华清远见 FARSIGHT 始于2004 | 嵌入式培训专家

10年口碑积累，成功培养50000多名研发工程师，铸就专业品牌形象

华清远见的企业理念是不仅要做良心教育、做专业教育，更要做受人尊敬的职业教育。

《ARM 系列处理器应用技术完全手册》

作者：华清远见

专业始于专注 卓识源于远见

第 4 章 ARM 指令寻址方式

本章目标

ARM 指令集可以分为跳转指令、数据处理指令、程序状态寄存器传输指令、Load/Store 指令、协处理器指令和异常中断产生指令。根据适用的指令类型不同，指令的寻址方式分为：数据处理指令操作数寻址方式和内存访问指令寻址方式。

专业始于专注 卓识源于远见

4.1 数据处理指令的寻址方式

4.1.1 数据处理指令的寻址方式概要

数据处理指令的基本语法格式如下。

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

其中<shifter_operand>有下面 11 种形式，如表 4.1 所示。

表 4.1 <shifter_operand>的寻址方式

	语 法	寻 址 方 式
1	#<immediate>	立即数寻址
2	<Rm>	寄存器寻址
3	<Rm>, LSL #<shift_imm>	立即数逻辑左移
4	<Rm>, LSL <Rs>	寄存器逻辑左移
5	<Rm>, LSR #<shift_imm>	立即数逻辑右移
6	<Rm>, LSR <Rs>	寄存器逻辑右移
7	<Rm>, ASR #<shift_imm>	立即数算术右移
8	<Rm>, ASR <Rs>	寄存器算术右移
9	<Rm>, ROR #<shift_imm>	立即数循环右移
10	<Rm>, ROR <Rs>	寄存器循环右移
11	<Rm>, RRX	寄存器扩展循环右移

数据处理指令的寻址方式根据<shifter_operand>的不同，相应的分为 11 种。

4.1.2 指令解码

图 4.1 显示了数据处理指令不同寻址方式下的解码格式。

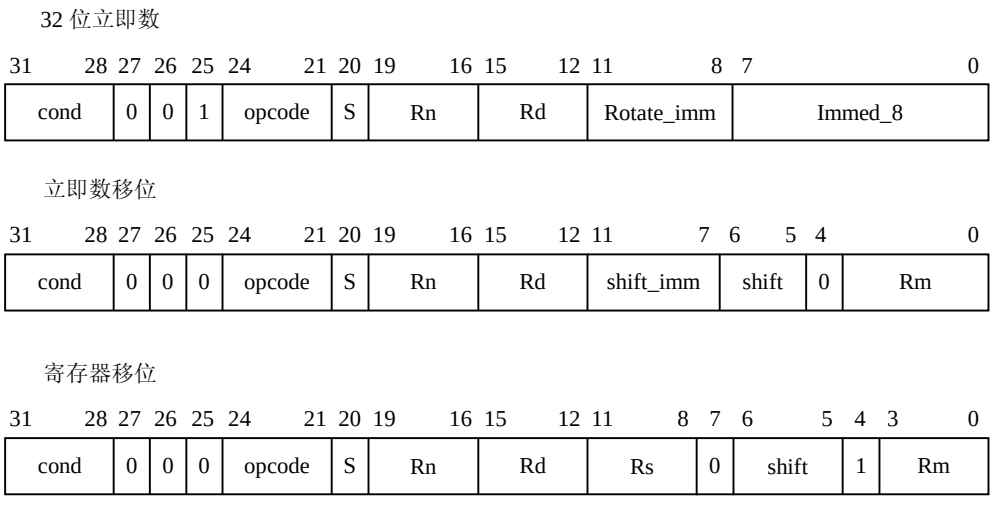


图 4.1 数据操作指令编码格式

编码格式中各域含义如下。

- <opcode>：确定具体指令。

- S：标识指令是否影响程序状态寄存器 CPSR 条件标志。
- Rd：指令操作的目的寄存器。
- Rn：指令第一源操作数。
- bit[11：0]：移位操作，详见本章移位操作一节。
- bit[25]：被用来区分是立即数移位操作还是寄存器移位操作。

如果指令编码出现下面情况：bit[25] = 0 并且 bit[4] = 1 并且 bit[7] = 1，则指令并非数据处理指令，它可能是 Load/Store 指令或算术指令。

4.1.3 移位操作

数据处理指令是在算术逻辑单元 ALU 中完成。ARM 处理器一个显著特征就是可以在操作数进入 ALU 之前，对操作数进行指定位数的左移或右移操作。这种功能明显增强了数据处理操作的灵活性。
移位操作可能产生进位，更新程序状态寄存器 CPSR 的进位标志 C。移位操作有下面 3 种基本方式。

1．立即数方式

没有任何一条 ARM 指令可以包含一个 32 位的立即数，数据处理指令编码格式中，第二个操作数有 12 位。指令的编码格式如图 4.1 所示。
指令中的立即数是由一个 8 bit 的常数移动 4 bit 偶数位（0，2，4，…，26，28，30）得到的。所以，每一条指令都包含一个 8 bit 的常数 X 和移位值 Y，得到的立即数=X 循环右移（2×Y）。

注意 8 位立即数一定要移偶数位。

下面列举了一些有效的立即数。
0xFF、0x104、0xFF0、0x FF00、0x FF000、0x FF000000、0x F000000F
下面是一些无效的立即数。
0x101、0x102、0x FF1、0x FF04、0x FF003、0x FFFFFFFF、0x F000001F
下面是一些应用立即数的指令。

```
MOV  r0，＃0                ;送 0 到 r0
ADD  r3，r3，＃1            ;r3 的值加 1
CMP  r7，＃1000             ;r7 的值和 1000 比较
BIC  r9，r8，＃0x FF00      ;将 r8 中 8～15 位清零，结果保存在 r9 中
```

2．寄存器方式

寄存器的值可以被直接用于数据操作指令，如：

```
MOV  r2，r0                 ;r0 的值送 r2
ADD  r4，r3，r2             ;r2 加 r3，结果送 r4
CMP  r7，r8                 ;比较 r7 和 r8 的值
```

3．寄存器移位方式

寄存器的值在被送到 ALU 之前，可以事先经过桶形移位寄存器的处理。预处理和移位发生在同一周期内，所以有效的使用移位寄存器，可以增加代码的执行效率。

具体的移位（或者循环移位）方式有下面几种。

- ASR：算术右移。
- LSL：逻辑左移。
- LSR：逻辑右移。
- ROR：循环右移。
- RRX：扩展的循环右移。

以上 5 种移位方式，移位值均可以由立即数或寄存器指定。下面是一些在指令中使用了移位操作的例子。

```
ADD  r2,r0,r1,LSR  #5
MOV  r1,r0,LSL  #2
RSB  r9,r5,r5,LSL  #1
SUB  r1,r2,r0,LSR #4
MOV  r2,r4,ROR  r0
```

4.1.4 寻址方式分类详解

数据处理指令的寻址方式根据<shifter_operand>的不同，相应的分为 11 种。详见表 4.1。下面对各类寻址方式进行详细说明。

1. #<immediate>

（1）编码格式

指令的编码格式如图 4.2 所示。

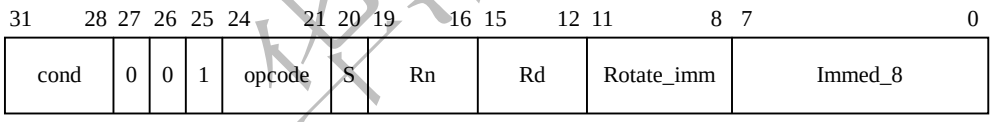


图 4.2 数据处理指令——立即数寻址编码格式

立即数寻址为数据处理指令提供了一个可直接操作的立即数。立即数的生成方法见前面章节介绍。如果移位值为 0，则移位进位值为程序状态寄存器 CPSR 的 C 标志位；否则，为 32-bit 立即数的 bit[31]。

（2）操作伪代码

```
Shifter_operand = immed_8 Rotate_Right (rotate_imm*2)
if  rotate_imm == 0 then
     shifter_carry_out = C flag
else  /* rotate_imm != 0*/
     shifter_carry_out = shifter_operand[31]
```

（3）说明

① 并不是所有的 32-bit 立即数都是可以使用的合法立即数。只有那些通过将一个 8-bit 的立即数循环右移偶数位可以得到的立即数才可以在指令中使用。

② 有些立即数可以通过不止一种方法得到。由于立即数的构造方法中移位包含了循环操作，而循环移位操作会影响 CPSR 的条件标志位 C。因此，同一个合法的立即数由于采用了不同的编码方式，将使这些指令的执行产生不同的结果，这是不能允许的。ARM 汇编器按照下面的规则来生成立即数的编码。

- 当立即数数值在 0 和 0xFF 范围时，令 immed_8=<immediate>，immed_4=0。

华清远见

- 其他情况下，汇编编译器选择使用 immed_4 数值最小的编码方式。

③ 为了更精确地控制立即数的生成，可以使用下面的语法格式控制立即数的生成。

```
#<immed_8>,<rotate_amout>
```

其中，<rotate_amout> = 2*rotate_imm

（4）举例

```
SUBS  r0,r0,#1                    ;寄存器 r0 中的数值减 1，结果保存到 r0
MOV  r0,#0xff00    ; 0xff00 → r0  ;将立即数 0xff00 放入 r0 保存
```

