第 3 章　指令系统与寻址方式

3.1　8086 寻址方式

3.1.1　指令

指令是计算机执行的某一功能操作的表征。在硬件上，如图 3.1（a）所示，指令就是完成某个要求计算机执行的功能操作所需要的一些控制信号，这些控制信号按照一定的节拍次序控制计算机各单元协调工作。

指令在形式上是以二进制编码表示的，这就是指令编码，如图 3.1（b）所示。指令的二进制编码可以被控制器译码并产生相应的控制信号，这就是指令的执行。因为指令的二进制编码是能被计算机所识别和执行的，所以称为机器指令。机器指令一般由操作码和操作数组成，操作码表示指令所要完成操作的功能和性质，操作数提供该操作的对象。

为了方便人对指令的使用，出现了指令的助记符形式。指令助记符是用人们容易理解和记忆的符号来表达指令，如图 3.1（c）所示的例子，“MOV”表示数据的“移动”。指令助记符形式既直观，又与指令是简单的对应关系，是人们用来表达指令和理解指令的主要形式。助记符语言，即汇编语言，是直接使用指令系统进行程序设计的主要方式。

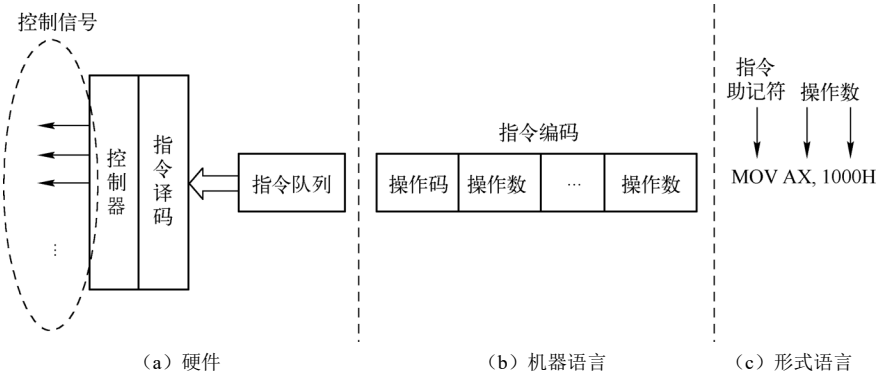

图 3.1　指令

3.1.2　操作数

指令根据指令中操作数的个数可分为无操作数指令、单操作数指令和双操作数指令，3 个操作数以上称为多操作数指令，如图 3.2 所示。操作数太多会加大控制系统的复杂性和指令编码的复杂性。在微机中，常见的是单操作数指令和双操作数指令。

在指令编码中的操作数字段表示的是操作数的存取方式，就是所谓的寻址方式，它要

通知 CPU 如何得到操作数：第一，数据的放在哪里；第二，应该如何存取数据。

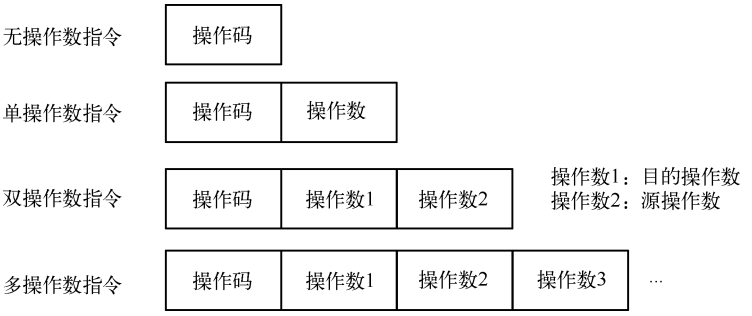

图 3.2　指令的操作数分类

关于第一个问题，操作数可以来源于或存储在以下三个地方。

（1）指令。操作数可以来自于指令。指令译码时如果发现操作数字段就是操作数，就将操作数直接经由内部总线送往目的。

（2）寄存器。寄存器是 CPU 内部的存储部件，常用于暂存中间结果，存取速度最快。

（3）存储器。存储器是数据最主要的存储区域。

第二个问题是针对操作数存放在存储器中而言的。此时，访问操作数依赖于操作数所在存储单元的地址——由段址和偏移地址组成的逻辑地址，这样就存在两方面的要求：提供段址和提供偏移地址。

（1）提供段址

段址总是由段寄存器提供的。正常情况下，如果段寄存器的使用符合 8086 默认的约定规则（详见第 2 章表 2.1），那么段寄存器就可以不指定。操作数也可以不存放在默认规定的段内，此时就需要指定段寄存器，称为段超越。

段超越在指令编码中是用段超越前缀表示的。在助记符形式中，段超越的格式是“<段寄存器>：操作数”。

【例 3.1】 分析 MOV AX , [1000h]和 MOV AX , CS : [1000h]的区别。

MOV AX , [1000h]　　MOV AX , CS : [1000h]

段址：DS　　段址：CS

偏移地址：1000h　　偏移地址：1000h

如表 3.1 所示是段寄存器的使用规则。

表 3.1　段寄存器的使用规则

存储器存取方式	默 认 段	段 超 越	偏 移 地 址
取指令	CS	无	IP
堆栈操作	SS	无	SP
源串	DS	无	SI
目的串	ES	无	DI
BP 作基址	SS	CS、DS、ES	寻址方式
通用数据读写	DS	CS、ES、SS	寻址方式

（2）提供偏移地址

偏移地址又称为有效地址（EA）。由表 3.1 可以看出，偏移地址由寻址方式指定。各种寻址方式提供各自不同的形成有效地址的方法。

3.1.3 寻址方式

1．立即寻址

立即寻址是指指令所需的操作数直接存放在指令中，即指令的操作数字段就是真正的操作数，这个操作数称为立即数。

如图 3.3 所示是一个立即寻址的示例。

立即寻址是一种特殊的寻址方式，只要取出指令，也就取出了操作数。但因为操作数是指令的一部分，不能更改，立即数的大小也受指令长度的限制，所以灵活性较差。立即寻址方式通常用于给寄存器赋初值，且仅用于源操作数。

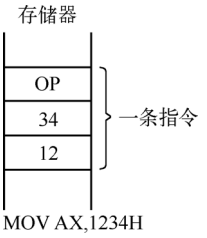

图 3.3　立即寻址的示例

2．寄存器寻址

寄存器寻址是指操作数存放在寄存器中，指令的操作数字段指出使用的寄存器。

例如：MOV AX，BX　；源操作数存放在 BX，目的操作数存放在 AX。

寄存器方式寻址编码简单，可以有效地缩短指令长度。使用寄存器存储操作数，可以利用寄存储的高速性质提高数据存取速度和指令执行速度。寄存器方式寻址的使用主要受限于寄存器的数量。

