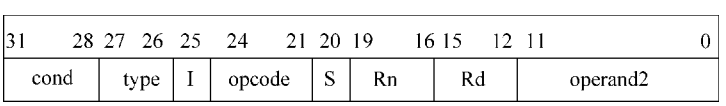
第3章 ARM指令集寻址方式

ARM指令寻址方式可分为四大类：数据处理指令寻址、Load/Store指令的寻址、批量Load/Store指令的寻址和协处理指令寻址。

3.1 ARM指令的编码格式

1. 一般编码格式

每条ARM指令占有4个字节，其指令长度为32位。典型的ARM指令编码格式如下。



其中：

cond(bit[31:28])：指令执行的条件码；

type(bit[27:26])：指令类型码，根据其编码的不同，所代表各类型如表3-1所示。

表3-1 指令类型码描述

type(bit[27:26])	描　　述
00	数据处理指令及杂类Load/Store指令
01	Load/Store指令
10	批量Load/Store指令及分支指令
11	协处理指令与软中断指令

I(bit[25])：第二操作数类型标志码。在数据处理指令里I=1时表示第二操作数是立即数，I=0时表示第二操作数是寄存器或寄存器移位形式。

opcode(bit[24:21])：指令操作码。

S(bit[20])：决定指令的操作结果是否影响CPSR。

Rn(bit[19:16])：包含第一个操作数的寄存器编码。

Rd(bit[15:12])：目标寄存器编码。

operand2(bit[11:0])：指令第二个操作数。

ARM汇编指令语法格式：

<opcode>{<cond>}{S}<Rd>,<Rn>,<operand2>

2. 指令条件码

条件码 cond(bit[31:28])在指令中共占有四位,其组合形式共有 16 种,如表 3-2 所示。

表 3-2 条件码

条 件 码	条件码助记符	描 述	PSR 中的标志位
0000	EQ	相等	Z=1
0001	NE	不相等	Z=0
0010	CS/HS	无符号大于等于	C=1
0011	CC/LO	无符号小于	C=0
0100	MI	负数	N=1
0101	PL	非负数	N=0
0110	VS	上溢出	V=1
0111	VC	没有上溢出	V=0
1000	HI	无符号数大于	C=1 且 Z=0
1001	LS	无符号小于等于	C=0 或 Z=1
1010	GE	有符号数大于等于	N=1 且 V=1 或 N=0 且 V=0
1011	LT	有符号数小于	N=1 且 V=0 或 N=0 且 V=1
1100	GT	有符号数大于	Z=0 且 N=V
1101	LE	有符号数小于/等于	Z=1 或 N!=V
1110	AL	无条件执行	

3.2 数据处理指令寻址方式

1. 数据处理指令第二操作数的构成方式

数据处理指令第二操作数 operand2 的构成有三种格式:

1) 立即数方式

每个立即数由一个 8 位的常数进行 32 位循环右移偶数位得到,其中循环右移的位数由一个 4 位二进制的两倍表示。即

<immediate>=immed_8 进行 32 位循环右移(2 * rotate_4)位

合法的立即数,例如:0xff,0x104。不合法的立即数,例如:0x101,0x102。

规则:当立即数值在 0~0xff 范围时,令 immed_8=immediate,rotate_4=0;在其他情况下,汇编编译器选择使 rotate_4 数值最小的编码方式。

2) 寄存器方式

操作数即为寄存器的数值,例如:

```
MOV R3,R2
ADD R0,R1,R2
```

3) 寄存器移位方式

操作数为寄存器的数值做相应的移位而得到。在 ARM 指令中移位操作包括逻辑左移、逻辑右移、算术左移、算术右移、循环右移和带扩展的循环右移,这些操作的功能如图 3-1

所示。

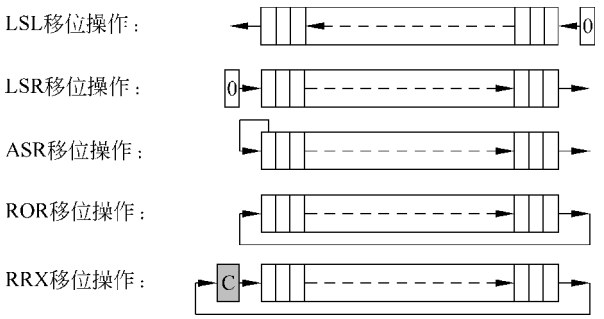


图 3-1　移位操作功能示意

LSL 逻辑左移：空出的最低有效位用 0 填充；

LSR 逻辑右移：空出的最高有效位用 0 填充；

ASL 算术左移：同 LSL；

ASR 算术右移：空出的最高有效位用“符号位”填充；

ROR 循环右移：移出的最低有效位依次填入空出的最高有效位；

RRX 带扩展的循环右移：将寄存器内容循环右移 1 位，空位用原来的 C 标志位填充，移出的最低有效位填入 C 标志位。

2. 具体寻址类型

数据处理指令寻址方式具体可分为 5 种类型，下面分别加以介绍。

1）第二操作数为立即数

汇编语法格式：

#<immediate>

指令编码格式如下：

31 28	27 26	25	24 21	20	19 16	15 12	11 8	7 0
cond	0 0	1	opcode	S	Rn	Rd	rotate_4	immed_8

