第5章

Cortex-M0+处理器指令集

ARM 架构是一种加载-保存架构，具有 32 位寻址范围。ARM 处理器是 RISC 处理器的典型代表，因为只有加载和保存指令才能访问存储器。数据处理指令仅对寄存器内容起作用。

本章介绍 ARM Cortex-M0＋处理器指令集，内容包括 Thumb 指令集、Keil MDK 汇编语言指令格式要点、寄存器传输指令、存储器加载和保存指令、多数据加载和访问访问指令、堆栈访问指令、算术运算指令、逻辑操作指令、移位操作指令、反序操作指令、扩展操作指令、程序流控制指令、存储器屏障指令、异常相关指令、休眠相关指令和其他指令。

通过学习 Cortex-M0＋处理器指令集，使读者进一步了解和掌握 Cortex-M0＋处理器核的工作原理，从而更深刻地理解 C 语言、汇编语言和底层 Cortex-M0＋处理器核之间的联系。

视频讲解

5.1 Thumb 指令集

在早期的 ARM 处理器中，使用了 32 位指令集，称为 ARM 指令。这个指令集具有较高的运行性能，与 8 位和 16 位的处理器相比，有更大的程序存储空间。但是，这也带来了较大的功耗。

1995 年，16 位的 Thumb-1 指令集首先应用于 ARM7TDMI 处理器，它是 ARM 指令集的子集。与 32 位的 RISC 结构相比，它提供更好的代码密度，将代码长度减少了 30%，但是性能也降低了 20%。通过使用多路复用器，它能与 ARM 指令集一起使用，如图 5.1 所示。

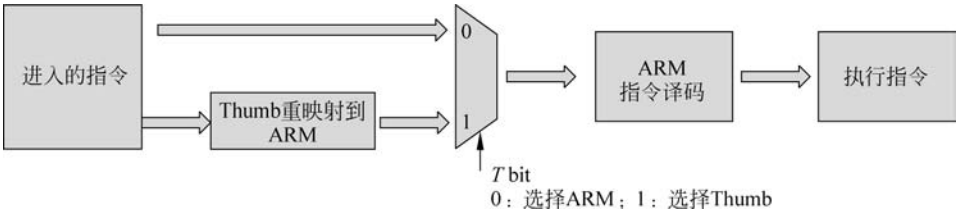


图 5.1　Thumb 指令选择

Thumb 指令流是一系列采用半字对齐的半字。每条 Thumb 指令要么是该流中的单个 16 位半字，要么是包含该流中两个连续半字的 32 位指令。

如果要解码的半字的位域[15:11]取下面任何值，则该半字是 32 位指令的第一个半字：

(1) 0b11101

(2) 0b11110

(3) 0b11111

否则，半字为 16 位指令。

Thumb-2 指令集由 32 位的 Thumb 指令和最初的 16 位 Thumb 指令组成，与 32 位的 ARM 指令集相比，代码长度减少了 26%，但保持了相似的运行性能。

Cortex-M0＋采用了 ARMv6-M 的结构，将电路规模降低到最小，它采用了 16 位 Thumb-1 的超集，以及 32 位 Thumb-2 的最小子集。

Cortex-M0＋支持的 16 位 Thumb 指令，如图 5.2 所示。

ADCS	ADDS	ADR	ANDS	ASRS	B	BIC	BLX	BKPT	BX
CMN	CMP	CPS	EORS	LDM	LDR	LDRH	LDRSH	LDRB	LDRSB
LSLS	LSRS	MOV	MVN	MULS	NOP	ORRS	POP	PUSH	REV
REV16	REVSH	ROR	RSB	SBCS	SEV	STM	STR	STRH	STRB
SUBS	SVC	SXTB	SXTH	TST	UXTB	UXTH	WFE	WFI	YIELD

图 5.2　Cortex-M0＋支持的 16 位 Thumb 指令

Cortex-M0＋支持的 32 位 Thumb 指令，如图 5.3 所示。

BL	DSB	DMB	ISB	MRS	MSR

图 5.3　Cortex-M0＋支持的 32 位 Thumb 指令

思考与练习 5-1：请说明 Thumb 指令集的主要特点。

思考与练习 5-2：请说明 Thumb-1 指令集和 Thumb-2 指令集的区别。

视频讲解

5.2　Keil MDK 汇编语言指令格式要点

本节介绍在 Keil μVision 中编写汇编语言程序的一些最基本知识，帮助读者学习汇编语言助记符指令的书写规则。

5.2.1　汇编语言源代码中的文字

汇编语言源代码可以包含数字、字符串、布尔值和单个字符文字。文字可以表示为：

(1) 十进制数，例如 123。

(2) 十六进制数，例如 0x7B。

(3) 2～9 的任何进制的数字。比如，5_204 表示五进制数 204，即

$$2\times5^2+0\times5^1+4\times5^0=54$$

等效于十进制数的 54。很明显，在采用五进制的数值表示中，有效的数字范围为 0～4，这个规则适用于采用其他进制的数值表示，即在采用 n 进制的数值表示中，有效的数字范围为 0～$n-1$。

(4) 浮点数，例如 123.4。

(5) 布尔值{TRUE}或{FALSE}。

(6) 单引号引起来的单个字符值，例如'W'。

(7) 双引号引起来的字符串，例如"This is a string"。

注：在大多数情况下，将包含单个字符的字符串看作单个字符值。例如，接受“ADD r0，r1，#"a"”，但是不接受“ADD r0，r1，#"ab"”。

此外还可以使用变量和名字来表示文字。

5.2.2　汇编语言源代码行的语法

汇编器解析并对汇编语言进行汇编以生成目标代码。如图 5.4 所示，Keil MDK 汇编语言源文件中的每一行代码都遵循下面的通用格式：

```
{symbol} {instruction|directive|pseudo - instrunction} {;comment}
```

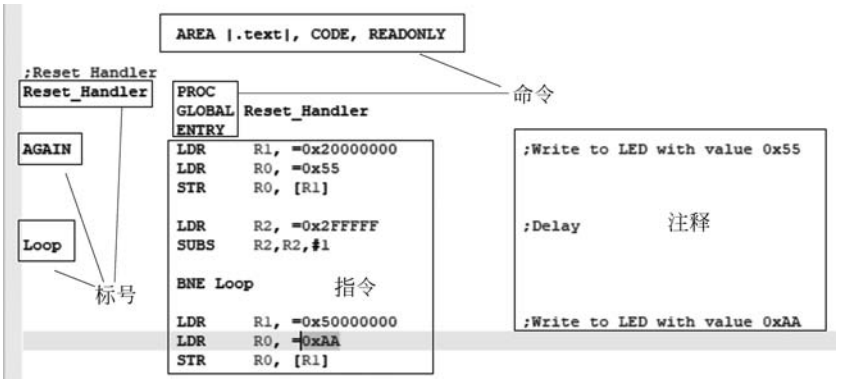


图 5.4　Keil MDK 汇编语言源代码行的格式

其中：

(1) {}表示可选部分，也就是说，每行代码的 3 个用符号{}括起来的部分都是可选的。

(2) symbol。通常是一个标号。在指令和伪指令中，它总是一个标号。在一些命令中，它是变量或常量的符号。

符号必须从第一列开始。除非用竖线(|)括起来，否则它不能包含空格或制表符之类的任何空格字符。

标号是地址的符号表示。可以使用标号来标记要从代码其他部分引用的特定地址。数字局部标签是标签的子类，标签的子类以 0～99 的数字开头。与其他标号不同，数字局部标号可以多次定义。这使得它们在使用宏生成标号时非常有用。

(3) instruction(指令)、directive(命令)、pseudo-instrunction(伪指令)。

① 命令用于指导汇编器和链接器对汇编语言程序的理解和处理，这些信息会影响汇编的过程或影响最终输出的镜像。命令并不能转换为机器指令(机器码)。

② 指令是指机器指令(机器码)，在汇编语言程序设计中使用汇编语言助记符指令来描述机器指令。本质上，汇编语言程序代码主要是由汇编语言助记符指令构成的。通过使用汇编器和链接器对汇编语言助记符指令进行汇编和链接，转换为可以在 Cortex-M0＋处理器上执行的机器指令，以实现和完成特定的任务。如图 5.5 所示，使用汇编器对图 5.4 给出的汇编源代码进行汇编后，生成的机器指令序列。

注：本章只详细介绍汇编语言助记符指令，而在第 6 章将详细介绍汇编语言助记符命令。

由图 5.4 和图 5.5 可知，一条汇编语言助记符指令由操作码和操作数构成。操作码说明指令要执行的操作类型，而操作数说明执行指令时，所需要操作对象的来源。操作数(操作对象)的来源可能是立即数、寄存器或存储器，这与每条指令的寻址方式有关。

(4) comment。注释是源代码行的最后一部分。每行上的第一个分号标记注释的开头，除非分号出现在字符串文字内。该行的结尾是注释的结尾。在代码行中，允许仅存在注释。在对汇编源代码进行汇编时，汇编器将忽略所有注释。可以加入空行使代码更具可读性。

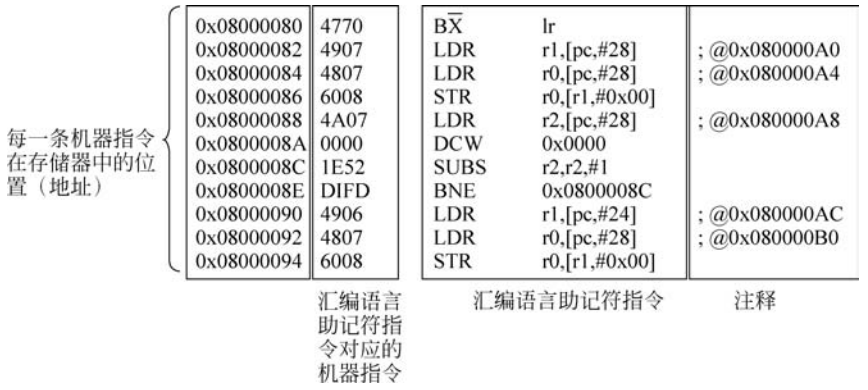


图 5.5　对图 5.4 给出的汇编语言源代码进行汇编后的结果

注：(1) 在 Keil μVision 软件开发工具中，输入指令助记符、伪指令、命令和符号寄存器名字时，要么全部都使用大写字母，要么全部使用小写字母，不允许使用大小写混合的形式。标号和注释可以大写、小写或者大小写混合。

(2) 为了更容易阅读汇编源文件代码，可以通过在每行的默认放置反斜杠字符(\)，将一行中很长的源代码分成几行。反斜杠后不能跟任何其他字符，包括空格和制表符。汇编器将反斜杠和后面跟随的行结束序列看作空白。此外，还可以加入空行使代码更具可读性。

