第3章 chapter 3

ARM920T 指令系统

本章主要内容如下：

(1) ARM 指令集概述，介绍 ARM 指令集全部指令编码及条件域；

(2) ARM 指令，讲述 ARM 指令的编码格式、指令含义、汇编格式和使用举例。

S3C2410A 嵌入式微处理器片内使用了 ARM920T 内核，因此 S3C2410A 使用 ARM920T 所支持的指令系统。ARM920T 指令系统的指令集结构版本为 v4T，ARM920T 指令系统含有 v4T 以上指令集结构版本的基础指令，在 v4T 以上指令集结构版本的微处理器中都可以运行。

ARM920T 是典型的 RISC 处理器，它实现了装入/存储结构，只有装入/存储指令才能访问存储器(内存储器)。而数据处理指令仅仅对寄存器内容进行操作。

ARM920T 处理器支持 32 位寻址空间。

ARM920T 支持指令长度为 32 位的 ARM 指令集和指令长度为 16 位的 Thumb 指令集。从功能上讲，Thumb 指令集是 ARM 指令集主要部分的一个子集。

当处理器正在执行 Thumb 指令时，称为处理器在 Thumb 状态操作。

当处理器正在执行 ARM 指令时，称为处理器在 ARM 状态操作。

ARM920T 处理器总是从 ARM 状态开始，必须用 BX 指令明确地转换到 Thumb 状态。

程序计数器 PC 也称为 R15 寄存器，在 ARM 状态，对每条指令以 1 个字(4 字节)作为地址增量；在 Thumb 状态，以 2 字节作为地址增量。

本章对 ARM 指令集中的每条指令给予详细描述。本章不去描述 Thumb 指令，有兴趣的读者可以参见参考文献[1]。

3.1 ARM 指令集概述

3.1.1 ARM 指令集概述

所有 ARM 指令长度均为 32 位，在存储器中以字边界对齐存储。因此在 ARM 状态，指令地址的最低 2 位总是为 0，即 bit[1:0]=00。所有的 ARM 指令，指令中凡涉及程

序地址操作数的,最低2位均被忽略,只有BX指令除外。BX指令用最低位确定分支处的代码,如果bit[0]=1,则分支到Thumb代码;如果bit[0]=0,则分支到ARM代码。

1. ARM指令分组

ARM指令从功能上能被分成以下6组。

1)分支指令

分支指令,也称为转移或跳转指令,可以向小地址方向分支形成一个循环;可以向大地址方向分支,根据CPSR中不同的条件码标志分支到不同的程序流;可以分支到子程序;可以通过分支把处理器从ARM状态转换到Thumb状态。

2)数据处理指令

这些指令对通用寄存器中的数据进行操作,不允许使用存储器操作数。通常对2个寄存器,执行像加、减或按位的某种逻辑操作,将结果存入第3个寄存器。

长乘指令将2个32位数相乘,结果为64位,保存在2个寄存器中。

3)状态寄存器访问指令

这些指令把CPSR或某个SPSR的内容送到通用寄存器,或者把通用寄存器的内容送到CPSR或某个SPSR。

4)单个寄存器装入或存储指令

这些指令能够:

- 从存储器装入一个字数据到寄存器,或保存寄存器的值到存储器;
- 从存储器装入一字节数据到寄存器bit[7:0],或保存寄存器bit[7:0]的值到存储器;
- 从存储器装入一个半字数据到寄存器bit[15:0],或保存寄存器bit[15:0]的值到存储器;
- 带符号扩展的字节/半字装入,字节或半字数据从存储器装入寄存器bit[7:0]或bit[15:0],高位用符号位扩展。

另外,存储器与寄存器数据交换指令可以在存储器和寄存器之间交换字节或字数据。

5)块数据装入或存储指令

这些指令从存储器装入数据到通用寄存器组的部分或全部寄存器,或保存通用寄存器组的部分或全部寄存器的内容到存储器。

6)协处理器指令

协处理器指令支持一种通用的扩展ARM结构的方法。支持存储器与协处理器寄存器之间数据的传输;支持ARM寄存器与协处理器寄存器之间数据的传输;支持指定协处理器内部执行某种操作。如果不存在协处理器,将产生未定义指令异常中断,由未定义指令陷阱来仿真协处理器指令的执行。

2. ARM指令的能力

以下介绍ARM指令的主要能力,详细内容见3.2节。

1）条件执行

所有 ARM 指令均可以在指令操作码助记符后，跟随一个条件码助记符后缀，依据 CPSR 中的条件码标志，有条件地被执行，而不需要使用分支指令实现条件分支。

数据处理指令可以指定设置或不设置 CPSR 中的条件码标志。

2）寄存器访问

在 ARM 状态，所有指令能够存取 R0～R14，大部分指令也允许存取 R15（PC）。MRS 和 MSR 指令能够传送 CPSR 或 SPSR 的内容到通用寄存器，在通用寄存器中通过使用通常的数据处理指令，对它们进行操作，然后可以再写回到 CPSR 或 SPSR 中。

3）在线式桶形移位器（barrel shifter）的访问

ARM 结构的逻辑单元有一个 32 位的桶形移位器，它有能力进行一般移位和循环移位。详见 3.2.3 节和 3.2.7 节。

3.1.2 ARM 指令集全部指令编码及条件域简介

1. ARM 指令集全部指令编码格式

ARM 指令集全部指令编码格式见图 3-1。

31 28	27	26	25	24 21	20	19 16	15 12	11 8	7 4	3 0	
Cond	0	0	I	OpCode	S	Rn	Rd	Operand2			注1
Cond	0	0	0	0 0 0 A	S	Rd	Rn	Rs	1 0 0 1	Rm	注2
Cond	0	0	0	0 1 U A	S	RdHi	RdLo	Rs	1 0 0 1	Rm	注3
Cond	0	0	0	1 0 B 0	0	Rn	Rd	0 0 0 0	1 0 0 1	Rm	注4
Cond	0	0	0	1 0 0 1	0	1 1 1 1	1 1 1 1	1 1 1 1	0 0 0 1	Rn	注5
Cond	0	0	0	P U 0 W	L	Rn	Rd	0 0 0 0	1 S H 1	Rm	注6
Cond	0	0	0	P U 1 W	L	Rn	Rd	Offset	1 S H 1	Offset	注7
Cond	0	1	1	P U B W	L	Rn	Rd	Offset			注8
Cond	0	1	1						1		注9
Cond	1	0	0	P U S W	L	Rn	Register List				注10
Cond	1	0	1	L	Offset						注11
Cond	1	1	0	P U N W	L	Rn	CRd	CP#	Offset		注12
Cond	1	1	1	0	CP Opc	CRn	CRd	CP#	CP	0 CRm	注13
Cond	1	1	1	0	CP Opc	L	CRn	Rd	CP#	CP 1 CRm	注14
Cond	1	1	1	1	lgnored by processor						注15

