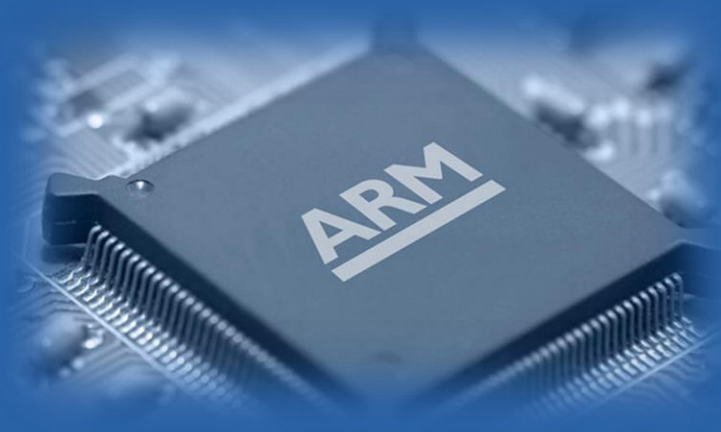




微机原理与接口技术

第四讲 ARM 概 述



```
bl    __isoc99_scanf
ldr   r2, [sp, #12]
ldr   r3, [sp, #8]
cmp   r2, r3
bgt   .L7
ldrlt    r1, [sp, #4]
ldrge    r1, [sp, #4]
sublt    r1, r1, #1
strlt    r1, [sp, #4]
```



第2章 ARM 处理器

2.1 ARM及其应用

2.2 ARM处理器的体系

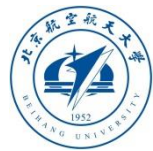
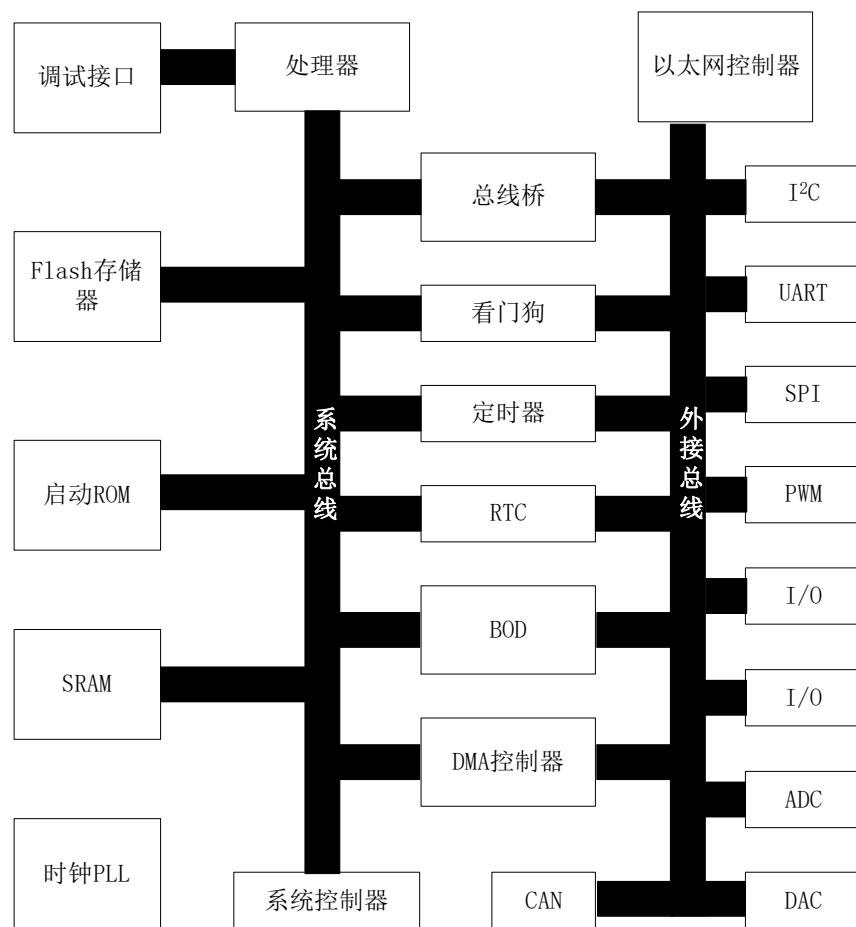
2.3 ARM的编程模型





