第 3 章　TMS320C54x 的指令系统

TMS320C54x 的指令由操作码和操作数两部分组成。在汇编前，操作码和操作数都是用助记符表示的。例如：

```
LD   #0FFH,A
```

的执行结果是将立即数 0FFH 传送至累加器 A 中。

3.1　寻址方式

TMS320C54x 提供了 7 种基本的数据寻址方式。

3.1.1　立即数寻址

立即数寻址指令中有一个固定的立即数。在一条指令中可有两种立即数：一种是短立即数（3、5、8 或 9 位），另一种是 16 位的长立即数。立即数可以包含在单字或双字指令中，3、5、8 或 9 位的短立即数包含在单字指令中，16 位的立即数包含在双字指令中。在一条指令中，立即数的长度是由所使用的指令的类型决定的。表 3-1 中列出了可以包含立即数的各条指令，并且指出了该指令中立即数的位数。

表 3-1　支持立即数寻址的指令

3 或 5 位立即数	8 位立即数	9 位立即数	16 位立即数
LD	FRAME，LD，RPT	LD	ADD，ORM，ADDM，RPT，AND，RPTZ，ANDM，ST，BITF，TM，CMPM，SUB，LD，XOR，MAC，XORM，OR

立即数寻址指令的语法中有一点需要注意，应在数值或符号前面加一个“#”号，表示这是一个立即数，否则会被认为是一个地址。例如，把立即数 0F0H 装入累加器 A，其正确的指令为：

```
LD   #0F0H,A
```

如果漏掉了“#”号，指令“LD　0F0H,A”就变成了把地址为 0F0H 的单元中的数据装入累加器 A 中。另外，如果在#0F0H 中不加第一个 0，编译器将提示错误信息。对于 9FFFH 以内的十六进制数，其前面必须加 0，以表明是立即数。

下面两个例子用 RPT 指令来说明立即数寻址。

【例 3-1】

```
RPT   #99              ;将紧跟 RPT 的下一条指令循环执行 100 次
```

在这个例子中，操作数是短立即数，与操作码在同一个字中。

【例 3-2】

```
RPT   #0FFFFH          ;将紧跟 RPT 的下一条指令循环执行 10000H 次
```

操作数是 16 位长立即数的双字指令，操作码占一个字，操作数紧跟其后也占一个字。

3.1.2 绝对地址寻址

绝对地址寻址指令中有一个固定的地址，指令按照此地址进行数据寻址。绝对地址的代码为16位，所以包含绝对地址寻址的指令至少两个字长。绝对地址寻址有以下4种类型。

1．数据存储器地址寻址

数据存储器地址（dmad）寻址就是用程序标号或数据来确定指令所需的数据存储器的地址。例如，把DATA1标注的数据存储器地址里的数据复制到由AR2指向的数据单元中：

```
MVKD   DATA1,*AR2
```

DATA1标注的地址就是一个dmad值。执行此指令的作用是将数据单元中的内容两两移动。应当注意的是，DATA1必须是程序中的标号，或者是一些DSP芯片内部已经定义的单元。例如，将串口的接收数据送到指定单元中：

```
MVDD   DRR20,*AR2
```

这里，DRR20是内部已经定义的单元，其地址已经固定在存储器中。

2．程序存储器地址寻址

程序存储器地址（pmad）寻址就是用一个符号或一个具体的数来确定程序存储器的地址。例如，把TABLE1标注的程序单元中的一个字复制到AR2所指向的数据单元中：

```
MVPD   TABLE1,*AR2
```

TABLE1所标注的地址就是一个pmad值。程序存储器地址寻址基本上和数据存储器地址寻址一样，区别仅在于空间不同。

3．端口寻址

端口（PA）寻址就是用一个符号或一个常数来确定外部I/O口地址。例如，把一个数从端口地址为F2F0的I/O口复制到AR5指向的数据单元中：

```
PORTR   F2F0,*AR5
```

F2F0指的是端口地址。实际上，端口地址寻址只涉及两条指令，即端口读（PORTR）指令和端口写（PORTW）指令。

对于DSP外部的存储空间，也只有两条特定的指令WRITA和READA可以用于对其进行读或写，具体用法参见累加器寻址方式。

4．*(lk)寻址

*(lk)寻址就是用一个符号或一个常数来确定数据存储器的地址。例如，把地址为BUFFER的数据单元中的数据装载到累加器A中：

```
LD   *(BUFFER),A
```

(lk)寻址的语法允许所有使用单数据存储器操作数（Smem）寻址的指令访问数据存储器中的任意单元而不改变数据页指针（DP）的值，也不用对AR进行初始化。当采用绝对地址寻址方式时，指令长度将在原来的基础上增加一个字。值得注意的是，使用(lk)寻址方式的指令不能与循环指令（RPT，RPTZ）一起使用。

3.1.3 累加器寻址

累加器寻址方式将累加器内的当前值作为地址去访问该程序单元。累加器寻址用累加器中的数作为一个地址，这种寻址方式可用来对存放数据的程序存储器寻址。只有两条指令（READA和

WRITA）可以采用累加器寻址方式。

READA 指令把累加器 A 所确定的程序单元中的一个字，传送到操作数 Smem 所确定的数据单元中。WRITA 指令把操作数 Smem 所确定的数据单元中的一个字，传送到累加器 A 所确定的程序单元中。

3.1.4 直接寻址

在直接寻址中，指令包含数据存储器地址的低 7 位。这 7 位作为偏移地址与 DP 或 SP 相结合，共同形成 16 位的数据存储器实际地址。虽然直接寻址不是偏移寻址的唯一方式，但这种方式的优点是每条指令只有一个字。直接寻址的语法格式中，用一个符号或一个常数来确定偏移值。例如：

```
ADD   SAMPLE,A          ;把数据单元 SAMPLE 中的内容加到 A 中
或    ADD   @x,A        ;将符号@加在变量 x 的前面
```

地址 SAMPLE 的低 7 位放在指令中。图 3-1 所示为直接寻址指令的格式。表 3-2 中给出了直接寻址指令各位的说明。

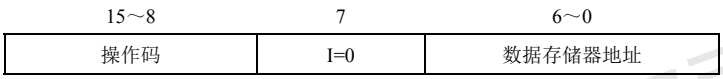

图 3-1　直接寻址指令的格式

表 3-2　直接寻址指令各位的说明

位	名　称	功　能
15～8	操作码	这 8 位包含指令的操作码
7	I=0	I=0，表示指令的寻址方式为直接寻址方式
6～0	数据存储器地址	这 7 位包含指令的数据存储器地址偏移

图 3-2 所示为直接寻址的方框图，DP 和 SP 都可以与 dmad（图中指令寄存器 IR 的低 7 位）相结合产生实际地址，位于 ST1 寄存器中的 CPL 位决定了选择哪种指针来产生实际地址。

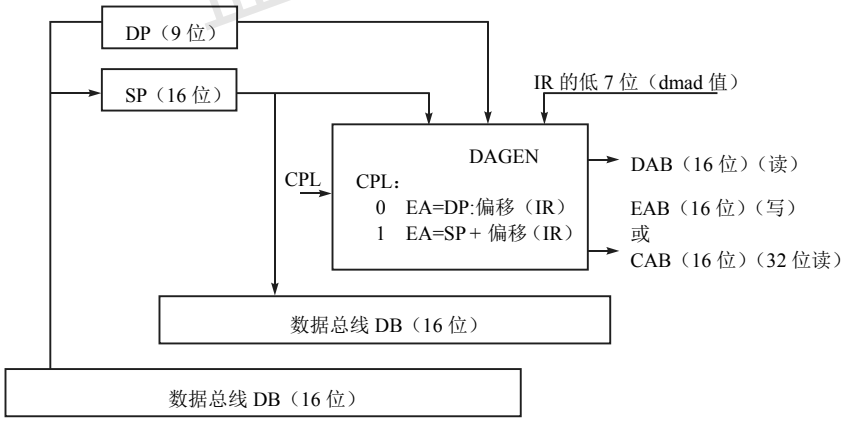

图 3-2　直接寻址的方框图

CPL=0，表示选择以 DP 为基准的直接寻址编译方式，将 dmad 值与 9 位的 DP 值相结合，形成 16 位的数据存储器地址。

IR 中的低 7 位（dmad 值）与 9 位的 DP 值连接在一起形成实际地址，如图 3-3 所示。

15～7	6～0
DP 值	IR 的低 7 位（dmad 值）

图 3-3　以 DP 为基准的直接寻址编译方式

因为 DP 值的范围为 0～511（$1～2^9$），所以以 DP 为基准的直接寻址编译方式把存储器分成 512 页。7 位 dmad 值的变化范围为 0～127，每页有 128 个可访问的单元。换言之，DP 指向 512 页中的一页，dmad 就指向了该页中的特定单元。访问第 1 页的单元 0 和访问第 2 页的单元 0 的唯一区别是 DP 值的变化。DP 值可由 LD 指令装入。RESET 指令将 DP 值赋为 0。注意：DP 值不能用上电进行初始化，在上电后它处于不定状态。所以，没有初始化 DP 值的程序可能工作不正常，所有的程序都必须对 DP 值进行初始化。

```
【例 3-3】LD      #x,DP       ;立即数 x 送 ST0 的 DP 位
          LD      @u,A        ;x 页 u 单元中的内容装送入 A
          ADD     @v,A        ;x 页 v 单元中的内容与 A 相加
```

CPL=1，表示选择以 SP 为基准的直接寻址编译方式，将 dmad 值加上（正偏移）SP 值，形成 16 位的数据存储器地址。

IR 的低 7 位（dmad 值）作为一个正偏移与 SP 值相加得到实际地址，如图 3-4 所示。

图 3-4　以 SP 为基准的直接寻址编译方式

SP 可以指向存储器中的任意一个地址。dmad 可以指向当前页中一个确定的单元，从而允许访问存储器任意基地址中连续的 128 个字。

```
【例 3-4】SSBX    CPL         ;对 ST1 寄存器的 CPL 置位, CPL=1
          LD      @X1, A      ;SP 值加 X1 所形成的地址中的内容送入 A
          ADD     @Y2, A      ;SP 值加 Y2 所形成的地址中的内容与 A 相加
```

由于 DP 与 SP 两种直接寻址编译方式是互相排斥的，当采用 SP 直接寻址编译方式后再用 DP 直接寻址编译方式之前，必须使用 RSBX 指令对 CPL 清 0。

3.1.5　间接寻址

间接寻址方式按照辅助寄存器中的地址访问存储器。在间接寻址中，64KW×16bit 数据存储器任意单元都可通过一个辅助寄存器中的 16 位地址进行访问。TMS320C54x 有 8 个 16 位辅助寄存器（AR0～AR7），两个辅助寄存器运算单元（ARAU0 和 ARAU1），可以根据辅助寄存器的内容进行操作，完成无符号的 16 位算术运算。

间接寻址方式很灵活，不仅能从存储器中读或写一个 16 位单数据存储器操作数，而且能在一条指令中访问两个数据单元（从两个独立的单元中读数据，或在读一个单元的同时写另一个单元，或读/写两个连续的单元）。

1．单数据存储器操作数间接寻址

单数据存储器操作数间接寻址指令的格式如图 3-5 所示，其各位说明如表 3-3 所示。

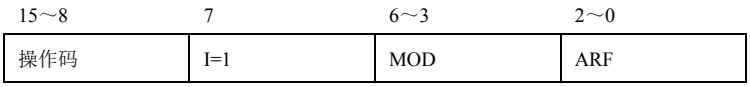

图 3-5　单数据存储器操作数间接寻址指令的格式

表 3-3　单数据存储器操作数间接寻址指令的各位说明

位	名　称	功　能
15～8	操作码	8 位域，包含指令的操作码
7	I=1	I=1，表示指令的寻址方式为间接寻址方式
6～3	MOD	4 位的方式域，定义间接寻址的类型。表 3-4 将详细说明 MOD 域定义的各种类型
2～0	ARF	3 位辅助寄存器定义寻址所使用的辅助寄存器，ARF 由 ST1 寄存器中的 CMPT 决定： CMPT=0，标准方式。若 ARF=0，则确定辅助寄存器，而不管 ARP 的值。在这种方式下，ARP 不能被修改，必须一直设为 0。 CMPT=1，兼容方式。若 ARF=0，则用 ARP 来选择辅助寄存器；否则，用 ARF 来选择，且当访问完成后，会把 ARF 装入 ARP。汇编指令中的*AR0 表示 ARP 所选择的辅助寄存器

