

第八章 Cortex-M3组成和结构

8.1 嵌入式系统简介

8.2 Cortex-M3 概述

8.3 Cortex-M3 基础

8.4 Cortex-M3 存储系统

8.5 Cortex-M3 指令集



8.1 嵌入式系统简介

➤ **嵌入式系统**是用来实现一种或多种功能并且常常带有实时计算要求的计算机系统。它作为一个完整设备的嵌入部分。

➤ **IEEE**：嵌入式系统是“用于监视、控制或者辅助操作机器和设备的装置”

➤ 国内普遍认同的嵌入式系统**定义**为：**以应用为中心，以计算机技术为基础，并且软硬件可裁剪，适用于应用系统对功能、可靠性、成本、体积、功耗有严格要求的专用计算机系统。**

➤ **专用计算机系统（非PC智能电子设备）**

➤ 面向应用

➤ 以计算机技术为基础

➤ 软件、硬件可裁剪

➤ 有严格约束：功能、可靠性、成本、体积、功耗



8.1 嵌入式系统简介



2、嵌入式系统发展历史

在20世纪30年代到40年代, 计算机某些时候常常只致力于完成一个任务, 但是它太大又太贵。

阿波罗导航电脑属于第一批被承认的现代嵌入式系统, 由麻省理工的Charles Stark Draper发明。

Intel 4004是第一个微处理器, 针对计算器和其他小型系统设计。

到了20世纪80年代中期, 大部分以前共同的外部系统组件已经被集成到了同样的芯片上。

微控制器的集成增加了传统计算机不会有的应用。



8.1 嵌入式系统简介



3、嵌入式系统特点

1). 嵌入式系统被设计用来做一些特殊的任务。

一些嵌入式系统有必须达到的实时性能要求; 另一些嵌入式系统在实时性能方面的要求会低些或者没有要求。

2). 许多嵌入式系统由在一个大设备里提供通用功能的小电脑零件组成。

3). 为嵌入式系统所写的程序指令被称为固件, 被存储在只读存储器或者Flash存储器芯片内。



8.1 嵌入式系统简介

4、嵌入式系统应用

嵌入式系统跨越现代生活的各个方面，其使用也有许多例子。

- 电信系统在网络到移动电话最终使用者的电话交换机上应用了很多嵌入式系统。
- 消费类电子产品包括个人数字助理(PDAs), mp3 播放器, 移动电话, 和 视频游戏机。



8.1 嵌入式系统简介

- 交通系统中从飞机到汽车越来越多地使用嵌入式系统。
- 新飞机中包含的先进航空电子设备如惯性导航系统和GPS接收机也有相当高的安全要求。



- 各种电动马达— 无刷直流电动机, 感应电机和直流电动机 — 正在使用电动/电子马达控制器。



8.1 嵌入式系统简介

5、嵌入式系统中的处理器

嵌入式处理器可以分为两大类：普通的微处理器 (μP) 和 微控制器 (μC)。

1. 相当大数目的基本CPU架构被使用。

冯·诺依曼和哈佛架构。

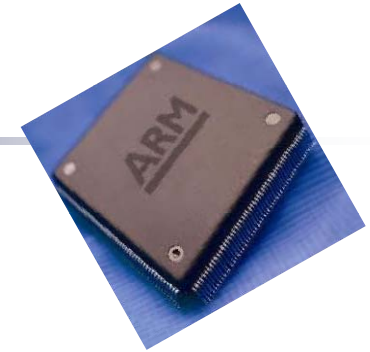
RISC 以及 non-RISC 和 VLIW.

2. 字长从4位变化到64位及以后(主要是在DSP处理器中)。

3. 大多数架构从大量的不同变种和样式中来。



8.1 嵌入式系统简介



ARM

ARM,是一个以Advanced RISC Machine为人所知的公司,和在此之前的the Acorn RISC Machine。

截至2007年,每年销量达十几亿的手机中有大约98%至少使用了一个ARM处理器。

截至2009年,ARM处理器大约占有所有嵌入式32位RISC处理器的大约90%。

ARM处理器广泛用于消费类电子产品,包括掌上电脑,移动电话,数字媒体和音乐播放器。



8.1 嵌入式系统简介

ARM公司的业务一直是出售IP核。

最成功的ARM7TDMI已经有亿万售出。

现在或先前ARM授权的公司包括：

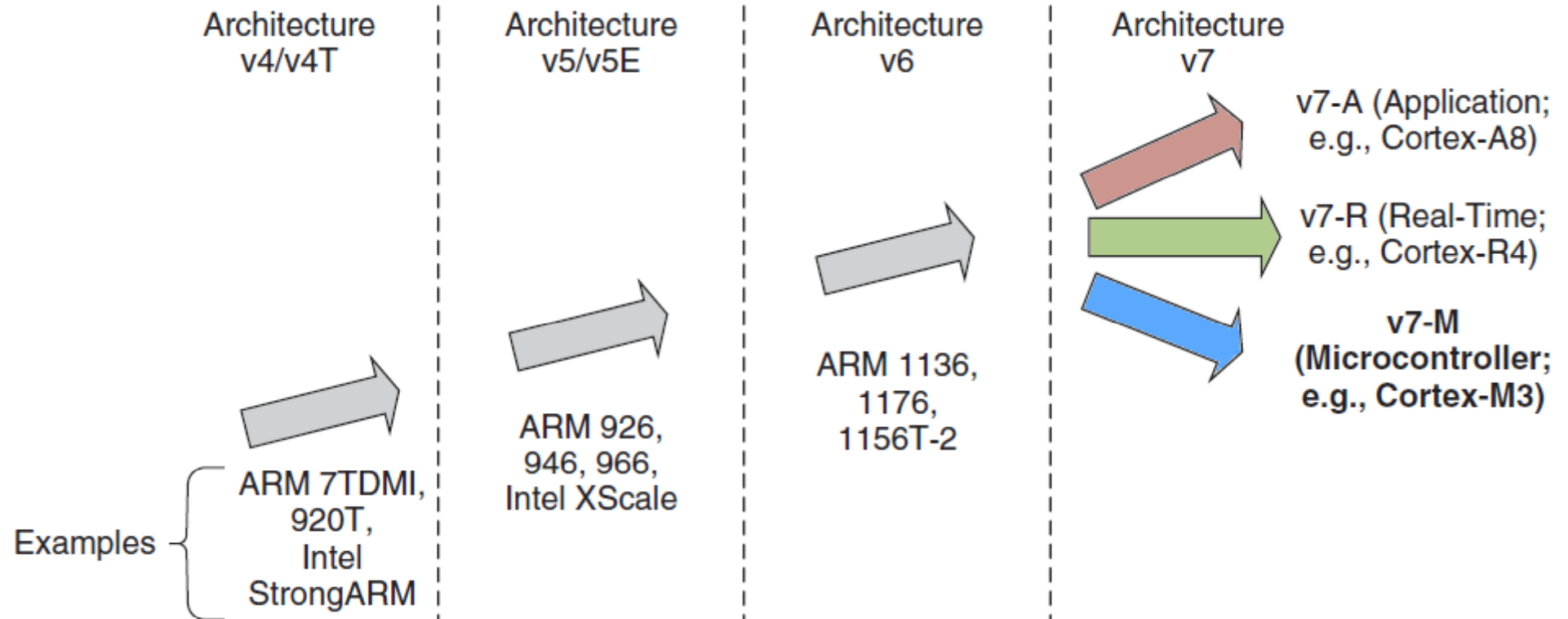
Alcatel-Lucent, Apple Inc., Atmel, Broadcom, Freescale, Intel ,Samsung, TI, ect.



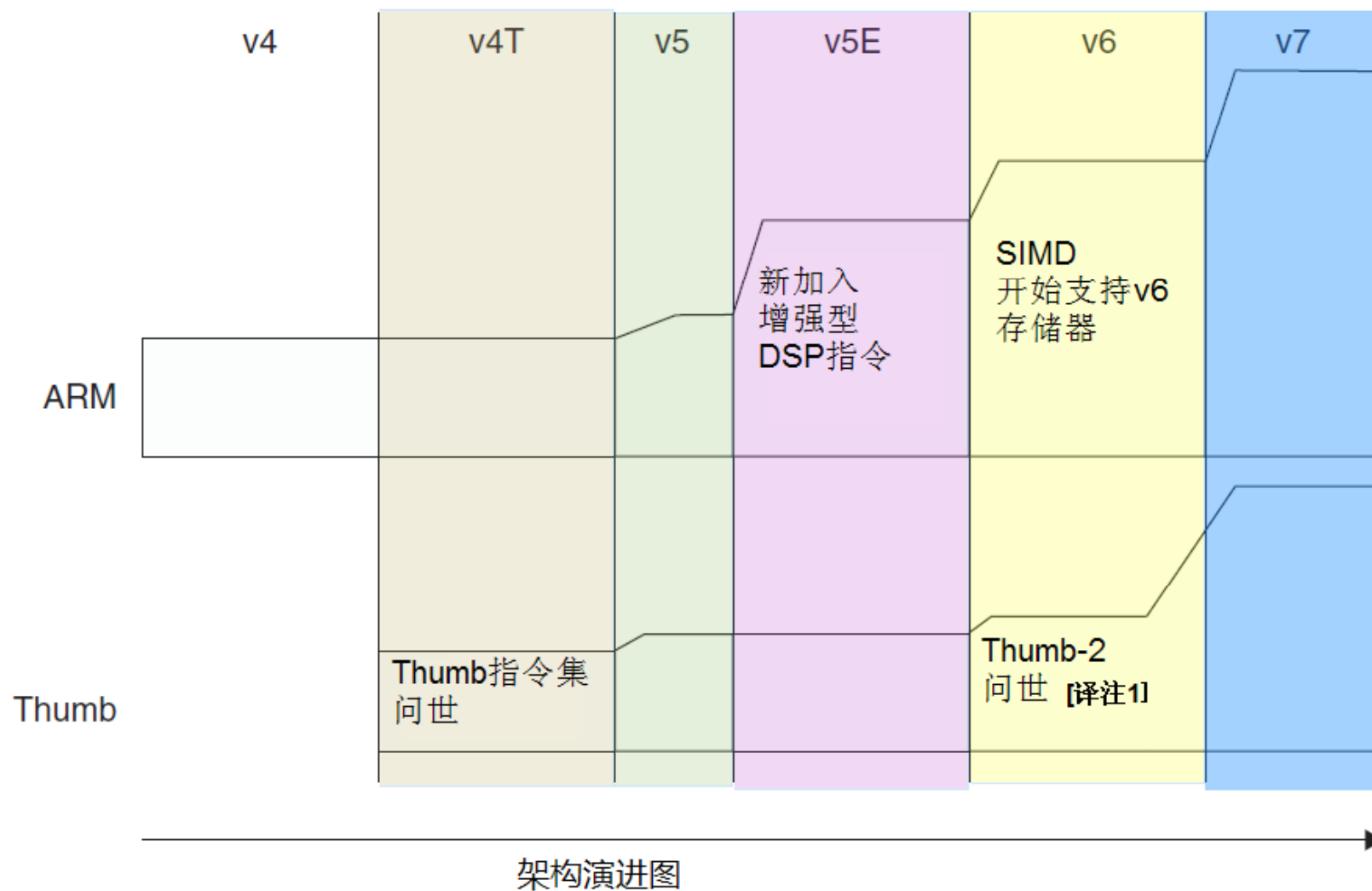
ARM控股的ARM处理器家庭的突出例子包括ARM7, ARM9, ARM11 and Cortex.



