第 3 章 ARM 微处理器的指令系统

本章介绍 ARM 指令集、Thumb 指令集，以及各类指令对应的寻址方式，通过对本章的阅读，希望读者能了解 ARM 微处理器所支持的指令集及具体的使用方法。

本章的主要内容有：

- ARM 指令集、Thumb 指令集概述。
- ARM 指令集的分类与具体应用。
- Thumb 指令集简介及应用场合。

3.1 ARM 微处理器的指令集概述

3.1.1 ARM 微处理器的指令的分类与格式

ARM微处理器的指令集是加载/存储型的，也即指令集仅能处理寄存器中的数据，而且处理结果都要放回寄存器中，而对系统存储器的访问则需要通过专门的加载/存储指令来完成。

ARM微处理器的指令集可以分为跳转指令、数据处理指令、程序状态寄存器（PSR）处理指令、加载/存储指令、协处理器指令和异常产生指令六大类，具体的指令及功能如表3-1所示（表中指令为基本ARM指令，不包括派生的ARM指令）。

表 3-1 ARM 指令及功能描述

助记符	指令功能描述
ADC	带进位加法指令
ADD	加法指令
AND	逻辑与指令
B	跳转指令
BIC	位清零指令
BL	带返回的跳转指令
BLX	带返回和状态切换的跳转指令
BX	带状态切换的跳转指令
CDP	协处理器数据操作指令
CMN	比较反值指令
CMP	比较指令
EOR	异或指令
LDC	存储器到协处理器的数据传输指令
LDM	加载多个寄存器指令
LDR	存储器到寄存器的数据传输指令
MCR	从 ARM 寄存器到协处理器寄存器的数据传输指令
MLA	乘加运算指令
MOV	数据传送指令
MRC	从协处理器寄存器到 ARM 寄存器的数据传输指令
MRS	传送 CPSR 或 SPSR 的内容到通用寄存器指令
MSR	传送通用寄存器到 CPSR 或 SPSR 的指令
MUL	32 位乘法指令
MLA	32 位乘加指令

MVN	数据取反传送指令
ORR	逻辑或指令
RSB	逆向减法指令
RSC	带借位的逆向减法指令
SBC	带借位减法指令
STC	协处理器寄存器写入存储器指令
STM	批量内存字写入指令
STR	寄存器到存储器的数据传输指令
SUB	减法指令
SWI	软件中断指令
SWP	交换指令
TEQ	相等测试指令
TST	位测试指令

3.1.2 指令的条件域

当处理器工作在ARM状态时，几乎所有的指令均根据CPSR中条件码的状态和指令的条件域有条件的执行。当指令的执行条件满足时，指令被执行，否则指令被忽略。

每一条ARM指令包含4位的条件码，位于指令的最高4位[31:28]。条件码共有16种，每种条件码可用两个字符表示，这两个字符可以添加在指令助记符的后面和指令同时使用。例如，跳转指令B可以加上后缀EQ变为BEQ表示“ 相等则跳转 ”，即当CPSR中的Z标志置位时发生跳转。

在16种条件标志码中，只有15种可以使用，如表3-2所示，第16种（1111）为系统保留，暂时不能使用。

表 3-2 指令的条件码

条件码	助记符后缀	标 志	含 义
0000	EQ	Z 置位	相等
0001	NE	Z 清零	不相等
0010	CS	C 置位	无符号数大于或等于
0011	CC	C 清零	无符号数小于
0100	MI	N 置位	负数
0101	PL	N 清零	正数或零
0110	VS	V 置位	溢出
0111	VC	V 清零	未溢出
1000	HI	C 置位 Z 清零	无符号数大于
1001	LS	C 清零 Z 置位	无符号数小于或等于
1010	GE	N 等于 V	带符号数大于或等于
1011	LT	N 不等于 V	带符号数小于
1100	GT	Z 清零且（N 等于 V）	带符号数大于
1101	LE	Z 置位或（N 不等于 V）	带符号数小于或等于
1110	AL	忽略	无条件执行

3.2 ARM 指令的寻址方式

所谓寻址方式就是处理器根据指令中给出的地址信息来寻找物理地址的方式。目前ARM指令系统支持如下几种常见的寻址方式。

3.2.1 立即寻址

立即寻址也叫立即数寻址，这是一种特殊的寻址方式，操作数本身就在指令中给出，只要取出指令也就取到了操作数。这个操作数被称为立即数，对应的寻址方式也就叫做立即寻址。例如以下指令：

```
ADD R0，R0，＃1          ；R0←R0＋1
ADD R0，R0，＃0x3f       ；R0←R0＋0x3f
```

在以上两条指令中，第二个源操作数即为立即数，要求以“ ＃ ”为前缀，对于以十六进制表示的立即数，还要求在“ ＃ ”后加上“ 0x ”或“ & ”。

3.2.2 寄存器寻址

寄存器寻址就是利用寄存器中的数值作为操作数，这种寻址方式是各类微处理器经常采用的一种方式，也是一种执行效率较高的寻址方式。以下指令：

```
ADD R0，R1，R2           ；R0←R1＋R2
```

该指令的执行效果是将寄存器R1和R2的内容相加，其结果存放在寄存器R0中。

3.2.2 寄存器间接寻址

寄存器间接寻址就是以寄存器中的值作为操作数的地址，而操作数本身存放在存储器中。例如以下指令：

```
ADD R0，R1，[R2]         ；R0←R1＋[R2]
LDR  R0，[R1]            ；R0←[R1]
STR  R0，[R1]            ；[R1]←R0
```

在第一条指令中，以寄存器 R2 的值作为操作数的地址，在存储器中取得一个操作数后与 R1 相加，结果存入寄存器 R0 中。

第二条指令将以 R1 的值为地址的存储器中的数据传送到 R0 中。

第三条指令将 R0 的值传送到以 R1 的值为地址的存储器中。

3.2.3 基址变址寻址

基址变址寻址就是将寄存器（该寄存器一般称作基址寄存器）的内容与指令中给出的地址偏移量相加，从而得到一个操作数的有效地址。变址寻址方式常用于访问某基地址附近的地址单元。采用变址寻址方式的指令常见有以下几种形式，如下所示：

```
LDR R0，[R1，＃4]          ；R0←[R1＋4]
LDR R0，[R1，＃4]！        ；R0←[R1＋4]、R1←R1＋4
LDR R0，[R1] ，＃4         ；R0←[R1]、R1←R1＋4
LDR R0，[R1，R2]           ；R0←[R1＋R2]
```