下面介绍表 3-4 中所用到的循环寻址和位倒序寻址。

表 3-4　单数据存储器操作数的间接寻址类型

MOD 域	操作码语法	功　能	说　明
0000	*ARx	addr=ARx	ARx 包含数据存储器地址
0001	*ARx−	addr=ARx，ARx=ARx−1	访问后，ARx 中的地址减 1
0010	*ARx+	addr=ARx，ARx=ARx+1	访问后，ARx 中的地址加 1
0011	*+ARx	addr=ARx+1，ARx=ARx+1	在寻址之前，ARx 中的地址加 1
0100	*AR−0B	addr=ARx，ARx=B(ARx−AR0)	访问后，从 ARx 中以位倒序进位的方式减去 AR0
0101	*ARx−0	addr=ARx，ARx=ARx−AR0	访问后，从 ARx 中减去 AR0
0110	*ARx+0	addr=ARx，ARx=ARx+AR0	访问后，把 AR0 加到 ARx 中
0111	*ARx+0B	addr=ARx，ARx=B(ARx+AR0)	访问后，把 AR0 以位倒序进位的方式加到 ARx 中
1000	*ARx−%	addr=ARx，ARx=circ(ARx−1)	访问后，ARx 中的地址以循环寻址的方式减 1
1001	*ARx−0%	addr=ARx，ARx=circ(ARx−AR0)	访问后，ARx 中的地址以循环寻址的方式减 AR0
1010	*ARx+%	addr=ARx，ARx=circ(ARx+1)	访问后，ARx 中的地址以循环寻址的方式加 1
1011	*ARx+0%	addr=ARx，ARx=circ(ARx+AR0)	访问后，把 AR0 以循环寻址的方式加到 ARx 中
1100	*ARx+(lk)	addr=ARx+lk，ARx=ARx	ARx 加上 16 位的长偏移（lk）的和作为数据存储器的地址，ARx 本身不被修改
1101	*+ARx(lk)	addr=ARx+lk，ARx=ARx+lk	在寻址之前，把一个带符号的 16 位的长偏移加到 ARx 中，再用新的 ARx 值作为数据存储器的地址
1110	*+ARx(lk)%	addr=circ(ARx+lk) ARx=circ(ARx+lk)	在寻址之前，把一个带符号的 16 位的长偏移以循环寻址的方式加到 ARx 中，再用新的 ARx 值作为数据存储器的地址
1111	*(lk)	addr=lk	一个无符号的 16 位的长偏移作为数据存储器的绝对地址（也属于绝对地址）

2．循环寻址

在卷积、相关和 FIR 滤波等许多算法中，都需要在存储器中实现一个循环缓冲区。一个循环缓冲区就是一个包含最近数据的滑动窗口。当新的数据到来时，就会覆盖最早的数据。循环缓冲区实现的关键是循环寻址的实现。长度为 R 的循环缓冲区必须从一个 N 位（N 是满足 $2^N>R$ 的最小整数）边界开始，也就是说，循环缓冲区基地址的最低 N 位必须为 0。R 的值必须装入循环缓冲区长度寄存器（BK）中。例如，含有 31 个字的循环缓冲区必须从最低 5 位为 0 的地址开始（xxxx xxxx xxx0 0000_2），且 31 这个值必须装入 BK 中。又如，含有 32 个字的循环缓冲区必须从最低 6 位为 0 的地址开始（即 xxxx xxxx xx00 0000_2）。

循环缓冲区的有效基地址（EFB）就是用户选定的辅助寄存器（ARx）的低 N 位清 0 后所得到的值。循环缓冲区的尾地址（EOB）是通过用 BK 的低 N 位代替 ARx 的低 N 位得到的。循环缓冲区的 INDEX 就是 ARx 的低 N 位，step 就是加到辅助寄存器中的值或从辅助寄存器中减去的值。循环寻址的算法描述为：

```
if 0≤index+step<BK:
    index=index+step
        else if index+step≥BK:
            index=index+step−BK
        else if index+step<0
            index=index+step+BK
```

图 3-6 所示为循环寻址的方框图，其中给出了 BK、ARx，以及循环缓冲区的 EOB、EFB 和 INDEX。对于一个需要 8 个循环缓冲单元的运算，循环指针在第一次的移动顺序是 1，2，3，4，5，6，7→8，第二次的移动顺序是 2，3，4，5，6，7→8→1，第三次的移动顺序是 3，4，5，6，7→8→1→2，依次循环，直到完成规定的循环次数。

3．位倒序寻址

位倒序寻址提高了指令执行速度和在 FFT 算法的程序中使用存储器的效率。在这种寻址方式中，AR0 存放的整数 N 是 FFT 点数的一半，一个辅助寄存器指向一个数据存放的物理单元。当使用位倒序寻址方式把 AR0 加到辅助寄存器中时，地址以位倒序的方式产生，即进位是从左向右的，而不是从右向左的。例如，以下为 0110 与 1100 以位倒序的方式相加：

$$\begin{array}{cccccc} & 0 & 1 & 1 & 0 \\ + & 1 & 1 & 0 & 0 \\ \hline & 1 & 0 & 0 & 1 \end{array}$$

假设辅助寄存器是 8 位的，AR2 表示存储器中数据的基地址 0110 0000，AR0 的值为 0000 1000，下面给出在位倒序寻址方式中，AR2 值的修改顺序和修改后的值：

```
*AR2+0B      ;AR2=01100000   (第 0 值)
*AR2+0B      ;AR2=01101000   (第 1 值)
*AR2+0B      ;AR2=01100100   (第 2 值)
*AR2+0B      ;AR2=01101100   (第 3 值)
*AR2+0B      ;AR2=01100010   (第 4 值)
*AR2+0B      ;AR2=01101010   (第 5 值)
*AR2+0B      ;AR2=01100110   (第 6 值)
*AR2+0B      ;AR2=01101110   (第 7 值)
```

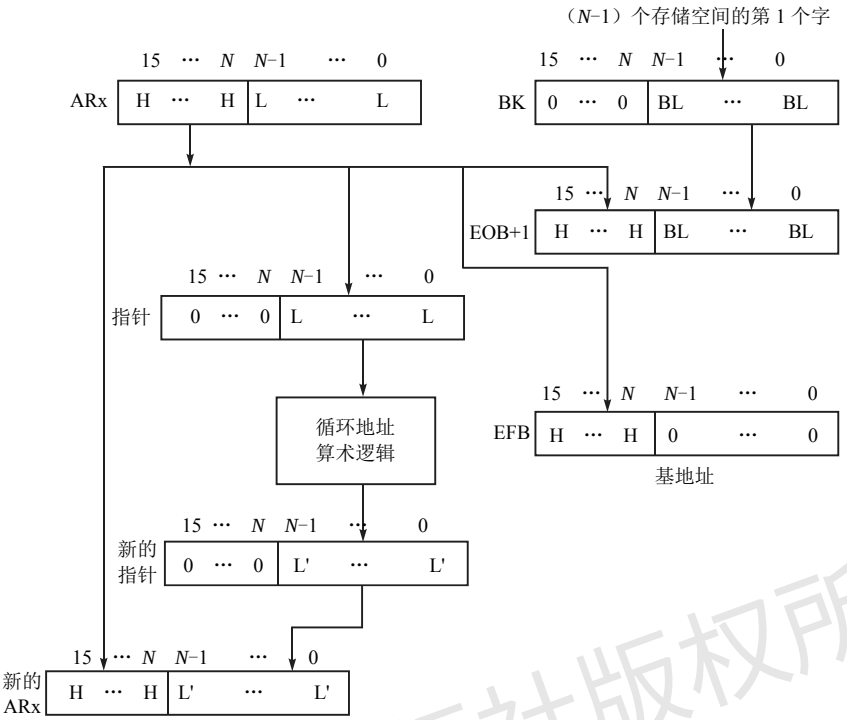

图 3-6　循环寻址的方框图

开始时，AR2 的值是 0110 0000，第一次寻址在正常方式下应该是 0110 0001，即在 0110 0000 的基础上加 1，但位倒序方式不是加 1，而是加上 AR0 的值，也就是：

(0110 0000) + (0000 1000)

所以结果就是 0110 1000。第二次寻址就是计算：

(0110 1000) + (0000 1000)

结果就是 0110 0100。依次计算其他位。应当注意的是，这些计算都采用从左到右的顺序。

表 3-5 给出了位倒序结果。

表 3-5　位倒序结果

原　　序	索引步长的位模式	AR2 低 4 位	位倒序结果
0	0000	0000	0
1	0001	1000	8
2	0010	0100	4
3	0011	1100	12
4	0100	0010	2
5	0101	1010	10
6	0110	0110	6
7	0111	1110	14
8	1000	0001	1
9	1001	1001	9
10	1010	0101	5
11	1011	1101	13

续表

原　　序	索引步长的位模式	AR2 低 4 位	位倒序结果
12	1100	0011	3
13	1101	1011	11
14	1110	0111	7
15	1111	1111	15

4．双数据存储器操作数寻址

双数据存储器操作数寻址方式用在完成两个读，或一个读同时并行一个写存储的指令中。这些指令只有一个字长，且只能以间接寻址方式工作。用 Xmem 和 Ymem 代表这两个数据存储器操作数，Xmem 表示读操作数，Ymem 在读两个操作数时表示另一个读操作数，而在一个读同时并行一个写的指令中表示写操作数。如果源操作数和目的操作数指向同一个单元，则在并行存储指令中（如 ST||LD），读在写之前。如果一个双操作数指令（如 ADD）指向同一个辅助寄存器，且这两个操作数的寻址方式不同，就用 Xmod 所确定的方式来寻址。双数据存储器操作数间接寻址指令的格式如图 3-7 所示，其各位说明见表 3-6。

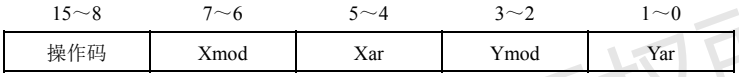

图 3-7　双数据存储器操作数间接寻址指令的格式

表 3-6　双数据存储器操作数间接寻址指令的各位说明

位	名称	功　　能
15～8	操作码	这 8 位包含指令的操作码
7～6	Xmod	定义用于访问 Xmem 的间接寻址方式的类型
5～4	Xar	这两位确定用于访问 Xmem 地址的辅助寄存器
3～2	Ymod	定义用于访问 Ymem 的间接寻址方式的类型
1～0	Yar	这两位确定包含 Ymem 的辅助寄存器

表 3-7 中列出了由指令的 Xar 和 Yar 域选择的辅助寄存器。

表 3-7　由指令的 Xar 和 Yar 域选择的辅助寄存器

Xar,Yar	辅助寄存器
00	AR2
01	AR3
10	AR4
11	AR5

表 3-8 中列出了双数据存储器操作数间接寻址的类型。

表 3-8　双数据存储器操作数间接寻址的类型

Xmod,Ymod	操作码语法	功　　能	说　　明
00	*ARx	addr=ARx	ARx 是数据存储器地址
01	*ARx−	addr=ARx，ARx=ARx−1	访问后，ARx 中的地址减 1
10	*ARx+	addr=ARx，ARx=ARx+1	访问后，ARx 中的地址加 1
11	*ARx+0%	addr=ARx，ARx=circ(ARx+AR0)	访问后，AR0 以循环寻址的方式加到 ARx 中

3.1.6 存储器映射寄存器寻址

存储器映射寄存器寻址方式用来修改存储器映射寄存器而不影响当前 DP 或 SP 的值，以存储器映射寄存器中的修改值去寻址。因为 DP 和 SP 的值不需要改变，因此写一个寄存器的开销是最小的。存储器映射寄存器寻址既可以在直接寻址中使用，又可以在间接寻址中使用。在直接寻址方式下，让数据存储器地址的高 9 位清 0，而不管 DP 或 SP 的值；在间接寻址方式下，只用当前辅助寄存器的低 7 位。例如，AR1 用来指向一个存储器映射寄存器，包含的值为 FF25H。既然 AR1 的低 7 位是 25H，且 PRD 的地址为 0025H，那么 AR1 指向定时周期寄存器。执行后，存放在 AR1 中的值为 0025H。存储器映射寄存器寻址如图 3-8 所示。