2．<Rm>

（1）编码格式

指令的编码格式如图 4.3 所示。

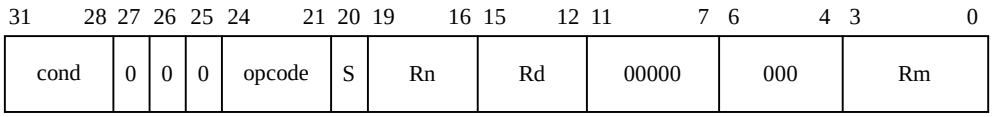


图 4.3 数据处理指令——寄存器寻址编码格式

指令的操作数即为寄存器中的数值。移位寄存器的进位为程序状态寄存器 CPSR 的 C 标志位。

指令的语法格式为：<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>

（2）操作伪代码

```
Shifter_operand = Rm
Shifter_carry_out = C Flag
```

（3）说明

① 从指令的解码格式来看，寄存器寻址方式和使用立即数逻辑左移寻址解码格式是相同的，只是其移位数为 0。

② 如果指令中的 Rm 或 Rn 指定为程序计数器 r15，则操作数的值为当前指令地址加 8。

（4）举例

```
MOV  r1,r2    ; r2 → r1
SUB  r0,r1,r2   ; r1 – r2 → r0
```

3．<Rm>, LSL #<shift_imm>

（1）编码格式

指令的编码格式如图 4.4 所示。

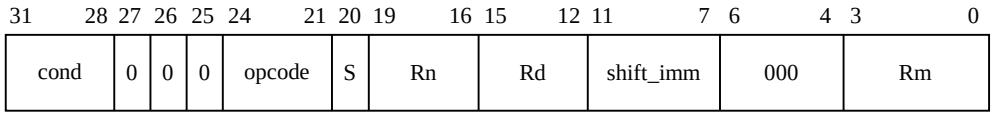


图 4.4 数据处理指令——立即数逻辑左移寻址编码格式

指令的操作数为寄存器 Rm 的数值逻辑左移 shift_imm 位。左移的范围在 0 到 31 之间。左移移出的位用 0 补齐。进位标志位是最后移出的位（如果移位数为 0，则为 C 标志位）。

指令的语法格式为：<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,LSL #<shift_imm>，其中：

- <Rm>为进行逻辑左移操作的寄存器；

- LSL 为逻辑左移操作标识；
- <shift_imm>为逻辑左移位数，范围为 0～31。

（2）操作伪代码

```
if  shift_imm == 0 then /*执行寄存器操作*/
     shifter_operand = Rm
     shifter_carry_out = C flag
else  /*移位寄存器大于零*/
     shifter_operand = Rm logical_shift_left shift_imm
     shifter_carry_out = Rm[32 – shift_imm]
```

（3）说明

① 如果移位立即数<shift_imm> =0，则该寻址方式为立即数直接寻址。

② 如果指令中的 Rm 或 Rn 指定为程序计数器 r15，则操作数的值为当前指令地址加 8。

（4）举例

```
SUB r0，r1，r2，LSL #10      ;r1 的值减去 r2 的值左移 10bit，结果放到 r0 寄存器
MOV r0，r2，LSL #3           ;r2 的值左移 3bit，结果放入 r0，即 r0 = r2×8
```

4．<Rm>, LSL　<Rs>

（1）编码格式

指令的编码格式如图 4.5 所示。

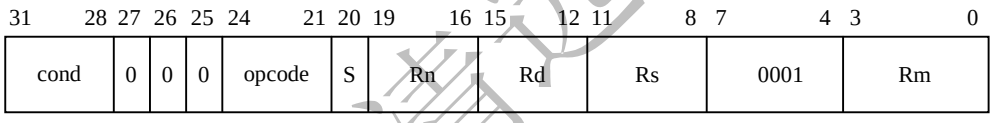


图 4.5　数据处理指令——寄存器逻辑左移寻址编码格式

寄存器逻辑左移十分适合寄存器值乘 2 的倍数操作。

这个指令是将寄存器 Rm 的值逻辑左移一定的位数。位移的位数由 Rs 的最低 8 位 bit[7：0]决定。Rm 移出的位用 0 补齐。进位值是移位寄存器最后移出的位，如果移位数大于 0，则进位值为 0。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,LSL  <Rs>
```

其中：

- <Rm>为指令被移位的寄存器；
- LSL 为逻辑左移操作标识；
- <Rs>为包含逻辑左移位数的寄存器。

（3）操作伪代码

```
if  Rs[7:0] = = 0 then
    shifter_operand = Rm
    shifter_carry_out = C flag
else  if  Rs[7:0] < 32 then
    shifter_operand = Rm  logical_shift_left  Rs[7:0]
    shifter_carry_out = Rm[32 – Rs[7:0]]
else  if  Rs[7:0] = = 32 then
    shifter_operand = 0
    shifter_carry_out = Rm[0]
```

华清远见

```
else  /*Rs 的后 8 位大于零*/
    shifter_operand = 0
    shifter_carry_out = 0
```

（4）说明

如果程序计数器 r15 被用作 Rd，Rm，Rn 或 Rs 中的任意一个，则指令的执行结果不可预知。

（5）举例

```
MOV  r0，r2，LSL r3        ;r2 的值左移 r3 位，结果放入 r0
ANDS r1，r1，r2，LSL r3    ;r2 的值左移 r3 位，然后和 r1 相与，结果放入 r1
```

5．<Rm>, LSR ＃<shift_imm>

（1）编码格式

指令的编码格式如图 4.6 所示。

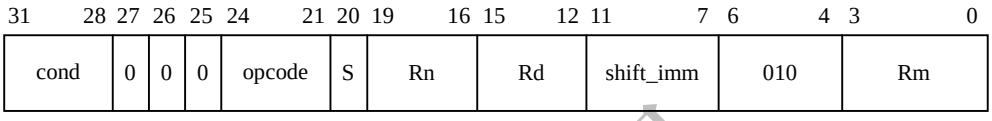


图 4.6　数据处理指令——立即数逻辑右移寻址编码格式

指令的操作数为寄存器 Rm 的值右移<shift_imm>位，相当于 Rm 的值除以一个 2 的倍数。<shift_imm>值的范围为 0～31，移位后空出的位添 0。循环器进位值为 Rm 最后移出的位。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,LSR ＃<shift_imm>
```

其中：

- <Rm>为被移位的寄存器；
- LSR 为逻辑右移操作标识；
- <shift_imm>为逻辑右移位数，范围为 0～31。

（3）操作伪代码

```
if  shift_imm == 0 then /*执行寄存器操作*/
     shifter_operand = 0
     shifter_carry_out = Rm[31]
else  /*移位立即数大于零*/
     shifter_operand = Rm logical_shift_Right shift_imm
     shifter_carry_out = Rm[shift_imm - 1]
```

（4）说明

① shift_imm 的取值范围为 0～31，当 shift_imm=0 时，移位位数为 32，所以移位位数范围为 1～32 位。

② 如果指令中的 Rm 或 Rn 指定为程序计数器 r15，则操作数的值为当前指令地址加 8。

6．<Rm>, LSR　<Rs>

（1）编码格式

指令的编码格式如图 4.7 所示。

华清远见

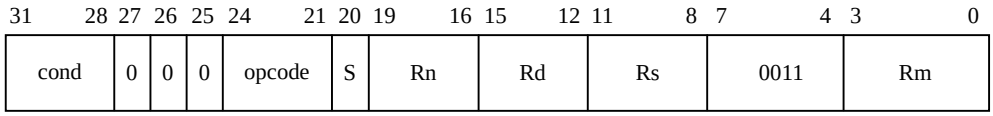


图 4.7　数据处理指令——寄存器逻辑右移寻址编码格式

此操作将寄存器 Rm 的数值逻辑右移一定的位数。移位的位数由 Rs 的最低 8 位 bit[7：0]决定。移出的位由 0 补齐。当 Rs[7：0]大于 0 而小于 32 时，进位标志 C 由最后移出的位决定，当 Rs[7：0]大于 32 时，进位标志位为 0，当 Rs[7：0]等于 0 时，进位标志不变。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,LSR  <Rs>
```

其中：

- <Rm>为指令被移位的寄存器；
- LSR 为逻辑右移操作标识；
- <Rs>为包含逻辑右移位数的寄存器。

（3）操作伪代码

```
if  Rs[7:0] = = 0 then
    shifter_operand = Rm
    shifter_carry_out = C flag
else  if  Rs[7:0] < 32 then
    shifter_operand = Rm  logical_shift_Right  Rs[7:0]
    shifter_carry_out = Rm[Rs[7:0] - 1]
else  if  Rs[7:0] = = 32 then
    shifter_operand = 0
    shifter_carry_out = Rm[31]
else  /*Rs 的后 8 位大于零*/
    shifter_operand = 0
    shifter_carry_out = 0
```

（4）说明

如果程序计数器 r15 被用作 Rd、Rm、Rn 或 Rs 中的任意一个，则指令的执行结果不可预知。

7．<Rm>, ASR ＃<shift_imm>

（1）编码格式

指令的编码格式如图 4.8 所示。

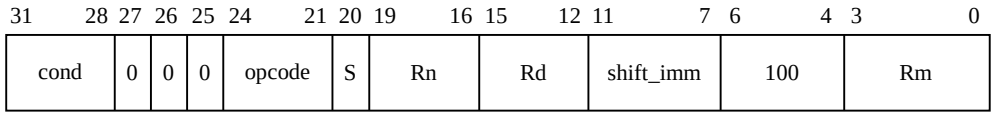


图 4.8　数据处理指令——立即数算术右移寻址编码格式

指令的操作数为寄存器 Rm 的数值逻辑右移<shift_imm>位。<shift_imm>的值范围为 0～31，当<shift_imm>等于 0 时，移位位数为 32，所以移位位数范围为 1～32 位。进位移位操作后，空出的位添 Rm 的最高位 Rm[31]。进位标志为 Rm 最后被移出的数值。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,ASR ＃<shift_imm>
```

其中：

- <Rm>为被移位的寄存器；
- ASR 为算术右移操作标识；
- <shift_imm>为算术右移位数，范围为 1～32，当 shift_imm 等于 0 时移位位数为 32。

（3）操作伪代码

```
if  shift_imm == 0 then /*执行寄存器操作*/
    if  Rm[31] = = 0 then
        shifter_operand = 0
        shifter_carry_out = Rm[31]
    else /*Rm[31] = = 1*/
        shifter_operand = 0xffffffff
        shifter_carry_out = Rm[31]
else  /*shift_imm > 0*/
    shifter_operand = Rm Arithmetic_shift_Right <shift_imm>
    shifter_carry_out = Rm[shift_imm - 1]
```

