第 3 章　8086/8088 指令系统

指令是 CPU 可以理解并执行的操作命令，指令系统是某种 CPU 所能执行的所有指令的集合。不同的 CPU 具有不同的指令系统，相互不一定兼容，但 80x86 系列高档 CPU 的指令系统兼容低档 CPU 的指令系统。本章主要介绍 8086/8088 的指令系统，包括数据传送(Data Transfer)类指令、算术运算(Arithmetic)类指令、逻辑运算(Logic)类指令、串操作(String manipulation)类指令、程序控制(Program Control)类指令和处理器控制(Processor Control)类指令。

本章的重点是理解常用指令的功能，熟悉 8086/8088 的寄存器组和各种寻址方式，它是全面掌握指令功能的关键。在学习指令时，应注意分析指令的功能、指令支持的寻址方式、指令对标志位的影响以及指令的其他方面(如指令执行时的约定设置、必须预置的参数、隐含使用的寄存器等)。

3.1　8086/8088 指令格式与寻址方式

3.1.1　指令的基本格式

1. 机器指令

通常，计算机执行指令来处理数据，为了更加明确地指出指令执行过程中数据的来源以及执行完成结果的去向，一条完整的指令应包括下列信息。

(1) 操作码：说明本条指令进行操作的性质及功能，表示该指令所要完成的操作，例如加、减、乘、除等。每条指令都有一个相应的操作码，计算机就是通过识别该操作码来完成不同操作的。

(2) 操作数的地址：也可称之为地址码，CPU 通过该地址可获得需要的操作数。

(3) 操作结果的存储地址：把本条指令执行完成之后产生的结果保存在这个存储地址中，便于再次使用。

(4) 下一条指令的地址：通常在程序顺序执行的情况下，下一条指令的地址由程序计数器给出，仅在程序的运行顺序改变的时候，下一条指令的地址才由指令给出。

计算机的指令格式与机器的字长、存储器的容量以及指令的功能都有很密切的关系。从提高指令功能的角度来看，指令中所包含的信息越多越好。但是这样会出现一个问题，就是在有些指令中，一部分信息没有被充分利用，造成指令存储空间的浪费，影响访问速度。因此，一条指令往往只包括两种信息：操作码和地址码(操作数)。

指令在计算机中均以二进制编码的形式进行存放，这种类型的指令也称为机器指令，格式如下：

操作码	地址码

机器指令是 CPU 能够直接识别并执行的命令(二进制代码)。但是对使用者而言,机器指令在记忆、阅读和书写方面都是比较困难的。因此,在学习指令时都采用容易记忆的符号和格式来表示,即助记符指令,或汇编语言指令。

2. 汇编语言指令

8086/8088 汇编语言的指令格式如下:

[标号] 指令助记符 [操作数表] [;注释]

其中,用方括号括起来的部分,可以有也可以没有。每部分之间用空格(至少一个)分开,每条语句一般占一行,一行最多可有 132 个字符(MASM6.0 以后的版本可以是 512 个字符)。

(1) 标号

标号是给指令或某一存储单元地址所起的名字。标号的标识符可由下列字符组成:

大小写字母:A～z ;数字:0～9 ;特殊字符:"?"、"·"、"@"、"—"、"$"。

数字不能作标识符的第一个字符,而圆点仅能用作第一个字符。标识符最长为 31 个字符。标识符后跟冒号。在一个源程序中,用户定义的每个标识符必须是唯一的,但不能是汇编程序采用的保留字。汇编程序的保留字(Reserved Word)主要有指令助记符、伪指令助记符、操作符、寄存器名以及预定义符号等。