图 3-1 ARM 指令集全部指令编码格式

图 3-1 中注 1～注 15 对应指令名称如下。

注 1：数据处理/程序状态寄存器传送指令；

注 2：乘、乘累加指令；

注 3：长乘、长乘累加指令；

注 4：单个数据交换指令；

注5：分支并且转换状态指令；

注6：半字、带符号字节/半字传送指令(寄存器偏移量)；

注7：半字、带符号字节/半字传送指令(立即数偏移量)；

注8：单个数据传送指令；

注9：未定义指令；

注10：块数据传送指令；

注11：分支、分支并且连接指令；

注12：协处理器数据传送指令；

注13：协处理器数据操作指令；

注14：协处理器寄存器传送指令；

注15：软件中断指令。

由图3-1可知，全部指令均为等长的32位，因此在存储器保存一条指令占4字节空间，保存指令的地址必须以字(4字节)边界对齐。由于采用了RISC结构，指令编码较为简单。除条件域外，指令中各个域的含义在3.2节中讲述。

2. 指令编码中的条件域

参见图3-1，指令编码格式中的bit[31:28]称为条件域。在ARM状态，所有指令都要根据CPSR中的条件码标志(简称标志位)和指令中条件域指定的内容，有条件地执行。指令中条件域bit[31:28]确定在哪一种情况下这条指令被执行。如果C、N、Z和V标志的状态满足指令中条件域编码要求，指令被执行；否则指令被忽略。

有15种可能的条件，每一种由2个字符代替，称为条件码助记符后缀(简称条件码助记符)，可以附加在指令助记符后，如表3-1所示。

表3-1 条件域编码与助记符后缀对应关系和含义

指令bit[31:28]	助记符后缀	CPSR中的条件码标志	含　义
0000	EQ	Z=1	相等
0001	NE	Z=0	不等
0010	CS/HS	C=1	无符号数高于或等于
0011	CC/LO	C=0	无符号数低于
0100	MI	N=1	负
0101	PL	N=0	正或0
0110	VS	V=1	溢出
0111	VC	V=0	不溢出
1000	HI	C=1并且Z=0	无符号数高于
1001	LS	C=0或Z=1	无符号数低于或等于
1010	GE	N=V	带符号数大于或等于

续表

指令 bit[31:28]	助记符后缀	CPSR 中的条件码标志	含 义
1011	LT	N<>V	带符号数小于
1100	GT	Z=0 并且(N=V)	带符号数大于
1101	LE	Z=1 或(N<>V)	带符号数小于或等于
1110	AL	忽略	总是执行

例如,分支指令 B 如果附加条件码助记符后缀 EQ,写作 BEQ,表示相等时(即 Z=1)这条指令才执行;如果 Z< >1,则这条指令不被执行,指令被忽略。

表 3-1 中 bit[31:28]代码为 1111 的情况没有使用,保留。表中代码为 1110 的,对应后缀为 AL,AL 可以在指令中出现,也可以不出现,两种情况都表示这条指令无条件地被执行,如指令 B 和 BAL 都表示无条件地执行指令。

3.2 ARM 指 令

本节按指令的编码格式划分,讲述 ARM 指令集中各指令的编码格式、指令含义、指令汇编格式和使用举例。本节仅对编码格式较为复杂的指令,以图示的方式给出编码格式。另外在讲述协处理器指令前,对协处理器进行简单介绍。

指令编码格式中的 bit[31:28]为条件域。所有指令都要根据 CPSR 中的条件码标志和指令中的条件域指定的内容,有条件地执行。为简单起见,以下介绍各指令时不再重复这部分内容。

3.2.1 分支并且转换状态指令(BX)

分支并且转换状态指令(BX),在指令中指定了一个 Rn 寄存器,将 Rn 内容复制到 PC,同时使 PC[0]=0。如果 Rn[0]=1,将处理器状态转换成 Thumb 状态,把目标地址处的代码解释为 Thumb 代码;如果 Rn[0]=0,将处理器状态转换成 ARM 状态,把目标地址处的代码解释为 ARM 代码。

1. 指令含义

通过复制一个通用寄存器 Rn 的内容到程序计数器 PC,指令实现分支功能。这条指令也允许处理器状态被转换。

2. 指令汇编格式

指令汇编格式如下:

```
BX{cond} Rn
```

其中:

{cond}　表示两个字符的条件码助记符,详见表 3-1。

Rn　表示合法的寄存器编号。Rn 的内容为分支目的地址。其中 bit[0]用于指示后续指令为 ARM 或 Thumb 指令。

应避免使用 R15 作为操作数 Rn,如果需要这样做,可以使用 MOV PC,PC 或 ADD PC,PC,#0 指令。

3. 使用举例

【例 3.1】 处理器从执行 ARM 指令代码处分支到标号为 Goto_THUMB 处,并且执行 Thumb 指令代码,然后又返回到 Back_ARM 处,执行 ARM 指令代码。

```
;假定处理器当前正在执行 ARM 指令
    ADR R1,Goto_THUMB+1          ;将分支目标地址送 R1,使 R1 的 bit[0]=1
    BX R1                        ;分支并且转换为 Thumb 状态
    ⋮
    CODE16                       ;汇编以下代码为 Thumb 指令
Goto_THUMB                       ;分支目标地址标号
    ⋮                            ;Thumb 指令代码
    ADR R2,Back_ARM              ;将分支目标地址送 R2,并且 R2 的 bit[0]=0
    BX R2                        ;分支且转换为 ARM 状态
    ⋮
    ALIGN                        ;字对齐
    CODE32                       ;汇编以下代码为 ARM 指令
Back_ARM                         ;分支目标地址标号
    ⋮                            ;ARM 指令代码
```

3.2.2 分支、分支并且连接指令(B、BL)