其中由一个 8 位的常数 immed_8 进行 32 位循环右移 rotate_4 的 2 倍位得到的立即数 immediate 作为数据处理指令的第二操作数。例如：

```
MOV R0,#0xFC0      ; R0<—0xFC0
```

2）第二操作数为寄存器

汇编语法格式：

<Rm>

指令编码格式如下：

31 28	27 26	25	24 21	20	19 16	15 12	11 4	3 0
cond	0 0	0	opcode	S	Rn	Rd	全为0	Rm

例如：

```
ADD R0,R1,R2          ; R0<—R1+R2
```

3）第二操作数为寄存器移位方式，且移位的位数为一个 5 位的立即数

汇编语法格式：

```
<Rm>,<shift> #<shift_imm>
```

指令编码格式如下：

31 28	27 26	25	24 21	20	19 16	15 12	11 7	6 5	4	3 0
cond	0 0	0	opcode	S	Rn	Rd	shift_amount	shift	0	Rm

其中：

shift_amount 表示移位数量；

shift 表示移位类型编码，bit[5]用 H 表示，bit[6]用 S 表示，其描述如表 3-3 所示。

表 3-3　shift 编码描述

S	H	描　述
0	0	逻辑左移 LSL
0	1	逻辑右移 LSR
1	0	算术右移 ASR
1	1	循环右移 ROR

指令的操作数<operand2>为寄存器 Rm 的数值按某种移位方式移动 shift_amount 位，这里 shift_amount 范围是 0～31，当 shift_amount＝0 时，移位位数为 32，则移位数范围为 1～32。例如：

```
MOV R0,R0,LSL #n          ; R0<—R0 * (2^n) ( n= 0～31 )
```

4）第二操作数为寄存器移位方式，且移位数值放在寄存器中

汇编语法格式：

```
<Rm>,<shift> <Rs>
```

指令编码格式如下：

31 28	27 26	25	24 21	20	19 16	15 12	11 8	7	6 5	4	3 0
cond	0 0	0	opcode	S	Rn	Rd	Rs	0	shift	1	Rm

其中：

寄存器 Rs 中存放着要移位的数量；

shift 表示移位类型编码，其描述如表 3-3 所示。

指令的操作数<operand2>为寄存器 Rm 的数值进行移位得到。移位的数由 Rs 的低 8 位 bit[7:0]决定。

注意事项：

当 R15 用做 Rn、Rm、Rd、Rs 任一个时，指令会产生不可预知的结果。

5）第二操作数为寄存器进行 RRX 移位得到

汇编语法格式：

＜Rm＞，RRX

指令编码格式如下：

31 28	27 26	25	24 21	20	19 16	15 12	11 8	7 4	3 0
cond	0 0	0	opcode	S	Rn	Rd	0 0 0 0	0 1 1 0	Rm

指令的操作数＜operand2＞为寄存器 Rm 中的数值进行带扩展的循环右移一位，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。

3.3 Load/Store 指令寻址

Load/Store 指令是对内存进行存储/加载数据操作的指令，根据访问的数据格式的不同，将这类指令的寻址分为字、无符号字节的 Load/Store 指令寻址和半字、有符号字节 Load/Store 指令寻址两大类。

3.3.1 地址计算方法

1. 寄存器间接寻址

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

```
LDR   R0,[R1]          ; R0 ←[R1]
STR   R0,[R1]          ; [R1]← R0
```

在第一条指令中，以寄存器 R1 的值作为操作数的地址，在存储器中取得一个操作数后将其送入 R0 中。

第二条指令将 R0 的值传送到以 R1 的值为地址的存储器中。

2. 基址加变址寻址

基址变址寻址就是将寄存器（该寄存器一般称作基址寄存器）的内容与指令中给出的地址偏移量相加，从而得到一个操作数的有效地址。变址寻址方式常用于访问某基地址附近的地址单元。

根据访问存储单元和基地址寄存器更新的先后顺序可以将基址加变址寻址分为两种：

（1）前变址法：基地址寄存器中的值和地址偏移量先作加减运算，生成的操作数作为内存访问的地址。

（2）后变址法：将基地址寄存器中的值直接作为内存访问的地址进行操作，内存访问完毕后基地址寄存器中的值和地址偏移量作加减运算，并更新基地址寄存器。

采用变址寻址方式的指令常见有以下几种形式：

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

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

在第二条指令中,将寄存器 R1 的内容加上 4 形成操作数的有效地址,从而取得操作数存入寄存器 R0 中;然后,R1 的内容自增 4(也就是 R1 的内容加 4 后写回到 R1 中)。