(2) 指令助记符

表示指令所执行的操作。为了方便记忆,一般由英文单词缩写或几个单词的第一个字符组成。

(3) 操作数

操作数表示参与操作的对象,可以是立即数、寄存器和存储单元。操作数可以有一个或两个,写在左边的操作数为目标操作数,右边的操作数为原操作数,操作数之间用逗号隔开。

(4) 注释

语句中分号后的内容是注释,通常是对该指令或该段程序功能的说明,目的是提高程序的可读性,它必须以分号";"开始。必要时,一个语句行也可以由分号开始作为阶段性注释。汇编程序在翻译源程序时将跳过该部分,不对它们做任何处理。

由于汇编语言指令易读、易记,因此下面的指令系统介绍都采用这一格式。

在绝大多数计算机的存储器中,操作数和指令字的写入或读出都是采用地址指定方式的。要学习指令系统,首先要掌握指令字和操作数的寻址方式。确定本条指令的数据地址以及下一条要执行的指令地址的方法就称为寻址方式。

3.1.2 指令的寻址方式

指令的寻址方式有两种:顺序寻址方式和跳转寻址方式。

(1) 顺序寻址方式:指令的顺序执行过程称为顺序寻址方式。若干条指令构成的程序在内存中都是按一定的顺序存放的,执行时逐条顺序地完成。通常使用程序计数器 PC 对指令顺序进行计数,这个数就是指令在内存中的地址。

(2) 跳转寻址方式:程序改变执行顺序时指令的寻址方式称为跳转寻址方式。在这种寻址方式下,下一条指令的地址码不是由程序计数器 PC 给出的,而是由本条指令的地址码给出的。跳转之后按照新的指令地址开始顺序执行。例如,当前 PC=1020H,而这时要执

行跳转指令 JMP 1350H，执行完这条指令后 PC 的值要变为 1350H。在结构化程序设计中用指令跳转寻址方式，可以实现程序转移或构成循环程序，缩短程序长度或将某些程序作为公共程序引用。

3.1.3 8086/8088 操作数的寻址方式

一条指令由操作码和操作数两部分组成。操作码说明计算机要执行哪种操作，而操作数存在于何处呢？操作数可以存放于操作码之后，即指令中。也可以存放于 CPU 内部的寄存器中，还可以存放于存储器数据区中。形成操作数有效地址的方式称为操作数寻址方式。操作数采用哪一种寻址方式，将会影响机器运行的速度和效率。如何寻址一个操作数，对程序的设计也是至关重要的。

1. 立即数寻址

采用立即数寻址方式的操作数就直接存放在机器代码中，紧跟在操作码之后。这条指令汇编成机器代码后，操作数作为指令的一部分存放在操作码之后的主存单元中。我们称这种操作数为立即数，它可以是 8 位数值(00H～0FFH)，也可以是 16 位数值(0000H～0FFFFH)。例如：将立即数 3000H 送至 AL 寄存器的指令为：

```
MOV  AX,3000H
```

指令功能：AX←3000H，指令代码：B8 00 30。读者可以在 DEBUG 调试程序中用汇编、反汇编等命令查看该指令在主存中的存储及执行结果。在该指令机器代码所在主存单元后的两个字节单元的内容为 00 30，可见 16 位立即数 3000H 紧跟在 MOV 指令后，存放在代码段中。注意高字节 30H 存放于相对高地址中，低字节存放于相对低地址单元中，如图 3-1 所示。

立即数寻址方式常用来给寄存器和存储单元赋初值。在汇编语言中，立即数是以常量形式出现的。常量可以是二进制数、十进制数、十六进制数(以 A～F 开头则要加个 0)、字符串(用单或双引号括起的字符，表示对应的 ASCII 码值，例如：'A'=41H)，还可以是标识符表示的符号常量、数值表达式等。

2. 寄存器寻址

寄存器寻址方式的操作数存放在 CPU 内部的寄存器中，它可以是 8 位寄存器：AH/AL/BH/BL/CH/CL/DH/DL；也可以是 16 位寄存器：AX/BX/CX/DX/SI/DI/BP/SP。另外，操作数还可以存放在 4 个段寄存器 CS/DS/SS/ES 中。