（4）说明

① 如果指令中的 Rm 或 Rn 指定为程序计数器 r15，则操作数的值为当前指令地址加 8。

8．<Rm>, ASR　<Rs>

（1）编码格式

指令的编码格式如图 4.9 所示。

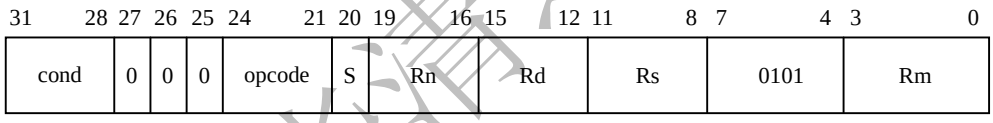


图 4.9　数据处理指令——寄存器算术右移寻址编码格式

此操作将寄存器 Rm 的数值算术右移一定的位数。移位后空缺的位由 Rm 的符号位（Rm[31]）填充。位移的位数由 Rs 的最低 8 位 bit[7：0]决定。当 Rs[7：0]大于零而小于 32 时，指令的操作数为寄存器 Rm 的数值算术右移 Rs[7：0]位，进位标志 C 为 Rm 最后被移出的位。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,ASR  <Rs>
```

其中：

- <Rm>为指令被移位的寄存器；
- ASR 为算术右移操作标识；
- <Rs>为包含算术右移位数的寄存器。

（3）操作伪代码

```
if  Rs[7:0] = = 0 then
    shifter_operand = Rm
    shifter_carry_out = C flag
else  if  Rs[7:0] < 32 then
    shifter_operand = Rm  Arithmeticl_shift_Right  Rs[7:0]
    shifter_carry_out = Rm[Rs[7:0] - 1]
else
    if  Rm[31] = =0 then
```

```
            shifter_operand = 0
            shifter_carry_out = Rm[31]
       else
            shifter_operand = 0xffffffff
            shifter_carry_out = Rm[31]
```

（4）说明

如果程序计数器 r15 被用作 Rd、Rm、Rn 或 Rs 中的任意一个，则指令的执行结果不可预知。

9．<Rm>, ROR ＃<shift_imm>

（1）编码格式

指令的编码格式如图 4.10 所示。

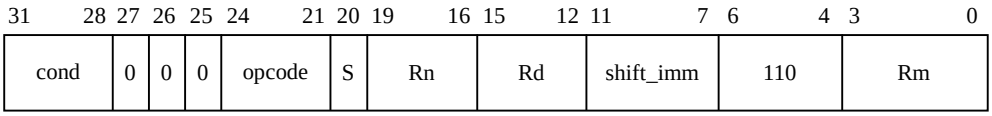


图 4.10　数据处理指令——立即数循环右移寻址编码格式

指令的操作数由寄存器 Rm 的数值循环右移一定的位数得到。移位的位数由 Rs 的最低 8 位 bits[7：0]决定。当 Rs[7：0]=0 时，指令的操作数为寄存器 Rm 的值，循环器的进位值为 CPSR 中的 C 条件标志位；否则，循环器的进位值为 Rm 最后被移出的位。

（2）语法格式

```
    <opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,ROR ＃<shift_imm>
```

其中：

- <Rm>为被移位的寄存器；
- ROR 为循环右移操作标识；
- <shift_imm>为循环右移位数，范围为 1～31，当 shift_imm 等于 0 时执行 RRX 操作。

（3）操作伪代码

```
    if  shift_imm == 0 then /*执行寄存器操作*/
        执行 RRX 操作
    else
        shifter_operand = Rm Rotate_Right shift_imm
        shifter_carry_out = Rm[shift_imm - 1]
```

（4）说明

如果指令中的 Rm 或 Rn 指定为程序计数器 r15，则操作数的值为当前指令地址加 8。

10．<Rm>, ROR　<Rs>

（1）编码格式

指令的编码格式如图 4.11 所示。

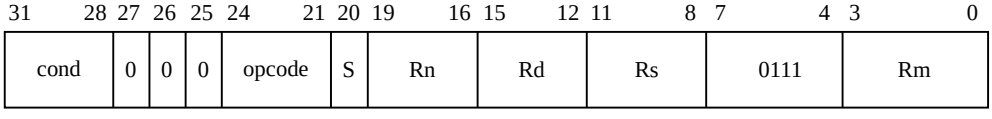


图 4.11　数据处理指令——寄存器循环右移寻址编码格式

指令的操作数由寄存器 Rm 的数值循环右移一定的位数。移位的位数由 Rs 的最低 8 位 bits[7：0]决定。当 Rs[7：0]=0 时，指令的操作数为寄存器 Rm 的值，循环器的进位值为 CPSR 中的 C 条件标志位；否则，循环器的进位值为 Rm 最后被移出的位。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,ROR  <Rs>
```

其中：

- <Rm>为指令被移位的寄存器；
- ROR 为循环右移操作标识；
- <Rs>为包含循环右移位数的寄存器。

（3）操作伪代码

```
if  Rs[7:0] = = 0 then
    shifter_operand = Rm
    shifter_carry_out = C flag
else  if  Rs[4:0] == 0 then
    shifter_operand = Rm
    shifter_carry_out = Rm[31]
else
    shifter_operand = Rm Rotate_Right Rs[4:0]
    shifter_carry_out = Rm[Rs[4:0] - 1]
```

（4）说明

如果程序计数器 r15 被用作 Rd、Rm、Rn 或 Rs 中的任意一个，则指令的执行结果不可预知。

11．<Rm>, RRX

（1）编码格式

指令的编码格式如图 4.12 所示。

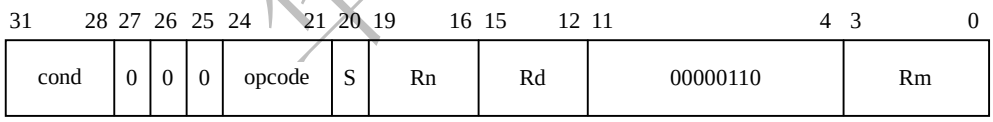


图 4.12　数据处理指令——扩展右移寻址编码格式

指令的操作数为寄存器 Rm 的数值右移一位，并用 CPSR 中的 C 条件标志位填补空出的位。CPSR 中的 C 条件标志位则用移出的位代替。

（2）语法格式

```
<opcode> {<cond>} {S} <Rd>,<Rn>,<Rm>,RRX
```

其中：

- <Rm>为指令被移位的寄存器；
- RRX 为扩展的循环右移操作。

（3）操作伪代码

```
shifter_operand = (C flag logical_shift_left 31) OR (Rm logical_shift_Right 1)
shifter_carry_out = Rm[0]
```

（4）说明

① 此种寻址方式的编码形式和“ROR ＃0”一致。

② 如果程序计数器 r15 被用作 Rd、Rm、Rn 或 Rs 中的任意一个，则指令的执行结果不可预知。

③ 可以实现 ADC 指令的功能。

4.2 内存访问指令寻址

根据内存访问指令的分类，内存访问指令的寻址方式可以分为以下几种。

① 字及无符号字节的 Load/Store 指令的寻址方式。
② 杂类 Load/Store 指令的寻址方式。
③ 批量 Load/Store 指令的寻址方式。
④ 协处理器 Load/Store 指令的寻址方式。

4.2.1 字及无符号字节的 Load/Store 指令的寻址方式

字及无符号字节的 Load/Store 指令语法格式如下：

```
LDR|STR{<cond>}{B}{T}  <Rd>，<addressing_mode>
```

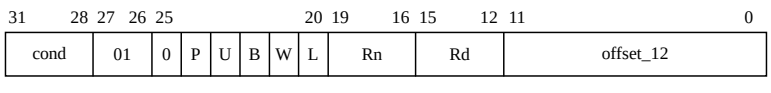
其中<addressing_mode>共有 9 种寻址方式，如表 4.2 所示。

表 4.2　　字及无符合字节的 Load/Store 指令的寻址方式

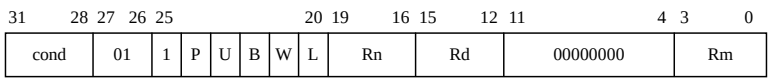
	格　　式	模　　式
1	[Rn，#±<offset_12>]	立即数偏移寻址 （Immediate offset）
2	[Rn，±Rm]	寄存器偏移寻址 （Register offset）
3	[Rn，Rm，<shift>#< offset_12>]	带移位的寄存器偏移寻址 （Scaled register offset）
4	[Rn，#±< offset_12>]!	立即数前索引寻址 （Immediate pre-indexed）
5	[Rn，±Rm]!	寄存器前索引寻址 （Register post-indexed）
6	[Rn，Rm，<shift>#< offset_12>]!	带移位的寄存器前索引寻址 (Scaled register pre-indexed)
7	[Rn]，#±< offset_12>	立即数后索引寻址 (Immediate post-indeded)
8	[Rn]，±<Rm>	寄存器后索引寻址 (Register post-indexed)
9	[Rn]，±<Rm>，<shift>#< offset_12>	带移位的寄存器后索引寻址 (Scaled register post-indexed)

字及无符号字节的 Load/Store 指令的解码格式如图 4.13 所示。

立即数



寄存数



寄存器偏移

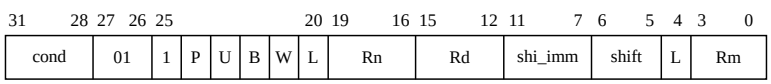


图 4.13　字及无符号字节的 Load/Store 指令的解码格式

编码格式中各位的含义如表 4.3 所示。

表 4.3　　字和无符号半字 Load/Store 指令编码格式各位含义

位　标　识	取　　值	含　　义
P	P=0	使用后索引寻址
	P=1	使用偏移地址或前索引寻址（由 W 位决定）
U	U=0	访问的地址=基址寄存器的值−偏移量（offset）
	U=1	访问的地址=基址寄存器的值+偏移量（offset）
B	B=0	字访问 Load/Store
	B=1	无符号字节访问 Load/Store
W	W=0	如果 P=0，该指令为 LDR、LDRB、STR 或 STRB 指令，且内存访问指令为正常访问指令；如果 P=1，指令执行不更新基地址
	W=1	如果 P=0，该指令为 LDRBT、LDRT、STRBT 或 STRT，且指令为非特权（用户模式）访问指令；如果 P=1，计算内存地址并更新基地址
L	L=0	Store 指令
	L=1	Load 指令