只有以下 8 条指令可以使用存储器映射寄存器寻址：

```
LDM     MMR, dst
MVDM    dmad, MMR
MVMD    MMR, dmad
MVMM    MMRx, MMRy
POPM    MMR
PSHM    MMR
STLM    src, MMR
STM     #lk, MMR
```

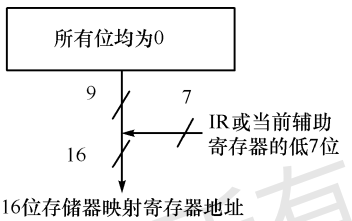

图 3-8　存储器映射寄存器寻址

但是，对于*ARx(lk)，*+ARx(lk)，*+ARx(lk)%，*(lk)等所表示的存储器映射寄存器不能使用该寻址方式，否则，编译时汇编器会给出警告性错误。

3.1.7 堆栈寻址

堆栈寻址方式把数据压入和弹出堆栈，按照后进先出的原则进行寻址。系统堆栈用来在中断和调用子程序期间自动存放程序计数器（PC）值，也能存放额外的数据项或传递数据值。处理器使用一个 16 位的存储器映射寄存器的一个 SP 对堆栈寻址，它总是指向存放在堆栈中的最后一个元素。

以下 4 条指令可以使用堆栈寻址方式访问堆栈。

PSHD——把数据存储器的一个值压入堆栈。

PSHM——把存储器映射寄存器中的一个值压入堆栈。

POPD——把数据存储器的一个值弹出堆栈。

POPM——把存储器映射寄存器中的一个值弹出堆栈。

图 3-9 说明了堆栈操作对 SP 的影响。

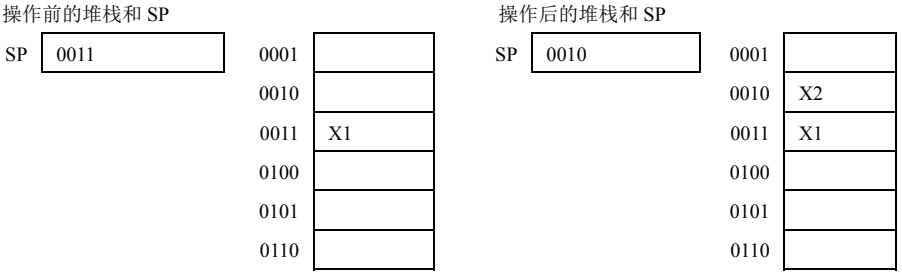

图 3-9　堆栈操作对 SP 的影响

有很多操作或指令都会影响堆栈里的内容，在中断和调用子程序时，堆栈保存当前的 PC 值。

影响堆栈的指令包括 CALA[D]，CALL[D]，CC[D]，INTR，TRAP，RET[D]，RETE[D]，RETEF[D]，RC[D]，FRAME 等。

3.2 指令系统

TMS320C54x 可以使用助记符方式和代数指令（表达式）方式两套指令系统。

3.2.1 符号与意义

表 3-9 中列出了 TMS320C54x 指令系统的符号和意义。

表 3-9 TMS320C54x 指令系统的符号与意义

符　号	意　义	符　号	意　义
A	累加器 A	n	紧跟 XC 指令的字数，n=1 或 2
ACC	累加器	N	指定在 RSBX、SSBX 和 XC 指令中修改的状态寄存器：N=0，修改 ST0；N=1，修改 ST1
ACCA	累加器 A	OVA	ST0 中的累加器 A 的溢出标志位
ACCB	累加器 B	OVB	ST0 中的累加器 B 的溢出标志位
ALU	算术逻辑单元	OVdst	目的累加器（A 或 B）的溢出标志位
ARx	特指某个辅助寄存器（AR0~AR7）	OVdst_	目的累加器反（A 或 B）的溢出标志位
ARP	ST0 中的辅助寄存器指针位，这 3 位指向当前辅助寄存器（AR）	OVsrc	源累加器（A 或 B）的溢出标志位
ASM	ST1 中的 5 位累加器移位方式位（−16≤ASM≤15）	OVM	ST1 中的溢出方式位
B	累加器 B	PA	用 16 位立即数表示的端口地址（0≤PA≤65535）
BRAF	ST1 中的块循环有效标志位	PAR	程序地址寄存器
BRC	块循环计数器	PC	程序计数器
BITC	指定测试位	pmad	用 16 位立即数表示的程序存储器地址（0≤pmad≤65535）
C16	ST1 中的双 16 位/双精度算术运算方式位	Pmem	程序存储器操作数
C	ST0 中的进位标志位	PMST	处理器工作方式状态
CC	两位条件代码（0≤CC≤3）	prog	程序存储器操作数
CMPT	ST1 中的间接寻址辅助寄存器修正方式位	[R]	凑整选项
CPL	ST1 中的直接寻址编译方式位	md	凑整
cond	用操作数表示条件执行指令使用的条件	RC	循环计数器
[d],[D]	延迟方式	RTN	在指令 RETF[D]中使用的快速返回寄存器
DAB	D 地址总线	REA	块循环结束地址
DAR	DAB 地址寄存器	RSA	块循环起始地址
dmad	用 16 位立即数表示的数据存储器地址（0≤dmad≤65535）	SBIT	用 4 位数指明在指令 RSBX、SSBX 和 XC 中修改的状态寄存器位数（0≤SBIT≤15）
Dmem	数据存储器操作数	SHFT	4 位移位数（0≤SHFT≤15）
DP	ST0 中的 9 位数据页指针（0≤DP≤511）	SHIFT	5 位移位数（−16≤SHIFT≤15）

续表

符　号	意　义	符　号	意　义
dst	目的累加器（A 或 B）	Sind	使用间接寻址的单数据存储器操作数
dst_	另一个目的累加器 if dst=A,then dst_=B if dst=B,then dst_=A	Smem	16 位单数据存储器操作数
EAB	E 地址总线	SP	堆栈指针
EAR	EAB 地址寄存器	src	源累加器（A 或 B）
extpmad	用 23 位立即数表示的程序存储器地址	ST0 ST1	状态寄存器 0 状态寄存器 1
FRCT	ST1 中的小数方式位	SXM	ST1 中的符号位扩展方式位
Hi(A)	累加器 A 的高阶位（31～16 位）	T	暂存器
HM	ST1 中的保持方式	TC	ST0 中的测试/控制标志位
IFR	中断标志寄存器	TOS	堆栈栈顶
INTM	ST1 中的中断方式位	TRN	状态转移寄存器
K	少于 9 位的短立即数	TS	由 T 中 0～15 位确定的移位数（−16≤TS≤31）
K3	3 位立即数（0≤K3≤7）	uns	无符号的数
K5	5 位立即数（−16≤K5≤15）	XF	ST1 中的外部标志状态位
K9	9 位立即数（0≤K9≤511）	XPC	程序计数器扩展寄存器
lk	16 位长立即数	Xmem	在双操作数指令和一些单操作数指令中，使用的 16 位双数据存储器操作数
Lmem	长字寻址的 32 位单数据存储器操作数	Ymem	在双操作数指令中使用的 16 位双数据存储器操作数
mmr MMR	存储器映射寄存器		
MMRx MMRy	存储器映射寄存器，AR0～AR7 或 SP		

3.2.2　TMS320C54x 指令

TMS320C54x 指令一共有 129 条，按功能分为算术运算指令、逻辑指令、程序控制指令、装入和存储指令、单个循环指令 5 类。

1．算术运算指令

算术运算指令包括加法指令、减法指令、乘法指令、乘加指令、乘减指令、双操作数指令和特殊操作指令。其中大部分指令只需要一个指令周期，只有个别指令需要 2～3 个指令周期。

（1）加法指令

加法指令共有 13 条，见表 3-10。

表 3-10　加法指令

助记符指令	代数指令	注　释
ADD　Smem,src	src=src+Smem	操作数 Smem 加到 ACC 中
ADD　Smem,TS,src	src=src+Smem<<TS	操作数 Smem 移位后，加到 ACC 中
ADD　Smem,16,src[,dst]	dst=src+Smem<<16	操作数 Smem 左移 16 位后，加到 ACC 中
ADD　Smem,[,SHIFT],src [,dst]	dst=src+Smem<<SHIFT	操作数 Smem 移位后，加到 ACC 中

续表

助记符指令	代数指令	注　　释
ADD　Xmem,SHFT,src	dst=src+Xmem<<SHFT	操作数 Xmem 移位后，加到 ACC 中
ADD　Xmem,Ymem,dst	dst=Xmem<<16+Ymem <<16	操作数 Xmem 和 Ymem 分别左移 16 位后，加到 ACC 中
ADD　#lk[,SHFT],src[,dst]	dst=src+ #lk<<SHFT	长立即数移位后，加到 ACC 中
ADD　#lk,16,src[,dst]	dst=src+ #lk<<16	长立即数左移 16 位后，加到 ACC 中
ADD　src,[,SHIFT][,dst]	dst=dst+src<<SHIFT	ACC 移位后，相加
ADD　src,ASM[,dst]	dst=dst+src<<ASM	ACC 按 ASM 移位后，相加
ADDC　Smem,src	src=src+Smem+C	操作数 Smem 带进位加到 ACC 中
ADDM　#lk,Smem	Smem=Smem+#lk	长立即数加到数据存储器中
ADDS　Smem,src	src=src+uns(Smem)	符号位不扩展的加法

DSP 表示整数时，包括有符号数和无符号数两种格式。作为有符号数表示时，其最高位表示符号，最高位为 0 表示其为正数，为 1 表示其为负数，最低位表示 1，次低位表示 2 的 1 次方，次高位表示 2 的 14 次方。作为无符号数表示时，最高位仍然作为数值位计算。例如，有符号数所能够表示的最大的正数为 07FFFH，等于十进制数 32767，而 0FFFFH 表示最大的负数−1。无符号数不能表示负数，它能够表示的最大的数为 0FFFFH，等于十进制数的 65535。

DSP 表示小数时，其符号和上面整数的表示一样，但是必须注意如何安排小数点的位置。原则上，小数点的位置可以根据程序员的爱好进行安排。为了便于数据处理，一般安排在最高位后（以下仅以小数点在最高位后的情况进行讨论），最高位表示符号位，次高位表示 0.5，然后是 0.25，依次减小一半。例如，4000H 表示小数 0.5，1000H 表示小数 0.25，而 0001H 表示 16 位定点 DSP 表示的最小的小数（有符号）0.000 030 517 578 125。

TMS320C54x 中提供了多条用于加法的指令，如 ADD、ADDC、ADDM 和 ADDS。其中，ADDS 用于无符号数的加法，ADDC 用于带进位的加法（如 32 位扩展精度加法），而 ADDM 专用于立即数的加法。

使用 ADD 指令完成加法：

```
LD    TEMP1, A          ;将变量 TEMP1 装入 A
ADD   TEMP2, A          ;将变量 TEMP2 与 A 相加, 结果放入 A 中
STL   A, TEMP3          ;将结果(低 16 位)存入变量 TEMP3
```

注意，这里完成计算 TEMP3=TEMP1+TEMP2，没有特意考虑 TEMP1 和 TEMP2 是整数还是小数。在加法和下面的减法中，整数运算和定点的小数运算都是一样的。

利用 ADDS 指令实现 32 位数据装入：

```
LD      #0, DP          ;设置数据页指针
LD      60H, 16, A      ;将 60H 的内容装入 A 的高 16 位
ADDS    61H, A          ;将 61H 的内容加到 A 的低 16 位中
DLD     60H, B          ;直接装入 32 位到 B 中
```

（2）减法指令

减法指令共有 13 条，见表 3-11。

TMS320C54x 中提供了多条用于减法的指令，如 SUB、SUBB、SUBC 和 SUBS。其中，SUBS 用于无符号数的减法，SUBB 用于带借位的减法（如 32 位扩展精度的减法），而 SUBC 为移位减，

DSP 中的除法就是用该指令来实现的。SUB 指令与 ADD 指令一样，有多种寻址方式。例如：

```
STM    #60H, AR3          ;将变量 TEMP1 的地址装入 AR3
STM    #61H, AR2          ;将变量 TEMP3 的地址装入 AR2
SUB    *AR2+, *AR3, B     ;将变量 TEMP3 左移 16 位, 同时变量 TEMP2 也左移 16 位
                          ;然后相减, 结果放入 B(高 16 位), 同时 AR2 加 1
STH    B, 63H             ;将相减的结果(高 16 位)存入变量 63H
```