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

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。

3.3.2 字、无符号字节寻址

在 Load/Store 指令中,字与无符号字节操作指令编码格式如下:

31 28	27	26	25	24	23	22	21	20	19 16	15 12	11 0
cond	0	1	I	P	U	B	W	L	Rn	Rd	Addressing_mode

汇编指令语法格式如下。

加载指令:

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

存储指令:

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

其中:

cond 为指令执行的条件,Rn 为基址寄存器,Rd 为源/目标寄存器,Addressing_mode 内存地址构成格式。

$$I=\begin{cases}1:\text{偏移量为寄存器或寄存器移位形式}\\0:\text{偏移量为 12 位立即数}\end{cases}$$

$$P=\begin{cases}1:\text{前变址操作}\\0:\text{后变址操作}\end{cases}$$

$$U=\begin{cases}1:\text{内存地址 address 为基址寄存器 Rn 值加上地址偏移量}\\0:\text{内存地址 address 为基址寄存器 Rn 值减去地址偏移量}\end{cases}$$

$$B=\begin{cases}1:\text{指令访问的是无符号的字节数据}\\0:\text{指令访问的是字数据}\end{cases}$$

$$W=\begin{cases}1:\text{执行基地址寄存器回写操作}\\0:\text{不执行基地址寄存器回写操作}\end{cases}$$

$$L=\begin{cases}1:\text{执行 Load 操作}\\0:\text{执行 Store 操作}\end{cases}$$

下面我们将重点讨论内存地址构成格式 Addressing_mode。

根据指令编码和汇编语法格式的不同，归纳起来共有以下 3 种内存地址构成格式。

1. Addressing_mode 中的偏移量为立即数

Addressing_mode 中的偏移量为立即数的汇编指令寻址按编码格式可分为以下 3 种形式。

前变址不回写形式：

[<Rn>,#+/-<immed_offset>]

前变址回写形式：

[<Rn>,#+/-<immed_offset>]!

后变址回写形式：

[<Rn>],#+/-<immed_offset>

其指令编码如下：

31 28	27	26	25	24	23	22	21	20	19 16	15 12	11 0
cond	0	1	0	P	U	B	W	L	Rn	Rd	immed_offset

$$P=\begin{cases}1：前变址操作\\0：后变址操作\end{cases}$$

$$U=\begin{cases}1：内存地址\ address\ 为基址寄存器\ Rn\ 值加上地址偏移量\\0：内存地址\ address\ 为基址寄存器\ Rn\ 值减去地址偏移量\end{cases}$$

$$B=\begin{cases}1：指令访问的是无符号的字节数据\\0：指令访问的是字数据\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

Addressing_mode 中的偏移量为立即数的汇编指令寻址编码对应的汇编语法格式类型如表 3-4 所示。下面介绍各汇编语法格式下的地址计算方法：

(1) [<Rn>,#+/-<immed_offset>]

内存地址为基地址寄存器值加上/减去 immed_offset，其加减由 U 的值来确定。

(2) [<Rn>,#+/-<immed_offset>]!

内存地址为基地址寄存器值加上/减去 immed_offset，其加减由 U 的值来确定。当指令执行时，生成的地址值将写入基地址寄存器。

(3) [<Rn>],#+/-<immed_offset>

内存地址为基地址寄存器值，当存储器操作完毕后，将基地址寄存器 Rn 值加上/减去 immed_offset，将所得到的值写回到基地址寄存器 Rn(更新基地址寄存器)，其加减由 U 的值来确定。

表 3-4　偏移量为立即数的指令编码类型

W	P	汇编语法格式
0	1	[<Rn>,#+/-<immed_offset>]
1	0	[<Rn>],#+/-<immed_offset>
1	1	[<Rn>,#+/-<immed_offset>]!

【例 3-1】

```
LDR R0,[R1,#4]; R0<—[R1+4]
LDR R0,[R1,#-4]; R0<—[R1-4]
LDR R0,[R1,#4]!; R0<—[R1+4],同时 R1=R1+4
LDR R0,[R1],#4; R0<—[R1],R1=R1+4
```

2. Addressing_mode 中的偏移量为寄存器的值

Addressing_mode 中的偏移量为寄存器的汇编指令寻址按编码格式可分为以下 3 种形式。

前变址不回写形式：

[<Rn>,+/-<Rm>]

前变址回写形式：

[<Rn>,+/-<Rm>]!