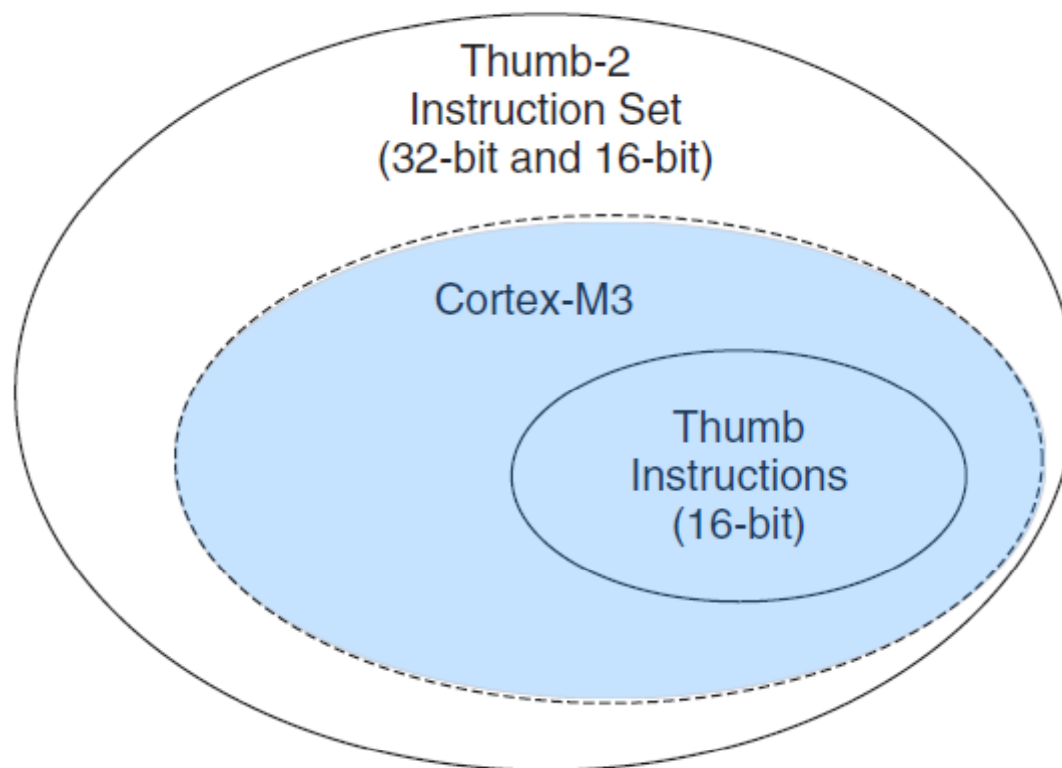
ARM处理器架构发展进程



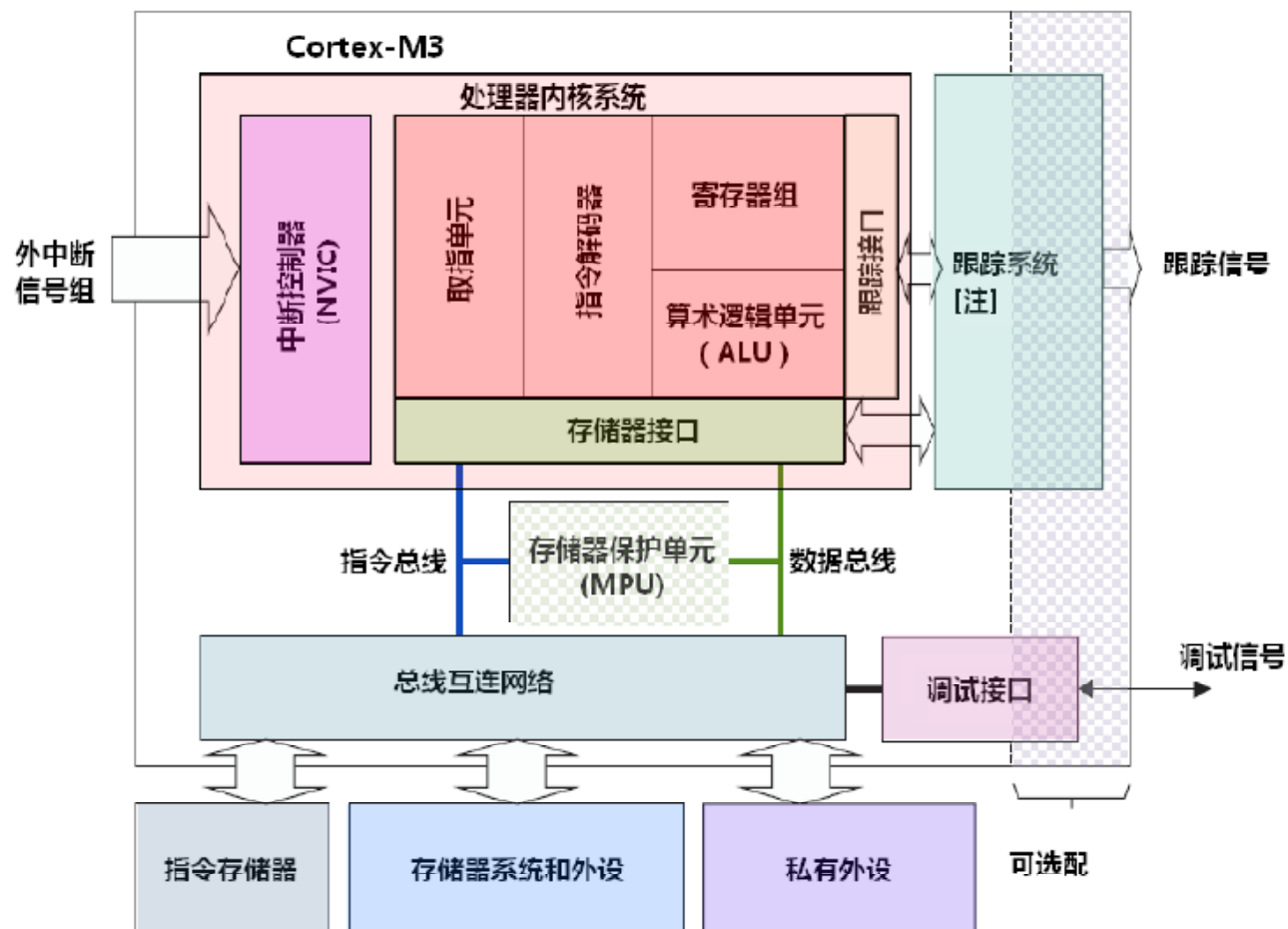
指令集演进图



Thumb-2指令集与Thumb指令集的关系



8.2 Cortex-M3 概述



Cortex-M3的简化视图

8.2 Cortex-M3 概述

Cortex-M3内部资源

1. 32-bit微处理器: 32-bit 数据路径, 32-bit 寄存器组, 32-bit 存储器接口。
2. 哈佛架构: 独立的指令总线 and 数据总线。这允许指令和数据在同一时间产生。
3. 存储空间: 4GB。
4. 寄存器: 寄存器(R0 到 R15)和特殊寄存器。
5. 运行模式: 线程模式和处理模式;特权级和用户级。



8.2 Cortex-M3 概述

6. 中断和异常: 内置在嵌套向量中断控制器; 支持11种系统异常外加240种外部IRQ。
7. 总线接口: 若干总线接口允许Cortex-M3同时取指令和取数据。
8. MPU: 一个可选的存储器保护单元, 允许对特权访问和用户程序访问制定访问规则。
9. 指令集: Thumb-2 指令集; 允许 32-位指令和16-位指令被同时使用。
10. 固定的内部调试组件: 提供调试操作支持和像断点调试这样的功能。



8.3 Cortex-M3 基础

1. 寄存器

1) 通用寄存器

1. R0~R7 (低寄存器):

可以被所有16-bit Thumb指令
和所有32-bit Thumb-2指令访问。

2. R8~R12 (高寄存器):

可以被所有32-bit Thumb-2指令
但不能被所有16-bit Thumb
指令访问。

D ₃₁	D ₀	
	R0	通用寄存器 (低组)
	R1	
	R2	
	R3	
	R4	
	R5	
	R6	
	R7	
	R8	通用寄存器 (高组)
	R9	
	R10	
	R11	
	R12	
	R13(MSP)	主堆栈指针(MSP)
	R13(PSP)	进程堆栈指针(PSP)
	R14(LR)	连接寄存器(LR)
	R15(PC)	程序寄存器(PC)



8.3 Cortex-M3 基础

2) 堆栈指针

R13是堆栈指针。两个堆栈指针是**库存的**所以在同一时间只有一个可见。

堆栈指针的最低两位总是0, 这意味着它们总是字对齐。

两个堆栈指针是:

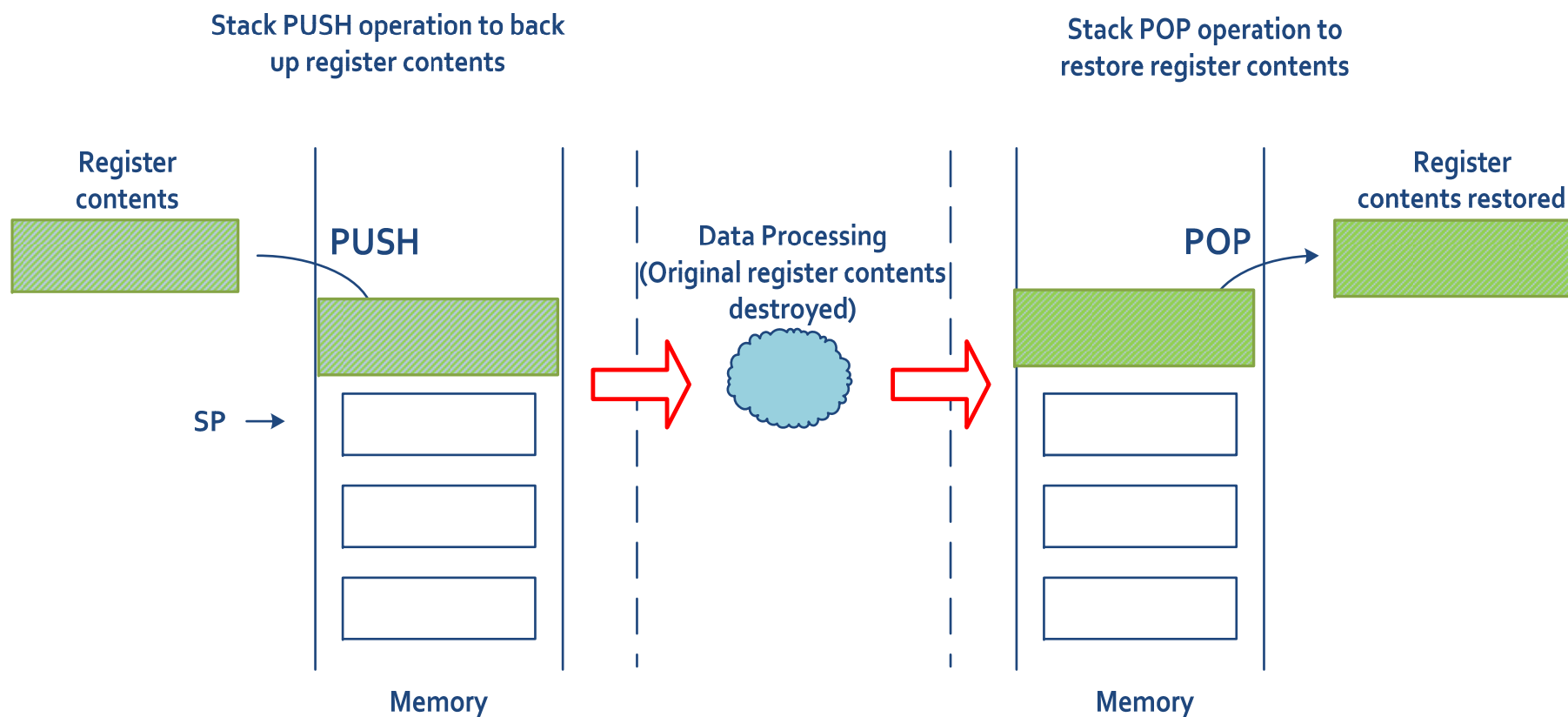
1. **主堆栈指针(MSP)**: 这是默认的堆栈指针。
2. **进程堆栈指针(PSP)**: 被基本级别的应用程序代码所使用。

D ₃₁	D ₀	
R0		通用寄存器 (低组)
R1		
R2		
R3		
R4		
R5		
R6		
R7		
R8		通用寄存器 (高组)
R9		
R10		
R11		
R12		
R13(MSP)		主堆栈指针(MSP)
R13(PSP)		进程堆栈指针(PSP)
R14(LR)		连接寄存器(LR)
R15(PC)		程序寄存器(PC)



8.3 Cortex-M3 基础

堆栈指针用来进行堆栈存储器操作,比如PUSH和POP。



堆栈存储器的基本概念



8.3 Cortex-M3 基础

汇编语言的语法:

```
PUSH {R0}      ; R13-4, then Memory[R13] R0  
POP {R0}       ; R0 Memory[R13], then R13 R13+4
```

你可以在一条指令中**PUSH**或**POP**多个寄存器:

```
subroutine_1  
  PUSH    {R0-R7, R12, R14}    ; Save registers  
  ...                               ; Do your processing  
  POP     {R0-R7, R12, R14}    ; Restore registers  
  BX      R14                  ; Return to calling function
```



3) 连接寄存器

R14 是连接寄存器 (LR)。当一个子程序或函数被调用时，LR用来存储返回的程序计数器。

Main ; Main program

...