5.2.3　汇编语言指令后缀的含义

对于 Cortex-M0＋的一些汇编助记符指令来说，需要在其后面添加后缀，如表 5.1 所示。

表 5.1　后缀及含义

后　缀	标　志	含　义
S	—	更新 APSR（标志）
EQ	Z=1	等于
NE	Z=0	不等于
CS/HS	C=1	高或者相同，无符号
CC/LO	C=0	低，无符号
MI	N=1	负数
PL	N=0	正数或零
VS	V=1	溢出
VC	V=0	无溢出
HI	C=1 和 Z=0	高，无符号
LS	C=0 或 Z=1	低或者相同，无符号
GE	N=V	大于或等于，有符号
LT	N!=V	小于，有符号
GT	Z=0 和 N=V	大于，有符号
LE	Z=1 和 N!=V	小于或等于，有符号

5.3　寄存器说明符的限制规则

本节介绍使用 0b1111 作为寄存器说明符和使用 0b1101 作为寄存器说明符的一些规则。

5.3.1 使用 0b1111 作为寄存器说明符的规则

Thumb 指令通常不允许将 0b1111 用作寄存器说明符。当允许寄存器的值为 0b1111 时，可能有多个含义。对于寄存器读，其含义为：

(1) 读取 PC 的值，当前指令的地址加上 4。某些指令不使用寄存器说明符而隐含读取 PC 值，例如条件分支指令 B<c>。

(2) 读取字对齐的 PC 值，即当前指令的地址+4，将[1:0]位强制设置为 0。这使得 ADR 和 LDR(literal)指令可以使用 PC 相对数据寻址。这些指令的 ARMv6-M 编码中隐含了寄存器说明符。

对于寄存器写，其含义为：

(1) 可以将 PC 指定为指令的目标寄存器。Thumb 交互工作定义了是忽略地址的位[0]还是确定执行指令的状态。如果它在分支之后选择执行状态，则位[0]的值必须为 1。

指令可以隐含写入 PC，比如 B<cond>，也可以使用寄存器掩码而不是寄存器说明符(POP)来编写。要跳转的地址可以是加载的值(例如 POP)、寄存器的值(比如 BX)或计算结果(例如 ADD)。

(2) 丢弃计算结果。在某些情况下，当一条指令是另一条更通用指令的特殊情况，但丢弃结果时，可以执行该操作。在这些情况下，在单独的页面上列出指令，在伪代码中有特殊情况，用于更通用的指令交叉引用另一页面。该方法不适用于 ARMv6-M 编码。

5.3.2 使用 0b1101 作为寄存器说明符的规则

在 Thumb 指令集中定义了 R13，它主要用作堆栈指针，使 R13 与 ARM 架构过程调用标准(ARM Architecture Procedure Call Standard，AAPCS)(PUSH 和 POP 指令支持的架构使用模型)保持一致。

1. R13<1:0>的定义

R13 的位[1:0]看作应为零或者保留(Should Be Zero or Preserved，SBZP)，向位[1:0]写入非零的值会导致无法预料的行为。读取位[1:0]将返回零。

2. R13 指令支持

ARMv6-M 中的 R13 指令支持仅限于以下各项：

(1) R12 作为 MOV(寄存器)指令的源或目标寄存器，比如

```
MOV SP, Rm
MOV Rd, SP
```

(2) 通过对齐的倍数向上或向下调整 R13，比如

```
SUB(SP 减去立即数)
ADD(SP 加上立即数)
ADD(SP 加上寄存器)                    //Rm 是 4 的倍数
```

(3) R13 作为 ADD(SP+寄存器)的第一个操作数<Rm>，其中 Rd 不是 SP。

(4) R13 作为 CMP(寄存器)指令中的第一个操作数<Rn>。CMP 对于检查堆栈非常有用。

(5) R13 作为 POP 或 PUSH 指令中的地址。

限制影响：

(1) 不推荐使用 ADD(寄存器)和 CMP(寄存器)的高寄存器形式，不建议使用 R13 作

为 Rd。

(2) ADD(SP＋寄存器)，其中 Rd 等于 13，且 Rm 没有字对齐。

视频讲解

5.4 寄存器传输指令

本节介绍写寄存器指令，下面对这些指令进行详细说明。

1. MOVS ＜Rd＞，＃imm8

该指令将立即数 imm8(范围 0～255)写到寄存器 Rd 中，该汇编助记符指令的机器码格式如图 5.6 所示。从该指令格式的机器码可知，Rd 寄存器的使用范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rd			imm8							

图 5.6 MOVS＜Rd＞，＃imm8 指令的机器码格式

指令 MOVS R1，＃0x14 的机器码表示为$(2114)_{16}$。该指令将立即数 0x14 的内容写到寄存器 R1 中。并且更新 APSR 寄存器中的标志 N、Z 和 C，但不修改标志 V。

注：(2114)的下标 16 表示 2114 为十六进制数。

2. MOV ＜Rd＞，＜Rm＞

该指令将寄存器 Rm 的内容写到寄存器 Rd 中，该汇编助记符指令的机器码格式如图 5.7 所示。从该指令的机器码格式可知，Rm 可用的寄存器范围为 R0～R15，Rd 可用的寄存器范围为 R0～R7。当 Rd 和 D 一起使用时，即{D，Rd}，可用寄存器范围扩展到 R0～R15。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	D	Rm				Rd		

图 5.7 MOV＜Rd＞，＜Rm＞指令的机器码格式

指令“MOV R0，SP”的机器码为$(4668)_{16}$，将 SP 的内容写到寄存器 R0 中，不更新 APSR 寄存器中的标志。

3. MOVS ＜Rd＞，＜Rm＞

该指令将寄存器 Rm 的内容复制到寄存器中 Rd，并且更新 APSR 寄存器中的 Z 和 N 标志，该汇编助记符指令的机器码格式如图 5.8 所示。从该指令的机器码可知，Rm 可用的寄存器范围为 R0～R7，Rd 可用的寄存器范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	Rm			Rd		

图 5.8 MOVS＜Rd＞,＜Rm＞指令的机器码格式

指令“MOVS R0，R1”的机器码为$(0008)_{16}$。该指令将寄存器 R1 的内容复制到寄存器 R0 中，并且更新 APSR 寄存器中的标志 N 和 Z。

4. MRS ＜Rd＞，＜SpecialReg＞

该指令是 32 位的 Thumb 指令，将由 SpecialReg 标识的特殊寄存器(如表 5.2 所示)内容写到寄存器 Rd 中。该指令不影响 APSR 寄存器中的任何标志位，该汇编助记符指令的机器码格式如图 5.9 所示。从该指令的机器码格式可知，可用的 Rd 寄存器的范围为 R0～R15。

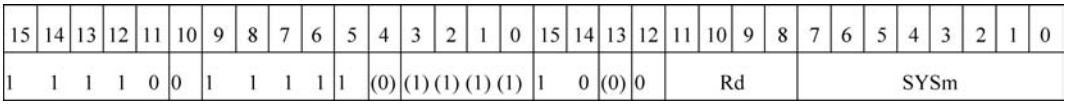


图 5.9　MRS < Rd >, < SpecialReg >指令的机器码格式

SYSm 的定义如表 5.2 所示。

表 5.2　SYSm 的定义

SYSm 的编码	7	6	5	4	3	2	1	0
IPSR	0	0	0	0	0	1	0	1
EPSR						1	1	0
APSR						0	0	0
PSR						0	1	1
IEPSR						1	1	1
IAPSR						0	0	1
EAPSR						0	1	0
MSP	0	0	0	0	1	0	0	0
PSP						0	0	1
PRIMASK	0	0	0	1	0	0	0	0
CONTROL						1	0	0

指令“MRS R1, CONTROL”的机器码为$(F3EF8514)_{16}$。该指令将 CONTROL 寄存器的内容写到寄存器 R1 中。

5. MSR ＜SpecialReg＞, ＜Rn＞

该指令将寄存器 Rn 中的内容写到 SpecialReg 标识的特殊寄存器中，该指令影响 APSR 寄存器中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.10 所示。从该指令的机器码格式可知，可用的 Rn 寄存器的范围为 R0～R15。SYSm 的编码格式如表 5.2 所示。

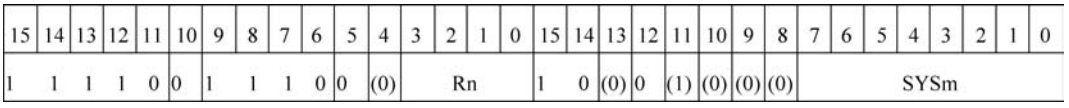


图 5.10　MSR < SpecialReg >,< Rn >指令的机器码格式

指令“MSR APSR, R0”的机器码为$(F3808800)_{16}$，将寄存器 R0 的内容写到 APSR 寄存器中。

思考与练习 5-3：说明下面指令所实现的功能。

(1) MOVS R0,＃0x000B _____

(2) MOVS R1,＃0x0 _____

(3) MOV R10,R12 _____

(4) MOVS R3,＃23 _____

(5) MOV R8,SP _____

5.5　存储器加载和保存指令

本节介绍存储器加载和保存指令。

视频讲解

5.5.1 存储器加载指令

下面对存储器加载指令进行详细说明。这些指令不会影响 APSR 寄存器中的任何标志位。

1. LDR <Rt>, [<Rn>,<Rm>]

该指令从[<Rn>+<Rm>]寄存器所指向存储器的地址中，取出一个字(32 位)，并将其写到寄存器 Rt 中，该汇编助记符指令的机器码格式如图 5.11 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	0	Rm			Rn			Rt		

图 5.11 LDR <Rt>, [<Rn>,<Rm>]指令的机器码格式

指令 LDR R0, [R1, R2]的机器码为$(5888)_{16}$。该指令从[R1+R2]所指向存储器的地址中，取出一个字(32 位)，并将其写到寄存器 R0 中。

2. LDRH <Rt>, [<Rn>,<Rm>]

该指令从[<Rn>+<Rm>]寄存器所指向存储器的地址中，取出半个字(16 位)，将其写到寄存器 Rt 的[15:0]位中，并将寄存器 Rt 的[31:16]位清零，该汇编助记符指令的机器码格式如图 5.12 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	Rm			Rn			Rt		

图 5.12 LDRH <Rt>, [<Rn>,<Rm>]指令的机器码格式

指令 LDRH R0, [R1, R2]的机器码为$(5A88)_{16}$。该指令从[R1+R2]所指向存储器的地址中，取出半个字(16 位)，将其写到寄存器 R0 的[15:0]位中。