如：将 BX 寄存器内容送至 AX 寄存器，指令如下，执行示意图如图 3-2 所示。

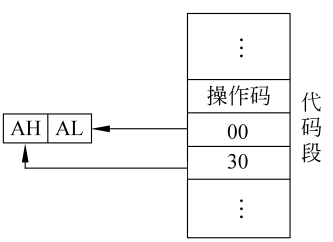

图 3-1　立即数寻址

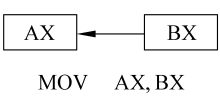

图 3-2　寄存器寻址

```
MOV  AX,BX
```

两个操作数均为寄存器寻址，实现的功能为：AX←BX。

寄存器寻址方式的操作数存放于 CPU 的某个内部寄存器中，取操作数时，不需要访问存储器，因而执行速度较快，是经常使用的寻址方式。在双操作数的指令中，操作数之一必须是寄存器寻址。汇编语言在表达寄存器寻址时使用寄存器名，其实质就是指它存放的内容（操作数）。

3. 存储器寻址

存储器寻址方式的操作数存放在主存储器中，在这种寻址方式下，指令中给出的是有关操作数所在存储器单元的地址信息。

8086/8088 的存储器空间是分段管理的，程序设计时采用逻辑地址。由于段地址放在默认的段寄存器中，所以只需要指明偏移地址。因此也把偏移地址称为有效地址 EA（Effective Address）。为了方便各种数据结构的存取，8086/8088 设计了多种存储器寻址方式。

（1）直接寻址

操作数地址的 16 位偏移量（有效地址 EA）直接包含在指令中。它与操作码一起存放在代码段区域，操作数一般在数据段区域中，即默认的段地址在 DS 段寄存器中，它的物理地址为数据段寄存器 DS 乘以 16 加上有效地址，如图 3-3 所示。

假设数据段寄存器的内容 3000H，将数据段偏移地址为 2000H 单元的内容送至 AX 寄存器的指令如下，执行示意图如图 3-3 所示。

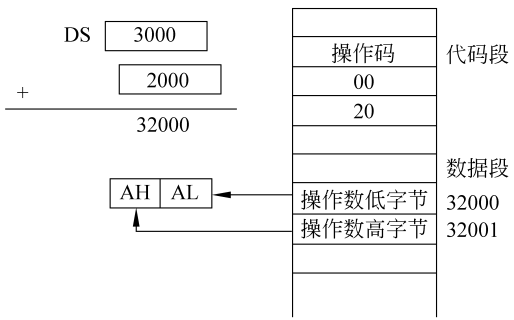

图 3-3　直接寻址示意图

```
MOV  AX,DS:[2000H]
```

（对 DS 来讲可以省略成 MOV AX,[2000H]，系统默认 DS 为数据段寄存器）

这种寻址方法是以数据段的地址为基础，可在多达 64KB 的范围内寻找操作数。

8086/8088 中允许段超越，即还允许操作数放在代码段、堆栈段或附加段区域中。此时需要在指令中指明段超越，格式为：

```
段寄存器:[偏移地址]
```

这时操作数的物理地址＝段寄存器×16＋偏移地址。

例如：下面两条指令的源操作数都是直接寻址。

```
MOV  AX,[2000H]                    ;数据段
MOV  BX,ES:[3000H]
```

第 2 条语句使用段超越，因此操作数在附加段中，物理地址＝(ES)×16＋3000H

（2）寄存器间接寻址

操作数在存储器中，但是操作数的有效地址 EA 包含在以下 4 个寄存器 SI、DI、BP、BX 之一中，可以分成两种情况：

一种是用 SI、DI、BX 的间接寻址，其默认的段地址在 DS 中，即(DS)×16 加上 SI、DI 或 BX 中的内容(有效地址 EA)，形成操作数的物理地址。如：

```
MOV  AX,[SI]
```

