



苏州大学

SOOCHOW UNIVERSITY



# 第2章 ARM Cortex-M4F处理器

**2.1**

**ARM Cortex-M4F处理器简介**

**2.2**

**指令系统**

**2.3**

**指令集与机器码对应表**

**2.4**

**汇编语言的基本语法**



苏州大学

SOOCHOW UNIVERSITY



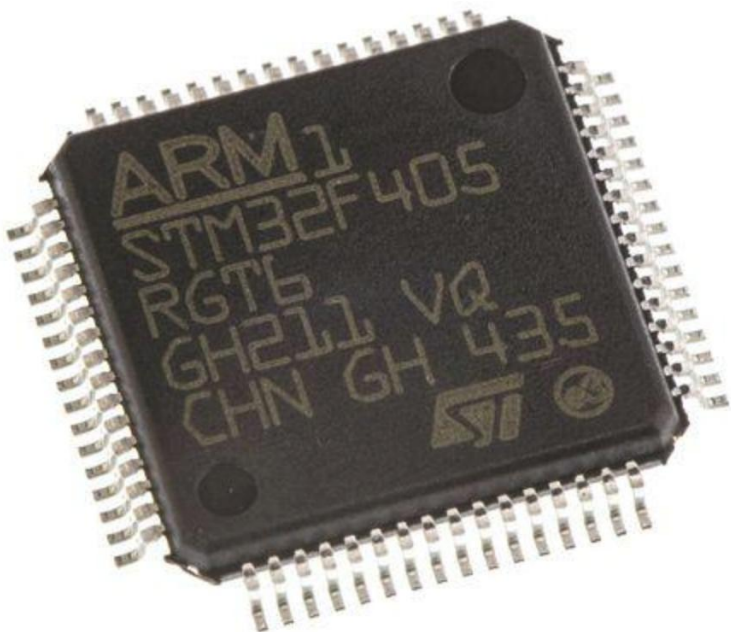
## 2.1 ARM Cortex-M4F处理器简介





## 2.1.1

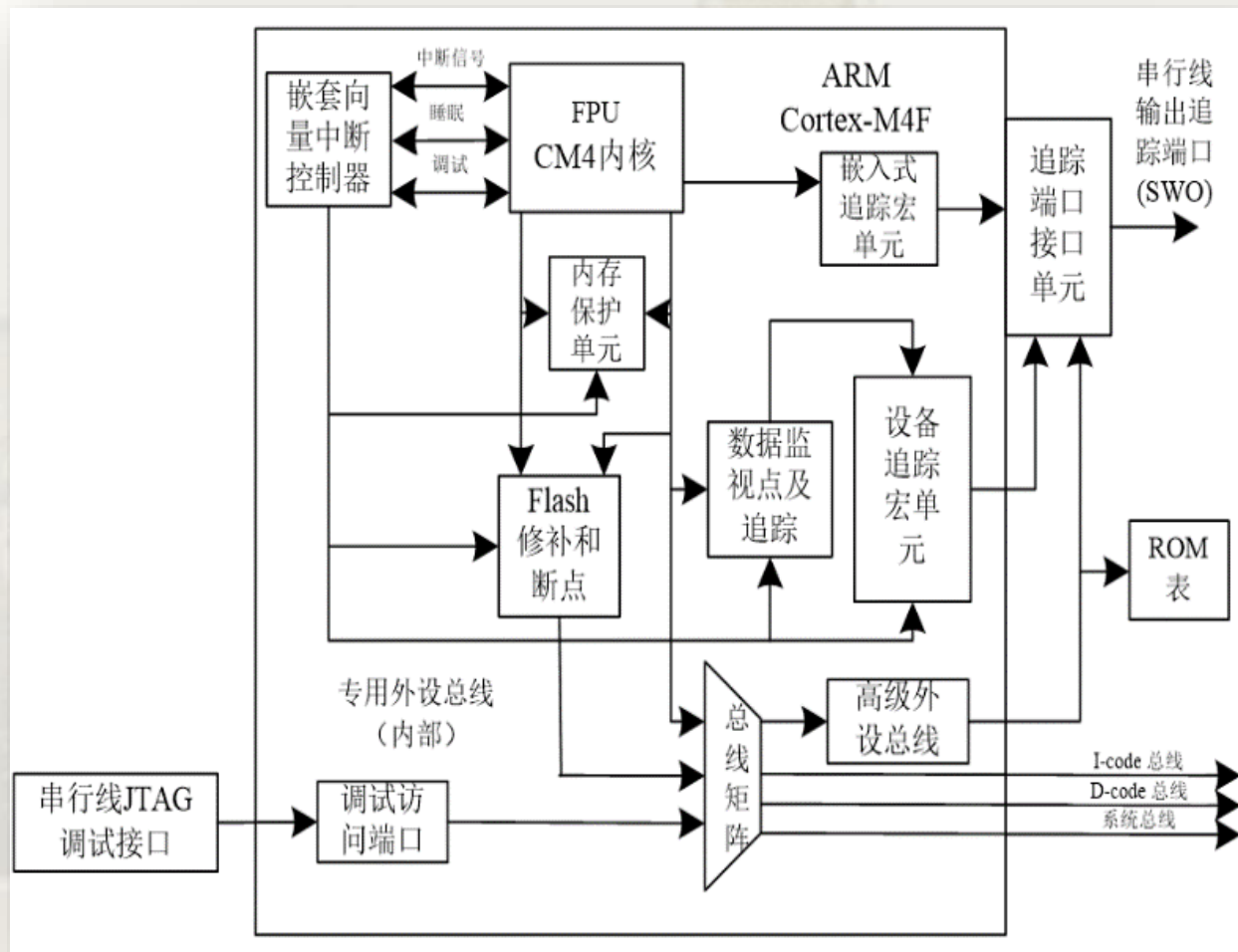
### ARM Cortex-M4F处理器内部结构概要——简介



- ARM Cortex-M4F是一种低功耗、高性能、高速度的处理器。
- 硬件方面支持除法指令，并且有着中断处理程序和线程两种模式。
- 在处理中断方面，M4F具有着自动保存处理器状态和回复低延迟中断。
- M4内核的定点运算的速度是M3内核的两倍，而浮点运算速度比M3内核快十倍以上，同时功耗只有一半。