ARM微处理器特点及功能结构

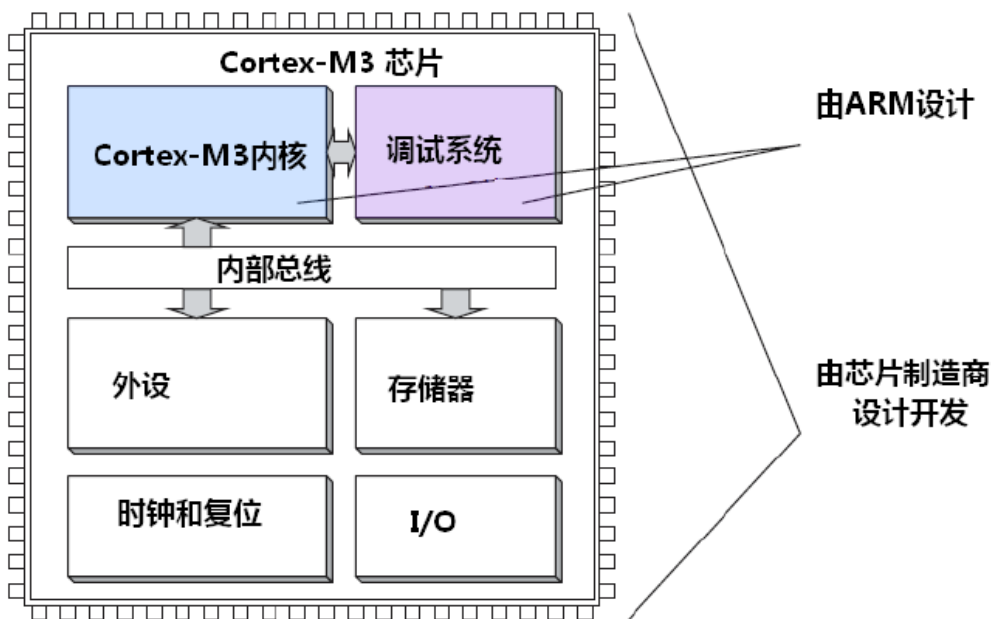
基于ARM架构的微控制器组成如图所示





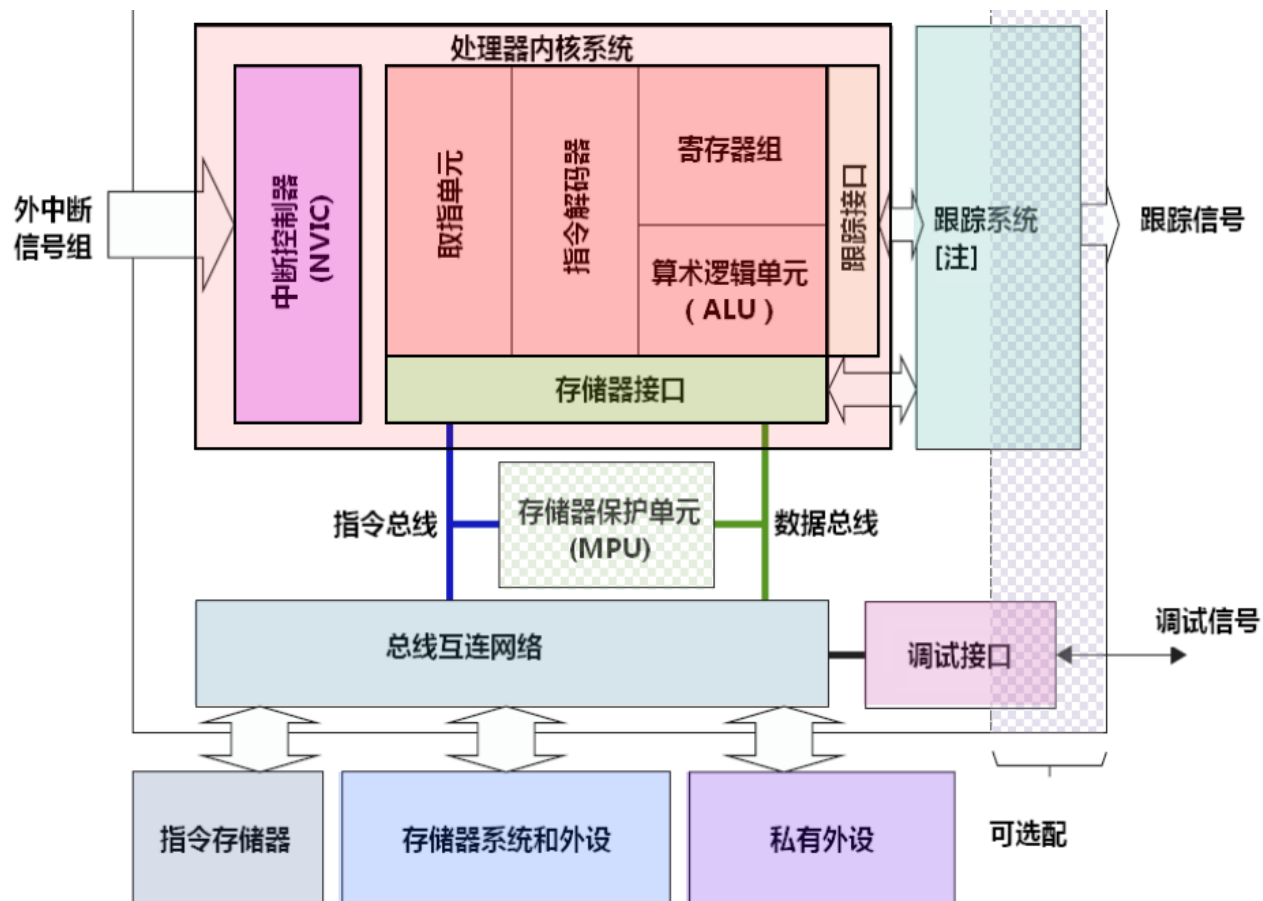
ARM体系结构及特点

- 任何微处理器都是由至少内核、存储器、总线、IO构成。ARM公司的芯片特点是内核部分都是统一的，由ARM设计，但是其它部分各个芯片制造商可以有自己的设计。





Cortex-M内部图



- 寄存器
- ALU
- 存储器
- 总线

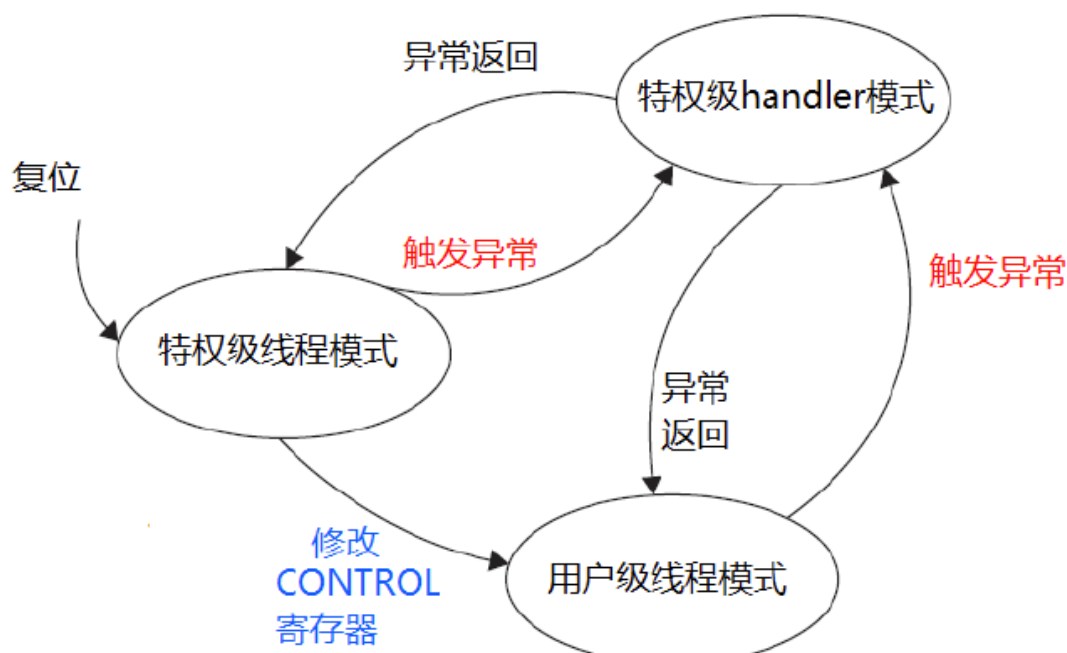




2.3 编程模型

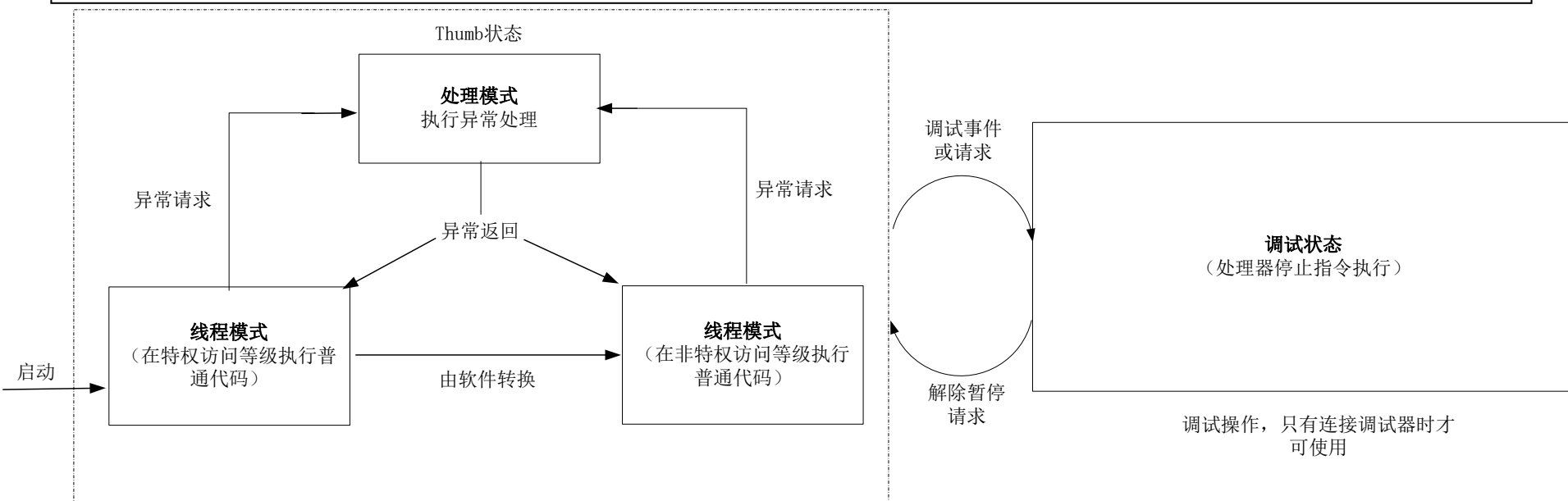
Cortex-M处理的编程模型统一

- 只有2种模式。
- 处理器（handler）模式和线程（thread）模式



2.3.1 Cortex-M微处理器工作状态和模式

Cortex-M微处理器有两种工作状态和两种模式，同时处理器还可以区分特权和非特权访问等级。特权访问等级可以访问处理器中的所有资源，而非特权访问等级下有些存储器区域是不能访问的，有些操作也是无法进行的。