后变址回写形式：

[<Rn>],+/-<Rm>

其指令编码如下：

31 28	27	26	25	24	23	22	21	20	19 16	15 12	11 4	3 0
cond	0	1	1	P	U	B	W	L	Rn	Rd	全为0	Rm

$$P=\begin{cases}1：前变址操作\\0：后变址操作\end{cases}$$

$$U=\begin{cases}1：内存地址\ address\ 为基址寄存器\ Rn\ 值加上地址偏移量\\0：内存地址\ address\ 为基址寄存器\ Rn\ 值减去地址偏移量\end{cases}$$

$$B=\begin{cases}1：指令访问的是无符号的字节数据\\0：指令访问的是字数据\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

Addressing_mode 中的偏移量为寄存器的汇编指令寻址编码对应的汇编语法格式类型如表 3-5 所示。下面介绍各汇编语法格式下的地址计算方法：

(1) [<Rn>,+/-<Rm>]

内存地址为基地址寄存器值加上/减去 Rm，其加减由 U 的值来确定。

(2) [<Rn>,+/-<Rm>]!

内存地址为基地址寄存器值加上/减去 Rm,其加减由 U 的值来确定。当指令执行时,生成的地址值将写入基地址寄存器。

(3) [<Rn>],+/-<Rm>

内存地址为基地址寄存器值,当存储器操作完毕后,将基地址寄存器 Rn 值加上/减去 Rm,将所得到的值写回到基地址寄存器 Rn(更新基地址寄存器),其加减由 U 的值来确定。

注意事项:

- 程序计数器 PC(R15)用作索引寄存器 Rm 时,会产生不可预知的结果。
- 当 Rn 和 Rm 为同一个寄存器时,会产生不可预知的结果。

表 3-5 偏移量为寄存器的指令编码类型对应关系

W	P	汇编语法格式
0	1	[<Rn>,+/-<Rm>]
1	0	[<Rn>],+/-<Rm>
1	1	[<Rn>,+/-<Rm>]!

【例 3-2】

```
LDR R0,[R1,R2]; R0<—[R1+R2]
LDR R0,[R1,-R2]; R0<—[R1-R2]
LDR R0,[R1,R2]!; R0<—[R1+R2],且 R1=R1+R2
LDR R0,[R1],R2; R0<—[R1], R1=R1+R2
```

3. Addressing_mode 中的偏移量通过寄存器移位得到

Addressing_mode 中的偏移量是通过寄存器移位所得到的,这类汇编指令寻址按编码格式可分为以下 3 种形式。

前变址不回写形式:

[<Rn>,+/-<Rm>,<shift>#shift_amount]

前变址回写形式:

[<Rn>,+/-<Rm>,<shift>#shift_amount]!

后变址回写形式:

[<Rn>],+/-<Rm>,<shift>#shift_amount

其指令编码格式如下:

31 28	27 26	25	24	23	22	21	20	19 16	15 12	11 7	6 5	4	3 0
cond	0 1	1	P	U	B	W	L	Rn	Rd	shift_amount	shift	0	Rm

$$P=\begin{cases}1:\text{前变址操作}\\0:\text{后变址操作}\end{cases}$$

$$U=\begin{cases}1:\text{内存地址 address 为基址寄存器 Rn 值加上地址偏移量}\\0:\text{内存地址 address 为基址寄存器 Rn 值减去地址偏移量}\end{cases}$$

$$B=\begin{cases}1：指令访问的是无符号的字节数据\\0：指令访问的是字数据\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

通过寄存器移位得到地址偏移量指令编码对应的汇编语法格式类型如表 3-6 所示。下面介绍各汇编语法格式下的地址计算方法：

(1) [<Rn>,+/-<Rm>,<shift>#shift_amount]

内存地址为基地址寄存器值加上/减去 Rm 通过移位 shift_amount 后所得到的数值，其加减由 U 的值来确定。

(2) [<Rn>,+/-<Rm>,<shift>#shift_amount]!

内存地址为基地址寄存器值加上/减去 Rm 通过移位 shift_amount 后所得到的数值，其加减由 U 的值来确定。当指令执行时，生成的地址值将写入基地址寄存器。

(3) [<Rn>],+/-<Rm>,<shift>#shift_amount

内存地址为基地址寄存器值，当存储器操作完毕后，将基地址寄存器 Rn 值加上/减去 Rm 通过移位 shift_amount 后所得到的数值，将所得到的值写回到基地址寄存器 Rn(更新基地址寄存器)，其加减由 U 的值来确定。

注意事项：

- 程序计数器 PC(R15)用作索引寄存器 Rm 时，会产生不可预知的结果。
- 当 Rn 和 Rm 为同一个寄存器时，会产生不可预知的结果。

表 3-6　偏移量为移位寄存器的指令编码类型对应关系

W	P	汇编语法格式
0	1	[<Rn>,+/-<Rm>,<shift>#shift_amount]
1	0	[<Rn>],+/-<Rm>,<shift>#shift_amount
1	1	[<Rn>,+/-<Rm>,<shift>#shift_amount]!