表 3-11　减法指令

助记符指令	代数指令	注　　释
SUB　Smem,src	src=src−Smem	从 ACC 中减去操作数 Smem
SUB　Smem,TS,src	src=src−Smem<<TS	从 ACC 中减去移位后的操作数 Smem
SUB　Smem,16,src[,dst]	dst=src−Smem<<16	从 ACC 中减去左移 16 位后的操作数 Smem
SUB　Smem[,SHIFT],src[,dst]	dst=src−Smem<<SHIFT	操作数 Smem 移位后，与 ACC 相减
SUB　Smem,SHFT,src	dst=src−Smem<<SHFT	操作数 Smem 移位后，与 ACC 相减
SUB　Xmem,Ymem,dst	dst=Xmem<<16−Ymem<<16	操作数 Xmem 和 Ymem 分别左移 16 位后，相减
SUB　#lk[,SHFT],src[,dst]	dst=src− #lk <<SHFT	长立即数移位后，与 ACC 相减
SUB　#lk,16,src[,dst]	dst=src− #lk <<16	长立即数左移 16 位后，与 ACC 相减
SUB　src[,SHIFT][,dst]	dst=dst−src <<SHIFT	源 ACC 移位后，与目的 ACC 相减
SUB　src,ASM[,dst]	dst=dst−src <<ASM	源 ACC 按 ASM 移位后，与目的 ACC 相减
SUBB　Smem,src	src=src−Smem−C	从 ACC 中带借位减操作数 Smem
SUBC　Smem,src	if (src−Smem<<15)→0 src=(src−Smem<<15)<<1+1 else src=src<<1	有条件减法
SUBS　Smem,src	src=src−uns(Smem)	符号位不扩展的减法

在 TMS320C54x 中没有提供专门的除法指令。一般有两种方法可以完成除法。一种是用乘法来代替，除以某个数相当于乘以其倒数，所以先求出其倒数，然后相乘。这种方法对于除以常数特别适用。另一种方法是使用 SUBC 指令，重复若干次减法完成除法。

下面这几条指令就是利用 SUBC 来完成整数除法（TEMP1/TEMP2）的：

```
LD     TEMP1, B           ;将被除数 TEMP1 装入 B 的低 16 位
RPT    #15                ;重复 SUBC 指令 16 次
SUBC TEMP, B              ;使用 SUBC 指令完成除法
STL    B, TEMP3           ;将商(B 的低 16 位)存入变量 TEMP3
STH    B, TEMP4           ;将余数(B 的高 16 位)存入变量 TEMP4
```

在 TMS320C54x 中实现 16 位的小数除法与前面的整数除法基本一样，也是使用 SUBC 指令来完成的。但有两点需要注意：第一，小数除法的结果一定是小数（小于 1），所以被除数一定小于除数，在执行 SUBC 指令前，应将被除数装入累加器 A 或 B 的高 16 位，而不是低 16 位，其结果的格式与整数除法一样。第二，应当考虑符号位对结果小数点的影响，所以应将商右移一位，得到正确的有符号数。

（3）乘法指令

乘法指令共有 10 条，见表 3-12。

在 TMS320C54x 中提供大量的乘法指令，其结果都是 32 位的，放在累加器 A 或 B 中。乘数在 TMS320C54x 的乘法指令中很灵活，可以是暂存器 T、立即数、存储单元和累加器 A 或 B 的高 16 位。若是无符号数乘法，则使用 MPYU 指令。这是一条专用于无符号数乘法的指令，而其他指令都是有符号数的乘法。

表 3-12　乘法指令

助记符指令	代数指令	注　　释
MPY　Smem,dst	dst=T*Smem	T 与操作数 Smem 相乘
MPYR　Smem,dst	dst=rnd(T*Smem)	T 与操作数 Smem 相乘（凑整）
MPY　Xmem,Ymem,dst	dst=Xmem*Ymem,T=Xmem	操作数 Xmem 和 Ymem 相乘
MPY　Smem,#lk,dst	dst=Smem* #lk,T=Smem	长立即数与操作数 Smem 相乘
MPY　#lk,dst	dst=T* #lk	长立即数与 T 相乘
MPYA　dst	dst=T*A(32～16)	ACCA 的高阶位与 T 相乘
MPYA　Smem	B=Smem*A(32～16) T=Smem	操作数与 ACCA 的高阶位相乘
MPYU　Smem,dst	dst=uns(T)*uns(Smem)	T 与无符号操作数 Smem 相乘
SQUR　Smem,dst	dst=Smem*Smem,T=Smem	操作数 Smem 的平方
SQUR　A,dst	dst=A(32～16)*A(32～16)	ACCA 的高阶位的平方

整数乘法举例：

```
SSBX    FRCT            ;清 FRCT 位, 准备整数乘法
LD      TEMP1, T        ;将变量 TEMP1 装入 T
MPY     TEMP2, A        ;完成 TEMP2*TEMP1, 结果放入 A(32 位)
```

小数乘法举例：

```
SSBX    FRCT            ;FRCT=1, 准备小数乘法
LD      TEMP1, 16, A    ;将变量 TEMP1 装入 A 的高 16 位
MPYA    TEMP2           ;完成 TEMP2 乘 A 的高 16 位, 结果在 B 中
                        ;同时将 TEMP2 装入 T
STH     B, TEMP3        ;将乘积结果的高 16 位存入变量 TEMP3
```

在 TMS320C54x 中，小数乘法与整数乘法基本相同，只是当两个有符号的小数相乘时，其结果的小数点的位置在次高位的后面，所以必须左移一位，才能得到正确的结果。TMS320C54x 中提供一个小数方式 FRCT 位，将其设置为 1，系统自动将乘积结果左移 1 位。两个小数（16 位）相乘后的结果为 32 位，如果精度允许，可以只存高 16 位，将低 16 位丢弃，这样仍可得到 16 位的结果。

（4）乘加与乘减指令

乘加与乘减指令共有 22 条，见表 3-13。

（5）双操作数指令

双操作数指令共有 6 条，见表 3-14。

（6）特殊操作指令

特殊操作指令共有 15 条，见表 3-15。

表 3-13　乘加与乘减指令

助记符指令	代数指令	注　　释
MAC　Smem,src	src=src+T*Smem	操作数 Smem 与 T 相乘后，加到 ACC 中
MAC　Xmem,Ymem src[,dst]	dst=src+Xmem*Ymem T=Xmem	操作数 Xmem 和 Ymem 相乘后，加到 ACC 中
MAC　#lk,src[,dst]	dst=src+T* #lk	T 与长立即数相乘后，加到 ACC 中

MAC　Smem,#lk,src[,dst]	dst=src+Smem* #lk
	T=Smem
		操作数 Smem 与长立即数相乘后，加到 ACC 中

MACR　Smem,src	src=rnd(src+T*Smem)	操作数 Smem 与 T 相乘后，加到 ACC 中（凑整）
MACR　Xmem,Ymem,src[,dst]	dst=rnd(src+Xmem*Ymem) T=Xmem	操作数 Xmem 和 Ymem 相乘后，加到 ACC 中（凑整）
MACA　Smem,[,B]	B=B+Smem*A(32～16) T=Smem	操作数 Smem 与 ACCA 的高阶位相乘后，加到 ACCB 中
MACA　T,src[,dst]	dst=src+T*A(32～16)	T 与 ACCA 的高阶位相乘后，加到 ACC 中
MACAR　Smem,[,B]	B=rnd(B+Smem*A(32～16)) T=Smem	操作数 Smem 与 ACCA 的高阶位相乘后，加到 ACCB 中（凑整）
MACAR　T,src[,dst]	dst=rnd(src+T*A(32～16))	T 与 ACCA 高阶位相乘后，加到 ACC 中（凑整）
MACD　Smem,pmad,src	src=src+Smem*pmad, T=Smem, (Smem+1)=Smem	带延时，操作数 Smem 与程序存储器地址的值相乘后，累加
MACP　Smem,pmad,src	src=src+Smem*pmad, T=Smem	操作数 Smem 与程序存储器地址的值相乘后，累加
MACSU　Xmem,Ymem,src	src=src+uns(Xmem)*Ymem T=Xmem	无符号操作数 Xmem 与有符号操作数 Ymem 相乘后，累加
MAS　Smem,src	src=src−T*Smem	操作数 Smem 与 T 相乘后，与 ACC 相减
MASR　Smem,src	src=rnd(src−T*Smem)	操作数 Smem 与 T 相乘后，与 ACC 相减（凑整）
MAS　Xmem,Ymem,src[,dst]	dst=src−Xmem*Ymem, T=Xmem	操作数 Xmem 和 Ymem 相乘后，与 ACC 相减
MASR　Xmem,Ymem,src[,dst]	dst=rnd(src−Xmem*Ymem) T=Xmem	操作数 Xmem 和 Ymem 相乘后，与 ACC 相减（凑整）
MASA　Smem [,B]	B=B−Smem*A(32～16) T=Smem	从 ACCB 中减去操作数与 ACCA 的乘积
MASA　T,src[,dst]	dst=src−T*A(32～16)	从 src 中减去 ACCA 的高阶位与 T 的乘积
MASAR　T,src[,dst]	dst=rnd(src−T*A(32～16))	从 src 中减去 ACCA 的高阶位与 T 的乘积（凑整）
SQURA　Smem,src	src=src+Smem*Smem, T=Smem	操作数 Smem 平方后累加
SQURS　Smem,src	src=src−Smem*Smem, T=Smem	操作数 Smem 平方后做减法

表 3-14　双操作数指令

助记符指令	代数指令	注　　释
DADD　Lmem,src[,dst]	if C16=0 dst=Lmem+src if C16=1 　dst(39～16)=Lmem(31～16)+src(31～16) 　dst(15～0)=Lmem(15～0)+src(15～0)	若 C16=0，完成双精度加法；若 C16=1，完成双 16 位加法

续表

助记符指令	代数指令	注　　释
DADST　Lmem,dst	if C16=0 dst=Lmem+(T<<16+T) if C16=1 dst(39～16)=Lmem(31～16)+T dst(15～0)=Lmem(15～0) −T	若 C16=0，完成双精度加法；若 C16=1，完成双 16 位加/减法
DRSUB　Lmem,src	if C16=0 src=Lmem−src if C16=1 src(39～16)=Lmem(31～16) −src(31～16) src(15～0)=Lmem(15～0) −src(15～0)	若 C16=0，完成双精度减法；若 C16=1，完成双 16 位减法
DSADT　Lmem,dst	if C16=0 dst=Lmem− (T<<16+T) if C16=1 dst(39～16)=Lmem(31～16) −T dst(15～0)=Lmem(15～0)+T	若 C16=0，完成双精度减法；若 C16=1，完成双 16 位加/减法
DSUB　Lmem,src	if C16=0 src=src−Lmem if C16=1 src(39～16)=src(31～16) − Lmem(31～16) src(15～0)=src(15～0) − Lmem(15～0)	若 C16=0，ACC 减双精度操作数 Lmem；若 C16=1，完成双 16 位减法
DSUBT　Lmem,dst	if C16=0 dst=Lmem− (T<<16+T) if C16=1 dst(39～16)=Lmem(31～16) −T dst(15～0)=Lmem(15～0) −T	若 C16=0，双精度操作数 Lmem 减 T；若 C16=1，双 16 位操作数 Lmem 减 T

