第 3 章　ARM9 指令系统

本章介绍 ARM 处理器的寻址方式、ARM 指令集和 Thumb 指令集。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。

3.1　ARM 处理器的寻址方式

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。

3.1.1　寄存器寻址

寄存器寻址就是利用寄存器中的内容作为操作数，寄存器本身就是操作数地址。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。例如以下指令：

```
MOV R2,R3              ; R2←R3        R3 中的内容赋给 R2
ADD  R2,R3,R4          ; R2←R3+R4     R3 和 R4 中的内容相加，结果赋给 R2
```

3.1.2　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，操作数没有存储在寄存器或存储器中，而是包含在指令的操作码中，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   R1,R1,#1234                    ; R1←R1+1234
ADD   R1,R1,#0x7f                    ; R1←R1+0x7f
```

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”。

3.1.3　寄存器间接寻址

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

```
LDR   R1,[R2]                        ; R1←[R2]
STR   R1,[R2]                        ; [R2]←R1
```

第一条指令以 R2 中的内容为地址，将该地址中的数据传送到 R1 中。

第二条指令将 R1 中的内容传送到以 R2 中的内容为地址的存储器中。

3.1.4　变址寻址

变址寻址就是将寄存器(该寄存器一般称为基址寄存器)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有以下几种常见形式,如下所示。

```
LDR R0,[R1,#8]          ; R0←[R1+8]
LDR R0,[R1,#8]!         ; R0←[R1+8],R1←R1+8
LDR R0,[R1],#2          ; R0←[R1],R1←R1+2
LDR R0,[R1,R2]          ; R0←[R1+R2]
```

在第一条指令中,将寄存器 R1 的内容加上 8 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 8 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 8 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 8 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。

3.1.5　寄存器移位寻址

寄存器移位寻址是 ARM 指令集独有的寻址方式,操作数由寄存器的数值进行相应移位而得到;移位的方式在指令中以助记符的形式给出,而移位的位数可用立即数或寄存器寻址方式表示。

ARM 微处理器内嵌的桶型移位器(barrel shifter),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的一个字段,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 2 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。

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,最后一个左移出的位放在状态寄存器的 C 位 CPSR[29]中,如图 3-1 所示。其中,操作数可以是通用寄存器,也可以是立即数(0～31)。

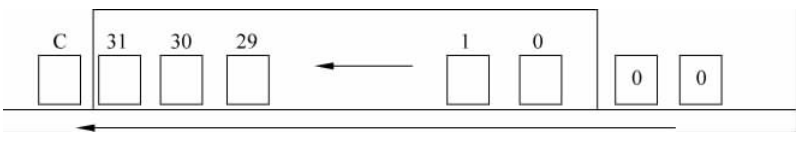

图 3-1　逻辑(或算术)左移

操作示例：

```
MOV     R0, R1, LSL#4          ; 将 R1 中的内容左移 4 位后传送到 R0 中
                               ; 其中把最后移出的位赋给程序状态寄存器的 C 位 CPSR[29]
```

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，最后一个右移出的位放在状态寄存器的 C 位 CPSR[29]中，如图 3-2 所示。其中，操作数可以是通用寄存器，也可以是立即数(0～31)。

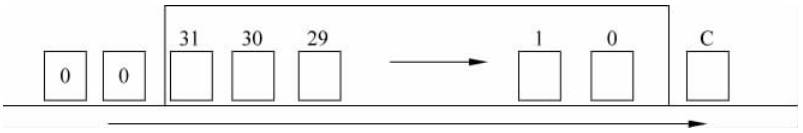

图 3-2　逻辑(或算术)右移

操作示例：

```
MOV     R0, R1, LSR#4          ; 将 R1 中的内容右移 4 位后传送到 R0 中
                               ; 其中把最后移出的位赋给程序状态寄存器的 C 位 CPSR[29]
```

3. 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，右端移出的位填充在左侧的空位处，最后一个右移出的位同时也放在状态寄存器的 C 位 CPSR[29]中，如图 3-3 所示。其中，操作数可以是通用寄存器，也可以是立即数(0～31)。

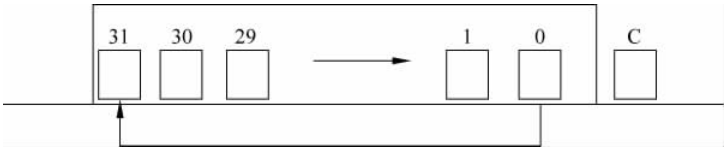

图 3-3　循环右移

操作示例：