【例 3-3】

```
LDR    R0,[R1,R2,LSL #2]     ; R0<—[R1+R2*4]
LDR    R0,[R1,R2,LSL #2]!    ; R0<—[R1+R2*4]且 R1=R1+R2*4
LDR    R0,[R1],R2,LSL #2     ; R0<—[R1],R1=R1+R2*4
```

3.3.3　半字、有符号字节寻址

这类指令可用来加载有符号字节、加载有符号半字、加载/存储无符号半字。一般称这类指令为“杂类的 Load/Store 指令”。

Load/Store 指令对半字、有符号字节操作指令编码格式如下：

汇编指令汇编语法格式如下：

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7	6	5	4	3 0
cond	0 0 0	P	U	I	W	L	Rn	Rd	addr_mode	1	S	H	1	addr_mode

加载有符号字节到寄存器：

LDR {<cond>}SB <Rd>,<addressing_mode>

加载有符号半字到寄存器：

LDR {<cond>}SH <Rd>,<addressing_mode>

加载无符号半字到寄存器：

LDR {<cond>}H <Rd>,<addressing_mode>

存储无符号半字到内存：

STR {<cond>}H <Rd>,<addressing_mode>

cond 为指令执行的条件。

$$P=\begin{cases}1：前变址操作\\0：后变址操作\end{cases}$$

$$U=\begin{cases}1：内存地址\ address\ 为基址寄存器\ Rn\ 值加上地址偏移量\\0：内存地址\ address\ 为基址寄存器\ Rn\ 值减去地址偏移量\end{cases}$$

$$I=\begin{cases}1：偏移量为\ 8\ 位立即数\\0：偏移量为寄存器\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

Rn 为基址寄存器，Rd 为源/目标寄存器，Addressing_mode 为内存地址构成格式，对应指令编码中的 S、H 位编码将在 4.3.2 节详细介绍。

下面我们将重点讨论内存地址构成格式 Addressing_mode。根据指令编码和汇编语法格式的不同，归纳起来共有以下两种内存地址构成格式。

1. Addressing_mode 中的偏移量为立即数

Addressing_mode 中的偏移量为立即数的汇编语法格式有以下 3 种：

前变址不回写形式：

[<Rn>,#+/-<immed_offset8>]

前变址回写形式：

[<Rn>,#+/-<immed_offset8>]!

后变址回写形式：

[<Rn>],#+/-<immed_offset8>

其指令编码如下：

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7	6	5	4	3 0
cond	0 0 0	P	U	I	W	L	Rn	Rd	offset_H	1	S	H	1	offset_L

$$P=\begin{cases}1：前变址操作\\0：后变址操作\end{cases}$$

$$U=\begin{cases}1：内存地址\ address\ 为基址寄存器\ Rn\ 值加上地址偏移量\\0：内存地址\ address\ 为基址寄存器\ Rn\ 值减去地址偏移量\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

从指令编码中可以看出,immed_offset8 是一个 8 位的偏移量,这个 8 位的偏移量是由两个 4 位的偏移量(offset_H、offset_L)构成,其中 offset_H 为 8 位偏移量的 bit[7:4]位,offset_L 为 8 位偏移量的 bit[3:0]位。

偏移量为立即数的指令编码对应的汇编语法格式类型如表 3-7 所示。下面介绍各汇编语法格式下的地址计算方法:

(1) [<Rn>,#+/-<immed_offset8>]

内存地址为基地址寄存器值加上/减去 immed_offset8,其加减由 U 的值来确定。

(2) [<Rn>,#+/-<immed_offset8>]!

内存地址为基地址寄存器值加上/减去 immed_offset8,其加减由 U 的值来确定。当指令执行时,生成的地址值将写入基地址寄存器。

(3) [<Rn>],#+/-<immed_offset8>

内存地址为基地址寄存器值,当存储器操作完毕后,将基地址寄存器 Rn 值加上/减去 immed_offset8,将所得到的值写回到基地址寄存器 Rn(更新基地址寄存器),其加减由 U 的值来确定。

表 3-7　偏移量为立即数的指令编码类型

W	P	汇编语法格式
0	1	[<Rn>,#+/-<immed_offset8>]
1	0	[<Rn>],#+/-<immed_offset8>
1	1	[<Rn>,#+/-<immed_offset8>]!