分支指令(B)使程序分支(转移)到确定的地址处执行程序。

分支并且连接指令(BL)除了使程序分支(转移)到确定的地址处执行程序外,还要保存返回地址到 LR 寄存器,即把 BL 指令的下一条指令的地址送 LR。

上述两条指令,只允许分支到 ARM 指令代码处,不允许分支到 Thumb 指令代码处。

1. 指令含义

对于分支指令(B),指令能在±32MB 地址范围内实现分支。

对于分支并且连接指令(BL),执行指令会将 PC 值写入当前寄存器组的连接寄存器 R14,写入的 PC 值是经过调整的、跟在分支并且连接指令后的指令的地址,同时 R14 的 bit[1:0]被清 0。

使用分支并且连接指令(BL)可以调用一个子程序,为了从子程序返回,如果 R14(LR)在子程序中没有被修改,可以使用 MOV PC,R14 指令实现返回。

2. 指令汇编格式

指令汇编格式如下：

```
B{L}{cond} <expression>
```

其中：

{L}　　出现 L 表示分支并且连接指令，不出现 L 表示分支指令。

{cond}　　条件码助记符。

＜expression＞　目标地址，由汇编器计算偏移量。

3. 使用举例

【例 3.2】 使用分支指令使部分代码循环 5 次。

```
        MOV     R0,#5                   ;R0 值为 5
Loop1
        SUBS    R0,#1                   ;R0 减 1 送 R0,设置标志位
        BNE     Loop1                   ;条件执行,不为 0 则分支到标号 Loop1 处
```

【例 3.3】 使用分支并且连接指令调用不同的子程序。

```
CMP       R0,#0                         ;比较,设置标志位
BLEQ      SUBEQPROG                     ;相等,则调用 SUBEQPROG
BLGT      SUBGTPROG                     ;大于,则调用 SUBGTPROG
BL        SUBLTPROG                     ;小于,则调用 SUBLTPROG
```

3.2.3 数据处理指令

ARM 数据处理指令可以分为 3 类：数据传送指令(如 MOV 和 MVN)、算术逻辑操作指令(如 ADD、SUB 或 AND 等)和比较指令(如 CMP 和 TST 等)。

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。

对于数据传送指令 MOV 和 MVN，指令中指定的目的寄存器的内容被覆盖，如果目的寄存器指定了 PC，如 MOV PC，R14，则可以实现程序的转移。

数据传送指令可以实现寄存器到寄存器，立即数到寄存器的传送。

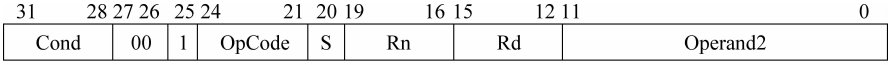
算术逻辑操作指令通常对指定的两个寄存器(或 1 个寄存器、1 个立即数)进行操作，结果存到第 3 个寄存器，允许选择修改或不修改 CPSR 中的条件码标志。

比较指令 TEQ、TST、CMP 和 CMN，通常对指定的两个寄存器(或 1 个寄存器、1 个立即数)进行比较，比较结果不保存到寄存器，只影响 CPSR 中的条件码标志。

上述指令通常允许对指定的操作数进行移位操作。

1. 指令编码格式

数据处理指令编码格式见图 3-2。



[15:12] 目的寄存器

[19:16] 第1操作数寄存器

[20] 设置条件码

0=不设置条件码　　1=设置条件码

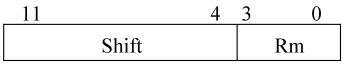
[24:21] 操作码

0000=AND	Rd=Op1 AND Op2;	1000=TST	由Op1 AND Op2设置条件码标志
0001=EOR	Rd=Op1 EOR Op2;	1001=TEQ	由Op1 EOR Op2设置条件码标志
0010=SUB	Rd=Op1−Op2;	1010=CMP	由Op1−Op2设置条件码标志
0011=RSB	Rd=Op2−Op1;	1011=CMN	由Op1+Op2设置条件码标志
0100=ADD	Rd=Op1+Op2;	1100=ORR	Rd=Op1 OR Op2
0101=ADC	Rd=Op1+Op2+C;	1101=MOV	Rd=Op2
0110=SBC	Rd=Op1−Op2+C−1;	1110=BIC	Rd=Op1 AND NOT Op2
0111=RSC	Rd=Op2−Op1+C−1;	1111=MVN	Rd=NOT Op2

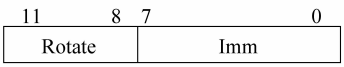
[25] 立即操作数

0=Operand2使用寄存器　　1=Operand2使用立即数

[11:0] Operand2类型选择



[3:0] 第2操作数寄存器　[11:4] 对Rm的移位操作



[7:0] 8位无符号立即数 [11:8] 对立即数的移位操作

[31:28] 条件域

图 3-2　数据处理指令编码格式

图 3-2 中,第 1 操作数总是寄存器 Rn。Rd 称为目的寄存器,TST、TEQ、CMP 和 CMN 指令不送结果到目的寄存器 Rd,其他指令产生的结果送 Rd。

第 2 操作数 Operand2 可以是寄存器 Rm 的值经过移位产生的 32 位值,或 8 位立即数经过循环右移产生的 32 位的值,指令中 bit[25]的值用来选择 Rm 或 8 位立即数。

CPSR 中的条件码标志可能被保护或由指令的结果设置,取决于指令中 bit[20]的值。但是对于指令 TST、TEQ、CMP 和 CMN,汇编器产生的指令编码一定会把指令的 bit[20]置 1,在执行指令时,由测试结果设置 CPSR 中的条件码标志。