```
MOV     R0, R1, ROR#4          ; 将 R1 中的内容循环右移 4 位后传送到 R0 中
                               ; 其中把最后移出的位赋给程序状态寄存器的 C 位 CPSR[29]
```

4. 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，最左端的位保持不变，最后一个右移出的位放在状态寄存器的 C 位 CPSR[29]中，如图 3-4 所示。其中，操作数可以是通用寄存器，也可以是立即数(0～31)。

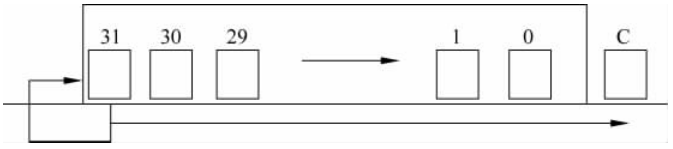

图 3-4　算术右移

这种移位对有符号数据使用时可以保持符号位不变。

操作示例：

```
MOV     R0, R1, ASR#4          ; 将 R1 中的内容右移 4 位后传送到 R0 中，符号位保持不变
                               ; 最后移出的位同时也送入状态位 C 中
```

5. RRX 操作

RRX 操作的格式为：

通用寄存器，RRX

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，左侧空位由状态寄存器 C 位来填充，右侧移出的位移进状态位 C 中，如图 3-5 所示。其中，操作数可以是通用寄存器，也可以是立即数(0～31)。

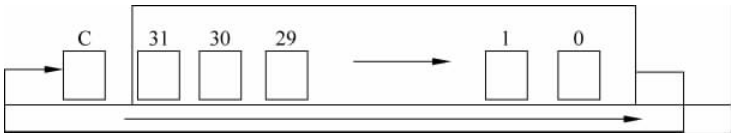

图 3-5　扩展的循环右移

操作示例：

```
MOV     R0, R1, RRX            ; 将 R1 中的内容进行带扩展的循环右移 1 位后传送到 R0 中
```

3.1.6　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。这种寻址方式可以一次对多个寄存器寻址，多个寄存器由小到大排列，最多可传送 16 个寄存器。例如：

```
LDMIA R1,{R2-R4,R5}            ; R2←[R1]
                               ; R3←[R1+4]
                               ; R4←[R1+8]
                               ; R5←[R1+12]
```

该指令的后缀 IA 表示在每次执行完加载/存储操作后，R1 按字长度增加，因此，指令可将连续存储单元的值传送到 R2～R5。

3.1.7 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,使用一个称做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4种类型的堆栈工作方式,ARM微处理器支持这4种类型的堆栈工作方式,即:

- 满递增方式(full ascending,FA),堆栈指针指向最后入栈的数据位置,且由低地址向高地址生成。
- 满递减方式(full decending,FD),堆栈指针指向最后入栈的数据位置,且由高地址向低地址生成。
- 空递增方式(empty ascending,EA),堆栈指针指向下一个入栈数据的空位置,且由低地址向高地址生成。
- 空递减方式(empty decending,ED),堆栈指针指向下一个入栈数据的空位置,且由高地址向低地址生成。

3.1.8 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 ARM指令集

ARM微处理器的指令集是加载/存储型的,即指令集仅能处理寄存器中的数据,处理结果仍要放回寄存器中,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。

3.2.1 指令格式

为了方便编写程序,ARM指令在汇编程序中用助记符表示。一般ARM指令的助记符格式如下:

＜opcode＞{＜cond＞}{s} ＜Rd＞,＜Rn＞,＜op2＞

其中各项介绍如下。

＜opcode＞：操作码，如 ADD 表示算术加操作指令。

{＜cond＞}：决定指令执行的条件码。

{s}：决定指令执行是否影响 CPSR 寄存器的值。

＜Rd＞：目的寄存器。

＜Rn＞：第一个操作数，为寄存器。

＜op2＞：第二个操作数。

注意：opcode、cond 与 s 之间没有分隔符，s 与 Rd 之间用空格隔开。

3.2.2　条件码

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地执行。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 3-1 所示，第 16 种(1111)为系统保留，暂时不能使用。

表 3-1　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

3.2.3　ARM 存储器访问指令

ARM 微处理器内部没有 RAM，而 ARM 除了寄存器(即 R0～R15)外没有别的存储单

元；在以 ARM 为核的嵌入式系统中，所有的外围模块都和存储单元一样，是 ARM 微处理器的不同的地址单元。不管这些模块的功能如何(如输入输出、定时器、存储器等)，也不管这些模块的位置如何(如片内或片外)，ARM 微处理器都把它们看作是外部存储器。其操作过程和对存储器的操作是相同的。因此，在 ARM 微处理器的数据传送中，数据的源和数据的目标只有两种：一种是 ARM 的寄存器 R0～R15；另一种就是外部存储器(它们可能是外围模块的寄存器、外部数据存储器或可访问的程序存储器等)。

因此，把数据从存储器到寄存器的传送叫加载，数据从寄存器到存储器的传送叫存储。具体的传送方式如图 3-6 所示。

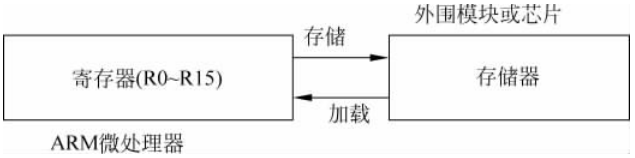

图 3-6　ARM 微处理器的数据传送方式

加载/存储指令可分为：单一数据加载/存储指令、批量数据加载/存储指令以及数据交换指令 3 类。常用的加载/存储指令如下所示。

1. 单一数据加载/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 指令是字加载指令，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。

指令示例：