注意事项:

- 程序计数器 PC(R15)用作基址寄存器 Rm 时,会产生不可预知的结果。
- 当 S=1 且 L=0 时,表示带符号数的存储指令,目前还没有实现此功能的 ARM 指令。

【例 3-4】

```
LDRSB R0,[R1,#4]     ; R0<—[R1+4]字节单元,R0 有符号扩展为 32 位
```

```
LDRH R2,[R1,#-4]    ; R2<—[R1-4]半字单元,R2 的高 16 位清零
STRH R0,[R1,#4]!    ; R0 低 16 位—>[R1+4],同时 R1=R1+4
LDRSH R0,[R1],#4    ; R0<—[R1]半字单元且 R0 有符号扩展为 32 位,R1=R1+4
```

2. Addressing_mode 中的偏移量为寄存器的值

Addressing_mode 中的偏移量为寄存器值的汇编语法格式有以下 3 种：

前变址不回写形式：

[<Rn>,+/-<Rm>]

前变址回写形式：

[<Rn>,+/-< Rm >]!

后变址回写形式：

[<Rn>],+/-< Rm >

其指令编码如下：

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7	6	5	4	3 0
cond	0 0 0	P	U	0	W	L	Rn	Rd	全为0	1	S	H	1	Rm

P=1：前变址操作；0：后变址操作

U=1：内存地址 address 为基址寄存器 Rn 值加上地址偏移量；0：内存地址 address 为基址寄存器 Rn 值减去地址偏移量

W=1：执行基地址寄存器回写操作；0：不执行基地址寄存器回写操作

L=1：执行 Load 操作；0：执行 Store 操作

偏移量为寄存器值的指令编码对应的汇编语法格式类型如表 3-8 所示。下面介绍各汇编语法格式下的地址计算方法：

(1) [<Rn>,+/-< Rm >]

内存地址为基地址寄存器值加上/减去 Rm，其加减由 U 的值来确定。

(2) [<Rn>,+/-< Rm >]!

内存地址为基地址寄存器值加上/减去 Rm，其加减由 U 的值来确定。当指令执行时，生成的地址值将写入基地址寄存器。

表 3-8 偏移量为寄存器值的指令编码类型对应关系

W	P	汇编语法格式
0	1	[<Rn>,+/-< Rm >]
1	0	[<Rn>],+/-< Rm >
1	1	[<Rn>,+/-< Rm >]!

(3) [<Rn>],+/-< Rm >

内存地址为基地址寄存器值,当存储器操作完毕后,将基地址寄存器 Rn 值加上/减去 Rm,将所得到的值写回到基地址寄存器 Rn(更新基地址寄存器),其加减由 U 的值来确定。

注意事项:

- 程序计数器 PC(R15)用作基址寄存器 Rn 或索引寄存器 Rm 时,会产生不可预知的结果。
- 基址寄存器 Rn 和索引寄存器 Rm 为同一个寄存器时,会产生不可预知的结果。

【例 3-5】

```
LDRSB R0,[R1,R5]     ; R0<—[R1+R5]字节单元,R0 有符号扩展为 32 位
LDRH R2,[R1,-R5]     ; R2<—[R1-R5]半字单元,R2 的高 16 位清零
STRH R0,[R1,R5]!     ; R0 低 16 位—>[R1+R5],同时 R1=R1+R5
LDRSH R0,[R1],R5     ; R0<—[R1]半字单元且 R0 有符号扩展为 32 位,R1=R1+R5
```

3.4 批量 Load/Store 指令寻址方式

ARM 指令系统提供了批量 Load/Store 指令寻址方式,即通常所说的多寄存器寻址,也就是一次可以传送几个寄存器的值,允许一条指令最多传送 16 个寄存器。

1. 编码格式

批量 Load/Store 指令汇编语法格式如下:

批量加载:

```
LDM {<cond>}<addr_mode> <Rn> {!},<register>{^}
```

批量存储:

```
STM {<cond>}<addr_mode> <Rn> {!},<register>{^}
```

批量 Load/Store 指令编码格式如下:

31 28	27 25	24	23	22	21	20	19 16	15 0
cond	1 0 0	P	U	S	W	L	Rn	register_list

cond 为指令执行的条件。

$$P=\begin{cases}1:\text{前变址操作}\\0:\text{后变址操作}\end{cases}$$

U 表示地址变化的方向:

$$U=\begin{cases}1:\text{地址向上变化(upwards)}\\0:\text{地址向下变化(downwards)}\end{cases}$$

$$W=\begin{cases}1:\text{执行基地址寄存器回写操作}\\0:\text{不执行基地址寄存器回写操作}\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

Rn 为基址寄存器，也就是内存地址块的最低地址值。

register_list 表示要加载或存储的寄存器列表，bit[15:0]可以表示 16 个寄存器，如果某位为 1，则该位的位置作为寄存器的编号，此寄存器参与加载或存储。例如，bit[8]为 1，则代表 R8 参与加载或存储。

