第3章 微处理器的指令系统

【学习目标】

8086/8088 CPU 的指令系统是 Intel 80x86 系列 CPU 共同的基础，其后续高型号微处理器的指令系统都是在此基础上新增了一些指令逐步扩充形成的。同时，它也是目前应用最广的一种指令系统。因此，本章将重点讨论 8086/8088 CPU 的指令系统。

通过本章对 8086/8088 CPU 寻址方式和指令系统的学习，应该掌握汇编语言程序设计所需要的汇编语言和编写程序段的基础知识。

【学习要求】

- 在理解与掌握各种寻址方式的基础上，着重掌握存储器寻址的各种寻址方式。
- 熟练掌握 4 类数据传送指令。难点是 XLAT、IN、OUT 指令。
- 学习算术运算类指令中的难点是带符号乘、除指令与十进制指令。
- 学习逻辑运算和移位循环类指令时，要着重理解 CL 的设置以及进位位的处理。
- 学习串操作类指令时，着重理解重复前缀(REP)的使用。
- 学习程序控制类指令时，着重理解条件转移的条件及测试条件。

3.1 8086/8088 的寻址方式

要熟悉指令的操作先要了解指令的寻址方式。8086/8088 的寻址方式分为两种不同的类型：数据寻址方式和程序存储器寻址方式。前者是寻址操作数地址；后者是寻址程序地址(在代码段中)。

3.1.1 数据寻址方式

数据寻址方式有多种，都是寻找操作数。在指令格式中，所有操作数的流向都是由源到目标，即它们在指令汇编语言格式的操作数区域中都是规定由右到左。源和目标可以是寄存器或存储器，但不能同时为存储器(除个别串操作指令 MOVS 外)。

1. 立即寻址

立即寻址的操作数就在指令中，当执行指令时，CPU 直接从紧跟着指令代码的后续地址单元中取得该立即数。立即数可以是 8 位，也可以是 16 位；并规定只能是整数类型的源操作数。这种寻址主要用来给寄存器赋初值，指令执行速度快。

下面列出了各种立即数寻址的 MOV 指令示例。

```
MOV AH,4CH                    ;把 4CH 传送到 AH 中
MOV AX,1234H                  ;把 1234H 传送到 AX 中
MOV CL,100                    ;把 100(64H)传送到 CL
MOV AX,'AB'                   ;把 ASCII 码 BA * (4241H)传送到 AX 中
MOV CL,10101101B              ;把二进制数 10101101 传送到 CL 中
MOV WORD PTR[SI],6180H  ;把立即数 6180H 传送到数据段由 SI 和 SI+1 所指的两存储单元中
```

2. 寄存器寻址

寄存器寻址的操作数就放在寄存器中，而寄存器名在指令中指出。在一条指令中，源操作数和目的操作数都可以采用寄存器寻址方式。这种寻址的指令长度短，操作数就在 CPU 内部，执行速度快。注意，使用时源与目标操作数应有相同的数据类型长度。

下面列出了各种寄存器寻址的 MOV 指令示例。注意，代码段寄存器不能用 MOV 指令来改变，因为若只改变 CS 而 IP 为未知数，则下一条指令的地址将是不确定的。

```
MOV BH,BL                     ;把 BL 复制到 BH 中
MOV CX,AX                     ;把 AX 复制到 CX 中
MOV DI,SI                     ;把 SI 复制到 DI 中
MOV AX,ES                     ;把 ES 复制到 AX 中
```

下面将讨论属于存储器寻址的各种数据寻址方式。存储器寻址比较复杂，当 CPU 寻找存储器操作数时，应根据指令给出的寻址方式，由 EU 先计算出操作数地址的偏移量(即有效地址 EA)。EA 的值由汇编程序根据指令所采用的寻址方式自动计算得出。计算 EA 的通式为：

EA＝基址值(BX 或 BP)＋变址值(SI 或 DI)＋位移量 DISP

3. 直接数据寻址

直接数据寻址有两种基本形式：直接寻址和位移寻址。

1) 直接寻址

直接寻址是指令中以位移量方式直接给出存储器操作数的偏移地址，即有效地址 EA＝DISP。这种寻址方式的指令执行速度快，用于存储单元与 AL、AX 之间的 MOV 指令。

下面列出了使用 AX、AL 的直接寻址指令示例。

```
MOV AX,[1680H]①     ;把数据段存储器地址 1680H 和 1681H 两单元的字内容复制到 AX 中
MOV AX,NUMBER       ;把数据段存储器地址 NUMBER 中的字内容复制到 AX 中
```

① 注意：汇编语言中很少采用绝对偏移地址(如 1680H)，通常采用符号地址。

```
MOV TWO,AL              ;把 AL 的字节内容复制到数据段存储单元 TWO 中
MOV ES:[3000H],AX       ;把 AX 的字内容复制到附加数据段存储单元 3000H 中
MOV AX,DATA             ;把数据段存储单元 DATA 的字内容复制到 AX 中
```

2）位移寻址

位移寻址也以位移量方式直接给出存储器操作数的偏移地址，但适合几乎所有将数据从存储单元传送到寄存器的指令。

下面列出了使用位移量的直接数据寻址的示例。

```
MOV CL,COW              ;把数据段存储单元 COW 的字节内容复制到 CL 中
MOV ES,NUMBER           ;把数据段存储器地址 NUMBER 中的字内容复制到 ES 中
MOV CX,DATA2            ;把数据段存储单元 DATA2 中的字内容复制到 CX 中
MOV DATA3,BP            ;把基址指针寄存器 BP 的内容复制到数据段存储单元 DATA3 中
MOV DI,SUM              ;把数据段存储单元 SUM 的字内容复制到 DI 中
MOV NUMBER,SP           ;把 SP 的内容复制到数据段存储单元 NUMBER 中
```

位移寻址与直接寻址的操作相同，只是它的指令为 4 字节而不是 3 字节。

4. 寄存器间接寻址

寄存器间接寻址的操作数一定是在存储器中，而存储单元的有效地址 EA 则由寄存器保存，这些寄存器是基址寄存器 BX、基址指针寄存器 BP、变址寄存器 SI 和 DI 之一或它们的某种组合。书写指令时，这些寄存器带有方括号[]。

下面给出了寄存器间接寻址的指令示例。

```
MOV AL,[BX]             ;把数据段中用 BX 寻址的存储单元的字节内容复制到 AL 中
MOV [SI],BL             ;把寄存器 BL 的内容复制到数据段由 SI 寻址的存储单元
MOV CX,[DX]             ;把数据段由 DX 寻址的存储单元的字内容复制到 CX 中
MOV [BP],CL①            ;把寄存器 CL 的内容复制到堆栈段由 BP 寻址的存储单元中
MOV [SI],[BX]           ;除数据串操作指令外，不允许由存储器到存储器的传送
```