在第一条指令中，将寄存器 R1 的内容加上 4 形成操作数的有效地址，从而取得操作数存入寄存器 R0 中。

在第二条指令中，将寄存器 R1 的内容加上 4 形成操作数的有效地址，从而取得操作数存入寄存器 R0 中，然后，R1 的内容自增 4 个字节。

在第三条指令中，以寄存器 R1 的内容作为操作数的有效地址，从而取得操作数存入寄存器 R0 中，然后，R1 的内容自增 4 个字节。

在第四条指令中，将寄存器 R1 的内容加上寄存器 R2 的内容形成操作数的有效地址，从而取得操作数存入寄存器 R0 中。

3.2.4 多寄存器寻址

采用多寄存器寻址方式，一条指令可以完成多个寄存器值的传送。这种寻址方式可以用一条指令完成传送最多 16 个通用寄存器的值。以下指令：

```
LDMIA R0，{R1，R2，R3，R4}        ；R1←[R0]
                                  ；R2←[R0＋4]
                                  ；R3←[R0＋8]
                                  ；R4←[R0＋12]
```

该指令的后缀 IA 表示在每次执行完加载/存储操作后，R0 按字长度增加，因此，指令可将连续存储单元的值传送到 R1～R4。

3.2.5 相对寻址

与基址变址寻址方式相类似，相对寻址以程序计数器 PC 的当前值为基地址，指令中的地址标号作为偏移量，将两者相加之后得到操作数的有效地址。以下程序段完成子程序的调用和返回，跳转指令 BL 采用了相对寻址方式：

```
    BL   NEXT                     ；跳转到子程序 NEXT 处执行
    ……
NEXT
    ……
    MOV  PC，LR                   ；从子程序返回
```

3.2.6 堆栈寻址

堆栈是一种数据结构，按先进后出（First In Last Out，FILO）的方式工作，使用一个称作堆栈指针的专用寄存器指示当前的操作位置，堆栈指针总是指向栈顶。

当堆栈指针指向最后压入堆栈的数据时，称为满堆栈（Full Stack），而当堆栈指针指向下一个将要放入数据的空位置时，称为空堆栈（Empty Stack）。

同时，根据堆栈的生成方式，又可以分为递增堆栈（Ascending Stack）和递减堆栈（Decending Stack），当堆栈由低地址向高地址生成时，称为递增堆栈，当堆栈由高地址向低地址生成时，称为递减堆栈。这样就有四种类型的堆栈工作方式，ARM 微处理器支持这四种类型的堆栈工作方式，即：

- 满递增堆栈：堆栈指针指向最后压入的数据，且由低地址向高地址生成。
- 满递减堆栈：堆栈指针指向最后压入的数据，且由高地址向低地址生成。
- 空递增堆栈：堆栈指针指向下一个将要放入数据的空位置，且由低地址向高地址生成。
- 空递减堆栈：堆栈指针指向下一个将要放入数据的空位置，且由高地址向低地址生成。

3.3 ARM 指令集

本节对 ARM 指令集的六大类指令进行详细的描述。

3.3.1 跳转指令

跳转指令用于实现程序流程的跳转，在 ARM 程序中有两种方法可以实现程序流程的跳转：

- 使用专门的跳转指令。
- 直接向程序计数器 PC 写入跳转地址值。

通过向程序计数器 PC 写入跳转地址值，可以实现在 4GB 的地址空间中的任意跳转，在跳转之

前结合使用

```
MOV   LR，PC
```

等类似指令，可以保存将来的返回地址值，从而实现在 4GB 连续的线性地址空间的子程序调用。

ARM 指令集中的跳转指令可以完成从当前指令向前或向后的 32MB 的地址空间的跳转，包括以下 4 条指令：

— B 跳转指令
— BL 带返回的跳转指令
— BLX 带返回和状态切换的跳转指令
— BX 带状态切换的跳转指令

1、 B 指令

B 指令的格式为：

B{条件} 目标地址

B 指令是最简单的跳转指令。一旦遇到一个 B 指令，ARM 处理器将立即跳转到给定的目标地址，从那里继续执行。注意存储在跳转指令中的实际值是相对当前 PC 值的一个偏移量，而不是一个绝对地址，它的值由汇编器来计算（参考寻址方式中的相对寻址）。它是 24 位有符号数，左移两位后有符号扩展为 32 位，表示的有效偏移为 26 位(前后 32MB 的地址空间)。以下指令：

```
B     Label        ；程序无条件跳转到标号 Label 处执行
CMP   R1，#0       ；当 CPSR 寄存器中的 Z 条件码置位时，程序跳转到标号 Label 处执行
BEQ   Label
```

2、 BL 指令

BL 指令的格式为：

BL{条件} 目标地址

BL 是另一个跳转指令，但跳转之前，会在寄存器 R14 中保存 PC 的当前内容，因此，可以通过将 R14 的内容重新加载到 PC 中，来返回到跳转指令之后的那个指令处执行。该指令是实现子程序调用的一个基本但常用的手段。以下指令：

```
BL    Label        ；当程序无条件跳转到标号 Label 处执行时，同时将当前的 PC 值保存到 R14 中
```

3、 BLX 指令

BLX 指令的格式为：

BLX 目标地址

BLX 指令从 ARM 指令集跳转到指令中所指定的目标地址，并将处理器的工作状态有 ARM 状态切换到 Thumb 状态，该指令同时将 PC 的当前内容保存到寄存器 R14 中。因此，当子程序使用 Thumb 指令集，而调用者使用 ARM 指令集时，可以通过 BLX 指令实现子程序的调用和处理器工作状态的切换。同时，子程序的返回可以通过将寄存器 R14 值复制到 PC 中来完成。