2.3.1 Cortex-M微处理器工作状态和模式

1.工作状态

第一种为调试状态：由于某种原因使处理器被暂停后，就进入调试状态并停止指令的执行。原因可以是调试事件或触发断点等。

第二种为Thumb状态：当处理器执行程序代码(Thumb指令)时，就处于Thumb状态。

2.3.1 Cortex-M微处理器工作状态和模式

调试状态仅用于调试操作，可以通过两种方式进入调试状态：调试器发起的暂停请求，或处理器中的调试部件产生的调试事件。在此状态下，调试器可以访问或修改处理器寄存器的数值。无论在Thumb状态还是调试状态，调试器都可以访问处理器内外的外设和系统存储器。

2.3.1 Cortex-M微处理器工作状态和模式

2、工作模式

Cortex-M微处理器支持两种工作模式：线程模式和处理模式。

线程模式(thread mode)：在复位或异常返回时进入该模式。Cortex-M微处理器处于线程模式时既能使用特权级，也可以使用用户级（非特权）代码。

处理模式(handler mode)：执行中断服务程序等异常处理。在处理模式下，处理器具有特权访问等级。

通过软件可以将处理器从特权线程模式切换到非特权线程模式，但是无法从非特权线程模式切换到特权线程模式。如果要进行这种切换的话，处理器需要借助于异常机制。

Cortex-M微处理器在启动后默认处于特权线程模式以及Thumb状态。

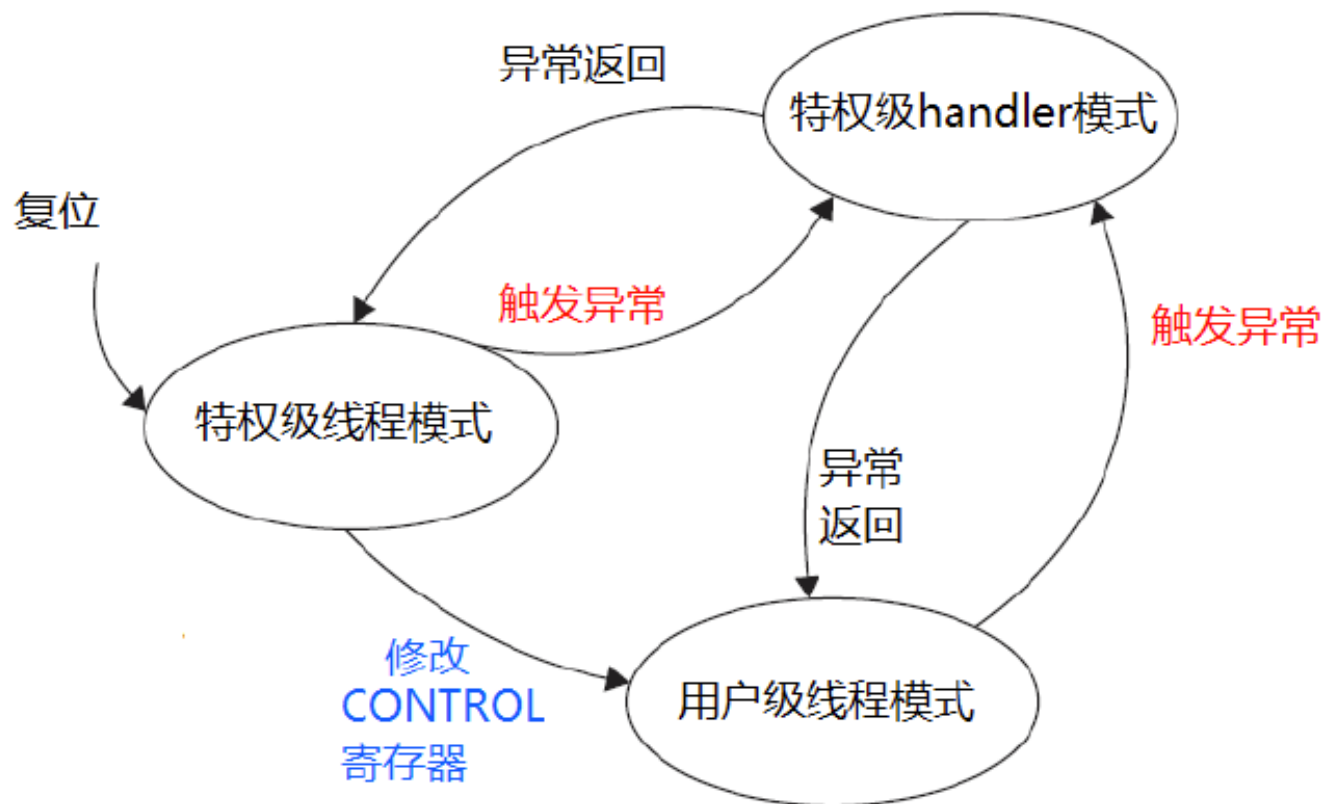


图 2.5 合法的操作模式转换图



模式改变

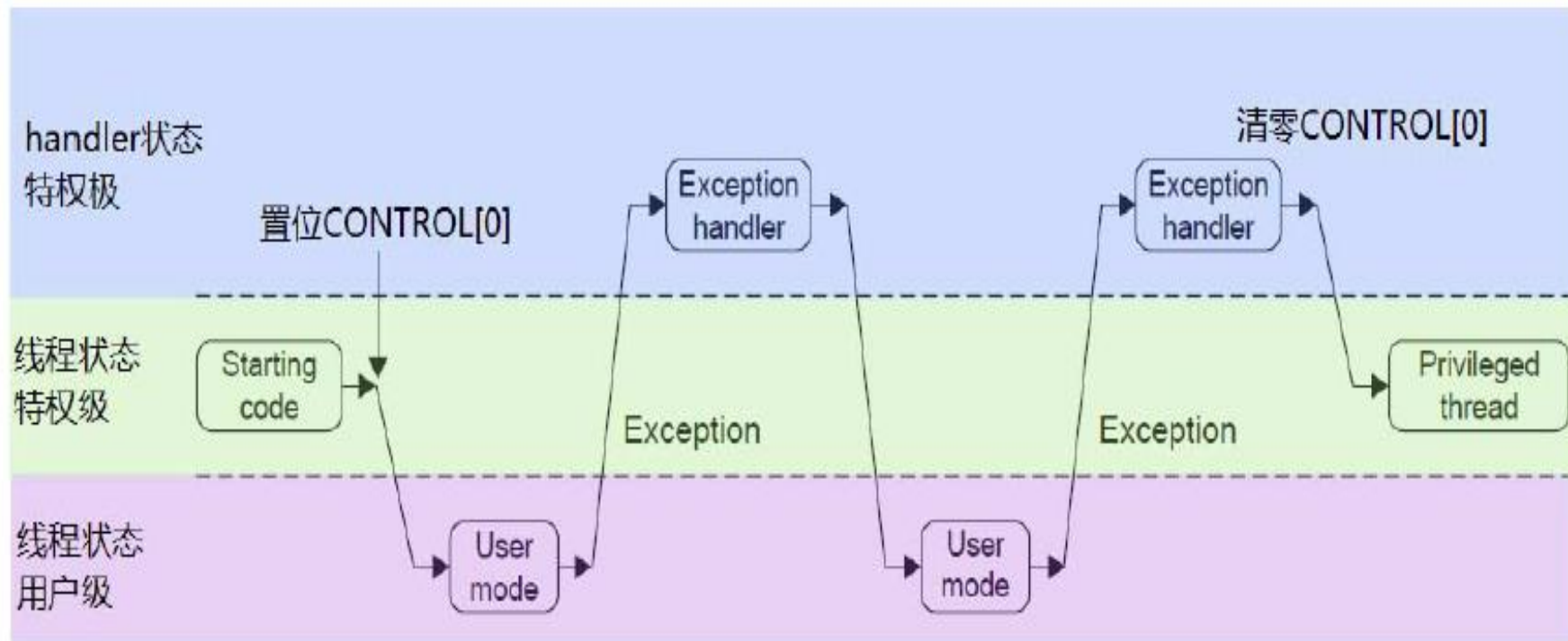
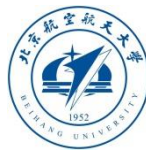
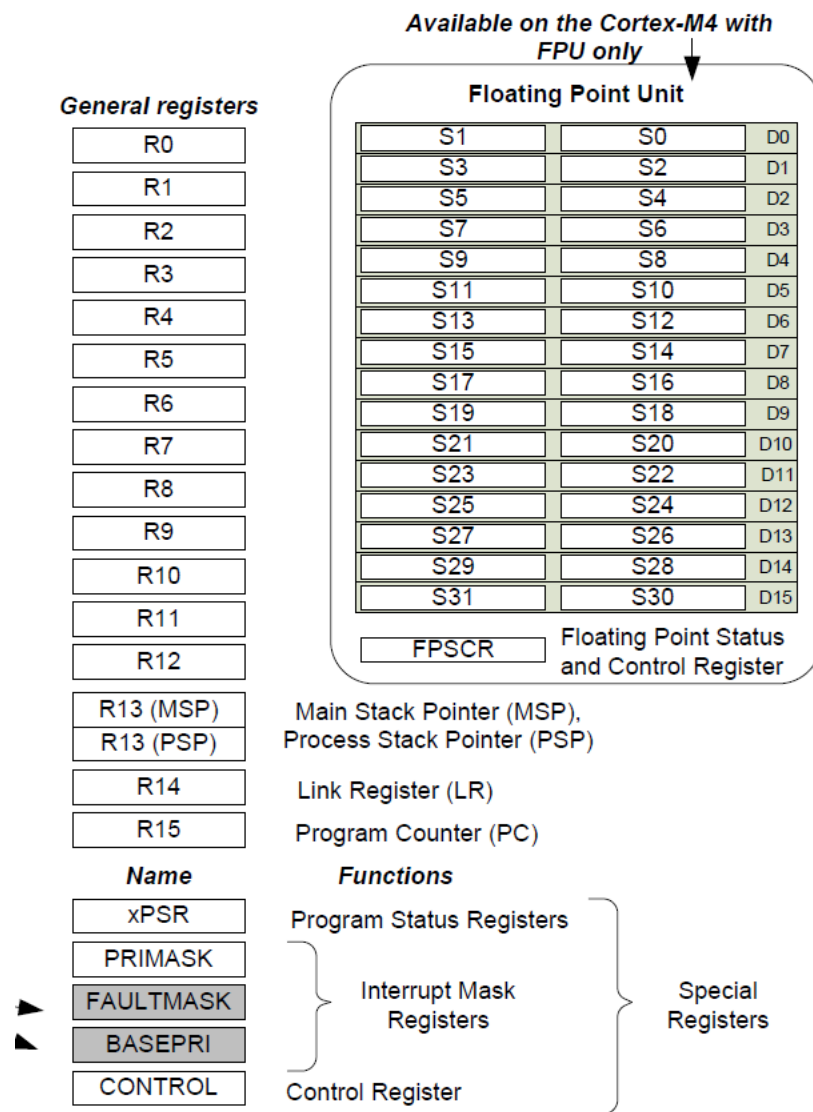