BL func1 ; Call function1
; PC = function1
; LR = next instruction

... ; program code

func1

...

BX LR ; Return



D ₃₁	D ₀	
R0		通用寄存器 (低组)
R1		
R2		
R3		
R4		
R5		
R6		
R7		
R8		通用寄存器 (高组)
R9		
R10		
R11		
R12		
R13(MSP)		主堆栈指针(MSP)
R13(PSP)		进程堆栈指针(PSP)
R14(LR)		连接寄存器(LR)
R15(PC)		程序寄存器(PC)

4) 程序计数器

R15 是程序计数器。可以在汇编语言中通过R15或PC访问。

示例:

0x1000 :

MOV R0, PC ; R0 = 0x1004

D ₃₁	D ₀	
R0		通用寄存器 (低组)
R1		
R2		
R3		
R4		
R5		
R6		
R7		
R8		通用寄存器 (高组)
R9		
R10		
R11		
R12		
R13(MSP)		主堆栈指针(MSP)
R13(PSP)		进程堆栈指针(PSP)
R14(LR)		连接寄存器(LR)
R15(PC)		程序寄存器(PC)

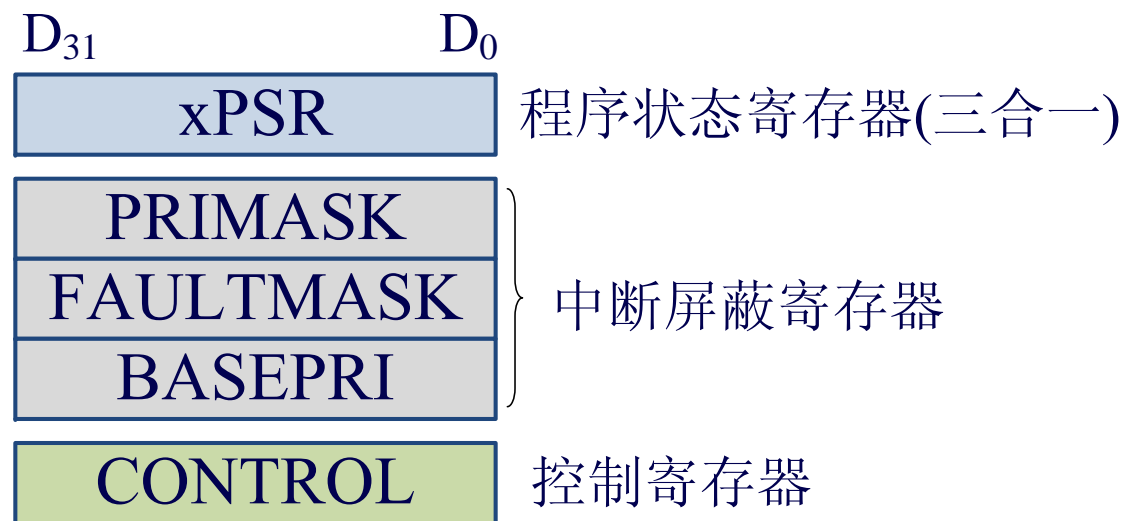


8.3 Cortex-M3 基础

2. 特殊寄存器

在Cortex-M3 处理器中的特殊寄存器包括:

1. 程序状态寄存器 (PSRs);
2. 中断屏蔽寄存器 (PRIMASK, FAULTMASK, and BASEPRI);
3. 控制寄存器 (CONTROL)。



Cortex-M3中的特殊寄存器



8.3 Cortex-M3 基础

1) 程序状态寄存器 (PSRs)

程序状态寄存器可以分为三个状态寄存器：

1. 应用 PSR (APSR)
2. 中断 PSR (IPSR)
3. 执行 PSR (EPSR)

Cortex-M3中的结合程序状态寄存器

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



8.3 Cortex-M3 基础

Cortex-M3中的程序状态寄存器 (PSRs)

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

EPSR 和IPSR 是只读的:

MRS	r0, APSR	; Read Flag state into R0
MRS	r0, IPSR	; Read Exception/Interrupt state
MRS	r0, EPSR	; Read Execution state
MSR	APSR, r0	; Write Flag state



8.3 Cortex-M3 基础

Cortex-M3 中程序状态寄存器的位域

Bit	Description
N	负
Z	零
C	进位/借位
V	溢出
Q	置顶饱和标记
ICI/IT	中断可持续指令(ICI) 位, IF-THEN 指令状态位
T	Thumb 状态, 总是 1; 试图清除此位将导致错误异常
异常号	指出处理器正在处理哪一个异常



8.3 Cortex-M3 基础

2) PRIMASK, FAULTMASK 和 BASEPRI 寄存器

PRIMASK, FAULTMASK, 和 BASEPRI 寄存器被用来禁用异常。

Cortex-M3 中断屏蔽寄存器

寄存器名	描述
PRIMASK	一个1-bit 寄存器。当置位时, 它允许NMI 和硬件默认异常; 所有其他的中断和异常将被屏蔽。
FAULTMASK	一个1-bit 寄存器。当置位时, 它只允许NMI, 所有中断和默认异常处理被忽略。
BASEPRI	一个9位寄存器。它定义了屏蔽优先级。 当它置位时, 所有同级的或低级的中断被忽略。



8.3 Cortex-M3 基础

当访问PRIMASK, FAULTMASK, 和BASEPRI 寄存器时, MRS和MSR指令被使用.



Example:

MRS	r0, BASEPRI	; Read BASEPRI register into R0
MRS	r0, PRIMASK	; Read PRIMASK register into R0
MRS	r0, FAULTMASK	; Read FAULTMASK register into R0
MSR	BASEPRI, r0	; Write R0 into BASEPRI register
MSR	PRIMASK, r0	; Write R0 into PRIMASK register
MSR	FAULTMASK, r0	; Write R0 into FAULTMASK register

在用户访问级, PRIMASK, FAULTMASK, 和BASEPRI寄存器不能被置位。



8.3 Cortex-M3 基础

3) 控制寄存器

控制寄存器被用来定义**特权级别**和选择堆栈指针。这个寄存器有**两位**。

Cortex-M3 控制寄存器

Bit	Function
CONTROL[1]	堆栈状态:(当访问级别改变时自动改变) 1 = 进程堆栈(PSP) 被使用 (针对用户级) 0 = 默认堆栈 (MSP) 被使用(针对特权级)
CONTROL[0]	指定的访问级别: 0 = 特权的线程模式 1 = 用户状态的线程模式



8.3 Cortex-M3 基础

● CONTROL[1]

在Cortex-M3中, 在[处理模式](#)中CONTROL[1] 位总是0 (MSP)。但是, 在线程或基本级别, 它可以为0或1。

● CONTROL[0]

CONTROL[0] 位只在[特权状态](#)可写。

使用MRS和MSR指令来访问控制寄存器:

MRS r0, CONTROL ; Read CONTROL register into R0

MSR CONTROL, r0 ; Write R0 into CONTROL register



8.3 Cortex-M3 基础

3. 操作模式

两种模式和两种权限级别。

在Cortex-M3中的操作模式和权限级别

	特权	用户
当运行一个异常	处理器模式	
当运行主程序	线程模式	线程模式

操作模式决定处理器运行正常程序或运行异常处理程序。



8.3 Cortex-M3 基础

特权级别提供了一种机制来保障访问存储器的关键区域，同时还提供了一个基本的安全模式。

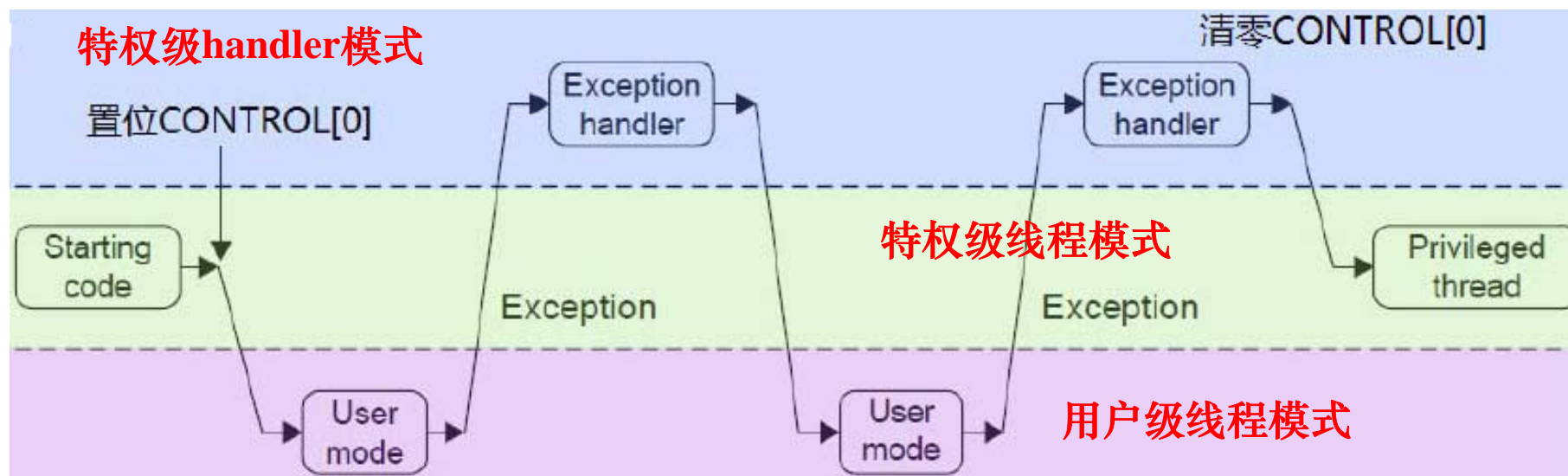
通过写Control register[0]=1，软件在特权访问级别可以使程序转换到用户访问级别。

