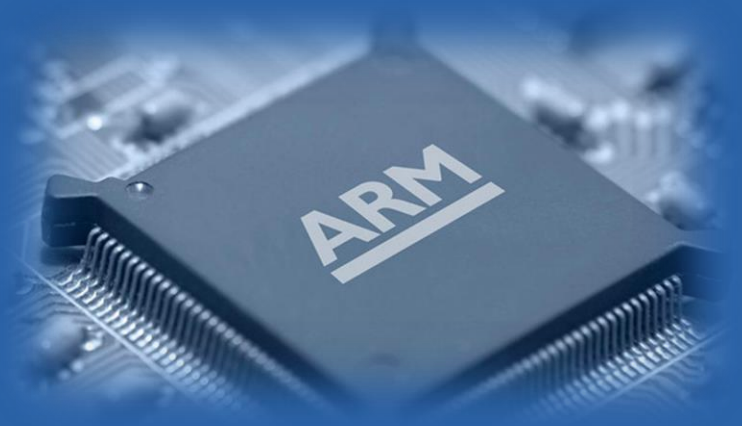
仪器科学与光电工程学院
School of Instrumentation Science and Opto-electronics Engineering

微机原理与接口技术

第五讲 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]
```

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。

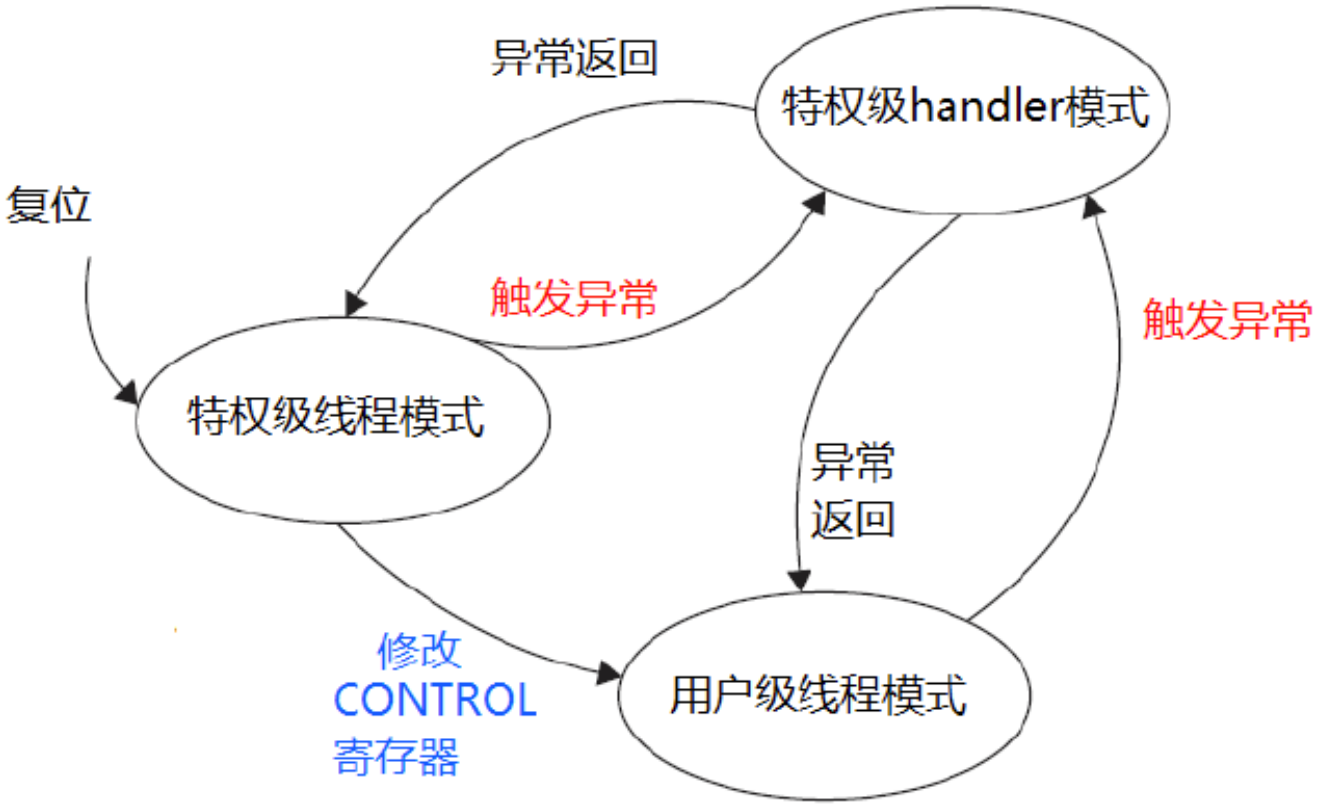
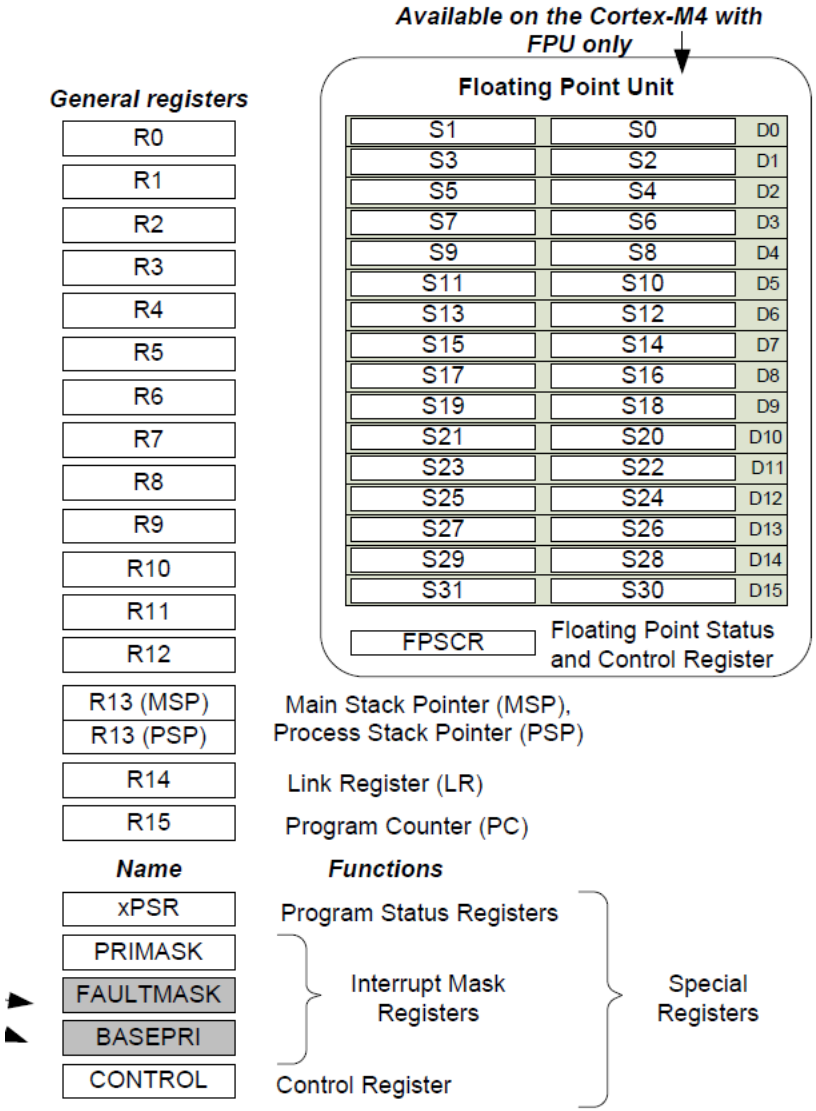


图 2.5　合法的操作模式转换图

2.3.2Cortex-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是可见的。

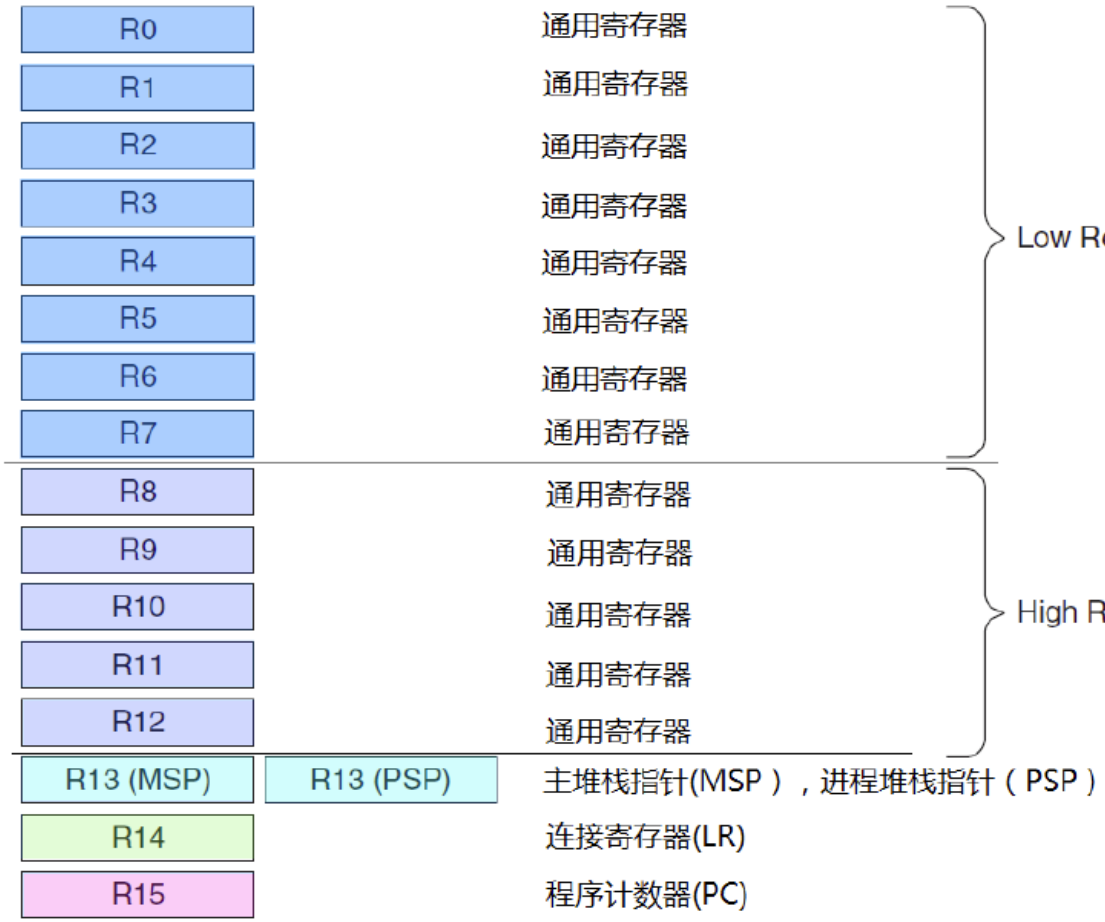
2.3.2 Cortex-M微处理器的寄存器组织

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（不处于异常服用例程中时）。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。

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