除立即寻址和寄存器寻址之外，以下介绍的其他几种寻址方式均属于存储器寻址，提供操作数的有效地址。

3．直接寻址

直接寻址是在指令中直接给出操作数所在内存单元的段内偏移量（即有效地址 EA）。如图 3.4（a）所示是一个直接寻址的示例。

例如：MOV AX，[2000h]；指源操作数存放存储器中，其偏移地址是 2000H，其寻址过程如图 3.4（b）所示。

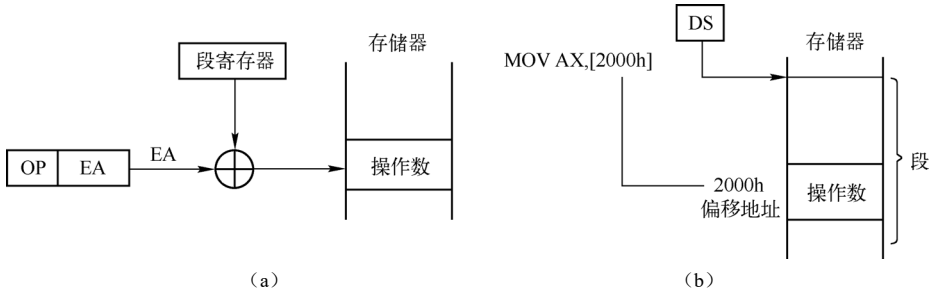

图 3.4　直接寻址的示例

直接寻址方式简单直观，但给出的操作数地址不能更改。

在汇编语言中，中括号“[”和“]”之中表示地址的概念，这一点在其他寻址方式中也是一样的。比较 MOV AX, 2000h 和 MOV AX, [2000h]的区别是：前一个是立即寻址，2000h 是立即数，而后一个是直接寻址，2000h 是操作数的有效地址 EA。

在汇编语言中，一般使用符号来代替地址的，这也就是汇编语言提供的变量功能。例如，如果已经在数据段定义了变量 VALUE，就可直接使用变量名来表示地址的概念，如 MOV AX, VALUE，汇编程序在将这条指令翻译为机器指令时，将会用 VALUE 真正的有效地址[XXXX]来替换 VALUE，从而变成 MOV AX, [XXXX]的形式。

4．基址寻址

基址寻址是在指令中直接由基址寄存器 BX 或 BP 提供操作数的有效地址，或者由 BX 或 BP 提供的基址与指令中提供的位移量之和形成操作数的有效地址，其寻址过程如图 3.5 所示。

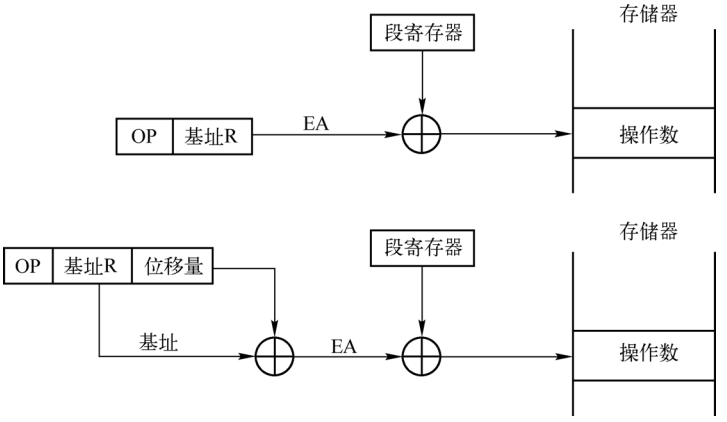

图 3.5　基址寻址过程

例如：　MOV AX, [BX]　　；BX 提供的是操作数的有效地址

　　MOV AX, [BP+1]　；BP 提供基址，它加上位移量 1 构成操作数的有效地址

注意：使用基址寄存器 BP 进行间接寻址时，默认的段寄存器是 SS。

在汇编语言中，可以使用变量名表达位移量。例如，如果已经定义了变量 VALUE，则可用 MOV AX, VALUE[BX]形式的基址寻址方式，也可以写成 MOV AX, [BX+VALUE]，还有 MOV AX, VALUE[BX+1]也是合法的。

另外，“没有位移量”和“位移量=0”的逻辑功能是一样，但它们的指令编码是不同的，后者需要在指令编码中给出位移量 0，所以位移量为 0 时应尽量写成无位移量的形式。

5．变址寻址

变址寻址是在指令中直接由变址寄存器 SI 或 DI 提供操作数的有效地址，或者由 SI 或 DI 提供的变址与指令中提供的位移量之和形成操作数的有效地址，其寻址过程如图 3.6 所示。

例如：　MOV AX, [SI]　　；SI 提供的是操作数的有效地址

　　MOV AX, [DI+2]　；DI 提供变址，它加上位移量 2 构成操作数的有效地址

6．基址变址寻址

基址变址寻址可以形象地看成是基址加变址的寻址方式，它是由基址寄存器 BX 或 BP 提供的基址加上由 SI 或 DI 提供的变址形成的操作数有效地址，或者再加上一个位移量，其寻址过程如图 3.7 所示。