5. 基址加变址寻址

基址加变址寻址类似于间接寻址，其操作数的有效地址 EA 由基址寄存器(BX 或 BP)的内容与变址寄存器(SI 或 DI)的内容之和来确定。

在使用基址加变址寻址时，通常，用基址寄存器保持存储器数组的起始地址，而变址寄存器保持数组元素的相对位置。如果是用 BP 寄存器寻址存储器数组，则由 BP 寄存器和变址寄存器两者生成有效地址。

下面给出了基址加变址寻址的指令示例。

```
MOV CL,[BX+SI]          ;把由 BX+SI 寻址的数据段存储单元的字节内容复制到 CL
MOV CX,[BP+DI]          ;把由 BP+DI 寻址的堆栈段存储单元内的数据字内容复制到 CX
MOV [BX+DI],SP          ;把 SP 的字内容存入由 BX+DI 寻址的数据段存储单元
MOV [BP+SI],CH          ;把寄存器 CH 的字节内容存入到由 BP+SI 寻址的堆栈段存储单元
```

① 注意：系统把由 BP 寻址的数据默认为在堆栈段中，其他间接寻址方式均默认为数据段。

6. 寄存器相对寻址

寄存器相对寻址是带有位移量 DISP 的基址或变址寄存器(BX、BP 或 DI、SI)寻址。下面给出了寄存器相对寻址的指令示例。

```
MOV CL,[SI+200H]      ;把由 SI+200H 寻址的数据段存储单元的字节内容装入 CL
MOV ARRAY[DI],BL      ;把 BL 中的字节内容存入由 ARRAY+DI 寻址的数据段存储单元
MOV LIST[DI+3],AX     ;把 AX 的字内容存入由 LIST+DI+3 之和寻址的数据段存储单元
MOV AX,ARRAY[BX]      ;把数据段中由 ARRAY+BXD 寻址的字内容装入 AX
MOV SI,[AL+12H]       ;把由 AL+12H 寻址的数据段存储单元的字内容装入 SI
```

注意：位移量可以是在方括号[]内加到寄存器上的一个带符号的数。例如，可以是指令 MOV AL,[SI+4]，也可以是指令 MOV AL,[SI−2]。位移量还可以是加在[]前面的符号偏移地址，例如 MOV AL,DATA[SI]。也可以同时出现两种形式的位移量，例如 MOV AL,DATA[SI+3]。在 8086/8088 中，位移量的取值范围是 −32 768(8000H)～+32 767(7FFFH)。

7. 相对基址加变址寻址

相对基址加变址寻址是用基址、变址与位移量 3 个分量之和形成有效地址的寻址方式。下面给出了相对基址加变址寻址的指令示例。

```
MOV BL,[BX+SI+100H]   ;把由 BX+SI+100H 寻址的数据段存储单元的字节内容装入 BL
MOV AX,ARRAY[BX+DI]   ;把由 ARRAY+BX+DI 之和寻址的数据段存储单元的字内容装入 AX
MOV LIST[BP+DI],BX    ;把 BX 的字内容存入由 LIST+BP+DI 之和寻址的数据段存储单元
MOV FILE[BP+DI+2],DL  ;把 DL 存入由 BP+DI+2 之和寻址的堆栈段存储单元
```

相对基址加变址寻址方式一般很少使用，通常用来寻址存储器的二维数组数据。

3.1.2 程序存储器寻址方式

程序存储器寻址方式即转移类指令(转移指令 JMP 和调用指令 CALL)的寻址方式。这种寻址方式最终是要确定一条指令的地址。

在 8086/8088 系统中，由于存储器采用分段结构，所以转移类指令有段内转移和段间转移之分。所有的条件转移指令只允许实现段内转移，而且是段内短转移，即只允许转移的地址范围在 −128～+127 字节内，由指令中直接给出 8 位地址位移量。对于无条件转移和调用指令又可分为段内短转移、段内直接转移、段内间接转移、段间直接转移和段间间接转移 5 种不同的寻址方式。

3.1.3 堆栈存储器寻址方式

堆栈是存储器中一个很重要的特定存储区，它用来暂时存放数据，并为程序保存返回地址。堆栈存储器是按“后进先出”(LIFO)原理操作的。用 PUSH 指令将数据压入堆栈，用 POP 指令将其弹出堆栈。CALL 指令用堆栈保存程序返回地址，而 RET(返回)指

令从堆栈取出返回地址。

堆栈存储器由堆栈段寄存器 SS 和堆栈指针 SP 寻址。SS 中记录的是其 16 位的段地址,它将确定堆栈段的段基址,而 SP 的 16 位偏移地址将指定当前堆栈的栈顶,即指出从堆栈段的段基址到栈顶的偏移量;栈顶是堆栈操作的唯一出口,它是堆栈地址较小的一端。

下面列出了可使用的一些 PUSH 和 POP 指令的示例。

```
PUSHF          ;把标志寄存器的内容复制到堆栈中
POPF           ;把从堆栈弹出的一个字装入标志寄存器 FLAGS
PUSH DS        ;把 DS 的内容复制到堆栈中
POP CS         ;非法操作
PUSHA          ;把通用寄存器 AX、CX、DX、BX、SP、BP、DI、SI 的内容复制到堆栈中
POPA           ;从堆栈中弹出数据并顺序装入 SI、DI、BP、SP、BX、DX、CX、AX 中
```

3.1.4 其他寻址方式

1. 串操作指令寻址方式

数据串(或称字符串)指令不能使用正常的存储器寻址方式来存取数据串指令中使用的操作数。执行数据串指令时,源串操作数第 1 个字节或字的有效地址应存放在源变址寄存器 SI 中(不允许修改),目标串操作数第 1 个字节或字的有效地址应存放在目标变址寄存器 DI 中(不允许修改)。在重复串操作时,8086/8088 能自动修改 SI 和 DI 的内容,以使它们能指向后面的字节或字。

2. I/O 端口寻址方式

1) 直接端口寻址

端口地址以 8 位立即数方式在指令中直接给出。例如,IN AL,*n* 指令是将端口号为 8 位立即数 *n* 的端口地址中的字节操作数输入到 AL,它所寻址的端口号只能在 0～255 范围内。

2) 间接端口寻址

这类似于寄存器间接寻址,16 位的 I/O 端口地址在 DX 寄存器中,即通过 DX 间接寻址,故可寻址的端口号为 0～65 535。例如,OUT DX,AL 指令是将 AL 的字节内容输出到由 DX 指出的端口中。