2. 指令含义

1) 各指令含义

数据处理指令依指令编码格式中 bit[24:21]分为 16 条指令,包括数据传送指令、算术逻辑操作指令和比较指令。各条指令含义见表 3-2。

表 3-2 数据处理各指令含义

指令格式	指令含义	指令功能描述
MOV{cond}{S} Rd,<Op2>	数据传送	Rd=<Op2>
MVN{cond}{S} Rd,<Op2>	数据求反传送	Rd=NOT <Op2>
ADD{cond}{S} Rd,Rn,<Op2>	加	Rd=Rn+<Op2>
ADC{cond}{S} Rd,Rn,<Op2>	带进位加	Rd=Rn+<Op2>+C
SUB{cond}{S} Rd,Rn,<Op2>	减	Rd=Rn-<Op2>
SBC{cond}{S} Rd,Rn,<Op2>	带进(借)位减	Rd=Rn-<Op2>+C-1
RSB{cond}{S} Rd,Rn,<Op2>	逆向减	Rd=<Op2>-Rn
RSC{cond}{S} Rd,Rn,<Op2>	带进(借)位逆向减	Rd=<Op2>-Rn+C-1
CMP{cond} Rn,<Op2>	比较,做减法	Rn-<Op2>,只设置 CPSR
CMN{cond} Rn,<Op2>	负数比较,做加法	Rn+<Op2>,只设置 CPSR
TST{cond} Rn,<Op2>	测试,按位逻辑与	Rn AND <Op2>,只设置 CPSR
TEQ{cond} Rn,<Op2>	测相等,按位逻辑异或	Rn EOR <Op2>,只设置 CPSR
AND{cond}{S} Rd,Rn,<Op2>	按位逻辑与	Rd=Rn AND <Op2>
EOR{cond}{S} Rd,Rn,<Op2>	按位逻辑异或	Rd=Rn EOR <Op2>
ORR{cond}{S} Rd,Rn,<Op2>	按位逻辑或	Rd=Rn OR <Op2>
BIC{cond}{S} Rd,Rn,<Op2>	位清 0	Rd=Rn AND NOT<Op2>

注：表中<Op2>表示图 3-2 中 Operand2,Operand2 的取得在随后内容中介绍。

2) 指令对 CPSR 中条件码标志位的影响

逻辑操作对操作数的对应位,执行按位操作。

在逻辑操作(AND、EOR、TST、TEQ、ORR、BIC)和数据传送操作(MOV、MVN)指令中,如果 S 位被置 1(并且 Rd 不是 R15),则 CPSR 中的 V 标志位不受影响;C 标志位由桶形移位器产生的 carry out 设置;当指令操作结果为全 0 时 Z 标志位被设置;N 标志位由指令操作结果的 bit[31]的值设置。

算术操作(SUB、RSB、ADD、ADC、SBC、RSC、CMP、CMN)指令中,每个操作数被看作 32 位整数(无符号数或带符号数的 2 的补码),如果指令中 S 位被置 1(并且 Rd 不是 R15),在发生溢出时,CPSR 中的 V 标志位被设置;C 标志位由 ALU 的 bit[31]产生的进位设置;如果指令操作结果为全 0 时,Z 标志位被设置;N 标志位将被设置成指令操作结果的 bit[31]的值。

3) 对寄存器 Rm 内容进行移位,结果作为 Operand2 的值

在图 3-2 的指令编码格式中,可以使用 bit[3:0]指定被移位的寄存器 Rm,bit[11:4]指定对 Rm 的移位量。bit[11:4]指定移位量的方法有两种,见图 3-3。

如图 3-3 所示,一种是直接使用 bit[11:7]中的值作为移位量,另一种是由 bit[11:8]

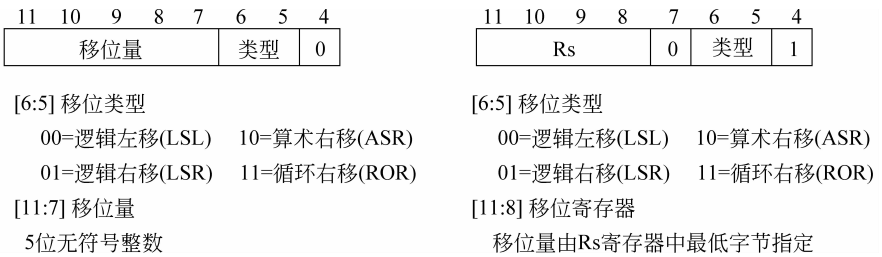


图 3-3 移位操作

指定 Rs 寄存器中最低一字节内容作为移位量,对 Rm 内容进行移位后得到的值作为 Operand2 的值。移位在桶形移位器中进行,移位与传送操作同时进行。

移位类型有逻辑左移(LSL)、逻辑右移(LSR)、算术右移(ASR)和循环右移(ROR)。当循环右移的移位量为 0 时,称为扩展循环右移(RRX)。

(1) 根据指令中 bit[11:7]指定的移位量对 Rm 移位。

参见图 3-2 和图 3-3,指令中 bit[11:7]指定的移位量为 0～31,对 Rm 内容进行移位产生的结果作为 Operand2 的值,有以下几种类型。参见图 3-4～图 3-8 中,carry out 送 CPSR 中的 C 位。

① 逻辑左移(LSL)。每一次移位,将最高位移到 CPSR 中的 C 位,其余各位左移一位,最低位补 0。如对 Rm 逻辑左移 6 次(LSL #6),产生的结果见图 3-4。当移位次数为 0 时(LSL #0),CPSR 中 C 位的值不变,Rm 的内容直接作为 Operand2 使用。

图 3-4 逻辑左移(LSL #6)

② 逻辑右移(LSR)。每一次移位,将最低位移到 CPSR 中的 C 位,其余各位右移一位,最高位补 0。如对 Rm 逻辑右移 6 次(LSR #6),产生的结果见图 3-5。

图 3-5 逻辑右移(LSR #6)

当移位次数为 0 时(LSR #0),看作移位 32 次。将全 0 送 Operand2,Rm[31]送 CPSR 中的 C 位。

③ 算术右移(ASR)。每一次移位,将符号位即最高位 bit[31]右移一位,同时保留 bit[31]位的值不变,其余操作同逻辑右移。如对 Rm 算术右移 6 次(ASR ＃6),产生的结果见图 3-6。

当移位次数为 0 时(ASR ＃0),看作移位 32 次。将 Rm[31]送 CPSR 中的 C 位。如果 Rm[31]＝0,则全 0 送 Operand2;如果 Rm[31]＝1,则全 1 送 Operand2。

图 3-6 算术右移(ASR ＃6)

④ 循环右移(ROR)。每一次移位,将最低位移到 CPSR 中的 C 位的同时,也移到最高位,其余各位右移一位。如对 Rm 循环右移 6 次(ROR ＃6),产生的结果见图 3-7。

在循环右移中,ROR ＃0 的功能有些特殊,称为扩展循环右移,写作 RRX。它的功能是将 CPSR 中的 C 位移入 Operand2 的最高位,原 Rm 内容右移一位,Rm 最低位移到 CPSR 中的 C 位,见图 3-8。

图 3-7 循环右移(ROR ＃6)

图 3-8 扩展循环右移(RRX)