## 2.1.1 ARM Cortex-M4F处理器内部结构概要——M4F内核

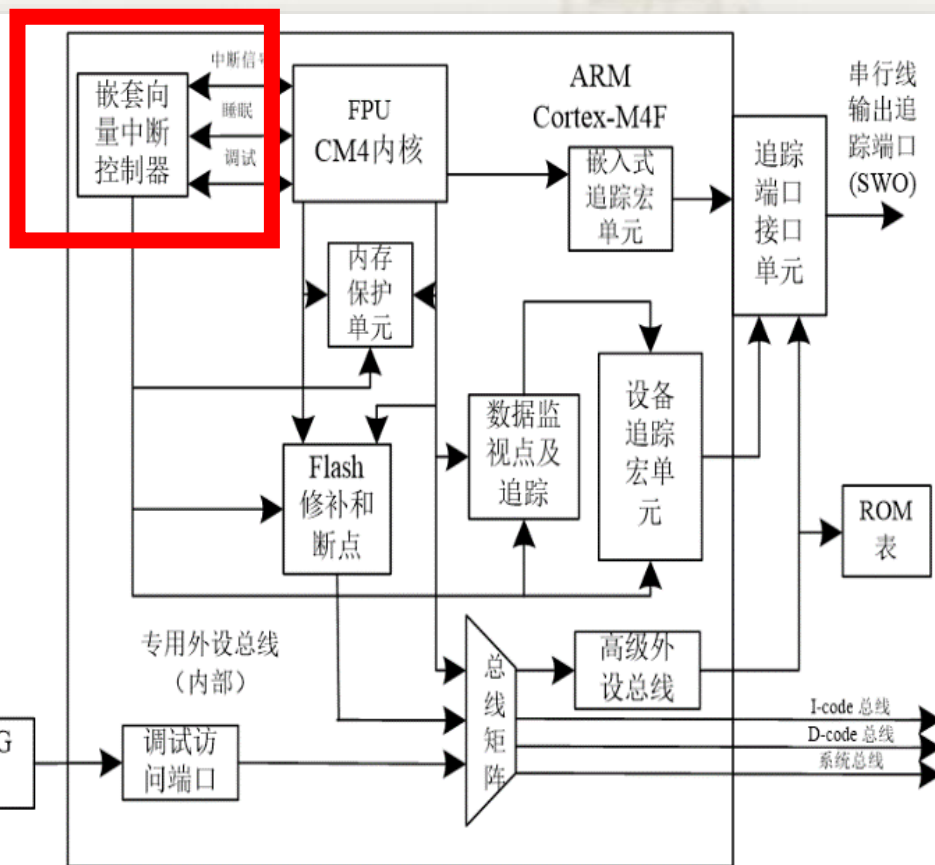


- 嵌套中断向量控制器
- 存储器保护单元
- 调试解决方案
- 总线接口
- 浮点运算单元



## 2.1.1 ARM Cortex-M4F处理器内部结构概要——嵌套中断向量控制器

**嵌套中断向量控制器 (Nested Vectored Interrupt Controller, NVIC)** 是一个在 Cortex M4F中内建的中断控制器。



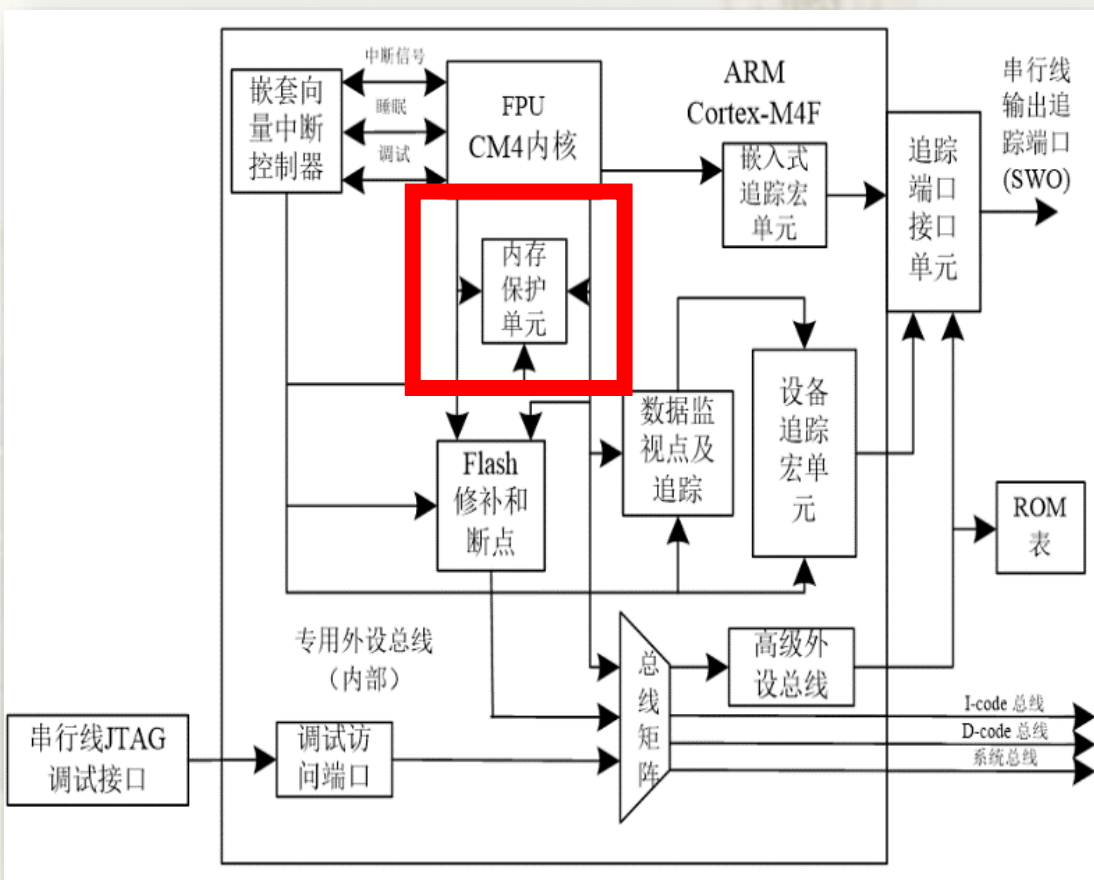
- 在MSP432系列芯片中，中断源数为64个，优先等级可配置范围为0-7，还可对优先等级进行分组。
- 为优化低功耗设计，在芯片可快速进入超低功耗状态，且在超低功耗睡眠模式下可唤醒。
- 包含一个24位倒计时定时器SysTick，即使系统在睡眠模式下也能工作，也为在RTOS同类内核芯片间移植带来便利。





## 2.1.1 ARM Cortex-M4F处理器内部结构概要——存储器保护单元

**存储器保护单元 (Memory Protection Unit , MPU ) 是指可以对一个选定的内存单元进行保护。**

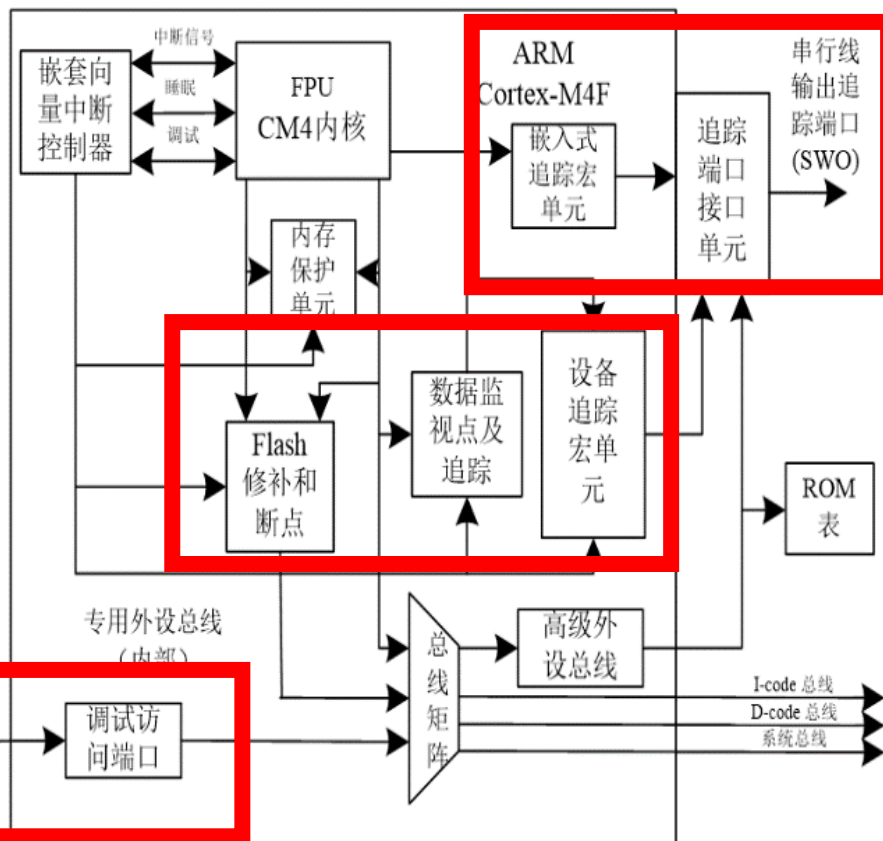


- 它将存储器划分为8个子区域。
- 子区域的优先级均是可自定义的。
- 处理器可以使指定的区域禁用和使能。



## 2.1.1 ARM Cortex-M4F处理器内部结构概要——调试解决方案

### 对存储器和寄存器进行调试访问。

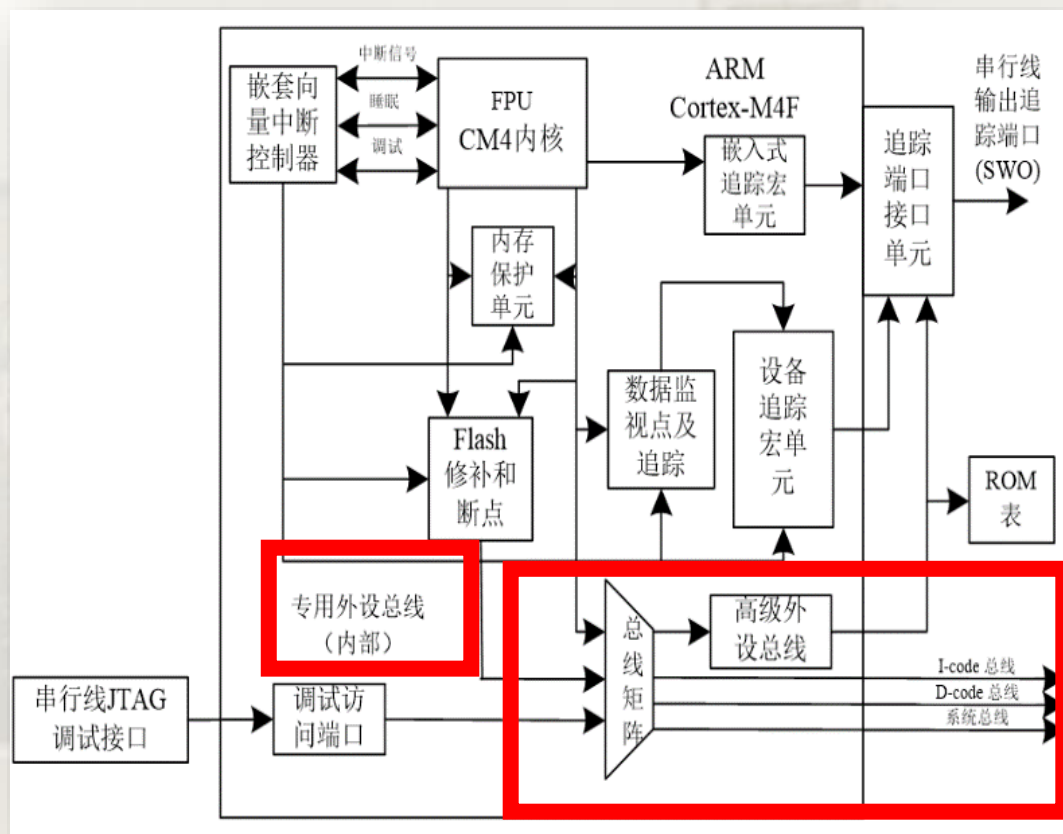


- 具有SWD或JTAG调试访问接口，或两种都包括。
- Flash 修补和断点单元用于实现硬件断点和代码修补。
- 数据观察点和触发单元用于实现观察点、触发资源和系统分析。
- 指令跟踪宏单元用于提供对printf()类型调试的支持。
- 跟踪端口接口单元用来连接跟踪端口分析仪，包括单线输出模式。