其源操作数是寄存器间接寻址，假设(SI)＝2000H，源操作数的物理地址等于：

$$(DS)\times 16+(SI)=32000H$$

寄存器间接寻址示意图如图 3-4 所示。

另一种是用寄存器 BP 的间接寻址，其操作数在堆栈段中。即堆栈段寄存器(SS)×16 与 BP 的内容(有效地址)相加作为操作数的物理地址。如：

```
MOV  AX,[BP]
```

$$源操作数的物理地址=(SS)\times 16+(BP)$$

在寄存器间接寻址的指令中也可以使用段超越。

例如：

```
MOV  AX,DS: [BP]
```

该指令的源操作数物理地址为：(DS)×16＋(BP)。

(3) 寄存器相对寻址

操作数在存储器中，由指定的寄存器内容，加上指令中给出的 8 位或 16 位偏移量作为操作数的有效地址。可以作为寄存器相对寻址的 4 个寄存器是 SI、DI、BX、BP。若用 SI、DI 和 BX 作寄存器相对寻址，则操作数默认在数据段，如：

```
MOV  AX,[SI+4000H]
```

源操作数的物理地址为：(DS)×16＋(SI)＋4000。

假设 SI＝2000H，寄存器相对寻址示意图如图 3-5 所示。

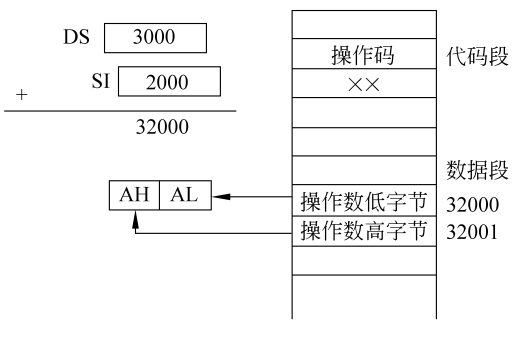

图 3-4　寄存器间接寻址示意图

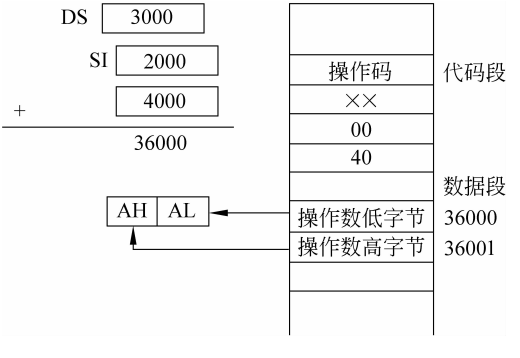

图 3-5　寄存器相对寻址示意图

同样，只要在指令中指定了段超越，则可以用其他的段寄存器作为段地址寄存器。

若用 BP 作为寄存器相对寻址，则 SS 为默认的段地址寄存器。例如：

```
MOV  AX,COUNT[BP]
```

源操作数的物理地址为：(SS)×16＋(SI)＋ COUNT。

(4) 基址加变址寻址

把 BX 和 BP 看做是基址寄存器，把 SI、DI 看做是变址寄存器，把一个基址寄存器(BX 或 BP)的内容加上一个变址寄存器(SI 或 DI)的内容，作为操作数的有效地址，即为基址变址寻址方式，例如：

```
MOV  AX,[BX+SI]
```

指令中源操作数为基址变址寻址方式。假设 SI＝2000H，BX＝4000H，则源操作数的物理地址为 36000H，执行示意图如图 3-6 所示。

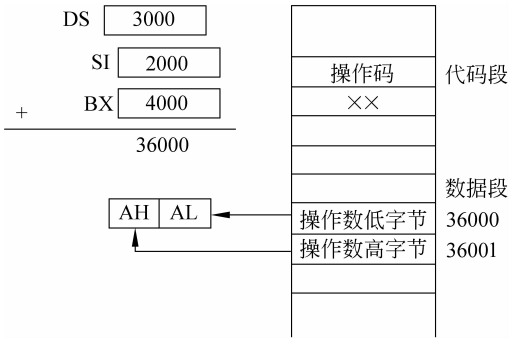

图 3-6　基址变址寻址示意图

当基址寄存器为 BP 时，默认的段地址寄存器为 SS。例如：

```
MOV  AX,[BP][SI]                    ;默认段寄存器为 SS
```