图 3.7 特权级和处理器模式的改变图





2.3.2 Cortex-M微处理器的寄存器组织



2.3.2 Cortex-M微处理器的寄存器组织

1.通用寄存器

R0~R7:通用寄存器组中的低寄存器，32位指令和大多数16位指令可以访问此类低寄存器。复位后的初始值是未定义的。

R8~R12:通用寄存器组中的高寄存器，32位指令和少量16位指令可以访问此类寄存器。复位后的初始值也是未定义的。

R13:堆栈指针寄存器，它实际上有两个物理寄存器，分别对应于主堆栈指针(MSP)和进程堆栈指针(PSP),因此系统可以同时支持两个堆栈。MSP为默认的栈指针，在复位后或处理器处于处理模式时使用。PSP只用于线程模式。两个堆栈指针在同一时间只有一个可见的。

2.3.2 Cortex-M微处理器的寄存器组织

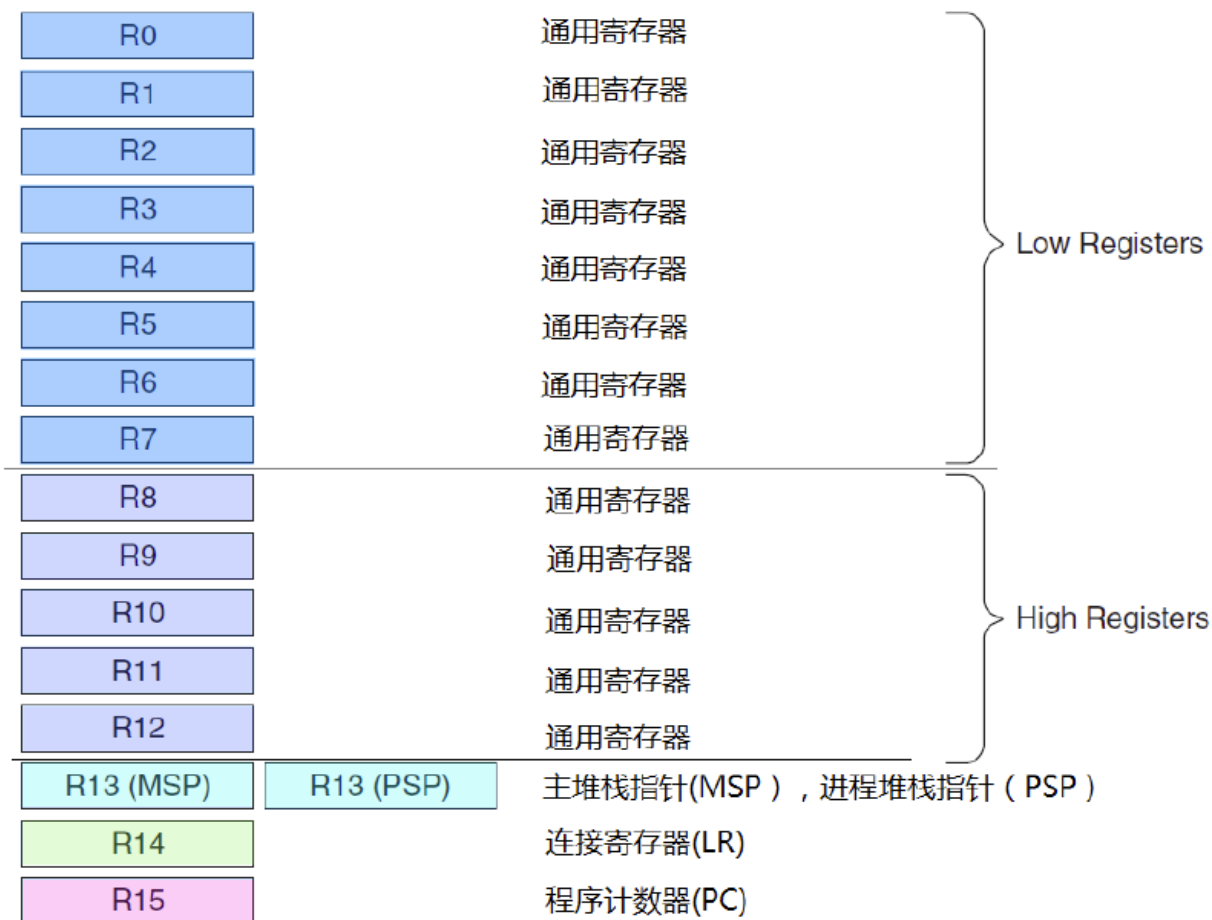
栈指针的选择由特殊寄存器**CONTROL**决定。

一般在操作系统中，**MSP**用于内核以及系统异常中断的代码的堆栈指针，**PSP**用于用户任务的应用堆栈指针。

好处：即使用户程序错误导致**堆栈溢出**，也可将其隔离在内核之外，不至于引起**内核崩溃**。



通用寄存器



主堆栈指针（**MSP**），或写作**SP_main**。是缺省的堆栈指针，它由**OS**内核、异常服务例程以及所有需要特权访问的应用程序代码来使用。可应用于线程模式和处理器模式。

进程堆栈指针（**PSP**）或写作**SP_process**。用于常规的应用程序代码（不处于异常服务例程中时）。只用于线程模式。



2.3.2Cortex-M微处理器的寄存器组织

1.通用寄存器

R14:链接寄存器(LR),当子程序或函数被调用时, LR用来保存返回的地址。在子程序或函数结束时, 程序控制可以通过将LR的数值加载到程序计数器(PC)中来返回调用程序处并继续执行。LR也用于异常返回, 但是在这里保存的是返回后的状态, 不是返回的地址, 异常返回是通过硬件自动出栈弹出之前压入的PC完成的。

R15:程序计数器(PC),指向“正在取指”的指令, 是可读写的。读操作返回的是当前指令地址加4; 写PC会引起程序的跳转操作。



连接寄存器R14

- **R14 是连接寄存器 (LR)。在汇编程序中写成LR 和R14。**
- **LR 用于在调用子程序时存储返回地址。**
- **例如，使用BL(分支并连接，Branch and Link)指令时，就自动填充LR 的值。**

main ;主程序

...