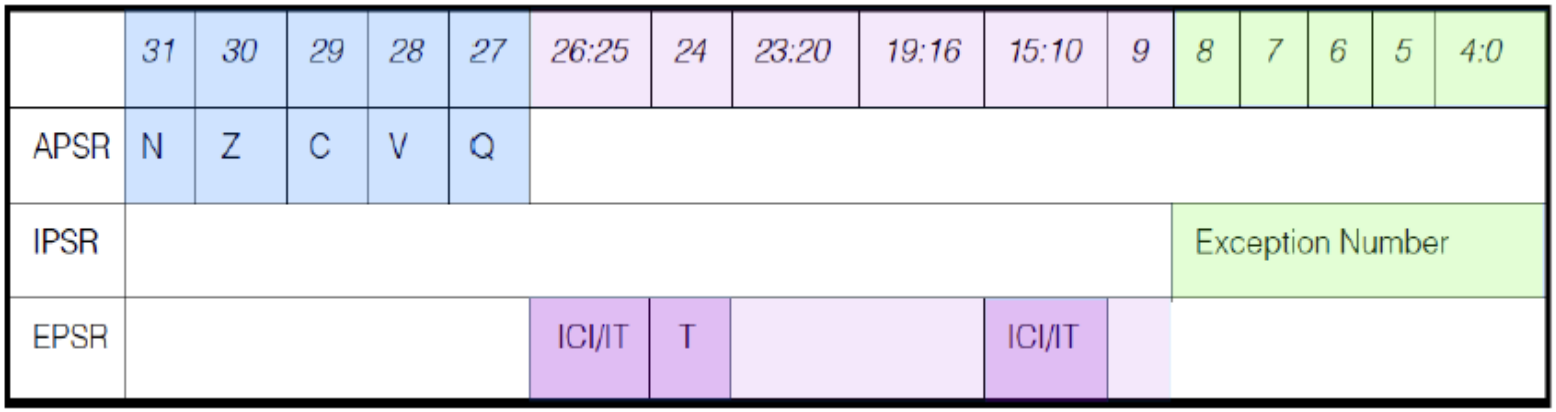
- 应用程序 PSR（APSR）
- 中断号 PSR（IPSR）
- 执行 PSR（EPSR）

	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。



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。

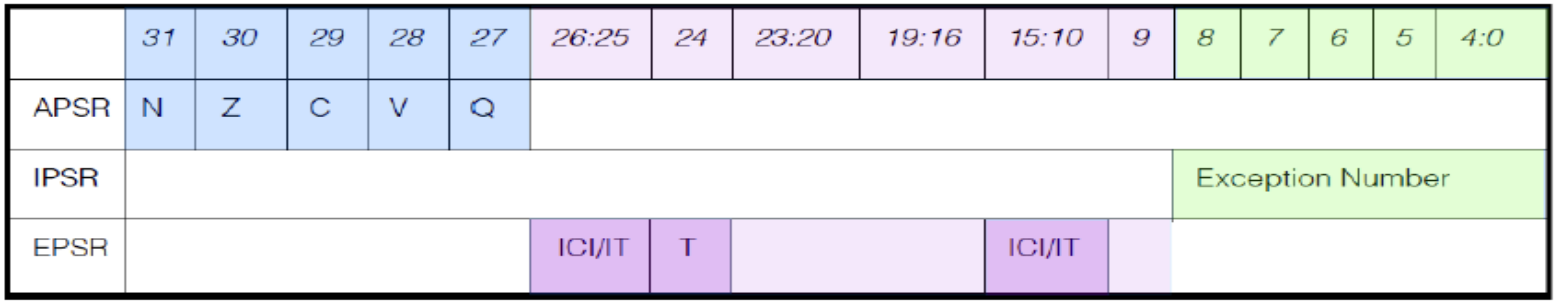


Table 4. APSR bit assignments

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	This is the number of the current exception: 0 = Thread mode. 1 = Reserved. 2 = NMI. 3 = HardFault. 4 = MemManage. 5 = BusFault 6 = UsageFault 7-10 = Reserved 11 = SVCall. 12 = Reserved for debug 13 = Reserved 14 = PendSV. 15 = SysTick. 16 = IRQ0. . . 256 = IRQ239. see *Exception types on page 39* for more information.