表 3-15　特殊操作指令

助记符指令	代数指令	注　　释
ABDST　Xmem,Ymem	B=B+|A(32～16)| A=(Xmem−Ymem)<<16	求绝对值
ABS　src[,dst]	dst=|src|	ACC 取绝对值
CMPL　src[,dst]	dst= $\overline{src}$	ACC 取反
DELAY　Smem	(Smem+1)=Smem	存储器延迟
EXP　src	T=符号所在的位数(src)	求 ACC 的指数
FIRS　Xmem,Ymem,pmad	B=B+A*pmad A=(Xmem+Ymem)<<16	对称有限冲激响应滤波器
LMS　Xmem,Ymem	B=B+Xmem*Ymem A=A+Xmem<<16+2^15	求最小均方值
MAX　dst	dst=max(A,B)	求 ACC 的最大值
MIN　dst	dst=min(A,B)	求 ACC 的最小值
NEG　src[,dst]	dst=−src	求 ACC 的反值
NORM　src[,dst]	dst=src<<TS dst=norm(src,TS)	归一化处理
POLY　Smem	B=Smem<<16 A=rnd(A(32～16)*T+B)	求多项式的值（凑整）
RND　src[,dst]	dst=src+2^15	对 ACC 进行四舍五入
SAT　src	饱和计算(src)	对 ACC 进行饱和计算
SQDST　Xmem,Ymem	B=B+A(32～16)*A(32～16) A=(Xmem−Ymem)<<16	求两点之间距离的平方

2．逻辑指令

逻辑指令包括与指令、或指令、异或指令、移位指令和测试指令。根据操作数的不同，这些指令需要 1～2 个指令周期。

（1）与指令

与指令共有 5 条，见表 3-16。

表 3-16　与指令

助记符指令	代数指令	注　　释
AND　Smem,src	src=src & Smem	操作数 Smem 和 ACC 相与
AND　#lk[,SHFT],src[,dst]	dst=src & #lk<<SHFT	长立即数移位后，和 ACC 相与
AND　#lk,16,src[,dst]	dst=src & #lk<<16	长立即数左移 16 位后，和 ACC 相与
AND　src[,SHIFT] [,dst]	dst=dst & src<<SHIFT	ACC 移位后，相与
ANDM　#lk,Smem	Smem=Smem & #lk	操作数 Smem 和长立即数相与

（2）或指令

或指令共有 5 条，见表 3-17。

表 3-17　或指令

助记符指令	代数指令	注　　释
OR　Smem,src	src=src | Smem	操作数 Smem 和 ACC 相或
OR　#lk[,SHFT],src[,dst]	dst=src | #lk<<SHFT	长立即数移位后，和 ACC 相或
OR　#lk,16,src[,dst]	dst=src | #lk<<16	长立即数左移 16 位后，和 ACC 相或
OR　src[,SHIFT] [,dst]	dst=dst | src<<SHIFT	ACC 移位后，相或
ORM　#lk,Smem	Smem=Smem | #lk	操作数 Smem 和长立即数相或

（3）异或指令

异或指令共有 5 条，见表 3-18。

表 3-18　异或指令

助记符指令	代数指令	注　　释
XOR　Smem,src	src=src ∧ Smem	操作数和 ACC 相异或
XOR　#lk[,SHFT],src [,dst]	dst=src ∧ #lk<<SHFT	长立即数移位后，和 ACC 相异或
XOR　#lk,16,src [,dst]	dst=src ∧ #lk<<16	长立即数左移 16 位后，和 ACC 相异或
XOR　src[,SHIFT] [,dst]	dst=dst ∧ src<<SHIFT	ACC 移位后，相异或
XORM　#lk,Smem	Smem=Smem ∧ #lk	操作数和长立即数相异或

（4）移位指令

移位指令共有 6 条，见表 3-19。

表 3-19　移位指令

助记符指令	代数指令	注　　释
ROL　src	带进位标志位循环左移	ACC 循环左移
ROL　TC　src	带 TC 位循环左移	ACC 带 TC 位循环左移
ROR　src	带进位标志位循环右移	ACC 循环右移
SFTA　src,SHIFT [,dst]	dst=src<<SHIFT(算术移位)	ACC 算术移位
SFTC　src	if src(31)=src(30) then src=src<<1	ACC 条件移位
SFTL　src,SHIFT [,dst]	dst=dst<<SHIFT(逻辑移位)	ACC 逻辑移位

（5）测试指令

测试指令共有 5 条，见表 3-20。

表 3-20　测试指令

助记符指令	代数指令	注　　释
BIT　Xmem,BITC	TC=Xmem(15−BITC)	测试指定位
BITF　Smem,#lk	TC=(Smem & #lk)	测试由立即数指定的位
BITF　Smem	TC=Smem(15−T(3−0))	测试由 T 指定的位
CMPM　Smem,#lk	TC=(Smem==#lk)	比较操作数和立即数
CMPR　CC,ARx	Compare　ARx　with AR0	比较辅助寄存器 ARx 和 AR0

3．程序控制指令

程序控制指令包括分支指令、调用指令、中断指令、返回指令、重复指令、堆栈操作指令和其他程序控制指令，这些指令根据不同情况分别需要 1～6 个指令周期。

（1）分支指令

分支指令共有 6 条，见表 3-21。

表 3-21　分支指令

助记符指令	代数指令	注　　释
B[D]　pmad	PC=pmad(15～0)	可以选择延时的无条件转移
BACC[D]　src	PC=src(15～0)	可以选择延时的指针指向的地址
BANZ[D]　pmad,Sind	if(Sind≠0) then PC=pmad(15～0)	当 AR 不为 0 时，转移
BC[D]　pmad,cond [,cond[,cond]]	if(cond(s)) then PC=pmad(15～0)	可以选择延时的条件转移
FB[D]　extpmad	PC=pmad(15～0) XPC=pmad(22～16)	可以选择延时的远程无条件转移
FBACC[D]　src	PC=src(15～0) XPC=src(22～16)	远程转移到 ACC 所指向的地址

（2）调用指令

调用指令共有 5 条，见表 3-22。

表 3-22　调用指令

助记符指令	代数指令	注　　释
CALA[D]　src	−−SP,PC+1[3]=TOS PC=src(15～0)	可选择延时的调用 ACC 所指向的子程序
CALL[D]　pmad	−−SP,PC+2[4]=TOS PC=pmad(15～0)	可以选择延时的无条件调用
CC[D]　pmad,cond[,cond[,cond]]	if(cond(s)) then　−−SP PC+2[4]=TOS PC=pmad(15～0)	可以选择延时的条件调用
FCALA[D]　src	−−SP,PC+1[3]=TOS PC=src(15～0) XPC=src(22～16)	可以选择延时的远程无条件调用

续表

助记符指令	代数指令	注　　释
FCALL[D]　extpmad	--SP,PC+2[4]=TOS PC=pmad(15～0) XPC=pmad(22～16)	可以选择延时的远程条件调用

（3）中断指令

中断指令共有两条，见表 3-23。

表 3-23　中断指令

助记符指令	代数指令	注　　释
INTRK	--SP,++PC=TOS PC=IPTR(15～7)+K<<2 INTM=1	软件中断
TRAPK	--SP,++PC=TOS PC=IPTR(15～7)+K<<2	软件中断

（4）返回指令

返回指令共有 6 条，见表 3-24。

表 3-24　返回指令

助记符指令	代数指令	注　　释
FRET[D]	XPC=TOS,++SP PC=TOS,++SP	可以选择延时远程返回
FRETE[D]	XPC=TOS,++SP PC=TOS,++SP，INTM=0	可以选择延时远程返回，且允许中断
RC[D]　cond[,cond[,cond]]	if(cond(s)) then PC=TOS,++SP	可以选择延时的条件返回
RET[D]	PC=TOS,++SP	可以选择延时的条件返回
RETE[D]	PC=TOS,++SP,INTM=0	可以选择延时的条件返回，且允许中断
RETF[D]	PC=RTN,++SP,INTM=0	可以选择延时的快速条件返回，且允许中断

（5）重复指令

重复指令共有 5 条，见表 3-25。

表 3-25　重复指令

助记符指令	代数指令	注　　释
RPT　Smem	循环执行一条指令，RC=Smem	循环执行下一条指令，RC 为操作数
RPT　#K	循环执行一条指令，RC=#K	循环执行下一条指令，RC 为短立即数
RPT　#lk	循环执行一条指令，RC=#lk	循环执行下一条指令，RC 为长立即数
RPTB[D]　pmad	循环执行一段指令，RSA=PC+2[4]，REA=pmad，BRAF=1	可以选择延迟的块循环
RPTZ　dst,#lk	循环执行一条指令，RC=#lk, dst=0	循环执行下一条指令，且对 ACC 清 0

（6）堆栈操作指令

堆栈操作指令共有 5 条，见表 3-26。

表 3-26　堆栈操作指令

助记符指令	代数指令	注　　释
FRAME K	SP=SP+K	堆栈指针立即数
POPD Smem	Smem=TOS,++SP	把一个值从栈顶弹出，存放到数据存储器中
POPM MMR	MMR=TOS,++SP	把一个值从栈顶弹出，存放到存储器映射寄存器中
PSHD Smem	--SP,Smem=TOS	把数据存储器的一个值压入堆栈
PSHM MMR	--SP,MMR=TOS	把存储器映射寄存器的一个值压入堆栈

（7）其他程序控制指令

其他程序控制指令共有 7 条，见表 3-27。

表 3-27　其他程序控制指令

助记符指令	代数指令	注　　释
IDLE　K	IDLE(K)	保持空闲状态，直到有中断产生
MAR　Smem	if CMPT=0,then modify ARx if CMPT=1 and ARx≠AR0, 　then modify ARx,ARP=x if CMPT=1 and ARx=AR0, 　then modify AR(ARP)	修改辅助寄存器
NOP	无	无任何操作
RESET	软件复位	软件复位
RSBX　N,S 位	S 位=0 ST(N,S 位)=0	状态寄存器复位（清 0）
SSBX　N,S 位	S 位=1 ST(N,S 位)=1	状态寄存器置位
XC　n,cond[,cond [,cond]]	如果满足条件，则执行下面的 *n* 条指令，*n*=1 或 2	条件执行

4．装入和存储指令

装入和存储指令包括一般存储指令、一般装入指令、条件存储指令、并行装入和存储指令、并行装入和乘法指令、并行存储和加减指令、并行存储和乘法指令以及其他装入和存储指令。这些指令根据不同情况分别需要 1～5 个指令周期。

（1）一般存储指令

一般存储指令共有 14 条，见表 3-28。

表 3-28　一般存储指令

助记符指令	代数指令	注　　释
DST　src,Lmem	Lmem=src	ACC 存放到长字中
ST　T,Smem	Smem=T	存储 T 的值
ST　TRN,Smem	Smem=TRN	存储 TRN 的值
ST　#lk,Smem	Smem=#lk	存储长立即数
STH　src,Smem	Smem=src(31～16)	ACC 的高阶位存放到数据存储器中
STH　src,ASM,Smem	Smem=src(31～16)<<(ASM)	ACC 的高阶位移动 ASM 位后，存放到数据存储器中
STH　src,SHFT,Xmem	Xmem=src(31～16)<<(SHFT)	ACC 的高阶位移位后，存放到数据存储器中
STH　src[,SHIFT],Smem	Smem=src(31～16)<<(SHIFT)	ACC 的高阶位移位后，存放到数据存储器中

续表

助记符指令	代数指令	注　　释
STL　src,Smem	Smem=src(15～0)	ACC 的低阶位存放到数据存储器中
STL　src,ASM,Smem	Smem=src(15～0)<<ASM	ACC 的低阶位移动 ASM 位后，存放到数据存储器中
STL　src,SHFT,Xmem	Xmem=src(15～0)<<SHFT	ACC 的低阶位移位后，存放到数据存储器中
STL　src[,SHIFT],Smem	Smem=src(15～0)<<SHIFT	ACC 的低阶位移位后，存放到数据存储器中
STL　M src,MMR	MMR=src(15～0)	ACC 的低阶位存放到存储器中
STM　#lk,MMR	MMR=#lk	ACC 的低阶位存放到存储器映射寄存器中