在使用指令中 bit[11:7]指定对 Rm 的移位量,bit[6:5]指定移位类型时,指令汇编格式举例见表 3-3。

(2) 使用指令中 bit[11:8]指定 Rs 寄存器,且用 Rs 中最低字节指定移位量。

参见图 3-2 和图 3-3,由指令中 bit[11:8]指定 Rs 寄存器,移位量保存在 Rs 寄存器的最低字节,对 Rm 寄存器的内容进行移位,产生的结果作为 Operand2 的值。

表 3-3 指令 bit[11:7]和 bit[6:5]指定 Rm 移位量和移位类型举例

指定对 Rm 的移位量和移位类型	指 令 举 例	指 令 含 义
Rm,LSL #5bit_shift_Imm	ADD R0,R2,R3,LSL #1	R3 的值逻辑左移 1 位,加 R2,和送 R0
Rm,LSR #5bit_shift_Imm	SUB R0,R2,R3,LSR #2	R3 的值逻辑右移 2 位,从 R2 中减去,差送 R0
Rm,ASR #5bit_shift_Imm	MOV R1,R0,ASR #2	R0 的值算术右移 2 位,送 R1
Rm,ROR #5bit_shift_Imm	SUB R1,R2,R4,ROR #6	R4 的值循环右移 6 位,从 R2 中减去,差送 R1
Rm,RRX	AND R2,R3,R4,RRX	R4 的值扩展循环右移,和 R3 与的结果送 R2

注:表中#5bit_shift_Imm 表示 5 位二进制立即数,作为移位量,对应指令中 bit[11:7]。

Rs 可以选择除 R15 外的任一通用寄存器。

如果 Rs 中指定的移位次数为 0,那么不改变 Rm 的内容作为 Operand2,并且 CPSR 中 C 位的值作为 carry out,即 C 位的值不变。

如果 Rs 中最低字节指定的移位次数为 1～31,进行的移位操作与产生的结果参见图 3-4～图 3-7。

如果 Rs 中最低字节指定的移位次数大于或等于 32,产生的结果如下。

- 对 LSL,如果移位次数等于 32,移位结果 Operand2 为全 0,Rm[0]作为 carry out。
- 对 LSL,如果移位次数大于 32,移位结果 Operand2 为全 0,carry out 为 0。
- 对 LSR,如果移位次数等于 32,移位结果 Operand2 为全 0,Rm[31]作为 carry out。
- 对 LSR,如果移位次数大于 32,移位结果 Operand2 为全 0,carry out 为 0。
- 对 ASR,如果移位次数大于或等于 32,用 Rm[31]填充 Operand2 各位,用 Rm[31] 作为 carry out。
- 对 ROR,如果移位次数等于 32,移位结果 Operand2 等于 Rm 的值,carry out 等于 Rm[31]。
- 对于 ROR,如果移位次数大于 32,用移位次数重复减 32,直到它们的差为 1～32,用这个值作为移位次数,移位结果如前述。

对于上述各种情况,carry out 的值均送往 CPSR 中的进位标志 C。

使用 Rs 指定移位量时,指令中 bit[6:5]指定移位类型,指令汇编格式举例见表 3-4。

表 3-4 用 Rs 指定 Rm 的移位量和指令中 bit[6:5]指定移位类型举例

指定 Rm 的移位量和移位类型	指 令 举 例	指 令 含 义
Rm, LSL Rs	ADD R0,R1,R2,LSL R3	移位量在 R3,R2 逻辑左移,加 R1,和送 R0
Rm, LSR Rs	SUB R0,R1,R2,LSR R4	移位量在 R4,R2 逻辑右移,从 R1 中减去,差送 R0
Rm, ASR Rs	AND R1,R2,R3,ASR R0	移位量在 R0,R3 算术右移,和 R2 逻辑与,结果送 R1
Rm, ROR Rs	MOV R2,R4,ROR R0	移位量在 R0,R4 循环右移,送 R2

4) 对指令中 bit[7:0]指定的 8 位无符号立即数循环右移

参见图 3-2,对指令中 bit[7:0]指定的 8 位无符号立即数进行循环右移时,用 bit[11:8]指定移位量,它是一个 4 位无符号整数。

进行移位操作时,要把指令中 bit[7:0]指定的 8 位无符号立即数作为最低字节,高位 bit[31:8]用 0 扩展,形成一个 32 位数,对这个 32 位数进行循环右移。移位的次数,由指令中 bit[11:8]指定的 4 位无符号数乘以 2 得到,分别为 0,2,4,…,30。移位过程参见图 3-7。

例如,对于指令

```
MOV R0,#0xff000000                ;汇编后等价指令为 MOV R0,#0xff,8
```

由于指令长度只有 32 位,指令无法直接得到一个 32 位立即数。但是在指令汇编格式中,允许使用一个 32 位立即数,如本例中#0xff000000,汇编器试图将这个立即数转变成一个 8 位的立即数,存放在指令的 bit[7:0];另外转变出一个移位量,存放在指令的 bit[11:8]。指令执行时用这个移位量乘以 2 得到移位次数。如果一个 32 位立即数不能转变成这样的格式,汇编器报告错误。在本例中,转变出的指令 bit[7:0]为 0xff,bit[11:8]为 0x04。

5) 关于 R15 和 CPSR 中的条件码标志

(1) 当目的寄存器 Rd 不是指定 R15 时,CPSR 中的条件码标志,如前所述,可能被更改。

(2) 当 Rd 指定为 R15 时,并且指令中 bit[20]即 S=0,表示指令操作结果不影响 CPSR 时,那么指令操作结果送 R15,且 CPSR 不受影响。

(3) 当 Rd 指定为 R15 时,并且指令中 bit[20]即 S=1,表示指令操作结果影响 CPSR 时,那么指令操作结果送 R15,并且当前方式的 SPSR 送到 CPSR。这样可以恢复 PC 和 CPSR,这种指令的使用,只能在非用户方式。

(4) 如果 R15 被用作操作数而不是目的寄存器 Rd,可以直接使用这个寄存器。但是由于流水线指令预取的功能,PC 的值即 R15 的值,对于指令中指定了移位量的情况,将是当前指令地址加 8 的值;而对于使用寄存器指定移位量的情况,将是当前指令地址加 12 的值。

(5) TEQ、TST、CMP 和 CMN 指令对 CPSR 的影响。

通常在指令助记符后缀中如果出现 S,表示操作结果影响 CPSR 中的条件码标志;如果不出现 S,表示不影响。但是即使不出现 S,汇编器对 TEQ、TST、CMP 和 CMN 指令,也会像出现 S 那样,将指令的 bit[20]设置为 1,使得这 4 条指令的执行会影响 CPSR 中相应的条件码标志。