BL function1 ;使用“分支并连接”指令呼叫function1

; PC= function1, 并且LR=main 的下一条指令地址

...

Function1

... ; function1 的代码

BX LR ;函数返回





程序计数器R15

- **R15 是程序计数器， “PC”**
- **指令流水线，读PC时返回的值与当前指令所在的地址存在差异**

The *Program Counter* (PC) is register R15. It contains the current program address. On reset, the processor loads the PC with the value of the reset vector, which is at address 0x00000004. Bit[0] of the value is loaded into the EPSR T-bit at reset and must be 1.



2.3.2 Cortex-M微处理器的寄存器组织

2.特殊功能寄存器

除了通用寄存器组中的寄存器外，处理器中还存在多个特殊功能寄存器。这些特殊功能寄存器表示处理器状态，定义了操作状态和中断/异常屏蔽。特殊功能寄存器分为程序状态寄存器、中断/异常屏蔽寄存器和控制寄存器3类。



状态寄存器

- 应用程序 PSR (APSR)
- 中断号 PSR (IPSR)
- 执行 PSR (EPSR)

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T				ICI/IT					



2.3.2 Cortex-M微处理器的寄存器组织

(1)xPSP程序状态寄存器

程序状态寄存器表示处理器的状态，可以分为3类：应用状态寄存器(APSR)，中断/异常状态寄存器(IPSR)、执行状态寄存器(EPSR),这三个寄存器可组合起来构成一个32位的寄存器，统称为xPSR。可以xPSR或PSR名称统一访问，也可单独访问。

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T				ICI/IT					

2.3.2 Cortex-M微处理器的寄存器组织

32位xPSR程序状态寄存器可分成3部分:

APSR:应用程序状态寄存器, 保存当前指令运算结果的状态。

IPSR:中断状态寄存器, 保存当前中断的向量号。

EPSR:执行状态寄存器。**LDM**、**STM**和**if-then**指令是多周期指令, 如果在执行多周期指令时发生了异常, 那么处理器会暂时停止以上指令的操作, 进入异常, 这时就需要执行状态寄存器来保护现场。**EPSR**包含两个重叠的区域: 可中断-可继续指令(**ICI**)区和**if-then**(**IT**)状态区。

T标志位为Thumb状态, 恒为1, **ICI/IT**保存被异常中断打断的指令流状态或者**IT**指令的状态。

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0
APSR	N	Z	C	V	Q											
IPSR												Exception Number				
EPSR						ICI/IT	T				ICI/IT					



Table 4. APSR bit assignments

Bits	Name	Description
[31]	N	Negative flag.
[30]	Z	Zero flag.
[29]	C	Carry or borrow flag.
[28]	V	Overflow flag.
[27]	Q	DSP overflow and saturation flag
[26:20]	-	Reserved
[19:16]	GE[3:0]	Greater than or Equal flags. See SEL on page 108 for more information.
[15:0]	-	Reserved

Table 5. IPSR bit assignments

Bits	Name	Function
[31:9]	-	Reserved
[8:0]	ISR_NUMBER	<p>This is the number of the current exception:</p> <ul style="list-style-type: none">0 = Thread mode.1 = Reserved.2 = NMI.3 = HardFault.4 = MemManage.5 = BusFault6 = UsageFault7-10 = Reserved11 = SVCcall.12 = Reserved for debug13 = Reserved14 = PendSV.15 = SysTick.16 = IRQ0.--256 = IRQ239. <p>see Exception types on page 39 for more information.</p>





(2)屏蔽寄存器

屏蔽寄存器分为3组，分别是**PRIMASK**、**FAULTMASK**和**BASEPRI**，用于控制异常或中断的使能和禁止。

PRIMASK:该寄存器只有一位，为片上外设中断总开关。当该位为**0**时，响应所有外设中断；当该位为**1**时，将阻止不可屏蔽中断(**NMI**)和**HardFault**异常之外的所有中断/异常。它的默认值是**0**，表示开放中断。

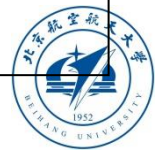




(2)屏蔽寄存器

FAULTMASK:异常屏蔽寄存器，相当于异常的总开关。该寄存器只有最低位有效，即第0位。当该位为0时，响应所有的异常；当该位为1时，除NMI外的所有异常被屏蔽。它的默认值也是0，表示开放中断。

BASEPRI:定义屏蔽优先级的阈值，该寄存器最多有9位。当它被设为某个值后，优先级号大于或等于阈值的中断都被屏蔽。例如，将BASEPRI设置为3，则3和3以上优先级的中断都被屏蔽，而0、1和2优先级的中断不会被屏蔽。其默认值是0，即不屏蔽任何中断。





PRIMASK, FAULTMASK 和 BASEPRI

名字	功能描述
PRIMASK	只有1位。为1时，关掉所有可屏蔽的异常，只剩NMI和硬fault可以响应。缺省值是0，表示没有关中断。
FAULTMASK	只有1位。置1时，只有NMI才能响应，所有其它的异常，包括中断和fault，通通关闭。它的缺省值也是0，表示没有关异常。
BASEPRI	最多有9位（由表达优先级的位数决定）。定义了被屏蔽优先级的阈值。当它被设成某个值后，所有优先级号大于等于此值的中断都被关（优先级号越大，优先级越低）。但若被设成0，则不关闭任何中断，0也是缺省值。





(3)控制寄存器

控制寄存器**CONTROL**有两个作用，一是定义线程模式的访问级别（特权或非特权）。二是选择堆栈指针（主栈指针或进程栈指针）。

位	功 能
CONTROL[1]	堆栈指针选择 0=选择主堆栈指针 MSP(复位后默认值) 1=选择进程堆栈指针 PSP 在线程或基础级(没有在响应异常),可以使用 PSP。在 handler 模式下,只允许使用 MSP,所以此时不得往该位写 1
CONTROL[0]	0=特权级的线程模式 1=用户级的线程模式 handler 模式永远都是特权级的