3. LDRB <Rt>, [<Rn>,<Rm>]

该指令从[<Rn>+<Rm>]寄存器所指向存储器的地址中，取出单字节(8 位)，将其写到寄存器 Rt 的[7:0]位中，并将寄存器 Rt 的[31:8]位清零，该汇编助记符指令的机器码格式如图 5.13 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	0	Rm			Rn			Rt		

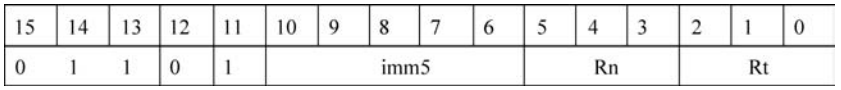
图 5.13 LDRB <Rt>, [<Rn>,<Rm>]指令的机器码格式

指令 LDRB R0, [R1, R2]的机器码为$(5C88)_{16}$。该指令从[R1+R2]寄存器所指向存储器中，取出 1 字节(8 位)，并将其写到寄存器 R0 的[7:0]位中。

4. LDR <Rt>, [<Rn>, #imm]

该指令从[<Rn> +imm]指向存储器的地址中，取出一个字(32 位)，并将其写到寄存器 Rt 中，该汇编助记符指令的机器码格式如图 5.14 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～124(7 位二进制数)，并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	imm5					Rn			Rt		

图 5.14　LDR < Rt >, [< Rn >, # imm]指令的机器码格式

符指令中给出的立即数 imm 右移两位，然后得到该汇编指令中立即数在机器码中的编码格式为 imm5，即为 5 位立即数。

(2) 将汇编语言助记符指令中的立即数 imm 零扩展到 32 位(imm5 << 2)。

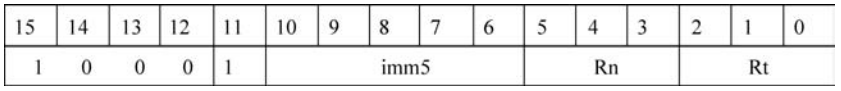
指令 LDR R0, [R1, #0x7C]的机器码为$(6FC8)_{16}$，该指令从[R1+0x7C]所指向存储器的地址中，取出一个字(32 位)，并将其写到寄存器 R0 中。

注：对于汇编指令中给出的立即数 0x7C，在将其转换为机器码格式时，右移两位，变成 0x1F，因此机器指令的格式为$(6FC8)_{16}$。

指令 LDR R0,[R1, #0x80]中的立即数 0x80 超过范围，因此该指令是错误的。指令 LDR R0,[R1, #0x7E]中的立即数 0x7E 没有字对齐(不是 4 的整数倍)，因此该指令也是错误的。

5. LDRH < Rt >, [< Rn >, # imm]

该指令从[< Rn > +imm]指向存储器的地址中，取出半个字(16 位)，并将其写到寄存器 Rt 的[15:0]位中，用零填充[31:16]位。该汇编助记符指令的机器码格式如图 5.15 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	1	imm5					Rn			Rt		

图 5.15　LDRH < Rt >, [< Rn >, # imm]指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～62(6 位二进制数)，并且该立即数是 2 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 1 位，然后得到该汇编指令中立即数在机器码中的编码格式为 imm5，即为 5 位立即数。

(2) 将该汇编语言助记符指令中的立即数 imm 零扩展到 32 位(imm5 << 1)。

指令 LDRH R0, [R1, #0x3E]的机器码为$(8FC8)_{16}$，该指令从[(R1)+(0x3E)]指向存储器的地址中，取出半个字(16 位)，并将其写到寄存器 R0 的[15:0]中，[31:16]用零填充。

注：对于汇编指令中给出的立即数 0x3E，在将其转换为机器码格式时，右移一位，变成 0x1F，因此机器指令的格式为$(8FC8)_{16}$。

指令 LDRH R0,[R1, #0x40]中的立即数 0x40 超过范围，因此该指令是错误的。指令 LDRH R0,[R1, #0x3F]中的立即数 0x3F 没有半字对齐，因此该指令也是错误的。

6. LDRB < Rt >, [< Rn >, # imm]

该指令从[< Rn > +imm]指向存储器的地址中，取出单字节(8 位)，并将其写到寄存器 Rt 的[7:0]位中，用零填充[31:8]位。该汇编助记符指令的机器码格式如图 5.16 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。

注：(1) 该汇编语言助记符指令中允许的立即数 imm5 的范围为 0～31(5 位二进制数)。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展的立即数 imm5。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	imm5					Rn			Rt		

图 5.16　LDRB＜Rt＞,[＜Rn＞,＃imm]指令的机器码格式

指令 LDRB R0,[R1,＃0x1F]的机器码为$(7FC8)_{16}$,该指令从[(R1)＋0x1F]指向存储器的地址中,取出单字节,并将其写到寄存器 Rt 的[7:0]中。

7. LDR＜Rt＞,＝立即数

该指令将立即数加载到寄存器 Rt 中。实际上,在 Keil μVision 编译器对该指令进行编译处理时,会转换为 LDR＜Rt＞,[pc,＃imm]的形式。因此,没有该指令的机器码编码格式,该指令为伪指令。该指令的存在只是方便编程人员直接对存储器空间的绝对地址进行操作而已。

注:该伪指令生成最有效的单个指令以加载任何 32 位数字。可以使用该伪指令生成超出 MOV 和 MVN 指令范围的常量。

指令 LDR R0,＝0x12345678 将立即数 0x12345678 的值加载到寄存器 R0 中。

8. LDR＜Rt＞,[PC,＃imm]

该指令从[PC＋imm]指向存储器的地址中,取出一个字,并将其写到寄存器 Rt 中,该汇编助记符指令的机器码格式如图 5.17 所示。可以看出,Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	Rt			imm8							

图 5.17　LDR＜Rt＞,[PC,＃imm]指令的机器码格式

注:(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～1020(10 位二进制数),并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时,将汇编语言助记符指令中给出的立即数 imm 右移 2 位,然后得到该汇编指令中立即数在机器码中的编码格式为 imm8,即为 8 位立即数。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm8＜＜2)。

指令 LDR R7,[PC,＃0x44]的机器码为$(4F11)_{16}$,该指令从[(PC)＋0x44]指向存储器的地址中,取出一个字,并将其写到寄存器 R7 中。

注:对于汇编指令中给出的立即数 0x44,在将其转换为机器码格式时,右移两位,变成 0x11,因此机器指令的格式为$(4F11)_{16}$。

9. LDR＜Rt＞,[SP,＃imm]

该指令从[SP＋imm]指向存储器的地址中,取出一个字,并将其写到寄存器 Rt 中,该汇编助记符指令的机器码格式如图 5.18 所示。可以看出,Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	1	Rt			imm8							

图 5.18　LDR＜Rt＞,[SP,＃imm]指令的机器码格式

注:(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～1020(10 位二进制数),并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时,将汇编语言

助记符指令中给出的立即数 imm 右移 2 位，然后得到该汇编指令中立即数在机器码中的编码格式为 imm8，即为 8 位立即数。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm8 << 2)。

指令 LDR R0，[SP，＃0x68]的机器码为$(981A)_{16}$，该指令从[(SP)＋0x68]指向存储器的地址中，取出一个字，并将其写到寄存器 R0 中。

注：对于汇编指令中给出的立即数 0x68，在将其转换为机器码格式时，右移两位，变成 0x1A，因此机器指令的格式为$(981A)_{16}$。

10. LDRSH ＜Rt＞，[＜Rn＞,＜Rm＞]

该指令从[Rn＋Rm]所指向的存储器中取出半个字(16 位)，并将其写到 Rt 寄存器的[15:0]位中。对于[31:16]来说，取决于第[15]位，采用符号扩展。当该位为 1 时，[31:16]各位均用 1 填充；当该位为 0 时，[31:16]各位均用 0 填充，该汇编助记符指令的机器码格式如图 5.19 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	1	1	1	Rm			Rn			Rt		

图 5.19　LDRSH ＜Rt＞，[＜Rn＞,＜Rm＞]指令的机器码格式

指令 LDRSH R0，[R1，R2]的机器码格式为$(5E88)_{16}$。该指令从[R1＋R2]所指向的存储器地址中取出半个字(16 位)，并将其写到 R0 寄存器的[15:0]位中，同时进行符号扩展。

11. LDRSB ＜Rt＞，[＜Rn＞,＜Rm＞]

从[Rn＋Rm]所指向存储器的地址中取出单字节(8 位)，并将其写到 Rt 寄存器[7:0]中。对于[31:8]来说，取决于第[7]位，采用符号扩展。当该位为 1 时，[31:8]各位均用 1 填充；当该位为 0 时，[31:8]各位均用 0 填充，该汇编助记符指令的机器码格式如图 5.20 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	1	Rm			Rn			Rt		

图 5.20　LDRSB ＜Rt＞，[＜Rn＞,＜Rm＞]指令的机器码格式

指令 LDRSB R0，[R1，R2]的机器码格式为$(5688)_{16}$。该指令从[R1＋R2]所指向的存储器中取出单字节(8 位)，并将其写到 R0 寄存器的[7:0]位中，同时进行符号扩展。

思考与练习 5-4：说明下面指令所实现的功能。

(1) LDR R1，＝0x54000000＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

(2) LDRSH R1，[R2，R3]＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

(3) LDR R0，LookUpTable＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

(4) LDR R3，[PC，＃100]＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿＿

5.5.2　存储器保存指令

本节介绍存储器保存指令，下面对这些指令进行详细说明。这些指令不影响 APSR 寄存器中的任何标志位。

1. STR ＜Rt＞，[＜Rn＞,＜Rm＞]

该指令将 Rt 寄存器中的字数据写到 [＜Rn＞＋＜Rm＞]所指向存储器的地址单元中，该汇编助

记符指令的机器码格式如图 5.21 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	Rm			Rn			Rt		

图 5.21　STR < Rt >, [< Rn >,< Rm >]指令的机器码格式

指令 STR R0，[R1，R2]的机器码格式为$(5088)_{16}$。该指令将 R0 寄存器中的字数据写到 [R1＋R2]所指向存储器地址单元中。

2. STRH ＜Rt＞，[＜Rn＞,＜Rm＞]

该指令将 Rt 寄存器的半字，即[15:0]位写到 [< Rn >+< Rm >]所指向存储器的地址单元中，该汇编助记符指令的机器码格式如图 5.22 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	1	Rm			Rn			Rt		

图 5.22　STRH < Rt >, [< Rn >,< Rm >]指令的机器码格式

指令 STRH R0，[R1，R2]的机器码格式为$(5288)_{16}$。该指令将 R0 寄存器中的[15:0]位数据写到 [R1＋R2]所指向存储器的地址单元中。

3. STRB ＜Rt＞，[＜Rn＞,＜Rm＞]

该指令将 Rt 寄存器的字节，即：[7:0]位写到 [< Rn >+< Rm >]所指向存储器的地址单元中，该汇编助记符指令的机器码格式如图 5.23 所示。可以看出，Rm、Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	1	0	Rm			Rn			Rt		