3.2 数据传送类指令

数据传送类指令包括 4 类:通用数据传送指令、目标地址传送指令、标志位传送指令、I/O 数据传送指令。

3.2.1 通用数据传送指令

通用数据传送指令包括基本的传送指令 MOV,堆栈操作指令 PUSH 和 POP,数据

交换指令 XCHG 与字节翻译指令 XLAT。

1. 基本的传送指令

MOV d,s;d←s

指令功能：将由源 s 指定的源操作数送到目标 d。

MOV 指令可实现的数据传送类型可归纳为以下 7 种。

1) MOV mem/reg1,mem/reg2

由 mem/reg2 所指定的存储单元或寄存器中的 8 位数据或 16 位数据传送到由 mem/reg1 所指定的存储单元或寄存器中，但不允许从存储器传送到存储器。

2) MOV mem/reg,data

将 8 位或 16 位立即数 data 传送到由 mem/reg 所指定的存储单元或寄存器中。

3) MOV reg,data

将 8 位或 16 位立即数 data 传送到由 reg 所指定的寄存器中。

4) MOV ac,mem

将存储单元中的 8 位或 16 位数据传送到累加器 ac 中。

5) MOV mem,ac

将累加器 AL(8 位)或 AX(16 位)中的数据传送到由 mem 所指定的存储单元中。

6) MOV mem/reg,segreg

将由 segreg 所指定的段寄存器(CS、DS、SS、ES 之一)的内容传送到由 mem/reg 所指定的存储单元或寄存器中。

7) MOV segreg,mem/reg

允许将由 mem/reg 指定的存储单元或寄存器中的 16 位数据传送到由 segreg 所指定的段寄存器(但代码段寄存器 CS 除外)中。

注意：MOV 指令不能直接实现从存储器到存储器之间的数据传送，但可以通过寄存器作为中转站完成这种传送。

2. 堆栈操作指令

1) PUSH s

字压入堆栈指令，允许将源操作数 s(16 位)压入堆栈。

2) POP d

字弹出堆栈指令，允许将堆栈中当前栈顶两相邻单元的数据字弹出到 d。

这是两条成对使用的进栈与出栈指令。其中，s 和 d 可以是 16 位寄存器或存储器两相邻单元，以保证堆栈按字操作。

PUSH 和 POP 是两条很重要的指令，用来保存并恢复来自堆栈存储器的数据。例如，在子程序调用或中断处理过程时，分别要保存返回地址或断点地址，在进入子程序或中断处理后，还需要保留通用寄存器的值；而在由子程序返回或由中断处理返回时，则要恢复通用寄存器的值，并分别将返回地址或中断地址恢复到指令指针寄存器中。

3. 数据交换指令

XCHG d,s

本指令的功能是将源操作数与目标操作数(字节或字)相互对应交换位置。交换可以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。但不能在两个存储单元之间交换,段寄存器与 IP 也不能作为源或目标操作数。例如:

XCHG AX,[SI+0400H]

设当前 CS=1000H,IP=0064H,DS=2000H,SI=3000H,AX=1234H,则该指令执行后,将把 AX 寄存器中的 1234H 与物理地址 23400H(=DS×16+SI+0400H=20000H+3000H+0400H)单元开始的数据字(设为 ABCDH)相互交换位置,即 AX=ABCDH;(23400H)=34H,(23401H)=12H。

4. 字节翻译指令

XLAT

字节翻译指令又称为代码转换指令,用于对不规则代码的转换。其具体功能是:可以将 AL 寄存器中设定的一个字节数值,变换为内存一段连续表格中的另一个相应的代码,以实现编码制的转换。

该指令是通过查表方式完成代码转换功能的,执行的操作是 AL←[BX+AL]。执行结果是将待转换的序号转换成对应的代码,并送回 AL 寄存器中,即执行 AL←[BX+AL]的操作。

例如,假设 0～9 的 7 段显示码表存放在偏移地址为 0030H 开始的内存区,如果要取出 5 所对应的 7 段码(12H),则可以用如下 3 条指令完成:

```
MOV BX,0030H
MOV AL,5
XLAT
```

3.2.2 目标地址传送指令

这是一类专用于 8086/8088 中传送地址码的指令,可传送存储器的逻辑地址(即存储器操作数的段地址或偏移地址)至指定寄存器中。

1. LEA d,s

这是取有效地址指令,其功能是把用于指定源操作数(它必须是存储器操作数)的 16 位偏移地址(即有效地址),传送到一个指定的 16 位通用寄存器中。例如:

LEA BX,[SI+100AH]

设当前 CS=1500H,IP=0200H,DS=2000H,SI=0030H,源操作数 1234H 存放在[SI+100AH]开始的存储器内存单元中,则该指令执行的结果,是将源操作数 1234H 的

有效地址 103AH 传送到 BX 寄存器中。

注意：LEA 指令和 MOV 指令的功能不同，比如 LEA BX,[SI]指令是将 SI 指示的偏移地址(SI 的内容)装入 BX；而 MOV BX,[SI]指令则是将由 SI 寻址的存储单元中的数据装入 BX。

2. LDS d,s

这是取某变量的 32 位地址指针的指令，其功能是从由指令的源 s 所指定的存储单元开始，由 4 个连续存储单元中取出某变量的地址指针(共 4 个字节)，将其前两个字节(即变量的偏移地址)传送到由指令的目标 d 所指定的某 16 位通用寄存器，后两字节(即变量的段地址)传送到 DS 段寄存器中。例如：

```
LDS SI,[DI+100AH]
```

设当前 CS=1000H，IP=0604H，DS=2000H，DI=2400H，待传送的某变量的地址指针其偏移地址为 0180H，段地址为 2230H，则该指令执行后，将物理地址 2340AH 单元开始的 4 个字节中前两个字节(偏移地址值)0180H 传送到 SI 寄存器中，后两个字节(段地址)2230H 传送到 DS 段寄存器中，并取代它的原值 2000H。

3. LES d,s

这条指令与 LDS d,s 指令的操作基本相同，其区别仅在于将把由源所指定的某变量的地址指针中后两个字节(段地址)传送到 ES 段寄存器，而不是 DS 段寄存器。例如：

```
LES DI,[BX]
```

设当前 DS=B000H，BX=080AH，B080AH 单元指定的存储字为 05A2H，B080CH 单元指定的存储字为 4000H，该指令执行后，则将某变量地址指针的前两个字节(即偏移地址)05A2H 装入 DI，而将地址指针的后两个字节(即段地址)4000H 装入 ES，于是，DI=05A2H，ES=4000H。