2.3.3 Cortex-M微处理器的存储组织

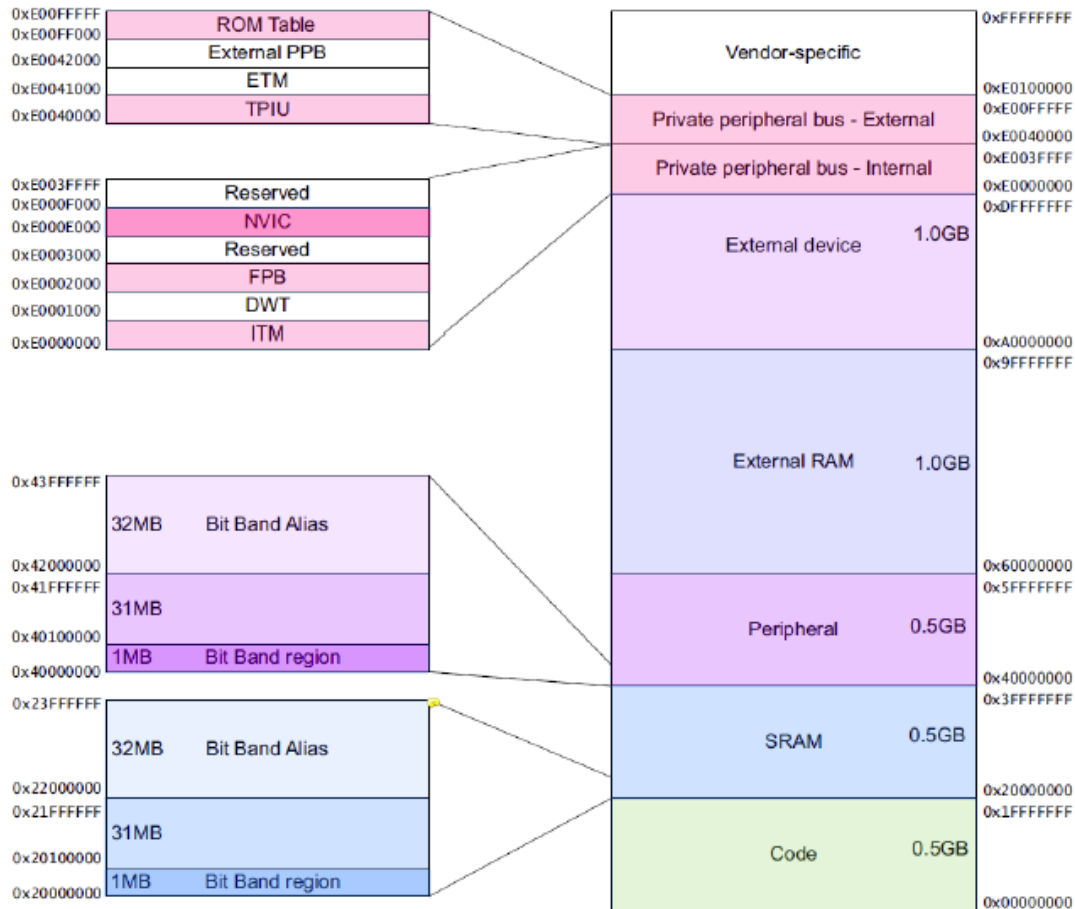
存储器映射

2.存储器映射Cortex-M微处理器支持多种存储器特性，以32位寻址，可寻址存储空间为4GB,无需将存储器分页。Cortex-M微处理器的4GB地址空间被划分为多个存储器区域。区域根据各自典型用法进行划分，这些区域主要用于：

- (1)程序代码访问（如code区域）
- (2)数据访问（如SRAM区域）
- (3)外设（如外设区域）
- (4)处理器的内部控制和调试部件（如私有外设总线）。



地址空间



2.3.3 Cortex-M微处理器的存储组织

这种区域的划分使存储系统的使用具有很大的灵活性。例如，程序既可以在code区域执行，也可以在SRAM区域执行。

代码区域(code区域)：用于存放指令或数据，通过ICode总线取指令，通过DCode总线读写数据。主要用于程序代码，也用于异常向量表。

片上SRAM区域：可以存放指令或数据，通过系统总线(system bus)取指令或读写数据。主要用于数据存储器。SRAM区域为可位寻址的区域，通过位段别名地址修改的位数值会被自动转换为位段区域的读-修改-写的操作。

片上外设(peripheral)区域：通过系统总线进行访问，主要用于片上外设的输入输出接口。

2.3.3 Cortex-M微处理器的存储组织

片外RAM：通过系统总线进行访问，主要用于扩展外部存储器。

片外外设区域：通过系统总线进行访问，主要用于扩展片外外设的输入输出接口。

私有外设总线(PPB)：分配给包括内置的中断控制器(NVIC)和调试部件在内的私有外设等使用。

系统区域(system区域)：分配给芯片厂商使用。

2.3.3 Cortex-M微处理器的存储组织

3.栈内存操作

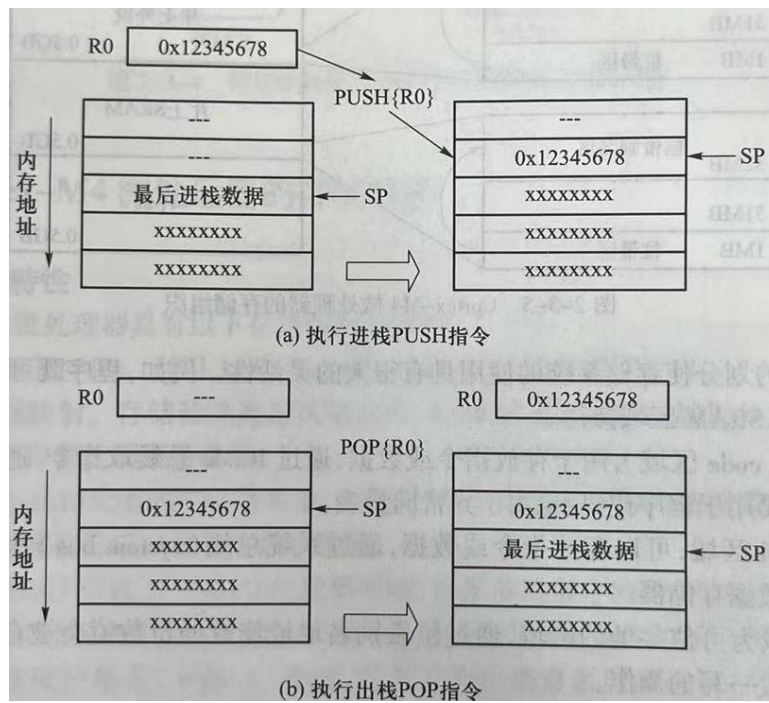
堆栈是按照后进先出原则组织的一个存储区域。堆栈的作用：

- (1)临时存储寄存器或寄存器组中的数据。
- (2)用于向函数或子程序中传递参数。
- (3)用于存储局部变量。
- (4)当中断等异常产生时用于保存处理器状态和寄存器数值。

2.3.3 Cortex-M微处理器的存储组织

3. 栈内存操作

Cortex-M微处理器使用的是“满递减”栈。执行PUSH指令时，处理器首先减小堆栈指针的值，然后将数据存储存储在堆栈指针所指的存储器位置。当执行POP指令时，堆栈指针指向的存储器位置的数据被读出，然后堆栈指针的值自动增加。



2.3.3 Cortex-M微处理器的存储组织

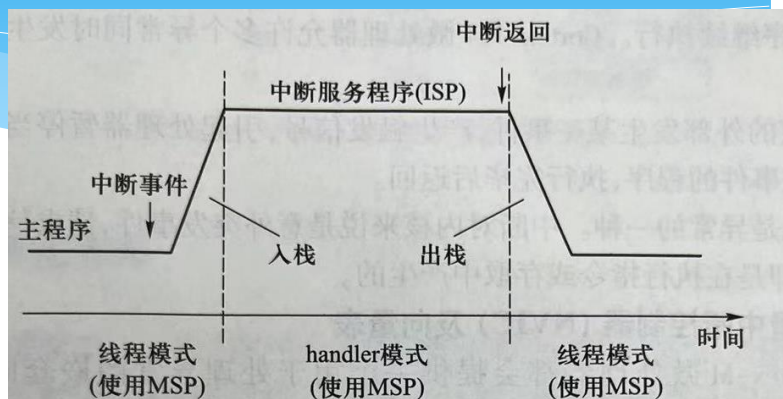
Cortex-M微处理器有主堆栈指针(MSP)和进程堆栈指针(PSP)两个堆栈指针,并且由CONTROL寄存器的第2位SPSEL的值控制。若该位为0,则线程模式在栈操作时使用MSP,否则线程模式使用PSP。

