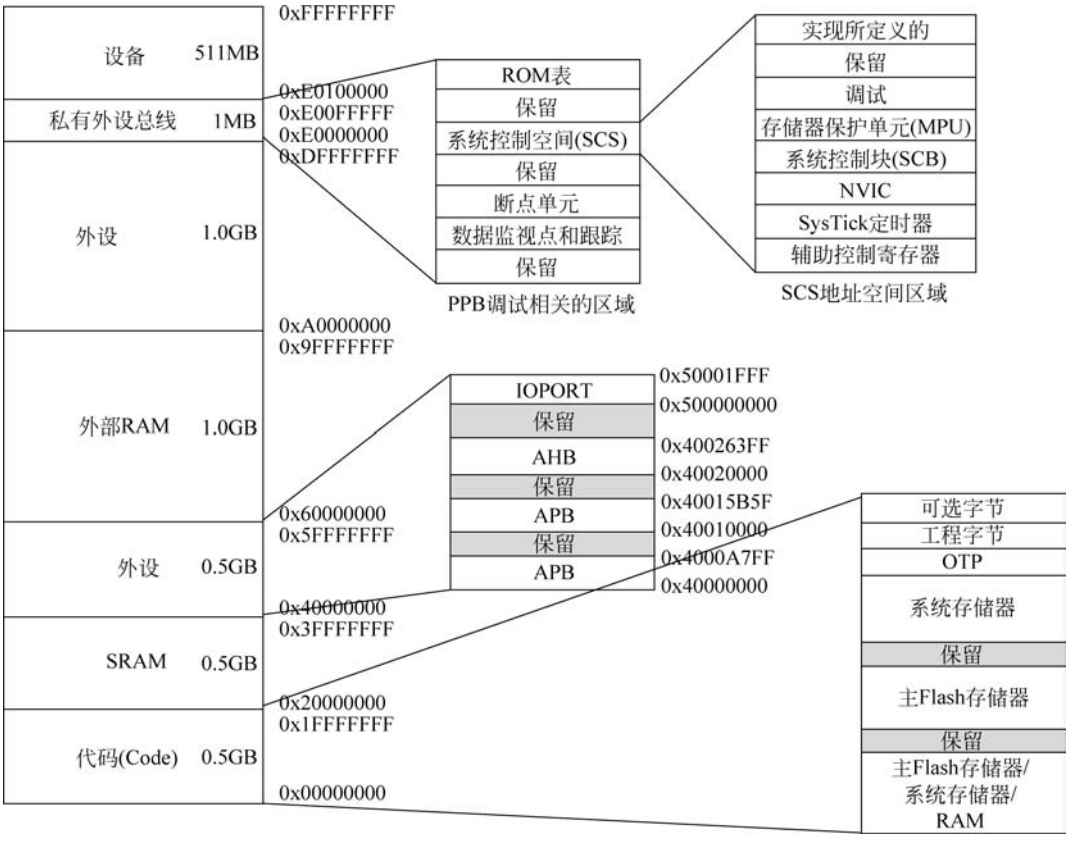

图 3.11　Cortex-M0＋处理器的存储器地址空间映射

表 3.3　ARMv6-M 地址映射关系

地　址	名　字	设备类型	XN?	缓　存	描　述
0x00000000～ 0x1FFFFFFF	代码 (Code)	标准	—	WT	通常为 ROM 或 Flash 存储器。来自地址 0x0 的存储器用于支持复位时系统启动引导的向量表。 程序代码的执行区域。此区域也可以放数据
0x20000000～ 0x3FFFFFFF	SRAM	标准	—	WBWA	SRAM 区域通常用于片上 RAM。 用于数据(比如堆或堆栈)的可执行区域。此区域也可以放代码
0x40000000～ 0x5FFFFFFF	外设	设备	XN	—	片上外设地址空间。 外部的设备存储器。包括 APB 和 AHB 外设
0x60000000～ 0x7FFFFFFF	RAM	标准	—	WBWA	具有回写功能的存储器,用于 L2/L3 缓存支持的写分配缓存属性。 用于数据的可执行区域
0x80000000～ 0x9FFFFFFF	RAM	标准	—	WT	具有直接写缓存属性的存储器。 用于数据的可执行区域

续表

地　址	名　字	设备类型	XN?	缓　存	描　述
0xA0000000～0xBFFFFFFF	设备	设备，可共享	XN	—	共享设备空间
0xC0000000～0xDFFFFFFF	设备	设备，不可共享	XN	—	非共享设备空间
0xE0000000～0xFFFFFFFF	系统	详见表 3.8 和表 3.9	XN	—	用于私有外设总线和供应商系统外设的系统段

其中：

(1) XN 表示从不执行区域(execute never region)。从 XN 区域执行代码的任何尝试都会出错，从而生成硬件故障异常。

(2) 缓存列指示用于标准存储器区域、内部和外部缓存的缓存策略，以支持系统缓存。声明的缓存类型可以降级，但不能升级。

① 直接写(Write-Through，WT)，可以看作非缓存。

② 回写和写分配(Write-Back，Write Allocate，WBWA)，可以看作直接写或非缓存。

(3) 在设备类型一列中，可共享(shareable)表示该区域支持同一存储器域中多个代理共享使用。这些代理可以是处理器和 DMA 代理的任意组合；强顺序(Strongly-Ordered，SO)表示强顺序存储器。强顺序存储器总是可共享的；标准(normal)表示处理器可以对交易进行重新排序以提高效率或者执行预测读取；设备(device)表示处理器将保持相对于到设备或强顺序存储器的其他交易的顺序。

注：(1) ARMv6-M 不支持诸如 LDREX 或 STREX 之类的互斥访问指令，也不支持任何形式的原子交换指令。在使用共享存储器的多处理环境中，软件必须考虑到这一点。

(2) 代码、SRAM 和外部 RAM 区域都可以保存程序。

(3) MPU 可以覆盖该部分介绍的默认存储器访问行为。

3.3.2　代码区域地址映射

代码(Code)区域地址映射关系，如表 3.4 所示。

表 3.4　Code 区域地址映射关系(STM32G071xx 和 STM32G081xx)

类　型	边界地址	大小/B	存储器功能
Code(代码)	0x1FFF 7880～0x1FFF FFFF	～34K	保留
	0x1FFF 7800～0x1FFF 787F	128	选项字节(由用户根据应用需求配置)
	0x1FFF 7500～0x1FFF 77FF	768	工程字节
	0x1FFF 7400～0x1FFF 74FF	256	保留
	0x1FFF 7000～0x1FFF 73FF	1K	OTP
	0x1FFF 0000～0x1FFF 6FFF	28K	系统存储器
	0x0802 0000～0x1FFF D7FF	～384M	保留
	0x0800 0000～0x0801 FFFF	128K	主 Flash 存储器
	0x0002 0000～0x07FF FFFF	～8M	保留
	0x0000 0000～0x0001 FFFF	128K	主 Flash 存储器，系统存储器或 SRAM，取决于启动引导配置(1)

注：(1) 在 STM32G0x1 中，可以通过 BOOT0 引脚，FLASH_SECR 寄存器中的 BOOT_LOCK 位以及用户选项自己中的引导配置位 nBOOT1、BOOT_SEL 和 nBOOT0 选择 3 种不同的引导模式，如表 3.5 所示。

表 3.5 启动模式配置

启动模式配置					选择的启动引导区域
BOOT_LOCK	nBOOT1	BOOT0	nBOOT_SEL	nBOOT0	
0	×	0	0	×	主 Flash 存储器
0	1	1	0	×	系统存储器
0	0	1	0	×	嵌入的 SRAM
0	×	×	1	1	主 Flash 存储器
0	1	×	1	0	系统存储器
0	0	×	1	0	嵌入的 SRAM
1	×	×	×	×	强制的主 Flash 存储器

Flash 存储器的构成形式为 72 位宽的存储单元(64 位加 8 个 ECC 位),可用于保存代码和数据常量。Flash 存储器的组织方式如下:

(1) 一个主存储器块,包含 64 个(具体数量和器件型号有关)2KB 的页面,每页有 8 行,每行 256 字节。

(2) 信息块。包含:

① 在系统存储器模式中,CPU 从系统存储器启动引导。该区域是保留区域,包含用于通过以下接口之一对 Flash 存储器进行重新编程的启动引导程序,这些接口包括 USART1、USART2、I2C1 和 I2C2(适用于所有器件),USART3、SPI1 和 SPI2(适用于 STM32G071xx 和 STM32G081xx、STM32G0B1xx 和 STM32G0C1xx),以及通过 USB(DFU)和 FDCAN2(适用于 STM32G0B1xx 和 STM32G0C1xx)。在生产线上,对芯片进行编程并提供保护,以防止伪造的写/擦除操作。

② 1KB(128 个双字)一次性可编程(One-Time Programmable,OTP)用于用户数据。OTP 数据无法删除,只能写入一次。如果只有 1 位为 0,则即使值 0x0000 0000 0000 0000,也无法再写入整个双字(64 位)。

当读出保护机制(ReaDout Protection,RDP)级别为 1,并且引导源不是主 Flash 存储器区域时,无法读取 OTP 区域。

③ 用于用户配置的选项字节。

复位后,在 SYSCLK 的第四个上升沿锁存引导模式配置。用户可以设置与所需引导模式相关的引导模式配置。

当从待机模式退出时,也会重新采样引导模式配置。因此,在待机模式下必须保持所要求的启动引导模式。当从待机模式退出时,CPU 从地址 0x00000000 获取堆栈顶部的值,然后从地址 0x00000004 的位置所在的启动存储器来启动代码。