3.2.3 标志位传送指令

标志位传送指令共有 4 条：LAHF、SAHF、PUSHF 和 POPF。

1. LAHF

指令功能：将标志寄存器 FLAGS 的低字节(共包含 5 个状态标志位)传送到 AH 寄存器中。

LSHF 指令执行后，AH 的 D_7、D_6、D_4、D_2 与 D_0 这 5 位将分别被设置成 SF(符号标志)、ZF(零标志)、AF(辅助进位标志)、PF(奇偶标志)与 CF(进位标志)5 位，而 AH 的 D_5、D_3、D_1 这 3 位没有意义。

2. SAHF

指令功能：将 AH 寄存器内容传送到标志寄存器 FLAGS 的低字节。

SAHF 与 LAHF 的功能相反，它常用来通过 AH 对标志寄存器的 SF、ZF、AF、PF 与

CF 标志位分别置 1 或复 0。

3. PUSHF

指令功能：将 16 位标志寄存器 FLAGS 内容入栈保护。其操作过程与前述的 PUSH 指令类似。

4. POPF

指令功能：将当前栈顶和次栈顶中的数据字弹出送回到标志寄存器 FLAGS 中。

PUSHF 与 POPF 这两条指令常成对出现，一般用在子程序和中断处理程序的首尾，用来保护和恢复主程序涉及的标志寄存器内容；必要时可用来修改标志寄存器的内容。

3.2.4 I/O 数据传送指令

1. IN 累加器，端口号

端口号可以用 8 位立即数直接给出，也可以将端口号事先安排在 DX 寄存器中，间接寻址 16 位长端口号（可寻址的端口号为 0～65 535）。IN 指令是将指定端口中的内容输入到累加器 AL/AX 中。其指令如下：

```
IN AL,PORT    ;AL←(端口 PORT),即把端口 PORT 中的字节内容读入 AL
IN AX,PORT    ;AX←(端口 PORT),即把由 PORT 两相邻端口中的字内容读入 AX
IN AL,DX      ;AL←(端口(DX)),即从 DX 所指的端口中读取一个字节内容送 AL
IN AX,DX      ;AX←(端口(DX)),即从 DX 和 DX+1 所指的两个端口中读取一个字内容送 AX
```

2. OUT 端口号，累加器

与 IN 指令相同，端口号可以由 8 位立即数给出，也可由 DX 寄存器间接给出。OUT 指令是把累加器 AL/AX 中的内容输出到指定的端口。其指令如下：

```
OUT PORT,AL  ;端口 PORT←AL,即把 AL 中的字节内容输出到由 PORT 直接指定的端口
OUT PORT,AX  ;端口 PORT←AX,即把 AX 中的字内容输出到由 PORT 直接指定的端口
OUT DX,AL    ;端口(DX)←AL,即把 AL 中的字节内容输出到由 DX 所指定的端口
OUT DX,AX    ;端口(DX)←AX,即把 AX 中的字内容输出到由 DX 所指定的端口
```

注意：I/O 指令只能用累加器作为执行 I/O 数据传送的机构，而不能用其他寄存器代替。另外，当用直接寻址的 I/O 指令时，寻址范围仅为 0～255，这适用于较小规模的微机系统；当需要寻址大于 255 的端口地址时，则必须用间接寻址的 I/O 指令。

3.3 算术运算类指令

算术运算类指令包括加、减、乘、除与十进制调整 5 类指令。

3.3.1 加法指令

1. ADD d,s;d←d+s

指令功能：将源操作数与目标操作数相加，结果保留在目标中。并根据结果置标

志位。

源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数；目标操作数不允许是立即数，其他同源操作数。而且不允许两者同时为存储器操作数。例如：

```
ADD WORD PTR[BX+106BH],1234H
```

设当前 CS=1000H，IP=0300H，DS=2000H，BX=1200H，则该指令执行后，将立即数 1234H 与物理地址为 2226BH 和 2226CH 中的存储器字 3344H 相加，结果 4578H 保留在目标地址 2226BH 和 2226CH 单元中。

2. ADC d,s;d←d+s+CF

带进位加法(ADC)指令的操作过程与 ADD 指令基本相同，唯一的不同是进位标志位 CF 的原状态也将一起参与加法运算，待运算结束，CF 将重新根据结果置成新的状态。

ADC 指令一般用于 16 位以上的多字节数字相加的软件中。

3. INC d;d←d+1

指令功能：将目标操作数当作无符号数，完成加 1 操作后，结果仍保留在目标中。

目标操作数可以是 8/16 位通用寄存器或存储器操作数，但不允许是立即数。

注意：对于间接寻址的存储单元加 1 指令，数据的长度必须用 TYPE PTR、WORD PTR 或 DWORD PTR 类型伪指令加以说明，否则，汇编程序不能确定是对字节、字还是双字加 1。

另外，INC 指令只影响 OF、SF、ZF、AF、PF 5 个标志，而不影响进位标志 CF，故不能利用 INC 指令来设置进位位，否则程序会出错。

3.3.2 减法指令

1. SUB d,s;d←d−s

指令功能：将目标操作数减去源操作数，其结果送回目标，并根据运算结果置标志位。源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数；目标操作数只允许是通用寄存器或存储器操作数。并且，不允许两个操作数同时为存储器操作数，也不允许做段寄存器的减法。例如：

```
SUB AX,[BX]
```

设当前 CS=1000H，IP=60C0H，DS=2000H，BX=970EH，则该指令执行后，将 AX 寄存器中的目标操作数 8811H 减去物理地址 2970EH 和 2970FH 单元中的源操作数 00FFH，并把结果 8712H 送回 AX 中。各标志位的改变为：O=0(没有溢出)，S=1(结果为负)，Z=0(结果不为 0)，A=1(有半进位)，P=1(奇偶性为偶)，C=0(没有借位)。

2. SBB d,s;d←d−s−CF

本指令与 SUB 指令的功能、执行过程基本相同，唯一不同的是完成减法运算时还要再减去进位标志 CF 的原状态。运算结束时，CF 将被置成新状态。这条指令通常用于比 16 位数宽的多字节减法，在多字节减法中，如同多字节加法操作时传递进位一样，它需要

传递借位。

例如，假定从存于 BX 和 AX 中的 4 字节数减去存于 SI 和 DI 中的 4 字节数，则程序段为：

```
SUB AX,DI
SBB BX,SI
```

3. DEC d；d←d－1

减 1 指令功能：将目标操作数的内容减 1 后送回目标。

目标操作数可以是 8/16 位通用寄存器和存储器操作数，但不允许是立即数。

注意：对于间接寻址存储器数据减 1 指令，要求用 TYPE PTR 类型伪指令来标识数据长。

例如：

```
DEC BYTE PTR[DI]          ;由 DI 寻址的数据段字节存储单元的内容减 1
DEC WORD PTR[BP]          ;由 BP 寻址的堆栈段字存储单元的内容减 1
```