## 2.1.1 ARM Cortex-M4F处理器内部结构概要——总线接口

ARM Cortex-M4F处理器提供先进的高性能总线（AHB-Lite）接口，包括的4个接口分别为：**ICode存储器接口**，**DCode存储器接口**和**系统接口**，还有**基于高性能外设总线（ASB）的外部专用外设总线（PPB）**



- 位段的操作可以细化到原子位段的读写操作。
- 对内存的访问是对齐的。
- 写数据时采用写缓冲区的方式。





## 2.1.1 ARM Cortex-M4F处理器内部结构概要——浮点运算单元

- 处理器可以处理单精度32位指令数据。
- 结合了乘法和累积指令用来提高计算的精度。
- 硬件能够进行加减法、乘除法以及平方根等运算操作，同时也支持所有的IEEE数据四舍五入模式。
- 拥有32个专用32位单精度寄存器，也可作为16个双字寄存器寻址。
- 通过采用解耦三级流水线来加快处理器运行速度。



## 2.1.2 ARM Cortex-M4F处理器存储器映像

**存储器映像是指：把这4GB空间当做存储器来看待，分成若干区间，都可安排实际的物理资源。**

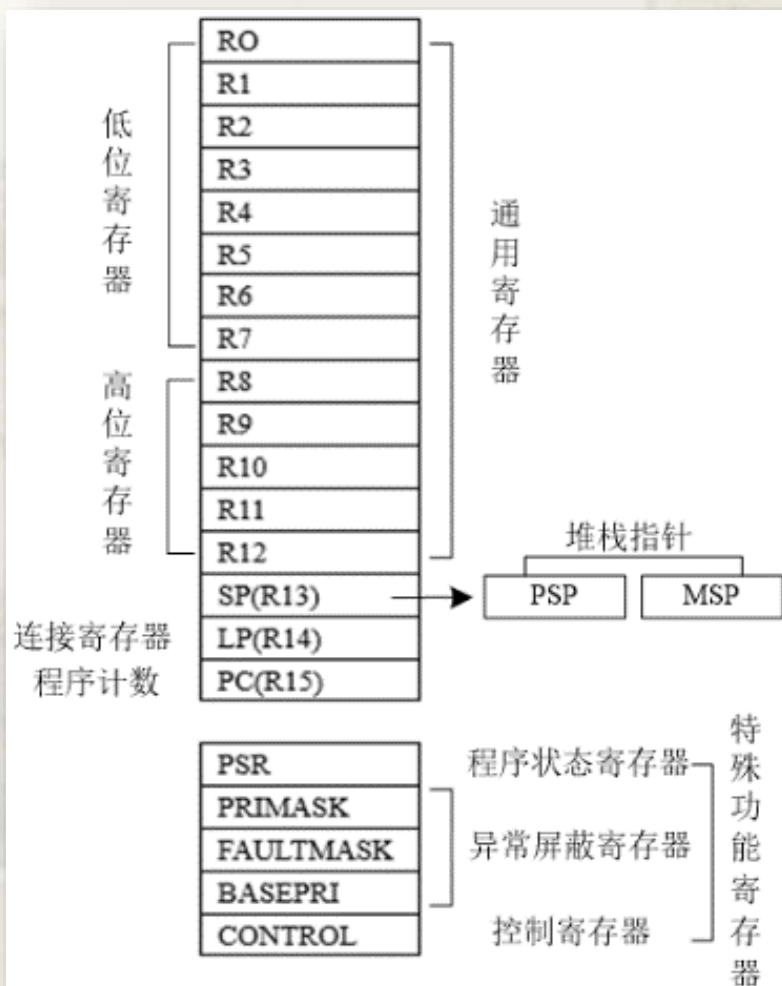
系统保留	511MB ↕	0xFFFFFFFF ↕ 0xE0100000 ↕
私有外部总线-外部	16MB ↕	0xE0FFFFFF ↕ 0xE0040000 ↕
私有外部总线-内部	256KB ↕	0xE003FFFF ↕ 0xE0000000 ↕
外部设备	1.0GB ↕	0xDFFFFFFF ↕ 0xA0000000 ↕
外 RAM	1.0GB ↕	0x9FFFFFFF ↕ 0x60000000 ↕
外围设备	0.5GB ↕	0x5FFFFFFF ↕ 0x40000000 ↕
SRAM	0.5GB ↕	0x3FFFFFFF ↕ 0x20000000 ↕
代码	0.5GB ↕	0x1FFFFFFF ↕ 0x00000000 ↕

- 该处理器直接寻址空间为4GB。
- 地址范围是：0x0000\_0000 ~ 0xFFFF\_FFFF。
- ARM定出的条条框框是粗线条的，它依然允许芯片制造商灵活地分配存储器空间，以制造出各具特色的MCU产品。
- CM4F的存储器系统支持小端配置和大端配置。一般具体某款芯片在出厂时已经被厂商定义过，例如MSP432采用小端格式。



## 2.1.3 ARM Cortex-M4F处理器的寄存器——概要

学习一个CPU，理解其内部寄存器用途是重要一环。

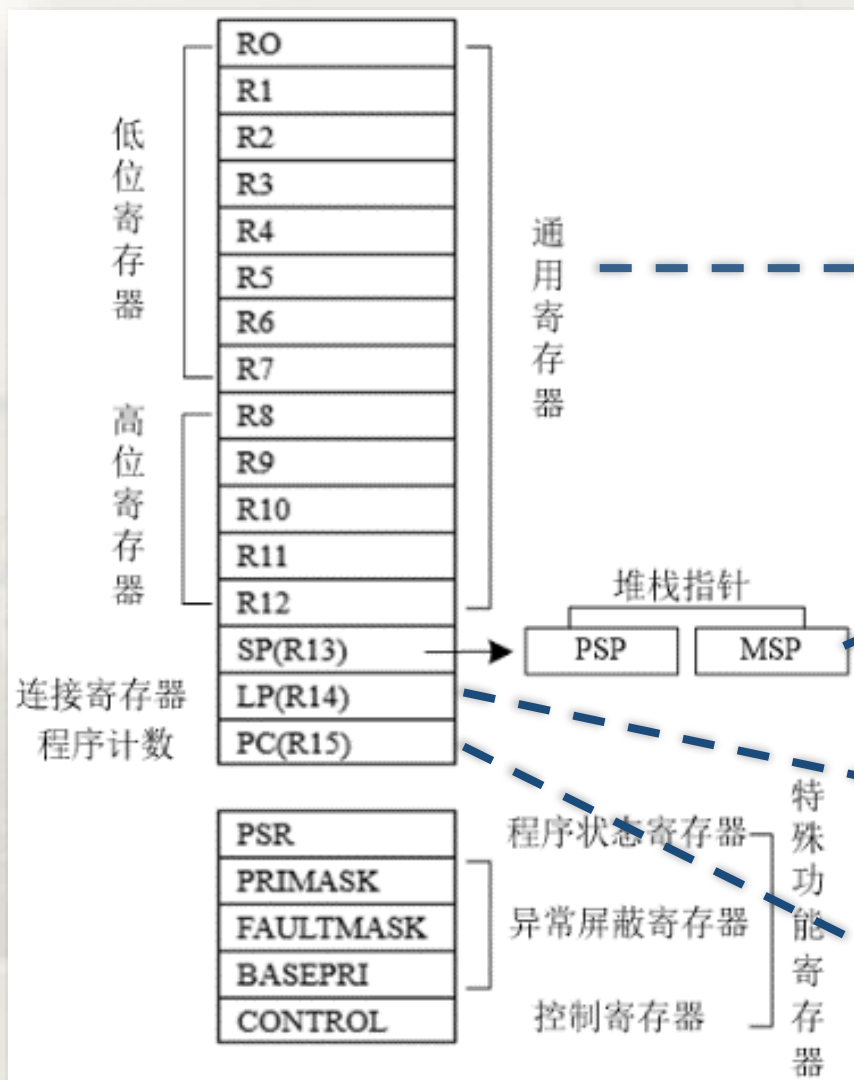


### CM4F处理器的寄存器包含：

- 用于数据处理与控制的寄存器。包括：R0 ~ R15，其中R13作为堆栈指针SP。SP实质上有两个（MSP与PSP），但在同一时刻只能有一个可以看到。
- 特殊功能寄存器，有预定义的功能，而且必须通过专用的指令来访问
- 浮点寄存器



## 2.1.3 ARM Cortex-M4F处理器的寄存器——数据处理与控制寄存器



大部分能够访问通用寄存器的指令都可以访问R0~R12。其中：低位寄存器（R0~R7）能够被所有访问通用寄存器的指令访问；高位寄存器（R8~R12）能够被所有32位通用寄存器指令访问，而不能被所有的16位指令访问。