用户程序不能够通过写控制寄存器直接变回特权状态。

它要经过一个异常处理程序设置Control register[0]=0使得处理器切换回特权访问级别。



8.3 Cortex-M3 基础

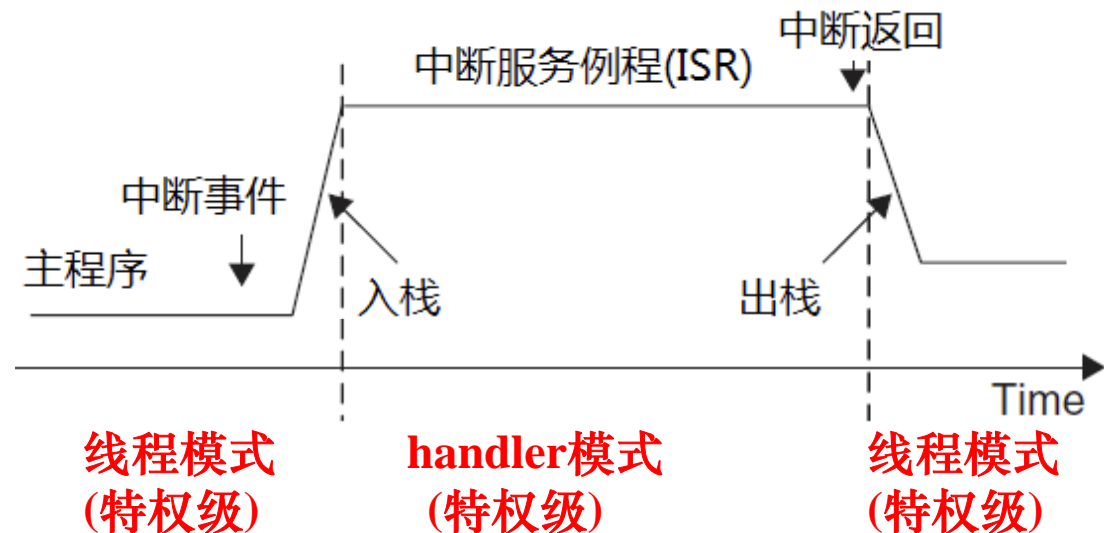


通过控制寄存器或异常来切换操作模式

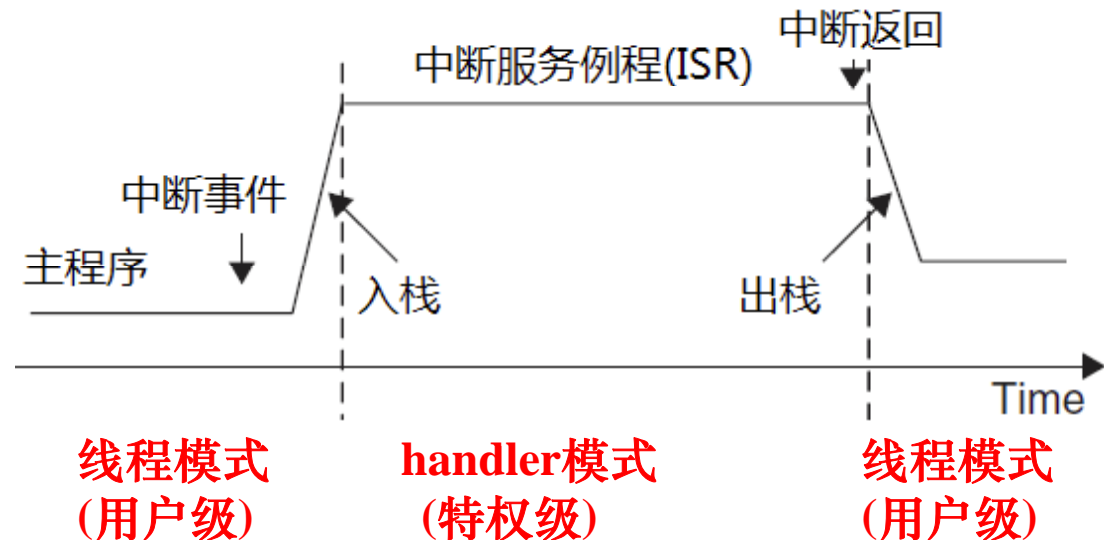


8.3 Cortex-M3 基础

当CONTROL[0]=0时，在异常处理的始末，只发生了处理器模式的转换。



若CONTROL[0]=1
(线程模式+用户级)，则在中断响应的始末，处理器模式和特权等级都要发生变化。



8.3 Cortex-M3 基础

4. 向量表

当CM3内核响应了一个发生的异常后，对应的异常服务例程(ESR)就会执行。为了决定ESR的入口地址，CM3使用了“向量表查表机制”。这里使用一张向量表。向量表其实是一个WORD（32位整数）数组，每个下标对应一种异常，该下标元素的值则是该ESR的入口地址。



例如：

复位是异常类型1。

复位向量的地址是1乘4, 等于 0x00000004, NMI 向量 (类型2) 位于：

$$2 * 4 = 0x00000008$$



地址 0x00000000被用作MSP的开始值。

8.3 Cortex-M3 基础

复位后的向量表定义

异常类型	偏移地址	异常向量
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	保留
12	0x30	调试监视
11	0x2C	SVC
7-10	0x1C-0x28	保留
6	0x18	使用故障
5	0x14	总线故障
4	0x10	存储器管理故障
3	0x0C	硬件故障
2	0x08	NMI
1	0x04	复位
0	0x00	MSP的开始值



8.3 Cortex-M3 基础

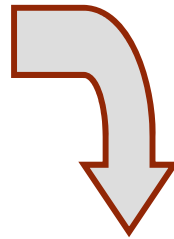
5. 栈内存的操作

1) 堆栈的基本操作

在寄存器中的数据可以通过PUSH操作保存到栈内存并且通过POP操作在稍后恢复到寄存器.

主程序

...
; R0 = X, R1 = Y, R2 = Z
BL function1



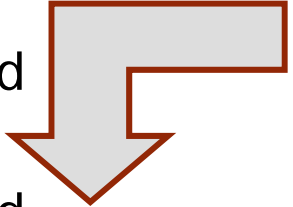
function1
PUSH {R0} ; store R0 to stack & adjust SP
PUSH {R1} ; store R1 to stack & adjust SP
PUSH {R2} ; store R2 to stack & adjust SP



8.3 Cortex-M3 基础

```
...                               ; Executing task (R0, R1 and R2
                                   ; could be changed)
                                   POP {R2}           ; restore R2 and SP re-
adjusted
                                   POP {R1}           ; restore R1 and SP re-
adjusted
                                   POP {R0}           ; restore R0 and SP re-
adjusted
                                   BX LR              ; Return

; Back to main program
; R0 = X, R1 = Y, R2 = Z
... ; next instructions
```



8.3 Cortex-M3 基础

2) 多寄存器堆栈操作

PUSH/POP指令支持一次操作多个寄存器.

PUSH {R0-R2} ;压入R0-R2

PUSH {R3-R5,R8, R12} ;压入R3-R5,R8, 以及R12

在POP时, 可以如下操作:

POP {R3-R5,R8, R12} ;弹出R3-R5, R8, 以及R12

POP {R0-R2} ;弹出R0-R2

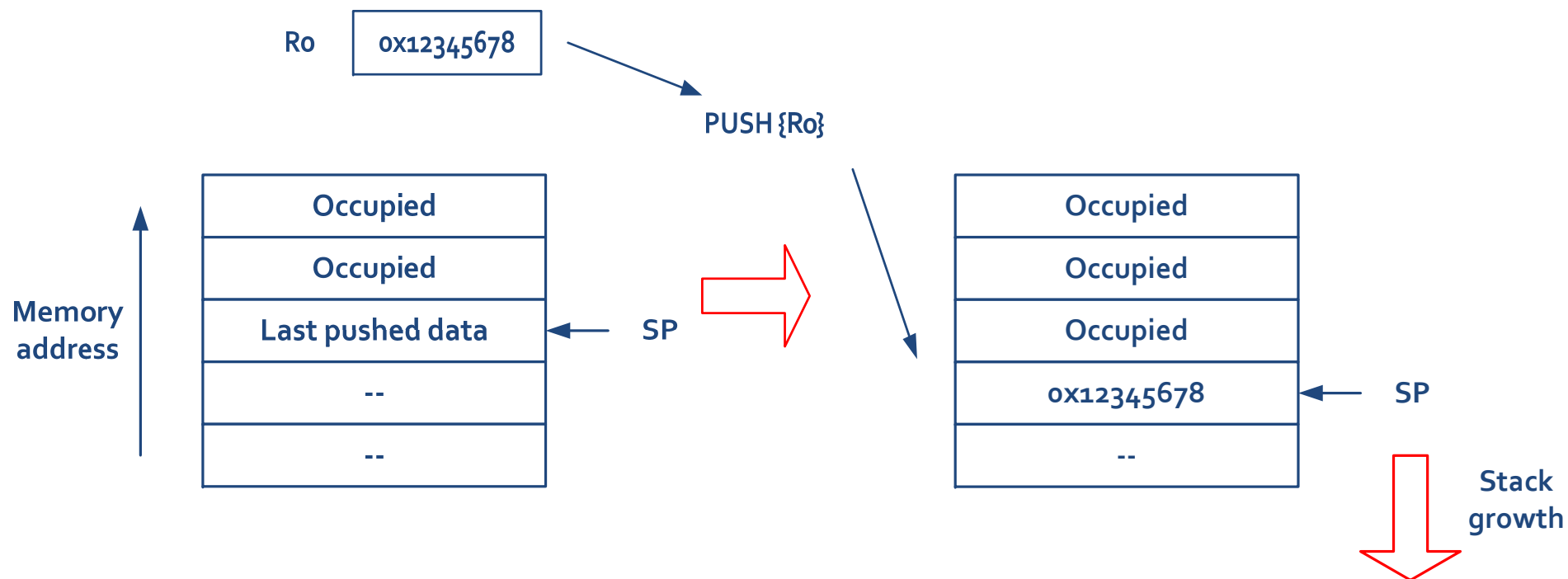
注意: 在寄存器列表中, 不管寄存器的序号是以什么顺序给出的, 汇编器都将把它们升序排序。然后先push序号大的寄存器, 所以也就先pop序号小的寄存器。如果不按升序写寄存器, 也许有些汇编器会给出语法错误。