图 5.23　STRB < Rt >, [< Rn >,< Rm >]指令的机器码格式

指令 STRB R0，[R1，R2]的机器码格式为$(5488)_{16}$。该指令将 R0 寄存器中的[7:0]位写到 [R1＋R2]所指向存储器的地址单元中。

4. STR ＜Rt＞，[＜Rn＞，＃imm]

该指令将 Rt 寄存器的字数据写到[< Rn > +imm]所指向存储器地址的单元中，该汇编助记符指令的机器码格式如图 5.24 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	imm5					Rn			Rt		

图 5.24　STR < Rt >, [< Rn >, ＃imm]指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～124(7 位二进制数)，并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 2 位(零扩展)，然后得到该汇编指令中立即数在机器码中的编码格式为 imm5，即为 5 位立即数。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm5 << 2)。

指令 STR R0，[R1，＃0x44]的机器码格式为$(6448)_{16}$，该指令将 R0 寄存器的字写到[R1＋0x44]所指向存储器地址的单元中。

注：对于汇编指令中给出的立即数 0x44，在将其转换为机器码格式时，右移两位，变成 0x11，因此机器指令的格式为$(6448)_{16}$。

5. STRH ＜Rt＞，[＜Rn＞，＃imm]

该指令将 Rt 寄存器的半字数据，即[15:0]位写到[＜Rn＞＋imm]所指向存储器地址的单元中。该汇编助记符指令的机器码格式如图 5.25 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	0	0	imm5					Rn			Rt		

图 5.25　STRH ＜Rt＞，[＜Rn＞，＃imm]指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～62(6 位二进制数)，并且该立即数是 2 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 1 位(零扩展)，然后得到该汇编指令中立即数在机器码中的编码格式为 imm5，即为 5 位立即数。

(2) 将该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm5 ＜＜1)。

指令 STRH R0，[R1，＃0x2]的机器码格式为$(8048)_{16}$。该指令将 R0 寄存器的半字，即[15:0]位写到[R1＋0x2]所指向存储器地址的单元中。

注：对于汇编指令中给出的立即数 0x2，在将其转换为机器码格式时，右移一位，变成 0x1，因此机器指令的格式为$(8048)_{16}$。

6. STRB ＜Rt＞，[＜Rn＞，＃imm]

该指令将 Rt 寄存器的字节数据写到[＜Rn＞＋imm]所指向存储器的单元中。该汇编助记符指令的机器码格式如图 5.26 所示。可以看出，Rn 和 Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	imm5					Rn			Rt		

图 5.26　STRB ＜Rt＞，[＜Rn＞，＃imm]指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～31(5 位二进制数)。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm5)。

指令 STRB R0，[R1，＃0x1]的机器码格式为$(7048)_{16}$。该指令将 R0 寄存器的字节[7:0]位写到[R1＋0x01]所指向存储器地址的单元中。

7. STR ＜Rt＞，[SP，＃imm]

该指令将 Rt 寄存器中的字数据写到[SP＋imm]所指向存储器地址的单元中。该汇编助记符指令的机器码格式如图 5.27 所示。可以看出，Rt 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	0	Rt			imm8							

图 5.27　STR ＜Rt＞，[SP，＃imm]指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～1020(10 位二进制数)，并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 2 位(零扩展)，然后得到该汇编指令中立即数在机器码中的编码格式为 imm8，即为 8 位立即数。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm8 << 2)。

指令 STR R0，[SP，＃0x8]的机器码格式为$(9002)_{16}$。该指令将将 R0 寄存器的字数据写到[SP＋(0x8)]所指向存储器地址的单元中。

注：对于汇编指令中给出的立即数 0x8，在将其转换为机器码格式时，右移两位，变成 0x2，因此机器指令的格式为$(9002)_{16}$。

思考与练习 5-5：说明下面指令所实现的功能。

(1) STR R0，[R5，R1]_____

(2) STR R2，[R0，＃const-struc]_____

视频讲解

5.6 多数据加载和保存指令

本节介绍多数据加载和保存指令。

5.6.1 多数据加载指令

本节介绍多数据加载指令，包括 LDM、LDMIA 和 LDMFD。它们使用基址寄存器中的地址从连续的存储器位置中加载多个寄存器后，加载多个增量。连续的存储器位置从该地址开始，并且当基址寄存器 Rn 不属于< registers >列表时，这些位置的最后一个地址之上的地址将写回到基址寄存器。

下面对 LDM 指令进行详细说明，该指令有两种形式，包括：

```
LDM < Rn >!,< registers >          < registers >中不包含< Rn >
LDM < Rn >,< registers >           < registers >中包含< Rn >
```

注：(1) !的作用是：使指令将修改后的值写回到< Rn >。如果没有!，则指令不会以这种方式修改< Rn >。

(2) registers 是要加载的一个或多个寄存器的列表，用逗号分隔，并且用{}符号包围。编号最小的寄存器从最低的存储器地址加载，直到从最高存储器地址到编号最高的寄存器。

该汇编助记符指令的机器码格式如图 5.28 所示。可以看出，Rn 所使用寄存器的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	1	Rn			register_list							

图 5.28 LDM 指令的机器码格式

注：图中 register_list 字段中的每一个比特位对应一个寄存器。register_list[7]对应寄存器 R7；register_list[6]对应寄存器 R6；register_list[5]对应寄存器 R5；register_list[4]对应寄存器 R4；register_list[3]对应寄存器 R3；register_list[2]对应寄存器 R2；register_list[1]对应寄存器 R1；register_list[0]对应寄存器 R0。当< register >存在某个寄存器时，register_list 字段中对应的位置为 1，否则为 0。

指令 LDM R0!，{R1，R2－R7}的机器码格式为$(C8FE)_{16}$。该指令实现下面的功能：

(1) 将寄存器 R0 所指向存储器地址单元的内容复制到寄存器 R1 中。

(2) 将寄存器 R0+4 所指向存储器地址单元的内容复制到寄存器 R2 中。

(3) 将寄存器 R0+8 所指向存储器地址单元的内容复制到寄存器 R3 中。

……

(7) 将寄存器 R0+24 所指向存储器地址单元的内容复制到寄存器 R7 中。

(8) 用 R0+4×7 的值更新寄存器 R0 的内容。

注：LDMIA 和 LDMFD 为 LDM 指令的伪指令。LDMIA 是 LDM 指令的别名，用法同 LDM。LDMFD 是同一指令的另一个名字，在使用软件管理堆栈的传统 ARM 系统中，该指令用于从全递减堆栈中还原数据。

5.6.2　多数据保存指令

本节介绍多数据保存指令，包括 STM、STMIA 和 STMEA。使用来自基址寄存器的地址将多个寄存器的内容保存到连续的存储器位置。随后的存储器位置从该地址开始，并且这些位置的最后一个地址之上的地址将写回到基址寄存器。指令格式如下：

```
STM < Rn >!, < registers >
```

注：(1) !的作用是：使指令将修改后的值写回到< Rn >。

(2) registers 是要加载的一个或多个寄存器的列表，用逗号分隔，并且用{}符号包围。编号最小的寄存器从最低的存储器地址加载，直到从最高存储器地址到编号最高的寄存器。不推荐在寄存器列表中使用< Rn >。

该汇编助记符指令的机器码格式如图 5.29 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	Rn			register_list							

图 5.29　STM < Rn >!, < registers >指令的机器码格式

注：图中 register_list 的具体含义同图 5.28。

指令 STM R0!, {R1, R2－R7}的机器码格式为$(C0FE)_{16}$。该指令实现下面的功能：

(1) 将寄存器 R1 的内容复制到寄存器 R0 所指向存储器地址的单元中。

(2) 将寄存器 R2 的内容复制到寄存器 R0+4 所指向存储器地址的单元中。

(3) 将寄存器 R3 的内容复制到寄存器 R0+8 所指向存储器地址的单元中。

……

(7) 将寄存器 R7 的内容复制到寄存器 R0+24 所指向存储器地址的单元中。

(8) 用 R0+7×4 的值更新 R0 寄存器的值。

思考与练习 5-6：说明下面指令所实现的功能。

(1) LDM R0,{R0,R3,R4}_____

(2) STMIA R1!,{R2－R4,R6}_____

5.7　堆栈访问指令

视频讲解

本节介绍堆栈访问指令，包括 PUSH 和 POP，下面对这些指令进行详细说明。

1. PUSH < registers >

该指令将一个/多个寄存器(包括 R0～R7 以及 LR)保存到堆栈(入栈)，并且更新堆栈指

针寄存器，该汇编助记符指令的机器码格式如图 5.30 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	0	M	register_list							

图 5.30　PUSH < registers >指令的机器码格式

注：(1) registers 是一个或多个寄存器的列表，用逗号分隔，并且用{}符号包围。它标识了将要保存的寄存器集，按顺序保存。编号最小的寄存器到最低的存储器地址，一直到编号最大的寄存器到最高的存储器地址。registers＝'0':M:'000000':register_list。

(2) 图中 register_list 的具体含义同图 5.28。

指令 PUSH{R0,R1,R2}，该指令机器码的格式为$(B407)_{16}$，该指令实现下面功能：

(1) (sp)＝(sp)－4×进入堆栈的寄存器的个数。

(2) 将寄存器 R0 的内容保存到(sp)所指向的存储器的地址。

(3) 将寄存器 R1 的内容保存到(sp)＋4 所指向的存储器的地址。

(4) 将寄存器 R2 的内容保存到(sp)＋8 所指向的存储器的地址。

(5) 将 sp 的内容更新到步骤(1)计算得到的 sp 内容。

2. POP ＜registers＞

该指令将存储器中的内容恢复到多个寄存器(包括 R0～R7 以及 PC)中，并且更新堆栈指针寄存器的机器码格式如图 5.31 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	0	P	register_list							

图 5.31　POP < registers >指令的机器码格式

注：(1) registers 是一个或多个寄存器的列表，用逗号分隔，并且用{}符号包围。它标识了将要加载的寄存器集。从最低的存储器地址加载到编号最小的寄存器，从最高的存储器地址加载到编号最大的寄存器。如果在< registers >中指定了 PC，则该指令将导致跳转到 PC 中加载的地址(数据)。