（2）一般装入指令

一般装入指令共有 21 条，见表 3-29。

表 3-29　一般装入指令

助记符指令	代数指令	注　　释
DLD　Lmem,dst	dst=Lmem	把长字装入 ACC
LD　Smem,dst	dst=Smem	把操作数 Smem 装入 ACC
LD　Smem,TS,dst	dst=Smem<<TS	操作数 Smem 移动由 TREG（5～0）决定的位数后，装入 ACC
LD　Smem,16,dst	dst=Smem<<16	操作数 Smem 左移 16 位后，装入 ACC
LD　Smem[,SHIFT],dst	dst=Smem<<SHIFT	操作数 Smem 移位后，装入 ACC
LD　Xmem,SHFT,dst	dst=Xmem<<SHFT	操作数 Xmem 移位后，装入 ACC
LD　#K,dst	dst=#K	把短立即数装入 ACC
LD　#lk[,SHFT],dst	dst=#lk<<SHFT	长立即数移位后，装入 ACC
LD　#lk,16,dst	dst=#lk<<16	长立即数左移 16 位后，装入 ACC
LD　src,ASM[,dst]	dst=src<<ASM	ACC 移动 ASM 位
LD　src[,SHIFT],dst	dst=src<<SHIFT	ACC 移位
LD　Smem,T	T=Smem	把操作数 Smem 装入 T
LD　Smem,DP	DP=Smem(8～0)	把操作数 Smem 装入 DP
LD　#k9,DP	DP=#k9	把 9 位操作数装入 DP
LD　#k5,ASM	ASM=#k5	把 5 位操作数装入 ASM
LD　#k3,ARP	ARP=#k3	把 3 位操作数装入 ARP
LD　Smem,ASM	ASM=Smem(4～0)	把操作数 Smem 的 4～0 位装入 ASM
LDM　MMR,dst	dst=MMR	把存储器映射寄存器的值装入 ACC
LDR　Smem,dst	dst=rnd(Smem)	把操作数 Smem 装入 ACC 的高阶位（凑整）
LDU　Smem,dst	dst=uns(Smem)	把无符号操作数 Smem 装入 ACC
LTD　Smem	T=Smem,(Smem+1)=Smem	把操作数 Smem 装入 T，并且插入延迟

（3）条件存储指令

条件存储指令共有 4 条，见表 3-30。

表 3-30　条件存储指令

助记符指令	代数指令	注　　释
CMPS　src,Smem	if src(31～16)>src(15～0) 　then Smem=src(31～16) if src(31～16)≤src(15～0) 　then Smem=src(15～0)	比较、选择并存储最大值

续表

助记符指令	代数指令	注　　释
SACCD　src,Xmem,cond	if (cond)　Xmem=src<<(ASM−16)	有条件存储 ACC
SRCCD　Xmem,cond	if (cond)　Xmem=BRC	有条件存储块循环计数器
STRCD　Xmem,cond	if (cond)　Xmem=T	有条件存储 T

（4）并行装入和存储指令

并行装入和存储指令共有 2 条，见表 3-31。

表 3-31　并行装入和存储指令

助记符指令	代数指令	注　　释
ST　src,Ymem ‖ LD　Xmem,dst	Ymem=src<<(ASM−16) ‖ dst=Xmem<<16	存储和装入 ACC 并行执行
ST　src,Ymem ‖ LD　Xmem,T	Ymem=src<<(ASM−16) ‖ T=Xmem	存储 ACC 和装入 T 并行执行

（5）并行装入和乘法指令

并行装入和乘法指令共有 4 条，见表 3-32。

表 3-32　并行装入和乘法指令

助记符指令	代数指令	注　　释
LD　Xmem,dst ‖ MAC　Ymem,dst_	dst=Xmem<<16 ‖ dst_=dst_+T*Ymem	装入和乘加并行执行
LD　Xmem,dst ‖ MACR　Ymem,dst_	dst=Xmem<<16 ‖ dst_=rnd(dst_+T*Ymem)	装入和乘加并行执行，可凑整
LD　Xmem,dst ‖ MAS　Ymem,dst_	dst=Xmem<<16 ‖ dst_=dst_−T*Ymem	装入和乘减并行执行
LD　Xmem,dst ‖ MASR　Ymem,dst_	dst=Xmem<<16 ‖ dst_=rnd(dst_−T*Ymem)	装入和乘减并行执行，可凑整

（6）并行存储和加减指令

并行存储和加减指令共有 2 条，见表 3-33。

表 3-33　并行存储和加减指令

助记符指令	代数指令	注　　释
ST　src,Ymem ‖ ADD　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=dst_+Xmem<<16	存储 ACC 和加法并行执行
ST　src,Ymem‖ SUB　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=(Xmem<<16)−dst_	存储 ACC 和减法并行执行

（7）并行存储和乘法指令

并行存储和乘法指令共有 5 条，见表 3-34。

表 3-34　并行存储和乘法指令

助记符指令	代数指令	注　　释
ST　src,Ymem‖ MAC　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=dst+T*Xmem	存储 ACC 和乘加并行执行
ST　src,Ymem ‖ MACR　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=rnd(dst+T*Xmem)	存储 ACC 和乘加并行执行
ST　src,Ymem ‖ MAS　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=dst−T*Xmem	存储 ACC 和乘减并行执行
ST　src,Ymem ‖ MASR　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=rnd(dst−T*Xmem)	存储 ACC 和乘减并行执行
ST　src,Ymem ‖ MPY　Xmem,dst	Ymem=src<<(ASM−16)‖ dst=T*Xmem	存储 ACC 和乘法并行执行

（8）其他存储和装入指令

其他存储和装入指令共有 12 条，见表 3-35。

表 3-35　其他存储和装入指令

助记符指令	代数指令	注　　释
MVDD　Xmem,Ymem	Ymem=Xmem	数据存储器内部转移
MVDK　Smem,dmad	dmad=Smem	目的地址寻址的数据存储器内部转移
MVDM　dmad,MMR	MMR=dmad	把数据存储器的值转移到存储器映射寄存器中
MVDP　Smem,pmad	pmad=Smem	把数据存储器的值转移到程序存储器中
MVKD　dmad,Smem	Smem=dmad	源地址寻址的数据存储器内部转移
MVMD　MMR,dmad	dmad=MMR	把存储器映射寄存器的值转移到数据存储器中
MVMM　MMRx,MMRy	MMRy=MMRx	在存储器映射寄存器之间转移数据
MVPD　pmad,Smem	Smem=pmad	把程序存储器的值转移到数据存储器中
PORTR　PA,Smem	Smem=PA	从端口把数据读到数据存储器中
PORTW　Smem,PA	PA=Smem	把数据存储器的值写到端口中
READA　Smem	Smem=A	把由 ACCA 寻址的程序存储器的值读到数据存储器中
WRITA　Smem	A=Smem	把数据存储器的值写到由 ACCA 寻址的程序存储器中

5．单个循环指令

TMS320C54x 还提供了单个循环指令，它们引起下一指令被重复执行。重复执行的次数由单个循环指令中的一个操作数决定，并等于操作数加 1。该操作数的值被存储在一个 16 位的循环计数器（RC）中。RC 中的值只能由单个循环指令中的操作数决定，其最大值是 65536。当下一条指令被重复执行时，绝对程序或数据地址将自动加 1。当重复指令被解码时，所有中断（包括 NMI，不包括 RS）均被屏蔽，直到下一条指令被重复执行完毕。重复的功能体现在一些指令中，如乘加或块移动指令，这样就提高了指令的执行速度。下列指令是因为重复执行而由多重循环变成单重循环的。

（1）单个循环指令

对单个数据存储器操作数指令而言，若有一个长的偏移地址或绝对地址，则指令不可被循环执行。单个循环指令共有 11 条，见表 3-36。

表 3-36　单个循环指令

名　　称	说　　明	名　　称	说　　明
FIRS	有限冲激响应滤波器	MVKD	源地址寻址的数据存储器内部转移
MACD	乘和移动结果延时存于 ACC 中	MVMD	存储器映射寄存器到数据存储器移动
MACP	乘和移动结果存于 ACC 中	MVPD	程序存储器到数据存储器移动
MVDK	目的地址寻址的数据存储器内部转移	READA	把程序存储器的值读到数据存储器中
MVDM	数据存储器到存储器映射寄存器移动	WRITA	把数据存储器的值写到程序存储器中
MVDP	数据存储器到程序存储器的移动		

（2）不可使用 RPT 或 RPTZ 指令循环执行的指令

不可使用 RPT 或 RPTZ 指令循环执行的指令共有 36 条，见表 3-37。

表 3-37　不可使用 RPT 或 RPTZ 指令循环执行的指令

名　称	说　明	名　称	说　明
ADDM	长立即数加到数据存储器中	INTR	中断
ANDM	数据存储器的值和长立即数相与	LD ARP	调用辅助寄存器指针（ARP）
B[D]	无条件跳转	LD DP	调用数据页指针
BACC[D]	跳转到 ACC 地址	MVMM	存储器映射寄存器之间的移动
BANZ[D]	跳转到非 0 的辅助寄存器	ORM	数据存储器的值和长立即数相或
BC[D]	条件转移	RC[D]	条件返回
CALA[D]	调用 ACC 地址	RESET	软件复位
CALL[D]	无条件调用	RET[D]	无条件返回
CC[D]	条件调用	RETF[D]	从中断返回
CMPR	和辅助寄存器相比较	RND	凑整
DST	长字（32 位）存储	RPT	重复执行下一条指令
FB[D]	无条件远程跳转	RPTB[D]	块重复
FBACC[D]	远程跳转至 ACC 所指定的位置	RPTZ	重复下一条指令并清除 ACC
FCALA[D]	远程调用子循环，地址由 ACC 指定	RSBX	状态寄存器复位（清 0）
FCALL[D]	无条件远程调用	SSBX	状态寄存器置位
FRET[D]	远程返回	TRAP	软件中断
FRETE[D]	中断使能并从中断中远程返回	XC	条件执行
IDLE	IDLE 指令	XORM	长立即数和数据存储器相异或

在 TMS320C54x 系列中，有一些特殊的 DSP 指令，它们在一个指令周期内用一条指令就可以实现一般需要几条指令才可实现的功能。例如，MAC 指令可以在一个指令周期中完成一次乘法和一次加法。这样既节省了时间，又提高了编程的灵活性。

3.3　流水线技术

DSP 芯片广泛采用流水线技术以减少指令执行时间，从而增强处理器的处理能力。TMS320 系列处理器的流水线深度分为 2～6 级。2～6 级流水线处理器可以并行处理 2～6 条指令，每条指令处于流水线的不同级。在理想情况下，一条 k 段流水能在 $k+(n-1)$个机器周期内处理 n 条指令。其中前 k 个机器周期用于完成第 1 条指令的执行，其余 $n-1$ 条指令的执行需要 $n-1$ 个机器周期。而在非流水处理器上执行 n 条指令则需要 nk 个机器周期。当指令数 n 较大时，流水线的填充和排空时间可以忽略不计，可以认为每个机器周期内执行的最大指令数为 k。但是，由于程序中存在数据相关、程序分支、中断及一些其他因素，因此很难达到这种理想情况。

流水线操作，是指在执行多条指令时，将每条指令的预取指、取指、译码、寻址、读数、执行等操作分级，相差一级重叠执行，即第 1 条指令还处于执行级，第 2 条指令的读数操作已在进行，第 3 条指令则已开始寻址，第 4 条指令则已开始译码，第 5 条指令则已开始取指，第 6 条指令则已开始预取指。

TMS320C54x 采用 6 级深度的流水线作业，它们之间彼此独立，即在任何一个机器周期内，可以有 1～6 条不同的指令同时工作，但每条指令工作在流水线的不同级上。这 6 级流水线的功能见表 3-38。

表 3-38　6 级流水线的功能

第 1 级	第 2 级	第 3 级	第 4 级	第 5 级	第 6 级
P（预取指）	F（取指）	D（译码）	A（寻址）	R（读数）	X（执行）

预取指级：在第 1 个机器周期，用 PC 中的内容加载 PAB。

取指级：在第 2 个机器周期，用读取到的指令字加载 PB。如果是多字指令，则需要几个机器周期才能将一条指令读出来。

译码级：在第 3 个机器周期，用 PB 的内容加载 IR，对 IR 内的指令进行译码，产生执行指令所需的一系列控制信号。

寻址级：如果需要，可用数据 1 读地址加载 DAB，或用数据 2 读地址加载 CAB，修正辅助寄存器和堆栈指针也在这一级进行。

读数级：读数据 1 加载 DB，或读数据 2 加载 CB，如果需要，用数据 3 写地址加载 EAB，以便在流水线的最后一级将数据送入数据存储器。

执行级：执行指令，或用写数据加载 EB。

3.3.1　延迟分支转移指令的流水线

在表 3-38 所示的流水线中，存储器的存取操作可分为两级：先用存储器的地址加载相应的地址总线，然后对存储器进行读/写操作。

【例 3-5】分支转移指令的流水线。

```
地址        指令
a1,a2       B,b1        ;这是一个 4 周期, 2 字的分支指令
a3          I3          ;这是任意的 1 周期, 1 字的指令
a4          I4          ;这是任意的 1 周期, 1 字的指令
…           …
b1          J1          ;这是任意的 1 周期, 1 字的指令
```