1．[Rn，#±<offset_12>]

（1）编码格式

指令的编码格式如图 4.14 所示。

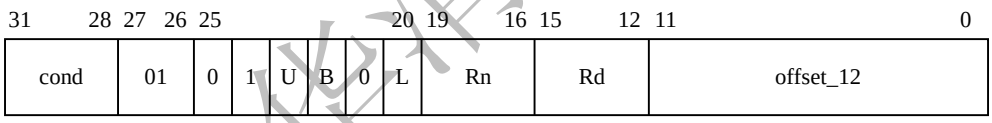


图 4.14　内存访问指令——立即数偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）立即数 offset_12。

编程中，在访问结构体或记录（record）类型的变量时，这些内存的操作指令是十分有效的。另外，在子程序中也常用这些指令访问本地变量和堆栈。

（2）语法格式

```
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，#±<offset_12>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <offset_12>为 12 位立即数，内存访问地址偏移量。

（3）操作伪代码

```
If  U = = 1  then
    Address = Rn + offset_12
Else
    Address = Rn – offset_12
```

（4）说明

① 如果指令中没有指定立即数，使用[<Rn>]，编译器按[<Rn>，＃0]形式编码。

② 如果 Rn 被指定为程序计数器 r15，其值为当前指令地址加 8。

华清远见

2．[Rn，±Rm]

（1）编码格式

指令的编码格式如图 4.15 所示。

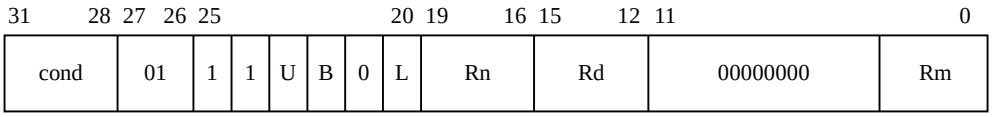


图 4.15　内存访问指令——寄存器偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）偏移寄存器 Rm 的值。

该寻址方式适合使用指针访问字节数组中的数据成员。

（2）语法格式

```
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量。

（3）操作伪代码

```
If  U = = 1  then
    Address = Rn + Rm
Else
    Address = Rn – Rm
```

（4）说明

如果 Rn 被指定为程序计数器 r15，其值为当前指令地址加 8；如果 r15 被用作偏移地址寄存器 Rm 的值，指令的执行结果不可预知。

3．[Rn，Rm，<shift>#< offset_12>]

（1）编码格式

指令的编码格式如图 4.16 所示。

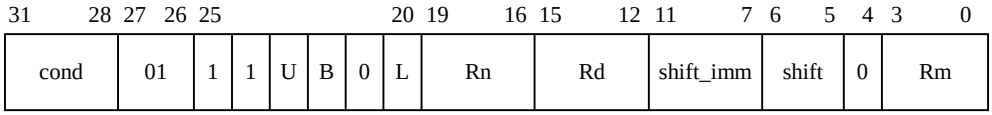


图 4.16　内存访问指令——带移位的寄存器偏移寻址编码格式

内存地址为 Rn 的值加/减通过移位操作后的 Rm 的值。

当数组中的成员长度大于 1 个字节时，使用该寻址方式可高效率地访问数组成员。

（2）语法格式

语法格式有以下 5 种。

```
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，LSL #< offset_12>]
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，LSR #< offset_12>]
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，ASR #< offset_12>]
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，ROR #< offset_12>]
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，RRX]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量；
- LSL 表示逻辑左移操作；
- LSR 表示逻辑右移操作；
- ASR 表示算术右移操作；
- ROR 表示循环右移操作；
- RRX 表示扩展的循环右移。
- <shift_imm>为移位立即数。

（3）操作伪代码

```
Case  shift  of
    0b00  /*LSL*/
        Index = Rm logic_shift_left  shift_imm
    0b01  /*LSR*/
        If  shift_imm = = 0  then /*LSR  #32*/
            Index = 0
        Else
            Index = Rm logical_shift_right  shift_imm
    0b10  /*ASR*/
        If  shift_imm = = 0 then /*ASR  #32*/
            If  Rm[31] = = 1 then
                Index = 0xffffffff
            Else
                Index = 0
        Else
            Index = Rm  Arithmetic_shift_Right  shift_imm
    0b11  /* ROR or RRX*/
        If  shift_imm = = 0 then /*RRX*/
            Index = (C flag Logical_shift_left 31) OR
                    (Rm logical_shift_Right 1)
        Else /*ROR*/
            Index = Rm Rotate_Right shift_imm
Endcase
If  U = = 1 then
    Address = Rn + index
Else /*U = = 0*/
    Address = Rn – index
```

（4）说明

如果 Rn 被指定为程序计数器 r15，其值为当前指令地址加 8；如果 r15 被用作偏移地址寄存器 Rm 的值，指令的执行结果不可预知。

4．[Rn，#±< offset_12>]!

（1）编码格式

指令的编码格式如图 4.17 所示。

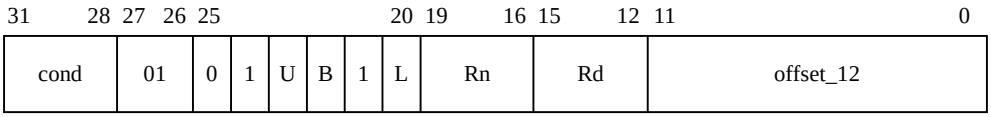


图 4.17　内存访问指令——前索引立即数偏移寻址编码格式

内存地址为基址寄存器 Rn 加/减立即数 offset_8 的值。当指令执行的条件<cc>满足时，生成的地址写回基址寄存器 Rn 中。

该寻址方式适合访问数组自动进行数组下标的更新。

（2）语法格式

```
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<offset_12>]！
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <offset_12>为 12 位立即数，内存访问地址偏移量；
- ！设置指令编码中的 W 位，更新指令基址寄存器。

（3）操作伪代码

```
If  U == 1  then
    Address = Rn + offset_12
Else
    Address = Rn – offset_12
If  ConditionPassed{cond}  then
    Rn = address
```

（4）说明

① 如果指令中没有指定立即数，使用[<Rn>]，编译器按[<Rn>，＃0]！形式编码。

② 如果 Rn 被指定为程序计数器 r15，指令的执行结果不可预知。

5．[Rn，±Rm]！

（1）编码格式

指令的编码格式如图 4.18 所示。

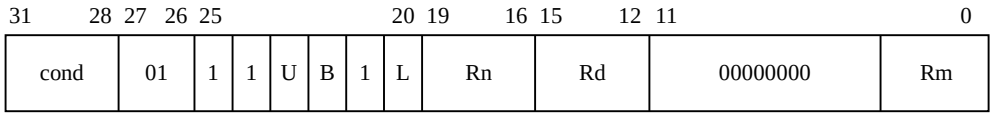


图 4.18　内存访问指令——前索引寄存器偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）偏移寄存器 Rm 的值。当指令的执行条件<cc>满足时，生成地地址将写回基址寄存器。

（2）语法格式

```
LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量；
- ！设置指令编码中的 W 位，更新指令基址寄存器。

（3）操作伪代码

```
If  U = = 1  then
    Address = Rn + Rm
```

```
    Else
        Address = Rn – Rm
    If  ConditionPassed{cond}  then
        Rn = address
```

（4）说明

如果 Rn 和 Rm 指定为同一寄存器，指令的执行结果不可预知。

6．[Rn，±Rm，<shift>#< offset_12>]！

（1）编码格式

指令的编码格式如图 4.19 所示。

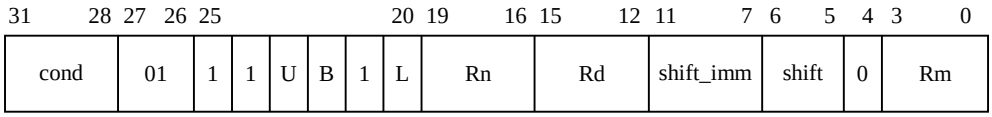


图 4.19　内存访问指令——带移位的前索引寄存器偏移寻址编码格式

内存地址为 Rn 的值加/减通过移位操作后的 Rm 的值。当指令的执行条件<cc>满足时，生成地地址将写回基址寄存器。

（2）语法格式

语法格式有以下 5 种。

```
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，LSL #< offset_12>] !
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，LSR #< offset_12>] !
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，ASR #< offset_12>] !
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，ROR #< offset_12>] !
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>，±<Rm>，RRX] !
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量；
- LSL 表示逻辑左移操作；
- LSR 表示逻辑右移操作；
- ASR 表示算术右移操作；
- ROR 表示循环右移操作；
- RRX 表示扩展的循环右移。
- <shift_imm>为移位立即数。
- ！设置指令编码中的 W 位，更新指令基址寄存器。

（3）操作伪代码

```
    Case  shift  of
        0b00  /*LSL*/
            Index = Rm logic_shift_left  shift_imm
        0b01  /*LSR*/
            If  shift_imm = = 0  then /*LSR  #32*/
                Index = 0
            Else
                Index = Rm logical_shift_right  shift_imm
        0b10  /*ASR*/
            If  shift_imm = = 0 then /*ASR  #32*/
```

```
                If  Rm[31] = = 1 then
                    Index = 0xffffffff
                Else
                    Index = 0
            Else
                Index = Rm  Arithmetic_shift_Right  shift_imm
        0b11  /* ROR or RRX*/
            If  shift_imm = = 0 then /*RRX*/
                Index = (C flag Logical_shift_left 31) OR
                        (Rm logical_shift_Right 1)
            Else /*ROR*/
                Index = Rm Rotate_Right shift_imm
    Endcase
    If  U = = 1 then
        Address = Rn + index
    Else /*U = = 0*/
        Address = Rn – index
    If  ConditionPassed{cond}  then
        Rn = address
```