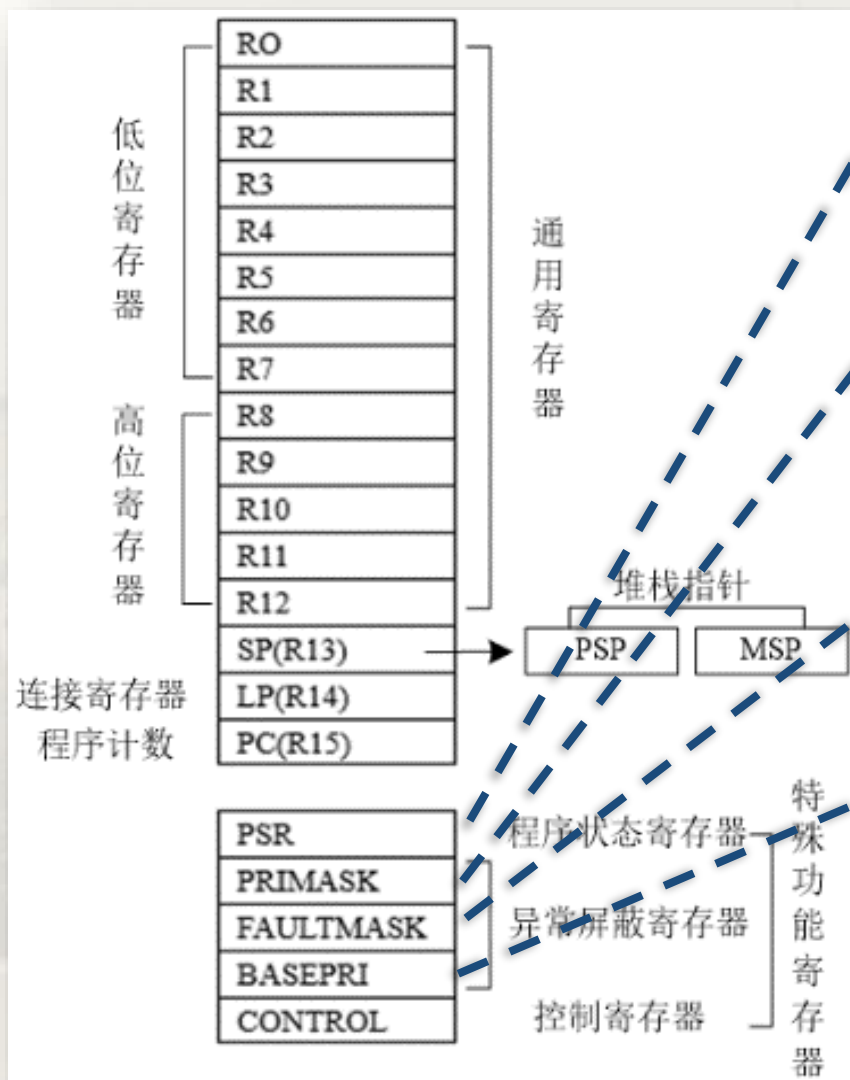
寄存器R13被用作堆栈指针（SP），用于访问堆栈。

寄存器R14为子程序连接寄存器（LR）

寄存器R15是程序计数寄存器（PC），指向当前的程序地址。



## 2.1.3 ARM Cortex-M4F处理器的寄存器——特殊功能寄存器



**程序状态字寄存器**在内部分为以下几个子寄存器：APSR、IPSR、EPSR。

**中断屏蔽寄存器**的D31~D1位保留，只有D0位（记为PM）有意义。当该位被置位时，除不可屏蔽中断和硬件错误之外的所有中断都会被屏蔽。

**错误屏蔽寄存器**与中断屏蔽寄存器的区别在于它能够屏蔽掉优先级更高的硬件错误异常。

**基本优先级屏蔽寄存器**提供了一种更加灵活的中断屏蔽机制。

**控制寄存器**的D31~D2位保留，D1、D0有意义。





### 2.1.3 ARM Cortex-M4F处理器的寄存器——浮点寄存器

浮点运算时M4F处理器的最大亮点之一，所以浮点寄存器只在Cortex-M4F处理器中存在，其中包含了用于**浮点数据处理**与**状态控制**的寄存器。

**浮点处理寄存器**：S0~S31和D0~D15，S0~S31都是32位寄存器，每个寄存器都可用来存放单精度浮点数，它们两两组合可用来存放双精度浮点数，两两组合成双精度寄存器时可用D0~D15来访问。

**浮点状态控制寄存器**：浮点状态控制寄存器提供了浮点系统的应用程序级控制，其包括浮点运算结果的状态信息与定义一些浮点运算的动作。

31	30	29	28	27	26	25	24	23~22	21~8	7	6~5	4	3	2	1	0
N	Z	C	V		AHP	DN	FZ	RMode		IDC		IXC	UFC	OFC	DZC	IOC

负标志N

零标志Z

进位/借位标志C

溢出标志V

交替半精度控制位AHP

默认NaN模式控制位DN

清零模式控制位FZ

、舍入模式控制位RMode

溢出累积异常位OFC

输入非正常累积异常位IDC

不精确累积异常位IXC

下溢累积异常位UFC



苏州大学

SOOCHOW UNIVERSITY



## 2.2 指令系统





## 2.2.1

### 指令系统——简介

#### CPU功能

从外部设备获得数据，通过加工、处理，再把处理结果送到CPU的外部世界。

#### 指令

设计一个CPU，首先需要设计一套可以执行特定功能的操作命令，这种操作命令称为指令。

#### 指令系统

CPU所能执行的各种指令的集合，称为该CPU的指令系统。



## 2.2 指令系统——ARM系统架构

在ARM系统中，使用**架构**（architecture）一词，即体系结构，主要指使用的指令集。

同一架构，可以衍生出许多不同处理器型号，比如：ARMv7-M是一种架构型号，其中v7是指版本号，而基于该架构处理器有Cortex-M3、Cortex-M4、Cortex-M4F等。

处理器型号	Thumb	Thumb-2	硬件乘法	硬件除法	饱和运算	DSP扩展	浮点	ARM架构	核心架构
Cortex-M0	大部分	子集	1或32个周期	无	无	无	无	ARMv6-M	冯诺依曼
Cortex-M1	大部分	子集	3或33个周期	无	无	无	无	ARMv6-M	冯诺依曼
Cortex-M3	全部	全部	1个周期	有	有	无	无	ARMv7-M	哈佛
Cortex-M4	全部	全部	1个周期	有	有	有	可选	ARMv7-M	哈佛

### ARM Cortex-M指令集概况





## 2.2.1 指令系统——指令简表与寻址方式

类型	保留字	含义
数据传送类	ADR	生成与 PC 指针相关的地址
	LDR、LDRH、LDRB、LDRSB、LDRSH、LDMIA	存储器中内容加载到寄存器中
	STR、STRH、STRB、STMIA	寄存器中内容存储至存储器中
	MOV、MVN	寄存器间数据传送指令
	PUSH、POP	进栈、出栈
数据操作类	算术运算类	ADC、ADD、SBC、SUB、MUL
		CMN、CMP
	逻辑运算类	AND、ORR、EOR、BIC
	数据序转类	REV、REVSH、REVH
	扩展类	SXTB、SXTH、UXTB、UXTH
	位操作类	TST
	移位类	ASR、LSL、LSR、ROR
	取补码类	NEG
	复制类	CPY
跳转控制类	B、B<cond>、BL、BLX、CBZ、CBNZ	跳转指令
其它指令	BKPT、SVC、NOP、CPSID、CPSIE	

基本指令简表





## 2.2.1 指令系统——指令简表与寻址方式

### 立即数寻址

在立即数寻址方式中，操作数直接通过指令给出，数据包含指令编码中，随着指令一起被编译成机器码存储于程序空间中。用“#”作为立即数的前导标识符

### 寄存器寻址

在寄存器寻址中，操作数来自于寄存器。

### 直接寻址

在直接寻址方式中，操作数来自于存储单元，指令中直接给出存储单元地址。指令码中，显示给出数据的位数，有字（4字节）、半字（2字节）、单字节三种情况。

### 偏移寻址及寄存器间接寻址

在偏移寻址中，操作数来自于存储单元，指令中通过寄存器及偏移量给出存储单元的地址。



## 2.2.1 指令系统——寻址方式举例

### 立即数寻址

SUB R1,R0,#1 //R1←R0-1  
MOV R0,#0xff //即数0xff装入R0寄存器

### 寄存器寻址

MOV R1,R2 // ←R2  
SUB R0,R1,R2 //0←R1-R2

### 直接寻址

LDR Rt,label //号label处连续取4字节至寄存器中  
LDRH Rt,label //址label处读取半字到Rt  
LDRB Rt,label //址label处读取字节到Rt

### 偏移寻址 及寄存器 间接寻址

LDR R3, [PC, #100] //为 ( PC + 100 ) 的存储器单元  
的内容加载到寄存器R3中  
LDR R3,[R4] //为R4的存储单元的内容加载到寄存  
器R3中



## 2.2.2

## 指令系统——数据传送类指令:取数指令

### 存储器中内容加载 ( load ) 到寄存器中的指令

编号	指令	说明
(1)	LDR Rt, [<Rn   SP> {, #imm}]	从 {SP/Rn+ #imm} 地址处, 取字到 Rt, imm=0,4,8,...,1020
	LDR Rt,[Rn, Rm]	从地址 Rn+Rm 处读取字到 Rt
	LDR Rt, label	从 label 指定的存储器单元取数至寄存器, label 必须在当前指令的-4~4KB 范围内, 且应 4 字节对齐
(2)	LDRH Rt, [Rn {, #imm}]	从 {Rn+ #imm} 地址处, 取半字到 Rt 中, imm=0,2,4,...,62
	LDRH Rt,[Rn, Rm]	从地址 Rn+Rm 处读取半字到 Rt
(3)	LDRB Rt, [Rn {, #imm}]	从 {Rn+ #imm} 地址处, 取字节到 Rt 中, imm=0~31
	LDRB Rt,[Rn, Rm]	从地址 Rn+Rm 处读取字节到 Rt
(4)	LDRSH Rt,[Rn, Rm]	从地址 Rn+ Rm 处读取半字至 Rt, 并带符号扩展至 32 位
(5)	LDRSB Rt,[Rn, Rm]	从地址 Rn+ Rm 处读取字节至 Rt, 并带符号扩展至 32 位
(6)	LDM Rn{!}, reglist	从 Rn 处读取多个字, 加载到 reglist 列表寄存器中, 每读一个字后 Rn 自增一次