4、 BX 指令

BX 指令的格式为：

BX{条件} 目标地址

BX 指令跳转到指令中所指定的目标地址，目标地址处的指令既可以是 ARM 指令，也可以是 Thumb 指令。

3.3.2 数据处理指令

数据处理指令可分为数据传送指令、算术逻辑运算指令和比较指令等。

数据传送指令用于在寄存器和存储器之间进行数据的双向传输。

算术逻辑运算指令完成常用的算术与逻辑的运算，该类指令不但将运算结果保存在目的寄存器中，同时更新 CPSR 中的相应条件标志位。

比较指令不保存运算结果，只更新 CPSR 中相应的条件标志位。

数据处理指令包括：

- MOV 数据传送指令
- MVN 数据取反传送指令
- CMP 比较指令
- CMN 反值比较指令
- TST 位测试指令
- TEQ 相等测试指令
- ADD 加法指令
- ADC 带进位加法指令
- SUB 减法指令
- SBC 带借位减法指令
- RSB 逆向减法指令
- RSC 带借位的逆向减法指令
- AND 逻辑与指令
- ORR 逻辑或指令
- EOR 逻辑异或指令
- BIC 位清除指令

1、 MOV 指令

MOV 指令的格式为：

MOV{条件}{S} 目的寄存器，源操作数

MOV 指令可完成从另一个寄存器、被移位的寄存器或将一个立即数加载到目的寄存器。其中 S 选项决定指令的操作是否影响 CPSR 中条件标志位的值，当没有 S 时指令不更新 CPSR 中条件标志位的值。

指令示例：

```
MOV   R1，R0           ；将寄存器 R0 的值传送到寄存器 R1
MOV   PC，R14          ；将寄存器 R14 的值传送到 PC，常用于子程序返回
MOV   R1，R0，LSL＃3   ；将寄存器 R0 的值左移 3 位后传送到 R1
```

2、 MVN 指令

MVN 指令的格式为：

MVN{条件}{S} 目的寄存器，源操作数

MVN 指令可完成从另一个寄存器、被移位的寄存器、或将一个立即数加载到目的寄存器。与 MOV 指令不同之处是在传送之前按位被取反了，即把一个被取反的值传送到目的寄存器中。其中 S 决定指令的操作是否影响 CPSR 中条件标志位的值，当没有 S 时指令不更新 CPSR 中条件标志位的值。

指令示例：

```
MVN   R0，＃0          ；将立即数 0 取反传送到寄存器 R0 中，完成后 R0=-1
```

3、 CMP 指令

CMP 指令的格式为：

CMP{条件} 操作数 1，操作数 2

CMP 指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行比较，同时更新 CPSR 中条件标志位的值。该指令进行一次减法运算，但不存储结果，只更改条件标志位。标志位表示的是操作数 1 与操作数 2 的关系(大、小、相等)，例如，当操作数 1 大于操作操作数 2，则此后的有 GT 后缀的指令将可以执行。

指令示例：

```
CMP   R1，R0           ；将寄存器 R1 的值与寄存器 R0 的值相减，并根据结果设置 CPSR 的标志位
CMP   R1，＃100        ；将寄存器 R1 的值与立即数 100 相减，并根据结果设置 CPSR 的标志位
```

4、 CMN 指令

CMN 指令的格式为：

CMN{条件} 操作数 1，操作数 2

CMN 指令用于把一个寄存器的内容和另一个寄存器的内容或立即数取反后进行比较，同时更新 CPSR 中条件标志位的值。该指令实际完成操作数 1 和操作数 2 相加，并根据结果更改条件标志位。

指令示例：

```
CMN   R1，R0          ；将寄存器 R1 的值与寄存器 R0 的值相加，并根据结果设置 CPSR 的标志位
CMN   R1，＃100       ；将寄存器 R1 的值与立即数 100 相加，并根据结果设置 CPSR 的标志位
```

5、 TST 指令

TST 指令的格式为：

TST{条件} 操作数 1，操作数 2

TST 指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行按位的与运算，并根据运算结果更新 CPSR 中条件标志位的值。操作数 1 是要测试的数据，而操作数 2 是一个位掩码，该指令一般用来检测是否设置了特定的位。

指令示例：

```
TST   R1，＃%1        ；用于测试在寄存器 R1 中是否设置了最低位（%表示二进制数）
TST   R1，＃0xffe     ；将寄存器 R1 的值与立即数 0xffe 按位与，并根据结果设置 CPSR 的标志位
```

6、 TEQ 指令

TEQ 指令的格式为：

TEQ{条件} 操作数 1，操作数 2

TEQ 指令用于把一个寄存器的内容和另一个寄存器的内容或立即数进行按位的异或运算，并根据运算结果更新 CPSR 中条件标志位的值。该指令通常用于比较操作数 1 和操作数 2 是否相等。

指令示例：

```
TEQ   R1，R2        ；将寄存器 R1 的值与寄存器 R2 的值按位异或，并根据结果设置 CPSR 的标志位
```

7、 ADD 指令

ADD 指令的格式为：

ADD{条件}{S} 目的寄存器，操作数 1，操作数 2

ADD 指令用于把两个操作数相加，并将结果存放到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。

指令示例：

```
ADD   R0，R1，R2              ； R0 = R1 + R2
ADD   R0，R1，#256            ； R0 = R1 + 256
ADD   R0，R2，R3，LSL#1       ； R0 = R2 + (R3 << 1)
```

8、 ADC 指令

ADC 指令的格式为：

ADC{条件}{S} 目的寄存器，操作数 1，操作数 2

ADC 指令用于把两个操作数相加，再加上 CPSR 中的 C 条件标志位的值，并将结果存放到目的寄存器中。它使用一个进位标志位，这样就可以做比 32 位大的数的加法，注意不要忘记设置 S 后缀来更改进位标志。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。

以下指令序列完成两个 128 位数的加法，第一个数由高到低存放在寄存器 R7～R4，第二个数由高到低存放在寄存器 R11～R8，运算结果由高到低存放在寄存器 R3～R0：

```
ADDS  R0，R4，R8          ； 加低端的字
ADCS  R1，R5，R9          ； 加第二个字，带进位
ADCS  R2，R6，R10         ； 加第三个字，带进位
ADC   R3，R7，R11         ； 加第四个字，带进位
```