(5) 相对基址变址寻址

基址变址寻址也可以加上一个相对位移量，如 COUNT、MASK 等，用于表示相对寻址。即寄存器(BX 或 BP)的内容加上一个变址寄存器(SI 或 DI)的内容，再加上指令中指定的 8 位或 16 位偏移量作为操作数的有效地址，例如：

```
MOV  AX,MASK[BX][SI]
MOV  BH,COUNT[DI][BP]
MOV  BH,COUNT[BP+DI]
```

若用 BX 作为基地址寄存器，则操作数在数据段中，若用 BP 作为基地址寄存器，则操作数在堆栈段中。当指令中使用段超越时，与基址变址寻址方式的情况相同。

3.2　数据传送类指令

数据传送指令的功能是把数据从一个位置传送到另一个位置。数据传送是计算机中最基本、最重要的一种操作，数据传送指令也是最常使用的一类指令。该类指令除标志操作指令外，其他均不影响标志位。本节主要介绍通用数据传送、交换传送、堆栈传送、地址传送和标志传送等指令。

3.2.1　通用数据传送指令

通用数据传送指令 MOV 把一个字节或一个字的操作数从源地址传送至目的地址。源操作数可以是立即数、寄存器或主存单元，目的操作数可以是寄存器或主存单元，但不能是立即数。MOV 指令是采用寻址方式最多的指令。

指令一般格式为：

```
MOV  OPRD1,OPRD2
```

其中，MOV 是指令助记符，OPRD1 是目的操作数，OPRD2 是源操作数。

指令功能：完成数据传送；把源操作数的数据传送给目的操作数，源操作数不变。

具体来说，一条数据传送指令能实现：

(1) CPU 内部寄存器之间数据的任意传送(除了代码段寄存器 CS 和指令指针 IP 以外)。例如：

```
MOV  AL,BL                ;字节传送
MOV  CX,BX                ;字传送
MOV  DS,BX
```

(2) 立即数传送至 CPU 内部的通用寄存器组(即 AX、BX、CX、DX、BP、SP、SI、DI)。例如：

```
MOV  CL,4
MOV  AX,03FFH
MOV  SI,057BH
```

(3) CPU 内部寄存器(除了 CS 和 IP 以外)与存储器(所有寻址方式)之间的数据传送。例如：

```
MOV  AL,BUFFER
MOV  AX,[SI]
MOV  [DI],CX
MOV  SI,BLOCK[BP]
MOV  DS,DATA[SI+BX]
MOV  DEST[BP+DI],ES
```

(4) 实现用立即数给存储单元赋值。例如：

```
MOV  [2000H],25H
MOV  [SI],35H
```

使用 MOV 指令应注意以下几个问题：

(1) 传送指令中，不允许对 IP 进行操作；CS 不能作为目的操作数。

(2) 两个操作数中，除立即寻址之外必须有一个为寄存器寻址方式，即两个存储器操作数之间不允许直接进行信息传送。

例如：当需要把地址(偏移地址)为 AREA1 的存储单元的内容，传送至同一段内的地址为 AREA2 的存储单元中去时，MOV 指令就不能直接完成这样的传送，而必须使用 CPU 内部寄存器作为桥梁来完成，即：

```
MOV  AL,AREA1
MOV  AREA2,AL
```

(3) 两个段寄存器之间不能直接传送信息，也不允许用立即寻址方式为段寄存器赋初值，如下面两条指令都是错误的。

```
MOV  DS,ES
MOV  DS,0
```