### 取数指令



## 2.2.2 指令系统——数据传送类指令:存数指令

### 寄存器中内容存储 ( store ) 至存储器中的指令

(7)	STR Rt, [<Rn   SP> {, #imm}]	把 Rt 中的字存储到地址 SP/Rn+#imm, imm=0,4,8,...,1020
	STR Rt, [Rn, Rm]	把 Rt 中的字存储到地址 Rn+ Rm 处
(8)	STRH Rt, [Rn {, #imm}]	把 Rt 中的低半字存储到地址 SP/Rn+#imm, imm=0,2,4,...,62
	STRH Rt, [Rn, Rm]	把 Rt 中的低半字存储到地址 Rn+ Rm 处
(9)	STRB Rt, [Rn {, #imm}]	把 Rt 中的低字节 SP/Rn+#imm。imm=0~31
	STRB Rt, [Rn, Rm]	把 Rt 中的低字节存储到地址 Rn+ Rm 处
(10)	STM Rn!, reglist	存储多个字到 Rn 处。每存一个字后 Rn 自增一次

#### 存数指令



## 2.2.2 指令系统——数据传送类指令:堆栈操作指令

堆栈 ( stack ) 操作指令 :

- **PUSH指令**将寄存器值存于堆栈中，最低编号寄存器使用最低存储地址空间，最高编号寄存器使用最高存储地址空间；
- **POP指令**将值从堆栈中弹回寄存器，最低编号寄存器使用最低存储地址空间，最高编号寄存器使用最高存储地址空间。

编号	指令	说明
(14)	PUSH reglist	进栈指令。SP 递减 4
(15)	POP reglist	出栈指令。SP 递增 4

堆栈操作  
指令举例

**PUSH {R0,R4-R7}**

@将R0,R4 ~ R7寄存器值入栈

**PUSH {R2,LR}**

@将R2,LR寄存器值入栈

**POP {R0,R6,PC}**  
至PC所指向的地址

@出栈值到R0,R6,PC中，同时跳转





## 2.2.2 指令系统——数据传送类指令:寄存器间数据传送指令

**MOV指令**：Rd表示目标寄存器；imm为立即数，范围0x00 ~ 0xff

编号	指令	说明
(11)	MOV Rd, Rm	$Rd \leftarrow Rm$ , Rd 只可以是 R0~R7。
(11)	MOVS Rd, #imm	MOVS 指令功能与 MOV 相同，且影响 N、Z 标志
(13)	MVN Rd, Rm	将寄存器 Rm 中数据取反，传送给寄存器 Rd，影响 N、Z 标志

寄存器间数据传送指令



## 2.2.3 指令系统——数据操作类指令:算术运算类指令

编号	指令	说明
(17)	ADC {Rd, } Rn, Rm	带进位加法。 $Rd \leftarrow Rn + Rm + C$ ，影响 N、Z、C 和 V 标志位
(18)	ADD {Rd } Rn, <Rm   #imm>	加法。 $Rd \leftarrow Rn + Rm$ ，影响 N、Z、C 和 V 标志位
(19)	RSB {Rd, } Rn, #0	$Rd \leftarrow 0 - Rn$ ，影响 N、Z、C 和 V 标志位（KDS 环境不支持）
(20)	SBC {Rd, } Rn, Rm	带借位减法。 $Rd \leftarrow Rn - Rm - C$ ，影响 N、Z、C 和 V 标志位
(21)	SUB {Rd } Rn, <Rm   #imm>	常规减法。 $Rd \leftarrow Rn - Rm / \#imm$ ，影响 N、Z、C 和 V 标志位
(22)	MUL Rd, Rn Rm	常规乘法， $Rd \leftarrow Rn * Rm$ ，同时更新 N、Z 状态标志，不影响 C、V 状态标志。该指令所得结果与操作数是否为无符号、有符号数无关。Rd、Rn、Rm 寄存器必须为 R0~R7，且 Rd 与 Rm 须一致。
(23)	CMN Rn, Rm	加比较指令。 $Rn + Rm$ ，更新 N、Z、C 和 V 标志，但不保存所得结果。Rn、Rm 寄存器必须为 R0~R7
(24)	CMP Rn, #imm	（减）比较指令。 $Rn - Rm / \#imm$ ，更新 N、Z、C 和 V 标志，但不保存所得结果。Rn、Rm 寄存器为 R0~R7，立即数 imm 范围 0~255
	CMP Rn, Rm	

算术类指令有加、减、乘、比较等指令



## 2.2.3 指令系统——数据操作类指令:算术运算类指令限制条件

指令	Rd	Rn	Rm	imm	限制条件
ADC	R0~R7	R0~R7	R0~R7	-	Rd 和 Rn 必须相同
ADD	R0~R15	R0~R15	R0~PC	-	Rd 和 Rn 必须相同; Rn 和 Rm 不能同时指定为 PC 寄存器;
	R0~R7	SP 或 PC	-	0~1020	立即数必须为 4 的整数倍
	SP	SP	-	0~508	立即数必须为 4 的整数倍
ADD	R0~R7	R0~R7	-	0~7	-
	R0~R7	R0~R7	-	0~255	Rd 和 Rn 必须相同
	R0~R7	R0~R7	R0~R7	-	-
RSB	R0~R7	R0~R7	-	-	-
SBC	R0~R7	R0~R7	R0~R7	-	Rd 和 Rn 必须相同
SUB	SP	SP	-	0~508	立即数必须为 4 的整数倍
SUB	R0~R7	R0~R7	-	0~7	-
	R0~R7	R0~R7	-	0~255	Rd 和 Rn 必须相同
	R0~R7	R0~R7	R0~R7	-	-

算术运算类指令限制条件



## 2.2.3 指令系统——数据操作类指令:逻辑运算类指令

**AND、EOR和ORR指令把寄存器Rn、Rm值逐位与、异或和或操作；BIC指令是将寄存器Rn的值与Rm的值的反码按位作逻辑“与”操作，结果保存到Rd。**

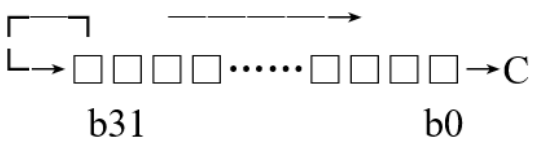
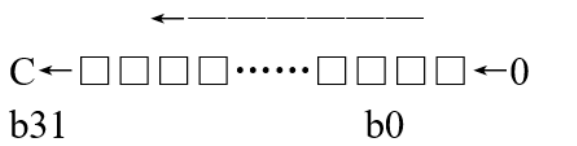
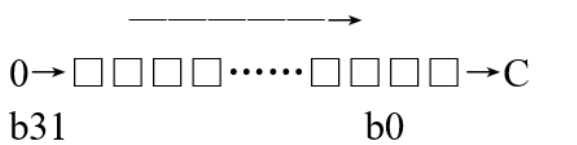
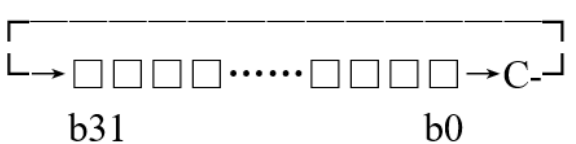
编号	指令	说明	举例
(25)	AND {Rd, } Rn, Rm	按位与	AND R2, R2, R1
(26)	ORR {Rd, } Rn, Rm	按位或	ORR R2, R2, R5
(27)	EOR {Rd, } Rn, Rm	按位异或	EOR R7, R7, R6
(28)	BIC {Rd, } Rn, Rm	位段清零	BIC R0, R0, R1

逻辑运算类指令



## 2.2.3 指令系统——数据操作类指令:移位类指令

**ASR、LSL、LSR和ROR指令，将寄存器Rm值由寄存器Rs或立即数imm决定移动位数，执行算术右移、逻辑左移、逻辑右移和循环右移。**

编号	指令	操作	举例
(29)	ASR {Rd, } Rm, Rs ASR {Rd, } Rm, #imm		算术右移 ASR R7, R5, #9
(30)	LSL {Rd, } Rm, Rs LSL {Rd, } Rm, #imm		逻辑左移 LSL R1, R2, #3
(31)	LSR {Rd, } Rm, Rs LSR {Rd, } Rm, #imm		逻辑右移 LSR R1, R2, #3
(32)	ROR {Rd, } Rm, Rs		循环右移 ROR R4, R4, R6