根据所选择的启动引导模式,可以按如下方式访问主 Flash 存储器、系统存储器或 SRAM:

(1) 从主 Flash 存储器启动:主 Flash 存储器在启动存储器空间(0x0000 0000)具有别名,但是仍可以从其原始存储空间(0x08000000)访问。换句话说,可以从地址 0x00000000 或 0x08000000 开始访问 Flash 存储器内容。

(2) 从系统存储器启动:系统存储器在启动引导存储器空间(0x00000000)中是别名,但仍可以从其原始的存储器空间 0x1FFF0000 访问。

(3) 从嵌入 SRAM 启动:SRAM 在引导存储器空间(0x00000000)中具有别名,但仍可以

从其原始存储空间(0x2000 0000)对其进行访问。

3.3.3 SRAM区域地址映射

SRAM区域地址映射关系，如表3.6所示。

表3.6 SRAM区域地址映射关系

类 型	边界地址	大小/B	存储器功能
SRAM	0x2000 9000～0x3FFF FFFF	～512M	保留
	0x2000 0000～0x2000 8FFF	36K	SRAM

STM32G071x8/xB器件提供了32KB的具有奇偶校验的嵌入式SRAM。硬件奇偶校验可以检测到存储器的数据错误，这将有助于提高应用程序的功能安全性。

当由于应用程序的安全性要求不高而不需要奇偶校验保护时，可以将奇偶效验存储位用作附加的SRAM，以将其总大小增加到36KB。

片内嵌入SRAM的优势是可以以零等待状态和CPU的时钟速度读写存储器。

3.3.4 外设区域地址映射

外设区域地址映射关系(不包括Cortex-M0＋内部外设)，如表3.7所示。

表3.7 外设区域地址映射关系

总 线	边界地址	大小/B	外 设
IOPORT	0x50001800～0x5FFFFFFF	～256M	保留
	0x50001400～0x500017FF	1K	GPIOF
	0x50001000～0x500013FF	1K	GPIOE
	0x50000C00～0x50000FFF	1K	GPIOD
	0x50000800～0x50000BFF	1K	GPIOC
	0x50000400～0x500007FF	1K	GPIOB
	0x50000000～0x500003FF	1K	GPIOA
AHB	0x40026400～0x4FFFFFFF	～256M	保留
	0x40026000～0x400263FF	1K	AES
	0x40025400～0x40025FFF	3K	保留
	0x40025000～0x400253FF	1K	RNG
	0x40023400～0x40024FFF	3K	保留
	0x40023000～0x400233FF	1K	CRC
	0x40022400～0x40022FFF	3K	保留
	0x40022000～0x400223FF	1K	FLASH
	0x40021C00～0x40021FFF	3K	保留
	0x40021800～0x40021BFF	1K	EXTI
	0x40021400～0x400217FF	1K	保留
	0x40021000～0x400213FF	1K	RCC
	0x40020C00～0x40020FFF	1K	保留
	0x40020800～0x40020BFF	2K	DMAMUX
	0x40020400～0x400207FF	1K	DMA2
	0x40020000～0x400203FF	1K	DMA1

续表

总　线	边界地址	大小/B	外　设
APB	0x40015C00～0x4001FFFF	32K	保留
	0x40015800～0x40015BFF	1K	DBG
	0x40014C00～0x400157FF	3K	保留
	0x40014800～0x40014BFF	1K	TIM17
	0x40014400～0x400147FF	1K	TIM16
	0x40014000～0x400143FF	1K	TIM15
	0x40013C00～0x40013FFF	1K	USART6
	0x40013800～0x40013BFF	1K	USART1
	0x40013400～0x400137FF	1K	保留
	0x40013000～0x400133FF	1K	SPI1/I2S1
	0x40012C00～0x40012FFF	1K	TIMI
	0x40012800～0x40012BFF	1K	保留
	0x40012400～0x400127FF	1K	ADC
	0x40010400～0x400123FF	8K	保留
	0x40010200～0x400103FF	1K	COMP
	0x40010080～0x400101FF		SYSCFG(ITLINE)(1)
	0x40010030～0x4001007F		VREFBUF
	0x40010000～0x4001002F		SYSCFG
	0x4000BC00～0x4000FFFF	17K	保留
	0x4000B400～0x4000BBFF	2K	FDCAN 消息 RAM
	0x4000B000～0x4000B3FF	1K	TAMP(＋BKP 寄存器)
	0x4000A800～0x4000AFFF	2K	保留
	0x4000A400～0x4000A7FF	1K	UCPD2
	0x4000A000～0x4000A3FF	1K	UCPD1
	0x40009C00～0x40009FFF	1K	USB RAM2
	0x40009800～0x40009BFF	1K	USB RAM1
	0x40009400～0x400097FF	1K	LPTIM2
	0x40008C00～0x400093FF	2K	保留
	0x40008800～0x40008BFF	1K	I2C3
	0x40008400～0x400087FF	1K	LPUART2
	0x40008000～0x400083FF	1K	LPUART1
	0x40007C00～0x40007FFF	1K	LPTIM1
	0x40007800～0x40007BFF	1K	CEC
	0x40007400～0x400077FF	1K	DAC
	0x40007000～0x400073FF	1K	PWR
	0x40006C00～0x40006FFF	1K	CRS
	0x40006800～0x40006BFF	1K	FDCAN2
	0x40006400～0x400067FF	1K	FDCAN1
	0x40006000～0x400063FF	1K	保留
	0x40005C00～0x40005FFF	1K	USB
	0x40005800～0x40005BFF	1K	I2C2
	0x40005400～0x400057FF	1K	I2C1
	0x40005000～0x400053FF	1K	USART5
	0x40004C00～0x40004FFF	1K	USART4

续表

总　线	边界地址	大小/B	外　设
APB	0x40004800～0x40004BFF	1K	USART3
	0x40004400～0x400047FF	1K	USART2
	0x40004000～0x400043FF	1K	保留
	0x40003C00～0x40003FFF	1K	SPI3
	0x40003800～0x40003BFF	1K	SPI2/I2S2
	0x40003400～0x400037FF	1K	保留
	0x40003000～0x400033FF	1K	IWDG
	0x40002C00～0x40002FFF	1K	WWDG
	0x40002800～0x40002BFF	1K	RTC
	0x40002400～0x400027FF	1K	保留
	0x40002000～0x400023FF	1K	TIM14
	0x40001800～0x40001FFF	2K	保留
	0x40001400～0x400017FF	1K	TIM7
	0x40001000～0x400013FF	1K	TIM6
	0x40000C00～0x40000FFF	1K	保留
	0x40000800～0x40000BFF	1K	TIM4
	0x40000400～0x400007FF	1K	TIM3
	0x40000000～0x400003FF	1K	TIM2

注：(1) SYSCFG(ITLINE)寄存器使用0x40010000作为参考外设基地址。

3.3.5　PPB地址空间映射

从0xE0000000开始的存储器映射的系统区域细分，如表3.8所示。

表3.8　0xE0000000～0xFFFFFFFF区域的存储空间映射

地　址	名　字	设备类型	XN?	描　述
0xE0000000～0xE00FFFFF	PPB(1)	强顺序	XN	1MB区域保留用于PPB。该区域支持关键资源，包括系统控制空间和调试功能
0xE0100000～0xFFFFFFFF	Vendor_SYS	设备	XN	供应商系统区域

注：(1) 在所有实现中，只能通过特权方式访问。

由图3.11可知，地址空间为0xE0000000～0xE00FFFFF的区域为PPB区域，该区域的地址空间映射如表3.9所示。

表3.9　PPB地址空间映射

资　源	地址范围
数据监视点和跟踪	0xE0001000～0xE0001FFF
断点单元	0xE0002000～0xE0002FFF
系统控制空间(System Control Space，SCS)	0xE000E000～0xE000EEFF
系统控制块(System Control Block，SCB)	0xE000ED00～0xE000ED8F
调试控制块(Debug Control Block，DCB)	0xE000EDF0～0xE000EEFF
ARMv6-M ROM表	0xE00FF000～0xE00FFFFF

注：表中地址不连续的区域为保留区域。

除了 SCB、DCB 和 SCS 中的其他调试控制外，其他与调试相关的资源在 ARMv6-M 系统地址映射的 PPB 区域内分配了固定的 4KB 区域。这些资源是：

(1) 断点单元(BreakPoint Unit，BPU)。这提供了断点支持。BPU 是 ARMv7-M 中可用的 Flash 补丁和断点块(Flash Patch and Breakpoint，FPB)的子集。

(2) ROM 表。表的入口为调试器提供了一种机制，以标识实现所支持的调试基础结构。

通过 DAP 接口，可以访问这些资源以及 SCS 中的调试寄存器。

在 ARMv6-M 架构中，PPB 区域中的通用规则包括：

(1) 将该区域定义为强顺序存储器。

(2) 始终以小端方式访问寄存器，与处理器当前的端状态无关。

(3) PPB 地址空间仅支持对齐的字访问。字节和半字访问是不可预测的。

注：这与 ARMv7-M 不同，后者在某些情况下支持字节和半字访问。对于 ARMv6-M，软件必须执行读—修改—写访问序列，在该序列中，软件必须修改 PPB 存储器区域中某个字内的字节字段。