S 用于恢复 CPSR 和强制用户位。当程序计数器 PC 包含在 LDM 指令的 register_list 中，且 S 为 1 时，则当前模式的 SPSR 被复制到 CPSR 中，使处理器的程序返回和状态的恢复成为一个原子操作。如果 register_list 中不包含程序计数器 PC，S 为 1 则加载或存储的是用户模式下的寄存器组。

addr_mode 表示地址的变化方式。

注意事项：

指令中寄存器和连续内存地址单元的对应关系：编号低的寄存器对应内存低地址单元，编号高的寄存器对应内存高地址单元。

2. 内存操作

批量 Load/Store 指令在实现寄存器组和连续的内存单元中数据传递时，地址的变化方式(addr_mode)有以下 4 种类型：

- 后增 IA (Increment After)：每次数据传送后地址加 4。
- 先增 IB (Increment Before)：每次数据传送前地址加 4。
- 后减 DA (Decrement After)：每次数据传送后地址减 4。
- 先减 DB (Decrement Before)：每次数据传送前地址减 4。

它们与指令编码中 P、U 的对应关系如表 3-9 所示。

表 3-9 LDM/STM 的地址变化方式

addr_mode	P	U
DA	0	0
IA	0	1
DB	1	0
IB	1	1

3. 堆栈操作

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，在 ARM 里常用 R13 作为栈指针(SP)。

根据堆栈指针的指向位置的不同可以将堆栈分为满堆栈和空堆栈：

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

根据堆栈的生成方式，又可以分为递增堆栈和递减堆栈：

- 递增堆栈(Ascending Stack)：当堆栈由低地址向高地址生成时。
- 递减堆栈(Decending Stack)：当堆栈由高地址向低地址生成时。

四种类型的堆栈工作方式,即

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

栈操作其实也是块拷贝操作,每一条栈操作指令都相应与一条块拷贝操作指令相对应,其对应关系如表 3-10 所示。

表 3-10　块拷贝与栈操作的对应关系

块拷贝 \ 栈操作		地址变化方向			
		向上		向下	
		满	空	满	空
增	先	STMIB STMFA			LDMIB LDMED
	后		STMIA STMEA	LDMIA LDMFD	
减	先		LDMDB LDMEA	STMDB STMFD	
	后	LDMDA LDMFA			STMDA STMED

例如:

```
LDMIA R0,{R5-R8};
```

功能解析:将内存单元[R0]~[R0+15]以字为单位读取到 R5~R8 中,低地址编号的字数据内存单元对应低编号寄存器。

3.5　协处理器指令寻址方式

ARM 支持协处理器操作,其操作要通过协处理器命令来实现,下面讨论协处理指令具体的寻址方式。

1. 协处理器加载/存储指令的寻址方式

协处理器的加载存储指令可以用来实现 ARM 处理器与协处理器之间的数据传输。

其汇编语法格式如下:

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

其中:

opcode 为指令操作码;

coproc 为协处理器名称；

addressing_mode 为指令寻址模式。

根据内存地址的构成方式，可分为索引格式和非索引格式。

1）内存地址索引格式

索引格式类似于 LDR/STR 指令寻址中的立即数作为地址偏移量的形式。

Addressing_mode 中的偏移量为 8 位立即数的汇编语法格式有以下 3 种：

前变址不回写形式：

[<Rn>,#+/-<imm_offset8>*4]

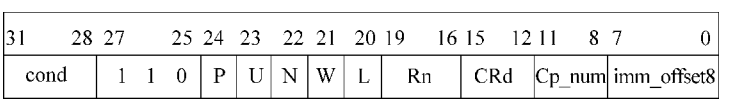
前变址回写形式：

[<Rn>,#+/-<imm_offset8>*4]!

后变址回写形式：

[<Rn>],#+/-<imm_offset8>*4

指令编码格式如下：



cond 为指令执行的条件。

cp_num 为协处理器的编号。

$$P=\begin{cases}1：前变址操作\\0：后变址操作\end{cases}$$

$$U=\begin{cases}1：内存地址\ address\ 为基址寄存器\ Rn\ 值加上地址偏移量\\0：内存地址\ address\ 为基址寄存器\ Rn\ 值减去地址偏移量\end{cases}$$

$$W=\begin{cases}1：执行基地址寄存器回写操作\\0：不执行基地址寄存器回写操作\end{cases}$$

$$L=\begin{cases}1：执行\ Load\ 操作\\0：执行\ Store\ 操作\end{cases}$$

N：依赖于具体的协处理器，一般用来表示传输数据的大小。

CRd 作为目标寄存器的协处理器寄存器。

偏移量为 8 位立即数的指令编码对应的汇编语法格式类型如表 3-11 所示。下面介绍各汇编语法格式下的地址计算方法：

表 3-11　协处理器加载/存储指令编码 W、P 与地址模式的关系

W	P	汇编语法格式
0	1	[<Rn>,#+/-<imm_offset8>*4]
1	0	[<Rn>],#+/-<imm_offset8>*4
1	1	[<Rn>,#+/-<imm_offset8>4*]!