分支转移指令的流水线图如图 3-10 所示。

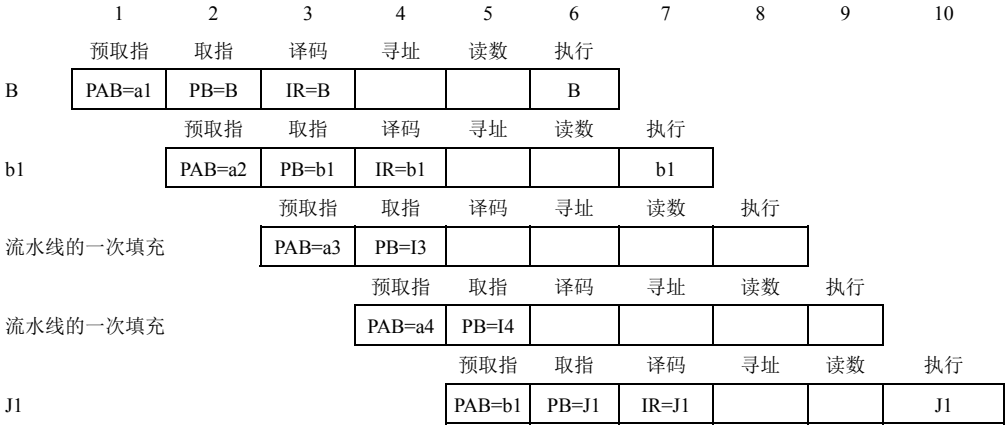

图 3-10　分支转移指令的流水线图

由图 3-10 可知：

周期 1：用分支转移指令的地址 a1 加载 PAB。

周期 2 和 3：取得双字分支转移指令（取指级）。

周期 4 和 5：I3 和 I4 指令取指。由于这两条指令处在分支转移指令之后，虽然已经取指，但不能进入译码级，且最终被丢弃。分支转移指令进入译码级，用新的值（b1）加载 PAB。

周期 6 和 7：双字分支转移指令进入流水线的执行级。在周期 6，J1 指令取指。

周期 8 和 9：由于 I3 和 I4 指令不允许执行，所以这两个周期均花在分支转移指令的执行上。

周期 10：执行 J1 指令。

由上可见，实际上，流水执行分支转移指令只需两个周期。但在周期 4 和周期 5，它还未被执行，不可能到 b1 地址去取指，只能无效地对 I3 和 I4 指令取指，这样一来，总共花了 4 个周期。为了把浪费掉的两个周期利用起来，可以采用延迟分支转移操作。

其方法是，允许跟在延迟分支转移指令后的两条单字、单周期指令 I3 和 I4 执行。这样，只有周期 6 和周期 7 花在延迟分支转移指令上，从而使延迟分支转移指令变成一条 2 周期指令。

【例 3-6】在完成 R=(x+y)*z 操作后转至 next。

可以分别编写出如下两段程序：

利用普通分支转移指令 B		利用延迟分支转移指令 BD	
LD	@x,A	LD	@x,A
ADD	@y,A	ADD	@y,A
STL	A, @s	STL	A, @s
LD	@s,T	LD	@s,T
MPY	@z,A	BD	next
STL	A,@r	MPY	@z,A
B	next	STL	A,@r
(共 8 个字, 10 个 T)		(共 8 个字, 8 个 T)	

可见，延迟分支转移指令可以节省两个周期。具有延迟操作功能的指令有：BD、BANZD、CALLD、FCALLD、RETED、FRETD、BACCD、FBD、CALAD、FCALAD、RETFD、FRETED、BCD、FBACCD、CCD、RETD 和 RCD。

3.3.2 条件执行指令的流水线

在 TMS320C54x 中，有一条条件执行指令 XC，其使用格式为：

XC n,cnd[,cnd[,cnd]]

如果条件满足，则执行下面 n（n=1 或 2）条指令；否则，下面 n 条指令改为执行 n 条 NOP 指令。

【例 3-7】条件执行指令的流水线。

地址	指令
a1	I1
a2	I2
a3	I3
a4	XC 2, cond
a5	I5
a6	I6

它的流水线图如图 3-11 所示。

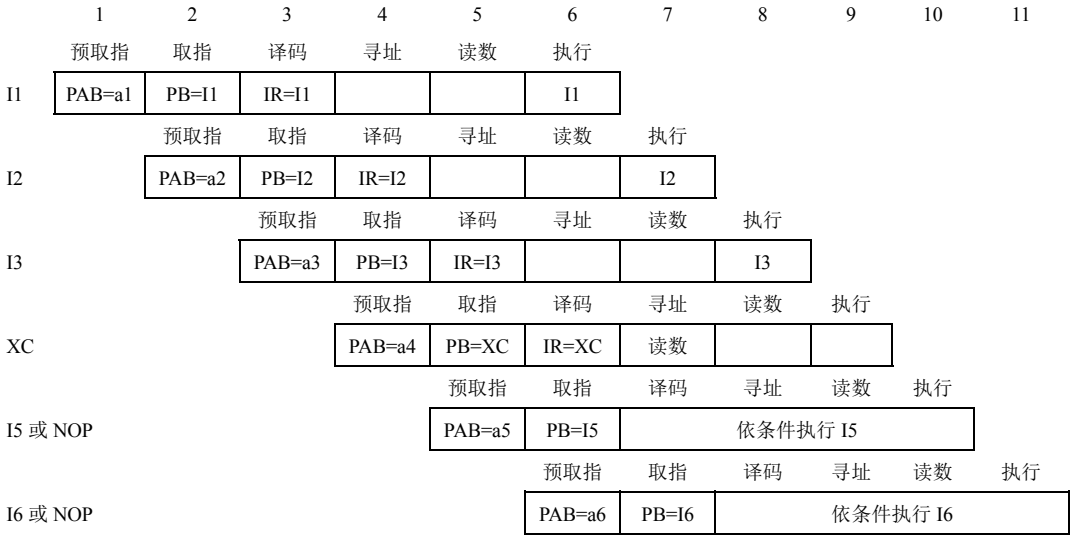

图 3-11　条件执行指令的流水线图

由图 3-11 可知：

周期 4：XC 指令地址 a4 加载 PAB。

周期 5：取 XC 指令的操作码。

周期 7：当 XC 指令在流水线中进行到寻址级时，求解 XC 指令所规定的条件。如果条件满足，则后面的两条指令 I5 和 I6 进入译码级并执行；如果条件不满足，则不对 I5 和 I6 指令进行译码。

条件执行指令是一条单字单周期指令，与条件跳转指令相比，具有快速选择其后一或两条指令是否执行的优点。XC 指令在执行前两个周期就已经求出条件，如果在这之后到执行前改变条件（如发生中断），将会造成无期望的结果。所以要尽力避免在 XC 指令执行前两个周期改变所规定的条件。

3.3.3　双寻址存储器的流水线冲突

TMS320C54x 内部双寻址存储器（DARAM）分成若干个独立的存储器块，允许 CPU 在单个周期内对其进行两次访问，包括下列三种情况：

① 在单周期内允许同时访问 DARAM 的不同存储器块，不会带来时序上的冲突。

② 当流水线中的一条指令访问某个存储器块时，允许流水线中处于同一级的另一条指令访问另一个存储器块，不会带来时序上的冲突。

③ 允许处于流水线不同级上的 2 条指令同时访问同一个存储器块，不会造成时序上的冲突。

CPU 之所以能够在单周期内对 DARAM 进行两次访问，是利用一次访问中对前、后半个周期分时进行访问的缘故：

对 PAB/PB 取指　　利用前半个周期

对 DAB/DB 读取第 1 个数据　　利用前半个周期

对 CAB/CB 读取第 2 个数据　　利用后半个周期

对 EAB/EB 将数据写入寄存器　　利用后半个周期

因此，如果 CPU 同时访问 DARAM 的同一个存储器块，就会发生时序上的冲突。例如，同时从同一个存储器块中取指和读数（都在前半个周期），或者同时对同一个存储器块进行写操作和读（第二个数）操作（都在后半个周期），都会造成时序上的冲突。此时，CPU 或者将写操作延迟一

个周期，或者插入一个空周期，自动地解决上述时序上的冲突。

【例 3-8】CPU 自动解决取指与读数冲突。

```
LD      *AR2+, A        ;AR2 指向装有代码的相同的存储器块
I2
I3
I4
```

图 3-12 给出了 CPU 自动解决取指与读数冲突的流水线图。这里，假定存储器块映射为程序存储器和数据存储器，当第一条指令读数时，就会与 I4 指令的取指发生冲突。TMS320C54x 将 I4 指令的取指延迟一个周期，就自动地解决了矛盾。对于图 3-12，还假定 I2 和 I3 指令不寻址存储器块中的数据。

对于单周期内访问两次内部 DARAM 的情况，CPU 可以在单周期内对每个存储器块访问一次，条件是同时寻址的是不同的存储器块。或者说，在流水线的某一级，一条指令访问某个存储器块，另一条指令访问另一个存储器块，那么，即使同时访问单寻址存储器，也不会产生时序上的冲突。但若访问同一个存储器块，就会出现时序上的冲突。此时，CPU 先在原来的周期上执行一次寻址操作，并将另一次寻址操作自动地延迟到下一个周期。这样，就会导致流水线等待一个周期。

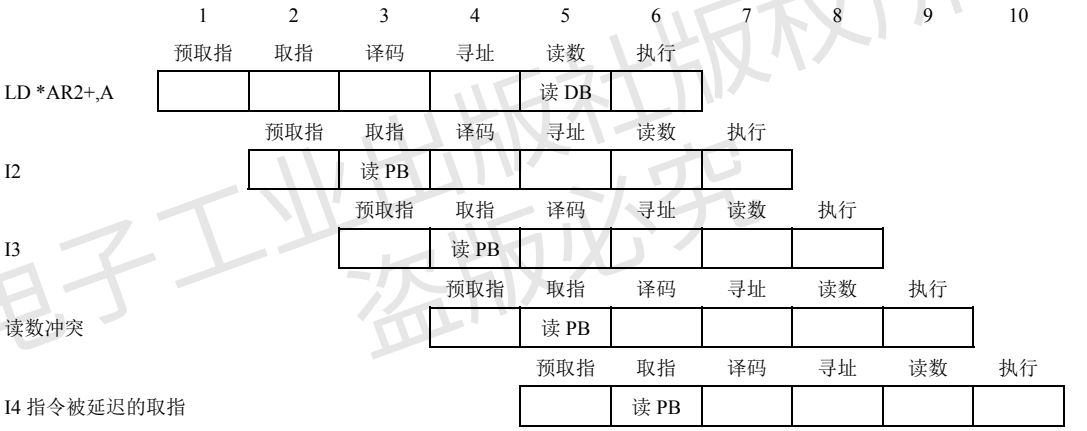

图 3-12　CPU 自动解决取指与读数冲突的流水线图

3.3.4　解决流水线冲突的方法

流水线操作允许多条指令同时寻址 CPU 资源。当一个 CPU 资源同时被一个以上流水线级访问时，可能导致时序上的冲突。其中，有些冲突可以由 CPU 通过延迟寻址的方法自动解决，但仍有一些冲突是不能预防的，需要重新安排指令或者插入空操作 NOP 指令来解决。

1．可能发生流水线冲突的情况

在流水线中，如果多条指令同时寻址存储器映射寄存器，就可能发生不能预防的冲突。存储器映射寄存器（MMR）包括：① 辅助寄存器（AR0～AR7）；② 循环缓冲区长度寄存器（BK）；③ 堆栈指针（SP）；④ 暂存器（T）；⑤ 处理器工作方式状态（PMST）寄存器；⑥ 状态寄存器（ST0 和 ST1）；⑦ 块循环计数器（BRC）；⑧ 存储器映射累加器（AG、AH、AL、BG、BH、BL）。