(4) 术语“设置”，表示写入 1；术语“清除”，表示写入 0。该术语适用于多个位，所有位均为写入值。

(5) 通过将 0 写入相应的寄存器位来禁用功能，并通过将 1 写入该位来使能。

(6) 在将某一位定义为在读取时清零的情况下，当该位的读取与将该位设置为 1 的事件一致时，该架构保证以下原子行为：

① 如果该位读取为 1，则通过读操作将该位清除为 0；

② 如果该位读取为 0，则将该位设置为 1，并通过后续的读取操作将其清零。

(7) 保留的寄存器或位字段必须看作为 UNK/SBZP。

(8) 对 PPB 的非特权访问会产生硬件故障错误，而不会引起 PPB 访问。

3.3.6 SCS 地址空间映射

SCS 是存储器映射的 4KB 地址空间，它提供了 32 位寄存器用于配置、状态报告和控制。SCS 寄存器分成以下几组：

(1) 系统控制和识别。

(2) CPUID 处理器标识空间。

(3) 系统配置和状态。

(4) 可选的系统定时器 SysTick。

(5) 嵌套向量中断控制器(Nested Vectored Interrupt Controller，NVIC)。

(6) 系统调试。

SCS 地址空间寄存器组的地址映射，如表 3.10 所示。

表 3.10 SCS 地址空间域的映射

组	地 址 范 围	功 能
系统控制和 ID 寄存器	0xE000E000～0xE000E00F	包括辅助控制寄存器
	0xE000ED00～0xE000ED8F	系统控制块(SCB)
	0xE000EF90～0xE000EFCF	由实现所定义的
SysTick	0xE000E010～0xE000E0FF	可选的系统定时器
NVIC	0xE000E100～0xE000ECFF	外部中断控制器

续表

组	地 址 范 围	功　　能
调试	0xE000EDF0～0xE000EEFF	调试控制和配置，只用于调试扩展
MPU	0xE000ED90～0xE000EDEF	可选的 MPU

注：保留未分配的地址空间。

在 ARMv6-M 中，SCS 中的系统控制块(SCB)提供了处理器的关键状态信息和控制功能。SCB 支持：

(1) 不同级别的软件复位控制。

(2) 通过控制表的指针来管理异常模型的基地址。

(3) 系统异常管理，包括：

① 异常使能。

② 将异常的状态设置为挂起，或则从异常中删除挂起的状态。

③ 将每个异常的状态显示为非活动的、挂起或者活动。

④ 设置可配置系统异常的优先级。

⑤ 提供其他控制功能和状态信息。

这不包括外部中断处理。NVIC 管理所有的外部中断。

(4) 当前正在执行代码和挂起的最高优先级异常的异常号。

(5) 其他控制和状态功能。

(6) 调试状态信息。这是通过调试专用寄存器区域中的控制和状态来实现的。

思考与练习 3-12 Cortex-M0＋处理器的存储器地址空间为_____。

思考与练习 3-13：Cortex-M0＋处理器的中断向量表的开始地址是_____。

思考与练习 3-14：Cortex-M0＋处理器的 PPB 所实现的功能是_____。

思考与练习 3-15：说明 Cortex-M0＋处理器 SCS 实现的功能。

思考与练习 3-16：说明 Cortex-M0＋处理器 SCB 实现的功能。

3.3.7　系统控制和 ID 寄存器

如表 3.11 所示，从存储器基地址开始按地址顺序显示系统控制和 ID 寄存器。

表 3.11　系统控制和 ID 寄存器

地　　址	名　　字	类　　型	复　　位	描　　述
0xE000E008	ACTLR	读/写	实现定义	辅助控制寄存器
0xE000ED00	CPUID	只读	实现定义	CPUID 基寄存器
0xE000ED04	ICSR	读/写	0x00000000	中断控制状态寄存器
0xE000ED08	VTOR	读/写	0x00000000(1)	向量表偏移寄存器
0xE000ED0C	AIRCR	读/写	[10:8]＝0b000	应用中断和复位控制寄存器
0xE000ED10	SCR	读/写	[4,2,1]＝0b000	系统控制寄存器
0xE000ED14	CCR	只读	[9:3]＝0b1111111	配置和控制寄存器
0xE000ED1C	SHPR2	读/写	SBZ(2)	系统句柄优先级寄存器 2
0xE000ED20	SHPR3	读/写	SBZ(3)	系统句柄优先级寄存器 3
0xE000ED24	SHCSR	读/写	0x00000000	系统句柄控制和状态寄存器
0xE000ED30	DFSR	读/写	0x00000000	调试故障状态寄存器

注：(1) 查看寄存器描述，以获取更多信息。

(2) SVCall 优先级位[31:30]是零。

(3) SysTick 位[31:30]和 PendSV 位[23:22]是零。

1. CPUID 基寄存器

CPUID 寄存器包含处理器部件号、版本和实现信息，该寄存器的位分配如图 3.12 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
IMPLEMENTER								VARIANT				ARCHITECTURE			
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PART No												REVISION			
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

图 3.12　CPUID 寄存器的位分配

图中：

(1) [31:24](IMPLEMENTER)：表示实施者代码，取值为 0x41，标识为 ARM。

(2) [23:20](VARIANT)：表示 rnpm 修订状态中的主要修订号，取值为 0x0，表示修订版 0。

(3) [19:16](ARCHITECTURE)：表示定义处理器架构的常数，取值为 0xC，表示 ARMv6-M 架构。

(4) [15:4](PART No)：表示处理器的器件号，取值为 0xC60，表示 Cortex-M0+。

(5) [3:0](REVISION)：表示 rnpm 修订状态中的小修订号 m，取值为 0x1，表示补丁 1。

2. 中断控制和状态寄存器

中断控制和状态寄存器(Interrupt Control and State Register，ICSR)提供：

(1) 不可屏蔽中断(Non-Maskable Interrupt，NMI)异常的设置挂起位。

(2) 为 PendSV 和 SysTick 异常设置挂起和清除挂起位。

此外，还给出正在挂起的最高优先级异常的异常号，该寄存器的位分配如图 3.13 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
NMIPENDSET	Reserved		PENDSVSET	PENDSVCLR	PENDSTSET	PENDSTCLR	Reserved		ISRPENDING	Reserved			VECTPENDING[6:4]		
rw			rw	w	rw	w			r				r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VECTPENDING[3:0]				RETOBASE	Reserved				VECTACTIVE[8:0]						
r	r	r	r	r			rw	rw	rw	rw	rw	rw	rw	rw	rw

图 3.13　ICSR 的位分配

在图 3.13 中：

(1) [31](NMIPENDSET)：表示 NMI 设置挂起位。当给该位写 1 时，将 NMI 异常状态修改为挂起；当读取该位时，0 表示没有挂起的 NMI 异常，1 表示正在挂起 NMI 异常。

由于 NMI 是优先级最高的异常，因此通常处理器一旦检测到对该位写入 1，就立即进入 NMI 异常句柄。当进入句柄时，将该位清零。

这意味着只有在处理器执行该句柄时重新使 NMI 信号有效时，NMI 异常句柄对该位的读取才返回 1。

(2) [30:29]：保留。

(3) [28](PENDSVSET)：设置 PENDSVSET 挂起位。当该位设置 1 时，将 PendSV 异常状态修改为挂起。读取该位时，0 表示没有挂起 PendSV 异常，1 表示正在挂起 PendSV 异常。对该位写 1 是使得 PendSV 异常挂起的唯一方法。

(4) [27](PENDSVCLR)：清除 PendSV 挂起位。当该位设置为 1 时，从 PendSV 异常中删除挂起状态。

(5) [26](PENDSTSET)：设置 SysTick 异常挂起位。当该位设置 1 时，将 SysTick 异常状态修改为挂起。读取该位时，0 表示没有挂起 SysTick 异常，1 表示正在挂起 SysTick 异常。

(6) [25](PENDSTCLR)：清除 SysTick 异常挂起位。当该位设置为 1 时，从 SysTick 异常中删除挂起状态。该位是只写位。在寄存器上读取它的值得到的结果是未知。

(7) [24:18]：保留。

(8) [17:12](VECTPENDING)：表示异常号。读取该位，返回 0 表示没有挂起的异常；非零表示优先级最高的挂起的使能的异常号。

从该值减去 16，即可获得 CMSIS IRQ 的编号，该编号表示了中断清除使能、设置使能、清除挂起、设置挂起和优先级寄存器中相应的位。

(9) [11:0]：保留。

当写 ICSR 时，如果执行下面的操作，则结果是不可预知的：

(1) 给 PENDSVSET 写 1，并且给 PENDSVCLR 位写 1。

(2) 给 PENDSTSET 写 1，并且给 PENDSTCLR 位写 1。

3. 向量表偏移寄存器

向量表偏移寄存器(Vector Table Offset Register，VTOS)表示向量表基地址与存储器地址 0x00000000 的偏移量，该寄存器的位分配如图 3.14 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
TBLOFF[31:16]															
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TBLOFF[15:7]									Reserved						
rw	rw	rw	rw	rw	rw	rw	rw	rw							

图 3.14　VTOS 的位分配

在图 3.14 中：

(1) [31:7](TBLOFF)：表示向量表的基本偏移字段。它包含表的基地址与存储器映射底部偏移量的位[31:7]。

(2) [6:0]：保留。

4. 应用中断和复位控制寄存器

应用中断和复位控制寄存器(Application Interrupt and Reset Control Register，AIRCR)为数据访问和系统的复位控制提供端状态。要写入该寄存器，必须将 0x05FA 写到 VECTKEY 字段，否则处理器将忽略该写入操作。该寄存器的位分配如图 3.15 所示。