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8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。Cotx-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）

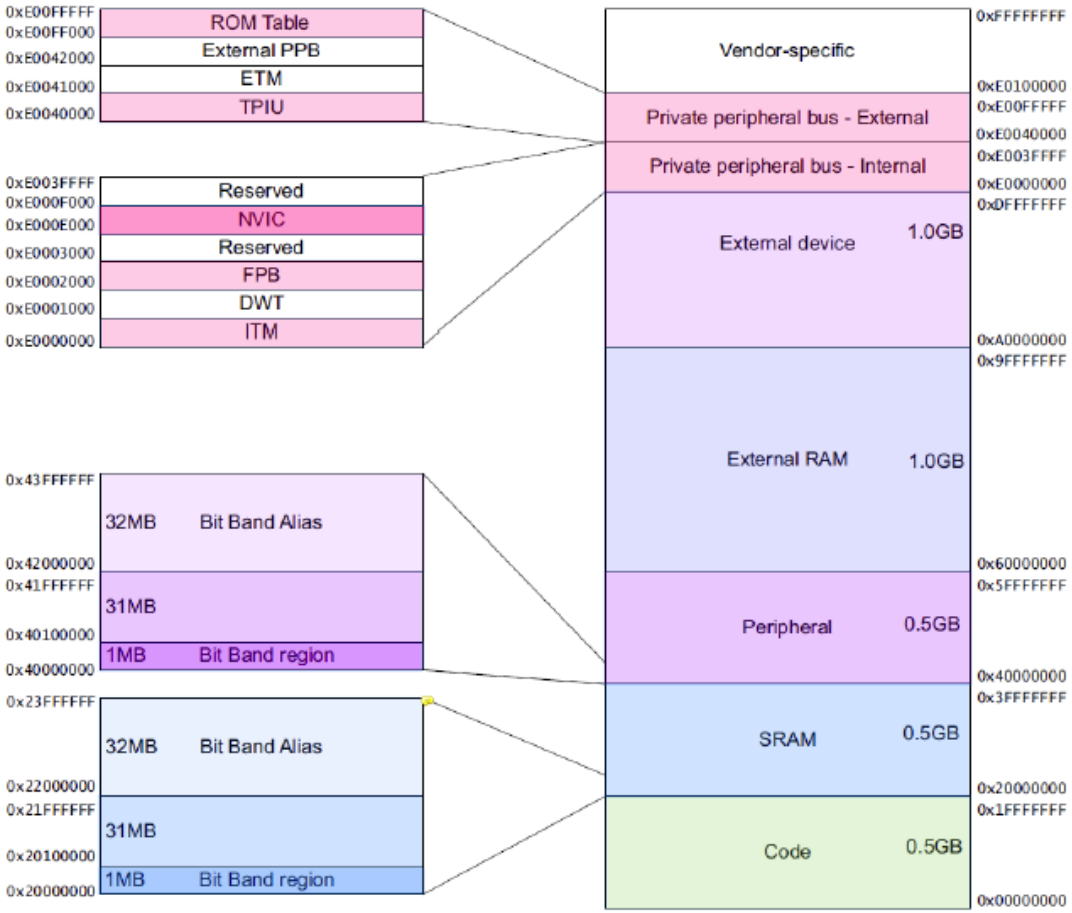
(2)数据访问（如SRAM区域）

(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


0xE00FFFFF
0xE00FF000
ROM Table
0xE0042000
External PPB
0xE0041000
ETM
0xE0040000
TPIU
0xE003FFFF
0xE000F000
Reserved
0xE000E000
NVIC
0xE0003000
Reserved
0xE0002000
FPB
0xE0001000
DWT
0xE0000000
ITM
0x43FFFFFF
32MB
Bit Band Alias
0x42000000
0x41FFFFFF
31MB
0x40100000
1MB
Bit Band region
0x40000000
0x23FFFFFF
32MB
Bit Band Alias
0x22000000
0x21FFFFFF
31MB
0x20100000
1MB
Bit Band region
0x20000000
Vendor-specific
0xFFFFFFFF
0xE0100000
Private peripheral bus - External
0xE00FFFFF
0xE0040000
Private peripheral bus - Internal
0xE003FFFF
0xE0000000
External device
1.0GB
0xDFFFFFFF
0xA0000000
External RAM
1.0GB
0x9FFFFFFF
0x60000000
Peripheral
0.5GB
0x5FFFFFFF
0x40000000
SRAM
0.5GB
0x3FFFFFFF
0x20000000
Code
0.5GB
0x1FFFFFFF
0x00000000

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。

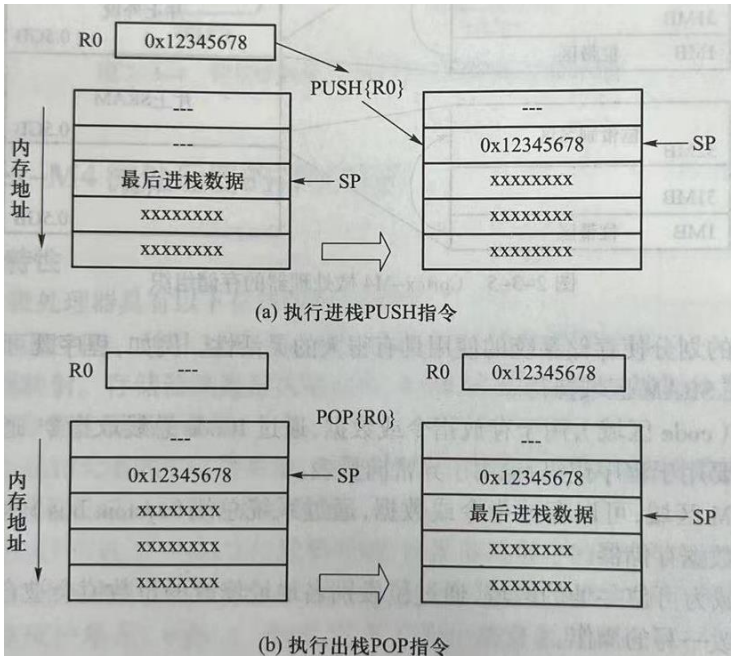
(3)用于存储局部变量。