9、 SUB 指令

SUB 指令的格式为：

SUB{条件}{S} 目的寄存器，操作数 1，操作数 2

SUB 指令用于把操作数 1 减去操作数 2，并将结果存放到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令可用于有符号数或无符号数的减法运算。

指令示例：

```
SUB   R0，R1，R2              ；R0 = R1 - R2
SUB   R0，R1，#256            ；R0 = R1 - 256
SUB   R0，R2，R3，LSL#1       ；R0 = R2 - (R3 << 1)
```

10、SBC 指令

SBC 指令的格式为：

SBC{条件}{S} 目的寄存器，操作数 1，操作数 2

SBC 指令用于把操作数 1 减去操作数 2，再减去 CPSR 中的 C 条件标志位的反码，并将结果存放到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令使用进位标志来表示借位，这样就可以做大于 32 位的减法，注意不要忘记设置 S 后缀来更改进位标志。该指令可用于有符号数或无符号数的减法运算。

指令示例：

```
SUBS  R0，R1，R2              ；R0 = R1 - R2 - ！C，并根据结果设置 CPSR 的进位标志位
```

11、RSB 指令

RSB 指令的格式为：

RSB{条件}{S} 目的寄存器，操作数 1，操作数 2

RSB 指令称为逆向减法指令，用于把操作数 2 减去操作数 1，并将结果存放到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令可用于有符号数或无符号数的减法运算。

指令示例：

```
RSB   R0，R1，R2              ；R0 = R2 – R1
RSB   R0，R1，#256            ；R0 = 256 – R1
RSB   R0，R2，R3，LSL#1       ；R0 = (R3 << 1) - R2
```

12、RSC 指令

RSC 指令的格式为：

RSC{条件}{S} 目的寄存器，操作数 1，操作数 2

RSC 指令用于把操作数 2 减去操作数 1，再减去 CPSR 中的 C 条件标志位的反码，并将结果存放到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令使用进位标志来表示借位，这样就可以做大于 32 位的减法，注意不要忘记设置 S 后缀来更改进位标志。该指令可用于有符号数或无符号数的减法运算。

指令示例：

```
RSC   R0，R1，R2              ；R0 = R2 – R1 - ！C
```

13、AND 指令

AND 指令的格式为：

AND{条件}{S} 目的寄存器，操作数 1，操作数 2

AND 指令用于在两个操作数上进行逻辑与运算，并把结果放置到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于屏蔽操作数 1 的某些位。

指令示例：

```
AND   R0，R0，＃3             ；该指令保持 R0 的 0、1 位，其余位清零。
```

14、ORR 指令

ORR 指令的格式为：

ORR{条件}{S} 目的寄存器，操作数 1，操作数 2

ORR 指令用于在两个操作数上进行逻辑或运算，并把结果放置到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于设置操作数 1 的某些位。

指令示例：

```
ORR   R0，R0，＃3              ； 该指令设置 R0 的 0、1 位，其余位保持不变。
```

15、EOR 指令

EOR 指令的格式为：

EOR{条件}{S} 目的寄存器，操作数 1，操作数 2

EOR 指令用于在两个操作数上进行逻辑异或运算，并把结果放置到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。该指令常用于反转操作数 1 的某些位。

指令示例：

```
EOR   R0，R0，＃3              ； 该指令反转 R0 的 0、1 位，其余位保持不变。
```

16、BIC 指令

BIC 指令的格式为：

BIC{条件}{S} 目的寄存器，操作数 1，操作数 2

BIC 指令用于清除操作数 1 的某些位，并把结果放置到目的寄存器中。操作数 1 应是一个寄存器，操作数 2 可以是一个寄存器，被移位的寄存器，或一个立即数。操作数 2 为 32 位的掩码，如果在掩码中设置了某一位，则清除这一位。未设置的掩码位保持不变。

指令示例：

```
BIC   R0，R0，＃%1011          ； 该指令清除 R0 中的位 0、1、和 3，其余的位保持不变。
```

3.3.3 乘法指令与乘加指令

ARM 微处理器支持的乘法指令与乘加指令共有 6 条，可分为运算结果为 32 位和运算结果为 64 位两类，与前面的数据处理指令不同，指令中的所有操作数、目的寄存器必须为通用寄存器，不能对操作数使用立即数或被移位的寄存器，同时，目的寄存器和操作数 1 必须是不同的寄存器。

乘法指令与乘加指令共有以下 6 条：

- MUL　32 位乘法指令
- MLA　32 位乘加指令
- SMULL　64 位有符号数乘法指令
- SMLAL　64 位有符号数乘加指令
- UMULL　64 位无符号数乘法指令
- UMLAL　64 位无符号数乘加指令

1、 MUL 指令

MUL 指令的格式为：

MUL{条件}{S} 目的寄存器，操作数 1，操作数 2

MUL 指令完成将操作数 1 与操作数 2 的乘法运算，并把结果放置到目的寄存器中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的有符号数或无符号数。

指令示例：

```
MUL   R0，R1，R2       ；R0 = R1 × R2
MULS  R0，R1，R2       ；R0 = R1 × R2，同时设置 CPSR 中的相关条件标志位
```

2、 MLA 指令

MLA 指令的格式为：

MLA{条件}{S} 目的寄存器，操作数 1，操作数 2，操作数 3

MLA 指令完成将操作数 1 与操作数 2 的乘法运算，再将乘积加上操作数 3，并把结果放置到目的寄存器中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的有符号数或无符号数。

指令示例：

```
MLA   R0，R1，R2，R3        ；R0 = R1 × R2 + R3
MLAS  R0，R1，R2，R3        ；R0 = R1 × R2 + R3，同时设置 CPSR 中的相关条件标志位
```

3、 SMULL 指令

SMULL 指令的格式为：

SMULL{条件}{S}　目的寄存器 Low，目的寄存器低 High，操作数 1，操作数 2

SMULL 指令完成将操作数 1 与操作数 2 的乘法运算，并把结果的低 32 位放置到目的寄存器 Low 中，结果的高 32 位放置到目的寄存器 High 中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的有符号数。