在图 3.14 中：

(1) [31:16](VECTKEYSTAT)：VECTKEY 注册键，为注册密钥。当读取时，未知；当写入时，将 0x05FA 写到 VECTKEY，否则忽略写操作。

(2) [15](ENDIANESS)：数据端比特。当读取该位时，返回值为 0，表示小端模式。

(3) [14:3]：保留。

(4) [2](SYSRESETREQ)：系统复位请求。当给该位设置为 1 时，请求一个系统级复位。当读取该位时，返回值为 0。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
VECTKEYSTAT															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ENDIANNESS	Reserved												SYSRESETREQ	VECTCLRACTIVE	Res.
r													w	w	

图 3.15 AIRCR 的位分配

(5) [1](VECTCLRACTIVE)：保留用于调试。当读取该位时，返回值为 0。当写入该位时，必须向该位写 0，否则行为不可预测。

(6) [0]：保留。

5. 系统控制寄存器

系统控制寄存器(System Control Register，SCR)控制进入和退出低功耗的功能。SCR 的位分配，如图 3.16 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved											SEVONPEND	Res.	SLEEPDEEP	SLEEPONEXIT	Res.
											rw		rw	rw	

图 3.16 SCR 的位分配

在图 3.16 中：

(1) [31:5](Reserved)：保留。

(2) [4](SEVONPEND)：表示在挂起位上发送事件。当该位为 0 时，只有允许的中断或事件才能唤醒处理器，禁止的中断将被排除在外；当该位为 1 时，使能的事件和所有中断(包括禁止的中断)都可以唤醒处理器。

当挂起一个事件或中断时，事件信号将处理器从 WFE 唤醒。如果处理器不等待一个事件，则寄存该事件并影响下一个 WFE。

在执行 SEV 指令或一个外部事件时，也会唤醒处理器。

(3) [3](Res.)：该位必须保持清零状态。

(4) [2](SLEEPDEEP)：控制处理器在低功耗模式时使用休眠或深度休眠。当该位为 0 时，使用休眠；当该位为 1 时，使用深度休眠。

(5) [1](SLEEPONEXIT)：当从句柄模式返回到线程模式时，表示退出时休眠。将该位设置为 1 可使中断驱动的应用程序避免返回到空的主应用程序。当该位为 0 时，返回线程时不休眠；当该位为 1 时，当从中断服务程序(Interrupt Service Routine，ISR)返回线程模式时，进入休眠或深度休眠。

(6) [0](Res.)：保留，必须保持清零状态。

6. 配置和控制寄存器

配置和控制寄存器(Configuration and Control Register，CCR)是只读存储器，它指示 Cortex-M0＋处理器行为的某些方面。CCR 的位分配，如图 3.17 所示。

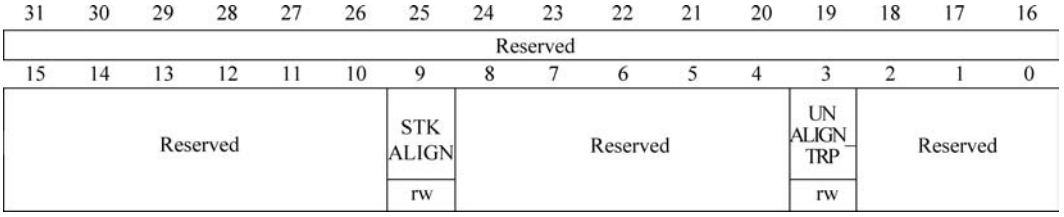

图 3.17　CCR 的位分配

在图 3.17 中：

(1) [31:10](Reserved)：保留，必须保持清零状态。

(2) [9](STKALIGN)：始终读为 1，表示在异常入口上 8 字节堆栈对齐。在异常入口处，处理器使用入栈 PSR 的第 9 位指示堆栈对齐。从异常返回时，它使用这个堆栈位恢复正确的堆栈对齐方式。

(3) [8:4](Reserved)：必须保持为清零状态。

(4) [3](UNALIGN_TRP)：总是读取为 1，指示所有未对齐的访问将产生一个硬件故障。

(5) [2:0](Res.)：必须保持为清零状态。

7. 系统句柄优先级寄存器

系统句柄优先级寄存器(System Handler Priority Register，SHPR)2 和 3，将具有可配置优先级的系统异常句柄的优先级设置为 0～192。SHPR2～SHPR3 是字访问的。SHPR2 的位分配如图 3.18 所示，SHPR3 的位分配如图 3.19 所示。

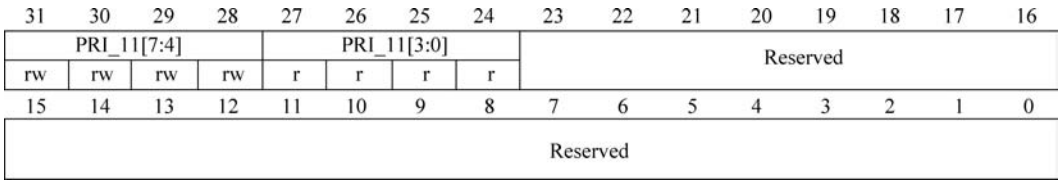

图 3.18　SHPR2 的位分配

图 3.18 中，[31:24](PRI_11)：系统句柄 11(SVCall)的优先级；[23:0]：保留，必须保持清零状态。

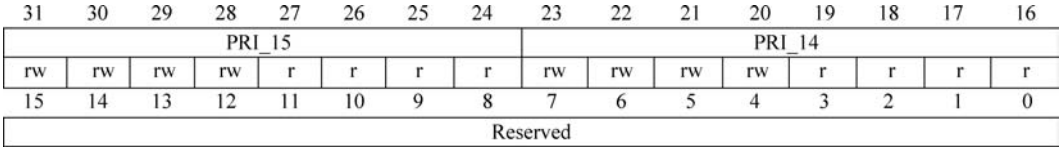

图 3.19　SHPR3 的位分配

图 3.19 中，[31:24](PRI_15)：系统句柄 15(SysTick 异常)的优先级；[23:16](PRI_14)：系统句柄 14(PendSV)的优先级。

注：当没有实现 SysTick 定时器时，该字段为保留字段。

当使用 CMSIS 访问系统异常优先级时，使用下面的 CMSIS 函数：

(1) uint32_t NVIC_GetPriority(IRQn_Type IRQn)。

(2) void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)。输入参数 IRQn 是 IRQ 的编号。

注：每个 PRI_N 字段为 8 位宽度，但是处理器仅实现每个字段的 bit[7:6]，而 bit[5:0]读取为 0，并忽略写入操作。

3.4　Cortex-M0＋的端及分配

端(Endian)是指保存在存储器中的字节顺序。根据字节在存储器中的保存顺序，将其划分为大端(Big Endian)和小端(Little Endian)。

1. 小端

对于一个 32 位字长的数据来说，最低字节保存该数据的第 0～7 位，如图 3.20(a)所示，也就是常说的“低址低字节，高址高字节”。

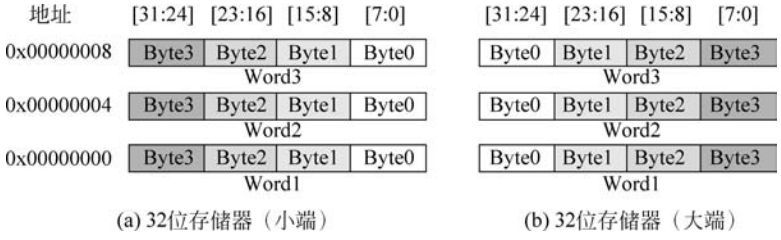

图 3.20　Cortex-M0＋小端和大端定义

2. 大端

对于一个 32 位字长的数据来说，最低字节保存该数据的第 24～31 位，如图 3.20(b)所示，也就是常说的“低址高字节，高址低字节”。

对于 Cortex-M0＋处理器来说，默认支持小端。然而，端概念只存在硬件这一层。

思考与练习 3-17：请说明在 Cortex-M0＋中端的含义。

思考与练习 3-18：请说明大端和小端的区别。


视频讲解

3.5　Cortex-M0＋处理器异常及处理

异常(Exception)是事件，它将使程序流退出当前的程序线程，然后执行和该事件相关的代码片段(子程序)。通过软件代码，可以使能或者禁止处理器核对异常事件的响应。事件可以是内部的也可以是外部的，如果事件来自外部，则称之为中断请求(Interrupt ReQuest，IRQ)。