(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

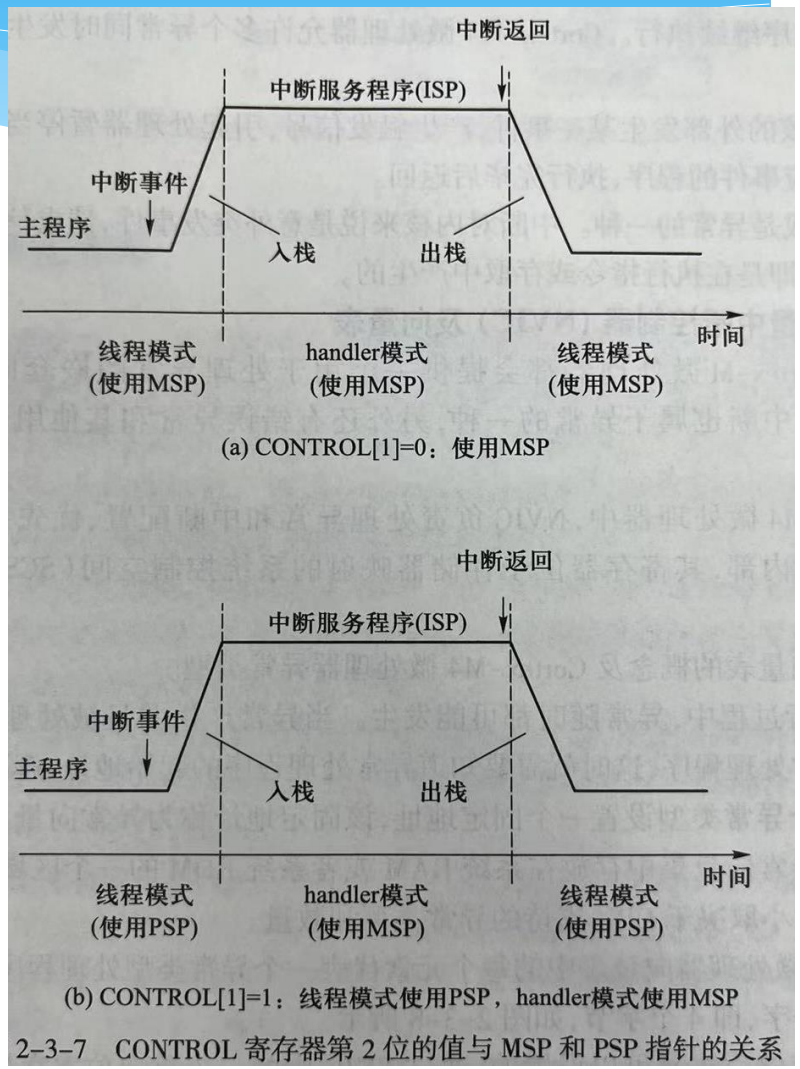
Cortex-M微处理器有主堆栈指针(MSP)和进程堆栈指针(PSP)两个堆栈指针,并且由CONTROL寄存器的第2位SPSEL的值控制。若该位为0,则线程模式在栈操作时使用MSP,否则线程模式使用PSP。

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。



2-3-7　CONTROL 寄存器第 2 位的值与 MSP 和 PSP 指针的关系

2.3.4 Cortex-M微处理器的异常和中断

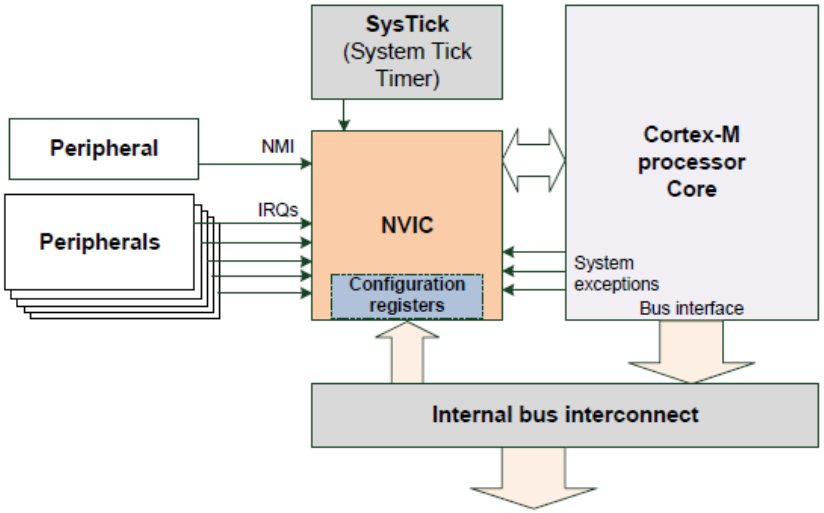
1.异常和中断

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。

中断控制



- 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，解决该问题的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。

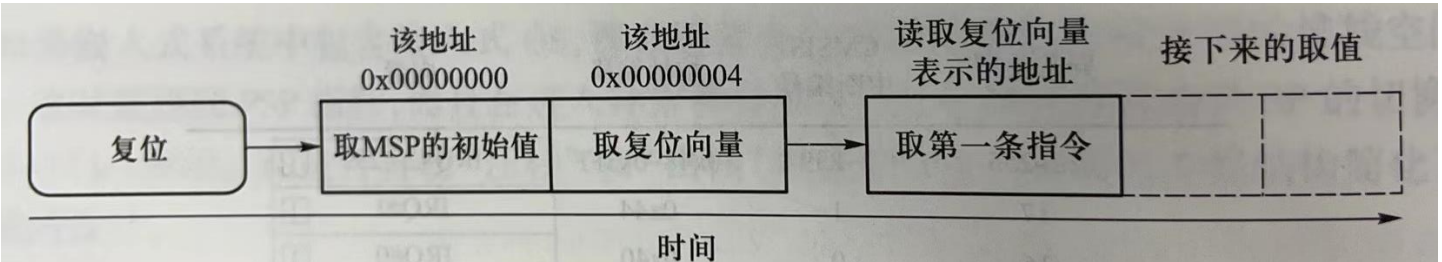


图 2-3-9　复位流程

2.3.5 复位和复位流程

Cortex-M中的栈操作是满递减的，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Thumb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。

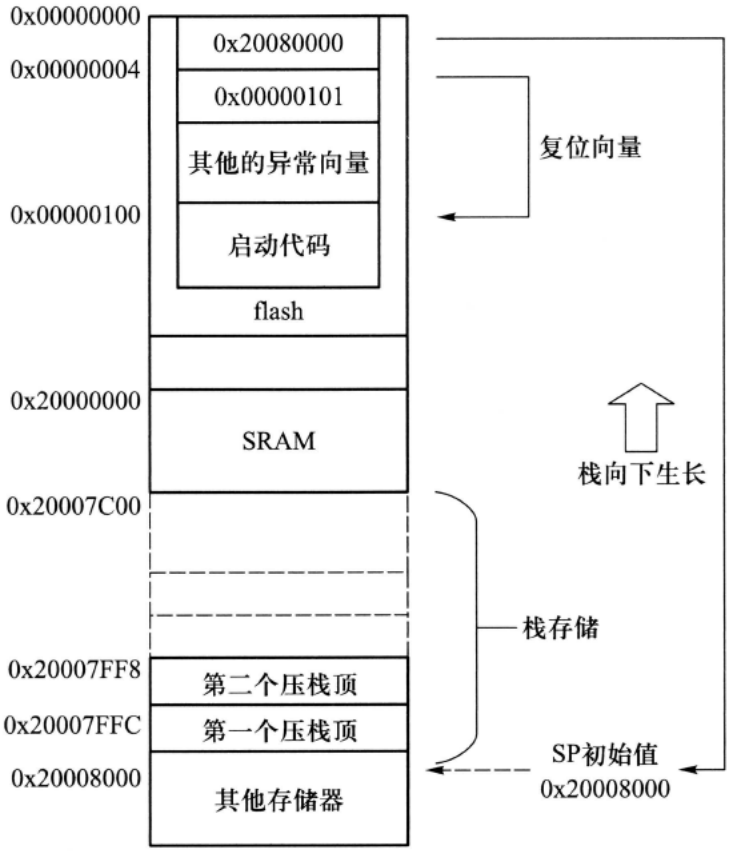
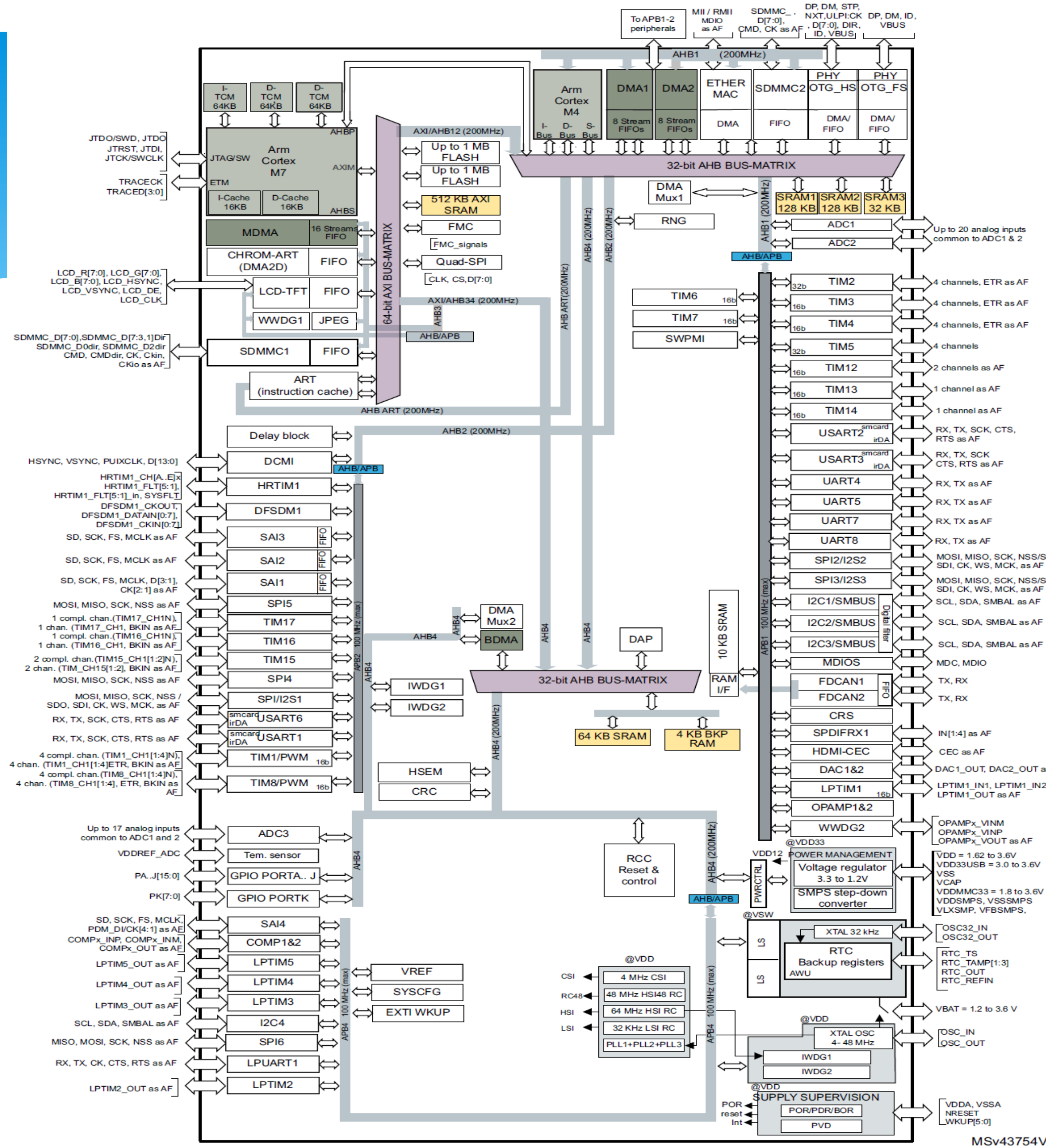


图 2-3-10　堆栈指针初始值和程序计数器初始值示例

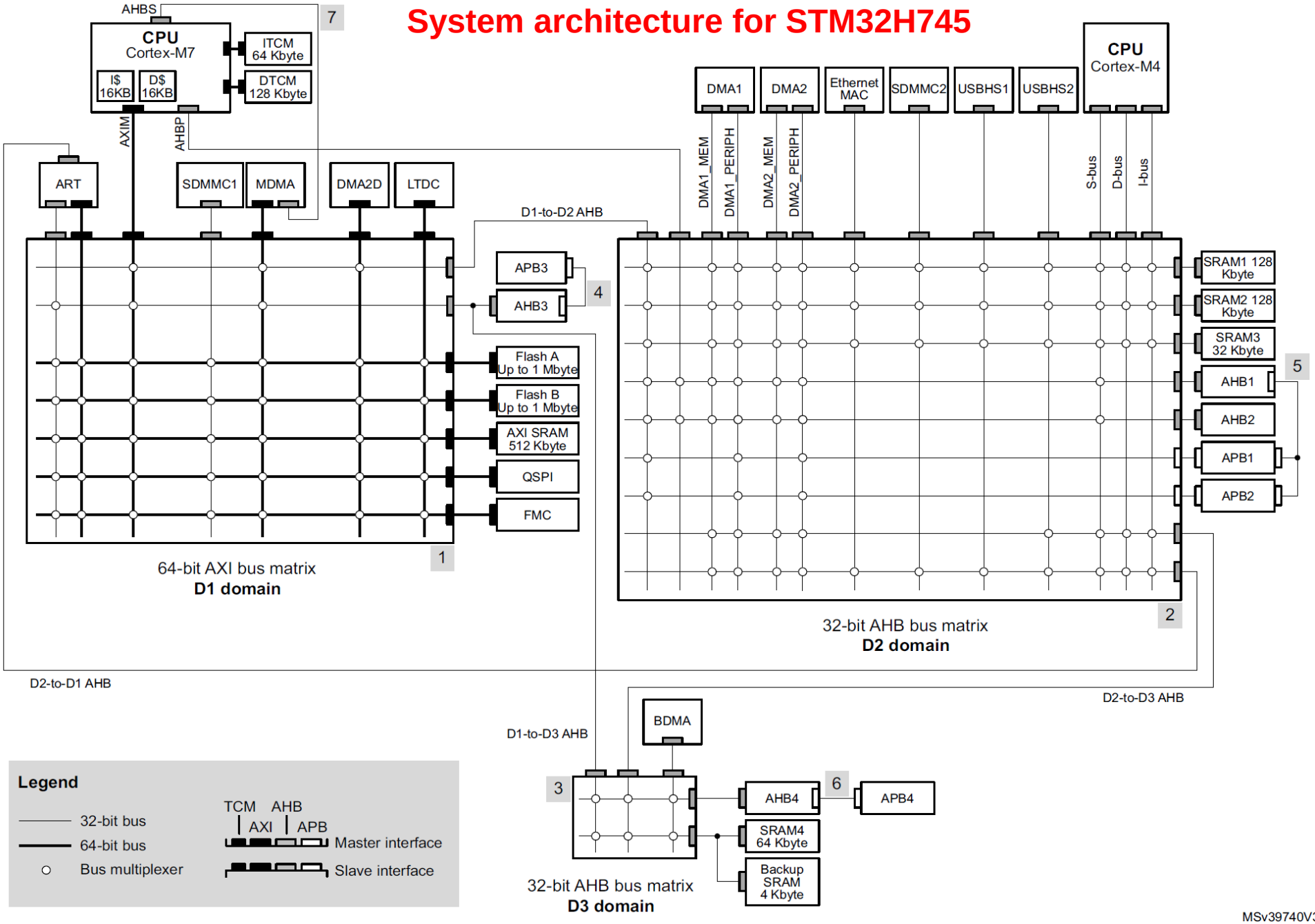
2.4 STM32F745芯片

* 2.4.1概述
* 2.4.2总线架构和存储器



Arm Cortex M7
Arm Cortex M4
64-bit AXI BUS-MATRIX
32-bit AHB BUS-MATRIX
MDMA
CHROM-ART (DMA2D)
LCD-TFT
WWDG1
JPEG
SDMMC1
ART (instruction cache)
Up to 1 MB FLASH
512 KB AXI SRAM
FMC
Quad-SPI
DMA1
DMA2
ETHER MAC
SDMMC2