8.3 Cortex-M3 基础

3) Cortex-M3 的堆栈实现

堆栈指针 (SP) 指向压入堆栈的最后一个数据, SP 在一个新的**PUSH**操作前递减。

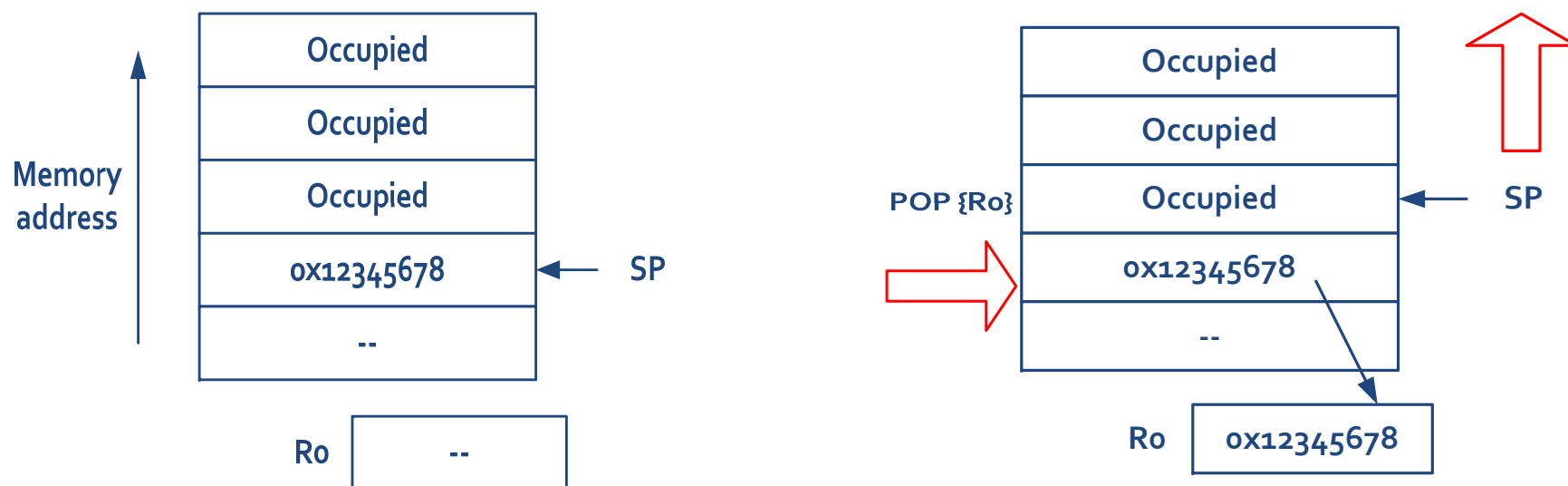


Cortex-M3 堆栈PUSH 实现



8.3 Cortex-M3 基础

对于POP操作, 数据从SP指向的存储区域读取, 然后堆栈指针递增。在存储位置的内容并未发生变化。



Cortex-M3 的堆栈POP 操作实现

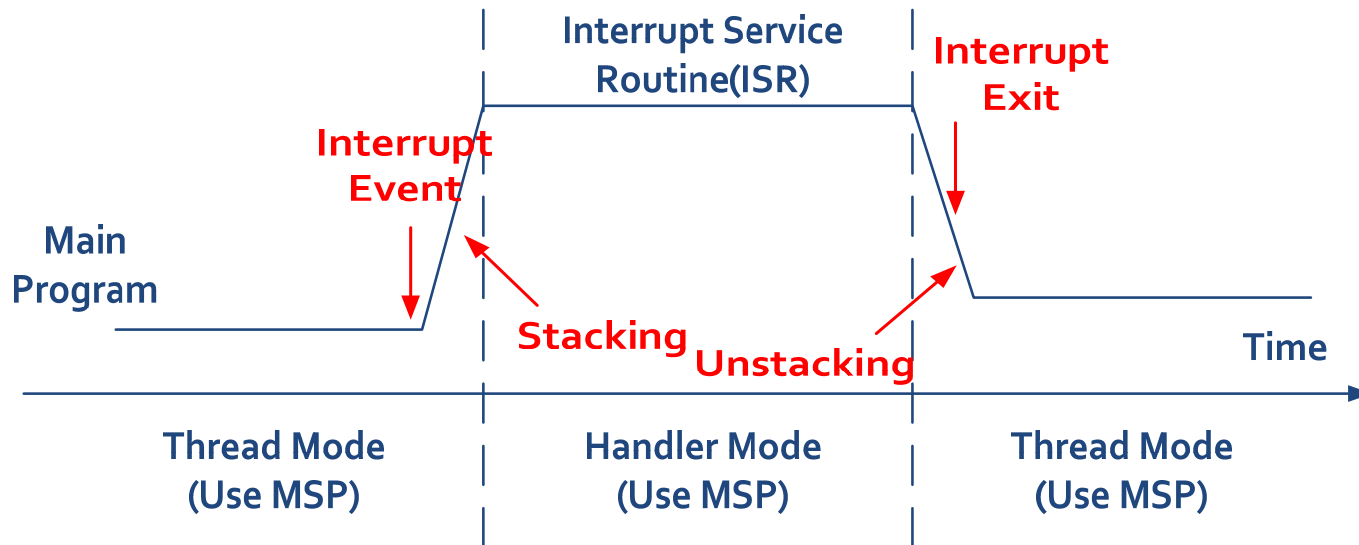


8.3 Cortex-M3 基础

4) 在Cortex-M3中的两种堆栈模式

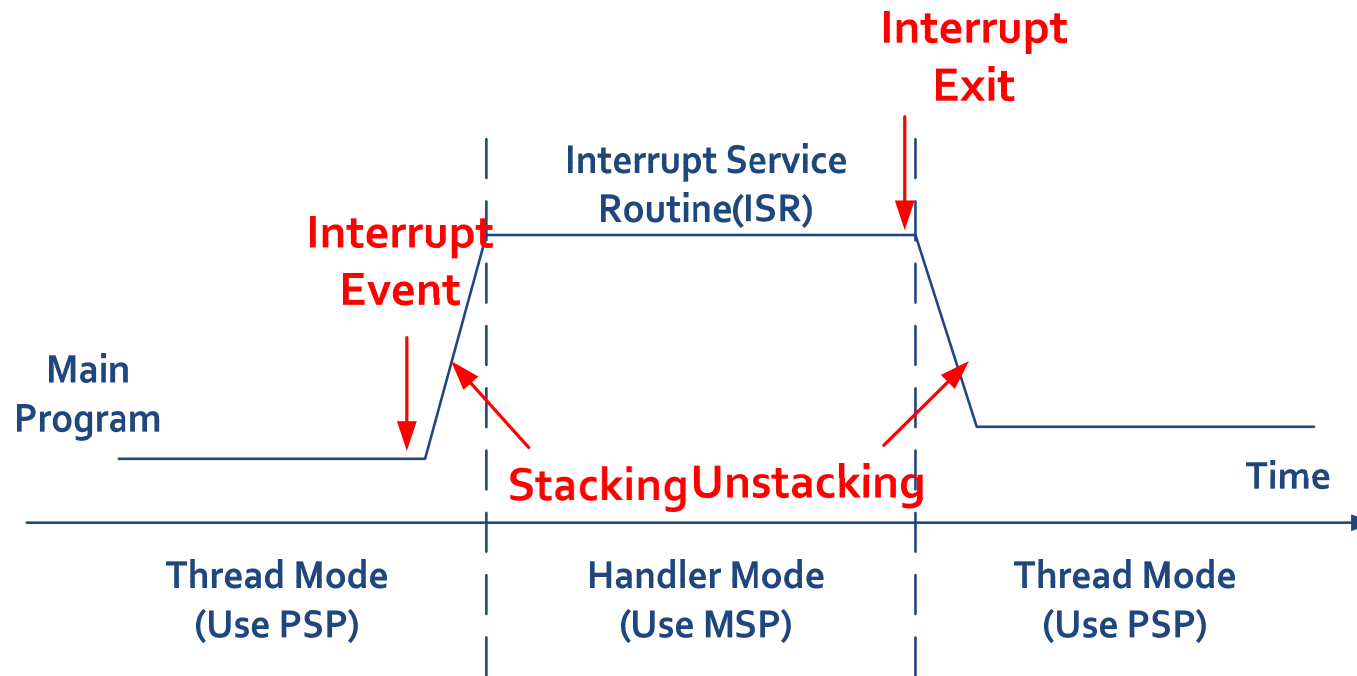
Cortex-M3 有两个堆栈指针: 主堆栈指针 (MSP) 和进程堆栈指针(PSP).

被使用的SP寄存器由控制寄存器的control[1]来控制。



Control [1] = 0: 线程级别和处理程序使用主堆栈

8.3 Cortex-M3 基础



Control[1] = 1:线程级别使用进程堆栈和处理程序使用主堆栈



8.3 Cortex-M3 基础

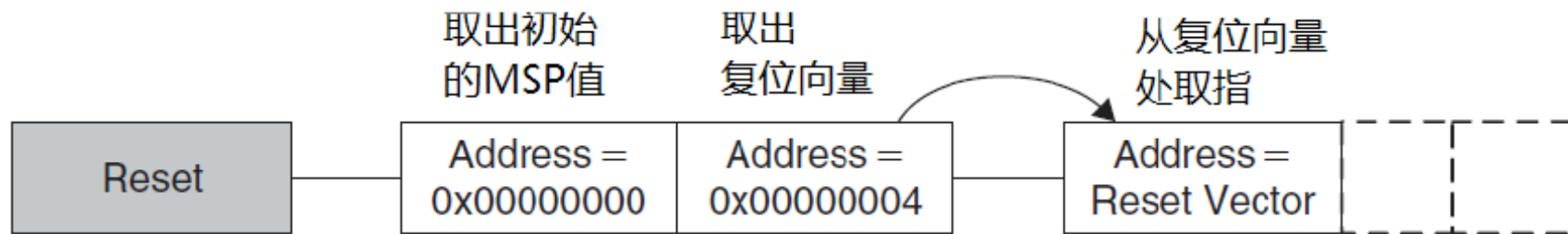
5) 复位序列

处理器退出复位后，它会从存储器中读取两个字：

1. 地址0x00000000: 保存了R13 (堆栈指针)的初始值
2. 地址0x00000004: 保存了复位向量(启动程序的起始地址; LSB应该被设为1来说明 Thumb 状态)



8.3 Cortex-M3 基础

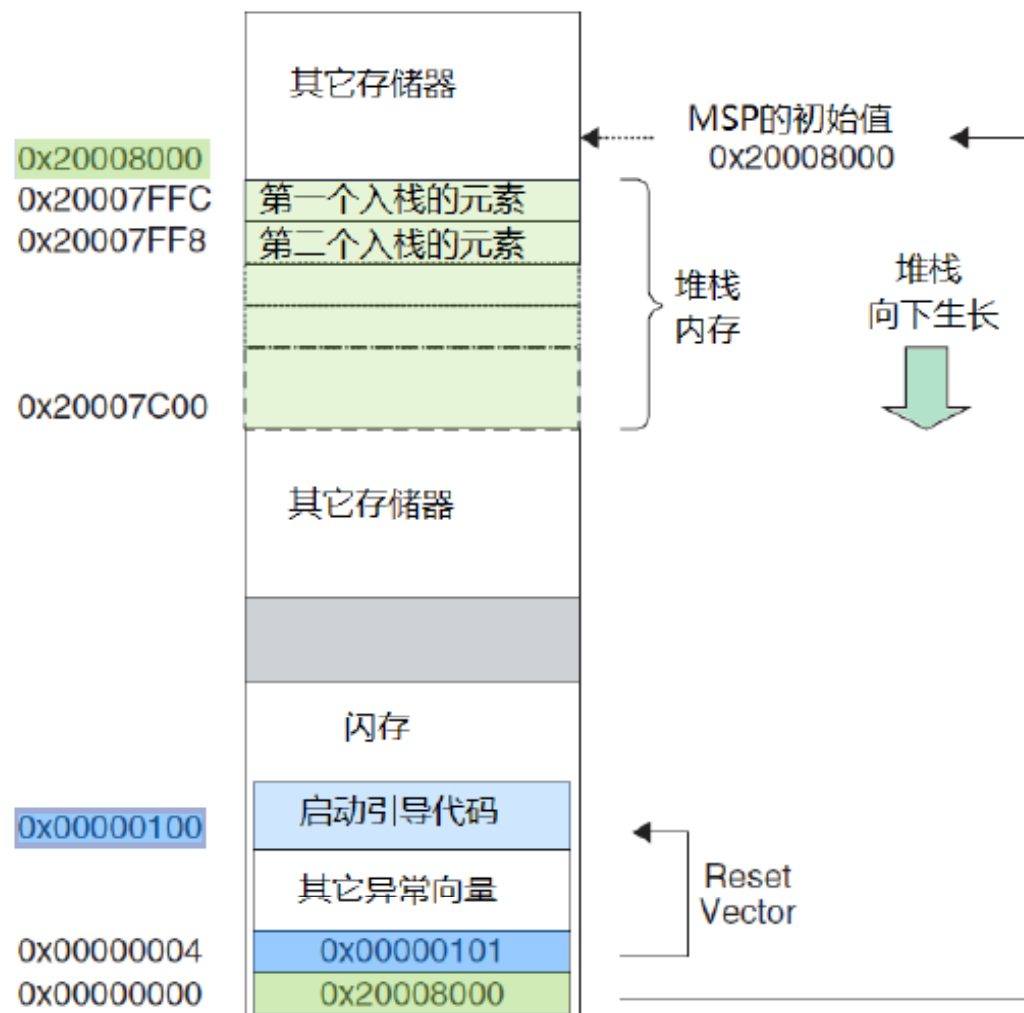


复位序列

向量表从最初的SP值开始。在Cortex-M3中，向量表中向量地址的LSB位必须被设为1来表明它们是Thumb代码。



8.3 Cortex-M3 基础



初始化堆栈指针值和初始化程序计数器 (PC) 值的例子

8.4 Cortex-M3 存储系统

1. 存储系统功能概述

- 1). 预先定义的内存映射指定了在访问存储位置时使用哪个总线接口。
- 2). Bit-band: 它提供了对内存或外设数据的单一比特的原子操作。
- 3). 提供了非对齐传输和专用通道。
- 4). 支持小端配置和大端配置。



8.4 Cortex-M3 存储系统

2. 内存映射

Cortex-M3 处理器有一个固定的存储映射。

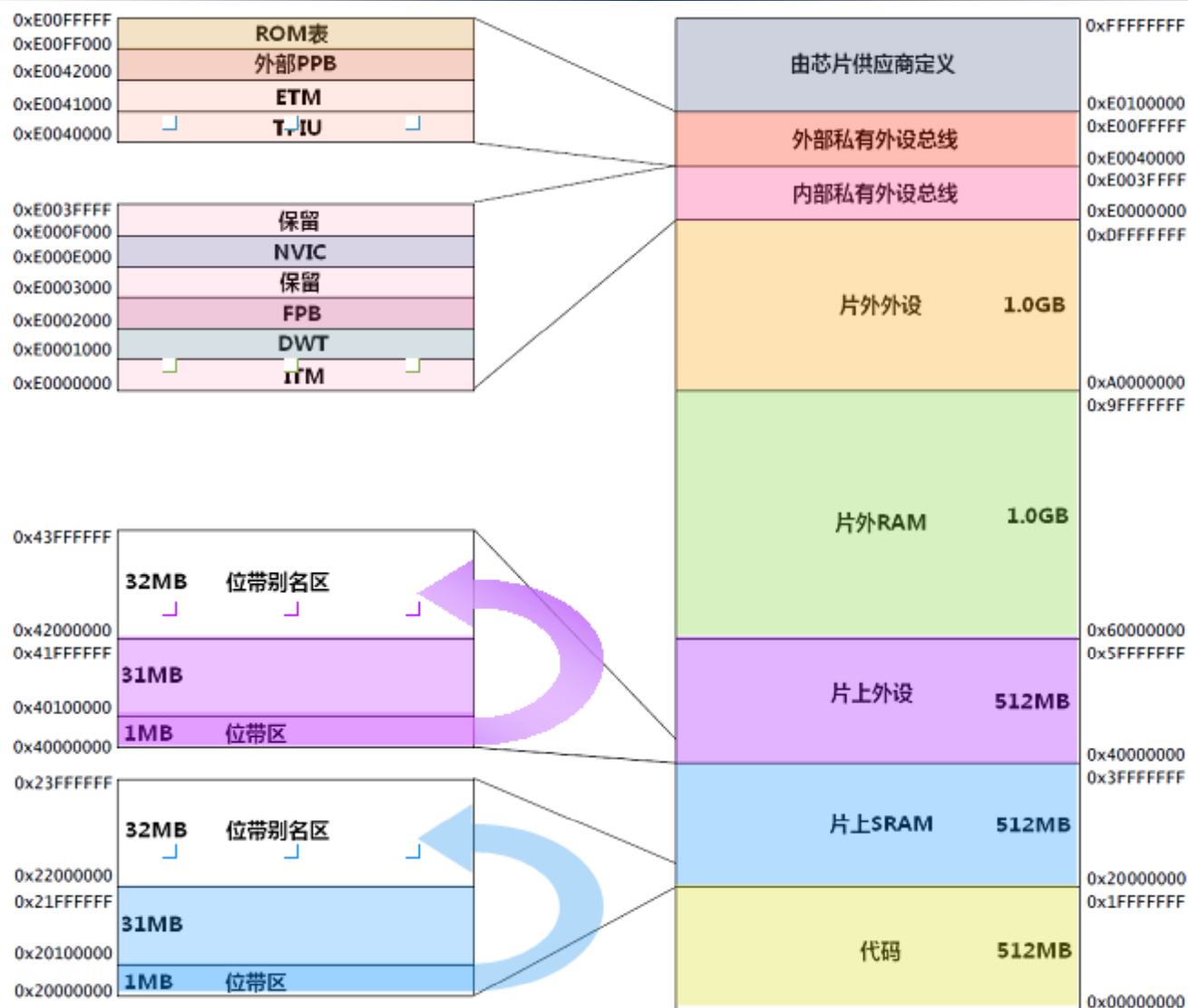
一些存储单元被分配给了私有外设，比如调试组件。

- 1). 闪存地址重载和断点单元(FPB)
- 2). 数据观察点和跟踪单元(DWT)
- 3). 仪表跟踪宏单元(ITM)
- 4). 嵌入式跟踪宏单元(ETM)
- 5). 跟踪端口接口单元(TPIU)
- 6). ROM表