3.5.1　异常所处的状态

每个异常均处于以下状态之一：

(1) 非活动(Inactive)。当异常处于该状态时，它既不活动也不挂起。

(2) 挂起(Pending)。当异常处于该状态时，表示它在等待处理器为它提供服务。一个来自外设或软件的中断请求可以将相应中断的状态改为挂起。

(3) 活动(Active)。处理器正在处理异常但尚未完成。异常句柄可以打断另一个异常的执行。在这种情况下，两个异常均处于活动状态。

注：异常句柄是指在异常模式中所执行的一段代码，也称为异常服务程序。如果异常由

IRQ 引起，则将其称为中断句柄(Interrupt Handler)/中断服务程序(Interrupt Service Route，ISR)。

(4) 活动和挂起。处理器正在处理异常，并且同一来源还有一个挂起的异常。

3.5.2 异常类型

在 Cortex-M0＋中，提供了不同的异常类型，以满足不同应用的需求，包括复位、不可屏蔽中断、硬件故障、请求管理调用、可挂起的系统调用、系统滴答和外部中断。

1. 复位

ARMv6-M 框架支持两级复位。复位包括：

(1) 上电复位用于复位处理器、SCS 和调试逻辑。

(2) 本地复位用于复位处理器和 SCS，不包括与调试相关的资源。

2. 不可屏蔽中断

不可屏蔽中断(Non-Maskable Interrupt，NMI)特点如下：

(1) 用户不可屏蔽 NMI。

(2) 它用于对安全性苛刻的系统中，比如工业控制或者汽车。

(3) 可以用于电源失败或者看门狗。

对于 STM32G0 来说，NMI 是由 SRAM 奇偶校验错误、Flash 存储器双 ECC 错误或时钟故障引起。

3. 硬件故障

硬件故障(HardFault)常用于处理程序执行时产生的错误，这些错误可以是尝试执行未知的操作码、总线接口或存储器系统的错误，也可以是尝试切换到 ARM 状态之类的非法操作。

4. 请求管理调用

请求管理调用(SuperVisor Call，SVC)是由 SVC 指令触发的异常。在操作系统环境中，应用程序可以使用 SVC 指令来访问 OS 内核功能和设备驱动程序。

5. 可挂起的系统调用

可挂起的系统调用(PendSV)是用于包含 OS(操作系统)的应用程序的另一个异常，SVC 异常在 SVC 指令执行后会马上开始，PendSV 在这一点上有所不同，它可以延迟执行，在 OS 上使用 PendSV 可以确保高优先级任务完成后才执行系统调度。

6. 系统滴答

NVIC 中的系统滴答(SysTick)定时器为 OS 应用可以使用的另外一个特性。几乎所有操作系统的运行都需要上下文(现场)切换，而这一过程通常需要依靠定时器来完成。Cortex-M0＋内集成了一个简单的定时器，这样使得操作系统的移植更加容易。

7. 外部中断

在 Cortex-M0＋中的 NVIC，支持最多 32 个中断请求(IRQ)。由于 STM32G0 提供了 SYSCFG 模块，使得 STM32G0 可以响应中断事件的数量大于 32 个，这是因为 SYSCFG 单元可以将几个中断组合到一个中断线上。通过读取 SYSCFG 模块中的 ITLINEx 寄存器，就可以快速确定产生中断请求的外设源。

只有用户使能外部中断后，才能使用它。如果禁止了外部中断，或者处理器正在运行另一个相同或者更高优先级的异常处理，则中断请求会被保存在挂起状态寄存器中。当处理完高优先级的中断或返回后，才能执行挂起的中断请求。对于 NVIC 来说，可接受的中断请求信号可以是高逻辑电平，也可以是中断脉冲(最少为一个时钟周期)。

注：(1) 在 MCU 外部接口中，外部中断信号可以是高电平也可以是低电平，或者可以通过编程配置。

(2) 软件可以修改外部中断的优先级，但是不能修改复位、NMI 和硬件故障的优先级。

3.5.3　异常优先级

在 Cortex-M0＋中，每个异常都有相关联的优先级，其中：

(1) 较低的优先级值意味着具有较高的优先级。

(2) 除了复位、硬件故障和 NMI 外，软件可配置其他所有异常的优先级。

如果软件没有配置任何优先级，则所有可配置优先级异常的优先级为 0。

注：可配置优先级的值为 0～192，以 64 为步长。具有固定负优先级值的复位、硬件故障和 NMI 异常始终有比其他任何异常更高的优先级。复位的优先级值为－3，NMI 的优先级值为－2，硬件故障的优先级值为－1，除此之外的其他异常(包括外设中断和软件异常)的优先级为 0～3。

为 IRQ[0]分配较高的优先级值，为 IRQ[1]分配较低的优先级值，则意味着 IRQ[1]的优先级高于 IRQ[0]。如果 IRQ[1]和 IRQ[0]均有效，则先处理 IRQ[1]，处理完后再处理 IRQ[0]。

如果多个挂起的异常具有相同的优先级，则优先处理具有最低优先级编号的异常。例如，如果 IRQ[0]和 IRQ[1]都处于挂起状态并且具有相同的优先级，则先处理 IRQ[0]。

当处理器执行一个异常句柄时，如果发生了具有更高优先级的异常，则会抢占该异常句柄。如果又发生与正在处理的异常具有相同优先级的异常，则不会抢占当前正在处理的句柄。然而，新中断的状态变为挂起。

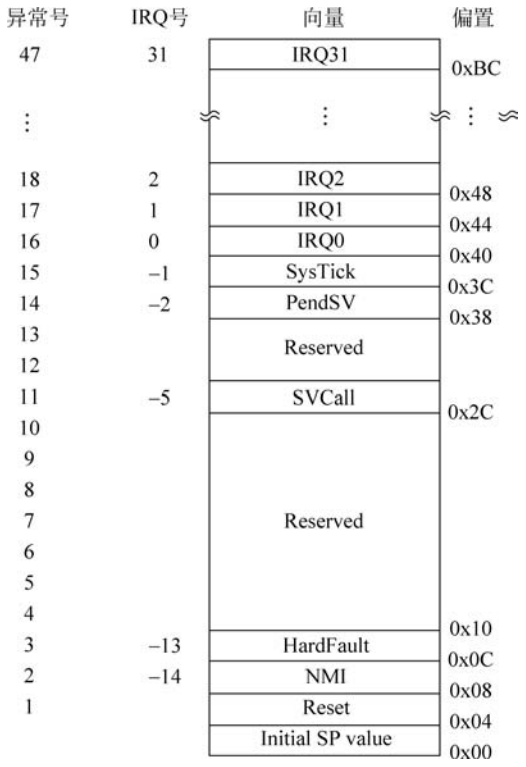

图 3.21　Cortex-M0＋向量表

3.5.4　向量表

向量表包含用于所有异常句柄的堆栈指针和起始地址(也称为异常向量)。向量表中异常向量的顺序，如图 3.21 所示。每个向量的最低有效位必须为 1，用来指示异常句柄是用 Thumb 代码编写的。

系统复位时，向量表固定在地址 0x00000000。具有特权级的软件可以写入向量表偏移寄存器(Vector Table Offset Register，VTOR)，以根据向量表的大小和 TBLOFF 设置的粒度，将向量表的起始地址重新定位到其他存储器位置。

注：为了简化软件层的应用程序设计，Cortex 微控制器软件接口标准(Cortex Microcontroller Software Interface Standard，CMSIS)只使用 IRQ 号。它对中断以外的其他异常使用负值。IPSR 返回异常号。

3.5.5 异常的进入和返回

本节介绍异常的进入和返回。包括术语说明、进入异常和异常返回。

1. 术语说明

在描述异常的处理时，会用到下面的术语。

1）抢占(preemption)

如图 3.22 所示，当处理器正在处理一个中断时，一个具有更高优先级的新请求到达时，新的异常可以抢占当前的中断。这称为嵌套的异常处理。

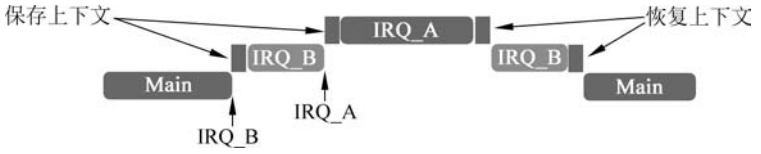

图 3.22 抢占和中断嵌套

注：图中的保存上下文(context)就是通常所说的保存现场，也就是说，在处理器进入异常句柄之前，先将进入异常句柄之前的处理器状态保存起来，处理器的状态包括寄存器以及状态标志等，这称为上文。此外，处理器还得知道在处理完中断句柄后该如何继续执行原来的程序，这称为下文。只有处理器能正确地保存上下文的信息，处理器才能在进入异常句柄后，正确地从异常句柄返回。应该说术语“上下文”比“现场”更能反映处理器处理异常的机制。

在处理完较高优先级的异常后，先前被打断的异常句柄将恢复继续执行。

Cortex-M0＋处理器内的微指令控制序列会自动地将上下文保存到当前堆栈，并在中断返回时将其恢复。

2）返回(return)

当完成处理异常句柄时，会发生这种情况，并且：

(1) 没有正在挂起需要待处理的具有足够优先级的异常。

(2) 已完成的异常句柄未处理迟到的异常。

处理器弹出堆栈，并且将处理器的状态恢复到发生中断之前的状态。

3）尾链(tail-chaining)

如图 3.23 所示，这种机制加快了异常处理的速度。在处理完一个异常句柄后，如果一个挂起的异常满足进入异常的要求，则跳过弹出堆栈的过程，并将控制权转移到新的异常句柄。

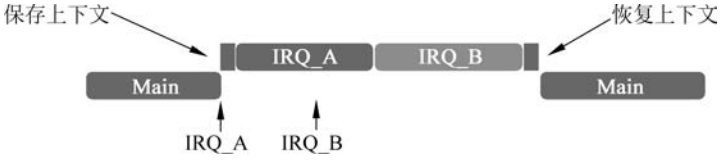

图 3.23 尾链机制

因此，将具有较低优先级(较高优先级值)的背靠背中断链接在一起，这样在处理异常句柄时，就能显著减少处理延迟并降低器件功耗。

4）迟到(late-arriving)