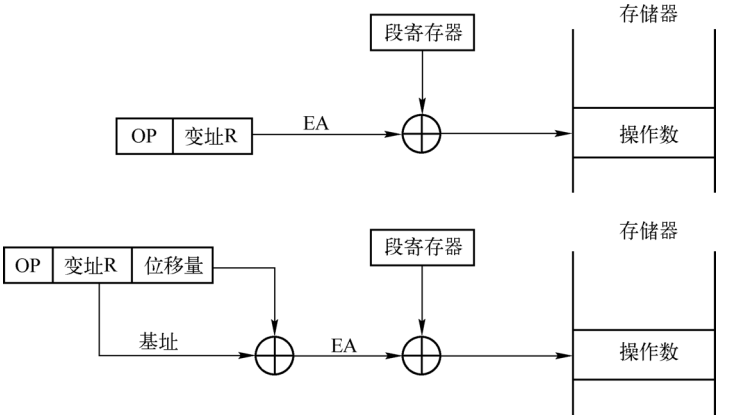

图 3.6　变址寻址过程

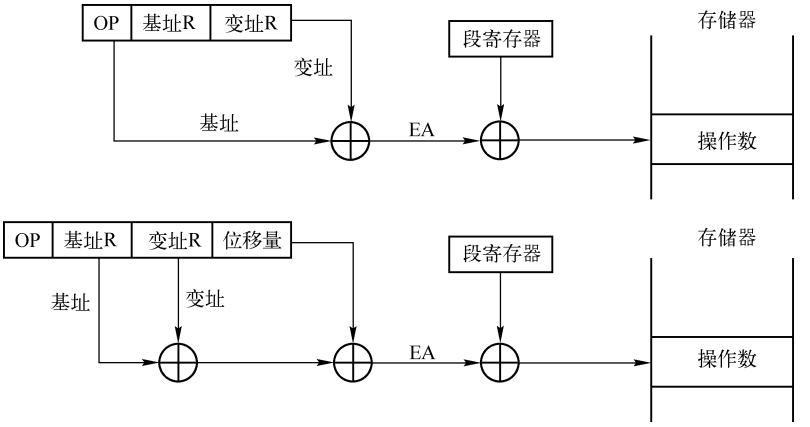

图 3.7　基址变址寻址过程

例如：　MOV AX, [BX][SI]　　；有效地址=基址(BX)+变址(SI)
　　MOV AX, [BX+SI]　　；另一种写法
　　MOV AX, [BP][DI+30h]　　；有效地址=基址(BP)+变址(DI)+位移量 30H
　　MOV AX, [BP+DI+30h]　　；另一种写法
　　MOV AX, 30h[BP][DI]　　；另一种写法

基址寻址、变址寻址和基址变址寻址都属于寄存器间接寻址方式，通过对寄存器内容的修改就可改变地址，显然比直接寻址方式灵活得多，也可以更加方便地表达数据间的关系。

【例 3.2】 使用直接寻址和基址寻址访问图 3.8 所示数组。

如图 3.8 所示，定义字数组 ARRAY。

（1）使用直接寻址访问数组

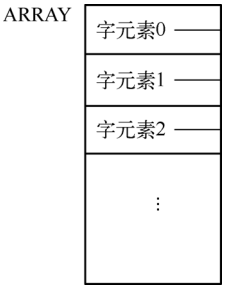

图 3.8　字数组

本例假定为与 MOV 指令配合，其访问指令如下。

访问元素 0：MOV AX, ARRAY

访问元素 1：MOV AX, ARRAY + 2

访问元素 2：MOV AX, ARRAY + 4

（2）使用基址寻址访问数组

本例假定为与 MOV 指令配合访问，其访问指令为：MOV AX, ARRAY[BX]

访问元素 0 时：(BX)=0

访问元素 1 时：将 BX 加 2，即(BX)=2

访问元素 2 时：将 BX 再加 2，即(BX)=4

可以将 BX 与数组下标联系起来。

7．隐含寻址

隐含寻址又称为固定寻址，操作数在指令中没有体现，而是隐含在固定位置上。

例如：　PUSHF　；操作数是 Flags 寄存器

　　MOVS　；源串在 DS：SI 中，目的串在 ES：DI 中

　　MUL BL ；被乘数在 AL 中，商在 AX 中

在微机中，采用隐含寻址的目的在于：①减少操作数个数，以便简化指令格式；②降低控制系统的复杂度，加快指令执行的速度。

3.2　8086 指令格式

8086 采用变长指令格式，指令由 1～6 字节构成，如图 3.9 所示。

指令包含以下字段：

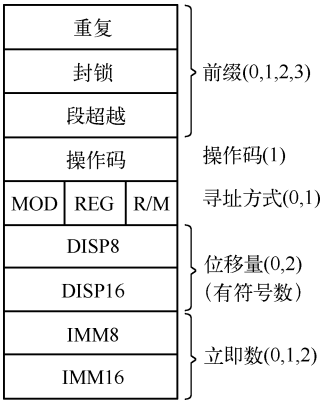

图 3.9　指令格式

1．指令前缀

前缀字段是可选字段，用来修改指令操作的某些属性。包括以下三方面内容：

① 段超越前缀：指定使用某个段寄存器代替指定中默认的段寄存器；

② 重复前缀：用于重复串的操作；

③ 封锁前缀：用于产生 Lock 信号，在指令执行期间封锁总线。

2．操作码

操作码确定指令所执行的操作。

3．寻址方式

寻址方式字段由 MOD、REG、R/M 组成，其中：REG 指定寄存器（双操作数指令中有一个操作数必须放在寄存器中）；MOD 与 R/M 确定另一个操作数，或者用于指定寄存器，或者用于确定存储器操作数的寻址方式。

4．位移量

当寻址方式中出现位移量时，位移量字段用来指定 8 位或 16 位有符号数表示的位移量。

5．立即数

在立即寻址方式中，立即数字段用来指定 8 位或 16 位的立即数。

如图 3.10 所示是 8086 一些常见的指令格式示例。

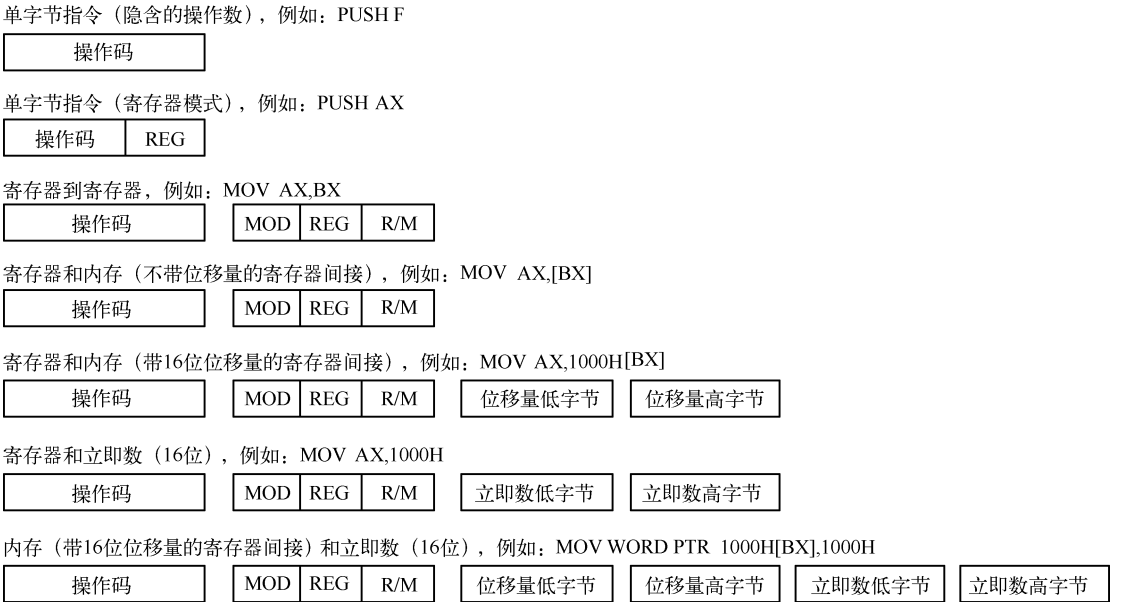

图 3.10　8086 一些常见的指令格式示例

3.3　8086 指令系统

8086 的指令系统从指令的功能上可以分为六大类：数据传送指令、算术运算指令、逻辑运算指令、串处理指令、控制转移指令和处理机控制指令。

3.3.1　数据传送指令

1．通用数据传送指令

（1）MOV

MOV 指令的格式、操作及寻址方式如表 3.2 所示。

表 3.2　MOV 指令

指　令	格　式	操　作	寻 址 方 式	标 志 位
MOV	MOV DST,SRC	（DST）←（SRC）	REG , REG MEM , REG REG , MEM	无影响