移位指令





## 2.2.3 指令系统——数据操作类指令:位测试指令

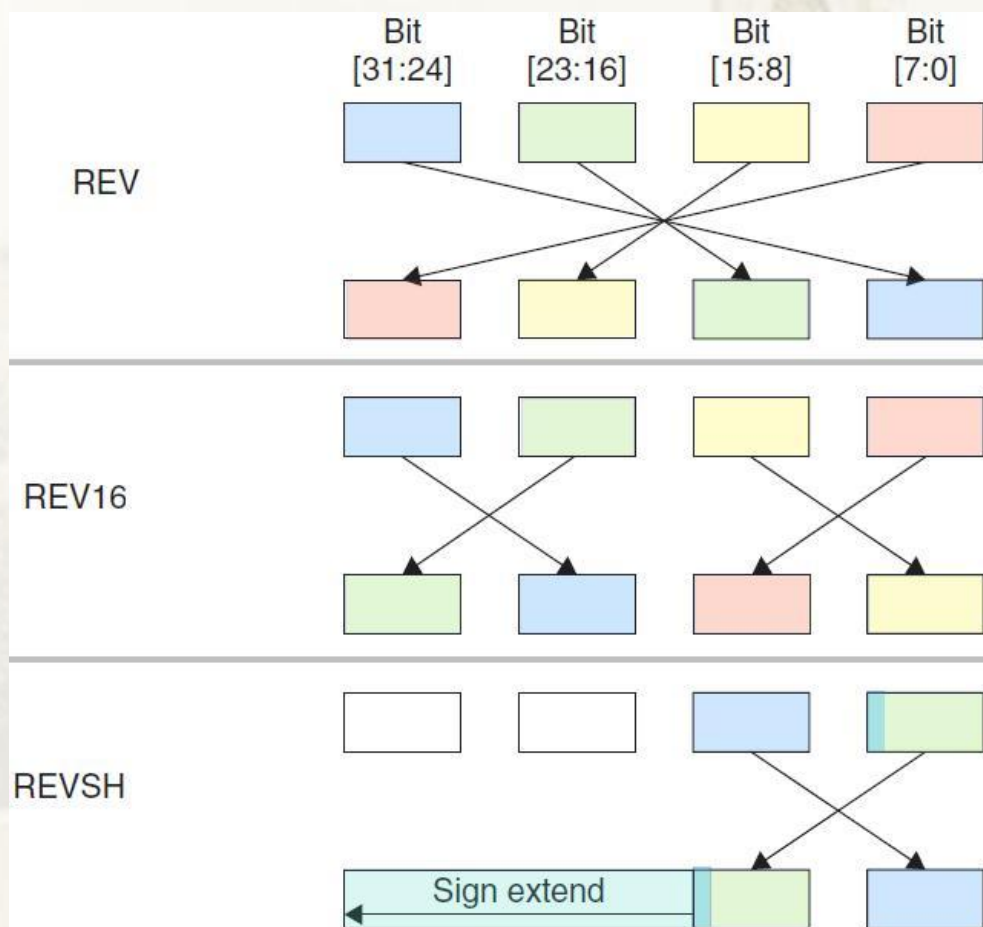
编号	指令	说明
(33)	TST Rn, Rm	将 Rn 寄存器值逐位与 Rm 寄存器值进行与操作，但不保存所得结果。为测试寄存器 Rn 某位为 0 或 1，将 Rn 寄存器某位置 1，其余位清零。寄存器 Rn、Rm 必须为 R0~R7。该指令根据结果，更新 N、Z 状态标志，但不影响 C、V 状态标志。

位测试指令



### 2.2.3 指令系统——数据操作类指令:数据序转指令 (1)

**该指令用于改变数据的字节顺序。**  $R_n$ 为源寄存器， $R_d$ 为目标寄存器，且必须为 $R_0 \sim R_7$ 之一。



REV指令将32位大端数据转小端存放或将32位小端数据转大端存放。

REV16指令将一个32位数据划分成两个16位大端数据，将这两个16位大端数据转小端存放；或将一个32位数据划分成两个16位小端数据，将这两个16位小端数据转大端存放

REVSH指令将16位带符号大端数据转成32位带符号小端数据或将16位带符号小端数据转成32位带符号大端数据



## 2.2.3 指令系统——数据操作类指令:数据序转指令（2）

编号	指令	说明
(34)	REV Rd, Rn	将 32 位大端数据转小端存放或将 32 位小端数据转大端存放
(35)	REV16 Rd, Rn	将一个 32 位数据划分成两个 16 位大端数据，将这两个 16 位大端数据转小端存放或将一个 32 位数据划分成两个 16 位小端数据，将这两个 16 位小端数据转大端存放
(36)	REVSH Rd, Rn	将 16 位带符号大端数据转成 32 位带符号小端数据或将 16 位带符号小端数据转成 32 位带符号大端数据

数据序转指令



## 2.2.3 指令系统——数据操作类指令:扩展类指令

编号	指令	说明
(37)	SXTB Rd, Rm	将操作数 Rm 的 Bit[7:0]带符号扩展到 32 位, 结果保存到 Rd 中
(38)	SXTH Rd, Rm	将操作数 Rm 的 Bit[15:0]带符号扩展到 32 位, 结果保存到 Rd 中
(39)	UXTB Rd, Rm	将操作数 Rm 的 Bit[7:0]无符号扩展到 32 位, 结果保存到 Rd 中
(40)	UXTH Rd, Rm	将操作数 Rm 的 Bit[15:0]无符号扩展到 32 位, 结果保存到 Rd 中

扩展类指令



## 2.2.4

### 指令系统——跳转控制类指令

编号	指令	跳转范围	说明
(41)	B{cond} label	-256B~+254B	转移到 Label 处对应的地址处。可以带（或不带）条件，所带条件见表 2-19，如：BEQ 表示标志位 Z=1 时转移。
(42)	BL label	-16MB~+16MB	转移到 Label 处对应的地址，并且把转移前的下条指令地址保存到 LR，并置寄存器 LR 的 Bit[0]为 1，保证了随后执行 POP {PC}或 BX 指令时成功返回分支
(43)	BX Rm	任意	转移到由寄存器 Rm 给出的地址，寄存器 Rm 的 Bit[0]必须为 1，否则会导致硬件故障
(44)	BLX Rm	任意	转移到由寄存器 Rm 给出的地址，并且把转移前的下条指令地址保存到 LR。寄存器 Rm 的 Bit[0]必须为 1，否则会导致硬件故障

#### 跳转控制类指令

#### 跳转控制 指令举例

**BEQ label    @条件转移，标志位Z=1时转移到label**

**BL funC      @调用子程序 funC，把转移前的下条指令地址保存到LR**

**BX LR        @返回到函数调用处**





## 2.2.5 指令系统——其他指令

类型	编号	指令	说明
断点指令	(45)	BKPT #imm	如果调试被使能，则进入调试状态（停机）。或者如果调试监视器异常被使能，则调用一个调试异常，否则调用一个错误异常。处理器忽视立即数 imm，立即数范围 0~255，表示断点调试的信息。不影响 N、Z、C、V 状态标志。
中断指令	(46)	CPSIE i	除了 NMI，使能总中断，不影响 N、Z、C、V 标志。
	(47)	CPSID i	除了 NMI，禁止总中断，不影响 N、Z、C、V 标志。
屏蔽指令	(48)	DMB	数据内存屏蔽（与流水线、MPU 和 cache 等有关）
	(49)	DSB	数据同步屏蔽（与流水线、MPU 和 cache 等有关）
	(50)	ISB	指令同步屏蔽（与流水线、MPU 等有关）
特殊寄存器操作指令	(51)	MRS Rd, spec_reg1	加载特殊功能寄存器值到通用寄存器。若当前执行模式不为特权模式，除 APSR 寄存器外，读其余所有寄存器值为 0。
	(52)	MSR spe_reg, Rn	存储通用寄存器的值到特殊功能寄存器。Rd 不允许为 SP 或 PC 寄存器，若当前执行模式不为特权模式，除 APSR 外，任何试图修改寄存器操作均被忽视。影响 N、Z、C、V 标志。
空操作	(53)	NOP	空操作，但无法保证能够延迟时间，处理器可能在执行阶段之前就将此指令从线程中移除。不影响 N、Z、C、V 标志。
发送事件指令	(54)	SEV	发送事件指令。在多处理器系统中，向所有处理器发送一个事件，也可置位本地寄存器。不影响 N、Z、C、V 标志。
操作系统服务调用指令	(55)	SVC #imm	操作系统服务调用，带立即数调用代码。SVC 指令触发 SVC 异常。处理器忽视立即数 imm，若需要，该值可通过异常处理程序重新取回，以确定哪些服务正在请求。执行 SVC 指令期间，当前任务优先级高于等于 SVC 指令调用处理程序时，将产生一个错误。不影响 N、Z、C、V 标志。
休眠指令	(56)	WFE	休眠并且在发生事件时被唤醒。不影响 N、Z、C、V 标志
	(57)	WFI	休眠并且在发生中断时被唤醒。不影响 N、Z、C、V 标志