OTG_HS
OTG_FS
SRAM1 128 KB
SRAM2 128 KB
SRAM3 32 KB
DMA Mux1
RNG
ADC1
ADC2
TIM2
TIM3
TIM4
TIM5
TIM12
TIM13
TIM14
TIM6
TIM7
SWPMI
USART2
USART3
UART4
UART5
UART7
UART8
SPI2/I2S2
SPI3/I2S3
I2C1/SMBUS
I2C2/SMBUS
I2C3/SMBUS
MDIOS
FDCAN1
FDCAN2
CRS
SPDIFRX1
HDMI-CEC
DAC1&2
LPTIM1
OPAMP1&2
WWDG2
Delay block
DCMI
HRTIM1
DFSDM1
SAI3
SAI2
SAI1
SPI5
TIM17
TIM16
TIM15
SPI4
SPI/I2S1
USART6
USART1
TIM1/PWM
TIM8/PWM
DMA Mux2
BDMA
DAP
10 KB SRAM
64 KB SRAM
4 KB BKP RAM
IWDG1
IWDG2
HSEM
CRC
ADC3
Tem. sensor
GPIO PORTA.. J
GPIO PORTK
RCC Reset & control
POWER MANAGEMENT
Voltage regulator 3.3 to 1.2V
SMPS step-down converter
SAI4
COMP1&2
LPTIM5
LPTIM4
LPTIM3
I2C4
SPI6
LPUART1
LPTIM2
VREF
SYSCFG
EXTI WKUP
4 MHz CSI
48 MHz HSI48 RC
64 MHz HSI RC
32 KHz LSI RC
PLL1+PLL2+PLL3
XTAL 32 kHz
RTC Backup registers
XTAL OSC 4- 48 MHz
SUPPLY SUPERVISION
POR/PDR/BOR
PVD
AHB1 (200MHz)
AXI/AHB12 (200MHz)
AXI/AHB34 (200MHz)
AHB ART (200MHz)
AHB2 (200MHz)
VDD = 1.62 to 3.6V
VBAT = 1.2 to 3.6 V
MSv43754V

System architecture for STM32H745



MSv39740V3

微机原理与接口技术

第3章 ARM的指令系统（1）

内容介绍

1. ARM的指令系统简介
2. ARM指令的寻址方式
3. ARM的核心指令

1. 简介

ARM处理器是基于精简指令集计算机(RISC)原理设计的，指令集和相关译码机制较为简单。