(1) [<Rn>,#+/-<imm_offset8>*4]

第一个内存地址编号为基地址寄存器 Rn 值加上/减去 imm_offset8 的 4 倍(其加减由 U 的值来确定),后续的每一个地址是前一个内存地址加 4,直到协处理器发出信号,结束本次数据传输为止。

(2) [<Rn>,#+/-<imm_offset8>*4]!

内存地址为基地址寄存器 Rn 值加上/减去 imm_offset8 的 4 倍(其加减由 U 的值来确定),后续的每一个地址是前一个内存地址加 4,直到协处理器发出信号,结束本次数据传输为止。当指令执行时,生成的地址值(也就是 Rn 值加上/减去 imm_offset8 的 4 倍)将写入基地址寄存器。

(3) [<Rn>],#+/-<imm_offset8>*4

内存地址为基地址寄存器 Rn 值,当存储器操作完毕后,将基地址寄存器 Rn 值加上/减去 imm_offset8 的 4 倍(其加减由 U 的值来确定),后续的每一个地址是前一个内存地址加 4,直到协处理器发出信号,结束本次数据传输为止。最后将 Rn 值加上/减去 imm_offset8 的 4 倍写回到基地址寄存器 Rn(更新基地址寄存器)。

2) 内存地址非索引格式

这种指令寻址汇编语法格式为:

[<Rn>],<user-define>

指令编码格式如下:

31 28	27 25	24	23	22	21	20	19 16	15 12	11 8	7 0
cond	1 1 0	0	U	N	0	L	Rn	CRd	Cp_num	user_defined

其中:

cond 为指令执行的条件;

Cp_num 为协处理器的编号;

U=1:内存地址 address 为基址寄存器 Rn 值加上地址偏移量
U=0:内存地址 address 为基址寄存器 Rn 值减去地址偏移量

L=1:执行 Load 操作
L=0:执行 Store 操作

N:依赖于具体的协处理器,一般用来表示传输数据的大小;

CRd 作为目标寄存器的协处理器寄存器;

user-define 为用户自定义内容。

这种寻址方式用来产生一段连续的内存地址,第 1 个地址值为基址寄存器<Rn>的值,后续的每一个地址是前一个内存地址加 4,直到协处理器发出信号,结束本次数据传输为止。

注意事项:

- 从索引格式和非索引格式寻址情况来看,它们有一个共同点:数据传输的数目是由协处理器来决定的。
- 在使用中要注意,这两种寻址方式最大可以传输 16 个字数据。

2. 协处理器数据处理指令的寻址方式

协处理器数据处理指令的寻址方式主要通过寄存器寻址，根据寄存器编码来查找相应的寄存器，这部分内容在指令系统中进行详细介绍。

思考与练习题

1. 在指令编码中，条件码占有几位，最多有多少个条件，各个条件是如何形成的？

2. 指令条件码中，V 标志位在什么情况下才能等于 1？

3. 在 ARM 指令中，什么是合法的立即数？判断下面各立即数是否合法，如果合法则写出在指令中的编码格式（也就是 8 位常数和 4 位的移位数）。

0x5430	0x108	0x304	0x501
0xfb10000	0x334000	0x3FC000	0x1FE0000
0x5580000	0x7F800	0x39C000	0x1FE80000

4. 分析逻辑右移、算术右移、循环右移、带扩展的循环右移它们间的差别。

5. ARM 数据处理指令具体的寻址方式有哪些？如果程序计数器 PC 作为目标寄存器，会产生什么结果？

6. 在 Load/Store 指令寻址中，字、无符号字节的 Load/Store 指令寻址和半字、有符号字节寻址，试分析它们之间的差别。

7. 块拷贝 Load/Store 指令在实现寄存器组和连续的内存单元中数据传递时，地址的变化方式有哪几种类型？并分析它们的地址变化情况。

8. 栈操作指令地址的变化方式有哪几种类型？并分析它们的地址变化情况，从而得出栈操作指令寻址和块拷贝 Load/Store 指令之间的对应关系。

9. 分析协处理器加载/存储指令的寻址方式中的内存地址索引格式中不同的汇编语法格式下内存地址的计算方法。

10. 写出下列指令的机器码，并分析指令操作功能。

```
MOV     R0,R1
MOV     R1,#0x198
ADDEQS  R1,R2,#0xAB
CMP     R2,#0Xab
LDR     R0,[R1,#4]
STR     R0,[R1,R1,LSL  #2]!
LDRH    R0,[R1,#4]
LDRSB   R0,[R2,#-2]!
STRB    R1,[R2,#0xA0]
LDMIA   R0,{R1,R2,R8}
STMDB   R0!,{R1-R5,R10,R11}
STMED   SP!,{R0-R3,LR}
```