4. NEG d；$d \leftarrow \bar{d}+1$

NEG 是一条求补码的指令，简称求补指令。

指令功能：将目标操作数取负后送回目标。

目标操作数可以是 8/16 位通用寄存器或存储器操作数。

NEG 指令是把目标操作数当成一个带符号数，如果原操作数是正数，则 NEG 指令执行后将其变成绝对值相等的负数(用补码表示)；如果原操作数是负数(用补码表示)，则 NEG 指令执行后将其变成绝对值相等的正数。例如：

```
NEG BYTE PTR[BX]
```

设当前 CS＝1000H，IP＝200AH，DS＝2000H，BX＝3000H，且由目标[BX]所指向的存储单元(＝DS×16＋BX＝23000H)已定义为字节变量(假定为 FDH)，则该指令执行后，将物理地址 23000H 中的目标操作数 FDH＝$[-3]_{补}$，变成＋3 送回物理地址 23000H 单元中。

5. CMP d,s；d－s，只置标志位

指令功能：将目标操作数与源操作数相减但不送回结果，只根据运算结果置标志位。源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数；目标操作数只可以是 8/16 位通用寄存器或存储器操作数。但不允许两个操作数同时为存储器操作数，也不允许做段寄存器比较。比较指令使用的寻址方式与前面介绍过的加法和减法指令相同。例如：

```
CMP [DI],BL               ;DI 寻址的数据段存储单元的字节内容减 BL
CMP CL,[BP]               ;用 CL 减由 BP 寻址的堆栈段存储单元的字节内容
CMP SI,TEMP[BX]           ;由 SI 减由 TEMP＋BX 寻址的数据段存储单元的字内容
```

注意：执行比较指令时，会影响标志位 OF、SF、ZF、AF、PF、CF。当判断两比较数的大小时，应区分无符号数与有符号数的不同判断条件：对于两无符号数比较，只需根据借

位标志 CF 即可判断；而对于两有符号数比较，则要根据溢出标志 OF 和符号标志 SF 两者的异或运算结果来判断。具体判断方法如下：若为两无符号数比较，当 ZF＝1 时，则表示 d＝s；当 ZF＝0 时，则表示 d≠s。如 CF＝0 时，表示无借位或够减，即 d≥s；如 CF＝1 时，表示有借位或不够减，即 d＜s。若为两有符号数比较，当 OF⊕SF＝0 时，则 d≥s；当 OF⊕SF＝1 时，则 d＜s。通常，比较指令后面跟一条条件转移指令，检查标志位的状态以决定程序的转向。

假如，要将 CL 的内容与 64H 作比较，当 CL≥64H 时，则程序转向存储器地址 SUBER 处继续执行。其程序段为：

```
CMP CL,64H                ;CL 与 64H 作比较
JAE SUBER                 ;如果等于或高于则跳转
```

以上的 JAE 为一条等于或高于的条件转移指令。

3.3.3 乘法指令

乘法指令用来实现两个二进制操作数的相乘运算，包括两条指令：无符号数乘法指令 MUL 和有符号数乘法指令 IMUL。

1. MUL s

MUL s 是无符号乘法指令，完成两个无符号的 8/16 位二进制数相乘的功能。被乘数隐含在累加器 AL/AX 中；指令中由 s 指定的源操作数作乘数，它可以是 8/16 位通用寄存器或存储器操作数。相乘所得双倍位长的积，按其高 8/16 位与低 8/16 位两部分分别存放到 AH 与 AL 或 DX 与 AX 中去，即对 8 位二进制数乘法，其 16 位积的高 8 位存于 AH，低 8 位存于 AL；而对 16 位二进制数乘法，其 32 位积的高 16 位存于 DX，低 16 位存于 AX。若运算结果的高位字节或高位字有效，即 AH≠0 或 DX≠0，则将 CF 和 OF 两标志位同时置 1；否则，CF＝OF＝0。据此，利用 CF 和 OF 标志可判断相乘结果的高位字节或高位字是否为有效数值。例如：

```
MUL BYTE PTR[BX+2AH]
```

设当前 CS＝3000H，IP＝0250H，AL＝12H，DS＝2000H，BX＝0234H，且源操作数已被定义为字节变量(66H)，则该指令执行后，乘积 072CH 存放于 AX 中。

2. IMUL s

IMUL s 是有符号乘法指令，完成两个带符号的 8/16 位二进制数相乘的功能。

对于两个带符号的数相乘，不能简单采用与无符号数乘法相同的操作过程，否则会产生完全错误的结果。为此，专门设置了 IMUL 指令。

IMUL 指令除计算对象是带符号二进制数以外，其他都与 MUL 是一样的，但结果不同。

IMUL 指令对 OF 和 CF 的影响是：若乘积的高一半是低一半的符号扩展，则 OF＝CF＝0；否则均为 1。它仍然可用来判断相乘的结果中高一半是否含有有效数值。另外，

IMUL 指令对其他标志位没有定义。例如：

```
IMUL CL                ;AX←(AL)×(CL)
IMUL CX                ;DX、AX←(AX)×(CX)
IMUL BYTE PTR[BX]      ;AX←(AL)×[BX],即 AL 中的和 BX 所指内存单元中的两个 8 位有
                       ;符号数相乘,结果送 AX 中
IMUL WORD PTR[DI]      ;DX、AX←(AX)×[DI],即 AX 中的和 DI、DI+1 所指内存单元中的两个
                       ;16 位有符号数相乘,结果送 DX 和 AX 中
```

有关 IMUL 指令的其他约定都与 MUL 指令相同。

3.3.4 除法指令

除法指令包括无符号二进制数除法指令 DIV 和有符号二进制数除法指令 IDIV。

1. DIV s

DIV s 指令完成两个不带符号的二进制数相除的功能。被除数隐含在累加器 AX(字节除)或 DX、AX(字除)中。指令中由 s 给出的源操作数作除数,可以是 8/16 位通用寄存器或存储器操作数。

对于字节除法,所得的商存于 AL,余数存于 AH。对于字除法,所得的商存于 AX,余数存于 DX。根据 8086 的约定,余数的符号应与被除数的符号一致。例如：

```
DIV BYTE PTR[BX+SI]
```

设当前 CS＝1000H,IP＝0406H,BX＝2000H,SI＝050EH,DS＝3000H,AX＝1500H,存储器中的源操作数已被定义为字节变量 22H,则该指令执行后,所得商数 9EH 存于 AL 中,余数 04H 存于 AH 中。