(2) 图中 register_list 的具体含义同图 5.28。registers＝P:'0000000':register_list。

指令 POP{R0,R1,R2}，该指令的机器码为$(BC07)_{16}$，该指令实现下面的功能：

(1) 将(sp)所指向的存储器的内容加载到寄存器 R0。

(2) 将(sp)＋4 所指向的存储器的内容加载到寄存器 R1。

(3) 将(sp)＋8 所指向的存储器的内容加载到寄存器 R2。

(4) 用(sp)＋4×出栈寄存器的个数的值更新寄存器 sp。

思考与练习 5-7：说明下面指令所实现的功能。

(1) PUSH{R0,R4－R7}_____

(2) PUSH{R2,LR}_____

(3) POP{R0,R6,PC}_____

视频讲解

5.8　算术运算指令

本节介绍算术运算指令，包括加法指令、减法指令和乘法指令。

5.8.1　加法指令

1. ADDS ＜Rd＞, ＜Rn＞, ＜Rm＞

该指令将 Rn 寄存器的内容和 Rm 寄存器的内容相加，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志，该汇编助记符指令的机器码格式如图 5.32 所示。可以看出，寄存器 Rm、Rn 和 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	Rm			Rn			Rd		

图 5.32　ADDS ＜Rd＞, ＜Rn＞, ＜Rm＞指令的机器码格式

指令 ADDS R0，R1，R2 的机器码为$(1888)_{16}$。该指令将 R1 寄存器的内容和 R2 寄存器的内容相加，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

2. ADDS ＜Rd＞, ＜Rn＞, ＃imm3

该指令将 Rn 寄存器的内容和立即数 imm3 相加，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志，该汇编助记符指令的机器码格式如图 5.33 所示。可以看出，寄存器 Rn 和 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Rn			Rd		

图 5.33　ADDS ＜Rd＞, ＜Rn＞, ＃imm3 指令的机器码格式

注：图中 imm3 的范围为 0～7。

指令 ADDS R0，R1，＃0x07 的机器码为$(1DC8)_{16}$。该指令将 R1 寄存器的内容和立即数 0x07 相加，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

3. ADDS ＜Rd＞, ＃imm8

该指令将 Rd 寄存器的内容和立即数 imm8 相加，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志，该汇编助记符指令的机器码格式如图 5.34 所示。可以看出，寄存器 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rd			imm8							

图 5.34　ADDS ＜Rd＞, ＃imm8 指令的机器码格式

注：图中 imm8 的范围为 0～255。

指令 ADDS R0，＃0x01 的机器码为$(3001)_{16}$。该指令将 R0 寄存器的内容和立即数 0x01 相加，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

4. ADD ＜Rd＞, ＜Rm＞

该指令将 Rd 寄存器的内容和 Rn 寄存器的内容相加，结果保存在寄存器 Rd 中，不更新寄存器 APSR 中的标志，该汇编助记符指令的机器码格式如图 5.35 所示。可以看出，寄存器 Rm 可用的范围为 R0～R15，DN 与 Rd 组合后可用的范围为 R0～R15。

指令 ADD R0，R1 的机器码为$(4408)_{16}$，该指令将 R0 寄存器的内容和 R1 寄存器的内容相加，结果保存在寄存器 R0 中，不更新寄存器 APSR 中的标志。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	DN	Rm				Rd		

图 5.35　ADD < Rd >, < Rm >指令的机器码格式

5. ADCS ＜Rd＞, ＜Rm＞

该指令将 Rd 寄存器的内容、Rm 寄存器的内容和进位标志相加，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志，该汇编助记符指令的机器码格式如图 5.36 所示。可以看出，寄存器 Rm 和 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	0	1	Rm			Rd		

图 5.36　ADCS < Rd >, < Rm >指令的机器码格式

指令 ADCS R0，R1 的机器码为$(4148)_{16}$。该寄存器将 R0 寄存器的内容、R1 寄存器的内容和进位标志相加，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

6. ADD ＜Rd＞, PC, ＃imm

该指令将 PC 寄存器的内容和立即数＃imm 相加，结果保存在寄存器 Rd 中，不更新寄存器 APSR 中的标志。该汇编助记符指令格式的机器码与指令 ADR < Rd >, < label >相同，如图 5.37 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	Rd			imm8							

图 5.37　ADR < Rd >, < label >指令的机器码格式

注：(1) 在相加的时候，必须将 PC 寄存器的内容与字对齐，即 Align(PC,4)。

(2) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～1020(10 位二进制数)，并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 2 位(零扩展)，然后得到该汇编指令中立即数在机器码中的编码格式为 imm8，即为 8 位立即数。

(3) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm8 << 2)。

指令 ADD R0，PC，＃0x04 的机器码为$(A001)_{16}$。该指令将 PC 寄存器的内容(字对齐)和立即数 0x04 相加，结果保存在寄存器 R0 中，不更新寄存器 APSR 中的标志。

7. ADR ＜Rd＞,＜label＞

该指令将 PC 寄存器的内容与标号所表示的偏移量进行相加，结果保存在寄存器 Rd 中，不更新 APSR 中的标志。< label >为将地址加载到< Rd >中的指令或文字数据项的标号。汇编器计算从 ADR 指令的 Align(PC,4)值到该标签的偏移量所需要的值。偏移量的允许值是 0～1020 范围内的正数(4 的倍数)。

指令 ADR R3，JumpTable 将 JumpTable 的地址加载到寄存器 R3 中。

8. ADD(SP 加立即数指令)

该类型指令有两个指令类型。

1) ADD < Rd >, SP, ＃imm

该指令实现将堆栈指针 SP 的内容和立即数 imm 相加，相加的结果写到寄存器 Rd 中。该汇编助记符指令的机器码格式如图 5.38 所示。可以看出，寄存器 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	Rd			imm8							

图 5.38　ADD < Rd >, SP, ＃imm8 指令的机器码格式

注：(1) 图中 imm8 的含义同图 5.37。

(2) 该指令中 imm 的范围为 0～1020，且 imm 为 4 的整数倍。

指令 ADD R3,SP,＃0x04 的机器码格式为$(AB01)_{16}$，该指令实现将堆栈指针 SP 的内容和立即数 0x04 相加，相加的结果写到寄存器 R3 中。

2) ADD SP,SP,＃imm

该指令实现将堆栈指针 SP 的内容和立即数 imm 相加，相加的结果写到堆栈指针中。该汇编助记符指令的机器码格式如图 5.39 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	0	imm7						

图 5.39　ADD SP,SP,＃< imm >指令的机器码格式

注：(1) 该汇编语言助记符指令中允许的立即数 imm 的范围为 0～508(9 位二进制数)，并且该立即数是 4 的整数倍。只是在对该立即数 imm 进行机器指令编码时，将汇编语言助记符指令中给出的立即数 imm 右移 2 位(零扩展)，然后得到该汇编指令中立即数在机器码中的编码格式为 imm7，即为 7 位立即数。

(2) 该汇编语言助记符指令中的立即数 imm＝零扩展到 32 位(imm7 << 2)。

指令 ADD SP,SP,＃08 的机器码格式为$(B002)_{16}$，该指令实现将堆栈指针 SP 的内容和立即数 0x08 相加，相加的结果写到堆栈指针 SP 中。

5.8.2　减法指令

1. SUBS < Rd >, < Rn >, < Rm >

该指令将寄存器 Rn 的内容减去寄存器 Rm 的内容，结果保存在 Rd 中，同时更新寄存器 APSR 寄存器中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.40 所示。可以看出，寄存器 Rm、Rn 和 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	1	Rm			Rn			Rd		

图 5.40　SUBS < Rd >, < Rn >, < Rm >指令的机器码格式

指令 SUBS R0, R1, R2 的机器码格式为$(1A88)_{16}$。该指令将寄存器 R1 的内容减去寄存器 R2 的内容，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

2. SUBS < Rd >, < Rn >, ＃imm3

该指令将 Rn 寄存器的内容和立即数 imm3 相减，结果保存在寄存器 Rd 中，同时更新寄

存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.41 所示。可以看出，寄存器 Rn 和 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	imm3			Rn			Rd		

图 5.41　SUBS＜Rd＞，＜Rn＞，＃imm3 指令的机器码格式

注：imm3 的范围为 0～7。

指令 SUBS R0，R1，＃0x01 的机器码为$(1E48)_{16}$。该指令将 R1 寄存器的内容和立即数 0x01 相减，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

3. SUBS＜Rd＞，＃imm8

该指令将 Rd 寄存器的内容和立即数＃imm8 相减，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.42 所示。可以看出，寄存器 Rd 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	1	Rd			imm8							

图 5.42　SUBS＜Rd＞，＃imm8 指令的机器码格式

注：imm8 的范围为 0～255。

指令 SUBS R0，＃0x01 的机器码为$(3801)_{16}$。该指令将 R0 寄存器的内容和立即数 0x01 相减，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

4. SBCS＜Rd＞，＜Rm＞

该指令将 Rd 寄存器的内容、Rm 寄存器的内容和借位标志相减，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.43 所示。可以看出，寄存器 Rd 和 Rm 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	0	Rm			Rd		

图 5.43　SBCS＜Rd＞，＜Rm＞指令的机器码格式

指令 SBCS R0，R0，R1 的机器码为$(4188)_{16}$。该寄存器将 R0 寄存器的内容、R1 寄存器的内容和借位标志相减，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

5. RSBS＜Rd＞，＜Rn＞，＃0

该指令用数字 0 减去寄存器 Rn 中的内容，结果保存在寄存器 Rd 中，并且更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.44 所示。可以看出，寄存器 Rd 和 Rn 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	0	1	Rn			Rd		

图 5.44　RSBS＜Rd＞，＜Rm＞，＃0 指令的机器码格式

指令 RSBS R0，R0，＃0 的机器码为$(4240)_{16}$。该指令用数字 0 减去寄存器 R0 中的内容，结果保存在寄存器 R0 中，并且更新寄存器 APSR 中的标志。

6. SUB SP，SP，＃imm

该指令将堆栈指针 SP 的内容减去立即数 imm，结果保存在堆栈指针 SP 中。该指令不影响 APSR 中的标志。该汇编助记符指令的机器码格式如图 5.45 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	0	0	1	imm7						

图 5.45　SUB SP，SP，＃imm 指令的机器码格式

注：图中 imm7 的含义同图 5.39。

指令 SUB SP，SP，＃0x18 的机器码格式为$(B086)_{16}$，该指令实现将堆栈指针 SP 的内容和立即数 0x18 相减，相减的结果写到堆栈指针 SP 中。

5.8.3　乘法指令

MULS ＜Rd＞，＜Rn＞，＜Rd＞

该指令将寄存器 Rn 的内容和寄存器 Rd 的内容相乘，低 32 位结果保存在 Rd 寄存器中，同时更新寄存器 APSR 中的 N 和 Z 标志，但不影响 C 和 V 标志。该汇编助记符指令的机器码格式如图 5.46 所示。可以看出，寄存器 Rd 和 Rn 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1	0	1	Rn			Rd		