Cortex-M3 处理器总共有 4 GB 的地址空间。



8.4 Cortex-M3 存储系统



Cortex-M3预先定义的内存映射

8.4 Cortex-M3 存储系统

SRAM: 0.5GB。

SRAM存储范围是用来连接内部的SRAM。

片上外设: 0.5GB。

支持 bit-band别名并且可以通过系统总线接口访问。

外部RAM: 1GB。允许执行程序。

外部设备: 1GB。不允许执行程序。

系统级组件+ 内部专用的外围总线+ 外部私有外设
总线+ 供应商特定系统外设: 0.5GB。

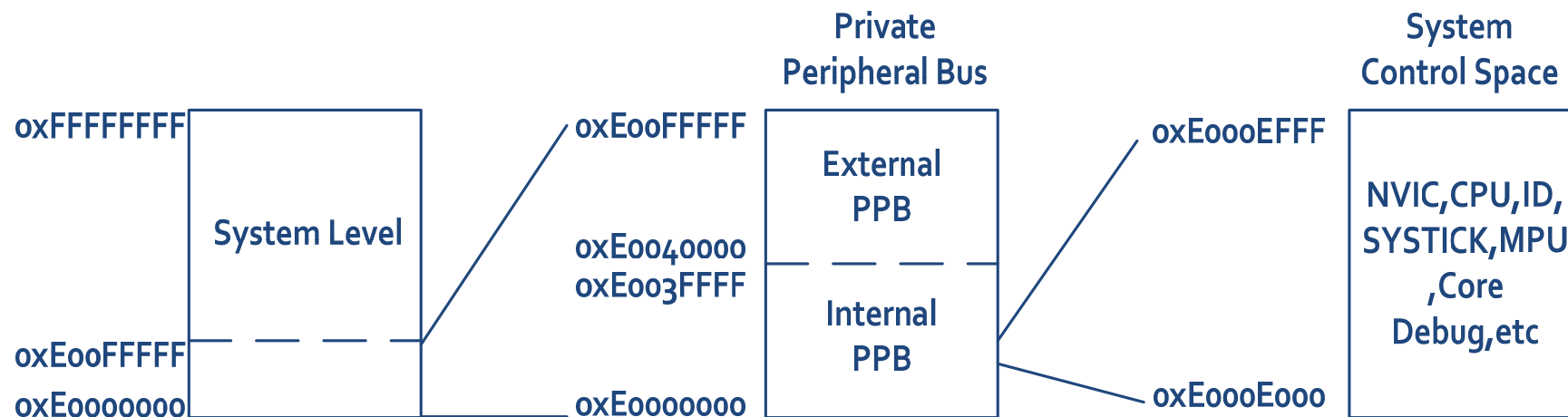


8.4 Cortex-M3 存储系统

私有外设总线:

1. AHB私有外设总线, 只用于Cortex-M3内部AHB 外设, 分别是: NVIC, FPB, DWT和ITM。
2. APB私有外设总线, 针对 Cortex-M3内部APB设备和外围设备。

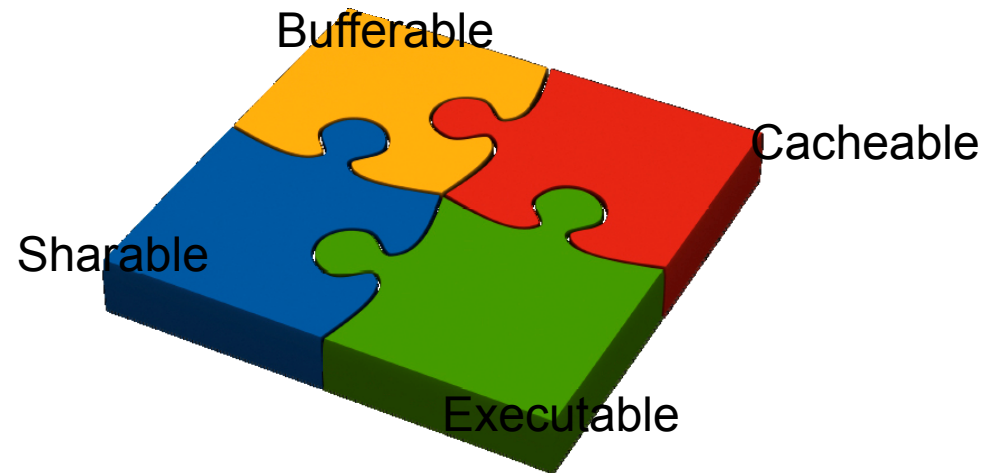
系统控制空间(SCS)



8.4 Cortex-M3 存储系统

3. 内存访问属性

存储器映射定义了内存的访问属性：



如果配了MPU，则可以通过它配置不同的存储区，并且覆盖缺省的访问属性。



8.4 Cortex-M3 存储系统

地址空间划分和属性

区域	地址	用于	可缓存	可执行, 缓冲
代码存储区	0x00000000–0x1FFFFFFF	执行指令	WT (直写)	可执行, 缓冲
SRAM 存储区域	0x20000000–0x3FFFFFFF	片上RAM	WB-WA (写回, 写分配)	可执行, 缓冲
片上外设	0x40000000–0x5FFFFFFF	外设	非缓存	非缓冲, 不可执行
外部 RAM 区域 1	0x60000000–0x7FFFFFFF	片上或片外存储器	WB-WA (写回, 写分配)	可执行
外部 RAM 区域 2	0x80000000–0x9FFFFFFF	片上或片外存储器	WT (直写)	可执行
外部设备1	0xA0000000–0xBFFFFFFF	外部设备和/或共享内存		不可执行, 非缓冲
外部设备2	0xC0000000–0xDFFFFFFF	外部设备和/或共享内存		不可执行, 非缓冲
系统区域	0xE0000000–0xFFFFFFFF	私有外设	非缓存	不可执行, 非缓冲
系统区域	0xE0000000–0xFFFFFFFF	供应商特定设备	缓存	不可执行, 缓冲

8.4 Cortex-M3 存储系统

4. 默认内存访问权限

CM3有一个缺省的存储访问许可，它能防止使用户代码访问系统控制存储空间，保护NVIC、MPU等关键部件。缺省访问许可在下列条件时生效：

- 1) 没有配备MPU;
- 2) 配备了MPU，但是MPU被禁用.

如果启用了MPU，则MPU 可以在地址空间划出若干区域，并为每个区规定不同的访问许可权限。

缺省的存储器访问许可权限如表所示



默认的内存访问许可

内存区域	地址	用户级许可权限
代码区	0x00000000 – 0x1FFFFFFF	无限制
片内RAM	0x20000000 – 0x3FFFFFFF	无限制
片上外设	0x40000000 – 0x5FFFFFFF	无限制
外部RAM	0x60000000 – 0x9FFFFFFF	无限制
外部外设	0xA0000000 – 0xDFFFFFFF	无限制
ITM	0xE0000000 – 0xE0000FFF	可以读。对于写操作，除了用户级下允许时的 stimulus 端口外，全部忽略
DWT	0xE0001000 – 0xE0001FFF	阻止访问，访问会引发一个总线fault
FPB	0xE0002000 – 0xE0003FFF	阻止访问，访问会引发一个总线fault
NVIC	0xE000E000 – 0xE000EFFF	阻止访问，访问会引发一个总线fault。例外：软件触发中断寄存器可以被编程为允许用户级访问
内部FPB	0xE000F000 – 0xE003FFFF	阻止访问，访问会引发一个总线fault
TPIU	0xE0040000 – 0xE0040FFF	阻止访问，访问会引发一个总线fault
ETM	0xE0041000 – 0xE0041FFF	阻止访问，访问会引发一个总线fault
外部FPB	0xE0042000 – 0xE0042FFF	阻止访问，访问会引发一个总线fault
ROM表	0xE00FF000 – 0xE00FFFFFFF	阻止访问，访问会引发一个总线fault
供应商指定	0xE0100000 – 0xFFFFFFFF	无限制



8.4 Cortex-M3 存储系统

5. Bit-Band操作

Bit-band操作支持一个单一的加载/存储指令访问(读/写)单个数据位。

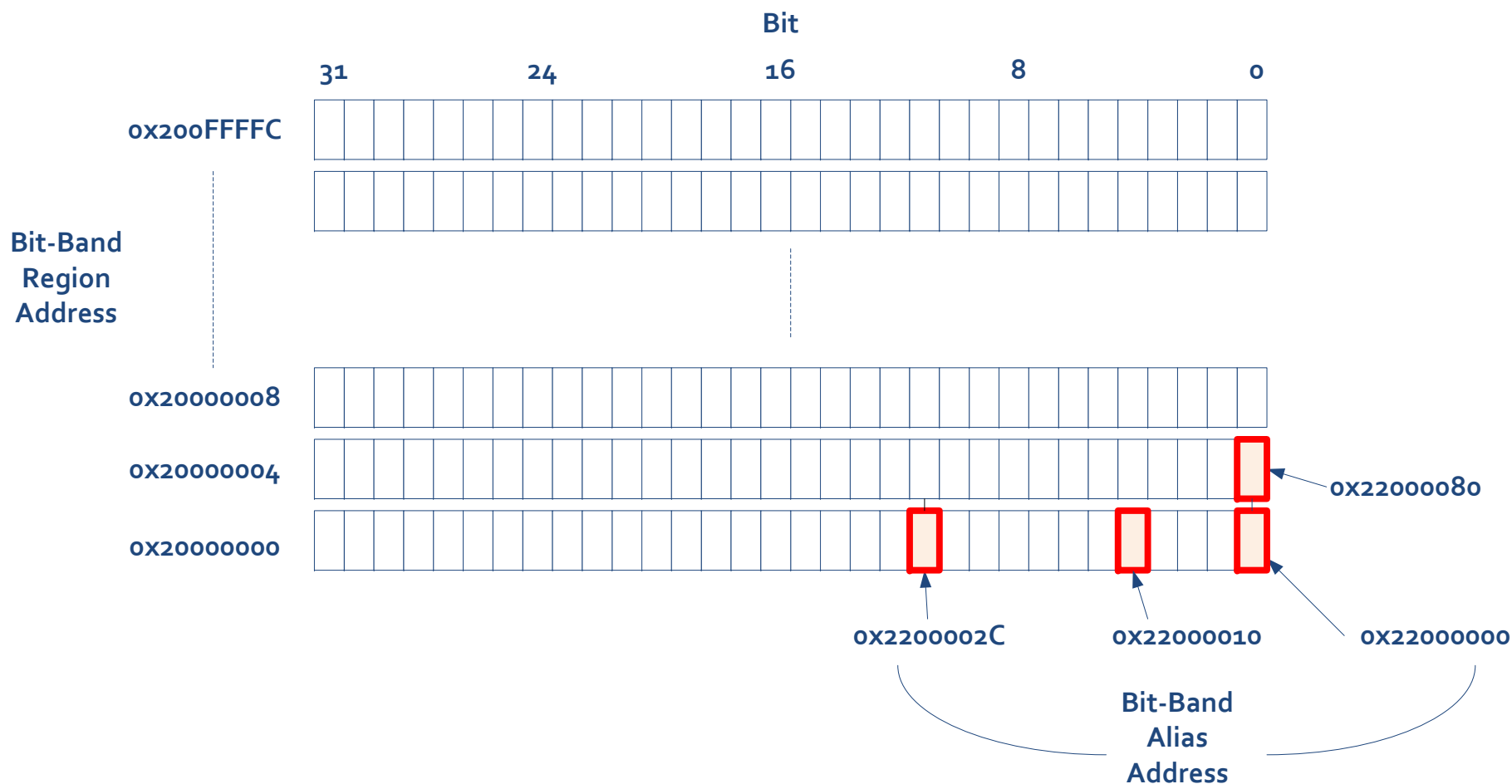
Bit-band区域:

1. SRAM 的第一个1MB范围;
2. 片内外设区的第一个1MB范围。

这两个位带中的地址除了可以像普通的RAM一样使用外，它们还都有自己的“***bit-band alias***”，位带别名区把每个比特扩展成一个32位的字。当通过位带别名区访问这些字时，就可以达到访问原始比特的目的。



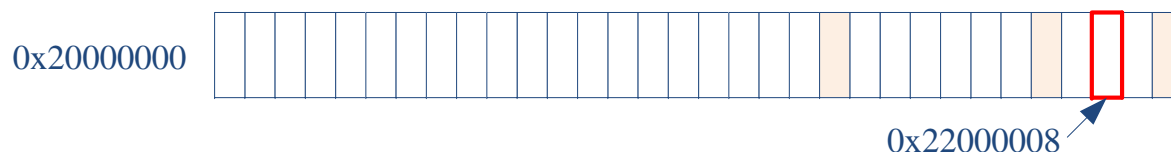
8.4 Cortex-M3 存储系统



位通过Bit-Band Alias访问Bit-Band Region



8.4 Cortex-M3 存储系统



设置地址0x20000000的字数据的比特2:

写操作:

1. 不使用Bit-Band:

```
LDR    R0,    =0x20000000    ; Setup address
LDR    R1,    [R0]            ; Read
ORR.W  R1,    #0x4            ; Modify bit
STR    R1,    [R0]            ; Write back result
```

2. 使用Bit-Band:

```
LDR    R0,    =0x22000008    ; Setup address
MOV    R1,    #1              ; Setup data
STR    R1,    [R0]            ; Write
```



8.4 Cortex-M3 存储系统

读操作:

1. 不使用 Bit-Band:

```
LDR      R0,    =0x20000000    ; Setup address
LDR      R1,    [R0]           ; Read
UBFX.W   R1,    R1,    #2,    #1    ; Extract bit[2]
```

2. 使用 Bit-Band:

```
LDR      R0,    =0x22000008    ; Setup address
LDR      R1,    [R0]           ; Read
```



8.4 Cortex-M3 存储系统

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]

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]



8.4 Cortex-M3 存储系统

6. 非对齐数据传送

CM3支持在单一的访问中使用非（地址）对齐的传送，数据存储器的访问无需对齐。

之前的ARM处理器只允许对齐的数据传送。这种对齐是说：以字为单位的传送，其地址的最低两位必须是0；以半字为单位的传送，其地址的LSB必须是0；以字节为单位的传送则无所谓对不对齐。



8.4 Cortex-M3 存储系统

1. 字长, 地址不是4的倍数。

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				[31:24]
Address N	[23:16]	[15:8]	[7:0]	

未对齐传递例子1

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4			[31:24]	[23:16]
Address N	[15:8]	[7:0]		

未对齐传递例子2

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4		[31:24]	[23:16]	[15:8]
Address N	[7:0]			

未对齐传递例子3



8.4 Cortex-M3 存储系统

2. 半字长, 其地址不是2的倍数

	Byte 3	Byte 2	Byte1	Byte 0
Address N+4				
Address N		[15:8]	[7:0]	

未对齐传递例子4

	Byte 3	Byte 2	Byte 1	Byte 0
Address N+4				[15:8]
Address N	[7:0]			

未对齐传递例子5



8.4 Cortex-M3 存储系统

在CM3中，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). 不支持多个数据的载入/存储指令 (LDM/STM)。
- 2). 堆栈操作 (PUSH/POP) 必须一致。
- 3). 互斥访问必须对齐。
- 4). 非对齐传递在bit-band 操作中不支持。

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。



8.4 Cortex-M3 存储系统

7. 互斥访问

SWP 指令 (swap) 用于在传统的ARM处理器上进行信号灯操作。

1. 什么是信号灯?

信号灯常用来对应用分配共享资源。当一个资源正在被一个过程使用时它被这个过程锁定。

2. 为什么使用互斥访问而不使用SWP指令?

当一个过程或应用需要使用某个资源, 它首先需要检查是否这个资源被锁定。如果它不能被使用, 可以表明该资源已被锁定。



8.4 Cortex-M3 存储系统

在 ARM V7架构中，存储器读和存储器写使用独立的总线。因此，SWP指令使用时不能再对存储区域进行原子访问。

因此，锁定的传输被互斥访问所取代。

3. SWP 指令和互斥访问的区别。

互斥访问操作的概念非常简单，但是与SWP不同；它允许一个信号灯的存储位置被另一个总线主机访问读而不是写。



8.4 Cortex-M3 存储系统

互斥访问指令包括:

1. LDREX (字)
2. LDREXB (字节)
3. LDREXH (半字)
4. STREX (字)
5. STREXB (字节)
6. and STREXH (半字)



LDREX/STREX的例子:

LDREX <Rxf>, [Rn, #offset]

STREX <Rd>, <Rxf> ,[Rn, #offset]



8.4 Cortex-M3 存储系统

8. 数据端模式

Cortex-M3 支持小端模式和大端模式。

在 Cortex-M3 中大端模式的定义和 ARM7 中不同。

在 ARM7TDMI 中,大端模式被称为“字不变大端”,然而在 Cortex-M3 中,大端模式被称为字节不变大端。

在 AHB 总线上 BE-8 模式下的数据传送使用和小端模式中一样的字节通道。



8.4 Cortex-M3 存储系统

Cortex-M3 (字节不变大端模式): 存储器上的数据

Address, Size	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x1000, 字	Data[7:0]	Data[15:8]	Data[23:16]	Data[31:24]
0x1000, 半字	Data[7:0]	Data[15:8]		
0x1002, 半字	Data[7:0]	Data[15:8]		
0x1000, 字节	Data[7:0]			
0x1001, 字节		Data[7:0]		
0x1002, 字节			Data[7:0]	
0x1003, 字节				Data[7:0]



8.4 Cortex-M3 存储系统

Cortex-M3 (字节不变大端模式): AHB 总线上的数据

Address, Size	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x1000, 字	Data[7:0]	Data[15:8]	Data[23:16]	Data[31:24]
0x1000, 半字			Data[7:0]	Data[15:8]
0x1002, 半字	Data[7:0]	Data[15:8]		
0x1000, 字节				Data[7:0]
0x1001, 字节			Data[7:0]	
0x1002, 字节		Data[7:0]		
0x1003, 字节	Data[7:0]			



8.4 Cortex-M3 存储系统

在AHB总线上BE-8模式下的数据传送使用和小端模式中一样的数据字节通道。在字或半字中的字节数据是逆向排序的。

ARM7 (字不变大端模式): AHB 总线上的数据

Address, Size	Bits 31-24	Bits 23-16	Bits 15-8	Bits 7-0
0x1000, 字	Data[7:0]	Data[15:8]	Data[23:16]	Data[31:24]
0x1000, 半字	Data[7:0]	Data[15:8]		
0x1002, 半字			Data[7:0]	Data[15:8]
0x1000, 字节	Data[7:0]			
0x1001, 字节		Data[7:0]		
0x1002, 字节			Data[7:0]	
0x1003, 字节				Data[7:0]



8.4 Cortex-M3 存储系统

数据端模式在处理器退出复位后被设置并且以后不能再被改变。

取指令总是使用小端模式, 另外: 对配置控制存储空间和外部PPB存储范围的数据访问也使用小端模式。

当你的设计不支持大端模式, 却有一些外设包含了大端模式时, 可以轻易地使用REV/REVH指令来完成端模式的转换。



8.5 Cortex-M3 指令集

1. 汇编基础

1) 汇编语言：基本语法

CM3典型的汇编指令格式：

标号

操作码 操作数1, 操作数2, ... ;注释

标号：用来让汇编器来计算程序转移的地址。

操作码：指令的助记符。

操作数：第1个操作数通常都给出本指令的执行结果存储处；
不同指令需要不同数目的操作数，并且对操作数的语法要求也可以不同。



8.5 Cortex-M3 指令集

2. 指令列表

16-Bit 数据处理指令

指令	功能
ADC	带进位加法
ADD	加法
ASR	算术右移
AND	按位与
BIC	按位清零 (一个值与另一个值的逻辑反转进行逻辑与)
CMN	负比较 (将一个数据和另一个数据的二进制补码进行比较, 同时更新标志)
CMP	比较 (比较两个数据并且更新标志)
CPY	拷贝 (从架构v6开始可用; 把一个高或低寄存器中的值移植到另一个高或低寄存器)
EOR	异或(Exclusive OR)
LSL	逻辑左移
LSR	逻辑右移
MOV	移动 (可用于寄存器至寄存器的传输或加载立即数据)
MUL	乘法



8.5 Cortex-M3 指令集

(Continued)

MVN	Move NOT (获得逻辑反的值)
NEG	取反 (获得二进制补码)
ORR	逻辑或
ROR	Rotate right
SBC	带借位减法
SUB	减法
TST	测试(被用作逻辑 与; Z 标志会更新但是与的结果不会被储存)
REV	反转在一个32-位寄存器中的字节顺序 (从架构v6开始可用)/REVH/REVSH
SXTB	带符号扩展一个字节到32位(从架构v6开始可用)
SXTH	带符号扩展一个半字到32位(从架构v6开始可用)
UXTB	无符号扩展一个字节到32位(a从架构v6开始可用)
UXTH	无符号扩展一个半字到32位(从架构v6开始可用)



8.5 Cortex-M3 指令集

16-Bit 转移指令

指令	功能
B	无条件转移
B<cond>	条件转移
BL	带连接的转移; 调用一个子程序, 并在LR中存储返回地址
BLX	带连接的分支和改变状态 (只有BLX <reg>)
CBZ	比较, 如果为0就转移(架构 v7)
CBNZ	比较, 如果非0就转移(架构 v7)
IT	IF-THEN (架构 v7)



8.5 Cortex-M3 指令集

16-Bit 加载和存储指令

指令	功能
LDR	从存储器中加载字到寄存器中
LDRH	从存储器中加载半字到寄存器中
LDRB	从存储器中加载字节到寄存器中
LDRSH	从存储器中加载半字，再经带符号扩展后存储到寄存器中
LDRSB	从存储器中加载字节，再经带符号扩展后存储到寄存器中
STR	把一个寄存器按字存储到存储器中
STRH	把一个寄存器的低半字存储到存储器中
STRB	把一个寄存器的低字节存储到存储器中
LDMIA	加载多个字，并且在加载后自增基址寄存器
STMIA	存储多个字，并且在存储后自增基址寄存器
PUSH	压入多个寄存器到堆栈中
POP	从堆栈中弹出多个值到寄存器中



8.5 Cortex-M3 指令集

其它 16-Bit 指令

指令	功能
SVC	系统服务调用
BKPT	断点指令; 如果启用了调试, 会进入调试模式 (停机), 或者如果启用了调试监控异常, 将调用调试异常; 否则会调用一个错误异常
NOP	无操作
CPSIE	启用 PRIMASK (CPSIE i)/FAULTMASK (CPSIE f) 寄存器 (设置寄存器为0)
CPSID	禁用 PRIMASK (CPSID i)/ FAULTMASK (CPSID f) 寄存器 (设置寄存器为1)