如图 3.24 所示，该机制可加快抢占速度。如果在保存当前异常的状态期间又发生了较高优先级的异常，则处理器将切换未处理较高优先级的异常，并为该异常启动向量获取。状态保存不受延迟到达的影响，因此对于两种异常状态，保存的状态都相同。从迟到异常的异常句柄返回时，将应用常规的尾链规则。

图 3.24　迟到机制

2. 异常进入

当存在具有足够优先级的挂起异常时，将处理挂起的异常，并且：

(1) 处理器处于线程模式。

(2) 新异常的优先级要高于正被处理的异常，在这种情况下，新异常抢占正在处理的异常。

当一个异常抢占另一个异常时，将嵌套异常。当处理器采纳(处理)一个异常时，除非异常是尾链或迟到的异常，处理器将信息压入当前的堆栈。该操作称为压栈(入栈)。如图 3.25 所示，8 个数据字的结构称为堆栈帧。堆栈帧包含以下信息：

图 3.25　堆栈帧

在压栈之后，堆栈指针指向堆栈中的最低地址。堆栈帧与双字地址对齐。堆栈帧包含返回地址。这是被中断打断的当前正在执行程序的下一条地址。当从异常返回时，将该值恢复到 PC，这样可以继续执行被打断的程序。

处理器提取向量，从向量表中读取异常句柄的起始地址。当完成压栈后，处理器开始执行异常句柄。同时处理器将 EXC_RETURN 值写入 LR。这指示哪个堆栈指针对应于堆栈帧，以及在进入异常之前处理器所处的模式。

如果在进入异常期间没有发生更高优先级的异常，则处理器开始执行异常句柄，并自动将当前挂起的中断状态修改为活动。

如果在进入异常期间，发生了一个更高优先级的异常，则处理器开始对该异常执行异常句柄，并且不会更改较早异常的挂起状态。这就是迟到的情况。

3. 异常返回

当处理器处理句柄模式，并且执行以下指令之一，尝试将 PC 设置为 EXC_RETURN 值时，将发生异常返回。

(1) 加载 PC 的 POP 指令。

（2）使用任何寄存器的 B PBX 指令。

处理器在异常入口处将 EXC_RETURN 值保存到 LR。异常机制依靠该值来检测处理器何时完成异常句柄。EXC_RETURN 值的第[31:4]位为 0xFFFFFFF。当处理器将匹配该模式的值加载到 PC 时，它将检测到该操作不是正常的分支操作，而是完成异常的最后处理。结果，它启动异常返回序列。EXC_RETURN 值的位[3:0]指示所要求返回的堆栈和处理器模式，如表 3.12 所示。

表 3.12　异常返回行为

EXC_RETURN	描　　述
0xFFFFFFF1	返回到句柄模式。 异常返回从主堆栈中得到的状态。 返回之后，使用 MSP 执行
0xFFFFFFF9	返回到线程模式。 异常返回从主堆栈中得到的状态。 返回之后，使用 MSP 执行
0xFFFFFFFD	返回到线程模式。 异常返回从进程堆栈中得到的状态。 返回之后，使用 PSP 执行
所有其他值	保留

思考与练习 3-19：请说明在 Cortex-M0＋中异常的定义以及处理异常的过程。

思考与练习 3-20：根据图 3.17，说明处理中断嵌套的过程。

思考与练习 3-21：请说明在 Cortex-M0 中向量表所实现的功能。

思考与练习 3-22：请说明在 Cortex-M0 中异常的类型。

3.5.6　NVIC 中断寄存器集

NVIC 的中断寄存器集如表 3.13 所示。本节详细介绍这些寄存器的功能。以帮助读者更好地理解处理器异常的原理和控制机制。

表 3.13　NVIC 的中断寄存器集

地　　址	名　　字	类　　型	复　位　值
0xE000E100	NVIC_ISER	读写	0x00000000
0xE000E180	NVIC_ICER	读写	0x00000000
0xE000E200	NVIC_ISPR	读写	0x00000000
0xE000E280	NVIC_ICPR	读写	0x00000000
0xE000E400～0xE000E4EF	NVIC_IPR0～NVIC_IPR7	读写	0x00000000

1. 中断设置使能寄存器

中断设置使能寄存器（NVIC Interrupt Set Enable Register，NVIC_ISER）使能中断，并显示了所使能的中断，该寄存器的位分配如图 3.26 所示。

在图 3.26 中，SETPENA[31:0]为中断设置使能位。当写入时：

① 0 表示没有影响。

② 1 表示使能中断。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPENA[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETPENA[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

图 3.26 中断设置使能寄存器的位分配

当读取时：

① 0 表示禁止中断。

② 1 表示使能中断。

如果使能了待处理的中断，则 NVIC 会根据其优先级激活该中断。如果未使能中断，则中断有效信号将中断的状态改为挂起，但 NVIC 不会激活该中断，无论其优先级如何。

2. 中断清除使能寄存器

中断清除使能寄存器(NVIC Interrupt Clear Enable Register，NVIC_ICER)禁用中断，并显示使能了哪些中断，该寄存器的位分配如图 3.27 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRENA[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRENA[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

图 3.27 中断清除使能寄存器的位分配

在图 3.27 中，CLRENA[31:0]为中断清除使能位。当写入时：

① 0 表示没有影响。

② 1 表示禁止中断。

当读取时：

① 0 表示禁止中断。

② 1 表示使能中断。

3. 中断设置挂起寄存器

中断设置挂起寄存器(NVIC Interrupt Set Pending Register，NVIC_ISPR)强制中断进入挂起状态，并指示正在挂起的中断，该寄存器的位分配如图 3.28 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
SETPEND[31:16]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs7	rs	rs	rs	rs	rs	rs
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SETPEND[15:0]															
rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs	rs

图 3.28 中断设置挂起寄存器位分配

在图 3.28 中，SETPEND[31:0]为中断设置挂起位。当写入时：

① 0 表示没有影响。

② 1 表示将中断状态改为挂起。

当读取时：

① 0 表示当前没有挂起中断。

② 1 表示当前有挂起中断。

注：将 1 写到 NVIC_ISPR 位的作用是，对于一个正在挂起的中断无效；禁用的中断将该中断的状态设置为挂起。

4. 中断清除挂起寄存器

中断清除挂起寄存器(NVIC Interrupt Clear Pending Register，NVIC_ICPR)从中断中删除挂起状态，并显示正在挂起的中断，该寄存器的位分配如图 3.29 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CLRPEND[31:16]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CLRPEND[15:0]															
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

图 3.29　中断清除挂起寄存器的位分配

在图 3.29 中，CLRPEND[31:0]为清除中断挂起位。当写入时：

① 0 表示没有影响。

② 1 表示删除挂起状态并中断。

当读取时：

① 0 表示中断没有挂起；

② 1 表示中断正在挂起。

5. 中断优先级寄存器

中断优先级寄存器(NVIC Interrupt Priority Register，NVIC_IPR0～NVIC_IPR7)为每个中断提供了 8 位的优先级字段。这些寄存器只能通过字访问，每个寄存器包含 4 个优先级字段，这 8 个寄存器的位分配如图 3.30 所示，这些位分配的含义如表 3.14 所示。

	31 24	23 16	15 8	7 0
NVIC_IPR7	PRI_31	PRI_30	PRI_29	PRI_28
⋮			⋮	
NVIC_IPRn	PRI_(4n+3)	PRI_(4n+2)	PRI_(4n+1)	PRI_(4n)
⋮			⋮	
NVIC_IPR0	PRI_3	PRI_2	PRI_1	PRI_0

图 3.30　中断优先级寄存器的位分配

表 3.14 NVIC_IPRx 位分配

比 特	名 字	功 能
[31:24]	优先级,字节偏移 3	每个优先级字段都有一个优先级值 0～192。值越小,相应的中断优先级越高。处理器仅实现每个字段的[7:6]位,[5:0]位读取为零,并忽略对[5:0]位的写入。这意味着将 255 写入优先级寄存器将会将值 192 保存到该寄存器
[24:16]	优先级,字节偏移 2	
[15:8]	优先级,字节偏移 1	
[7:0]	优先级,字节偏移 0	

3.5.7 电平和脉冲中断

Cortex-M0＋中断对电平和脉冲均敏感。脉冲中断也称为边沿触发中断。

电平敏感中断一直保持有效,直到外设将中断信号设置为无效为止。通常,发生这种情况是因为 ISR 访问外设,导致其清除了中断请求。脉冲中断时在处理器时钟的上升沿同步采样的中断信号。为了确保 NVIC 检测到中断,外设必须在至少一个时钟周期内使中断信号有效,在此期间 NVIC 检测到脉冲并锁存中断。

当处理器进入 ISR 时,它会自动从中断中删除挂起中断。对于电平敏感的中断,如果在处理器从 ISR 返回之前没有使该信号无效,则中断再次变为挂起,处理器必须再次执行它的 ISR。这意味着外设可以保持有效的中断信号,直到不需要继续服务为止。

Cortex-M0＋处理器锁存所有中断。由于以下原因之一,挂起外设中断:

(1) NVIC 检测到中断信号有效,而相应的中断使能无效。

(2) NVIC 检测到中断信号的上升沿。

(3) 软件写相应的中断设置挂起寄存器位。

挂起中断保持挂起,直到出现下面的情况之一:

(1) 处理器进入中断的 ISR。这会将中断的状态从挂起改为活动。然后:

① 对于电平敏感中断,当处理器从 ISR 返回时,NVIC 采样中断信号。如果该信号有效,则中断状态变为挂起,这可能导致处理器立即重新进入 ISR;否则,中断状态将变为非活动状态。

② 对于脉冲中断,NVIC 持续监视中断信号,如果发出脉冲信号,则中断状态将变为挂起并激活。在这种情况下,当处理器从 ISR 返回时,中断状态将变为挂起,这可能导致处理器立即重新进入 ISR。如果在处理器处于 ISR 中时没有发出中断信号,则当处理器从 ISR 返回时,中断状态变为非活动状态。

(2) 软件写入相应的中断清除挂起寄存器位。

对于电平敏感的中断,如果中断信号仍然有效,中断的状态不会变化;否则,中断状态将变为非活动状态。

对于脉冲中断,中断状态变为:

① 非活动状态(如果状态为挂起)。

② 活动(如果状态是活动和挂起)。

确保软件使用正确对齐的寄存器访问。处理器不支持对 NVIC 寄存器的非对齐访问。即使禁止一个中断,它也可以进入挂起状态。禁用中断只会阻止处理器接收该中断。

在对 VTOR 进行编程以重定位向量表之前,请确保新的向量表入口已经设置了故障句柄、NMI 和所有类似中断的使能异常。

3.6 Cortex-M0＋存储器保护单元


视频讲解

本节介绍 Cortex-M0＋内集成的存储器保护单元(Memory Protection Unit,MPU)。MPU 将

存储器映射到多个区域，并定义每个区域的位置、大小、访问权限和存储器属性。它支持：

(1) 每个区域独立的属性设置。

(2) 重叠区域。

(3) 将存储器属性导出到系统。

存储器属性会影响对区域的存储器访问的行为。Cortex-M0＋ MPU 定义：

(1) 8 个单独的存储区域(0～7)。

(2) 背景区域(background region)。

当存储区域重叠时，存储器的访问将受到编号最大的区域属性的影响。例如，区域 7 的属性优先于与区域 7 重叠的任何区域的属性。

背景区域具有与默认存储器映射相同的存储器属性，但只能从特权软件访问。

Cortex-M0＋MPU 的存储器映射是统一的。这意味着指令访问和数据访问具有相同的区域设置。如果程序访问 MPU 禁止的存储器位置，则处理器会生成硬件故障异常。

在搭载有操作系统的环境下，内核可以根据要执行的进程动态更新 MPU 区域的设置。典型情况下，一个嵌入式的操作系统使用 MPU 进行存储器保护。

可用的 MPU 属性如表 3.15 所示。

表 3.15　可用的 MPU 属性

存储器类型	共　享　性	其 他 属 性	描　　述
强顺序 (Strong-ordered)	—	—	对强顺序存储器的所有访问都按程序的顺序进行。假定所有强顺序区域都是共享的
设备 (Device)	共享的	—	多个处理器共享的存储器映射的外设
	非共享的	—	仅单个处理器使用的存储器映射的外设
普通 (Normal)	共享的	不可缓存(non-cacheable) 直接写(write-through) 可缓存的写回(cacheable write-back) 可缓存的(cacheable)	在多个处理器之间共享的普通存储器
	非共享的	不可缓存(non-cacheable) 直接写(write-through) 可缓存的写回 (cacheable write-back) 可缓存的(cacheable)	仅单个处理器使用的普通存储器

3.6.1　MPU 寄存器

在 MPU 单元中，提供了 MPU 寄存器，用于定义 MPU 的区域及其属性，如表 3.16 所示。

表 3.16　MPU 寄存器总结

地　　址	名　　字	类　　型	复　位　值
0xE0000ED90	MPU_TYPE	只读	0x00000000/0x000008000
0xE0000ED94	MPU_CTRL	读写	0x00000000
0xE0000ED98	MPU_RNR	读写	未知
0xE0000ED9C	MPU_RBAR	读写	未知
0xE0000EDA0	MPU_RASR	读写	未知

1. MPU_TYPE 寄存器

MPU 类型寄存器(MPU Type Register,MPU_TYPE)指示是否存在 MPU,如果存在,则指示它支持多个区域。该寄存器的位分配如图 3.31 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved								IREGION[7:0]							
								r	r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DREGION[7:0]								Reserved							SEPARATE
r	r	r	r	r	r	r	r								r

图 3.31 MPU_TYPE 寄存器的位分配

在图 3.31 中:

(1) [31:24]: 保留(Reserved)。

(2) [23:16]: IREGION[7:0]。表示支持的 MPU 指令区域的数量。该字段的值总是 0x00。MPU 存储器映射是统一的,由 DREGION 字段描述。

(3) [15:8]: DREGION[7:0]。表示支持的 MPU 数据区域的数量。取值为:

① 0x00=0 个区域(如果使用的器件中不包含 MPU)。

② 0x08=8 个区域(如果使用的器件中包含 MPU)。该器件使用该取值。

(4) [7:1]: 保留(Reserved)。

(5) [0]: SEPARATE。表示支持统一的或独立的指令和数据存储器映射。

① 0 表示统一的指令和数据存储器映射。该器件使用该取值。

② 1 表示独立的指令和数据存储器映射。

2. MPU_CTRL 寄存器

如图 3.32 所示,MPU 控制寄存器(MPU Control Register,MPU_CTRL)的功能包括:

(1) 使能 MPU。

(2) 使能默认的存储器映射背景区域。

(3) 当在硬件故障或不可屏蔽中断句柄中时,使能 MPU。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved													PRIVDEFENA	HFNMIENA	ENABLE
													rw	rw	rw

图 3.32 MPU_CTRL 寄存器的位分配

在图 3.32 中:

(1) [31:3]: 保留(Reserved)。

(2) [2]: PRIVDEFENA。使能特权软件访问默认的存储器映射。

① 0 表示如果使能 MPU,则禁止使用默认的存储器映射。对任何使能区域未覆盖的位置的任何存储器访问都会导致故障。

② 1 表示如果使能 MPU,则使能使用默认的存储器映射作为特权软件访问的背景区域。

注:当使能时,背景区域的作用就好像是区域编号-1。任何定义和使能区域都优先于该

默认设置。如果禁止 MPU，则处理器将忽略该位。

(3) [1]：HFNMIENA。在硬件故障和 NMI 句柄期间，使能 MPU。当使能 MPU 时，

① 0 表示在硬件故障和 NMI 句柄期间，禁止 MPU，无论 ENABLE 位的值如何设置。

② 1 表示在硬件故障和 NMI 句柄期间，使能 MPU。

当禁止 MPU 时，如果将该位设置为 1，则行为不可预测。

(4) [0]：ENABLE。使能 MPU。该位为：

① 0 表示禁止 MPU。

② 1 表示使能 MPU。

当 ENABLE 和 PRIVDEFENA 位都设置为 1 时：

(1) 特权访问的默认存储器映射如前面 3.3.1 节所述。特权软件对非使能存储区域地址的访问，其行为由默认存储器映射定义。

(2) 非特权软件对非使能存储区域地址的访问，将引起存储器管理(MemManage)故障。

XN 和强顺序规则始终应用于系统控制空间，与 ENABLE 位的值无关。

当 ENABLE 位设置为 1 时，除非 PRIVDEFENA 位设置为 1，否则至少必须使能存储器映射的一个区域用于系统运行。如果 PRIVDEFENA 位设置位 1，但是没有使能任何区域，则仅特权软件可以运行。

当 ENABLE 位设置为 0 时，系统使用默认的存储器设置，这就好像没有实现 MPU 一样。默认的存储器映射适合于特权和非特权的软件访问。

当使能 MPU 后，始终允许访问系统控制空间和向量表。是否可以访问其他区域，要根据区域和 PRIVDEFENA 是否设置为 1 判断。

除非 HFNMIENA 设置为 1，否则当处理器执行优先级为－1 或－2 的异常句柄时，不会使能 MPU。这些优先级仅在处理硬件故障或 NMI 异常时才可能。设置 HFNMIENA 位为 1 将使能 MPU。

3. MPU_RNR 寄存器

MPU 区域编号寄存器(MPU Region Number Register，MPU_RNR)选择 MPU_RBAR 和 MPU_RASR 寄存器引用的存储器区域，该寄存器的位分配如图 3.33 所示。

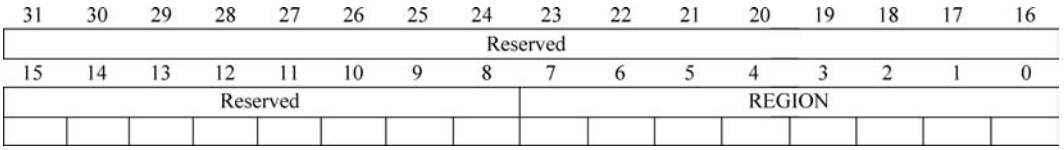

图 3.33　MPU_RNR 寄存器的位分配

在图 3.33 中：

(1) [31:8]：保留(Reserved)，必须保持清零状态。

(2) [7:0]：REGION。该字段指示 MPU_RBAR 和 MPU_RASR 寄存器引用的 MPU 区域。MPU 支持 8 个存储器区域，因此该字段允许的值为 0～7。