指令示例：

```
SMULL R0，R1，R2，R3        ；R0 = （R2 × R3）的低 32 位
                            ；R1 = （R2 × R3）的高 32 位
```

4、 SMLAL 指令

SMLAL 指令的格式为：

SMLAL{条件}{S}　目的寄存器 Low，目的寄存器低 High，操作数 1，操作数 2

SMLAL 指令完成将操作数 1 与操作数 2 的乘法运算，并把结果的低 32 位同目的寄存器 Low 中的值相加后又放置到目的寄存器 Low 中，结果的高 32 位同目的寄存器 High 中的值相加后又放置到目的寄存器 High 中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的有符号数。

对于目的寄存器 Low，在指令执行前存放 64 位加数的低 32 位，指令执行后存放结果的低 32 位。

对于目的寄存器 High，在指令执行前存放 64 位加数的高 32 位，指令执行后存放结果的高 32 位。

指令示例：

```
SMLAL R0，R1，R2，R3        ；R0 = （R2 × R3）的低 32 位 ＋ R0
                            ；R1 = （R2 × R3）的高 32 位 ＋ R1
```

5、 UMULL 指令

UMULL 指令的格式为：

UMULL{条件}{S}　目的寄存器 Low，目的寄存器低 High，操作数 1，操作数 2

UMULL 指令完成将操作数 1 与操作数 2 的乘法运算，并把结果的低 32 位放置到目的寄存器 Low 中，结果的高 32 位放置到目的寄存器 High 中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的无符号数。

指令示例：

```
UMULL R0，R1，R2，R3        ；R0 = （R2 × R3）的低 32 位
                            ；R1 = （R2 × R3）的高 32 位
```

6、 UMLAL 指令

UMLAL 指令的格式为：

UMLAL{条件}{S}　目的寄存器 Low，目的寄存器低 High，操作数 1，操作数 2

UMLAL 指令完成将操作数 1 与操作数 2 的乘法运算，并把结果的低 32 位同目的寄存器 Low 中的值相加后又放置到目的寄存器 Low 中，结果的高 32 位同目的寄存器 High 中的值相加后又放置到目的寄存器 High 中，同时可以根据运算结果设置 CPSR 中相应的条件标志位。其中，操作数 1 和操作数 2 均为 32 位的无符号数。

对于目的寄存器 Low，在指令执行前存放 64 位加数的低 32 位，指令执行后存放结果的低 32 位。

对于目的寄存器 High，在指令执行前存放 64 位加数的高 32 位，指令执行后存放结果的高 32 位。

指令示例：

```
UMLAL R0，R1，R2，R3        ；R0 = （R2 × R3）的低 32 位 ＋ R0
                            ；R1 = （R2 × R3）的高 32 位 ＋ R1
```

3.3.4 程序状态寄存器访问指令

ARM 微处理器支持程序状态寄存器访问指令，用于在程序状态寄存器和通用寄存器之间传送数据，程序状态寄存器访问指令包括以下两条：

— MRS 程序状态寄存器到通用寄存器的数据传送指令

— MSR 通用寄存器到程序状态寄存器的数据传送指令

1、 MRS 指令

MRS 指令的格式为：

MRS{条件}　通用寄存器，程序状态寄存器（CPSR 或 SPSR）

MRS 指令用于将程序状态寄存器的内容传送到通用寄存器中。该指令一般用在以下几种情况：

－ 当需要改变程序状态寄存器的内容时，可用 MRS 将程序状态寄存器的内容读入通用寄存器，修改后再写回程序状态寄存器。

－ 当在异常处理或进程切换时，需要保存程序状态寄存器的值，可先用该指令读出程序状态寄存器的值，然后保存。

指令示例：

```
MRS   R0，CPSR          ；传送 CPSR 的内容到 R0
MRS   R0，SPSR          ；传送 SPSR 的内容到 R0
```

2、 MSR 指令

MSR 指令的格式为：

MSR{条件}　程序状态寄存器（CPSR 或 SPSR）_<域>，操作数

MSR 指令用于将操作数的内容传送到程序状态寄存器的特定域中。其中，操作数可以为通用寄存器或立即数。<域>用于设置程序状态寄存器中需要操作的位，32 位的程序状态寄存器可分为 4 个域：

位[31：24]为条件标志位域，用 f 表示；

位[23：16]为状态位域，用 s 表示；

位[15：8]为扩展位域，用 x 表示；

位[7：0]为控制位域，用 c 表示；

该指令通常用于恢复或改变程序状态寄存器的内容，在使用时，一般要在 MSR 指令中指明将要操作的域。

指令示例：

```
MSR   CPSR，R0          ；传送 R0 的内容到 CPSR
MSR   SPSR，R0          ；传送 R0 的内容到 SPSR
MSR   CPSR_c，R0        ；传送 R0 的内容到 SPSR，但仅仅修改 CPSR 中的控制位域
```

3.3.5 加载/存储指令

ARM 微处理器支持加载/存储指令用于在寄存器和存储器之间传送数据，加载指令用于将存储器中的数据传送到寄存器，存储指令则完成相反的操作。常用的加载存储指令如下：

— LDR　字数据加载指令

— LDRB　字节数据加载指令

— LDRH　半字数据加载指令

— STR　字数据存储指令

— STRB　字节数据存储指令

— STRH　半字数据存储指令

1、LDR 指令

LDR 指令的格式为：

LDR{条件} 目的寄存器，<存储器地址>

LDR 指令用于从存储器中将一个 32 位的字数据传送到目的寄存器中。该指令通常用于从存储器中读取 32 位的字数据到通用寄存器，然后对数据进行处理。当程序计数器 PC 作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。该指令在程序设计中比较常用，且寻址方式灵活多样，请读者认真掌握。