			REG , IMM MEM , IMM	

说明：SRC 是源操作数，DST 是目的操作数，REG 是寄存器，MEM 是内存操作数，IMM 是立即数；另外，在计算机文档中，常使用小括号表示引用内容或值。

MOV 指令完成源操作数到目的操作数的传送。例如：

```
MOV   DS, AX      ;(DS)←(AX)        将 AX 中的内容传送到 DS
MOV   CH, 20      ;(CH)←20          将 20 传送到 CH，结果 CH 中的内容为 20
MOV   BL, AH      ;(BL)←(AH)        将 AH 中的内容传送到 BL
MOV   AX, [BX]    ;(AX)←((BX))      将基址寻址得到的内存中的字传送到 AX
MOV   [DI], AX    ;((DI))←(AX)      将 AX 中的内容传送到变址寻址确定的内存中
MOV   CL, AX      ;×                字节和字长度不符
```

通过上面指令，可以看出 MOV 指令的以下两个特点：

① 数据传送类指令不需要经过 ALU 处理，所以对标志位没有影响（除 POPF 指令外）。

② MOV 指令是双操作数指令。在 8086 系统中，双操作数的使用是有限制的，如图 3.11 所示。

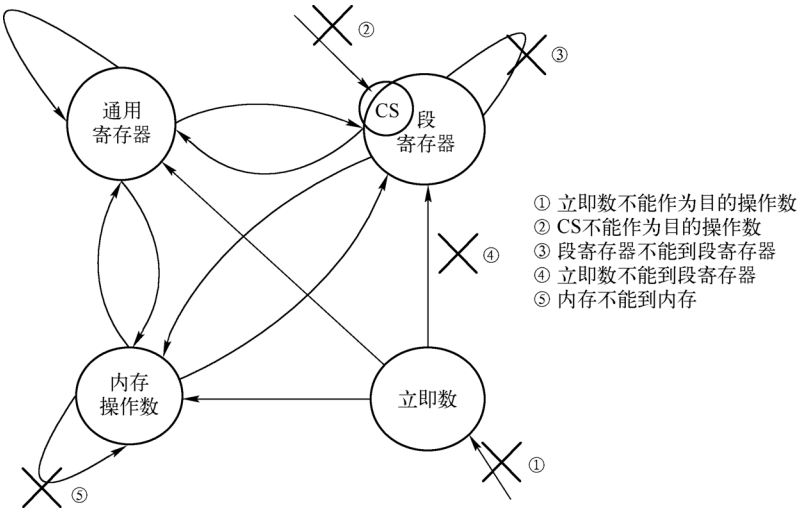

图 3.11　双操作数的使用规定

由图 3.11 可知，以下 MOV 指令是错误的：

```
MOV 1, AX                    ; ×，立即数不能作为目的操作数
MOV CS, AX                   ; ×，CS 不能作为目的操作数
MOV ES, DS                   ; ×，段寄存器不能到段寄存器
MOV DS,2000h                 ; ×，立即数不能到段寄存器
MOV byte ptr [BX],[1000h]    ; ×，内存不能到内存。
```

（2）XCHG

XCHG 指令的格式、操作及寻址方式如表 3.3 所示。

表 3.3　XCHG 指令

指　令	格　式	操　作	寻址方式	标志位
XCHG	XCHG OPR1,OPR2	（OPR1）↔（OPR2）	REG, REG MEM, REG REG, MEM	无影响

XCHG 指令完成两个操作数的交换，两个操作数既是源又是目的。例如：

```
XCHG    AX，BX          ; (AX) ↔ (BX)
XCHG    BX，[1000H]     ; (BX) ↔ (1000h)
XCHG    [2100H]，DH     ; (2100h) ↔ (DH)
```

2．堆栈指令

8086 的堆栈采用存储器方式实现，8086 将堆栈组织为一个逻辑段，称为堆栈段。堆栈段的段址由段寄存器 SS 指定，堆栈指针由 SP 寄存器担当，如图 3.12 所示。

8086 的堆栈有以下特点：

① 堆栈指针寄存器 SP 存放的是栈顶的地址，即总指向最后入栈数据的存储单元；

② 堆栈操作以字为单位进行；

③ 堆栈的栈底在低地址端，堆栈的增长方向是由高地址向低地址增长。“入栈”时 SP 要先减 2，“出栈”时 SP 要加 2。

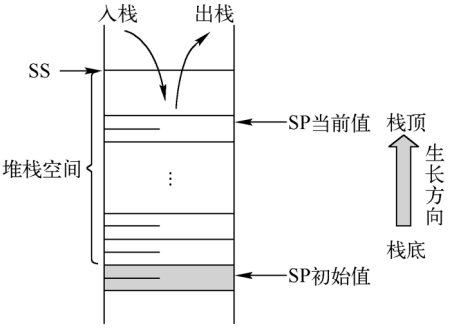

图 3.12　堆栈组织

（1）PUSH 和 POP

PUSH 是入栈指令，它执行两步工作：先修改堆栈指针，即将(SP)–2 送 SP；再将源操作数存储到栈顶（SP 指向）。源操作数可以是寄存器操作数或内存操作数，但不能是立即数。

POP 是出栈指令，它也是两步工作：先从栈顶（SP 指向）取一个字送到目的操作数，再改堆栈指针，即将(SP) + 2 送 SP。

PUSH 和 POP 指令的格式、操作及寻址方式如表 3.4 所示。

表 3.4　PUSH 和 POP 指令

指　令	格　式	操　作	寻址方式	标志位
PUSH	PUSH SRC	(SP) ← (SP)–2 ((SP)+1), (SP)) ← (SRC)	REG MEM	无影响
POP	POP DST	(DST) ← ((SP)+1), (SP)) (SP) ← (SP)+2		