2. IDIV s

IDIV 指令完成将两个带符号的二进制数相除的功能。它与 DIV 指令的主要区别在于对符号位处理的约定,其他约定相同。具体地说,如果源操作数是字节/字数据,被除数应为字/双字数据并隐含存放于 AX/DX、AX 中。如果被除数也是字节/字数据在 AL/AX中,那么,应将 AL/AX 的符号位(AL_7)/(AX_{15})扩展到 AH/DX 寄存器后,才能开始字节/字除法运算,运算结果商数在 AL/AX 寄存器中,AL_7/AX_{15}是商数的符号位;余数在 AH/DX 中,AH_7/DX_{15}是余数的符号位,它应与被除数的符号一致。在这种情况下,允许的最大商数为＋127/＋32 767,最小商数为－127/－32 767。例如：

```
IDIV BX                ;将 DX 和 AX 中的 32 位数除以 BX 中的 16 位数,商在 AX 中,余数在 DX 中
IDIV BYTE PTR[SI]      ;将 AX 中的 16 位数除以 SI 所指内存单元的 8 位数,所得的商在 AL 中,
                       ;余数在 AH 中
```

3. CBW 和 CWD

CBW 和 CWD 是两条专门为 IDIV 指令设置的符号扩展指令,用来扩展被除数字节/字为字/双字的符号,所扩充的高位字节/字部分均为低位的符号位。它们在使用时应安

排在 IDIV 指令之前，执行结果对标志位没有影响。

CBW 指令将 AL 的最高有效位 D_7 扩展至 AH，即若 AL 的最高有效位是 0，则 AH＝00H；若 AL 的最高有效位为 1，则 AH＝FFH。该指令在执行后，AL 不变。

CWD 指令将 AX 的最高有效位 D_{15} 扩展形成 DX，即：若 AX 的最高有效位为 0，则 DX＝0000H；若 AX 的最高有效位为 1，则 DX＝FFFFH。该指令在执行后，AX 不变。

符号扩展指令常用来获得除法指令所需要的被除数。例如，AX＝FF00H 表示有符号数－256；执行 CWD 指令后，则 DX＝FFFFH，DX、AX 仍表示有符号数－256。

对无符号数除法应该采用直接使高 8 位或高 16 位清 0 的方法，以获得倍长的被除数。

3.3.5 十进制调整指令

十进制数在计算机中也是用二进制来表示的，这就是二进制编码的十进制数——BCD 码。8086 支持压缩 BCD 码和非压缩 BCD 码，相应的十进制调整指令也分为压缩 BCD 码调整指令和非压缩 BCD 码调整指令。

1. DAA

DAA 是加法的十进制调整指令，必须跟在 ADD 或 ADC 指令之后使用。其功能是将存于 AL 中的 2 位 BCD 码加法运算的结果调整为 2 位压缩型十进制数，仍保留在 AL 中。

AL 中的运算结果在出现非法码(1010B～1111B)或本位向高位(指 BCD 码)有进位(由 AF＝1 或 CF＝1 表示低位向高位或高位向更高位有进位)时，由 DAA 自动进行加 6 调整。由于 DAA 指令只能对 AL 中的结果进行调整，因此，对于多字节的十进制加法，只能从低字节开始，逐个字节地进行运算和调整。例如，设当前 AX＝6698，BX＝2877，如果要将这两个十进制数相加，结果保留在 AX 中，则需要用下列几条指令完成。

```
ADD AL,BL                ;低字节相加
DAA                      ;低字节调整
MOV CL,AL
MOV AL,AH
ADC AL,BH                ;高字节相加
DAA                      ;高字节调整
MOV AH,AL
MOV AL,CL
```

2. DAS

DAS 是减法的十进制调整指令，必须跟在 SUB 或 SBB 指令之后，将 AL 寄存器中的减法运算结果调整为 2 位压缩型十进制数，仍保留在 AL 中。

减法是加法的逆运算，对减法的调整操作是减 6 调整。

3. AAA

AAA 是加法的 ASCII 码调整指令，也是只能跟在 ADD 指令之后使用。其功能

是将存于 AL 寄存器中的一位 ASCII 码数加法运算的结果调整为一位非压缩型十进制数，仍保留在 AL 中；如果向高位有进位(AF＝1)，则进到 AH 中。调整过程与 DAA 相似。

4. AAS

AAS 是减法的 ASCII 码调整指令，也必须跟在 SUB 或 SBB 指令之后，用来将 AL 寄存器中的减法运算结果调整为一位非压缩型十进制数；如果有借位，则保留在借位标志 CF 中。

5. AAM

AAM 是乘法的 ASCII 码调整指令。由于 8086/8088 指令系统中不允许采用压缩型十进制数乘法运算，故只设置了一条 AAM 指令，用来将 AL 中的乘法运算结果调整为两位非压缩型十进制数，其高位在 AH 中，低位在 AL 中。参加乘法运算的十进制数必须是非压缩型，故通常在 MUL 指令之前安排两条 AND 指令。例如：

```
AND AL,0FH
AND BL,0FH
MUL BL
AAM
```

执行 MUL 指令的结果，会在 AL 中得到 8 位二进制数结果，用 AAM 指令可将 AL 中的结果调整为两位非压缩型十进制数，并保留在 AX 中。其调整操作是：将 AL 寄存器中的结果除以 10，所得商数即为高位十进制数置入 AH 中，所得余数即为低位十进制数置入 AL 中。

6. AAD

AAD 是除法的 ASCII 码调整指令。它与上述调整指令的操作不同，是在除法之前进行调整操作的。

AAD 指令的调整操作是将累加器 AX 中的两位非压缩型十进制的被除数调整为二进制数，保留在 AL 中。其具体做法是将 AH 中的高位十进制数乘以 10，与 AL 中的低位十进制数相加，结果保留在 AL 中。例如，一个数据为 67，用非压缩型 BCD 码表示时，则 AH 中为 00000110，AL 中为 00000111；调整时执行 AAD 指令，该指令将 AH 中的内容乘以 10，再加到 AL 中，故得到的结果为 43H。

3.4 逻辑运算和移位循环类指令

这类指令可分为 3 种类型：逻辑运算、移位、循环。

3.4.1 逻辑运算指令

1. AND d,s;d←d∧s,按位“与”操作

源操作数可以是 8/16 位通用寄存器、存储器操作数或立即数；目标操作数只允许是

通用寄存器或存储器操作数。例如：

```
AND AX,ALPHA
```

设当前 CS=2000H，IP=0400H，DS=1000H，AX=F0F0H，ALPHA 是数据段中偏移地址为 0500H 和 0501H 地址中的字变量 7788H 的名字。则执行该指令后，将累加器 AX 中的 F0F0H 与物理地址 10500H 和 10501H 地址中的数据字 7788H 进行逻辑“与”运算后得结果为 7080H，并把它送回 AX 寄存器中。