```
LDR   R3,[R4]                ; 将存储器地址为 R4 的字数据读入寄存器 R3
LDR   R3,[R1,R2]             ; 将存储器地址为 R1+R2 的字数据读入寄存器 R3
LDR   R3,[R1,#8]             ; 将存储器地址为 R1+8 的字数据读入寄存器 R3
LDR   R3,[R1,R2]!            ; 将存储器地址为 R1+R2 的字数据读入寄存器 R3,并将新地址
                             ; R1+R2 写入 R1
LDR   R3,[R1,#8]!            ; 将存储器地址为 R1+8 的字数据读入寄存器 R3,并将新地址
                             ; R1+8 写入 R1
LDR   R3,[R1],R2             ; 将存储器地址为 R1 的字数据读入寄存器 R3,并将新地址 R1+R2
                             ; 写入 R1
LDR   R3,[R1,R2,LSL#3]!      ; 将存储器地址为 R1+R2×8 的字数据读入寄存器 R3,并将新地
                             ; 址 R1+R2×8 写入 R1
LDR   R3,[R1],R2,LSL#3       ; 将存储器地址为 R1 的字数据读入寄存器 R3,并将新地址 R1+
                             ; R2×8 写入 R1
```

注：R15 不可以作为偏移寄存器使用。

(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 指令是字节加载指令,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    R3,[R1]     ; 将存储器地址为 R1 的字节数据读入寄存器 R3,并将 R3 的高 24 位清零
LDRB    R3,[R1,#8]  ; 将存储器地址为 R1+8 的字节数据读入寄存器 R3,并将 R3 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 指令是无符号半字加载指令,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   R3,[R1]      ; 将存储器地址为 R1 的半字数据读入寄存器 R3,并将 R3 的高 16 位清零
LDRH   R3,[R1,#8]   ; 将存储器地址为 R1+8 的半字数据读入寄存器 R3,并将 R3 的高 16 位清零
LDRH   R3,[R1,R2]   ; 将存储器地址为 R1+R2 的半字数据读入寄存器 R3,并将 R3 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 指令是字存储指令,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   R3,[R1],#8    ; 将 R3 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   R3,[R1,#8]    ; 将 R3 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 指令是无符号字节存储指令,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   R3,[R1]      ; 将寄存器 R3 中的字节数据写入以 R1 为地址的存储器中
STRB   R3,[R1,#8]   ; 将寄存器 R3 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 指令是无符号半字存储指令,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   R3,[R1]      ; 将寄存器 R3 中的半字数据写入以 R1 为地址的存储器中
STRH   R3,[R1,#8]   ; 将寄存器 R3 中的半字数据写入以 R1+8 为地址的存储器中
```

2. 批量数据加载/存储指令

ARM 微处理器所支持的批量数据加载/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(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)指令用于在由基址寄存器所指示的一片连续存储器和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,递增方式。