【例 3.3】 设 SP=0008H、AX=55AAH，画出执行 PUSH AX 和 POP BX 的堆栈变化示意图。

执行 PUSH AX 指令，堆栈变化如图 3.13（a）所示。

执行 POP BX 指令，堆栈变化如图 3.13（b）所示。

堆栈是系统非常重要的结构，其“先入后出”或“后入先出”的特性使得凡是具有“保存”和“恢复”类似概念的行为都可以使用堆栈实现，许多有类似行为特点的系统操作（如子程序调用返回、中断等）也都依赖于堆栈的支持。所以，在应用时堆栈被破坏是非常

严重的错误，有以下几点需要注意。

① 按照存储对齐的原理，在堆栈初始化时，即给 SP 赋初值时，SP 的初值应该是偶数。

② 堆栈的大小（SP 的初值的大小）应设置得足够大，防止堆栈溢出。所谓堆栈溢出是指堆栈满（SP=0）时再入栈的情况。从 PUSH 的操作可以看出，8086 的入栈操作并没有判断溢出，机器认为此问题应由程序员负责。

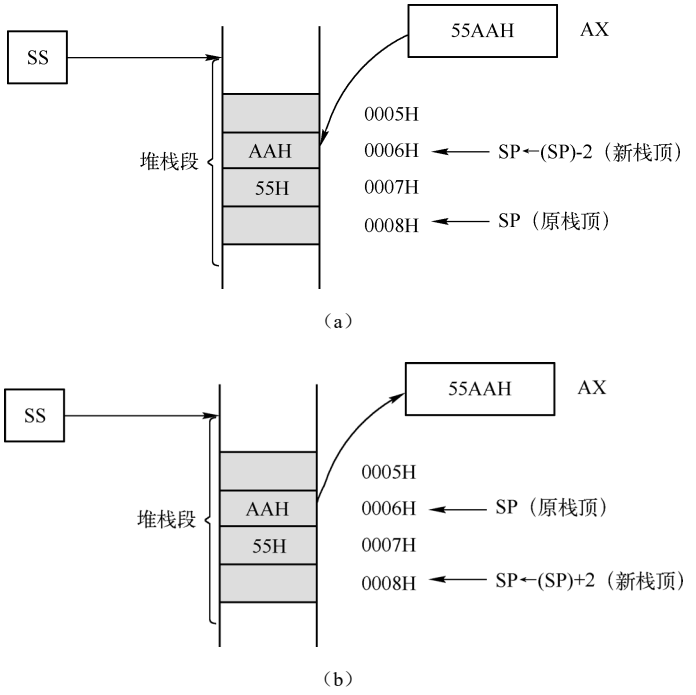

图 3.13　堆栈变化示意图

③ 堆栈的使用应严格遵循“平衡”原则，即入栈数和出栈数的平衡，或者说，要保证进入程序模块时的 SP 值应和退出程序模块时的 SP 值相同。

④ 注意堆栈的“先入后出”特性，即“保存”的次序和“恢复”的次序是相反的。

（2）PUSHF 和 POPF

PUSHF：标志寄存器入栈，用于保存标志寄存器内容。

POPF：标志寄存器出栈，即从堆栈中弹出一个字送到标志寄存器，用于恢复标志寄存器内容。

PUSHF 和 POPF 指令的格式、操作如表 3.5 所示。

表 3.5　PUSHF 和 POPF 指令

指　令	格　式	操　作	标 志 位
PUSH	PUSHF	(SP) ← (SP)-2 ((SP)+1), (SP)) ← (Flags)	无影响
POPF	POPF	(Flags) ← ((SP)+1), (SP)) (SP) ← (SP)+2	影响

3．地址传送指令

（1）LEA

LEA 指令的格式、操作及寻址方式如表 3.6 所示。

表 3.6　LEA 指令

指　令	格　式	操　作	寻 址 方 式	标　志　位
LEA	LEA REG , SRC	(REG) ← SRC	REG , MEM	无影响

LEA 指令是将源操作数的有效地址送到指定的寄存器，所以源操作数必然是内存操作数，而目的操作数是寄存器。

【例 3.4】 举例比较 MOV 指令和 LEA 指令。

如图 3.14 所示，MOV 指令传送的是操作数，MOV AX,[1000H]执行后(AX)=55AAH；LEA 指令传送的是操作数地址，LEA AX,[1000H]执行后(AX)=1000H。

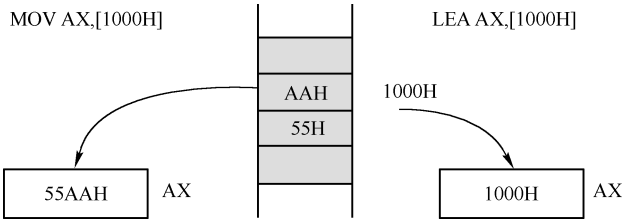

图 3.14　MOV 指令与 LEA 指令操作对比

（2）LDS

LDS 指令用于传送长指针。所谓长指针是指包含 16 位段址和 16 位偏移地址的完整逻辑地址，长指针在内存保存时占两个字，低字（低地址）保存 16 位偏移地址，高字（高地址）保存 16 位段址。LDS 指令将偏移地址送入指定的寄存器，段址送入 DS 段寄存器（隐含寻址）。LDS 指令的格式、操作及寻址方式如表 3.7 所示。

表 3.7　LDS 指令

指　令	格　式	操　作	寻 址 方 式	标　志　位
LDS	LEA REG , SRC	(REG) ← （SRC） (DS) ← （SRC+2）	REG , MEM	无影响

【例 3.5】 长指针 8000H:55AAH 保存在双字变量 PTR 中，用图例说明 LDS SI,PTR 指令的执行效果。

LDS，SI，PTR 的执行效果如图 3.15 所示。

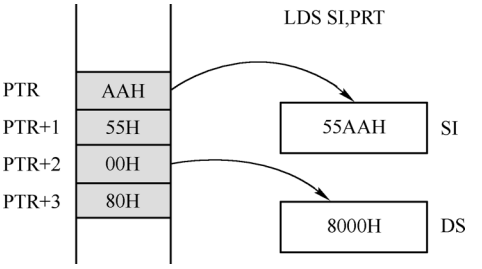

图 3.15　LDS 传送长指针的示例

4．类型转换指令

CBW 和 CWD 是符号扩展命令，符号扩展是将较少位数的有符号数扩展为较多位数的有符号数的方法。CBW 和 CWD 指令的格式、操作如表 3.8 所示。

表 3.8　CBW 和 CWD 指令

指　令	格　式	操　作	标 志 位
CBW	CBW	(AX) ←符号扩展— (AL)	无影响
CWD	CWD	(DX : AX) ←符号扩展— (AX)	