2. OR d,s;d←d∨s,按位“或”操作

源操作数与目标操作数的约定同 AND 指令。

3. XOR d,s;d←d⊕s,按位“异或”操作

源操作数与目标操作数的约定同 AND 指令。

4. NOT d;d←$\bar{d}$,按位取反操作

5. TEST d,s;d∧s,按位“与”操作，不送回结果

有关的约定和操作过程与 AND 指令相同，只是 TEST 指令不传送结果。

3.4.2 移位指令与循环移位指令

移位指令分为算术移位和逻辑移位。算术移位是对带符号数进行移位，在移位过程中必须保持符号不变；而逻辑移位是对无符号数移位，总是用 0 填补已空出的位。根据移位操作的结果置标志寄存器中的状态标志（AF 标志除外）。若移位位数是 1 位，移位结果使最高位（符号位）发生变化，则将溢出标志 OF 置 1；若移多位，则 OF 标志将无效。

循环移位指令是将操作数首尾相接进行移位，分为不带进位位与带进位位循环移位。这类指令只影响 CF 和 OF 标志。CF 标志总是保持移出的最后一位的状态。若只循环移 1 位，且使最高位发生变化，则 OF 标志置 1；若循环移多位，则 OF 标志无效。

所有移位与循环移位指令的目标操作数只允许是 8/16 位通用寄存器或存储器操作数，指令中的 count（计数值）可以是 1，也可以是 n（$n \leqslant 255$）。若移 1 位，指令的 count 字段直接写 1；若移 n 位时，则必须将 n 事先装入 CL 寄存器中，故 count 字段只能书写 CL 而不能用立即数 n。例如：

```
SAL BX,1                ;BX 的内容算术左移 1 位
ROR AX,1                ;AX 的内容循环右移 1 位
```

又例如：

```
MOV CL,6
SAR DX,CL               ;DX 的内容算术右移 6 位
RCL AX,CL               ;AX 的内容连同 CF 循环左移 6 位
```

3.5 串操作类指令

串操作类指令是唯一能在存储器内的源与目标之间进行操作的指令。

串操作指令对向量和数组操作提供了很好的支持，可有效地加快处理速度、缩短程序长度。它们能对字符串进行各种基本的操作，如传送(MOVS)、比较(CMPS)、搜索(SCAS)、读(LODS)和写(STOS)等。对任何一个基本操作指令，可以用加一个重复前缀指令来指示该操作要重复执行，所需重复的次数由 CX 中的初值确定。被处理的串长度可达 64KB。

为缩短指令长度，串操作指令均采用隐含寻址方式，源数据串一般在当前数据段中，即由 DS 段寄存器提供段地址，其偏移地址必须由源变址寄存器 SI 提供；目标串必须在附加段中，即由 ES 段寄存器提供段地址，其偏移地址必须由目标变址寄存器 DI 提供。

为加快串操作的执行，可在基本串操作指令的前方加上重复前缀，共有无条件重复(REP)、相等/为 0 时重复(REPE/REPZ)和不等/不为 0 重复(REPNE/REPNZ)5 种重复前缀。带有重复前缀的串操作指令，每处理一个元素能自动修改 CX 的内容(按字节/字处理减 1/减 2)，以完成计数功能。当 CX≠0 时，继续串操作，直到 CX=0 才结束串操作。

无条件重复前缀(REP)常与串传送(MOVS)指令连用，完成传送整个串操作，即执行到 CX=0 为止。REPE 和 REPZ 具有相同的含义，只有当 ZF=1，且 CX≠0 时才重复执行串操作，常与串比较(CMPS)指令连用，比较操作一直进行到 ZF=0 或 CX=0 时为止。与此相反，REPNE 和 REPNZ 具有相同的含义，只有当 ZF=0，且 CX≠0 时才重复执行串操作，常与串搜索(SCAS)指令连用，搜索操作一直进行到 ZF=1 或 CX=0 为止。

串操作指令对 SI 和 DI 寄存器的修改与两个因素有关，一是和被处理的串是字节串还是字串有关；二是和当前的方向标志 DF 的状态有关。当 DF=0，表示串操作由低地址向高地址进行，SI 和 DI 内容应递增，其初值应该是源串和目标串的首地址；当 DF=1 时，则情况正好相反。

下面分别介绍 8086/8088 的 5 种基本的串操作指令。

3.5.1 MOVS 目标串，源串

串传送(MOVS)指令的功能：将由 SI 作为指针的源串中的一个字节或字，传送到由 DI 作为指针的目标串中，且相应地自动修改 SI/DI，使之指向下一个元素。如果加上 REP 前缀，则每传送一个元素，CX 自动减 1，直到 CX=0 为止。例如：

```
REP MOVSB
```

设当前 CS=6180H，IP=120AH，DS=1000H，SI=2000H，ES=3000H，DI=1020H，CX=0064H，DF=0。则该指令执行后，将源串的 100 个字节传送到目标串，每传送一个字节，SI+1，DI+1，CX−1，直到 CX=0 为止。

3.5.2 CMPS 目标串,源串

串比较(CMPS)指令的功能：将由 SI 作为指针的源串中的一个元素减去由 DI 作为指针的目标串中相对应的一个元素,不回送结果,只根据结果特征置标志位;并相应地修改 SI 和 DI 内容指向下一个元素。通常,在 CMPS 指令前加重复前缀 REPE/REPZ,用来确定两个串中的第一个不相同的数据。

3.5.3 SCAS 目标串

串搜索(SCAS)指令的功能：用来从目标数据串中搜索(或查找)某个关键字,要求将待查找的关键字在执行该指令之前事先置入 AX 或 AL 中,取决于 W=1 或 0。

搜索的实质是将 AX 或 AL 中的关键字减去由 DI 所指向的数据段目标数据串中的一个元素,不传送结果,只根据结果置标志位,然后修改 DI 的内容指向下一个元素。通常,在 SCAS 前加重复前缀 REPNE/REPNZ,用来从目标数据串中寻找关键字,操作一直进行到 ZF=1(查到了某关键字)或 CX=0(终未查找到)为止。

3.5.4 LODS 源串

读串(LODS)指令的功能：用来将源串中由 SI 所指向的元素取到 AX/AL 寄存器中,修改 SI 的内容指向下一个元素。该指令一般不加重复前缀,常用来和其他指令结合起来完成复杂的串操作功能。