指令示例：

```
LDR   R0，[R1]                ；将存储器地址为 R1 的字数据读入寄存器 R0。
LDR   R0，[R1，R2]            ；将存储器地址为 R1+R2 的字数据读入寄存器 R0。
LDR   R0，[R1，＃8]           ；将存储器地址为 R1+8 的字数据读入寄存器 R0。
LDR   R0，[R1，R2] ！         ；将存储器地址为 R1+R2 的字数据读入寄存器 R0，并将新地址 R1＋R2 写入 R1。
LDR   R0，[R1，＃8] ！        ；将存储器地址为 R1+8 的字数据读入寄存器 R0，并将新地址 R1＋8 写入 R1。
LDR   R0，[R1]，R2            ；将存储器地址为 R1 的字数据读入寄存器 R0，并将新地址 R1＋R2 写入 R1。
LDR   R0，[R1，R2，LSL＃2]！  ；将存储器地址为 R1＋R2×4 的字数据读入寄存器 R0，并将新地址 R1＋R2×4 写入 R1。
LDR   R0，[R1]，R2，LSL＃2    ；将存储器地址为 R1 的字数据读入寄存器 R0，并将新地址 R1＋R2×4 写入 R1。
```

2、LDRB 指令

LDRB 指令的格式为：

LDR{条件}B 目的寄存器，<存储器地址>

LDRB 指令用于从存储器中将一个 8 位的字节数据传送到目的寄存器中，同时将寄存器的高 24 位清零。该指令通常用于从存储器中读取 8 位的字节数据到通用寄存器，然后对数据进行处理。当程序计数器 PC 作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

```
LDRB  R0，[R1]          ；将存储器地址为 R1 的字节数据读入寄存器 R0，并将 R0 的高 24 位清零。
LDRB  R0，[R1，＃8]     ；将存储器地址为 R1＋8 的字节数据读入寄存器 R0，并将 R0 的高 24 位清零。
```

3、LDRH 指令

LDRH 指令的格式为：

LDR{条件}H 目的寄存器，<存储器地址>

LDRH 指令用于从存储器中将一个 16 位的半字数据传送到目的寄存器中，同时将寄存器的高 16 位清零。该指令通常用于从存储器中读取 16 位的半字数据到通用寄存器，然后对数据进行处理。当程序计数器 PC 作为目的寄存器时，指令从存储器中读取的字数据被当作目的地址，从而可以实现程序流程的跳转。

指令示例：

```
LDRH  R0，[R1]          ；将存储器地址为 R1 的半字数据读入寄存器 R0，并将 R0 的高 16 位清零。
LDRH  R0，[R1，＃8]     ；将存储器地址为 R1＋8 的半字数据读入寄存器 R0，并将 R0 的高 16 位清零。
LDRH  R0，[R1，R2]      ；将存储器地址为 R1＋R2 的半字数据读入寄存器 R0，并将 R0 的高 16 位清零。
```

4、STR 指令

STR 指令的格式为：

STR{条件} 源寄存器，<存储器地址>

STR 指令用于从源寄存器中将一个 32 位的字数据传送到存储器中。该指令在程序设计中比较常用，且寻址方式灵活多样，使用方式可参考指令 LDR。

指令示例：

```
STR   R0，[R1]，＃8     ;将 R0 中的字数据写入以 R1 为地址的存储器中，并将新地址 R1＋8 写入 R1。
STR   R0，[R1，＃8]     ；将 R0 中的字数据写入以 R1＋8 为地址的存储器中。
```

5、STRB 指令

STRB 指令的格式为：

STR{条件}B 源寄存器，<存储器地址>

STRB 指令用于从源寄存器中将一个 8 位的字节数据传送到存储器中。该字节数据为源寄存器中的低 8 位。

指令示例：

```
STRB  R0，[R1]          ；将寄存器 R0 中的字节数据写入以 R1 为地址的存储器中。
STRB  R0，[R1，＃8]     ；将寄存器 R0 中的字节数据写入以 R1＋8 为地址的存储器中。
```

6、STRH 指令

STRH 指令的格式为：

STR{条件}H 源寄存器，<存储器地址>

STRH 指令用于从源寄存器中将一个 16 位的半字数据传送到存储器中。该半字数据为源寄存器中的低 16 位。

指令示例：

```
STRH  R0，[R1]          ；将寄存器 R0 中的半字数据写入以 R1 为地址的存储器中。
STRH  R0，[R1，＃8]     ；将寄存器 R0 中的半字数据写入以 R1＋8 为地址的存储器中。
```

3.3.6 批量数据加载/存储指令

ARM 微处理器所支持批量数据加载/存储指令可以一次在一片连续的存储器单元和多个寄存器之间传送数据，批量加载指令用于将一片连续的存储器中的数据传送到多个寄存器，批量数据存储指令则完成相反的操作。常用的加载存储指令如下：

— LDM　　批量数据加载指令

— STM　　批量数据存储指令

LDM（或 STM）指令

LDM（或 STM）指令的格式为：

LDM（或 STM）{条件}{类型} 基址寄存器{！}，寄存器列表{^}

LDM（或 STM）指令用于从由基址寄存器所指示的一片连续存储器到寄存器列表所指示的多个寄存器之间传送数据，该指令的常见用途是将多个寄存器的内容入栈或出栈。其中，{类型}为以下几种情况：

IA　每次传送后地址加 1；

IB　每次传送前地址加 1；

DA　每次传送后地址减 1；

DB　每次传送前地址减 1；

FD　满递减堆栈；

ED　空递减堆栈；

FA　满递增堆栈；

EA　空递增堆栈；

{！}为可选后缀，若选用该后缀，则当数据传送完毕之后，将最后的地址写入基址寄存器，否则基址寄存器的内容不改变。

基址寄存器不允许为 R15，寄存器列表可以为 R0～R15 的任意组合。

{^}为可选后缀，当指令为 LDM 且寄存器列表中包含 R15，选用该后缀时表示：除了正常的数据传送之外，还将 SPSR 复制到 CPSR。同时，该后缀还表示传入或传出的是用户模式下的寄存器，而不是当前模式下的寄存器。

指令示例：

```
STMFD  R13!，{R0，R4-R12，LR}      ；将寄存器列表中的寄存器（R0，R4 到 R12，LR）存入堆栈。
LDMFD  R13!，{R0，R4-R12，PC}      ；将堆栈内容恢复到寄存器（R0，R4 到 R12，LR）。
```

3.3.7 数据交换指令

ARM 微处理器所支持数据交换指令能在存储器和寄存器之间交换数据。数据交换指令有如下两条：

— SWP 字数据交换指令

— SWPB 字节数据交换指令

1、SWP 指令

SWP 指令的格式为：

SWP{条件} 目的寄存器，源寄存器 1，[源寄存器 2]