IB 每次传送前地址加 1,递增方式。

DA 每次传送后地址减 1,递减方式。

DB 每次传送前地址减 1,递减方式。

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,若使用连续的寄存器时,可以使用“-”表示省略。

{^}为可选后缀,这是一个只是在数据块传送中使用的后缀,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～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～R12,LR)
```

在通用存储区，数据存储的方式和堆栈区相近。R1、R2 和 R3 三个寄存器的 4 种后缀指令执行前后的存储情况如图 3-7 所示。

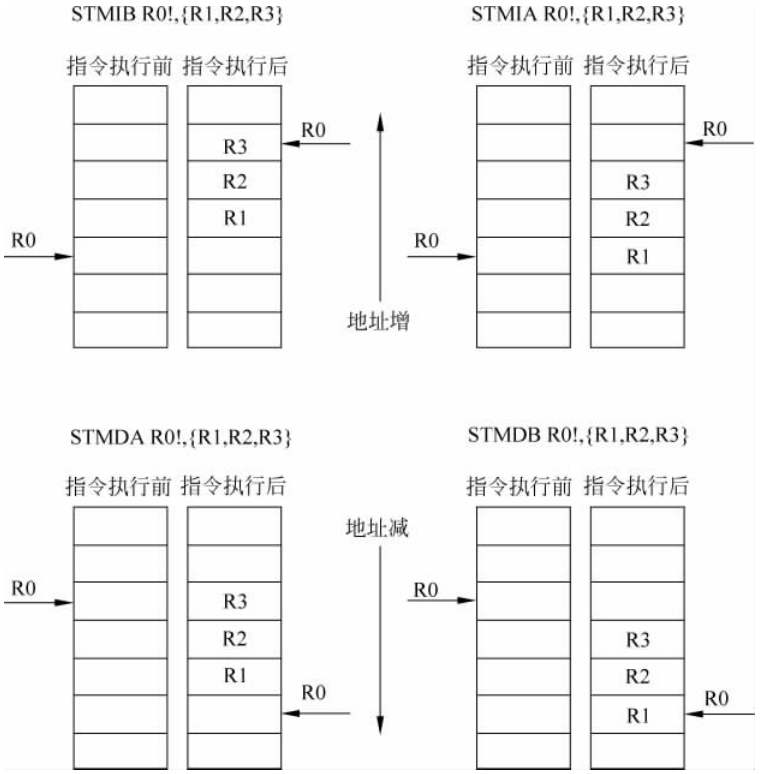

图 3-7　数据存储示意图

3. 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 指令是数据字交换指令，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     R1,R2,[R3]    ; 将 R3 所指向的存储器中的字数据传送到 R1,同时将 R2 中的字数据传
                      ; 送到 R3 所指向的存储单元
SWPEQ   R1,R1,[R2]    ; Z=1 时,完成将 R2 所指向的存储器中的字数据与 R1 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 指令是字节交换指令,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 位清零,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   R1,R2,[R3]      ; 将 R3 所指向的存储器中的字节数据传送到 R1,R1 的高 24 位
                       ; 清零,同时将 R2 中的低 8 位数据传送到 R3 所指向的存储单元
SWPB   R1,R1,[R2]      ; 将 R2 所指向的存储器中的字节数据与 R1 中的低 8 位数据交换
```

3.2.4　ARM 数据处理类指令

数据处理指令只能对寄存器的内容进行操作,不允许对存储器中的数据进行操作,也不允许指令直接使用存储器的数据或在寄存器与存储器之间传送数据。数据处理指令可分为三大类:数据传送指令、算术逻辑运算指令和比较指令。

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。

比较指令是对指定的两个寄存器(或一个寄存器,一个立即数)进行比较,不保存运算结果,只影响 CPSR 中相应的条件标志位。

1. 数据传送指令 MOV 和 MVN

(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 指令可完成在寄存器之间或寄存器与第 2 操作数之间进行的数据传送。

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   R4,R5            ; 将寄存器 R5 的内容传送到寄存器 R4
MOV   PC,R14           ; 将寄存器 R14 的内容传送到 PC,常用于子程序返回
MOVNE R4,R5,LSL#2      ; 当 Z=0 时,将寄存器 R5 的内容逻辑左移 2 位后传送到 R4
```

(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 指令可完成在寄存器之间或寄存器与第 2 操作数之间进行的数据非传送。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
```

2. 算术逻辑运算指令

(1) 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 指令是加法指令,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：

```
ADDS     R0,R3,R4              ; R0 = R3 + R4,设置标志位
ADDC     R0,R3,#10             ; R0 = R3 + 10
ADD   R0,R2,R3,LSL#2           ; R0 = R2 + R3×4
```