通常，在访问 MPU_RBAR 或 MPU_RASR 之前，需要将所要求的区域号写到该寄存器中。但是，可以通过将 MPU_RBAR 寄存器的 VALID 位置为 1 来更改区域号。

4. MPU_RBAR 寄存器

MPU 区域基地址寄存器(MPU Region Base Address Register，MPU_RBAR)定义由

MPU_RNR 选择的 MPU 区域的基地址，并且写入该寄存器可以更新 MPU_RNR 的值。将该寄存器的 VALID 位设置为 1 来写入 MPU_RBAR，以更改当前区域编号并更新 MPU_RNR。该寄存器的位分配如图 3.34 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADDR[31:N]...															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADDR[N-1:5]											VALID	REGION[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

图 3.34 MPU_RBAR 寄存器的位分配

在图 3.34 中：

(1) [31:N]：ADDR[31:N]为区域基地址字段，N 的具体取值取决于区域的大小。

(2) [N−1:5]：保留字段，硬件将其强制设置为 0。

(3) [4]：VALID。MPU 区域号有效。当写入时，

① 0：MPU_RNR 寄存器没有变化。处理器更新 MPU_RNR 中指定的区域的基地址，且忽略 REGION 字段的值。

② 1：处理器将 MPU_RNR 的值更新为 REGION 字段的值，且更新 REGION 字段中指定的区域的基地址。

当读取该字段时，总是返回 0。

(4) [3:0]：REGION[3:0]。MPU 区域字段。对于写行为，参见 VALID 字段的描述。当读取该字段的时候，返回当前区域的编号，它由 MPU_RNR 寄存器指定。

如果区域大小为 32B，ADDR 字段是[31:5]，因此没有保留字段。ADDR 字段是 MPU_RBAR 的[31:N]位。区域大小，由 MPU_RASR 指定，N 值由下式定义：

$$N=\log_2(\text{以字节为单位的区域大小})$$

如果在 MPU_RASR 中将区域大小配置为 4GB，则没有有效的 ADDR 字段。在这种情况下，区域完全占据了整个存储器映射空间，基地址为 0x00000000。

基地址与区域大小必须对齐。比如一个 64KB 区域必须对齐 64KB 的整数倍边界。比如，在 0x00010000 或 0x00020000 的边界上。

5. MPU_RASR 寄存器

MPU 区域属性和大小寄存器(MPU Region Attribute and Size Register，MPU_RASR)定义由 MPU_RNR 指定的 MPU 区域的区域大小和存储器属性，并使能该区域和任何子区域，该寄存器的位分配如图 3.35 所示。

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved			XN	Reserved	AP[2:0]			Reserved					S	C	B
			rw		rw	rw	rw			rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SRD[7:0]								Reserved		SIZE					ENABLE
rw	rw	rw	rw	rw	rw	rw	rw			rw	rw	rw	rw	rw	rw

图 3.35 MPU_RASR 寄存器的位分配

在图 3.35 中：

(1) [31:29]：保留(Reserved)。

(2) [28]：XN，指令访问禁止位。

① 0 表示使能取指令。

② 1 表示禁止取指令。

(3) [27]：Reserved，保留，硬件强制设置为 0。

(4) [26:24]：AP[2:0]，访问允许字段。详见后面的说明。

(5) [23:19]：Reserved，保留，硬件强制设置为 0。

(6) [18]：S，可共享的位，详见后面的说明。

(7) [17]：C，可缓存的位，详见后面的说明。

(8) [16]：B，可缓冲的位，详见后面的说明。

(9) [15:8]：SRD，子区域禁止位(Subregion Disable Bits，SRD)。对于该字段中的每一位：

① 0 表示使能对应的子区域。

② 1 表示禁止对应的子区域。

(10) [7:6]：保留(Reserved)，硬件强制设置为 0。

(11) [5:1]：Size，MPU 保护区域的大小。指定 MPU 区域的大小。允许的最小值为 7 (b00111)。以字节为单位的区域的大小与 SIZE 字段值之间的关系为：

$$(\text{以字节为单位的区域}) = 2^{(\text{SIZE}+1)}$$

最小允许的区域大小为 256B，对应的 SIZE 的值为 7，表 3.17 给出了 SIZE 字段值与对应的区域大小，以及 MPU_RBAR 中的 N 值。

表 3.17 SIZE 字段值的对应关系

SIZE 的值	区域大小	N 的值	注释
b00111(7)	256B	8	允许的最小值
b01001(9)	1KB	10	—
b10011(19)	1MB	20	—
b11101(29)	1GB	30	—
b11111(31)	4GB	32	可能的最大值

(12) [0]：ENABLE，区域使能位。当复位时，所有区域的区域使能位都将复位为 0。这使得可以对要使能的区域进行编程。

3.6.2 MPU 访问权限属性

本节介绍 MPU 的访问权限属性。MPU_RASR 的访问许可位 C、B、S、AP 和 XN 控制对相对应存储区域的访问。如果访问一个没有授权的存储器区域，则 MPU 会产生许可故障。C、B、S 编码和存储器属性之间的关系，如表 3.18 所示。AP 编码和软件特权级的关系，如表 3.19 所示。

表 3.18 C、B、S 编码

C	B	S	存储器类型	共享性	其他属性
0	0	—	强顺序	可共享	—
	1	—	设备	可共享	—

续表

C	B	S	存储器类型	共 享 性	其 他 属 性
1	0	0	普通	不可共享	内部和外部直接写。没有写分配
		1		可共享	
	1	0	普通	不可共享	内部和外部写回。没有写分配
		1		可共享	

表 3.19 AP 编码

AP[2:0]	特 权 权 限	非特权权限	功 能
000	无法访问	无法访问	所有的访问产生权限故障
001	读写	无法访问	只能由特权权限的软件访问
010	读写	只读	由非特权权限的软件写将产生权限故障
011	读写	读写	完全访问
100	不可预知	不可预知	保留
101	只读	无法访问	只能由特权权限的软件读取
110	只读	只读	只读。由特权或非特权权限软件
111	只读	只读	只读。由特权或非特权权限软件

3.6.3 更新 MPU 区域

要更新一个 MPU 区域的属性，需要更新 MPU_RNR、MPU_RBAR 和 MPU_RASR 寄存器。

注：建议参考第 5 章的内容来学习本节内容。

假设寄存器 R1 保存着区域的编号，寄存器 R2 保存大小/使能，寄存器 R3 保存属性，寄存器 R4 保存地址。则更新 MPU 区域的指令如下：

```
LDR R0, = MPU_RNR                ; 0xE000ED98, MPU 区域编号寄存器
STR R1, [R0, #0x0]               ;区域编号
STR R4, [R0, #0x4]               ;区域基地址
STRH R2, [R0, #0x8]              ;区域大小和使能
STRH R3, [R0, #0xA]              ;区域属性
```

软件必须使用存储器屏障指令：

(1) 在设置 MPU 之前，如果可能存在未完成的存储器传输时(例如具有缓冲性质的写操作)，可能会受到 MPU 设置更改的影响。

(2) 在设置 MPU 之后，如果其中包含存储器传输，则必须使用新的 MPU 设置。

但是，如果 MPU 设置过程起始于进入异常句柄，或者后面跟着异常返回时，则不需要指令同步屏障指令，因为异常进入和异常返回机制会引起存储器屏障行为。

例如，如果希望所有存储器访问行为在编程序列后立即生效，则使用 DSB 指令和 ISB 指令。在改变 MPU 设置后，比如在上下文切换结束时，就需要 DSB 指令。如果代码编程 MPU 区域或者使用分支或调用进入了 MPU 区域，则要求使用 ISB。如果使用从异常返回或通过采纳一个异常来进入程序序列，则不需要 ISB。

3.6.4 子区域及用法

区域被分为 8 个大小相等的子区域。在 MPU_RASR 的 SRD 字段中设置相应的位以禁

用子区域。SRD的最低有效位控制第一个子区域，最高有效位控制最后一个子区域。禁用子区域意味着与禁用范围匹配的另一个区域将匹配。如果没有其他启用的区域与禁用的子区域重叠，则MPU发生故障。下面给出一个使用例子，如图3.36所示。在该例子中，两个带有相同地址的区域重叠。区域1是128KB，区域2是512KB。为了确保区域1的属性能应用于区域2的第128KB区域，将区域2的SRD字段设置为b00000011，以禁止前两个子区域。

图3.36　子区域及用法

3.6.5　MPU设计技巧和提示

为了避免出现不期望的行为，在更新中断句柄可能访问的区域属性之前禁止中断。当设置MPU时，如果先前已经对MPU进行过编程，禁用未使用的区域，以阻止任何先前的区域设置影响新的MPU设置。

通常，微控制器只有一个处理器并且没有高速缓存。在这样的系统中对MPU编程，如表3.20所示。

表3.20　微控制器的存储器区域属性

存储器区域	C	B	S	存储器类型和属性
Flash存储器	1	0	0	普通存储器，非共享，直接写
内部SRAM	1	0	1	普通存储器，可共享，直接写
外部SRAM	1	1	1	普通存储器，可共享，写回，写分配
外设	0	1	1	设备存储器，可共享

在大多数微控制器实现中，可共享性和缓存策略不会影响系统行为。但是，将这些设置用于MPU区域可以使应用程序代码更具有可移植性。表3.20给出的值用于典型情况。在特殊系统中，例如多处理器设计或具有单独DMA引擎的设计中，可共享性属性非常重要。在这种情况下，请参考存储设备制造商的建议。