SWP 指令用于将源寄存器 2 所指向的存储器中的字数据传送到目的寄存器中，同时将源寄存器 1 中的字数据传送到源寄存器 2 所指向的存储器中。显然，当源寄存器 1 和目的寄存器为同一个寄存器时，指令交换该寄存器和存储器的内容。

指令示例：

```
SWP   R0，R1，[R2]          ；将 R2 所指向的存储器中的字数据传送到 R0，同时将 R1 中的字数据传送到 R2 所指向的存储单元。
SWP   R0，R0，[R1]          ；该指令完成将 R1 所指向的存储器中的字数据与 R0 中的字数据交换。
```

2、SWPB 指令

SWPB 指令的格式为：

SWP{条件}B 目的寄存器，源寄存器 1，[源寄存器 2]

SWPB 指令用于将源寄存器 2 所指向的存储器中的字节数据传送到目的寄存器中，目的寄存器的高 24 清零，同时将源寄存器 1 中的字节数据传送到源寄存器 2 所指向的存储器中。显然，当源寄存器 1 和目的寄存器为同一个寄存器时，指令交换该寄存器和存储器的内容。

指令示例：

```
SWPB  R0，R1，[R2]          ；将 R2 所指向的存储器中的字节数据传送到 R0，R0 的高 24 位清零，同时将 R1 中的低 8 位数据传送到 R2 所指向的存储单元。
SWPB  R0，R0，[R1]          ；该指令完成将 R1 所指向的存储器中的字节数据与 R0 中的低 8 位数据交换。
```

3.3.8 移位指令（操作）

ARM 微处理器内嵌的桶型移位器（Barrel Shifter），支持数据的各种移位操作，移位操作在 ARM 指令集中不作为单独的指令使用，它只能作为指令格式中是一个字段，在汇编语言中表示为指令中的选项。例如，数据处理指令的第二个操作数为寄存器时，就可以加入移位操作选项对它进行各种移位操作。移位操作包括如下 6 种类型，ASL 和 LSL 是等价的，可以自由互换：

— LSL 逻辑左移

— ASL 算术左移

— LSR 逻辑右移

— ASR 算术右移

— ROR 循环右移

— RRX 带扩展的循环右移

1、LSL（或 ASL）操作

LSL（或 ASL）操作的格式为：

通用寄存器，LSL（或 ASL） 操作数

LSL（或 ASL）可完成对通用寄存器中的内容进行逻辑（或算术）的左移操作，按操作数所指定的数量向左移位，低位用零来填充。其中，操作数可以是通用寄存器，也可以是立即数（0～31）。

操作示例：

```
MOV    R0, R1, LSL#2      ；将 R1 中的内容左移两位后传送到 R0 中。
```

2、LSR 操作

LSR 操作的格式为：

通用寄存器，LSR 操作数

LSR 可完成对通用寄存器中的内容进行右移的操作，按操作数所指定的数量向右移位，左端用零来填充。其中，操作数可以是通用寄存器，也可以是立即数（0～31）。

操作示例：

```
MOV    R0, R1, LSR#2      ；将 R1 中的内容右移两位后传送到 R0 中，左端用零来填充。
```

3、ASR 操作

ASR 操作的格式为：

通用寄存器，ASR 操作数

ASR 可完成对通用寄存器中的内容进行右移的操作，按操作数所指定的数量向右移位，左端用第 31 位的值来填充。其中，操作数可以是通用寄存器，也可以是立即数（0～31）。

操作示例：

```
MOV    R0, R1, ASR#2      ；将 R1 中的内容右移两位后传送到 R0 中，左端用第 31 位的值来填充。
```

4、ROR 操作

ROR 操作的格式为：

通用寄存器，ROR 操作数

ROR 可完成对通用寄存器中的内容进行循环右移的操作，按操作数所指定的数量向右循环移位，左端用右端移出的位来填充。其中，操作数可以是通用寄存器，也可以是立即数（0～31）。显然，当进行 32 位的循环右移操作时，通用寄存器中的值不改变。

操作示例：

```
MOV    R0, R1, ROR#2      ；将 R1 中的内容循环右移两位后传送到 R0 中。
```

5、RRX 操作

RRX 操作的格式为：

通用寄存器，RRX 操作数

RRX 可完成对通用寄存器中的内容进行带扩展的循环右移的操作，按操作数所指定的数量向右循环移位，左端用进位标志位 C 来填充。其中，操作数可以是通用寄存器，也可以是立即数（0～31）。

操作示例：

```
MOV    R0, R1, RRX#2      ；将 R1 中的内容进行带扩展的循环右移两位后传送到 R0 中。
```

3.3.9 协处理器指令

ARM 微处理器可支持多达 16 个协处理器，用于各种协处理操作，在程序执行的过程中，每个协处理器只执行针对自身的协处理指令，忽略 ARM 处理器和其他协处理器的指令。

ARM 的协处理器指令主要用于 ARM 处理器初始化 ARM 协处理器的数据处理操作，以及在 ARM 处理器的寄存器和协处理器的寄存器之间传送数据，和在 ARM 协处理器的寄存器和存储器之间传送数据。ARM 协处理器指令包括以下 5 条：

- CDP　协处理器数操作指令
- LDC　协处理器数据加载指令
- STC　协处理器数据存储指令
- MCR　ARM 处理器寄存器到协处理器寄存器的数据传送指令

— MRC 协处理器寄存器到 ARM 处理器寄存器的数据传送指令

1、CDP 指令

CDP 指令的格式为：

CDP{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2。

CDP 指令用于 ARM 处理器通知 ARM 协处理器执行特定的操作,若协处理器不能成功完成特定的操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器和源寄存器均为协处理器的寄存器，指令不涉及 ARM 处理器的寄存器和存储器。

指令示例：

```
CDP   P3，2，C12，C10，C3，4          ；该指令完成协处理器 P3 的初始化
```

2、LDC 指令

LDC 指令的格式为：

LDC{条件}{L} 协处理器编码,目的寄存器，[源寄存器]