苏州大学

SOOCHOW UNIVERSITY



## 2.3

# 指令集与机器码对应表





## 2.3 指令集与机器码对应表——如何找到机器码

### 第一步 生成.lst 文件

需要在集成环境里对工程进行设置，让它能够生成.lst文件，在本书里集成环境为CCS。也可对于汇编和C/C++工程编译生成.lst文件。

### 第二步 记录， 找规律

将指令按照正确的格式给变量赋值，写入main函数中编译，编译结束后查看main.o.lst文件，找到指令对应的十六进制机器码，进行记录

### 第三步： 确定机器 码

每个变量注意取不同的值，多次进行编译记录，然后找出规律，确定每个变量变化所改变的机器码的部分。至此就可以完全确定这个指令所对应的机器码。



## 2.3 指令集与机器码对应表——指令集与机器码对应表（1）

分类	序号	助记符	指令格式	机器码	实例	
					指令	机器码
数据 传送 类 指令	1	LDR	LDR Rd,[RN,RM] LDR Rd,label LDR Rd,[<RN SP>{,#imm}]	0101 100m mmnn nddd 0100 1ddd vvvv vvvv 0110 1vvv vvnn nddd	LDR r4, [r4, r5] LDR r5, =runpin LDR r1, [r5, #0]	5964 4D0A 6829
	2	LDRH	LDRH Rd,[Rn{,#imm}] LDRH Rd,[Rn,Rm]	1000 1vvv vvnn nddd 0101 101m mmnn nddd	LDRH r4, [r4, #30] LDRH r4, [r4, r5]	8be4 5b64
	3	LDRB	LDRB Rd,[Rn{,#imm}] LDRB Rd,[Rn,Rm]	0111 1vvv vvnn nddd 0101 110m mmnn nddd	LDRB r4, [r4, #30] LDRB r4, [r4, r5]	7FA4 5D64
	4	LDRSH	LDRSH Rd,[Rn,Rm]V	0101 111m mmnn nddd	LDRSH r4, [r4, r5]	5F64
	5	LDRSB	LDRSB Rd,[Rn,Rm]	0101 011m mmnn nddd	LDRSB r4, [r4, r5]	5764
	6	LDM	LDM Rn{!},Reglist	1100 1nnn rrrr rrrr	LDM r0,{r0,r3,r4}	C819
	7	STR	STR Rd,[<RN SP>{,#imm}] STR Rd,[Rn,Rm]	0110 0vvv vvnn nddd 0101 000m mmnn nddd	STR r0, [r5, #4] STR r0,[r5,r4]	6068 5128
	8	STRH	STRH Rd,[Rn{,#imm}] STRH Rd,[Rn,Rm]	1000 0vvv vvnn nddd 0101 001m mmnn nddd	STRh r0,[r5,#4] STRh r0,[r5,r4]	80A8 5328
	9	STRB	STRB Rd,[Rn{,#imm}] STRB Rd,[Rn,Rm]	0111 0vvv vvnn nddd 0101 010m mmnn nddd	STRB r0,[r5,#4] STRB r0,[r5,r4]	7128 5528
	10	STM	STM Rn!,reglist	1100 0nnn rrrr rrrr	STMIA r0,{r0,r3,r4}	C019
	11	MOV	MOV Rd,Rm	0001 1100 00nn nddd	MOV r1,r2	1C11
	12	MOVS	MOVS Rd,#imm	0010 0ddd vvvv vvvv	MOVS r1,#8	2108
	13	MVN	MVN Rd,Rm	0100 0011 11mm mddd	MVN r1,r3	43D9
	14	PUSH	PUSH reglist	1011 010R rrrr rrrr	PUSH {r1}	B402
	15	POP	POP reglist	1011 110R rrrr rrrr	PUSH {r1}	BC02
	16	ADR	ADR Rd,label	1010 0ddd vvvv vvvv	ADR R3, loop	A303



## 2.3 指令集与机器码对应表——指令集与机器码对应表（2）

分类	序号	助记符	指令格式	机器码	实例	
					指令	机器码
数据操作类指令	17	ADC	ADC {Rd} Rn,Rm	0100 0001 01mm mddd	ADC r2,r3	415A
	18	ADD	ADD {Rd} Rn<Rm/#imm>	0001 100m mmnn nddd	ADD r2,r3	18D2
			ADD Rn,#imm	0011 0ddd vvvv vvvv	ADD r2,#12	320C
			ADD Rd,Rn,#imm	0001 110v vvnn nddd	ADD r2,r3,#1	1C5A
			ADD Rd,Rn,#imm	0001 100m mmnn nddd	ADD r2,r3,r4	191A
	19	RSB	RSB {Rd},Rn,Rm	32 位指令	RSB r5,#2	C5F1 0205
	20	SBC	SBC {Rd}, Rn, Rm	0100 0001 10mm mnnn	SBC R7,R7,R1	418F
	21	SUB	SUB Rd,#imm	0011 1ddd vvvv vvvv	SUB r4,#1	3C01
			SUB Rd,Rn,#imm	0001 111v vvnn nddd	SUB r3,r4,#1	1E63
			SUB Rd,Rn,Rm	0001 101m mmnn nddd	SUB r3,r4,r1	1A63
	22	MUL	MUL Rd,Rn,Rm	0100 0011 01mm mddd	MUL r1,r2,r1	4351
	23	CMN	CMN Rn,Rm	0100 0010 11mm mnnn	CMN r1,r2	42D1
	24	CMP	CMP Rn,#imm	0010 1nnn vvvv vvvv	CMP r4,#1	2C01
			CMP Rn,Rm	0100 0010 10mm mnnn	CMP r4,r1	428C
	25	AND	AND {Rd},Rn,Rm	0100 0000 00mm mddd	AND r1,r2	4011
	26	ORR	ORR {Rd},Rn,Rm	0100 0011 00mm mddd	ORR r1,r2	4311
	27	EOR	EOR {Rd},Rn,Rm	0100 0000 01mm mddd	EOR r1,r2	4051
	28	BIC	BIC {Rd},Rn,Rm	0100 0011 10mm mddd	BIC r1,r2	4391
	29	REV	REV Rd,Rn	1011 1010 00nn nddd	REV r1,r2	BA11
	30	REV16	REV16 Rd,Rn	1011 1010 01nn nddd	REV16 r3,r3	BA5B
	31	REVSH	REVSH Rd,Rn	1011 1010 11nn nddd	REVSH r4,r3	BADC





## 2.3 指令集与机器码对应表——指令集与机器码对应表（3）

分类	序号	助记符	指令格式	机器码	实例	
					指令	机器码
	32	SXTB	SXTB Rd,Rm	1011 0010 01mm mddd	SXTB r4,r3	B25C
	33	SXTH	SXTH Rd,Rm	1011 0010 00mm mddd	SXTH r4,r3	B21C
	34	UXTB	UXTB Rd,Rm	1011 0010 11mm mddd	UXTB r4,r3	B2DC
	35	UXTH	UXTH Rd,Rm	1011 0010 10mm mddd	UXTH r4,r3	B29C
	36	TST	TST Rn,Rm	0100 0010 00mm mnnn	TST r1,r2	4211
	37	ASR	ASR {Rd,}Rm,Rs ASR {Rd,}Rm,#imm	0100 0001 00ss smmm 0001 0vvv vvm mddd	ASR r3,r3,r5 ASR r7,r5,#6	412B 11AF
	38	LSL	LSL {Rd,}Rm,Rs LSL {Rd,}Rm, #imm	0100 0000 10ss smmm 0000 0vvv vvm mddd	LSL r5,r6 LSL r5,r6,#6	40B5 01B5
	39	LSR	LSR {Rd,}Rm,Rs LSR {Rd,}Rm, #imm	0100 0000 11ss sddd 0000 1vvv vvm mddd	LSR r5,r6 LSR r5,r6,#6	40F5 09B5
	40	ROR	ROR {Rd,}Rm,Rs	0100 0001 11ss sddd	ROR r5,r6	41F5
跳转类指令	41	B	B label B{cond} label	1110 0vvv vvvv vvvv 1101 cccc vvvv vvvv	B loop BNE loop	E006 D106
	42	BL	BL label	32 位指令	BL loop	F807 F000
	43	BX	BX Rm	0100 0111 00mm m000	BX r1	4708
	44	BLX	BLX Rm	0100 0111 10mm m000	BLX r1	4788



## 2.3 指令集与机器码对应表——指令集与机器码对应表（4）

分类	序号	助记符	指令格式	机器码	实例	
					指令	机器码
其它指令	45	BKPT	BKPT #imm	1011 1110 vvvv vvvv	BKPT	BE00
	46	CPSIE	CPSIE i	1011 0110 0110 0010	CPSIE i	B662
	47	CPSID	CPSID i	1011 0110 0111 0010	CPSID i	B672
	48	DMB	DMB	32 位指令	DMB	F3BF 8F5F
	49	DSB	DSB	32 位指令	DSB	F3BF 8F4F
	50	ISB	ISB	32 位指令	ISB	F3BF 8F6F
	51	MRS	MRS Rd,spec_reg1	32 位指令	MRS R5, PRIMASK	F3EF 8510
	52	MSR	MSR spec_reg,Rn	32 位指令	MSR PRIMASK,R5	F385 8810
	53	NOP	NOP		NOP	46C0
	54	SEV	SEV	1011 1111 0100 0000	SEV	BF40
	55	SVC	SVC #imm	1101 1111 vvvv vvvv	SVC #12	DF0C
	56	WFE	WFE	1011 1111 0010 0000	WFE	BF20
	57	WFI	WFI	1011 1111 0011 0000	WFI	BF30



苏州大学

SOOCHOW UNIVERSITY



## 2.4

# 汇编语言的基本语法





## 2.4

### 汇编语言的基本语法——汇编语言简介

机器语言

能够在MCU内直接执行的指令序列是**机器语言**。

汇编语言

用助记符号来表示机器指令便于记忆，这就形成了**汇编语言**。

汇编语言

用汇编语言写成的程序不能直接放入MCU的程序存储器中去执行，必须先转为机器语言。把用汇编语言写成的源程序“翻译”成机器语言的工具叫汇编程序或汇编器（Assembler），以下统一称作**汇编器**。



## 2.4.1

### 汇编语言的基本语法——汇编语言格式

- 编语言源程序可以用通用的文本编辑软件编辑，以ASCII码形式存盘。
- 具体的编译器对汇编语言源程序的格式有一定的要求，为了能够正确地产生目标代码以及方便汇编语言的编写，编译器还提供了一些在汇编时使用的命令、操作符号，在编写汇编程序时，也必须正确使用它们。
- 由于编译器提供的指令仅是为了更好地做好“翻译”工作，并不产生具体的机器指令，因此这些指令被称为**伪指令**。
- 汇编语言源程序以行为单位进行设计，每一行最多可以包含以下四个部分：