(4) 目的操作数不能用立即寻址方式。

也就是说 MOV 指令可以实现立即数到寄存器、立即数到存储单元的传送，寄存器与寄存器之间、寄存器与存储器之间、寄存器与段寄存器之间的传送，以及存储器与段寄存器之间的传送。

3.2.2 交换传送指令

交换传送指令 XCHG 用来实现源操作数和目的操作数内容的交换，指令一般格式为：

```
XCHG  OPRD1,OPRD2
```

功能：完成两个操作数的交换。

把一个字节或一个字的源操作数与一个字节或一个字的目的操作数相交换。交换能在通用寄存器与累加器之间、通用寄存器之间、通用寄存器与存储器之间进行。但段寄存器和立即数不能作为交换指令的一个操作数。

例如：

```
XCHG  AL,AH                    ;累加器 AX 高 8 位和低 8 位交换
XCHG  AX,DI                    ;两个寄存器的内容交换
XCHG  AX,BUFFER                ;累加器和存储单元的内容交换
XCHG  DATA[SI],DH              ;一个 8 位寄存器的内容与一个字节单元的内容交换
```

3.2.3 堆栈操作指令

堆栈是一块“后进先出”的存储区域，使用 SS 段寄存器记录其段地址。堆栈只有一个出口，即当前栈顶，用堆栈指针寄存器 SP 存放栈顶的偏移地址。堆栈有两种基本操作，对应有两条基本指令，即入栈指令 PUSH 和出栈指令 POP。