不同的ARM体系架构，指令集也不同

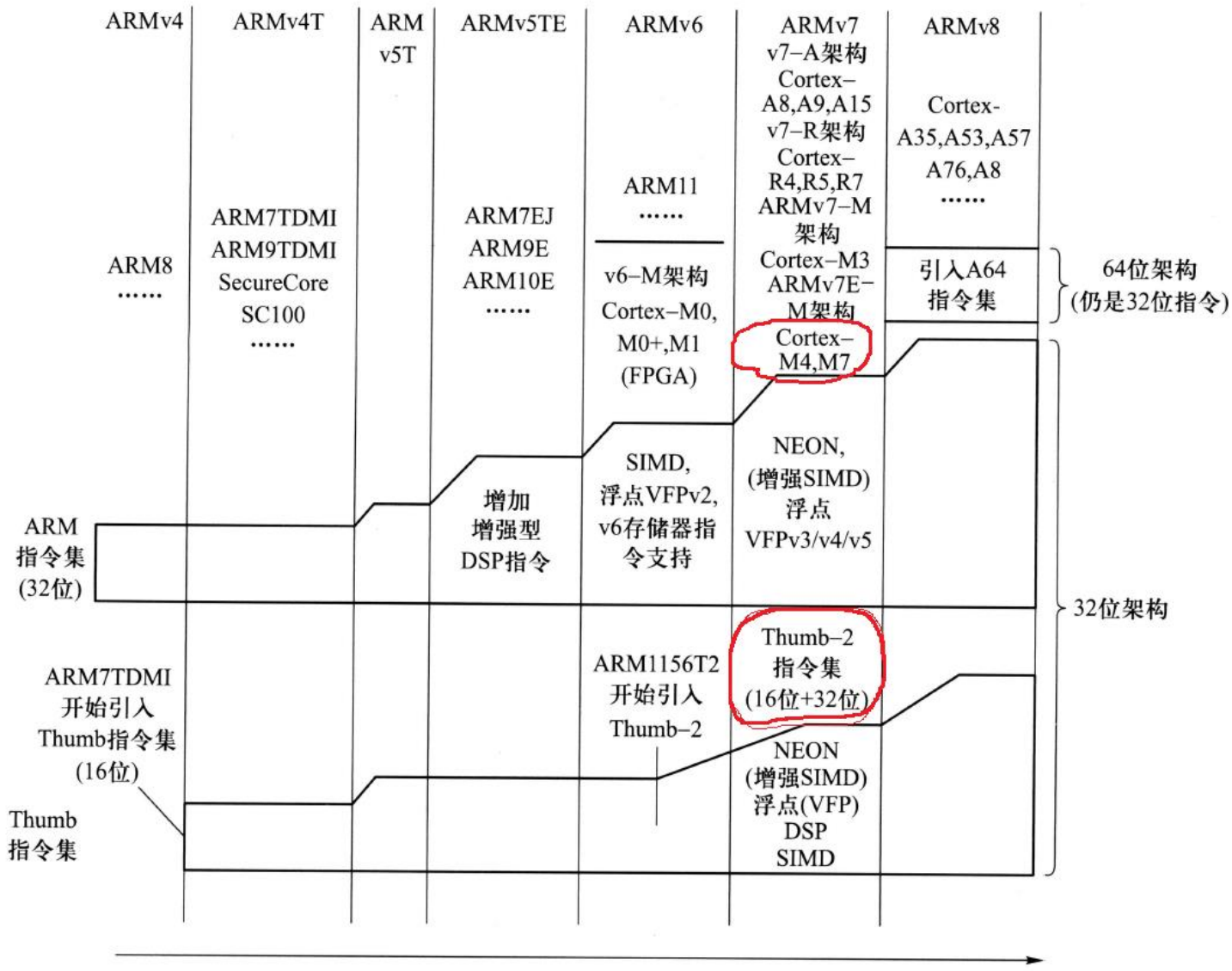


图 3–1–1　ARM 架构演进图

3.1.1 ARM 指令系统简介

体系架构与指令系统

由于不同指令集的语法不同， ARM 开发工具中新支持的统一汇编语言UAL（*Unified Assembler Language*）统一了ARM 指令集和Thumb 指令集的语法，在书写Thumb-2 指令时不需要分析这条指令是32 位还是16 位指令，汇编器会按照最简原则自动完成汇编，但在统一汇编语言UAL 中可以利用后缀".N” 和 “.W” 来指定指令长度。

```
ADD     R1,  R2                ; 自动汇编成 16 位代码
ADD     R1,  R2,  #0x8000      ; 有大于 8 位数范围立即数的指令汇编成 32 位代码
ADD     R1,  R1,  R2           ; 自动汇编成 16 位代码
```

3.1.1 ARM 指令系统简介

体系架构与指令系统

如果有特殊指定可以在指令后加后缀，.W 后缀指定32 位代码格式，.N后缀指定16 位代码格式，.N 后缀不能把原是32 位的代码变成16 位的代码。

```
ADD.W  R1,    R2                    ; 原来是 16 位代码,指定汇编为 32 位代码
ADD.N  R1,    R1,    R2             ; 汇编为 16 位代码
```

* 大部分16位指令只能存取R0-7低存储器。少数才可以访问R8-15

3.1.2 指令格式

* 语法格式

<opcode>{<cond>}{S} <Rd>,<Rn>{,<operand2>}

其中：<>号内的项是必须的，{}号内的项是可选的.

opcode：指令助记符；

cond：执行条件；

S：是否影响APSR寄存器的值；

Rd：目标寄存器；

Rn：第1个操作数的寄存器；

operand2：第2个操作数；

3.1.2 指令格式

* ARM指令集——条件码

ARM指令的基本格式如下：

<opcode> {<cond>} {S} <Rd> ,<Rn>{,<operand2>}

条件执行指令必须位于if-then指令模块中。

• 指令条件码表

操作码	条件助记符	标志	含义
0000	EQ	Z=1	相等
0001	NE	Z=0	不相等
0010	CS/HS	C=1	无符号数大于或等于
0011	CC/LO	C=0	无符号数小于
0100	MI	N=1	负数
0101	PL	N=0	正数或零
0110	VS	V=1	溢出