图 5.46　MULS＜Rd＞，＜Rn＞，＜Rd＞指令的机器码格式

指令 MULS R0，R1，R0 的机器码为$(4348)_{16}$。该指令将寄存器 R1 的内容和寄存器 R0 的内容相乘，低 32 位结果保存在 R0 寄存器中，同时更新寄存器 APSR 中的标志。

5.8.4　比较指令

1. CMP ＜Rn＞，＜Rm＞

该指令比较寄存器 Rn 和寄存器 Rm 的内容，得到(Rn)－(Rm)的结果，但不保存该结果，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.47 所示。可以看出，寄存器 Rn 和 Rm 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	0	Rm			Rn		

图 5.47　CMP＜Rn＞，＜Rm＞指令的机器码格式

指令 CMP R0，R1 的机器码为$(4288)_{16}$。该指令比较寄存器 R0 和寄存器 R1 的内容，得到(R0)－(R1)的结果，但不保存该结果，同时更新寄存器 APSR 中的标志。

2. CMP ＜Rn＞，＃imm8

该指令将寄存器 Rn 的内容和立即数＃imm8(零扩展到 32 位)进行比较，得到(Rn)－imm8 的结果，但不保存该结果，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.48 所示。可以看出，寄存器 Rn 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	Rn			imm8							

图 5.48　CMP < Rn >，＃imm8 指令的机器码格式

注：图中立即数 imm8 的范围为 0～255(其零扩展到 32 位)。

指令 CMP R0，＃0x01 的机器码为$(2801)_{16}$。该指令将寄存器 R0 的内容和立即数 0x01 进行比较，得到(R0)－ 0x01 的结果，但不保存该结果，同时更新寄存器 APSR 中的标志。

3. CMN ＜Rn＞，＜Rm＞

该指令比较寄存器 Rn 的内容和对寄存器 Rm 取反后内容，得到(Rn)＋(Rm)的结果，但不保存该结果，同时更新寄存器 APSR 中的 N、Z、C 和 V 标志。该汇编助记符指令的机器码格式如图 5.49 所示。可以看出，寄存器 Rn 和 Rm 可用的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	Rm			Rn		

图 5.49　CMN < Rn >，< Rm >指令的机器码格式

指令 CMN R0，R1 的机器码为$(42C8)_{16}$。该指令比较寄存器 R0 和对寄存器 R1 取反后的内容，得到(R0)＋(R1)的结果，但不保存该结果，同时更新寄存器 APSR 中的标志。

思考与练习 5-8：将保存在寄存器 R0 和 R1 内的 64 位整数，与保存在寄存器 R2 和 R3 内的 64 位整数相加，结果保存在寄存器 R0 和 R1 中。使用 ADDS 和 ADCS 指令实现该 64 位整数相加功能。

思考与练习 5-9：将保存在寄存器 R1、R2 和 R3 内的 96 位整数，与保存在寄存器 R4、R5 和 R6 内的 96 位整数相减，结果保存在寄存器 R4、R5 和 R6 中。使用 SUBS 和 SBCS 指令实现该 96 位整数相减功能。

视频讲解

5.9　逻辑操作指令

本节介绍逻辑操作指令，下面对这些指令进行详细说明。

1. ANDS ＜Rd＞，＜Rm＞

该指令将寄存器 Rd 和寄存器 Rm 中的内容做“逻辑与”运算，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的标志 N、Z 和 C，但不更新标志 V。该汇编助记符指令的机器码格式如图 5.50 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	0	Rm			Rd		

图 5.50　ANDS < Rd >，< Rm >指令的机器码格式

指令 ANDS R0，R1 的机器码为$(4008)_{16}$。该指令将寄存器 R0 和寄存器 R1 中的内容做“逻辑与”运算，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

2. ORRS ＜Rd＞，＜Rm＞

该指令将寄存器 Rd 和寄存器 Rm 中的内容做“逻辑或”运算，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的 N、Z 和 C 标志，但不更新标志 V。该汇编助记符指令的机器码格

式如图 5.51 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1	0	0	Rm			Rd		

图 5.51　ORRS＜Rd＞，＜Rm＞指令的机器码格式

指令 ORRS R0，R1 的机器码为$(4308)_{16}$。该指令将寄存器 R0 和寄存器 R1 中的内容做“逻辑或”运算，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

3. EORS＜Rd＞，＜Rm＞

该指令将寄存器 Rd 和寄存器 Rm 中的内容做“逻辑异或”运算，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的标志 N、Z 和 C，但不更新标志 V。该汇编助记符指令的机器码格式如图 5.52 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	0	1	Rm			Rd		

图 5.52　EORS＜Rd＞，＜Rm＞指令的机器码格式

指令 EORS R0，R1 的机器码为$(4048)_{16}$。该指令将寄存器 R0 和寄存器 R1 中的内容做“逻辑异或”运算，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

4. MVNS＜Rd＞，＜Rm＞

该指令将寄存器 Rm 的[31:0]位按位做“逻辑取反”运算，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的标志 N、Z 和 C，但不更新标志 V。该汇编助记符指令的机器码格式如图 5.53 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1	1	1	Rm			Rd		

图 5.53　MVNS＜Rd＞，＜Rm＞指令的机器码格式

指令 MVNS R0，R1 的机器码为$(43C8)_{16}$。该指令将寄存器 R1 的[31:0]位按位做“逻辑取反”运算，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

5. BICS＜Rd＞，＜Rm＞

该指令将寄存器 Rm 的[31:0]位按位做“逻辑取反”运算，然后与寄存器 Rd 中的[31:0]位做“逻辑与”运算，结果保存在寄存器 Rd 中，同时更新寄存器 APSR 中的标志 N、Z 和 C，但不更新标志 V。该汇编助记符指令的机器码格式如图 5.54 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1	1	0	Rm			Rd		

图 5.54　BICS＜Rd＞，＜Rm＞指令的机器码格式

指令 BICS R0，R1 的机器码为$(4388)_{16}$。该指令将寄存器 R1 的[31:0]位按位做“逻辑取反”运算，然后与寄存器 R0 中的[31:0]位做“逻辑与”操作，结果保存在寄存器 R0 中，同时更新寄存器 APSR 中的标志。

6. TST ＜Rd＞, ＜Rm＞

该指令将寄存器 Rd 和寄存器 Rm 中的内容做“逻辑与”运算，但是不保存结果，同时更新寄存器 APSR 中的标志 N、Z 和 C，但不更新标志 V。该汇编助记符指令的机器码格式如图 5.55 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	0	0	Rm			Rd		

图 5.55　TST＜Rd＞, ＜Rm＞指令的机器码格式

指令 TST R0，R1 的机器码为$(4208)_{16}$。该指令将寄存器 R0 和寄存器 R1 中的内容做“逻辑与”运算，但是不保存结果，并且更新寄存器 APSR 中的标志。

思考与练习 5-10：说明下面指令所实现的功能。

(1) ANDS R2，R2，R1 _____

(2) ORRS R2，R2，R5 _____

(3) ANDS R5，R5，R8 _____

(4) EORS R7，R7，R6 _____

(5) BICS R0，R0，R1 _____

视频讲解

5.10　移位操作指令

本节介绍右移和左移操作指令，下面对这些指令进行详细介绍。

5.10.1　右移指令

1. ASRS ＜Rd＞, ＜Rm＞

该指令执行算术右移操作。将保存在寄存器 Rd 中的数据向右移动 Rm 所指定的次数，移位的结果保存在寄存器 Rd 中，即 Rd＝Rd≫Rm。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。在算术右移时，最高位的规则如图 5.56(a)所示。

图 5.56　算术和逻辑右移操作

该汇编助记符指令的机器码格式如图 5.57 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	0	0	Rm			Rd		

图 5.57　ASRS＜Rd＞, ＜Rm＞指令的机器码格式

指令 ASRS R0，R1 的机器码为$(4108)_{16}$。该指令将保存在寄存器 R0 中的数据向右移动 R1 所指定的次数，移位的结果保存在寄存器 R0 中。在右移过程中，最后移出去的位保存在

寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

2. ASRS ＜Rd＞，＜Rm＞，＃imm

该指令执行算术右移操作。将保存在寄存器 Rm 中的数据向右移动立即数 imm 所指定的次数，移位的结果保存在寄存器 Rd 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。该汇编助记符指令的机器码格式如图 5.58 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	imm5					Rm			Rd		

图 5.58　ASRS ＜Rd＞，＜Rm＞，＃imm 指令的格式

注：(1) 汇编指令中立即数 imm 的范围为 1～32。

(2) 当 imm＜32 时，imm5＝imm；当 imm＝32 时，imm5＝0。

指令 ASRS R0，R1，＃0x01 的机器码为$(1048)_{16}$。该指令将保存在寄存器 R1 中的数据向右移动 1 次，移位的结果保存在寄存器 R0 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

3. LSRS ＜Rd＞，＜Rm＞

该指令执行逻辑右移操作。将保存在寄存器 Rd 中的数据向右移动 Rm 所指定的次数，移位的结果保存在寄存器 Rd 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。在逻辑右移时，最高位的规则如图 5.56(b)所示。该汇编助记符指令的机器码格式如图 5.59 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	Rm			Rd		

图 5.59　LSRS ＜Rd＞，＜Rm＞指令的机器码格式

指令 LSRS R0，R1 的机器码为$(40C8)_{16}$。该指令将保存在寄存器 R0 中的数据向右移动 R1 所指定的次数，移位的结果保存在寄存器 R0 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

4. LSRS ＜Rd＞，＜Rm＞，＃imm

该指令执行逻辑右移操作。将保存在寄存器 Rm 中的数据向右移动立即数＃imm 所指定的次数，移位的结果保存在寄存器 Rd 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。该汇编助记符指令的机器码格式如图 5.60 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	imm5					Rm			Rd		

图 5.60　LSRS ＜Rd＞，＜Rm＞，＃imm 指令的机器码格式

注：(1) 汇编指令中立即数 imm 的范围为 1～32。

(2) 当 imm＜32 时，imm5＝imm；当 imm＝32 时，imm5＝0。

指令 LSRS R0，R1，＃0x01 的机器码为$(0848)_{16}$。该指令将保存在寄存器 R1 中的数据

向右移动 1 次，移位的结果保存在寄存器 R0 中。在右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

5. RORS ＜Rd＞，＜Rm＞

该指令执行循环右移操作。将保存在寄存器 Rd 中的数据向右循环移动 Rm 所指定的次数，移位的结果保存在寄存器 Rd 中。在循环右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。循环右移的规则如图 5.61 所示。该汇编助记符指令的机器码格式如图 5.62 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