从处理模式到线程模式的异常返回期间,栈指针的选择可由异常返回(EXC_RETURN)的数值决定,处理器硬件会相应地自动更新SPSEL的数值。

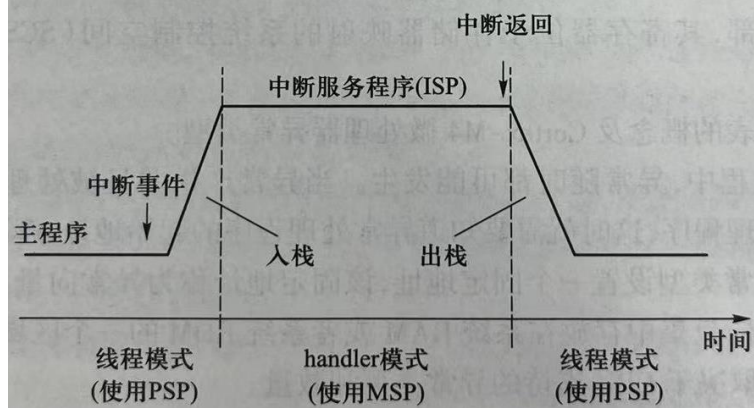
2.3.3 Cortex-M微处理器的存储组织

无操作系统OS：线程模式和处理模式可只使用MSP。在异常事件产生后，处理器在进入中断服务程序(ISR)前首先将多个寄存器中的数值压入堆栈中。在ISR结束时，这些数值又会被恢复到寄存器组中。

有操作系统OS：那么通常会将应用任务和内核所用的堆栈空间分离开。这时要用到PSP指针，而且在进入异常和异常退出时会发生堆栈指针SP的切换。在进行自动“压栈”和“出栈”时使用PSP。分离的堆栈结构简化了操作系统的设计。



(a) CONTROL[1]=0: 使用MSP



(b) CONTROL[1]=1: 线程模式使用PSP, handler模式使用MSP

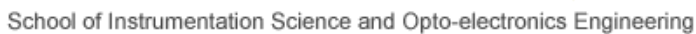
2.3.4 Cortex-M微处理器的异常和中断

1.异常和中断

异常：是处理器内核产生复位、存取失败、遇到未定义指令时，正常执行的程序被暂停，转到相应的处理程序执行称为异常。每个异常对应一个异常处理程序。当异常程序执行完毕后，返回当前程序继续执行。Cortex-M微处理器允许多个异常同时发生，并按固定的优先级进行处理。

中断：是内核的外部发生某一事件，产生触发信号，引起处理器暂停当前执行的程序，转去执行处理相应事件的程序，执行完毕后返回。中断可看成是异常的一种。

中断对内核来说是意外突发事件，请求信号来自外部；异常是内核产生的，即是在执行指令或存取中产生的。



The diagram illustrates the NVIC (Nested Vectored Interrupt Controller) and its connections:

- System Tick Timer:** A gray box at the top labeled "SysTick (System Tick Timer)" with an arrow pointing down to the NVIC.
- Cortex-M processor Core:** A large light purple box on the right labeled "Cortex-M processor Core". It has a bidirectional arrow connecting to the NVIC and a "Bus interface" connection to the internal bus interconnect.
- Internal bus interconnect:** A gray box at the bottom labeled "Internal bus interconnect". It has a bidirectional arrow connecting to the NVIC and a downward arrow pointing away from the diagram.
- Peripherals:** On the left, a single box labeled "Peripheral" and a stack of boxes labeled "Peripherals" have arrows pointing to the NVIC. The single peripheral connects via "NMI" (Non-Maskable Interrupt), and the stack of peripherals connects via "IRQs" (Interrupt Requests).
- NVIC:** A central orange box labeled "NVIC" containing a blue box labeled "Configuration registers".

- Cortex-M支持大量异常，包括16-4-1=11个系统异常，和最多240个外部中断——简称IRQ。
- 由外设产生的中断信号，除了SysTick的之外，全都连接到NVIC的中断输入信号线。



2.3.4 Cortex-M微处理器的异常和中断

2.嵌套向量中断控制器(NVIC)及向量表

所有的Cortex-M微处理器都会提供一个用于处理异常的嵌套向量中断控制器，也就是NVIC。中断也属于异常的一种，另外还有错误异常和其他用于OS支持的系统异常。

在Cortex-M微处理器中，NVIC负责处理异常和中断配置、优先级以及中断屏蔽。NVIC在处理器内部，其寄存器位于存储器映射的系统控制空间(SCS)处，并且是可编程的。

2.3.4 Cortex-M微处理器的异常和中断

向量表的概念及Cortex-M微处理器异常类型：

在系统运行过程中，异常随时都可能发生。当异常产生并且被处理器内核接受后，处理器将执行异常处理程序，这时就需要知道异常处理程序的起始地址，解决该方法是在内存中为每个异常类型设置一个固定地址，该固定地址称为异常向量。将处理器所支持的所有异常的异常向量集中存放在系统RAM或者ROM的一个区域，就构成异常向量表。

2.3.4 Cortex-M微处理器的异常和中断

Cortex-M微处理器向量表中的每个元素代表一个异常类型处理程序的起始地址，每个起始地址占一个字，即4个字节。

异常类型	CMSIS 中断编号	地址偏移	向量
18~255	2~239	0x48~0x3FF	IRQ#2-#239 [1]
17	1	0x44	IRQ#1 [1]
16	0	0x40	IRQ#0 [1]
15	-1	0x3C	SysTick [1]
14	-2	0x38	PendSV [1]
NA	NA	0x34	保留
12	-4	0x30	调试监控 [1]
11	-5	0x2C	SVC [1]
NA	NA	0x28	保留
NA	NA	0x24	保留
NA	NA	0x20	保留
NA	NA	0x1C	保留
6	-10	0x18	使用错误 [1]
5	-11	0x14	总线错误 [1]
4	-12	0x10	MemManage错误[1]
3	-13	0x0C	HardFault [1]
2	-14	0x08	NMI [1]
1	NA	0x04	复位 [1]
NA	NA	0x00	MSP初始值

图 2-3-8 Cortex-M4 微处理器异常类型

2.3.5 复位和复位流程

复位的种类有：上电复位、掉电复位、复位引脚复位、看门狗复位、软件复位等。对于嵌入式系统来讲，有内核复位和系统复位，他们属于软件复位。

内核复位是指只复位处理器，而不复位如GPIO、TIM、USART、SPI等外设寄存器。系统复位既复位处理器，又复位外设寄存器。

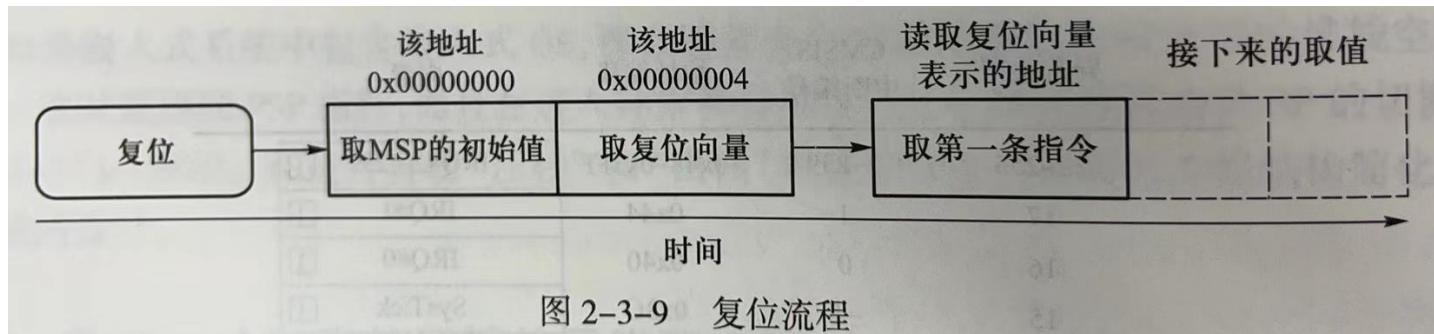
2.3.5 复位和复位流程

Cortex-M复位类型有三种：

- (1)上电复位。复位微控制器中的所有部分，包括处理器、调试支持部件和外设等。
- (2)系统复位。只会复位处理器和外设，不包括处理器的调试支持部件。
- (3)处理器复位。只复位处理器。