(2) 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 指令是带进位加法指令,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。

以下指令序列实现 64 位二进制数的加法：R2、R1=R2、R1+R4、R3

```
ADDS   R1,R1,R3                ; R1= R1+R3
ADC    R2,R2,R4                ; R2= R2+R4+C
```

(3) 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 指令是减法指令,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。该指令可用于有符号数或无符号数的减法运算。

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：

```
SUBS R0,R3,R4                  ; R0 = R3-R4,设置标志位
SUB   R0,R1,#0x10              ; R0 = R1-0x10
SUB   R0,R2,R3,LSL#1           ; R0 = R2-(R3 << 1)
```

(4) 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 指令是带借位减法指令,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。该指令可用于有符号数或无符号数的减法运算。

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。

指令示例:

```
SBC   R2,R2,R4                ; R2 = R2-R4-C̄
```

(5) 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 指令是反减法指令,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。该指令可用于有符号数或无符号数的减法运算。

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:

```
RSB   R0,R1,R2                ; R0 = R2-R1
RSB   R0,R1,#0x10             ; R0 = 0x10-R1
RSB   R0,R2,R3,LSL#1          ; R0 =R3×2-R2
```

(6) 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 指令是带借位反减法指令,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。该指令可用于有符号数或无符号数的减法运算。

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:

```
RSCS    R6,R4,R3,LSL#1        ; R6 = R3×2-R4-C̄ 同时刷新标志位
```

(7) 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 指令是逻辑与指令，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。该指令常用于屏蔽操作数 1 的某些位。

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：

```
AND   R5,R6,R8                ; R5=R6∧R8
AND   R2,R2,#3                ; 该指令保持 R2 的 0、1 位，其余位清零
```

(8) 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 指令是逻辑或指令，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。该指令常用于设置操作数 1 的某些位。

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：

```
ORR   R5,R6,R8                ; R5=R6∨R8
ORR   R2,R2,#3                ; 该指令设置 R2 的 0、1 位，其余位保持不变
```

(9) 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 指令是逻辑异或指令，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。该指令常用于反转操作数 1 的某些位。

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：

```
EOR   R5,R6,R8                ; R5=R6⊕R8
EOR   R2,R2,#3                ; 该指令反转 R2 的 0、1 位为 1，其余位保持不变
```

(10) 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 指令是位清除指令，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。

指令示例：

```
BIC   R0,R2,#2_0011           ; 清除 R2 中的 0、1 位，其余的位保持不变
```

```
BICS   R0,R2,#0x80000000      ; 清除 R2 中的 31 位,其余的位保持不变,刷新标志位
```

(11) 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 指令是乘法指令,完成操作数 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(不会影响 V)。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,R4,R5                ; R0 = R4 × R5
MULS  R0,R4,R5                ; R0 = R4 × R5,同时设置条件标志位
```

(12) 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 指令是乘加指令,完成操作数 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(不会影响 V)。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
```

(13) 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 High,操作数 1,操作数 2
```

SMULL 指令是带符号长乘法指令,完成操作数 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     R1,R2,R3,R4         ; R1 = (R3 × R4)的低 32 位
                              ; R2 = (R3 × R4)的高 32 位
```

(14) 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 High,操作数 1,操作数 2
```

SMLAL 指令是长乘加指令,完成操作数 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    R1,R2,R3,R4        ; R1 = (R3×R4)的低 32 位 + R1
                            ; R2 = (R3×R4)的高 32 位 + R2
```

(15) 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 High,操作数 1,操作数 2
```

UMULL 指令是无符号乘法指令，完成操作数 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    R1,R2,R3,R4        ; R1 = (R3×R4)的低 32 位
                            ; R2 = (R3×R4)的高 32 位
```

(16) 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 High,操作数 1,操作数 2
```

UMLAL 指令是无符号长乘加指令，完成操作数 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    R1,R2,R3,R4        ; R1 = (R3×R4)的低 32 位 + R1
                            ; R2= (R3×R4)的高 32 位 + R2
```

3. 比较和测试指令

(1) CMP 指令

CMP 指令的格式为：

```
CMP{条件} 操作数 1,操作数 2
```

CMP 指令是比较指令，该指令是做一次减法运算，但不存储结果，只是刷新条件标志

位,根据条件标志位判断操作数的大小。对条件位的影响是:结果为正数则N=0,结果为负数则N=1;结果为0则Z=1,结果不为0则Z=0;如果产生借位则C=0,没有借位则C=1;结果符号溢出则V=1,否则V=0。