如图 3-13 所示为流水线冲突情况分析。可以看出，如果 TMS320C54x 系统的源程序是用 C 语言编写的，经过编译生成的代码是没有流水线冲突问题的。如果用汇编语言编写程序，则使用 CALU 操作，或者早在初始化期间就对 MMR 进行设置，也不会发生流水线冲突。利用保护性 MMR 写指令，自动插入等待周期，也可以避免发生冲突。利用等待周期表，通过插入 NOP 指令可以处理好对 MMR 的写操作。

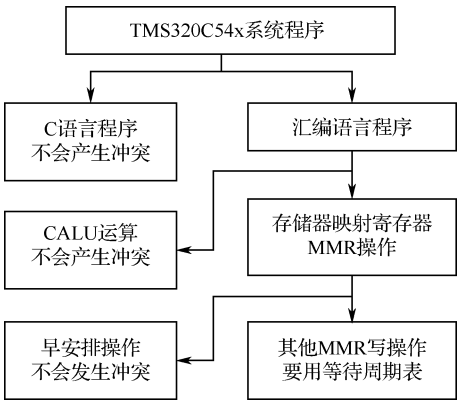

图 3-13　流水线冲突情况分析

流水线冲突是 TMS320C54x 中的一个重要问题，如果解决不好，发生了时序上的冲突，将会影响程序的执行结果。

例如，对辅助寄存器执行标准的写操作引起的时间等待，就是一种流水线冲突问题，如图 3-14 所示。

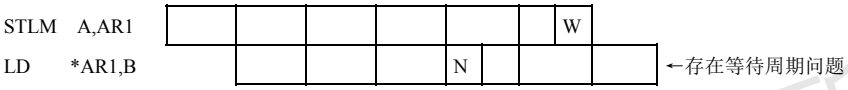

图 3-14　存在等待周期的流水线冲突

在图 3-14 中，W 表示写到 AR1 中，N 表示指令需要 AR1 中的值。STLM 这样的指令，是在流水线的执行级进行写操作的，而 LD 指令又在寻址级生成地址。这样，在第 2 条指令需要 AR1 进行间接寻址读数时，第 1 条指令还没有为 AR1 准备好数据。如果不采取措施，程序执行结果就会出错。

（1）用 STM 指令解决等待周期问题

把上述第 1 条指令改用 STM 指令，如图 3-15 所示，情况就会发生变化。

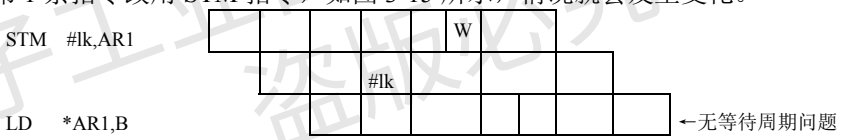

图 3-15　STM 指令解决等待周期问题

这里的 STM 指令是一种保护性操作，一旦常数译码后，马上就写到 AR1 中，接下来的 LD 指令就能顺利地形成正确的地址，并取得操作数后加载累加器 B。除 STM 外，还有一些指令，如 MVDK、MVMM 和 MVMD 等指令也有类似的问题。

（2）用 NOP 指令解决等待周期问题

在图 3-16 中，STLM 指令在执行级将累加器 A 中的内容写入 AR0，而 STM 原来是在读数级将常数 10 写入 AR1 的。第 2 条指令与第 1 条指令发生冲突。因为两者同时利用 E 总线进行写操作。此时，TMS320C54x 内部自动地将 STM 的写操作延迟一个周期，缓解了这一冲突。然而，在继续执行 LD 指令时，需要根据 AR1 间接寻址操作数，由于 AR1 还没准备好，因而发生新的时序上的冲突。

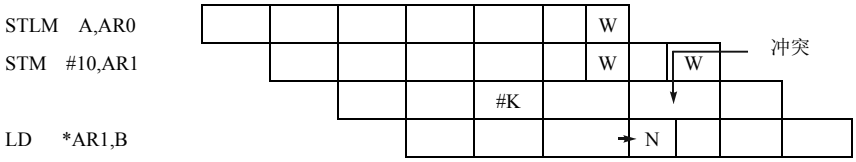

图 3-16　无 NOP 指令时序产生冲突

解决这一冲突最简单的方法是，在 STLM 指令后面插入 NOP 指令或者任何一条与程序无关的单字指令，如图 3-17 所示。

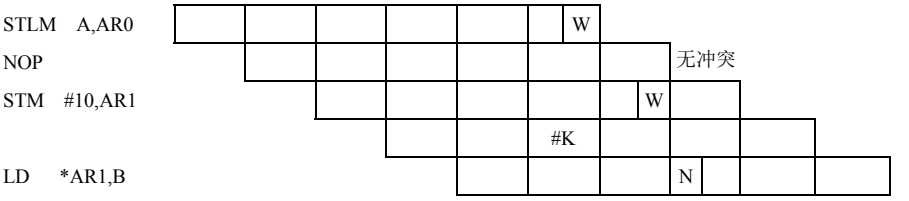

图 3-17　用 NOP 指令解决冲突

2．用等待周期表解决流水线冲突

在对存储器映射寄存器及 ST0、ST1、PMST 寄存器的控制字段进行写操作时，有可能与后续指令造成时序上的冲突。可以通过在这些写操作指令后插入若干条 NOP 指令来解决这些冲突。表 3-39 至表 3-41 列出了等待周期表，这些表中给出了对存储器映射寄存器及控制字段进行写操作的各种指令所需插入的等待周期。

表 3-39　等待周期表 1

控制字段	不插入	插入 1 个	插入 2 个
T	STM　#lk,T MVDK Smem,T LD　Smem,T LD　Smem,T||ST	所有其他存储指令，包括 EXP	
ASM	LD　#k5,ASM LD　Smem,ASM	所有其他存储指令	
DP CPL=0	LD　#k9,DP LD　Smem,DP		STM　#lk,ST0 ST　#lk,ST0 所有其他存储指令插入 3 个
SXM C16 FRCT OVM		所有存储指令，包括 SSXM 和 RSXM	
A 或 B		修改 ACC，然后读 MMR	
在 RPTB[D]中 前读 BRC	STM　#lk,BRC ST　#lk,BRC MVDK Smem,BRC MVMD MMR,BRC	所有其他存储指令	SRCCD （在循环中） 见说明 4

表 3-40　等待周期表 2

控制字段	插入 2 个	插入 3 个	插入 5 个	插入 6 个
DROM	STM,ST MVDK,MVMD	所有其他存储指令		
OVLY IPTR MP / $\overline{\text{MC}}$		所有其他存储指令	STM,ST MVDK,MVMD 见说明 5	所有其他存储指令 见说明 5
DBRAF				RSBX 见说明 3
CPL		RSBX，SSBX		

表 3-41　等待周期表 3

控制字段	不插入	插入 1 个	插入 2 个	插入 3 个
ARx	STM,ST,MVDK,MVMM, MVMD 见说明 2	POPM,POPD 其他 MV 指令 见说明 2	STLM,STL,STH 所有其他存储指令 见说明 1	
BK		STM,ST,MVDK, MVMM,MVMD 见说明 2	POPM 其他 MV 指令 见说明 2	STLM,STL,STH 所有其他存储指令 见说明 1
SP	if CPL=0 STM,MVDK, MVMM, MVMD 见说明 2	if CPL=1 STM, MVDK, MVMM, MVMD 见说明 2	if CPL=0 STLM,STH,STL 所有其他存储指令 见说明 1	if CPL=1 STLM,STH,STL 所有其他存储指令 见说明 1
当 CPL=1 时 暗含 SP 改变		FRAME POPM/POPD PSHM/PSHD		

表 3-39 至表 3-41 中的说明如下。

说明 1：下一条指令不能使用 STM、MVDK 或 MVMD 指令写入任何 ARx、BK 或 SP。

说明 2：在该指令前，不要在流水线的执行级用一条指令写入任何 ARx、BK 或 SP。

说明 3：随后的 6 条单字指令不能包含在 RPTB 循环的最后指令中。

说明 4：在 RPTB 循环的最后指令之前，SRCCD 必须是 2 字指令。

说明 5：所列插入等待要从新激活的存储空间中对第 1 条指令进行取指，如分支、调用或返回类型指令。

【例 3-9】利用表 3-39，插入 NOP 指令以解决流水线冲突。

```
SSBX    SXM
NOP
LD      @a,B
```

由于 LD　@a,B 是一条单字指令，不提供隐含的等待周期，因此根据表 3-39，应当在 SSBX　SXM 指令之后插入一条 NOP 指令。而对于以下程序：

```
SSBX    SXM
LD      *(x),B
```

由于 LD　*(x),B 是一条双字指令，它隐含一个等待周期，故 SSBX 指令后就不用再插入 NOP 指令了。

【例 3-10】利用隐含等待周期解决流水线冲突。

```
LD      @GAIN, T
STM     #input, AR1
MPY     *AR1+, A
```

在 LD 中写 T 和在 STM 中写 AR1 要用 E 总线，由于 STM 是一条双字指令，隐含一个等待周期，因此对于 AR1 来说，等待周期为 0。

【例 3-11】利用表 3-41，插入 NOP 指令以解决流水线冲突。

```
STLM    B, AR2
NOP
```

```
STM     #input, AR3
MPY     AR2+, *AR3+, A
```

在 STM 中写 AR3 要用 E 总线，而在 STLM 中写 AR2 要用 E 总线，发生冲突。查表 3-41 可得，控制字段为 AR3，STLM 指令后应插入两个 NOP 指令，但由于下一条指令 STM 隐含一个等待周期，故只需要插入一条 NOP 指令。

【例 3-12】利用表 3-39，插入 NOP 指令以解决流水线冲突。

```
MAC     @x, B
STLM    B,ST0
NOP
NOP
NOP
ADD     @table,A,B
```

最后一条指令 ADD　@table,A,B 是一条直接寻址指令。如果 CPL=0，则需要用到 ST0 寄存器中的 DP 值。由表 3-39 可查出，当控制字段 CPL=0 时，应在 STLM 指令后插入三条 NOP 指令；而当 CPL=1 时，不需要 DP 值，也就不需要插入 NOP 指令了。

【例 3-13】利用表 3-40 及说明，插入 NOP 指令以解决流水线冲突。

```
RPTB    endloop−1
…
RSBX    BRAF
NOP
NOP
NOP
NOP
NOP
NOP
…
endloop−1
```

查表 3-40 及说明 3，当控制字段为 BRAF 时，应在 RSBX 指令的后面插入 6 条 NOP 指令。也就是说，在 RSBX 指令后面应当有 6 个字，但不包含 RPTB 循环中的最后一条指令。

习题 3

1．TMS320C54x 有哪些寻址方式？它们是如何寻址的？
2．当使用位倒序寻址方式时，应使用什么辅助寄存器？试述地址以位倒序方式产生的过程。
3．TMS320C54x 有哪些分支转移形式？它们是如何工作的？
4．带延迟的分支转移指令与不带延迟的分支转移指令有何差异?
5．可重复操作指令的特点是什么？其最多重复次数是多少？
6．RC 在执行减 1 操作时能否被访问？
7．进行块重复操作要用到几个计数器或寄存器？块重复可否嵌套？重复次数如何设置？
8．长度为 R 的循环缓冲区必须从一个 N 位地址的边界开始，N 与 R 应满足何种关系？

9．*(lk)寻址方式的指令可与循环指令（RPT、RPTZ）一起使用吗？

10．直接寻址方式可以用于程序存储器的寻址吗？

11．汇编指令中的*ARx 表示的是 ARF 所选择的辅助寄存器吗？

12．用双操作数指令编程有何特点？用何种寻址方式获得操作数，且只能用哪些辅助寄存器？

13．TMS320C54x 芯片的流水线共有多少个操作阶段？每个阶段执行什么任务？完成一条指令都需要哪些操作周期？

14．TMS320C54x 芯片的流水线冲突是怎样产生的？有哪些方法可以避免流水线冲突？

15．试分析下列程序的流水线冲突，画出流水线图。如何解决流水线冲突？

```
STLM    A, AR0
STM     #10, AR1
LD      *AR1, B
```

16．试根据等待周期表，确定下列程序需要插入几条 NOP 指令。

```
①    LD      @GAIN, T
     STM     #input, AR1
     MPY     *AR1+, A
②    STLM    B, AR2
     STM     #input, AR3
     MPY     *AR2+, *AR3+, A
③    MAC     @x, B
     STLM    B, ST0
     ADD     @table, A, B
```