（4）说明

① 当 PC 用作基址寄存器 Rn 或 Rm 时，指令执行结果不可预知。

② 当 Rn 和 Rm 是同一个寄存器时，指令的执行结果不可预知。

7．[Rn]，#±< offset_12>

（1）编码格式

指令的编码格式如图 4.20 所示。

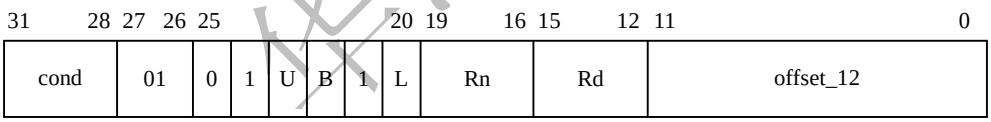


图 4.20　内存访问指令——后索引立即数偏移寻址编码格式

指令使用基址寄存器 Rn 的值作为实际内存访问地址。当指令的执行条件满足时，将基址寄存器的值加/减偏移量产生新的地址值回写到 Rn 寄存器中。

（2）语法格式

```
    LDR|STR{<cond>}{B}{T}  <Rd>，[<Rn>]，±<offset_12>
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <offset_12>为 12 位立即数，内存访问地址偏移量。

（3）操作伪代码

```
    Address = Rn
    If  conditionPassed{cond}  then
        If  U = = 1 then
            Rn = Rn + offset_12
        Else
            Rn = Rn – offset_12
```

华清远见

（4）说明

① LDRBT、LDRT、STRBT 和 STRT 指令只支持后索引寻址。

② 如果 Rn 被指定为程序计数器 r15，指令的执行结果不可预知。

8．[Rn]，±<Rm>

（1）编码格式

指令的编码格式如图 4.21 所示。

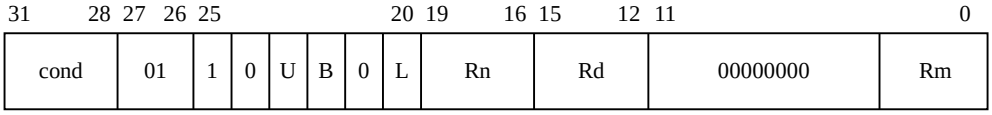


图 4.21　内存访问指令——后索引寄存器偏移寻址编码格式

指令访问地址为实际的基址寄存器的值。当指令的执行条件满足时，将基址寄存器的值加/减索引寄存器 Rm 的值回写到 Rn 基址寄存器。

（2）语法格式

```
LDR|STR{<cond>}{B}{T}  <Rd>, [Rn], ±<Rm>
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量。

（3）操作伪代码

```
Address = Rn
If  conditionPassed{cond}  then
    If  U = = 1  then
        Rn = Rn + Rm
    Else
        Rn = Rn – Rm
```

（4）说明

① LDRBT、LDRT、STRBT 和 STRT 指令只支持后索引寻址。

② 如果 Rm 和 Rn 指定为同一寄存器，指令的执行结果不可预知。

9．[Rn]，±Rm，<shift>#< offset_12>]

（1）编码格式

指令的编码格式如图 4.22 所示。

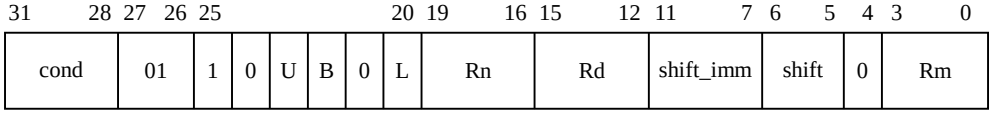


图 4.22　内存访问指令——带移位的后索引寄存器偏移寻址编码格式

实际的内存访问地址为寄存器 Rn 的值。当指令的执行条件满足时，将基址寄存器值加/减一个地址偏移量产生新的地址值。

（2）语法格式

语法格式有以下 5 种。

```
LDR|STR{<cond>}{B}{T}  <Rd>, [<Rn>], ±<Rm>, LSL #< offset_12>
LDR|STR{<cond>}{B}{T}  <Rd>, [<Rn>], ±<Rm>, LSR #< offset_12>
LDR|STR{<cond>}{B}{T}  <Rd>, [<Rn>], ±<Rm>, ASR #< offset_12>
LDR|STR{<cond>}{B}{T}  <Rd>, [<Rn>], ±<Rm>, ROR #< offset_12>
LDR|STR{<cond>}{B}{T}  <Rd>, [<Rn>], ±<Rm>, RRX
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量；
- LSL 表示逻辑左移操作；
- LSR 表示逻辑右移操作；
- ASR 表示算术右移操作；
- ROR 表示循环右移操作；
- RRX 表示扩展的循环右移。
- <shift_imm>为移位立即数。

（3）操作伪代码

```
Address = Rn
Case  shift  of
    0b00  /*LSL*/
         Index = Rm logic_shift_left  shift_imm
    0b01  /*LSR*/
         If  shift_imm = = 0  then /*LSR  #32*/
             Index = 0
         Else
             Index = Rm logical_shift_right  shift_imm
    0b10  /*ASR*/
         If  shift_imm = = 0 then /*ASR  #32*/
             If  Rm[31] = = 1 then
                 Index = 0xffffffff
             Else
                 Index = 0
         Else
             Index = Rm  Arithmetic_shift_Right  shift_imm
    0b11  /* ROR or RRX*/
         If  shift_imm = = 0 then /*RRX*/
             Index = (C flag Logical_shift_left 31) OR
                     (Rm logical_shift_Right 1)
         Else /*ROR*/
             Index = Rm Rotate_Right shift_imm
Endcase
If  ConditionPassed{cond}  then
   If  U = = 1 then
       Rn = Rn + index
   Else /*U = = 0*/
       Rn = Rn – index
```

（4）说明

① LDRBT、LDRT、STRBT 和 STRT 指令只支持后索引寻址。

② 当 PC 用作基址寄存器 Rn 或 Rm 时，指令执行结果不可预知。

③ 当 Rn 和 Rm 是同一个寄存器时，指令的执行结果不可预知。

4.2.2 杂类 Load/Store 指令的寻址方式

使用该类寻址方式的指令的语法格式如下。

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，<addressing_mode>
```

使用该类寻址方式的指令包括：（有符号/无符号）半字 Load/Store 指令、有符号字节 Load/Store 指令和双字 Load/Store 指令。

该类寻址方式分为 6 种类型，如表 4.4 所示。

表 4.4　　杂类 Load/Store 指令的寻址方式

	格　　式	模　　式
1	[Rn，#±<offset_8>]	立即数偏移寻址 （Immediate offset）
2	[Rn，±Rm]	寄存器偏移寻址 （Register offset）
3	[Rn，#±< offset_8>]!	立即数前索引寻址 （Immediate pre-indexed）
4	[Rn，±Rm]!	寄存器前索引寻址 （Register post-indexed）
5	[Rn]，#±< offset_8>	立即数后索引寻址 (Immediate post-indeded)
6	[Rn]，±<Rm>	寄存器后索引寻址 (Register post-indexed)

杂类 Load/Store 指令的解码格式如图 4.23 所示。

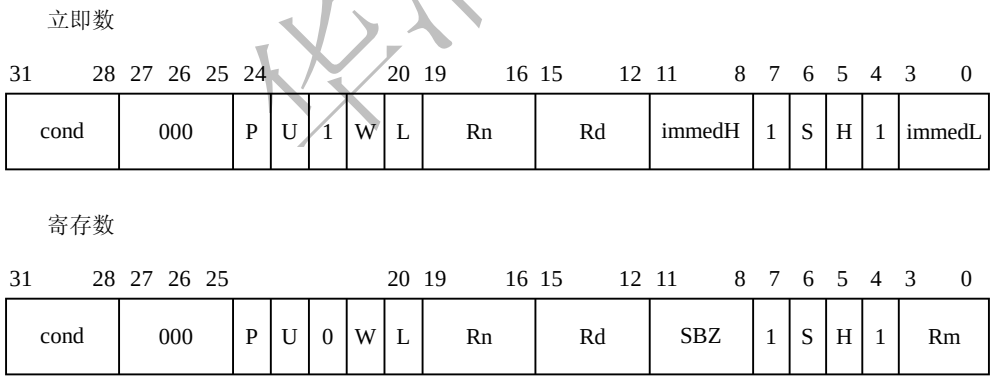


图 4.23　杂类 Load/Store 指令解码格式

编码格式中各标志位的含义如表 4.5 所示。

表 4.5　　杂类 Load/Store 指令编码格式各标志位含义

位　标　识	取　　值	含　　义
P	P=0	使用后索引寻址
	P=1	使用偏移地址或前索引寻址（由 W 位决定）

续表

位　标　识	取　　值	含　　义
U	U=0	访问的地址=基址寄存器的值-偏移量（offset）
	U=1	访问的地址=基址寄存器的值+偏移量（offset）

华清远见

W	W=0	如果 P=0，使用后索引寻址；P=1，指令不改变基址寄存器的值
	W=1	如果 P=0，未定义指令；如果 P=1，将计算的内存访问地址回写到基址寄存器
L	L=0	Store 指令
	L=1	Load 指令
S	S=0	无符号半字内存访问
	S=1	有符号半字内存访问
H	H=0	字节访问
	H=1	半字访问

注意

当 S=0 并且 H=0 时，并非无符号的字节内存访问指令。无符号的内存访问指令不使用该种寻址方式，详见本章上一节。

当 S=1 并且 L=0 时，并非是有符号的存储指令，而是未定义指令。ARM 指令并未区分有符号和无符号的字节和半字存储。

1．[Rn，#±<offset_8>]