1. 入栈指令 PUSH

指令一般格式：

```
PUSH  OPRD
```

其中 OPRD 是源操作数，它可以是 CPU 内部的 16 位通用寄存器、段寄存器(CS 除外)和内存操作数(所有寻址方式)。入栈操作对象必须是 16 位数。

功能：将数据压入堆栈。

执行步骤为：SP－2→SP；操作数低 8 位送至 SP 所指向的堆栈单元；操作数高 8 位送至 SP＋1 所指向的堆栈单元。

例如：PUSH BX (假设 BX＝1234H，SP＝2000H)。

该指令执行前后的堆栈变化情况如图 3-7 所示。

2. 出栈指令 POP

指令一般格式：

```
POP  OPRD
```

其中 OPRD 是目的操作数，对指令执行的要求与入栈指令相同。

功能：将数据弹出堆栈。

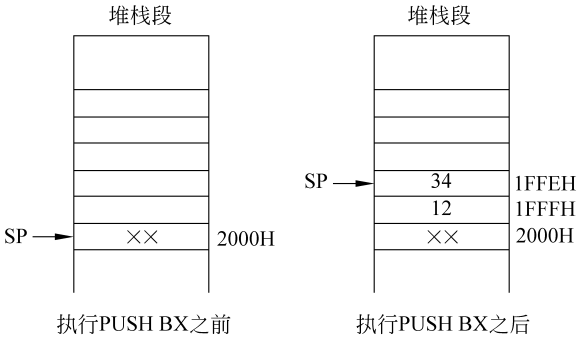

图 3-7　入栈指令 PUSH

执行步骤为：SP 所指向的堆栈单元的内容→目的操作数低 8 位，SP＋1 所指向的堆栈单元的内容→目的操作数高 8 位；SP＋2→SP。

例如：POP AX(假设执行该指令前 SP＝1FFEH)，指令执行前后的堆栈变化情况如图 3-8 所示。

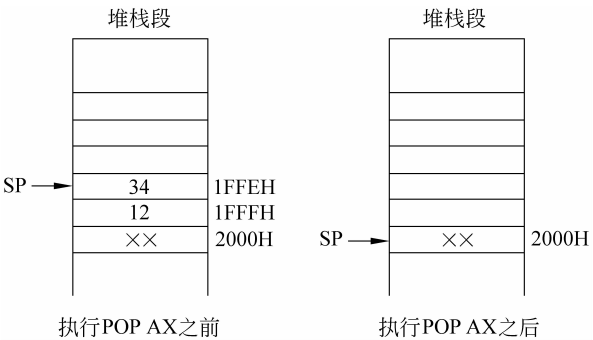

图 3-8　出栈指令 POP

执行该指令后 AX＝1234H(即 SP 所指向单元的内容)，SP＝2000H。

3.2.4　有效地址传送指令

有效地址传送指令 LEA(Load Effective Address)将存储器操作数的有效地址(段内偏移地址)传送至 16 位通用寄存器中。

一般格式：

```
LEA  OPRD1,OPRD2
```

功能：把源操作数 OPRD2 的地址偏移量传送至目的操作数 OPRD1。

要求：源操作数必须是一个存储器操作数，目的操作数必须是一个 16 位的通用寄存器。这条指令通常用来使一个寄存器作为地址指针。

例如：

```
LEA  BX,BUFR                ;把变量 BUFR 的有效地址(偏移地址)送到 BX
LEA  AX,[1234H]             ;将源操作的有效地址送到 AX,指令执行后 AX 中为 1234H
LEA  BX,[BP+SI]             ;指令执行后,BX 内容为 BP+SI 的值
```

```
LEA  BX,TABLE                ;把变量 TABLE 的偏移地址送到 BX
```

3.2.5 换码指令

换码指令 XLAT 是将 AL 寄存器的内容转换成以 BX 为表基址，AL 为表中位移量的表中值。换码指令常用于将一种代码转换为另一种代码，如键盘位置码转换为 ASCII 码，数字 0～9 转换为 7 段显示码等。

指令一般格式：

```
XLAT
```

指令功能：表的内容预先已经存在，表的首地址存放于 BX 寄存器，AL 存放了相对于表首地址的位移量，该指令执行后(BX＋AL)单元的内容→AL。

例如：假设内存中已经存放了 0～9 所对应的 ASCII 码(30H～39H)，其首地址为 TABLE，如图 3-9 所示，要求利用换码指令 XLAT 编程完成将数字 3 转换成对应的 ASCII 码。程序如下：

```
LEA  BX,TABLE
MOV  AL,3
XLAT
```

第 1 条语句是把 TABLE 的偏移地址(有效地址)送 BX，第 2 条语句是把要转换值送入累加器 AL，执行第 3 条语句后，AL＝33H，即 3 的 ASCII 码。

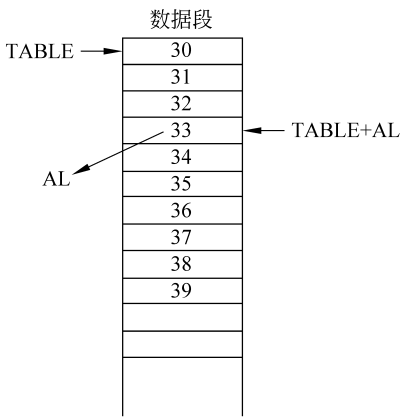

图 3-9　换码指令 XLAT 的功能

在使用换码指令前，首先在主存中要建立一个字节表格，表格的内容是要转换成的目的代码，需要转换的代码存放于 AL 寄存器中，要求被转换的代码应是相对表格首地址的位移量。设置好后，执行换码指令，即将 AL 寄存器的内容转换为目的代码。XLAT 指令默认的缓冲区在数据段。

XLAT 指令中没有显式指明操作数，而是默认使用 BX 和 AL 寄存器。这种采用默认操作数的方法称为隐含寻址方式，指令系统中有许多指令采用隐含寻址方式。

3.2.6 标志寄存器传送指令

可完成标志位传送的指令有 4 条：

1. 读取标志指令 LAHF(Load AH with Flag)

一般格式：

```
LAHF
```

功能：将标志寄存器中的低 8 位(包括 SF、ZF、AF、PF 和 CF)传送至 AH 寄存器的指定位，空位没有定义，如图 3-10 所示。

2. 设置标志指令 SAHF(Store AH with Flag)

一般格式：