3. 指令汇编格式

对于数据传送指令 MOV 和 MVN,格式为:

```
<opcode>{cond}{S} Rd,<Op2>
```

对于不保存结果,只影响 CPSR 中条件码标志的指令 CMP、CMN、TEQ 和 TST,格式为:

```
<opcode>{cond} Rn,<Op2>
```

对于指令 AND、EOR、SUB、RSB、ADD、ADC、SBC、RSC、ORR 和 BIC，格式为：

```
<opcode>{cond}{S} Rd,Rn,<Op2>
```

其中：

＜opcode＞	表示指令助记符。
＜Op2＞	由 Rm{,＜shift＞}或＜＃expression＞组成。
{cond}	表示条件码助记符。
{S}	表示由指令操作结果设置 CPSR 中的条件码标志。CMP、CMN、TEQ 和 TST 指令，无论出现 S 与否，汇编器自动使指令 bit[20]＝1。
Rd、Rn 和 Rm	表示寄存器编号。Rd 为目的寄存器，Rn 为第 1 操作数寄存器，Rm 的值被移位后作为第 2 操作数。
＜＃expression＞	如果被使用，汇编器将试图产生一个可移位的 8 位立即数及一个移位量，与 expression 匹配。如果无法匹配，将产生一个错误信息。
＜shift＞	由＜shiftname＞＜register＞或＜shiftname＞＃expression，或 RRX 组成。
＜shiftname＞	可以是 LSL、LSR、ASR、ROR 或 ASL。算术左移(ASL)与逻辑左移(LSL)功能相同，汇编器产生相同的代码。

4. 使用举例

1) 数据传送和数据求反传送指令举例

```
MOVS    R4,R3,LSL #2          ;R4 等于 R3 逻辑左移 2 位的值,设置标志位
MOVS    PC,R14                ;PC=R14,且 CPSR=SPSR_<mode>,用于从异常返回
MOV     R15,LR                ;PC=R14,用于从子程序返回
MVN     R0,R1                 ;R1 的值求反送 R0
MVN     R2,#0xf0              ;R2=0xffffff0f
MVN     R0,#0                 ;R0=0xffffffff,即 R0=-1
MOVS    R4,R4,LSR #32         ;R4 的 bit[31]送 CPSR 的 C 位,R4 结果为 0
```

2) 算术操作指令举例

```
ADDEQ   R3,R5,R6              ;执行本指令前,先判断 Z 标志。如果 Z=1,则 R3=R5+R6;
                              ;如果 Z=0,则忽略本条指令
ADDS    R2,R2,#2              ;R2=R2+2,设置标志位
ADDS    R2,R1,R0,LSL #2       ;R2=R1+R0<<2,设置标志位

;以下两条指令实现 64 位二进制数的加法：R2、R1=R2、R1+R4、R3
ADDS    R1,R1,R3              ;R1=R1+R3,设置标志位
ADC     R2,R2,R4              ;R2=R2+R4+C
```

```
SUB     R0,R1,R2              ;R0=R1-R2
SUB     R2,R1,#0x10           ;R2=R1-0x10
SUBS    R1,R1,#2              ;R1=R1-2,设置标志位
SUBS    R4,R5,R7,LSR R2       ;逻辑右移 R7,移位次数在 R2 中最低字节,
                              ;移位后的值作为<Op2>,R4=R5-<Op2>

;以下 3 条指令实现 96 位二进制数的减法: R6、R2、R1=R6、R2、R1-R5、R4、R3
SUBS    R1,R1,R3              ;R1=R1-R3,设置标志位
SBCS    R2,R2,R4              ;R2=R2-R4+C-1,设置标志位
SBC     R6,R6,R5              ;R6=R6-R5+C-1

RSB     R1,R0,#0xff00         ;R1=0xff00-R0
RSBS    R2,R1,R0,LSL #2       ;R2=R0<<2-R1,设置标志位
RSC     R2,R0,#0              ;R2=0-R0+C-1
```

3）逻辑操作指令举例

```
AND     R3,R2,R4              ;R3=R2 AND R4
ANDS    R1,R1,#0x0f           ;R1=R1 AND 0x0f,设置标志位
ORR     R1,R1,#0xff           ;将 R1 的最低 8 位置 1
EOR     R1,R1,#0xff           ;将 R1 的最低 8 位求反
BIC     R1,R0,#0x0f           ;R1=R0 AND NOT 0x0f
ANDEQ   R5,R6,R7,RRX          ;条件执行,R7 的值扩展循环右移,和 R6 与的结果送 R5
```

4）比较与测试指令举例

```
CMP     R2,#0x01              ;R2-0x01 的结果不保存,设置标志位,
                              ;汇编器自动使指令的 bit[20]=1
CMPS    R2,#0x01              ;R2-0x01 的结果不保存,设置标志位
CMN     R2,#1                 ;R2 的值加 1,用来判断 R2 的值是-1 的补码,如果是,
                              ;CPSR 中的 Z 位被置 1
TST     R2,#0x01              ;R2 AND 0x01,设置标志位,用来判断 R2 中
                              ;最低位是否为 0
TEQ     R1,R2                 ;R1 EOR R2,设置标志位,用于测试 R1 和 R2 是否相等
TEQS    R4,#3                 ;测试 R4 是否等于 3。即使指令中不写 S,汇编器也会
                              ;自动使指令中的 bit[20]=1
TSTNE   R1,R5,ASR R1          ;条件执行,R5 内容算术右移,移位次数在 R1 中最低
                              ;字节,移位结果和 R1 与,设置标志位
```

5）使用移位操作的指令举例

```
MOV     R0,R1,LSL #2          ;R1 内容逻辑左移 2 位送 R0
MOV     R0,R1,LSR #2          ;R1 内容逻辑右移 2 位送 R0
ADD     R2,R0,R1,ASR #2       ;R1 内容算术右移 2 位,加 R0,和送 R2
SUB     R0,R2,R0,ROR #4       ;R2 减 R0 内容循环右移 4 位的值,差送 R0
MOV     R0,R1,RRX             ;R1 内容扩展循环右移,送 R0
```

```
MOV     R1,R2,ROR R3          ;R2 内容循环右移,移位次数在 R3 低字节中,移位结果送 R1
MOV     R0,#0xf000000f        ;汇编后产生指令 MOV R0,#0xff,4
                              ;表示对立即数 0xff 循环右移 4 位,对应图 3-2
                              ;指令的 bit[11:8]为 2,bit[7:0]为 0xff
```