（1）编码格式

指令的编码格式如图 4.24 所示。

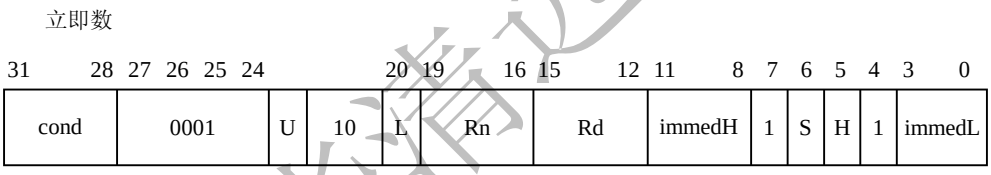


图 4.24　杂项内存访问指令——立即数偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）立即数 offset_8。

编程中，在访问结构体或记录（record）类型的变量时，这些内存的操作指令是十分有效的。另外，在子程序中，也常用这些指令访问本地变量和堆栈。当 offset_8=0 时，内存访问地址即基址寄存器 Rn 的值。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[<Rn>，#±<offset_12>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址。
- <offset_8>为 8 位立即数，内存访问地址偏移量。

（3）操作伪代码

```
offset_8 = (immedH << 4) OR immedL
If  U = = 1  then
    Address = Rn + offset_8
Else
    Address = Rn – offset_8
```

（4）说明

① 如果指令中没有指定立即数，使用[<Rn>]，编译器按[<Rn>，＃0]形式编码。

② 如果 Rn 被指定为程序计数器 r15，其值为当前指令地址加 8。

2．[Rn，±Rm]

（1）编码格式

指令的编码格式如图 4.25 所示。

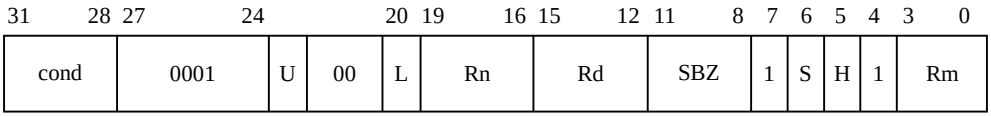


图 4.25　杂项内存访问指令——寄存器偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）偏移寄存器 Rm 的值。

该寻址方式适合使用指针访问数组中的单个数据成员。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[<Rn>，±<Rm>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量。

（3）操作伪代码

```
If  U = = 1  then
    Address = Rn + Rm
Else
    Address = Rn – Rm
```

（4）说明

如果 Rn 被指定为程序计数器 r15，其值为当前指令地址加 8；如果 r15 被用作偏移地址寄存器 Rm 的值，指令的执行结果不可预知。

3．[Rn，#±< offset_8>]!

（1）编码格式

指令的编码格式如图 4.26 所示。

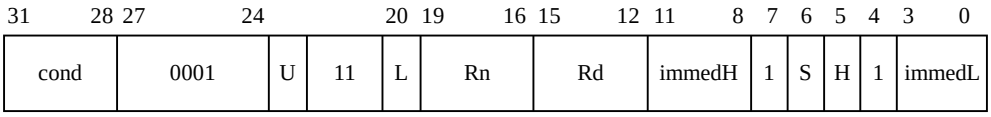


图 4.26　杂类内存访问指令——前索引立即数偏移寻址编码格式

内存地址为基址寄存器 Rn 加/减立即数 offset_8 的值。当指令执行的条件<cc>满足时，生成的地址写回基址寄存器 Rn 中。

该寻址方式适合访问数组自动进行数组下标的更新。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[<Rn>，±<offset_8>] !
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <offset_8>为 8 位立即数，内存访问地址偏移量，在指令编码格式中被拆为 immedH 和 immedL 两部分；

- ！设置指令编码中的 W 位，更新指令基址寄存器。

（3）操作伪代码

```
offset_8 = (immedH) << 4 OR immedL
If  U == 1  then
    Address = Rn + offset_8
Else
    Address = Rn – offset_8
If  ConditionPassed{cond}  then
    Rn = address
```

（4）说明

① 如果指令中没有指定立即数，使用[<Rn>]，编译器按[<Rn>，＃0]！形式编码。

② 如果 Rn 被指定为程序计数器 r15，指令的执行结果不可预知。

4．[Rn，±Rm]！

（1）编码格式

指令的编码格式如图 4.27 所示。

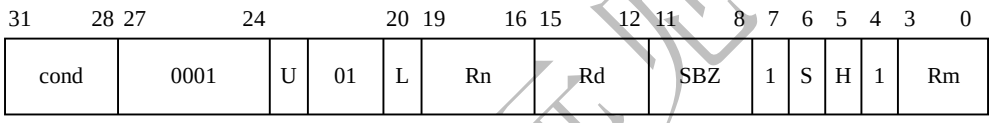


图 4.27　杂项内存访问指令——前索引寄存器偏移寻址编码格式

内存访问地址为基址寄存器 Rn 的值加（或减）偏移寄存器 Rm 的值。当指令的执行条件<cc>满足时，生成地地址将写回基址寄存器。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[<Rn>，±<Rm>]
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量；
- ！设置指令编码中的 W 位，更新指令基址寄存器。

（3）操作伪代码

```
If  U = = 1  then
    Address = Rn + Rm
Else
    Address = Rn – Rm
If  ConditionPassed{cond}  then
    Rn = address
```

（4）说明

① 如果 Rn 和 Rm 指定为同一寄存器，指令的执行结果不可预知。

② 如果程序计数器 r15 被用作 Rm 或 Rn，则指令的执行结果不可预知。

5．[Rn]，#±< offset_8>

（1）编码格式

指令的编码格式如图 4.28 所示。

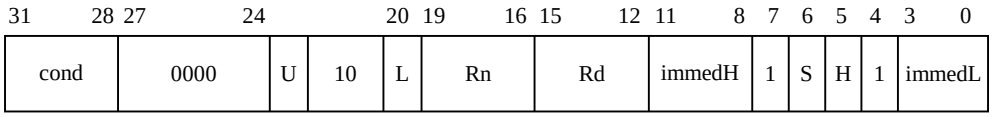


图 4.28　杂项内存访问指令——后索引立即数偏移寻址编码格式

指令使用基址寄存器 Rn 的值作为实际内存访问地址。当指令的执行条件满足时，将基址寄存器的值加/减偏移量生产新的地址值回写到 Rn 寄存器中。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[<Rn>]，±<offset_8>
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <offset_8>为 8 位立即数，内存访问地址偏移量。

（3）操作伪代码

```
Address = Rn
Offset_8 = (immedH << 4) OR immedL
If  conditionPassed{cond}  then
    If  U = = 1 then
        Rn = Rn + offset_8
    Else
        Rn = Rn – offset_8
```

（4）说明

① 当指令中没有指定立即数时，汇编器按“[<Rn>]，＃0”编码。

② 如果 Rn 被指定为程序计数器 r15，指令的执行结果不可预知。

6．[Rn]，±<Rm>

（1）编码格式

指令的编码格式如图 4.29 所示。

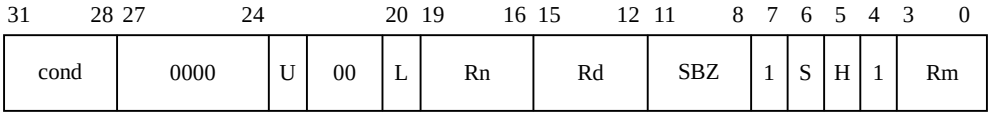


图 4.29　杂项内存访问指令——后索引寄存器偏移寻址编码格式

指令访问地址为实际的基址寄存器的值。当指令的执行条件满足时，将基址寄存器的值加/减索引寄存器 Rm 的值回写到 Rn 基址寄存器。

（2）语法格式

```
LDR|STR{<cond>}H|SH|SB|D  <Rd>，[Rn]，±<Rm>
```

其中：

- Rn 为基址寄存器，该寄存器包含内存访问的基地址；
- <Rm>为偏移地址寄存器，包含内存访问地址偏移量。

（3）操作伪代码

```
Address = Rn
If  conditionPassed{cond}  then
    If  U = = 1  then
        Rn = Rn + Rm
    Else
        Rn = Rn – Rm
```

（4）说明

① 程序寄存器 r15 被指定为 Rm 或 Rn，指令的执行结果不可预知。

② 如果 Rm 和 Rn 指定为同一寄存器，指令的执行结果不可预知。

4.2.3 批量 Load/Store 指令寻址方式

批量 Load/Store 指令将一片连续内存单元的数据加载到通用寄存器组中或将一组通用寄存器的数据存储到内存单元中。

批量 Load/Store 指令的寻址模式产生一个内存单元的地址范围，指令寄存器和内存单元的对应关系满足这样的规则，即编号低的寄存器对应于内存中低地址单元，编号高的寄存器对应于内存中的高地址单元。

指令的语法格式如下。

```
LDM|STM{<cond>}<addressing_mode>  <Rn>{!}，<registers><^>
```

指令的寻址方式如表 4.6 所示。

表 4.6　　批量 Load/Store 指令的寻址方式

	格　　式	模　　式
1	IA（Increment After）	后递增方式
2	IB（Increment Before）	先递增方式
3	DA（Decrement After）	后递减方式
4	DB（Decrement Before）	先递减方式

指令的编码格式如图 4.30 所示。

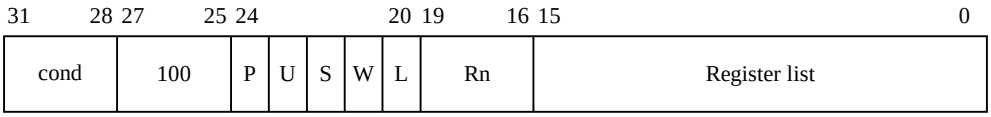


图 4.30　批量 Load/Store 指令编码格式

编码格式中各标志位的含义如表 4.7 所示。

表 4.7　　批量 Load/Store 指令编码格式各标志位含义

位标识	取　　值	含　　义
P	P=0	Rn 包含的地址，是所要访问的内存块的高地址（U=0）还是低地址（U=1）
	P=1	标识 Rn 所指向的内存单元是否被访问
U	U=0	Rn 所指内存单元为所要访问的内存单元块的高地址
	U=1	Rn 所指内存单元为所要访问的内存单元块的低地址
S	S=0	当程序计数器 PC 作为要加载的寄存器之一时，S 标识是否将 spsr 内容拷贝到 cpsr；对于不加载 PC 的 load 指令和所有 store 指令，S 标识特权模式下，使用用户模式寄存器组代替当前模式下寄存器组
	S=1	
W	W=0	数据传送完毕，更新地址寄存器内容