3.5.5 STOS 目标串

写串(STOS)指令的功能：用来将 AX/AL 寄存器中的一个字或字节写入由 DI 作为指针的目标串中,同时修改 DI 以指向串中的下一个元素。该指令一般不加重复前缀,常与其他指令结合起来完成较复杂的串操作功能。若利用重复操作,可以建立一串相同的值。

3.6 程序控制指令

程序控制指令就是用来控制程序流向的一类指令。本节介绍无条件转移、条件转移、循环控制和中断 4 种程序控制指令。

3.6.1 无条件转移指令

在无条件转移类指令中,除介绍无条件转移指令 JMP 外,也一并介绍无条件调用过

程指令 CALL 与从过程返回指令 RET,因为,后两条指令在实质上也是无条件地控制程序流向的转移的,但它们在使用上与 JMP 有所不同。

1. JMP 目标标号

JMP 指令允许程序流无条件地转移到由目标标号指定的地址,去继续执行从该地址开始的程序。

转移可分为段内转移和段间转移两类。段内转移是指在同一代码段的范围之内进行转移,此时,只需要改变指令指针 IP 寄存器的内容,即用新的转移目标地址(指偏移地址)代替原有的 IP 值就可实现转移;而段间转移则是要转移到一个新的代码段去执行指令,此时不仅要修改 IP 的内容,还要修改段寄存器 CS 的内容才能实现转移。当然,此时的转移目标地址应由新的段地址和偏移地址两部分组成。根据目标地址的位置与寻址方式的不同,JMP 指令有以下 4 种基本格式。

1) 段内直接转移

段内直接转移是指目标地址就在当前代码段内,其偏移地址(即目标地址的偏移量)与本指令当前 IP 值(即 JMP 指令的下一条指令的地址)之间的字节距离即位移量将在指令中直接给出。此时,目标标号偏移地址为:

目标标号偏移地址=(IP)+指令中位移量

式中,(IP)是指 IP 的当前值。位移量的字节数则根据微处理器的位数而定。

对于 16 位微处理器而言,段内直接转移的指令格式又分为 2 字节和 3 字节两种,它们的第 1 字节是操作码,而第 2 字节或第 2、3 字节为位移量(最高位为符号位)。若位移量只有一个字节,则称为段内短转移,其目标标号与本指令之间的距离不能超过+127 和-128 字节范围;若位移量占两个字节,则称为段内近转移,其目标标号与本指令之间的距离不能超过±32KB 范围。注意,段的偏移地址是周期性循环计数的,这意味着在偏移地址 FFFFH 之后的一个位置是偏移地址 0000H。由于这个原因,如果指令指针 IP 指向偏移地址 FFFFH,而要转移到存储器中的后两个字节地址,则程序流将在偏移地址 0001H 处继续执行。

2) 段内间接转移

段内间接转移是一种间接寻址方式,是将段内的目标地址(指偏移地址或按间接寻址方式计算出的有效地址)先存放在某通用寄存器或存储器的某两个连续地址中,这时指令中只需给出该寄存器号或存储单元地址即可。例如:

```
JMP BX
```

此指令中的 BX 没有方括号[],但仍表示间接指向内存区的某地址单元。BX 中的内容即转移目标的偏移地址。设当前 CS=1200H,IP=2400H,BX=3502H,则该指令执行后,BX 寄存器中的内容 3502H 取代原 IP 值,CPU 将转到物理地址 15502H 单元中去执行后续指令。

注意:为区分段内的短转移(位移量为 8 位)和近转移(位移量为 16 位),其指令格式常以 JMP SHORT ABC 和 JMP NEAR PTR ABC 的汇编语言形式表示。

3) 段间直接转移

段间转移是指程序由当前代码段转移到其他代码段,由于其转移的范围超过

±32KB,故段间转移指令也称为远转移。在远转移时,目标标号是在其他代码段中,若指令中直接给出目标标号的段地址和偏移地址,则构成段间直接转移指令。例如:

```
JMP FAR PTR ADDR2
```

这是一条段间直接远转移指令,ADDR2 为目标标号。设当前 CS=2100H,IP=1500H,目标地址在另一代码段中,其段地址为 6500H,偏移地址为 020CH,则该指令执行后,CPU 将转移到另一代码段中物理地址为 6520CH 目标地址中去执行后续指令。

一般来说,在执行段间直接(远)转移指令时,目标标号的段内偏移地址送入 IP,而目标标号所在段的段地址送入 CS。在汇编语言中,目标标号可使用符号地址,而机器语言中则要指定目标(或转向)地址的偏移地址和段地址。

4) 段间间接转移

段间间接转移是指以间接寻址方式来实现由当前代码段转移到其他代码段。例如:

```
JMP DWORD PTR [BX+ADDR3]
```

设当前 CS = 1000H, IP = 026AH, DS = 2000H, BX = 1400H, ADDR3 = 020AH, (2160AH)=0EH,(2160BH)=32H,(2160CH)=00H,(2160DH)=40H,则执行指令时,目标地址的偏移地址 320EH 送入 IP,而其段地址 4000H 送入 CS,于是,该指令执行后,CPU 将转到另一代码段物理地址为 4320EH 的单元中去执行后续程序。

需要指出的是,段间转移和段内间接转移都必须用无条件转移指令,而条件转移指令则只能用段内直接寻址方式,并且,其转移范围只能是本指令所在位置前后的-128～+127 个字节。

2. CALL 过程名

这是无条件调用过程指令。

"过程"即"子程序";调用过程也即调用子程序。CALL 指令将迫使 CPU 暂停执行调用程序(或称为主程序)后续的下一条指令(即断点),转去执行指定的过程;待过程执行完毕,再用返回指令 RET 将程序返回到断点处继续执行。

8086/8088 指令系统中把处于当前代码段的过程称为近过程,用 NEAR 表示,而把其他代码段的过程称为远过程,用 FAR 表示。当调用过程时,如果是近过程,只需将当前 IP 值入栈;如果是远过程,则必须将当前 CS 和 IP 的值一起入栈。

CALL 指令与 JMP 类似,也有 4 种不同的寻址方式和 4 种基本格式,举例如下。

1) CALL N_PROC

N_PROC 是一个近过程名,采用段内直接寻址方式。

执行段内直接调用指令 CALL 时,第 1 步操作是把过程的返回地址(即调用程序中 CALL 指令的下一条指令的地址)压入堆栈中,以便过程返回调用程序(主程序)时使用。第 2 步操作则是转移到过程的入口地址去继续执行。指令中的近过程名将给出目标(转向)地址(即过程的入口地址)。

2) CALL BX

这是一条段内间接寻址的调用过程指令,事先已将过程入口的偏移地址置入 BX 寄