根据补码的性质，为了保证有符号数不变，扩展时需要在高位添加的二进制位由有符号数的符号位决定，正数时在高位添加的是全“0”，而负数时添加的是全“1”。

例如：

(AL)=127，即 01111111B，则执行 CBW 指令后，(AX)=0000000001111111B，仍然是 127；

(AL)=–127，即 10000001B，则执行 CBW 指令后，(AX)=1111111110000001B，也是–127。

5．输入/输出指令

I/O 指令是专门用于从 I/O 端口输入/输出数据的指令，包括 IN 和 OUT 指令，其指令的格式、操作及寻址方式如表 3.9 所示。

表 3.9　IN 和 OUT 指令

指　令	格　式	操　作	寻 址 方 式	标 志 位
IN	IN AL, <端口地址>	(AL)←(PORT)	AL, PORT(8 位) AL, DX	无影响
OUT	OUT <端口地址>, AL	(port)←(AL)	PORT(8 位), AL DX, AL	

I/O 指令有两个操作数，一个操作数是输入/输出的数据，必须放在 AL 寄存器中；另一个操作数是 I/O 端口的寻址方式，给出 I/O 端口地址。I/O 端口的寻址方式是在 I/O 端口空间的寻址，与存储空间的寻址方式是不同的，它有两种寻址方式。

（1）直接寻址

若 I/O 端口地址在 8 位以内（0～0FFH），该地址可以以常数的形式直接出现在指令中。

（2）间接寻址

若 I/O 端口地址超过 8 位，则必须用 DX 代替该地址出现，即(DX)表示 I/O 端口地址。当然，I/O 端口地址在 8 位以内时，也可用间接寻址。

例如：
```
IN AL, 20H    ; 从 20H 端口读取数据，存放在 AL 中
IN AL, DX     ; 从(DX)端口读取数据，存放在 AL 中
OUT 21H, AL ; 将(AL)从 21H 端口输出
OUT DX, AL  ; 将(AL)从(DX)端口输出
```

3.3.2　算术运算指令

1．加法指令

（1）ADD

ADD 指令执行加法运算，其格式、操作及寻址方式如表 3.10 所示。

表 3.10　ADD 指令

指　令	格　式	操　作	寻址方式	标志位
ADD	ADD DST,SRC	(DST)←(DST)+(SRC)	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	对所有状态标志位产生影响

从 ADD 指令的说明可以看出微处理器的两个原理：

① 被加数放在目的操作数中，求出的和又放回了目的操作数，这是计算机中为减少指令操作数常用的方法，已经成为所有运算类指令的统一标准；

② 加法操作是 ALU 的功能，凡是经过 ALU 运算处理的指令都会对状态标志位产生影响，一条指令对状态标志位的影响是该指令功能的重要方面。

ADD 指令对标志位的影响主要包括进位标志 CF、溢出标志 OF、零标志 ZF、符号标志 SF。执行加法指令时，CF 是根据是否产生了最高位的进位而设置的，当有进位时 CF=1，否则 CF=0；OF 的设置需要根据操作数的符号及其变化情况来设置：如果两个操作数的符号相同，而结果的符号与之相反，则 OF=1，否则 OF=0；ZF 和 SF 的设置相对简单，不再赘述。

ADD 指令应考虑溢出问题，即加出来的结果可能会超出计算机的表示范围。但是，对于无符号数的加法应该使用 CF 标志来判断是否溢出；而对于有符号数的加法应该使用 OF 标志来判断是否溢出。

（2）ADC

ADC 指令的格式、操作及寻址方式如表 3.11 所示。

表 3.11　ADC 指令

指　令	格　式	操　作	寻址方式	标志位
ADC	ADC DST,SRC	(DST)←(DST)+(SRC)+CF	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	对所有状态标志位产生影响

ADC 指令与 ADD 指令的区别在于：ADC 指令多加了一个 CF。虽然只多加了一个 CF，但就使得 8086 具备了对超字长数的处理能力。

【例 3.6】 求 10008432H + 20007F00H = ？

$$+\begin{array}{r}1\ 0008432\\ \underline{20007F00}\end{array}=\begin{array}{r}1\ 000\\ \underline{2000}\end{array}+\text{进位}+\begin{array}{r}8432\\ \underline{7F00}\end{array}$$，即先加低字，再加高字并低字加的进位。

```
所以：   MOV AX, 8432H   ; (AX) ←8432H
         ADD AX, 7F00H   ; (AX) ←(AX) +7F00H ，先加低字
                         ; (AX)= 0332H，CF=1
         MOV DX, 1000H   ; (DX) ←1000H
         ADC DX, 2000H   ; (DX) ←(DX) +2000H+CF，加高字并低字加的进位
                         ; (DX)=3001H
                         ; (DX:AX)=30010332H，双字结果
```

（3）INC

INC 是“加 1”指令，“加 1”操作经常在循环控制中使用。INC 指令的格式、操作及寻址方式如表 3.12 所示。

表 3.12　INC 指令

指　令	格　式	操　作	寻址方式	标志位
INC	INC DST	(DST)←(DST)+1	REG MEM	除 CF 外，对其他状态标志位均产生影响

“加 1”操作可以使用 ADD 指令实现，但 INC 指令更加优化，需要注意的是，INC 指令不影响 CF 标志。

2．减法指令

（1）SUB，SBB 和 DEC

减法运算在机器中实质上是补码的加法运算。因此，8086 类似于加法指令也安排了三条减法指令：SUB 实现减法；SBB 相比 SUB，多减去 CF（减法中的借位），用于超字长数的减法；DEC 实现“减 1”操作，不影响 CF。减法指令的格式、操作及寻址方式如表 3.13 所示。

表 3.13　SUB、SBB 和 DEC 指令

指　令	格　式	操　作	寻址方式	标志位
SUB	SUB DST,SRC	(DST)←(DST)–(SRC)	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	对所有状态标志位产生影响
SBB	SBB DST,SRC	(DST)←(DST)–(SRC)–CF		
DEC	DEC DST	(DST)←(DST)–1	REG MEM	除 CF 外，对其他状态标志位均产生影响

（2）CMP

8086 在减法类指令中还设计了 CMP 指令，称为“比较”指令。CMP 指令的格式、操作及寻址方式如表 3.14 所示。

表 3.14　CMP 指令

指　令	格　式	操　作	寻址方式	标志位
CMP	CMP DST,SRC	(DST)–(SRC)	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	对所有状态标志位产生影响