	W=1	
L	L=0	Store 指令
	L=1	Load 指令

1．IA 寻址

（1）编码格式

指令的编码格式如图 4.31 所示。

该寻址方式指定一片连续的内存地址空间，地址空间的大小<address_length>等于寄存器列表中寄存器数目的 4 倍。内存地址范围起始地址<start_address>等于基址寄存器 Rn 的值。结束地址<end_address>等于起始地址<start_address>加上地址空间大小<address_length>。

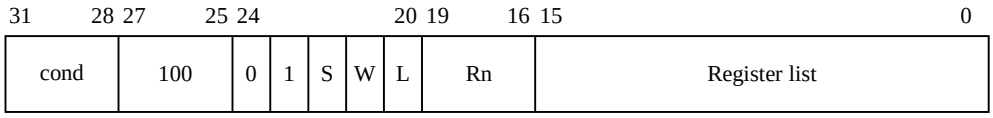


图 4.31　批量 Load/Store 指令——后增加寻址

地址空间中的每个内存单元对应寄存器列表中的一个寄存器。编号低的寄存器对应于内存中低地址单元，编号高的寄存器对应于内存中的高地址单元。

当指令执行条件满足并且指令编码格式中 W 位置位，基址寄存器 Rn 的值等于内存地址范围结束地址<end_address>加 4。

（2）语法格式

```
LDM|STM{<cond>}IA  <Rn>{!}，<registers><^>
```

其中：

- IA 标识指令使用“后增加”寻址方式；
- Rn 为基址寄存器，包含内存访问的基地址；
- <registers>为指令操作的寄存器列表；
- <^>表示如果寄存器列表中包含程序计数器 PC，是否将 spsr 拷贝到 cpsr。

（3）操作伪代码

```
Start_address = Rn
End_address = Rn + (Number_of_Set_Bits_In(register_list)*4) – 4
If  conditionPassed(cond) and W = = 1 then
    Rn = Rn + (Number_of_Set_Bits_In(register_list)*4)
```

2．DA 寻址

（1）编码格式

指令的编码格式如图 4.32 所示。

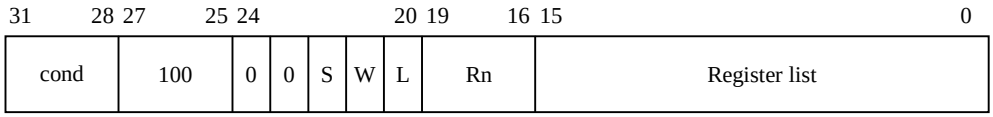


图 4.32　批量 Load/Store 指令——后递减寻址

该寻址方式指定一片连续的内存地址空间，地址空间的大小<address_length>等于寄存器列表中寄存器数目的 4 倍。内存地址范围起始地址<start_address>等于基址寄存器 Rn 的值减去地址空间大小<address_length>并加 4。结束地址<end_address>等于基址寄存器的值。

地址空间中的每个内存单元对应寄存器列表中的一个寄存器。编号低的寄存器对应于内存中低地址单元，编号高的寄存器对应于内存中的高地址单元。

当指令执行条件满足并且指令编码格式中 W 位置位时，基址寄存器 Rn 的值等于内存地址范围起始地址<start_address>减 4。

（2）语法格式

```
LDM|STM{<cond>}IA  <Rn>{!}，<registers><^>
```

其中：

- DA 标识指令使用“后递减”寻址方式；
- Rn 为基址寄存器，包含内存访问的基地址；
- <registers>为指令操作的寄存器列表；
- <^>表示如果寄存器列表中包含程序计数器 PC，是否将 spsr 拷贝到 cpsr。

（3）操作伪代码

```
Start_address = Rn – (Number_of_Set_Bits_In(register_list)*4) + 4
End_address = Rn
If  conditionPassed(cond) and W = = 1 then
    Rn = Rn - (Number_of_Set_Bits_In(register_list)*4)
```

3．IB 寻址

（1）编码格式

指令的编码格式如图 4.33 所示。

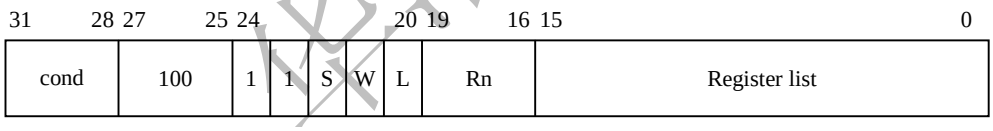


图 4.33　批量 Load/Store 指令——前增加寻址

该寻址方式指定一片连续的内存地址空间，地址空间的大小<address_length>等于寄存器列表中寄存器数目的 4 倍。内存地址范围起始地址<start_address>等于基址寄存器 Rn 的值加 4。结束地址<end_address>等于起始地址<start_address>加上地址空间大小<address_length>。

地址空间中的每个内存单元对应寄存器列表中的一个寄存器。编号低的寄存器对应于内存中低地址单元，编号高的寄存器对应于内存中的高地址单元。

当指令执行条件满足并且指令编码格式中 W 位置位，基址寄存器 Rn 的值等于内存地址范围结束地址<end_address>。

（2）语法格式

```
LDM|STM{<cond>}IB  <Rn>{!}，<registers><^>
```

其中：

- IB 标识指令使用“前增加”寻址方式；
- Rn 为基址寄存器，包含内存访问的基地址；
- <registers>为指令操作的寄存器列表；
- <^>表示如果寄存器列表中包含程序计数器 PC，是否将 spsr 拷贝到 cpsr。

（3）操作伪代码

```
Start_address = Rn + 4
End_address = Rn + (Number_of_Set_Bits_In(register_list)*4)
If  ConditionPassed(cond) and W= = 1  then
    Rn = Rn + (Number_Of_Set_Bits_In(register_list)*4)
```

4．DB 寻址

（1）编码格式

指令的编码格式如图 4.34 所示。

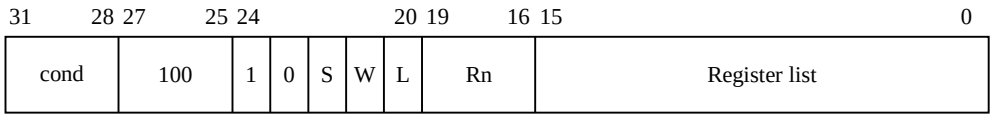


图 4.34　批量 Load/Store 指令——前递减寻址

该寻址方式指定一片连续的内存地址空间，地址空间的大小<address_length>等于寄存器列表中寄存器数目的 4 倍。内存地址范围起始地址<start_address>等于基址寄存器 Rn 的值减地址空间的大小<address_length>。结束地址<end_address>等于基址寄存器的值减 4。

地址空间中的每个内存单元对应寄存器列表中的一个寄存器。编号低的寄存器对应于内存中低地址单元，编号高的寄存器对应于内存中的高地址单元。

当指令执行条件满足并且指令编码格式中 W 位置位，基址寄存器 Rn 的值等于内存地址范围起始地址<address_address>。

（2）语法格式

```
LDM|STM{<cond>}DB  <Rn>{!}，<registers><^>
```

其中：

- DB 标识指令使用“前递减”寻址方式；
- Rn 为基址寄存器，包含内存访问的基地址；
- <registers>为指令操作的寄存器列表；
- <^>表示如果寄存器列表中包含程序计数器 PC，是否将 spsr 拷贝到 cpsr。

（3）操作伪代码

```
Start_address = Rn - (Number_Of_Set_Bits_In(register_list)*4)
End_address = Rn - 4
If  ConditionPassed(cond) and W = = 1  then
    Rn = Rn – (Number_Of_Set_Bits_In(register_list)*4)
```

4.2.4　堆栈操作寻址方式

堆栈操作寻址方式和批量 Load/Store 指令寻址方式十分类似。但对于堆栈的操作，数据写入内存和从内存中读出要使用不同的寻址模式，因为进栈操作（pop）和出栈操作（push）要在不同的方向上调整堆栈。下面详细讨论如何使用合适的寻址方式实现数据的堆栈操作。

根据不同的寻址方式，将堆栈分为以下 4 种。

① Full 栈：堆栈指针指向栈顶元素（last used location）。

② Empty 栈：堆栈指针指向第一个可用元素（the first unused location）。

③ 递减栈：堆栈向内存地址减小的方向生长。

④ 递增栈：堆栈向内存地址增加的方向生长。

根据堆栈的不同种类，将其寻址方式分为以下 4 种。

① 满递减 FD（Full Descending）。

② 空递减 ED（Empty Descending）。

③ 满递增 FA（Full Ascending）。

④ 空递增 EA（Empty Ascending）。

注意 如果程序中有对协处理器数据的进栈/出栈操作，最好使用 FD 或 EA 类型堆栈。这样可以使用一条 STC 或 LDC 指令将数据进栈或出栈。

表 4.8 显示了堆栈的寻址方式和批量 Load/Store 指令寻址方式的对应关系。

表 4.8　　堆栈寻址方式和批量 Load/Store 指令寻址方式对应关系

批量数据寻址方式	堆栈寻址方式	L 位	P 位	U 位
LDMDA	LDMFA	1	0	0
LDMIA	LDMFD	1	0	1
LDMDB	LDMEA	1	1	0
LDMIB	LDMED	1	1	1
STMDA	STMED	0	0	0
STMIA	STMEA	0	0	1
STMDB	STMFD	0	1	0
STMIB	STMFA	0	1	1

4.2.5　协处理器 Load/Store 寻址方式

协处理器 Load/Store 指令的语法格式如下。

```
<opcode>{<cond>}{L}  <coproc>，<CRd>，<addressing_mode>
```