2.3.5 复位和复位流程

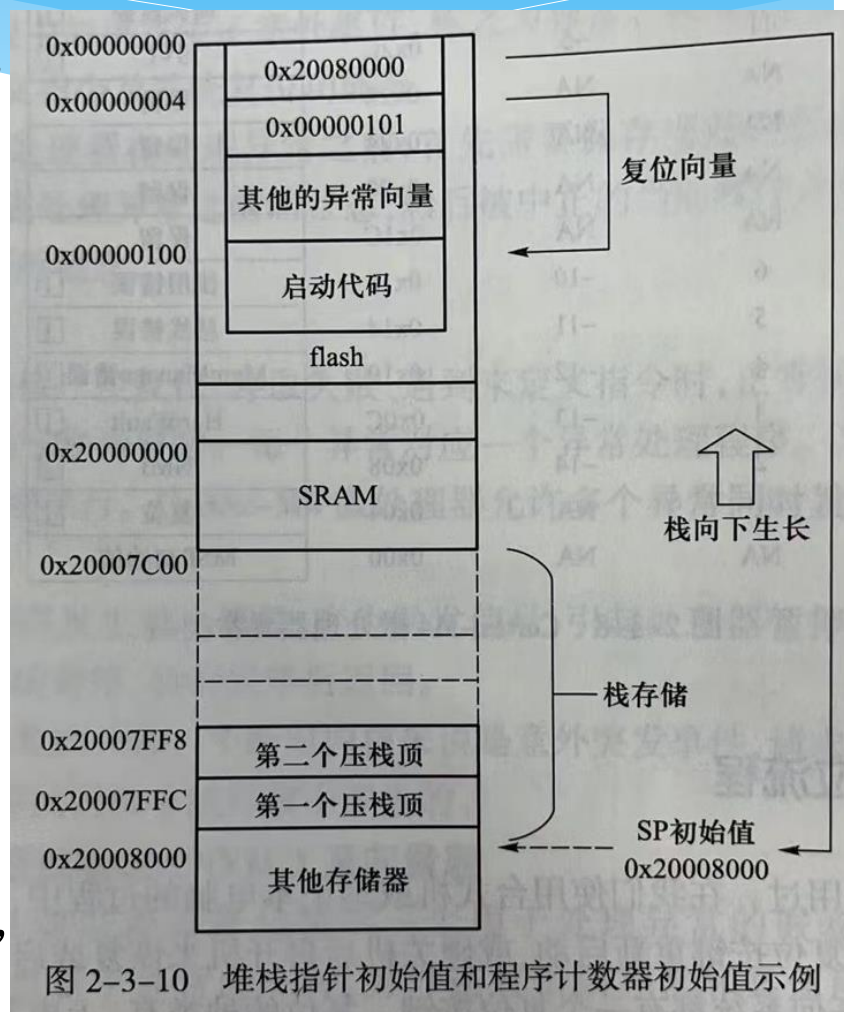
为在复位后能够使系统正常运行，在复位后以及处理器开始执行程序前，Cortex-M微处理器会从存储器中读出头两个字。第一个字表示主栈指针的初始值，第二个字代表复位处理起始地址的复位向量。处理器读出这两个字后，就会将这些数值赋给MSP和程序计数器PC,然后以此来取第一条指令。



2.3.5 复位和复位流程

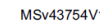
Cortex-M中的栈操作是基于满递减的，所以堆栈指针SP的初始值应该设置在栈顶的位置。例如，若存储器区域为0x20007C00-0x20007FFF(1KB)，则初始的栈指针就应该为0x20008000。

为表示处理器处于Tumb状态，向量表中向量地址的最低位应该为1。复位向量为0x101，而启动代码是从0x100开始的。在取出复位向量后，Cortex-M微处理器就可以从复位向量地址处执行程序，并开始正常操作。



2.4 STM32F745 芯片

- * 2.4.1概述
- * 2.4.2总线架构和存储器



2.4.2 总线架构和存储器

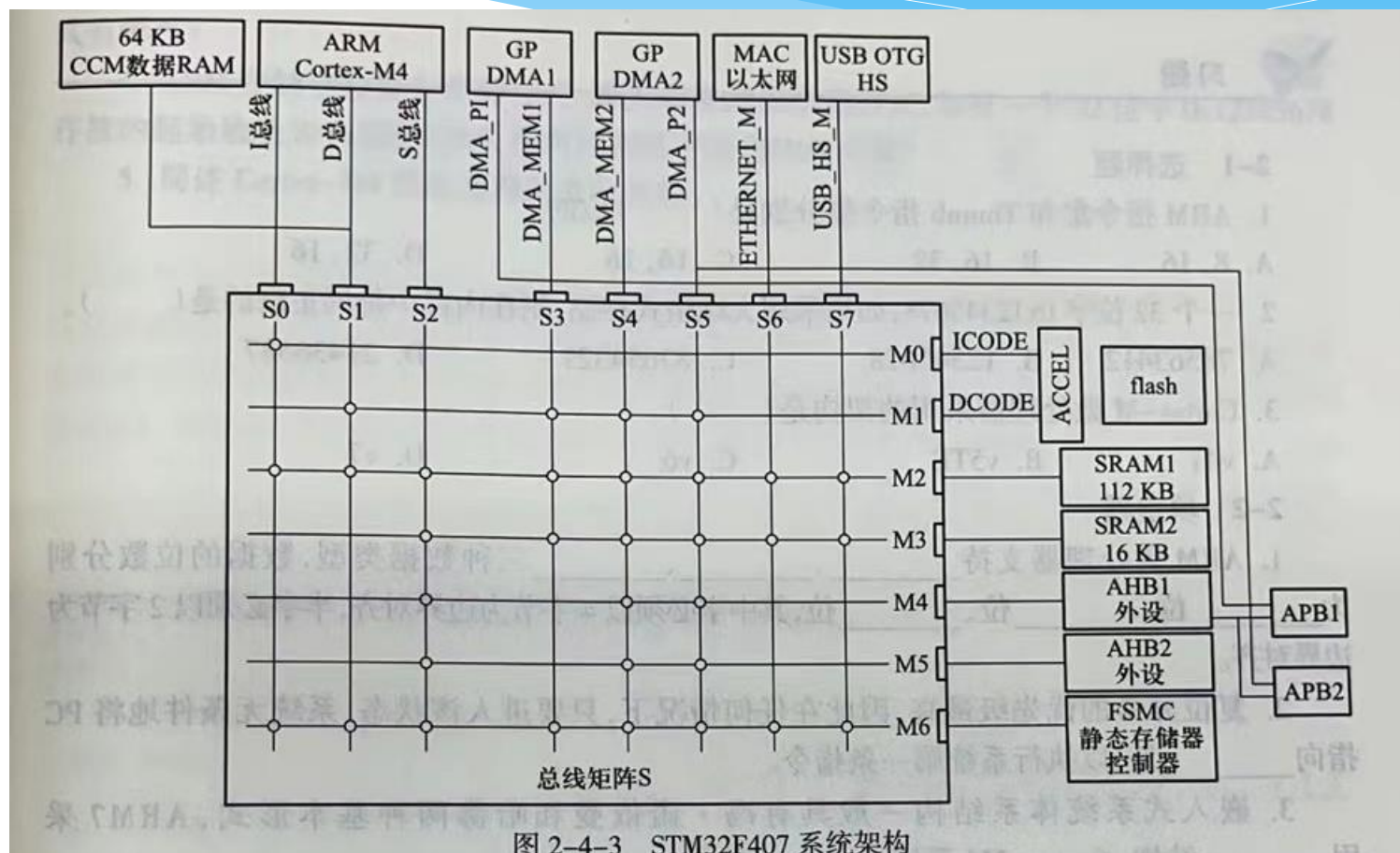


图 2-4-3 STM32F407 系统架构