6) 程序举例

【例 3.4】 如果 R0=1 或者 R1=2,则程序分支到标号为 Label0 处;否则,执行标号为 Label1 处的代码。

```
        CMP    R0,#1          ;比较 R0 是否等于 1,设置标志位
        BEQ    Label0         ;相等,分支到 Label0 处
        CMP    R1,#2          ;比较 R1 是否等于 2,设置标志位
        BEQ    Label0         ;相等,分支到 Label0 处
Label1                        ;R0<>1 同时 R1<>2
        ⋮
Label0                        ;由 R0=1 或 R1=2 分支过来
        ⋮

        ;也可以将上面 4 条指令,改成如下 3 条指令,实现相同功能
        CMP    R0,#1          ;比较 R0 是否等于 1,设置标志位
        CMPNE  R1,#2          ;如果前一条指令比较结果不相等(条件码 NE),
                              ;才执行本条指令,比较 R1 是否等于 2,设置标志位;
                              ;如果前一条指令比较相等,则不执行本条指令
        BEQ    Label0         ;Z=1,则分支到 Label0 处
Label1                        ;R0<>1 同时 R1<>2
        ⋮
Label0                        ;由 R0=1 或 R1=2 分支过来
        ⋮
```

【例 3.5】 求 R0 的绝对值,再求 R1 的绝对值,将这两个绝对值相加,和存 R2。求绝对值的方法是:当 Rn>=0 时,Rn 的值不变;否则,将 Rn 的值求补。

```
TEQ    R0,#0                  ;由 R0 EOR 0,设置标志位
RSBMI  R0,R0,#0               ;如果标志位 MI=1,表示 R0<0,则执行本指令,R0=0-R0
TEQ    R1,#0                  ;由 R1 EOR 0,设置标志位
RSBMI  R1,R1,#0               ;如果标志位 MI=1,表示 R1<0,则执行本指令,R1=0-R1
ADD    R2,R1,R0               ;R2=R1+R0
```

【例 3.6】 对于 R1 中的无符号数,判断其值的不同范围,做不同的计算。

方法如下:

如果 R1 低于 9,则 R2=R0*8;

如果 R1 等于 9,则 R2=R0*9;

如果 R1 高于 9,则 R2=R0*10。

```
MOV    R2,R0,LSL #3           ;R2=R0<<3,即 R2=R0*8
CMP    R1,#9                  ;比较 R1 是否等于 9
```

```
ADDCS  R2,R2,R0              ;如果标志位 CS=1,表示 R1 高于、等于 9 成立,
                             ;则执行本条指令,R2=R2+R0,即 R2=R0*9;
                             ;如果 CS=0,表示 R1 低于 9,则不执行本条指令
ADDHI  R2,R2,R0              ;如果 HI=1,表示 R1 高于 9,则执行本条指令,
                             ;R2=R2+R0,即 R2=R0*10
```

【例 3.7】 求 R0*4+R1*5−R2*7 的值,假定它们都是无符号数,运算结果也不会产生进位,结果存 R3 中。

```
MOV    R0,R0,LSL #2          ;R0=R0<<2,即 R0=R0*4
ADD    R1,R1,R1,LSL #2       ;R1=R1+R1<<2,即 R1=R1*5
RSB    R2,R2,R2,LSL #3       ;R2=R2<<3-R2,即 R2=R2*7
ADD    R3,R0,R1              ;R3=R0+R1
SUB    R3,R3,R2              ;R3=R3-R2
```

参考例 3.7,可以分别用一条指令实现以下不同的计算。

例如,求 R0×2^n 的值(n=0,1,2,3,4;2^n=1,2,4,8,16)。

```
MOV    R0,R0,LSL #n          ;R0 左移 0,1,2,3,4 次,
                             ;实现 R0 乘 1,2,4,8,16
```

例如,求 R1×(2^n+1)的值(n=1,2,3,4;2^n+1=3,5,9,17)

```
ADD    R1,R1,R1,LSL #n       ;R1 加 R1 左移 1,2,3,4 次的值,
                             ;实现 R1 乘 3,5,9,17
```

例如,求 R2×(2^n−1)的值(n=2,3,4,5;2^n−1=3,7,15,31)

```
RSB    R2,R2,R2,LSL #n       ;R2 左移 2,3,4,5 次的值减 R2,
                             ;实现 R2 乘 3,7,15,31
```

【例 3.8】 从子程序返回和从异常返回的区别。

从子程序返回时,不会将当前方式的 SPSR 恢复到 CPSR。

另外,在用户(系统)方式时,不存在当前方式的 SPSR。

```
       ⋮
       BL    SUB1            ;将下一条指令的地址送 LR 寄存器,分支到 SUB1 处,
                             ;实现子程序调用
       ⋮
SUB1                         ;SUB1 子程序
       ⋮
       MOV   PC,LR           ;将 LR 送 PC,从子程序返回
```

从异常返回时,除了将保存在 LR 中的返回地址送 PC 外,还应将当前方式的 SPSR 恢复到 CPSR。

```
       ⋮
       MOVS  PC,LR           ;将 LR 送 PC,指令助记符后缀 S 表示将当前方式 SPSR
                             ;送 CPSR,从异常返回
```

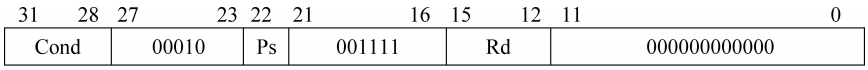
3.2.4 程序状态寄存器传送指令(MRS、MSR)