表 4.9 显示了该类指令的寻址方式。

表 4.9　　协处理器 Load/Store 指令寻址方式

	格　　式	说　　明
1	[<Rn>,#±<offset_8>*4]	立即数偏移寻址
2	[<Rn>,#±<offset_8>*4]!	前索引立即数偏移寻址
3	[<Rn>],#±<offset_8>*4	后索引立即数偏移寻址
4	[<Rn>], <option>	直接寻址

协处理器 Load/Store 指令的编码方式如图 4.35 所示。

编码格式中各标志位的含义如表 4.10 所示。

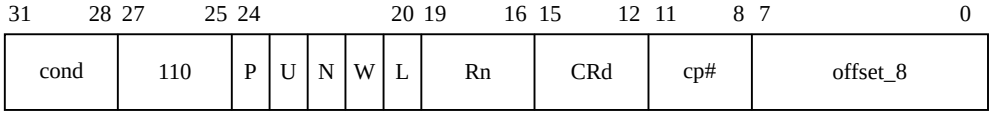


图 4.35　协处理器 Load/Store 指令编码格式

表 4.10　　协处理器 Load/Store 指令编码格式各标志位含义

位　标　识	取　　值	含　　义

P	P=0	标识使用偏移寻址还是前索引寻址（由 W 位决定）
	P=1	标识使用后索引寻址还是直接寻址（由 W 位决定）
U	U=0	从基地址中减去偏移量 offset
	U=1	从基地址中加上偏移量 offset
N	N=0	和具体使用的协处理器相关
	N=1	
W	W=0	指令执行结束，不改变基址寄存器的值
	W=1	访问的内存地址回写到基址寄存器
L	L=0	Store 指令
	L=1	Load 指令

1．[<Rn>,#±<offset_8>*4]

（1）编码格式

指令的编码格式如图 4.36 所示。

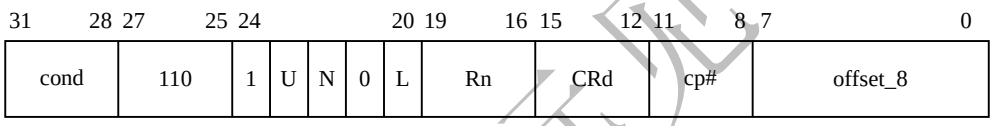


图 4.36　协处理器 Load/Store 指令——立即数寻址

该寻址方式指定一片连续的内存地址空间。访问内存单元的第一个地址<first_addressing>等于基址寄存器<Rn>的值加上/减去指令中寄存器值的 4 倍。接下来的内存访问地址是前一个访问地址加 4。当协处理器发出传输中止信号时，数据传送结束。

这种寻址方式的数据传输数目由协处理器决定。

注意　这种寻址方式最多允许传输 16 的字。

（2）语法格式

```
<opcode>{<cond>}{L}  <coproc>，<CRd>，[<Rn>,#±<offset_8>*4]
```

其中：

- <Rn>为基址寄存器，包含寻址操作的基地址；
- <offset_8>为 8 位立即数，该值的 4 倍为地址偏移量。

（3）操作伪代码

```
If  ConditionPassed(cond)  then
    If  U = = 1 then
        Address = Rn + offset_8 * 4
    Else  /*U = = 0*/
        Address = Rn – offset_8 * 4
    Start_address = address
    While (NotFinished(coprocessor[cp_num]))
        Address = address +4
    End_address = address
```

（4）说明

如果基址寄存器指定为程序计数器 r15，则基地址为当前执行指令地址加 8。

2．[<Rn>,#±<offset_8>*4]!

（1）编码格式

指令的编码格式如图 4.37 所示。

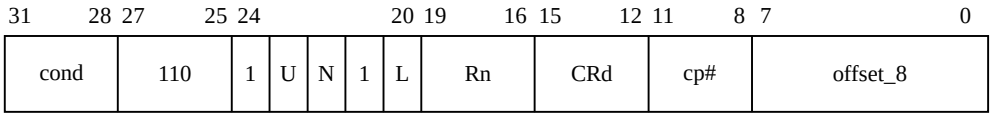


图 4.37　协处理器 Load/Store 指令——前索引立即数寻址

该寻址方式指定一片连续的内存地址空间。访问内存单元的第一个地址<first_addressing>等于基址寄存器<Rn>的值加上/减去指令中寄存器值的 4 倍。如果指令的条件域满足，产生的<first_addressing>回写到基址寄存器 Rn 中。接下来的内存访问地址是前一个访问地址加 4。当协处理器发出传输中止信号时，数据传送结束。

这种寻址方式的数据传输数目由协处理器决定。

注意　这种寻址方式最多允许传输 16 的字。

（2）语法格式

```
<opcode>{<cond>}{L}  <coproc>，<CRd>，[<Rn>,#±<offset_8>*4]!
```

其中：

- <Rn>为基址寄存器，包含寻址操作的基地址；
- <offset_8>为 8 位立即数，该值的 4 倍为地址偏移量；
- ！设置指令编码中的 W 位，更新指令基地址。

（3）操作伪代码

```
If  ConditionPassed(cond)  then
    If  U = = 1 then
        Rn = Rn + offset_8 * 4
    Else  /*U = = 0*/
        Rn = Rn – offset_8 * 4
    Start_address = Rn
    Address = start_address
    While (NotFinished(coprocessor[cp_num]))
        Address = address +4
    End_address = address
```

（4）说明

如果基址寄存器指定为程序计数器 r15，则指令的执行结果不可预知。

3．[<Rn>],#±<Offset_8>*4

（1）编码格式

指令的编码格式如图 4.38 所示。

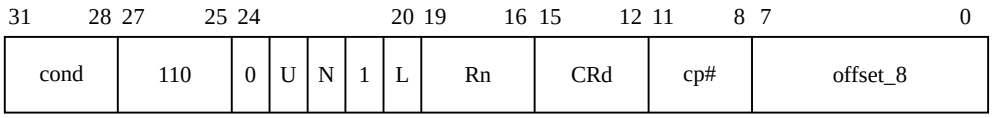


图 4.38　协处理器 Load/Store 指令——后索引立即数寻址

该寻址方式指定一片连续的内存地址空间。访问内存单元的第一个地址<first_addressing>等于基址寄存器<Rn>的值。接下来的内存访问地址是前一个访问地址加 4。当协处理器发出传输中止信号时，数据传送结束。如果指令的条件域满足，Rn 基址寄存器的值更新为 Rn 的值加上/减去 8 位立即数的 4 倍。
这种寻址方式的数据传输数目由协处理器决定。

注意　这种寻址方式最多允许传输 16 的字。

（2）语法格式

```
<opcode>{<cond>}{L}  <coproc>，<CRd>，[<Rn>]，#±<offset_8>*4
```

其中：

- <Rn>为基址寄存器，包含寻址操作的基地址；
- <offset_8>为 8 位立即数，该值的 4 倍为地址偏移量。

（3）操作伪代码

```
If  ConditionPassed(cond)  then
    Start_address = Rn
    If  U = = 1 then
        Rn = Rn + offset_8 * 4
    Else  /*U = = 0*/
        Rn = Rn – offset_8 * 4
    Address = start_address
    While (NotFinished(coprocessor[cp_num]))
        Address = address +4
    End_address = address
```

（4）说明
如果基址寄存器指定为程序计数器 r15，则指令的执行结果不可预知。

4．[<Rn>], <Option>

（1）编码格式
指令的编码格式如图 4.39 所示。

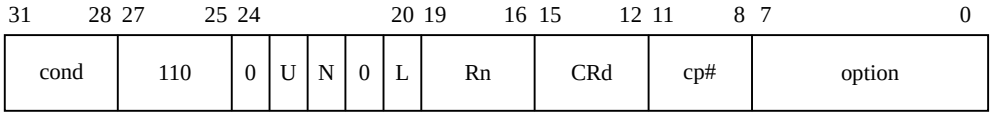


图 4.39　协处理器 Load/Store 指令——直接寻址

该寻址方式指定一片连续的内存地址空间。访问内存单元的第一个地址<first_addressing>等于基址寄存器<Rn>的值。接下来的内存访问地址是前一个访问地址加 4。当协处理器发出传输中止信号时，数据传送结束。
指令不更新基址寄存器的值。指令编码格式中 bits[7：0]保留，所以可以将空闲位用作协处理器指令扩展。
这种寻址方式的数据传输数目由协处理器决定，最多可以传输 16 字。
（2）语法格式

```
<opcode>{<cond>}{L}  <coproc>，<CRd>，[<Rn>]，<Option>
```

其中：

- <Rn>为基址寄存器，包含寻址操作的基地址；
- <option>用作协处理器指令扩展。

（3）操作伪代码

```
If  ConditionPassed(cond)  then
    Start_address = Rn
    Address = start_address
    While (NotFinished(coprocessor[cp_num]))
        Address = address +4
    End_address = address
```

（4）说明

如果基址寄存器指定为程序计数器 r15，则寻址基地址为当前指令地址加 8。

联系方式

集团官网：www.hqyj.com　　嵌入式学院：www.embedu.org　　移动互联网学院：www.3g-edu.org

企业学院：www.farsight.com.cn　　物联网学院：www.topsight.cn　　研发中心：dev.hqyj.com

集团总部地址：北京市海淀区西三旗悦秀路北京明园大学校内 华清远见教育集团

北京地址：北京市海淀区西三旗悦秀路北京明园大学校区，电话：010-82600386/5

上海地址：上海市徐汇区漕溪路银海大厦 A 座 8 层，电话：021-54485127

深圳地址：深圳市龙华新区人民北路美丽 AAA 大厦 15 层，电话：0755-22193762

成都地址：成都市武侯区科华北路 99 号科华大厦 6 层，电话：028-85405115

南京地址：南京市白下区汉中路 185 号鸿运大厦 10 层，电话：025-86551900

武汉地址：武汉市工程大学卓刀泉校区科技孵化器大楼 8 层，电话：027-87804688

西安地址：西安市高新区高新一路 12 号创业大厦 D3 楼 5 层，电话：029-68785218

华清远见