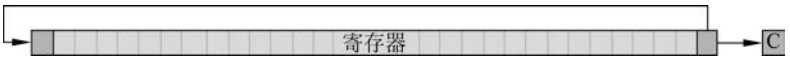


图 5.61　循环右移操作

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	1	1	1	Rm			Rd		

图 5.62　RORS＜Rd＞，＜Rm＞指令的机器码格式

指令 RORS R0，R1 的机器码为$(41C8)_{16}$。该指令将保存在寄存器 R0 中的数据向右循环移动 R1 所指定的次数，移位的结果保存在寄存器 R0 中。在循环右移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

5.10.2　左移指令

1. LSLS ＜Rd＞，＜Rm＞

该指令执行逻辑左移操作。将保存在寄存器 Rd 中的数据向左移动 Rm 所指定的次数，移位的结果保存在寄存器 Rd 中。在左移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。逻辑左移的规则如图 5.63 所示。该汇编助记符指令的机器码格式如图 5.64 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

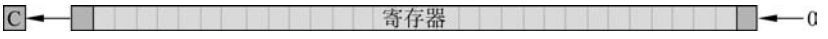


图 5.63　逻辑左移操作

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	0	Rm			Rd		

图 5.64　LSLS＜Rd＞，＜Rm＞指令的机器码格式

指令 LSLS R0，R1 的机器码为$(4088)_{16}$。该指令将保存在寄存器 R0 中的数据向左移动 R1 所指定的次数，移位的结果保存在寄存器 R0 中。在左移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

2. LSLS ＜Rd＞，＜Rm＞，＃imm

该指令执行逻辑左移操作。将保存在寄存器 Rd 中的数据向左移动立即数 imm 所指定的次数，移位的结果保存在寄存器 Rd 中，在左移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。该汇编助记符指令的机器码格式如图 5.65 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

注：imm5 与 imm 之间的关系见 LSRS 指令。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	imm5					Rm			Rd		

图 5.65　LSLS < Rd >, < Rm >, ＃imm 指令的机器码格式

指令 LSLS R0，R1，＃0x01 的机器码为$(0048)_{16}$。该指令将保存在寄存器 R1 中的数据向左移动 1 次，移位的结果保存在寄存器 R0 中。在左移过程中，最后移出去的位保存在寄存器 APSR 的 C 标志中，也更新 N 和 Z 标志。

思考与练习 5-11：说明下面指令所实现的功能。

(1) ASRS R7，R5，＃9 _____

(2) LSLS R1，R2，＃3 _____

(3) LSRS R4，R5，＃6 _____

(4) RORS R4，R4，R6 _____

视频讲解

5.11　反序操作指令

本节介绍反序操作指令，下面对这些指令进行详细说明。

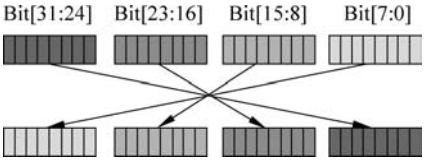


图 5.66　REV 操作

1. REV < Rd >, < Rm >

该指令将寄存器 Rm 中字节的顺序按逆序重新排列，结果保存在寄存器 Rd 中，如图 5.66 所示。该汇编助记符指令的机器码格式如图 5.67 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	0	0	Rm			Rd		

图 5.67　REV < Rd >, < Rm >指令的机器码格式

指令 REV R0，R1 的机器码为$(BA08)_{16}$。该指令将寄存器中 R1 的数据逆序重新排列{R1[7:0]，R1[15:8]，R1[23:16]，R1[31:24]}，结果保存在寄存器 R0 中。

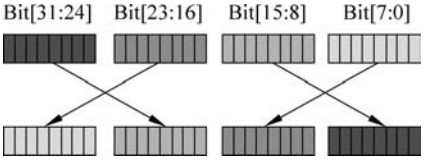


图 5.68　REV16 操作

2. REV16 < Rd >, < Rm >

该指令将寄存器 Rm 中的内容以半字为边界，半字内的两个字节逆序重新排列，结果保存在寄存器 Rd 中，如图 5.68 所示。该汇编助记符指令的机器码格式如图 5.69 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	0	1	Rm			Rd		

图 5.69　REV16 < Rd >, < Rm >指令的机器码格式

指令 REV16 R0，R1 的机器码为$(BA48)_{16}$。该指令将寄存器中 R1 的数据以半字为边界，半字内的字节按逆序重新排列，即{R1[23:16]，R1[31:24]，R1[7:0]，R1[15:8]}，结果保存在寄存器 R0 中。

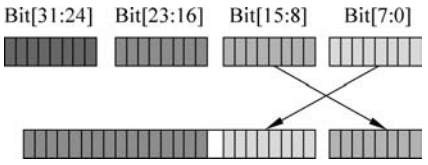


图 5.70　REVSH 操作

3. REVSH ＜Rd＞, ＜Rm＞

该指令将寄存器 Rm 中的低半字内的两个字节逆序重新排列，结果保存在寄存器 Rd[15:0]中，对于 Rd[31:16]中的内容由交换字节后 R[7]的内容决定，即符号扩展，如图 5.70 所示。该汇编助记符指令的机器码格式如图 5.71 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	0	1	0	1	1	Rm			Rd		

图 5.71　REVSH ＜Rd＞, ＜Rm＞指令的机器码格式

指令 REVSH R0，R1 的机器为$(BAC8)_{16}$。该指令将寄存器中 R1 的低半字内的两字节按逆序重新排列，结果保存在寄存器 R0[15:0]中，对于 R0[31:16]中的内容由交换字节后 R0[7]的内容决定，即符号扩展，表示为：R0＝符号扩展{R1[7:0]，R1[15:8]}。

思考与练习 5-12：说明下面指令所实现的功能。

(1) REV R3，R7 _____

(2) REV16 R0，R0 _____

(3) REVSH R0，R5 _____

视频讲解

5.12　扩展操作指令

本节介绍扩展操作指令，下面对这些指令进行详细说明。

1. SXTB ＜Rd＞, ＜Rm＞

将寄存器 Rm 中的[7:0]进行符号扩展，结果保存在寄存器 Rd 中。该汇编助记符指令的机器码格式如图 5.72 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	1	0	0	1	Rm			Rd		

图 5.72　SXTB ＜Rd＞, ＜Rm＞指令的机器码格式

指令 SXTB R0，R1 的机器码为$(B248)_{16}$。该指令将寄存器 R1 中的[7:0]进行符号扩展，结果保存在寄存器 R0 中，表示为 R0＝符号扩展{R1[7:0]}。

2. SXTH ＜Rd＞, ＜Rm＞

将寄存器 Rm 中的[15:0]进行符号扩展，结果保存在寄存器 Rd 中。该汇编助记符指令的机器码格式如图 5.73 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	1	0	0	0	Rm			Rd		

图 5.73　SXTH ＜Rd＞, ＜Rm＞指令的机器码格式

指令 SXTH R0，R1 的机器码为$(B208)_{16}$。该指令将寄存器 R1 中的[15:0]进行符号扩展，结果保存在寄存器 R0 中，表示为 R0＝符号扩展{R1[15:0]}。

3. UXTB ＜Rd＞, ＜Rm＞

将寄存器 Rm 中的[7:0]进行零扩展，结果保存在寄存器 Rd 中。该汇编助记符指令的机器码格式如图 5.74 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	1	0	1	1	Rm			Rd		

图 5.74　UXTB＜Rd＞, ＜Rm＞指令的机器码格式

指令 UXTB R0, R1 的机器码为$(B2C8)_{16}$。该指令将寄存器 R1 中的[7:0]进行零扩展，结果保存在寄存器 R0 中，表示为 R0＝零扩展{R1[7:0]}。

4. UXTH ＜Rd＞, ＜Rm＞

将寄存器 Rm 中的[15:0]进行零扩展，结果保存在寄存器 Rd 中。该汇编助记符指令的机器码格式如图 5.75 所示。可以看出，寄存器 Rm 和 Rd 的范围为 R0～R7。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	0	1	0	1	0	Rm			Rd		

图 5.75　UXTH＜Rd＞, ＜Rm＞指令的机器码格式

指令 UXTH R0, R1 的机器码为$(B288)_{16}$。该指令将寄存器 R1 中的[15:0]进行零扩展，结果保存在寄存器 R0 中，表示为 R0＝零扩展{R1[15:0]}。

思考与练习 5-13：说明下面指令所实现的功能。

(1) SXTH R4, R6 _____

(2) UXTB R3, R1 _____

视频讲解

5.13　程序流控制指令

本节介绍程序流控制指令，下面对这些指令进行详细说明。

1. B ＜label＞

实现无条件跳转，该汇编助记符指令的机器码格式如图 5.76 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	0	imm11										

图 5.76　B＜label＞指令的机器码格式

注：(1) label 是要跳转到的指令的标号。imm11 为汇编器计算得到的偏移量，该偏移量允许的范围为－2048～＋2046 之间的偶数。

(2) 当前 B 指令的 PC 值加 4，作为计算与目标标号地址之间偏移量的 PC 值，目标标号的地址减去该 PC 值得到偏移量，然后将得到的偏移量的值除以 2，则是 imm11 的值。

指令 B loop 无条件跳转到 loop 标号地址以执行指令。

2. B ＜cond＞＜label＞

有条件跳转指令，根据寄存器 APSR 中的 N、Z、C 和 V 标志，跳转到 label 所标识的地址。该汇编助记符指令的机器码格式如图 5.77 所示。其中，cond 为条件码，其编码格式如表 5.3 所示。

注：(1) label 是要跳转到的指令的标号。imm8 为汇编器计算得到的偏移量，该偏移量允

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	cond				imm8							

图 5.77　B < cond > < label >指令的机器码格式

许的范围为－256～＋254 之间的偶数。

(2) 当前 B 指令的 PC 值加 4，作为计算与目标标号地址之间偏移量的 PC 值，目标标号的地址减去该 PC 值得到偏移量，然后将得到的偏移量的值除以 2，则是 imm8 的值。

表 5.3　cond 的编码格式

cond	助记符扩展	含　义	条件标志
0000	EQ	相等	Z＝1
0001	NE	不相等	Z＝0
0010	CS(1)	设置进位	C＝1
0011	CC(2)	清除进位	C＝0
0100	MI	减，负的	N＝1
0101	PL	加，正的或零	N＝0
0110	VS	溢出	V＝1
0111	VC	无溢出	V＝0
1000	HI	无符号的较高	C＝1 且 Z＝0
1001	LS	无符号的较低或相同	C＝0 或 Z＝1
1010	GE	有符号的大于或等于	N＝V
1011	LT	有符号的小于	N!＝V
1100	GT	有符号的大于	Z＝0 且 N＝V
1101	LE	有符号的小于或等于	Z＝1 或 N!＝V
1110(3)	None(AL)(4)	总是(无条件)	任何