只有程序状态寄存器传送指令(MRS、MSR),才允许读/写程序状态寄存器 CPSR 或 SPSR_<mode>。

这两条指令配合可以实现对程序状态寄存器的读—修改—写操作,常用于对 FIQ、IRQ 设置允许/禁止;转换处理器的操作方式;也可用于修改条件码标志。

1. 指令编码格式

指令编码格式见图 3-9～图 3-11。



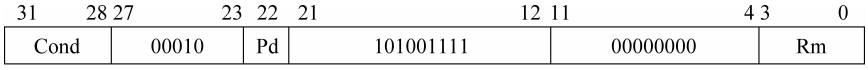
[15:12] 目的寄存器

[22] 源PSR

0=CPSR 1=SPSR_<current mode>

[31:28] 条件域

图 3-9 MRS(传送 PSR 内容到寄存器)指令编码格式



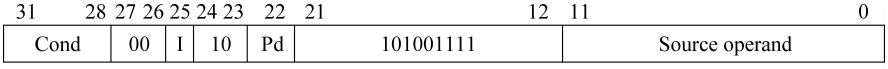
[3:0] 源寄存器

[22] 目的PSR

0=CPSR 1=SPSR_<current mode>

[31:28] 条件域

图 3-10 MSR(传送寄存器内容到 PSR)指令编码格式



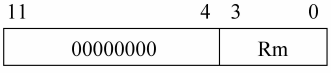
[22] 目的PSR

0=CPSR 1=SPSR_<current mode>

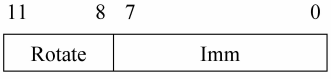
[25] 立即操作数

0=源操作数在寄存器 1=立即数

[11:0] 源操作数



[3:0] 源寄存器



[7:0] 8位无符号立即数

[11:8] 对Imm的移位量

[31:28] 条件域

图 3-11 MSR(传送寄存器内容或立即数到 PSR 标志位)指令编码格式

2. 指令含义

MRS 指令允许将 CPSR 或 SPSR_＜mode＞的内容传送到一个通用寄存器。

MSR 指令允许将一个通用寄存器的内容传送到 CPSR 或 SPSR_＜mode＞寄存器。

MSR 指令也允许将一个立即数或寄存器的内容只传送到 CPSR 或 SPSR_＜mode＞寄存器的条件码标志(N、Z、C 和 V),而不影响其他控制位。在这种情况下,指定寄存器的最高 4 位或立即数的最高 4 位的内容被写入 CPSR 或 SPSR_＜mode＞的最高 4 位(条件码标志)。

指令中的操作数有如下限制。

(1) 在用户方式下,CPSR 的控制位被保护,不能改变,只有条件码标志能被改变。在特权方式,允许改变整个 CPSR。

(2) 程序不要改变 CPSR 的 T 状态位,否则处理器进入未定义状态。

(3) 访问哪一个 SPSR 寄存器取决于当时的执行方式,若处理器在 FIQ 方式,则 SPSR_fiq 被访问。

(4) 不能将 R15 指定为源或目的寄存器。

(5) 在用户方式,不能使用 SPSR 寄存器,因为这种方式不存在这样的寄存器。

(6) CPSR 和 SPSR_＜mode＞寄存器的保留位不要修改。可以采用读出 CPSR 或 SPSR_＜mode＞的内容,只修改它们中需要并且允许修改的位,然后再写回原寄存器。这样可以避免误操作修改了保留位。

例如,以下代码采用读—修改—写的方法,实现了处理器操作方式的改变:

```
MRS    R1,CPSR                    ;CPSR 内容送 R1
BIC    R1,R1,#0x1f                ;修改,清 0 方式位
ORR    R1,R1,#new_mode            ;修改,设置新的方式位
MSR    CPSR,R1                    ;写回
```

3. 指令汇编格式

对于传送 PSR 内容到寄存器的指令 MRS,格式为:

```
MRS{cond} Rd,<psr>
```

对于传送寄存器内容到 PSR 的指令 MSR,格式为:

```
MSR{cond} <psr>,Rm
```

对于只传送寄存器最高 4 位到 PSR 条件码标志的指令 MSR,格式为:

```
MSR{cond} <psrf>,Rm
```

对于只传送立即数到 PSR 的最高 4 位的指令 MSR,格式为:

```
MSR{cond} <psrf>,<#expression>
```

这种格式中＜#expression＞应该是 32 位立即数,它的最高 4 位写入 PSR 的 N、Z、

C 和 V 位。

其中：

{cond}	表示条件码助记符。
Rd 和 Rm	表示除 R15 外的其他通用寄存器编号。
<psr>	指 CPSR,CPSR_all,SPSR 或 SPSR_all。其中 CPSR 和 CPSR_all 是同义词,SPSR 和 SPSR_all 是同义词。
<psrf>	指 CPSR_flg 或 SPSR_flg。
<#expression>	是一个表达式,经过计算后得到 32 位立即数,汇编器试图产生一个 8 位立即数,经过循环、右移后能够产生一个 32 位数,与表达式计算后得到的 32 位数相等;如果不能实现,则给出错误。

例如指令：

```
MSR CPSR_flg,#0x50000000
```

由于指令长度为 32 位,所以不能直接使用 32 位立即数。汇编器试图产生一个 8 位立即数,填入指令的 bit[7:0],产生一个循环右移移位量填入指令 bit[11:8],在指令执行期间通过移位(移位方法参考数据处理指令)得到一个 32 位数,与汇编指令中立即数匹配。汇编器如果无法实现数据匹配,则给出错误。本条指令只使用了 32 位立即数的最高 4 位。

4. 使用举例

在用户方式和特权方式,某些相同格式的指令,产生的作用是不相同的。

```
;在用户方式
MSR  CPSR,R0                        ;R0[31:28]送 CPSR[31:28]
MSR  CPSR_flg,R0                    ;R0[31:28]送 CPSR[31:28]
MSR  CPSR_flg,#0xf0000000           ;0xf 送 CPSR[31:28]
MRS  R0,CPSR                        ;CPSR[31:0]送 R0[31:0]
;在特权方式
MSR  CPSR,R0                        ;R0[31:0]送 CPSR[31:0]
MSR  CPSR_flg,R0                    ;R0[31:28]送 CPSR[31:28]
MSR  CPSR_flg,#0xf0000000           ;0xf 送 CPSR[31:28]
MSR  SPSR,R0                        ;R0[31:0]送 SPSR_< mode> [31:0]
MSR  SPSR_flg,R0                    ;R0[31:28]送 SPSR_< mode> [31:28]
MSR  SPSR_flg,#0x30000000           ;0x3 送 SPSR_flg[31:28]
MRS  R1,SPSR                        ;SPSR_< mode> [31:0]送 R1[31:0]
```

【例 3.9】 允许 FIQ 中断,禁止 FIQ 中断(特权方式)。

```
ENABLE_FIQ                          ;允许 FIQ 中断
       MRS    R0,CPSR               ;CPSR[31:0]送 R0[31:0]
       BIC    R0,R0,#0x40           ;R0[6]清 0,其余位不变,允许 FIQ 中断
       MSR    CPSR,R0               ;R0[31:0]送 CPSR[31:0]
```