标号：	操作码	操作数	注释
-----	-----	-----	----





## 2.4.1 汇编语言的基本语法——标号

标号:	操作码	操作数	注释
-----	-----	-----	----

- ① 如果一个语句有标号，则标号必须书写在汇编语句的开头部分。
- ② 可以组成标号的字符有：字母A~Z、字母a~z、数字0~9、下划线“\_”、美元符号“\$”，但开头的第一个符号不能为数字和\$。
- ③ 编译器对标号中字母的大小写敏感，但指令不区分大小。
- ④ 标号长度基本上不受限制，但实际使用时通常不要超过20个字符。若希望更多的编译器能够识别，建议标号（或变量名）的长度小于8个字符。
- ⑤ 标号后必须带冒号“:”。
- ⑥ 一个标号在一个文件（程序）中只能定义一次，否则重复定义，不能通过编译。
- ⑦ 一行语句只能有一个标号，编译器将把当前程序计数器的值赋给该标号。



## 2.4.1 汇编语言的基本语法——操作码

标号:	操作码	操作数	注释
-----	-----	-----	----

- ① 操作码包括指令码和伪指令，其中伪指令是指CCS开发环境ARM Cortex-M4F汇编编译器可以识别的伪指令。
- ② 对于有标号的行，必须用至少一个空格或制表符（TAB）将标号与操作码隔开。编译器对标号中字母的大小写敏感，但指令不区分大小。
- ③ 对于没有标号的行，不能从第一列开始写指令码，应以空格或制表符（TAB）开头。
- ④ 编译器不区分操作码中字母的大小写。



## 2.4.1 汇编语言的基本语法——操作数

标号:	操作码	操作数	注释
-----	-----	-----	----

- ① 操作数可以是地址、标号或指令码定义的常数，也可以是由伪运算符构成的表达式。
- ② 若一条指令或伪指令有操作数，则操作数与操作码之间必须用空格隔开书写。
- ③ 操作数多于一个的，操作数之间用逗号“,”分隔。操作数也可以是ARM Cortex-M4F内部寄存器，或者另一条指令的特定参数。
- ④ 操作数中一般都有一个存放结果的寄存器，这个寄存器在操作数的最前面。



## 2.4.1 汇编语言的基本语法——操作数注意点

### 常数标识

编译器识别的常数有十进制（默认不需要前缀标识）、十六进制（0x前缀标识）、二进制（用0b前缀标识）。

### “#”表示立即数

一个常数前添加“#”表示一个立即数，不加“#”时，表示一个地址。

### “#”使用代码举例

```
mov    r3, 1    //给寄存器r3赋值为1（这个语句不对）  
mov    r3, #1   //寄存器r3赋值为1（这个语句对）
```

### 圆点“.”

若圆点“.”单独出现在语句的操作码之后的操作数位置上，则代表当前程序计数器的值被放置在圆点的位置。



## 2.4.1

## 汇编语言的基本语法——操作数:伪运算符

运算符	功能	类型	实例	
+	加法	二元	mov 3,#30+40	等价于 mov r3,#70
-	减法	二元	mov r3,#40-30	等价于 mov r3,#10
*	乘法	二元	mov r3,#5*4	等价于 mov r3,#20
/	除法	二元	mov r3,#20/4	等价于 mov r3,#5
%	取模	二元	mov r3,#20%7	等价于 mov r3,#6
	逻辑或	二元	mov r3,#1  0	等价于 mov r3,#1
&&	逻辑与	二元	mov 3,#1&&0	等价于 mov r3,#0
<<	左移	二元	mov r3,#4<<2	等价于 mov r3,#16
>>	右移	二元	mov r3,#4>>2	等价于 mov r3,#1
^	按位异或	二元	mov r3,#4^6	等价于 mov r3,#2
&	按位与	二元	mov r3,#4^2	等价于 mov r3,#0
	按位或	二元	mov r3,#4 2	等价于 mov r3,#6
==	等于	二元	mov r3,#1==0	等价于 mov r3,#0
!=	不等于	二元	mov r3,#1!=0	等价于 mov r3,#1
<=	小于等于	二元	mov r3,#1<=0	等价于 mov r3,#0
>=	大于等于	二元	mov r3,#1>=0	等价于 mov r3,#1
+	正号	一元	mov r3,#+1	等价于 mov r3,#1
-	负号	一元	ldr r3,#-325	等价于 ldr r3,=0xffffebb
~	取反运算	一元	ldr r3,#~325	等价于 ldr r3,=0xffffeba
>	大于	一元	mov r3,#1>0	
<	小于	一元	mov r3,#1<=0	

CCS ARM Cortex-M4F编译器识别的伪运算符





## 2.4.1 汇编语言的基本语法——注释

标号： 操作码 操作数 注释

- ① 注释即是说明文字，类似于C语言，多行注释以 “/\*” 开始，以 “\*/” 结束。这种注释可以包含多行，也可以独占一行。
- ② 在CCS环境的ARM Cortex-M4F处理器汇编语言中，单行注释以 “#” 引导或者用 “//” 引导。用 “#” 引导，“#” 必须为单行的第一个字符。



## 2.4.2 常用伪指令简介（CCS环境为例介绍）——系统预定义的段

- C语言程序在经过gcc编译器最终生成.elf格式的可执行文件。
- .elf可执行程序是以段为单位来组织文件的。
- 通常划分为如下几个段：.text、.data和.bss，其中，.text是只读的代码区，.data是可读可写的数据区，而.bss则是可读可写且没有初始化的数据区。

.text	@表明以下代码在.text段
.data	@表明以下代码在.data段
.bss	@表明以下代码在.bss段



## 2.4.2 常用伪指令简介——常量的定义

- 使用常量定义，能够提高程序代码的可读性，并且使代码维护更加简单。
- 常量的定义可以使用.equ汇编指令，
- 常量的定义还可以使用.set汇编指令，其语法结构与.equ相同。
- 例如：

```
.equ    _NVIC_ICER, 0xE000E180
```

```
.....
```

```
LDR     R0,=_NVIC_ICER    @将0xE000E180放到R0中
```

```
.set    ROM_size, 128 * 1024           @ROM大小为131072字节 (128KB)
```

```
.set    start_ROM, 0xE0000000
```

```
.set    end_ROM, start_ROM + ROMsize   @ROM结束地址为0xE0020000
```



## 2.4.2 常用伪指令简介——程序中插入常量

用于程序中插入不同类型常量的常用伪指令：

插入数据的类型	GNU汇编器
字	.word（例如，.word 0x12345678）
半字	.hword（例如，.word 0x1234）
字节	.byte（例如，.byte 0x12）
字符串	ascii/.asciz（如，.ascii “hello\n”，.asciz与.ascii，只是生成的字符串以 ‘\0’结尾）

```
LDR R3,=NUMNER      @得到NUMNER的存储地址
LDR R4,[R3]          @将0x123456789读到R4
.....
LDR R0,=HELLO_TEXT  @得到HELLO_TEXT的起始地址
BL  PrintText        @调用PrintText函数显示字符串
.....
ALIGN 4
NUMNER:
    .word  0x123456789
HELLO_TEXT:
    .asciz “hello\n”    @以'\0'结束的字符
```

插入字和字符串常量代码：



## 2.4.2 常用伪指令简介——条件伪指令，文件包含伪指令

- `.if`条件伪指令后面紧跟着一个恒定的表达式（即该表达式的值为真），并且最后要以`.endif`结尾。中间如果有其他条件，可以用`.else`填写汇编语句。
- `.ifdef`标号，表示如果标号被定义，执行下面的代码。

`.include "filename"`

- `.include`是一个附加文件的链接指示命令，利用它可以把另一个源文件插入当前的源文件一起汇编，成为一个完整的源程序。
- `filename`是一个文件名，可以包含文件的绝对路径或相对路径，但建议对于一个工程的相关文件放到同一个文件夹中，更多的时候能使用相对路径。





## 2.4.2 常用伪指令简介——其他伪指令（1）

- **.section伪指令**：用户可以通过.section伪指令来自定义一个段。例如：

```
.section .isr_vector, "a"    @定义一个.isr_vector段, "a"表示允许段
```

- **.global伪指令**：.global伪指令可以用来定义一个全局符号。例如：

```
.global symbol    @定义一个全局符号symbol
```

- **.extern伪指令**：.extern伪指令的语法为：**.extern symbol**，声明**symbol为外部函数**，调用时可以遍访所有文件找到该函数并且使用它。

```
.extern main    @声明main为外部函数
```

```
bl main    @ 进入main函数
```



## 2.4.2 常用伪指令简介——其他常用伪指令

- **.align伪指令**：.align伪指令可以通过添加填充字节使当前位置满足一定的对齐方式。语法结构为：`.align [exp[, fill]]`，其中，exp为0-16之间的数字，表示下一条指令对齐至 $2^{\text{exp}}$ 位置，若未指定，则将当前位置对齐到下一个字的位置，fill给出为对齐而填充的字节值，可省略，默认为0x00。

`.align 3` @把当前位置计数器值增加到 $2^3$ 的倍数上，若已是 $2^3$ 的倍数，不做改变

- **.end伪指令**：.end伪指令声明汇编文件的结束。



苏州大学

SOOCHOW UNIVERSITY



谢谢！