指令示例:

```
CMP  R1,#0x30              ; 比较R1和0x30
ADDCS R5,R5,#0x20          ; 如果C=1,则R5=R5+0x20
ADDCC R5,R5,#0x10          ; 如果C=0,则R5=R5+0x10
```

(2) CMN指令

CMN指令的格式为:

```
CMN{条件} 操作数1,操作数2
```

CMN指令是比较非指令,也是做一次减法运算,用操作数1减去操作数2的非值,结果不保存,只是刷新条件标志位,对条件标志位的影响和比较指令CMP相同。

指令示例:

```
CMN  R1,#0x00              ; 比较R1和0xFFFFFFFF
ADDCS R5,R5,#0x20          ; 如果C=1,则R5=R5+0x20
ADDCC R5,R5,#0x10          ; 如果C=0,则R5=R5+0x10
```

(3) TST指令

TST指令的格式为:

```
TST{条件} 操作数1,操作数2
```

TST指令是位测试指令,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。

指令示例:

```
TST   R2,#0x01             ; 将寄存器R2的值与立即数0x01按位与,并根据结果设置
                           ; CPSR的标志位,用来判断R2中最低位是否为0
```

(4) TEQ指令

TEQ指令的格式为:

```
TEQ{条件} 操作数1,操作数2
```

TEQ指令是测试指令,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,#0x10             ; 将寄存器R1的值与0x10按位异或,并根据结果设置CPSR的
                           ; 标志位,若Z=1则表示R1的内容是0x10
```

3.2.5 ARM分支指令

在ARM指令集中,没有专门的子程序调用指令,把分支和子程序调用看成是同一种操

作，分支指令用于实现程序流程的跳转，在 ARM 程序中可以通过使用专门的跳转指令或是直接向程序计数器 PC 写入跳转地址值的方法来实现。

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 3 条指令。

1. B 指令

B 指令的格式为：

```
B{条件} 目标地址
```

B 指令是分支指令，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）。

指令实例：

```
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 指令是分支和链接指令，一种可以存储分支处地址的分支指令，可用于子程序调用。具体实现过程是跳转之前，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，返回到跳转指令之后的那个指令处执行。

指令实例：

```
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. BX 指令

BX 指令的格式为：

```
BX{条件} 目标地址
```

BX 指令是分支和交换指令，可以在 ARM 指令集和 Thumb 指令集之间跳转的分支指令。

指令实例：

```
        CODE32                  ; ARM 程序段,32 位编码
ARM1                            ; 语句标号
        ADR R0,THUMB1+1         ; 把 THUMB1 所在的地址赋给 R0
        BX R0                   ; 跳转到 THUMB1 指令集
          …
        CODE16
THUMB1
                  …
        ADR R0,ARM1             ; 把语句标号 ARM1 所在的地址赋给 R0
        BIC R0,R0,#01           ; 末位 R0[0]清零
        BX R0                   ; 跳转到 ARM 指令集
```

3.2.6 ARM 协处理器指令

ARM 作为 32 位处理器,虽然能进行长乘法和乘加等运算,但没有除法指令和更复杂的运算指令。因此, ARM 可以通过外接协处理器来解决此问题,协处理器是一种专门用于进行辅助运算的芯片,其本身除了运算功能以外没有其他功能,因此,不能独立工作,必须和 CPU 一起工作,ARM 处理器可支持多达 16 个协处理器,每个协处理器都有自己的编号,命名为 Pn,每个协处理器都有自己的寄存器,命名为 Cn。

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。

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 指令是协处理器数据操作指令,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   P1,2,C1,C2,C3     ; 命令 1 号(P1)协处理器把自己的寄存器(协处理器寄存器)C2 和 C3 作
                        ; 为操作数,进行第 2 方式的操作,结果放在 C1(协处理器寄存器)中
```

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 指令是协处理器加载指令,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,[R5]          ; 将 ARM 处理器的寄存器 R5 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 指令是协处理器存储指令，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：

```
STCEQ     P2,C4,[R5]      ; 当 Z=1 时,执行将协处理器 P2 的寄存器 C4 中的字数据传送
                          ; 到 ARM 处理器的寄存器 R5 所指向的存储器中
```