0111	VC	V=0	没有溢出
1000	HI	C=1,Z=0	无符号数大于
1001	LS	C=0,Z=1	无符号数小于或等于
1010	GE	N=V	有符号数大于或等于
1011	LT	N!=V	有符号数小于
1100	GT	Z=0,N=V	有符号数大于
1101	LE	Z=1,N!=V	有符号数小于或等于
1110	AL	任何	无条件执行 (指令默认条件)
1111	NV	任何	从不执行(不要使用)

3.1.2 指令格式

* ARM指令集——条件执行

Example 3-1: absolute value shows the use of a conditional instruction to find the absolute value of a number. R0 = abs(R1).

Example 3-1: absolute value

```
MOVS    R0, R1          ; R0 = R1, setting flags.
IT      MI              ; Skipping next instruction if value 0 or
                        ; positive.
RSBMI   R0, R0, #0      ; If negative, R0 = -R0.
```

Example 3-2: compare and update value shows the use of conditional instructions to update the value of R4 if the signed values R0 is greater than R1 and R2 is greater than R3.

Example 3-2: compare and update value

```
CMP     R0, R1          ; Compare R0 and R1, setting flags.
ITT     GT              ; Skip next two instructions unless GT condition
                        ; holds.
CMPGT   R2, R3          ; If 'greater than', compare R2 and R3, setting
                        ; flags.
MOVGT   R4, R5          ; If still 'greater than', do R4 = R5.