8.5 Cortex-M3 指令集

32-Bit 数据处理指令

指令	功能
ADC	带进位加法
ADD	加法
ADDW	宽加法(#immed_12)
AND	按位与
ASR	算术右移
BIC	位清零 (把一个数按位取反后, 与另一个数逻辑与)
BFC	位域清零
CMP	比较 (比较两个数据并更新标志)
CLZ	计算前导0的数目
EOR	按位异或
LSL	逻辑左移
LSR	逻辑右移
MLA	乘法累加



8.5 Cortex-M3 指令集

(Continued)

MSL	乘和减
MOVW	把16位立即数加载到寄存器的低16位, 高16位清0
MOV	加载16位立即数到寄存器(编译器会产生MOVW)
MOVT	把16位立即数加载到寄存器的高16位, 低16位不受影响
MVN	传送一个数的补码
MUL	乘法
ORR	按位或
ORN	逻辑或非(把源操作数按位取反后, 再执行按位或)
RBIT	位反转
REV	对一个32位整数按字节反转.
ROR	循环右移
RSB	反向减法
RRX	带扩展的循环右移
SBFX	从一个32位整数中提取任意长度和位置的位段, 并且带符号扩展成32位整数
SDIV	带符号除法



8.5 Cortex-M3 指令集

(Continued)

SMLAL	带符号长乘加
SMULL	带符号长乘
SSAT	带符号饱和运算
SBC	带借位减法
SUB	减法
SUBW	宽减法(#immed_12)
SXTB	带符号字节扩展到32位数
TEQ	测试是否相等(被用作逻辑异或; 标志会更新,但结果不会存储)
TST	测试(被用作逻辑与; Z 标志会更新但与的结果不会被存储)
UBFX	无符号位域提取
UDIV	无符号除法
USAT	无符号饱和操作
UXTB	无符号扩展字节至32位
UXTH	无符号扩展半字至32位



8.5 Cortex-M3 指令集

32-Bit 加载和存储指令

指令	功能
LDR	从存储器向寄存器加载字
LDRB	从存储器向寄存器加载字节
LDRH	从存储器向寄存器加载半字
LDRSB	从存储器中取字节，对它进行符号扩展，把它存入寄存器
LDRSH	从存储器中取半字，对它进行符号扩展，把它存入寄存器
LDM	从连续的存储器空间向多个寄存器加载字
LDRD	从连续的存储器空间向 2 个寄存器加载双字
STR	把寄存器中的字存储到存储器中
STRB	把寄存器中的低字节存储到存储器中
STRH	把寄存器中的低半字存储到存储器中
STM	从多个寄存器向存储器存储多个字
STRD	从 2 个寄存器向存储器存储双字数据
PUSH	Push 多个寄存器
POP	Pop 多个寄存器



8.5 Cortex-M3 指令集

32-Bit 转移指令

指令	功能
B	无条件转移
BL	转移并连接(调用子程序)
TBB	字节转移表。使用单字节偏移表向前转移
TBH	半字转移表。使用半字偏移表向前转移

其它 32-Bit 指令

指令	功能
LDREX	独占加载字到寄存器,并且在内核中标明一段地址进入了互斥访问.
STREX	独占存储字到存储器



8.5 Cortex-M3 指令集

(Continued)

CLREX	清除本地处理器的独占访问记录
MRS	加载专用寄存器的内容到通用寄存器
MSR	加载通用寄存器的内容到专用寄存器
NOP	无操作
SEV	发送事件
WFE	休眠并且在发生事件时被唤醒
WFI	休眠并且在发生中断时被唤醒
ISB	指令同步隔离
DSB	数据同步隔离
DMB	数据存储器隔离



8.5 Cortex-M3 指令集

3 指令描述

3.1 汇编语言: 传送数据

数据传输可以是下列类型之一:



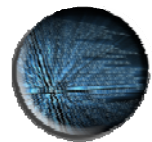
1. 在寄存器和寄存器间传送数据



2. 在寄存器和存储器间传送数据



3. 在寄存器和特殊功能寄存器间传送数据



4. 把一个立即数传送给寄存器



8.5 Cortex-M3 指令集

访问存储器的基本指令是加载(Load)和存储(Store)。

常用的内存访问指令

例子	描述
LDRB Rd, [Rn, #offset]	从地址Rn + offset 读取一个字节加载到Rd
LDRH Rd, [Rn, #offset]	从地址Rn + offset 读取一个半字加载到Rd
LDR Rd, [Rn, #offset]	从地址Rn + offset 读取一个字加载到Rd
LDRD Rd1,Rd2, [Rn, #offset]	从地址Rn + offset 读取一个双字加载到Rd1,Rd2
STRB Rd, [Rn, #offset]	把Rd中的低字节存储到地址Rn+offset处
STRH Rd, [Rn, #offset]	把Rd中的半字存储到地址Rn+offset处
STR Rd, [Rn, #offset]	把Rd中的字存储到地址Rn+offset处
STRD Rd1,Rd2, [Rn, #offset]	把Rd1,Rd2中的双字存储到地址Rn+offset处



8.5 Cortex-M3 指令集

多个加载和存储操作:

LDM (加载多个); STM (存储多个).

常用多重存储器访问方式

例子	描述
LDMIA Rd!, <reg list>	从Rd处读取多个字, 并依次送到寄存器列表中的寄存器。每读一个字后Rd自增一次, 16位宽度。
STMIA Rd!, <reg list>	依次将寄存器列表中各寄存器的值存储到Rd指定的地址。每存储一个字后Rd自增一次, 16位宽度。
LDMIA.W Rd(!), <reg list>	从Rd处读取多个字, 并依次送到寄存器列表中的寄存器。每读一个字后Rd自增一次, 32位宽度。
LDMDB.W Rd(!), <reg list>	从Rd处读取多个字, 并依次送到寄存器列表中的寄存器。每读一个字后Rd自减一次, 32位宽度。
STMIA.W Rd(!), <reg list>	依次将寄存器列表中各寄存器的值存储到Rd指定的地址。每存储一个字后Rd自增一次, 32位宽度。
STMDB.W Rd(!), <reg list>	依次将寄存器列表中各寄存器的值存储到Rd指定的地址。每存储一个字后Rd自减依次, 32位宽度。

8.5 Cortex-M3 指令集

通常,一条PUSH指令中的寄存器列表,与相应的POP指令中的寄存器列表相同,但是这并不总是必须的。

```
PUSH    {R0-R3, LR}    ; Save register contents at
                        ; beginning of subroutine
....
                        ; Processing
POP      {R0-R3, PC}    ; restore registers and return
```

传送立即数到寄存器:

```
MOV R0, #0x12    ; Set R0 to 0x12
```

对于大的取值(超过 8 比特),你可能需要使用一条Thumb-2 传送指令。

```
MOVW.W   R0,#0x789A ; Set R0 to 0x789A
```



8.5 Cortex-M3 指令集

也可以使用LDR (在ARM汇编中使用的一条伪指令):

```
LDR R0, =0x3456789A
```

这并不是一条真正的汇编命令,但是ARM的汇编器会把它转换成一条PC相关的加载指令来产生需要的数据。

对于 LDR, 如果地址是一个程序地址值, 它会自动把LSB为设为1。

```
LDR R0, =address1 ; R0 set to 0x4001
```

...

address1

```
0x4000: MOV R0, R1 ;address1 contains program code
```

...



8.5 Cortex-M3 指令集

3.2 汇编语言：数据处理

很多数据操作指令有多种指令格式。

例：ADD

ADD R0, R1 ; $R0 = R0 + R1$

ADD R0, #0x12 ; $R0 = R0 + 0x12$

ADD.W R0, R1, R2 ; $R0 = R1 + R2$

当代码中使用16位指令时，ADD指令改变PSR的相应标志位。

ADD.W R0, R1, R2 ; Flag unchanged

ADDS.W R0, R1, R2 ; Flag change



8.5 Cortex-M3 指令集

算术指令的例子

指 令	操 作
ADD Rd, Rn, Rm ; $Rd = Rn + Rm$ ADD Rd, Rm ; $Rd = Rd + Rm$ ADD Rd, #immed ; $Rd = Rd + \#immed$	加法
ADC Rd, Rn, Rm ; $Rd = Rn + Rm + \text{carry}$ ADC Rd, Rm ; $Rd = Rd + Rm + \text{carry}$ ADC Rd, #immed ; $Rd = Rd + \#immed + \text{carry}$	带进位的加法
ADDW Rd, Rn, #immed ; $Rd = Rn + \#immed$	寄存器与12位立即数相加
SUB Rd, Rn, Rm ; $Rd = Rn - Rm$ SUB Rd, #immed ; $Rd = Rd - \#immed$ SUB Rd, Rn, #immed ; $Rd = Rn - \#immed$	减法
RSB.W Rd, Rn, #immed ; $Rd = \#immed - Rn$ RSB.W Rd, Rn, Rm ; $Rd = Rm - Rn$ MUL Rd, Rm ; $Rd = Rd * Rm$ Multiply MUL.W Rd, Rn, Rm ; $Rd = Rn * Rm$	反向减法
UDIV Rd, Rn, Rm ; $Rd = Rn / Rm$ SDIV Rd, Rn, Rm ; $Rd = Rn / Rm$	无符号、有符号除法



8.5 Cortex-M3 指令集

3.3 汇编语言：调用和非条件跳转

最基本的分支指令如下：

B label ; 跳转到label所指向的分支

BX reg ; 跳转到reg寄存器所指向的地址

在BX指令中，寄存器中LSB的值决定了处理器的下一个状态：是ARM (LSB=0)呢，还是Thumb (LSB=1)。既然CM3只在Thumb中运行，就必须保证reg的LSB=1，否则出错。

调用一个函数：

BL label ; 跳转到label指向的地址并且在LR中保存返回地址

BLX reg ; 跳转到reg寄存器所指向的地址并且
;在LR中保存返回地址.



8.5 Cortex-M3 指令集

3.4 汇编语言：判断和条件跳转

在APSR中，有五个标志位；其中四个用作条件分支选择。

APSR中可以影响条件转移的**4**个标志位

标志	PSR 位	描述
N	31	Negative flag (上次运算结果是否为负标志)
Z	30	Zero (上次运算结果是否为0标志)
C	29	Carry (进位或借位标志)
V	28	Overflow (运算结果越界标志)



8.5 Cortex-M3 指令集

通过4个标志的组合(*N*, *Z*, *C*, *V*), 一共定义了15个条件跳转

跳转及条件执行判据

符号	条件	标志
EQ	Equal	Z set
NE	Not equal	Z clear
CS/HS	Carry set/unsigned higher or same	C set
CC/LO	Carry clear/unsigned lower	C clear
MI	Minus/negative	N set
PL	Plus/positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set



8.5 Cortex-M3 指令集

(续)

LT	Signed less than	N set and V clear, or N clear and V set ($N \neq V$)
GT	Signed greater than	Z clear, and either N set and V set, or N clear and V clear ($Z == 0, N == V$)
LE	Signed less than or equal	Z set, or N set and V clear, or N clear and V set ($Z == 1$ or $N \neq V$)
AL	Always (unconditional)	—

例:

BEQ label ; 如果Z标志位为1, 则跳转到地址'label'

Thumb-2 version

BEQ.W label ; 如果Z标志位为1, 则跳转到地址'label'



8.5 Cortex-M3 指令集

4 Cortex-M3中一些有用的指令

4.1 MRS 和MSR

访问一些特殊寄存器

MRS <Rn>, <SReg> ; 从特殊寄存器读出

MSR <SReg>, <Rn> ; 向特殊寄存器写入

MRS和MSR可以访问的特殊寄存器名

符 号	描 述
IPSR	中断状态寄存器
EPSR	执行状态寄存器
APSR	以前操作标志
IEPSR	IPSR和EPSR组合
IAPSR	IPSR和APSR组合



8.5 Cortex-M3 指令集

(续)

EAPSR	EPSR和APSR组合
PSR	EPSR、IPSR和APSR组合
MSP	主堆栈指针
PSP	进程堆栈指针
PRIMASK	正常的异常屏蔽寄存器
BASEPRI	正常的异常优先级屏蔽寄存器
BASEPRI_MAX	带条件写的正常的异常优先级屏蔽寄存器 (新的优先级必须高于旧的优先级)
FAULTMASK	错误异常屏蔽寄存器(同时屏蔽正常的中断)
CONTROL	控制寄存器



Example:

LDR R0, =0x20008000 ; 为程序堆栈指针赋新值(PSP)
MSR PSP, R0



第八章 Cortex-M3组成和结构



END