4. MCR 和 MRC 指令

MCR 和 MRC 指令的格式为：

```
MCR/MRC {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
```

这两条指令是用来在两个寄存器之间进行数据传送。从 ARM 处理器寄存器中的数据传送到协处理器寄存器使用 MCR 指令；从协处理器寄存器中的数据传送到 ARM 处理器寄存器使用 MRC 指令。

ARM 处理器指定一个寄存器作为传送数据的源或接收数据的目标。协处理器指定两个寄存器，同时可以像 CDP 指令一样指定操作要求。这类指令可以用在浮点数运算和传送中，ARM 处理器把数据传送给协处理器，然后可以从协处理器读出浮点数的计算结果。

指令示例：

```
MCR   P2,3,R2,C4,C5,6     ; 将 ARM 处理器寄存器 R2 中的数据传送到协处理器 P2
                          ; 的寄存器 C4 和 C5 中。具体实现是：指定协处理器 P2 执行第 6
                          ; 种操作,操作数是 C4 和 C5,把操作结果传送给 R2
MRC   P0,3,R2,C4,C5,6     ; 将协处理器 P0 的寄存器中的数据传送到 ARM 处理器寄存器
                          ; 中。具体实现是：指定协处理器 P0 执行第 3 种操作,操作类型
                          ; 是 6,操作数之一是 R2,结果放在 C4 中
```

3.2.7　ARM 软件中断指令

ARM 指令集中的软件中断指令是唯一一条不使用寄存器的 ARM 指令，也是一条可以条件执行的指令。因为 ARM 指令在用户模式中受到很大的局限，有一些资源不能够访问。所以，在需要访问这些资源时，使用软件控制的唯一方法就是使用软件中断指令 SWI。

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                    ;实现中断,指明调用2号功能段
```

执行SWI指令,软件中断进入的是管理模式,中断后会改变程序状态寄存器中的相关位。中断后ARM处理器把0x00000008赋给PC,并把中断处地址保存在LR中,同时把CPSR保存在SPSR中。

3.3 Thumb指令集

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位代码优势的同时,大大地节省了系统的存储空间。

Thumb指令集与ARM指令集在以下几个方面有区别。

- 跳转指令。条件跳转在范围上有更多的限制,转向子程序只具有无条件转移。
- 数据处理指令。对通用寄存器进行操作,操作结果须放入其中一个操作数寄存器,而不是第三个寄存器。
- 单寄存器加载和存储指令。Thumb状态下,单寄存器加载和存储指令只能访问寄存器R0～R7。
- 批量寄存器加载和存储指令。LDM和STM指令可以将任何范围为R0～R7的寄存器子集加载或存储,PUSH和POP指令使用堆栈指针R13作为基址实现满递减堆栈,除R0～R7外,PUSH指令还可以存储连接寄存器R14,并且POP指令可以加载程序指令PC。
- Thumb指令集没有包含进行异常处理时需要的一些指令,因此,在异常中断时还是需要使用ARM指令。这种限制决定了Thumb指令不能单独使用而需要与ARM指令配合使用。

Thumb数据处理指令、存储器访问指令中的加载/存储指令使用方法和ARM指令集中相对应的指令类似,本节不做详细介绍,仅对有区别的指令介绍。

1. PUSH和POP

指令格式:

```
PUSH {低寄存器的全部或其子集}
POP {低寄存器的全部或其子集}
PUSH {低寄存器的全部或其子集,LR}
POP {低寄存器的全部或其子集,PC}
```

这两条指令是栈操作指令,用于在寄存器和堆栈之间进行成组的数据传送,PUSH指令用于把寄存器列表中的寄存器数据推进堆栈;POP指令用于把栈区的数据弹出列表的寄

存器中。

堆栈指针是隐含的地址基址，Thumb 指令中的堆栈是满递减堆栈，堆栈向下增长，堆栈指针总是指向最后入栈的数据。使用入栈指令 PUSH 时，每传送一个数据，堆栈指针就自动减 4；使用出栈指令 POP 时，每传送一个数据，堆栈指针就自动加 4。

POP {低寄存器的全部或其子集，PC}这条指令引起处理器转移到从堆栈弹出给 PC 的地址，这通常是从子程序返回，其中 LR 在子程序开头压进堆栈。这些指令不影响条件码标志。

指令示例：