```

3.1.2 指令格式

* ARM指令集——第2个操作数

ARM指令的基本格式如下：

```
<opcode> {<cond>} {S}   <Rd> ,<Rn>{,<operand2>}
```

灵活的使用第2个操作数“operand2”能够提高代码效率。它有如下的形式：

- #immed_xr——常数表达式；
- Rm——寄存器方式；
- Rm,shift——寄存器移位方式；

3.1.2 指令格式

Thumb-2指令可以支持immed_8 、immed_12 和immed_16 等不同格式的立即数。immed_8 格式是一个8 位范围的立即数格式，immed_12 格式立即数是由一个8 位立即数按照循环码移位获得的数值， immed_16 是一个16 位范围的立即数格式。

以immed_12为例：

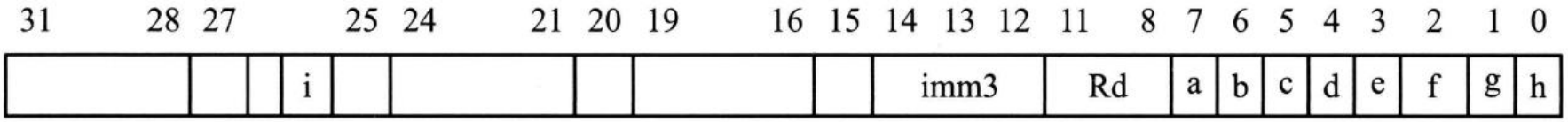


图 3-1-3　第二操作数为 immed_12 格式立即数的指令编码格式（Thumb-2 指令集）

a 、b 、c 、……、h 是8 位立即数的具体位(immed_8) ， 而i 、3 位imm3 和a 组成了循环右移的位数，移位位数的范围是8~31 。

3.1.2 指令格式

表 3-1-2　Thumb-2 指令集中 immed_12 格式立即数的编码表

循环码 i：imm3：a	生成的立即数 （二进制格式，a~h 为任一二进制数值）	说明
0000x	00000000 00000000 00000000 abcdefgh	无移位
0001x	00000000 abcdefgh 00000000 abcdefgh	
0010x	abcdefgh 00000000 abcdefgh 00000000	
0011x	abcdefgh abcdefgh abcdefgh abcdefgh	
01000	1bcdefgh 00000000 00000000 00000000	向右移 8 位
01001	01bcdefg h0000000 00000000 00000000	向右移 9 位
01010	001bcdef gh000000 00000000 00000000	向右移 10 位
01011	0001bcde fgh00000 00000000 00000000	向右移 11 位
（n）	8 位二进制常数继续向右移位	向右移 n 位
11101	00000000 00000000 000001bc defgh000	向右移 29 位
11110	00000000 00000000 0000001b cdefgh00	向右移 30 位
11111	00000000 00000000 00000001 bcdefgh0	向右移 31 位

3.1.2 指令格式

合法的立即数：0xFF 、0xl0l 、0x104 、0xFFFF 、0x10001 、0x20002 、0xlF001F 、0x32003200

不合法的立即数：0x10002 、0x2E001E、0xF000000F

3.1.2 指令格式

* ARM指令集——第2个操作数

▪Rm——寄存器方式

在寄存器方式下，操作数即为寄存器的数值。

例如：

SUB　　R1,R1,R2

3.1.2 指令格式

* ARM指令集——第2个操作数

▪Rm,shift——寄存器移位方式

将寄存器的移位结果作为操作数，但Rm值保持不变，移位方法如下：

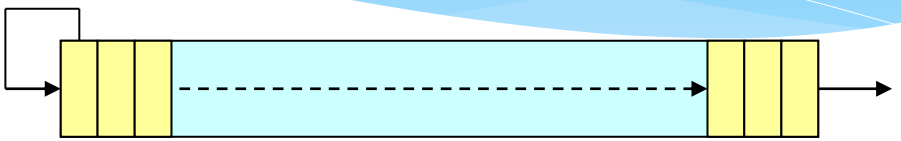
操作码	说明	操作码	说明
LSL #n	逻辑左移n位	ROR #n	循环右移n位
LSR #n	逻辑右移n位	RRX	带扩展的循环右移1位
ASR #n	算术右移n位	Type Rs	Type为移位的一种类型，Rs为偏移量寄存器，低8位有效。

ARM指令集——第2个操作数

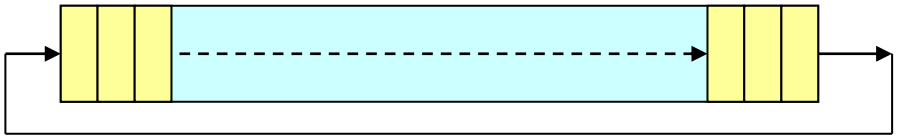
LSL移位操作：

LSR移位操作：

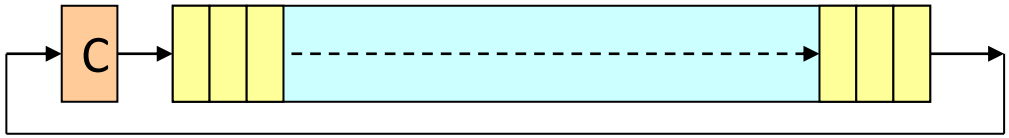
ASR移位操作：



ROR移位操作：

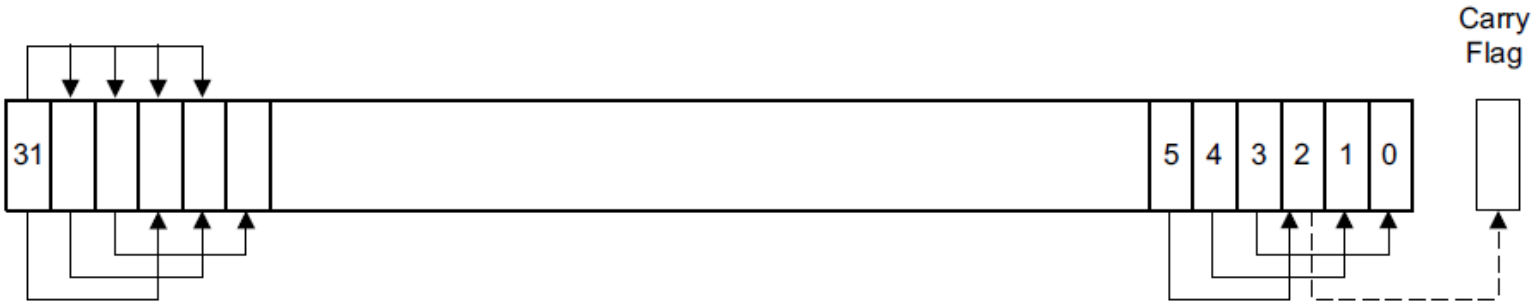


RRX移位操作：



操作码	说明	操作码	说明
LSL #n	逻辑左移n位	ROR #n	循环右移n位
LSR #n	逻辑右移n位	RRX	带扩展的循环右移1位
ASR #n	算术右移n位	Type Rs	Type为移位的一种类型，Rs为偏移量寄存器，低8位有效。

* 算数右移3位



R0=0x800000A5,

3.1.2 指令格式

* ARM指令集——第2个操作数

  - Rm,shift——寄存器移位方式

  例如：

  ADD　　R1,R1,R1,LSL #3　;R1=R1+R1*8=9R1

  SUB　　R1,R1,R2,LSR R3　;R1=R1-(R2/2^{R3})

指令的编码

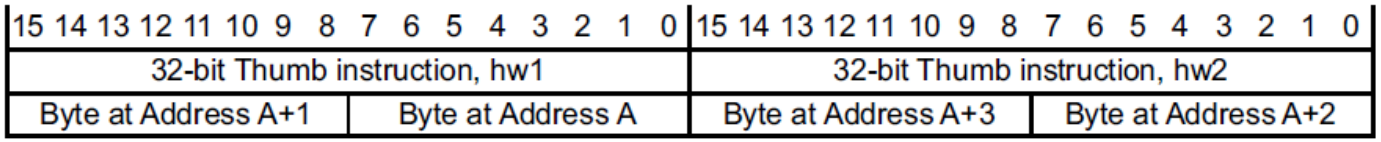
- 每一条指令是16位长的半字或由2个半字组成的32位指令。所有指令是16位对称方式存储的。
- 如果一个半字的高5位[15:11]是如下的数值，那么此半字就是一个32位指令的前半字。否则，此半字就是1条16位指令。

  0b11101

  0b11110

  0b11111

- 32位指令由hw1和hw2两个半字组成，hw1位于低地址。下图所示的指令编码图，给出了其组成的4个字节的存放顺序。



32位指令在内存中存放的字节顺序