从 CMP 指令的操作说明中可以看出，它执行了目的操作数和源操作数的减法，但不保存减法结果，其目的只是为了得到操作数相减而产生的状态标志位。

例如：　CMP AX,1　　；　执行了(AX)–1，不保存结果，所以 AX 中的数没有变化，

　　；　但 Flags 寄存器中的标志位是(AX)–1 产生的。

在上例中，如果 ZF=1，表示(AX)–1 的结果是 0，也就说明了(AX)=1；因此 CMP AX,1 是为了通过 CMP 产生的标志位来观察(AX)和 1 之间的关系，称为“比较(AX)和 1”。

（3）NEG

NEG 是求补指令。从计算机原理上说，它执行了“求反加 1”的求补操作；从数学效果上说，它实现了算术求反，即给一个有符号数前面加一个负号。NEG 指令的格式、操作及寻址方式如表 3.15 所示。

表 3.15　NEG 指令

指　令	格　式	操　作	寻 址 方 式	标　志　位
NEG	NEG DST	(DST) ← －(DST)	REG MEM	均影响

3．乘法指令

乘法指令有两条，包括有符号数乘法指令（IMUL）和无符号数乘法指令（MUL），其格式、操作及寻址方式如表 3.16 所示。

表 3.16　MUL 和 IMUL 指令

指　令	格　式	操　作	寻 址 方 式	标　志　位
MUL	MUL SRC	无符号数乘法 (AX)←(AL)×8 位(SRC) 或 (DX:AX)←(AX)×16 位(SRC)	REG MEM	影响，只有 CF 和 OF 有意义 当结果的高半部分为零时，CF=OF=0；否则，为 1
IMUL	IMUL SRC	有符号数乘法 (AX)←(AL)×8 位(SRC) 或 (DX:AX)←(AX)×16 位(SRC)		当结果的高半部分是低半部分的符号扩展时，CF=OF=0；否则，为 1

计算机中的乘法器原理如图 3.16 所示。由图可以看出，乘法对于有符号数相乘和无符号数相乘是不同的，有符号数的乘法多了求绝对值和结果求补（符号相同不变，符号相异求补）两个工作。因此，在进行乘法运算时，程序员的责任首先是要选择是无符号数乘法还是有符号数乘法，即用 MUL 还是用 IMUL 指令。

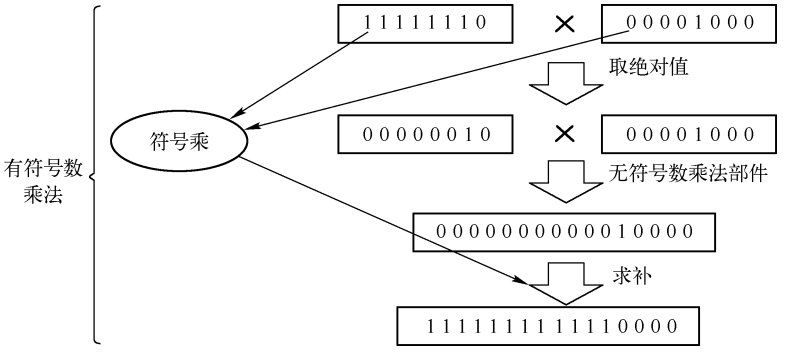

图 3.16　乘法器原理示意图

乘法指令的操作如图 3.17 所示。

图 3.17　乘法指令操作示意图

由图 3.17 可知，乘法指令具有以下特点：

① 乘法分为 8 位乘法和 16 位乘法，即 8 位被乘数×8 位乘数=16 位结果，16 位被乘数×16 位乘数=32 位结果；

② 乘法指令是典型的隐含寻址指令：8 位乘法的被乘数在 AL 中，指令给出 8 位乘数，16 位结果在 AX 中；16 位乘法的被乘数在 AX 中，指令给出 16 位乘数，32 位结果在 DX:AX 中；

③ 乘法指令不允许立即数，因为，可能无法根据乘数确定是 8 位乘法还是 16 位乘法；

④ 乘法指令影响标志位，但只有 OF 和 CF 有意义，表示是否产生了乘积的高半部分。可以由 OF 和 CF 确定 8 位乘法的乘积(AX)是 8 位（AH 内容不影响数值）还是 16 位；或 16 位乘法的乘积，(DX:AX)是 16 位的（DX 内容不影响的值）还是 32 位的。

例如：
```
MUL   BL  ; (AX)←(AL)×(BL)
IMUL  BX  ; (DX:AX)←(AX)×(BX)，程序员容易忘掉 DX 存放结果的高字
MUL   8   ; ×：不允许立即数
```

【例 3.7】 下面程序段实现 $X \cdot Y \cdot 5$，X、Y 为已定义的无符号字变量，该程序段是否正确？

```
MOV   AX,X
MOV   DX,5
MUL   Y
MUL   DX
```

该程序段错误，原因是：MUL Y 实现了(AX)乘上 Y，是 16 位乘法，得到 32 位的乘积，乘积的高 16 位放在 DX 中，低 16 位放在 AX 中。按照该程序段的指令，乘积的高 16 位将使 DX 原先存放的数 5 被冲掉了。

4．除法指令

除法指令与乘法指令一样也区分无符号数除法和有符号数除法指令，包括无符号数除法指令 DIV 和有符号数除法指令 IDIV，其格式、操作及寻址方式如表 3.17 和图 3.18 所示。

表 3.17　DIV 和 IDIV 指令

指　令	格　式	操　作	寻址方式	标　志　位
DIV	DIV SRC	无符号数除法 (AL)←(AX)÷8 位(SRC)的商 (AH)←(AX)÷8 位(SRC)的余数 或 (AX)←(DX:AX)÷16 位(SRC)的商 (AX)←(DX:AX)÷16 位(SRC) 的余数	REG MEM	有影响，无意义
I DIV	IDIV SRC	有符号数除法 (AL)←(AX)÷8 位(SRC)的商 (AH)←(AX)÷8 位(SRC)的余数 或 (AX)←(DX:AX)÷16 位(SRC)的商 (AX)←(DX:AX)÷16 位(SRC) 的余数		