```
PUSH {R0,R4,R6}          ; 把 R0、R4、R6 的数据顺序推进栈区
PUSH {R4-R7,LR}          ; 把 R4、R5、R6、R7、LR 顺序入栈
POP {R0,R4,R6}           ; 把 R0、R4、R6 的数据弹出栈区
POP {R0-R7,PC}           ; 恢复现场
```

2. 分支指令

(1) B 指令

这是 Thumb 指令集中唯一的有条件指令。

指令格式为：

```
B{条件}目标地址
```

若使用条件，则目标地址必须在当前指令的－256～＋256 字节范围内。若指令是无条件的，则目标地址必须在±2KB 范围内。若条件满足或不使用条件，则 B 指令引起处理器转移到目标地址。目标地址必须在指定限制内。ARM 链接器不能增加代码来产生更长的转移。

指令示例：

```
        CMP R2,#0x20      ; 比较 R2 和 0x20
        BNE START         ; 不相等时，即当 Z=0 时，则跳转
        …
START ADD R3,R4           ; 跳转处
```

(2) BL 指令

BL 指令的格式为：

```
BL 目标地址
```

BL 指令是分支和链接指令，将下一条指令的地址复制到 R14(连接寄存器)，并引起处理器转移到目标地址，但目标地址不可以是 ARM 指令。BL 指令不能转移到当前指令±4MB 以外的地址。

指令示例：

```
        BL START          ; 分支跳转
        ADD R3,R4         ; 分支的下一条指令
        …
START
      ADD R0,R1           ; 指令
```

(3) BX

BX 指令的格式为：

BX 寄存器

BX 指令是分支和交换指令，寄存器的地址是目标地址，其中的位[0]不是地址信息。当寄存器的位[0]为 1 时，表明目标地址处是 Thumb 指令；当寄存器的位[0]为 0 时，表明目标地址处是 ARM 指令，此时，要求字对准。

指令示例：

```
        CODE16                  ; Thumb 程序段
        ADR   R0,ARM1           ; 把标号 ARM1 处地址赋给 R0
        BIC   R0,#01            ; 清零 R0 的位[0]
        BX    R0                ; 跳转到 ARM1 指令集
LET1    …
        CODE32
ARM1
              …
        ADD   R3,R4             ; ARM 语句
        ADR   R0,LET+1          ; 把 LET1 所在的地址赋给 R0,置 R0 的位[0]
        BX    R0                ; 跳转到 ARM 指令集
```

3. SWI 指令

指令格式：

SWI 立即数

SWI 指令为软件中断指令，用于产生软件中断，即能够引起 SWI 异常。这意味着处理器状态切换到 ARM 态；处理器模式切换到管理模式，CPSR 保存到管理模式的 SPSR 中，执行转移到 SWI 向量地址。立即数要求是一个 8 位的无符号数，范围是 0～255 之间的整数。

指令示例：

```
SWI 12                          ; 产生软件中断,进入管理方式时带入参数 12,
                                ; 作为 SWI 中断请求号
```

3.4 本章小结

本章系统地介绍了 ARM9 处理器支持的 8 种寻址方式，ARM9 处理器上使用的 ARM/Thumb 指令集中的基本指令，以及各指令的应用场合及方法。通过本章内容的学习，要求读者掌握各种指令的含义和用法，能够阅读汇编指令编写的程序，编写简单的汇编程序。

3.5 习 题 3

1. ARM 指令有哪几种寻址方式？试分别叙述其各自的特点并举例说明。

2. 简述 ARM9 指令集的分类。

3. 简述 ARM9 指令的分类有哪些。

4. 请叙述处理器如何实现 ARM 状态和 Thumb 状态的切换。

5. ARM9 指令集支持哪几种协处理器指令？试分别简述并列举特点。

6. 简述 ARM9 的 LDM/STM 堆栈指令中空、满、递增、递减的含义。

7. 假设 R4 的内容为 0x6000，寄存器 R5、R6、R7 内容分别为 0x01、0x02、0x03，存储器内容为空。执行下列指令后，PC 内容如何变化？存储器及 R4、R5、R6、R7 的内容如何变化？

```
STMIB   R4!{R5,R6,R7}
LDMIA   R4!{R5,R6,R7}
```

8. BIC 指令的作用是什么？

9. CMP 指令的操作数是什么？写一个程序，判断 R1 的值是否大于 0x20，是则将 R1 减去 0x10。

10. BX 和 BL 指令有什么不同？