注：(1) HS(无符号较高或相同)是 CS 的同义词。

(2) LO(无符号的较低)是 CC 的同义词。

(3) 永远不会在 ARMv6-M Thumb 指令中对该值进行编码。

(4) AL 是为总是而提供的一个可选的助记符扩展名。

指令 BEQ loop 的功能是：当寄存器 APSR 寄存器中的标志位 Z 等于 1 时，将跳转到标号为 loop 的位置执行指令。

注：B 指令具体条件的表示参考 5.2.3 节介绍的后缀含义。

3. BL ＜label＞

该指令表示跳转和链接，跳转到一个地址，并且将返回地址保存到寄存器 LR 中。跳转地址为当前 PC±16M 字节的范围内。该指令通常用于调用一个子程序或者函数。一旦完成了函数，通过执行指令 BX LR 可以返回。

指令中的 label 是要跳转到的指令的标号。汇编器计算从 BL 指令的 PC 值(该 PC 值加 4 后才作为计算真正使用的 PC 值)到该标号的偏移量所需的值，然后选择将 imm32 设置为该偏移量的编码。允许的偏移量是－16777216～＋16777214。

BL 指令是 Thumb 指令集中的 32 位指令。该汇编助记符指令的机器码格式如图 5.78 所示。其中：

I1＝NOT(J1 EOR S)；I2＝NOT(J2 EOR S)；imm32＝SignExtend(S:I1:I2:imm10:imm11:'0',32)。很明显，偏移量地址包括 S、I1、I2、imm10 和 imm11。

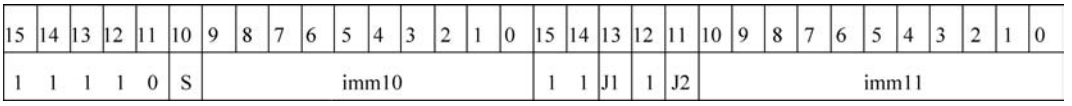


图 5.78　BL < label >指令的机器码格式

注：在引入 Thumb-2 技术之前，J1 和 J2 均为 1，导致分支的范围较小。这些指令可作为两个单独的 16 位指令执行，第一条指令 instr1 将 LR 设置为 PC＋有符号扩展(instr1 < 10:0 >:'000000000000',32)，第二条指令完成该操作。在 ARMv6T2、ARMv6-M 和 ARMv7 中，不再可能将 BL 指令拆分为两个 16 位指令。

指令 BL functionA，该指令将 PC 值修改为 functionA 标号所表示的地址值，寄存器 LR 的值等于 PC＋4。

4. BX ＜Rm＞

该指令表示跳转和交换，跳转到寄存器所指定的地址，根据寄存器的第 0 位的值(1 表示 Thumb，0 表示 ARM)，在 ARM 和 Thumb 模式之间切换处理器的状态。ARMv6-M 仅支持 Thumb 执行。尝试更改指令执行状态会导致目标地址上指令的异常。该汇编助记符指令的机器码格式如图 5.79 所示。可以看出，寄存器 Rm 可用的范围为 R0～R15。

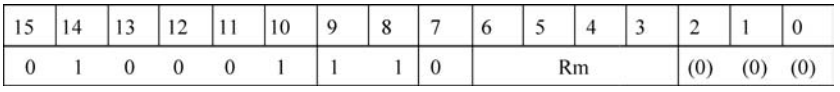


图 5.79　BX < Rm >指令的机器码格式

指令 BX R0 的机器码为$(4700)_{16}$。该指令将 PC 值修改为 R0 寄存器内的内容即 PC＝R0。

5. BLX ＜Rm＞

跳转和带有交换的链接。跳转到寄存器所指定的地址，将返回地址保存到寄存器 LR 中，并且根据寄存器的第 0 位的值(1 表示 Thumb，0 表示 ARM)，在 ARM 和 Thumb 模式之间切换处理器的状态。ARMv6-M 仅支持 Thumb 执行。尝试更改指令执行状态会导致目标地址上指令的异常。

注：返回地址＝当前指令的 PC 值＋4－2＝当前指令的 PC 值＋2，第 0 位需要设置为 1。

该汇编助记符指令的机器码格式如图 5.80 所示。可以看出，寄存器 Rm 可用的范围为 R0～R15。

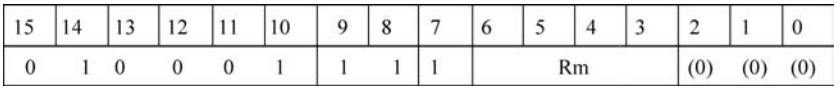


图 5.80　BLX < Rm >指令的机器码格式

指令 BLX R0 的机器码为$(4780)_2$。该指令将 PC 值修改为 R0 寄存器内的内容，即 PC＝R0，并且寄存器 LR 的值等于当前 PC＋2(LR 的第 0 位应该为 1，以保持 Thumb 状态)。

思考与练习 5-14：说明下面指令所实现的功能。

(1) B loopA _____

(2) BL funC _____

(3) BX LR _____

(4) BLX R0 _____

(5) BEQ labelD _____

视频讲解

5.14 存储器屏障指令

本节介绍存储器屏障指令，下面对这些指令进行详细说明。

1. DMB

数据存储器屏蔽指令(Data Memory Barrier，DMB)。在提交新的存储器访问之前，确保已经完成所有的存储器访问。DMB 指令是 Thumb 指令集上的 32 位指令。该汇编助记符指令的机器码格式如图 5.81 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	1	1	(1)	(1)	(1)	(1)	1	0	(0)	0	(1)	(1)	(1)	(1)	0	1	0	1	option			

图 5.81 DMB 指令的机器码格式

注：option 指定 DMB 操作的可选限制。SY DMB 操作可以确保所有访问的顺序，option 编码为 1111。可以省略。保留 option 的所有其他编码。相应指令执行系统(SY)DMB 操作，但是软件不依赖该行为。

2. DSB

数据同步屏蔽指令(Data Synchronization Barrier，DSB)。在执行下一条指令前，确保已经完成所有的存储器访问。DSB 指令是 Thumb 指令集上的 32 位指令。该汇编助记符指令的机器码格式如图 5.82 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	1	1	(1)	(1)	(1)	(1)	1	0	(0)	0	(1)	(1)	(1)	(1)	0	1	0	0	option			

图 5.82 DSB 指令的机器码格式

注：option 指定 DSB 操作的可选限制。SY DMB 操作可以确保完成所有的访问，option 编码为 1111。可以省略。保留 option 的所有其他编码。相应指令执行系统(SY)DSB 操作，但是软件不依赖该行为。

3. ISB

指令同步屏蔽指令(Instruction Synchronization Barrier，ISB)。在执行新的指令前，刷新流水线，确保已经完成先前所有的指令。ISB 指令是 Thumb 指令集上的 32 位指令。该汇编助记符指令的机器码格式如图 5.83 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	1	0	1	1	(1)	(1)	(1)	(1)	1	0	(0)	0	(1)	(1)	(1)	(1)	0	1	1	0	option			

图 5.83 ISB 指令的机器码格式

注：option 指定 ISB 操作的可选限制。完整的系统 ISB 操作，option 编码为 1111。可以省略。保留 option 的所有其他编码。相应指令执行完整的系统 ISB 操作，但是软件不依赖该行为。

视频讲解

5.15 异常相关指令

本节介绍异常相关指令，下面对这些指令进行详细说明。

1. SVC ＃＜imm8＞

管理员调用指令，触发 SVC 异常。该汇编助记符指令的机器码格式如图 5.84 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	1	1	1	1	imm8							

图 5.84　SVC ＃＜imm8＞指令的机器码格式

指令 SVC ＃3 的机器码为$(DF03)_{16}$。该指令表示触发 SVC 异常，其参数为 3。

2. CPS＜effect＞i

该指令修改处理器的状态，使能/禁止中断，该指令不会阻塞 NMI 和硬件故障句柄。该指令中的＜effect＞指定对 PRIMASK 所要求的效果。＜effect＞为下面其中之一：

(1) IE。中断使能，将 PRIMASK.PM 设置为 0。

(2) ID。中断禁止，将 PRIMASK.PM 设置为 1。

指令中的 i 表示 PRIMASK 受到的影响。当设置为 1 时，将当前优先级提高为 0。PRIMASK 是 1 位寄存器，只能从特权级执行中访问它。该汇编助记符指令的机器码格式如图 5.85 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	0	1	1	0	0	1	1	im	(0)	(0)	(1)	(0)

图 5.85　CPS＜effect＞i 指令的机器码格式

指令 CPSIE i 的机器码为$(B662)_{16}$，该指令使能中断，清除 PRIMASK；指令 CPSID i 的机器码为$(B672)_{16}$，该指令禁止中断，设置 PRIMASK。

视频讲解

5.16　休眠相关指令

本节介绍与休眠状态相关的指令，下面对这些指令进行详细说明。

1. WFI

该指令等待中断，停止执行程序，直到中断到来或者处理器进入调试状态。该汇编助记符指令的机器码格式如图 5.86 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	1	1	0	0	0	0

图 5.86　WFI 指令的机器码格式

2. WFE

该指令等待事件，停止执行程序，直到事件到来(由内部事件寄存器设置)或者处理器进入调试状态。该汇编助记符指令的机器码格式如图 5.87 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	1	0	0	0	0	0

图 5.87　WFE 指令的机器码格式

3. SEV

在多处理环境(包括自身)中,向所有处理器发送事件。该汇编助记符指令的机器码格式如图 5.88 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	1	0	0	0	0	0	0

图 5.88 SEV 指令的机器码格式


视频讲解

5.17 其他指令

本节介绍其他指令,下面进行详细说明。

1. NOP

该指令为空操作指令,该指令用于产生指令对齐,或者引入延迟。该汇编助记符指令的机器码格式如图 5.89 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0

图 5.89 NOP 指令的机器码格式

2. BKPT ＃＜imm8＞

该指令为断点指令,将处理器置位停止阶段。指令中的 imm8 标识保存在指令中的一个 8 位数。ARM 硬件忽略这个值,但是调试器可以用它来保存关于断点的额外信息。

在该阶段,通过调试器,用户执行调试任务。由调试器插入 BKPT 指令,用于代替原来的指令。该汇编助记符指令的机器码格式如图 5.90 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	0	imm8							

图 5.90 BKPT ＃＜imm8＞指令的机器码格式

指令 BKPT ＃5 的机器码格式为$(BE05)_{16}$,该指令表示标号为 5 的断点。

3. YIELD

YIELD 是一个提示指令。它使具有多线程功能的软件能够向硬件指示其正在执行任务(例如自旋锁),可以将其交换以提高系统的整体性能。

如果硬件支持该功能,则可以使用它提示挂起和恢复多个代码线程。该汇编助记符指令的机器码格式如图 5.91 所示。

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	1	1	1	1	0	0	0	1	0	0	0	0

图 5.91 YIELD 指令的机器码格式