LDC 指令用于将源寄存器所指向的存储器中的字数据传送到目的寄存器中,若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。

指令示例：

```
LDC   P3，C4，[R0]          ；将 ARM 处理器的寄存器 R0 所指向的存储器中的字数据传送到协处理器 P3 的寄存器 C4 中。
```

3、STC 指令

STC 指令的格式为：

STC{条件}{L} 协处理器编码,源寄存器，[目的寄存器]

STC 指令用于将源寄存器中的字数据传送到目的寄存器所指向的存储器中,若协处理器不能成功完成传送操作，则产生未定义指令异常。其中，{L}选项表示指令为长读取操作，如用于双精度数据的传输。

指令示例：

```
STC   P3，C4，[R0]          ；将协处理器 P3 的寄存器 C4 中的字数据传送到 ARM 处理器的寄存器 R0 所指向的存储器中。
```

4、MCR 指令

MCR 指令的格式为：

MCR{条件} 协处理器编码，协处理器操作码 1，源寄存器，目的寄存器 1，目的寄存器 2，协处理器操作码 2。

MCR 指令用于将 ARM 处理器寄存器中的数据传送到协处理器寄存器中,若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，源寄存器为 ARM 处理器的寄存器，目的寄存器 1 和目的寄存器 2 均为协处理器的寄存器。

指令示例：

```
MCR   P3，3，R0，C4，C5，6     ；该指令将 ARM 处理器寄存器 R0 中的数据传送到协处理器 P3 的寄存器 C4 和 C5 中。
```

5、MRC 指令

MRC 指令的格式为：

MRC{条件} 协处理器编码，协处理器操作码 1，目的寄存器，源寄存器 1，源寄存器 2，协处理器操作码 2。

MRC 指令用于将协处理器寄存器中的数据传送到 ARM 处理器寄存器中,若协处理器不能成功完成操作，则产生未定义指令异常。其中协处理器操作码 1 和协处理器操作码 2 为协处理器将要执行的操作，目的寄存器为 ARM 处理器的寄存器，源寄存器 1 和源寄存器 2 均为协处理器的寄存器。

指令示例：

```
MRC   P3，3，R0，C4，C5，6     ；该指令将协处理器 P3 的寄存器中的数据传送到 ARM 处理器寄存器
```

中。

3.3.10 异常产生指令

ARM 微处理器所支持的异常指令有如下两条：

— SWI 软件中断指令

— BKPT 断点中断指令

1、SWI 指令

SWI 指令的格式为：

SWI{条件} 24 位的立即数

SWI 指令用于产生软件中断，以便用户程序能调用操作系统的系统例程。操作系统在 SWI 的异常处理程序中提供相应的系统服务，指令中 24 位的立即数指定用户程序调用系统例程的类型，相关参数通过通用寄存器传递，当指令中 24 位的立即数被忽略时，用户程序调用系统例程的类型由通用寄存器 R0 的内容决定，同时，参数通过其他通用寄存器传递。

指令示例：

```
SWI   0x02          ；该指令调用操作系统编号位 02 的系统例程。
```

2、BKPT 指令

BKPT 指令的格式为：

BKPT 16 位的立即数

BKPT 指令产生软件断点中断，可用于程序的调试。

3.4 Thumb 指令及应用

为兼容数据总线宽度为 16 位的应用系统，ARM 体系结构除了支持执行效率很高的 32 位 ARM 指令集以外，同时支持 16 位的 Thumb 指令集。Thumb 指令集是 ARM 指令集的一个子集，允许指令编码为 16 位的长度。与等价的 32 位代码相比较，Thumb 指令集在保留 32 代码优势的同时，大大的节省了系统的存储空间。

所有的 Thumb 指令都有对应的 ARM 指令，而且 Thumb 的编程模型也对应于 ARM 的编程模型，在应用程序的编写过程中，只要遵循一定调用的规则，Thumb 子程序和 ARM 子程序就可以互相调用。当处理器在执行 ARM 程序段时，称 ARM 处理器处于 ARM 工作状态，当处理器在执行 Thumb 程序段时，称 ARM 处理器处于 Thumb 工作状态。

与 ARM 指令集相比较，Thumb 指令集中的数据处理指令的操作数仍然是 32 位，指令地址也为 32 位，但 Thumb 指令集为实现 16 位的指令长度，舍弃了 ARM 指令集的一些特性，如大多数的 Thumb 指令是无条件执行的，而几乎所有的 ARM 指令都是有条件执行的；大多数的 Thumb 数据处理指令的目的寄存器与其中一个源寄存器相同。

由于 Thumb 指令的长度为 16 位，即只用 ARM 指令一半的位数来实现同样的功能，所以，要实现特定的程序功能，所需的 Thumb 指令的条数较 ARM 指令多。在一般的情况下，Thumb 指令与 ARM 指令的时间效率和空间效率关系为：

— Thumb 代码所需的存储空间约为 ARM 代码的 60%～70%

— Thumb 代码使用的指令数比 ARM 代码多约 30%～40%

— 若使用 32 位的存储器，ARM 代码比 Thumb 代码快约 40%

— 若使用 16 位的存储器，Thumb 代码比 ARM 代码快约 40%～50%

— 与 ARM 代码相比较，使用 Thumb 代码，存储器的功耗会降低约 30%

显然，ARM 指令集和 Thumb 指令集各有其优点，若对系统的性能有较高要求，应使用 32 位的存储系统和 ARM 指令集，若对系统的成本及功耗有较高要求，则应使用 16 位的存储系统和 Thumb 指令集。当然，若两者结合使用，充分发挥其各自的优点，会取得更好的效果。

3.5 本章小节

本章系统的介绍了 ARM 指令集中的基本指令，以及各指令的应用场合及方法，由基本指令还可以派生出一些新的指令，但使用方法与基本指令类似。与常见的如 X86 体系结构的汇编指令相比较，ARM 指令系统无论是从指令集本身，还是从寻址方式上，都相对复杂一些。

Thumb 指令集作为 ARM 指令集的一个子集，其使用方法与 ARM 指令集类似，在此未作详细的描述，但这并不意味着 Thumb 指令集不如 ARM 指令集重要，事实上，他们各自有其自己的应用场合。