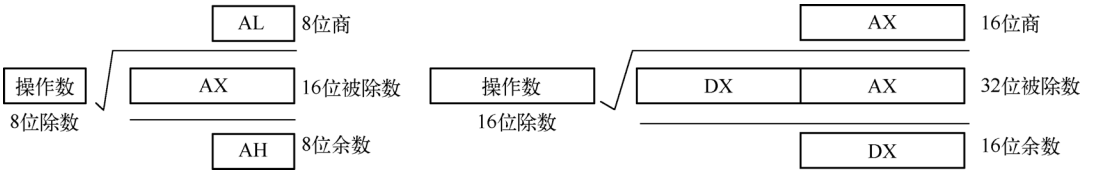

图 3.18　除法指令操作示意图

由图 3.18 可知，除法指令具有以下特点：

① 除法分为 8 位除法和 16 位除法，即 16 位被除数÷8 位除数=8 位商和 8 位余数，32 位被除数÷16 位除数=16 位商和 16 位余数；

② 除法指令也是使用隐含寻址的指令：8 位除法的 16 位被除数在 AX 中，指令给出 8 位除数，8 位商在 AL 中，8 位余数在 AH 中；16 位除法的 32 位被除数在 DX:AX 中，指令给出 16 位除数，16 位商在 AX 中，16 位余数在 DX 中；

③ 除法指令也不允许立即数。

【例 3.8】 求(AL) / 7 的值。

分为无符号数和有符号数两种情况处理。

（1）对于无符号数

```
MOV BL, 7       ;除法指令不允许立即数
MOV AH, 0       ;将 AL 扩展为 AX
DIV BL          ;计算(AX)÷BL
```

（2）对于有符号数

```
MOV BL, 7       ;除法指令不允许立即数
CBW             ;将 AL 符号扩展为 AX
IDIV BL         ;计算(AX)÷BL
```

【例 3.9】 求(AX) / (CX)的值。

分为无符号数和有符号数两种情况处理。

（1）对于无符号数

```
MOV DX, 0       ;将 AX 扩展为 DX:AX
DIV CX          ;计算(DX:AX)÷CX
```

（2）对于有符号数

```
CWD             ;将 AX 符号扩展为 DX:AX
IDIV CX         ;计算(DX:AX)÷CX
```

除法指令有可能会产生两个严重错误，一个是除零错误，另一个是除法溢出。对于除以零，在数学上就不允许。除法溢出指的是商超过了 AL（8 位除法）或 AX（16 位除法）能表示的范围。在上面两个例子中，如果忘记了“将 AL 扩展为 AX”或“将 AX 扩展为 DX:AX”就很容易产生除法溢出错误。

5．十进制调整指令

在计算机中还有一种用二进制编码表示十进制数的方法，就是 BCD 码。它用 4 位二进制数码表示一个十进制数码，如表 3.18 所示。

表 3.18　十进制数的 BCD 码表示法

十进制数码	0	1	2	3	4	5	6	7	8	9
BCD 码	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

在微机中表示十进制数的 BCD 码分为压缩 BCD 码和非压缩 BCD 码两种格式，前者使用 4 位二进制数码表示一个十进制数码，一个字节包含两个 BCD 码；后者使用 8 位表示一个十进制数码，低 4 位是 BCD 码，高 4 位无意义。

在微机中可以使用 BCD 码进行算术运算，其方法是，首先使用二进制算术运算指令进行二进制运算，然后使用十进制调整指令将二进制运算结果调整为 BCD 码。十进制调整指令包括压缩 BCD 码的加法调整指令 DDA 和减法调整指令 DDS，以及非压缩 BCD 码的加法调整指令 AAA、减法调整指令 AAS、乘法调整指令 AAM 和除法调整指令 AAD。

3.3.3　逻辑指令

1．逻辑运算指令

（1）AND、OR、NOT 和 XOR

逻辑运算指令执行按位的布尔逻辑运算，包括与（AND）、或（OR）、非（NOT）、异或（XOR），相应的指令分别是 AND、OR、NOT、XOR，其格式操作及寻址方式如表 3.19 所示。

表 3.19　AND、OR、NOT 和 XOR 指令

指　令	格　式	操　作	寻址方式	标志位
AND	AND DST,SRC	(DST)←(DST)∧(SRC)	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	影响 CF=0 OF=0
OR	OR DST,SRC	(DST)←(DST)∨(SRC)		
XOR	XOR DST,SRC	(DST)←(DST)⊕(SRC)		
NOT	NOT DST	(DST)←$\overline{(DST)}$	REG MEM	无

根据布尔逻辑运算的性质：0∧X=0、1∨X=1、1⊕X=$\overline{X}$，所以 AND 常用于“清位”，OR 常用于“置位”，XOR 常用来对位求反。

例如：　AND AL, 80H　　; (AL)与 80H，　(AL)=$D_7$0000000B

　　OR　BL, 80H　　; (BL)或 80H，　(BL)=1$D_6D_5D_4D_3D_2D_1D_0$B

　　XOR CL, 80H　　; (CL)异或 80H，　(CL)= $\overline{D_7}$ $D_6D_5D_4D_3D_2D_1D_0$B

NOT 指令是对操作数的按位求反，称为逻辑求反。而 NEG 指令是算术求反，两者是不同的。

【例 3.10】 对(DX:AX)中的双字进行算术求反。

```
NOT AX          ;
```

```
NOT DX          ; 对 DX:AX 中的双字求反
ADD AX,1        ;
ADC DX,0        ; 对 DX:AX 中的双字加 1
```

（2）TEST

TEST 指令只是执行两个操作数的与操作，但不保存结果，目的是为了得到与操作所产生的标志位。通过观察 TEST 产生的标志位，可以了解操作数的某位或某些位的状态。TEST 指令的格式、操作及寻址方式如表 3.20 所示。

表 3.20　TEST 指令

指　令	格　式	操　作	寻址方式	标　志　位
TEST	TEST DST,SRC	(DST)∧(SRC)	REG, REG REG, MEM MEM, REG REG, IMM MEM, IMM	影响 CF=0 OF=0

```
例如：  TEST AX, 8000H  ;  (AX)∧8000H（特征码，或称为掩码）
                        ;  如果 ZF=0，即结果非 0，则 AX 最高位是 1
                        ;  如果 ZF=1，即结果是 0，则 AX 最高位是 0
        TEST AL, 7      ;  (AX)∧7，如果 ZF=1，则 AL 的 0、1、2 位全为 0
```

2．移位指令

移位指令的指令格式是：移位指令 DST, 1/CL

其中，1/CL 表示移动的位数。如果只移 1 位，指令中可用立即数 1 指出；如果超过 1 位，必须用(CL)指明。

移位指令影响标志位 CF、OF、PF、SF 和 ZF。移位指令包括以下几种。

（1）逻辑左移 SHL 和算术左移 SAL

逻辑左移和算术左移的操作是：目的操作数左移，移出的最高位放入 CF 标志，空出的最低位填入 0，如图 3.19 所示。

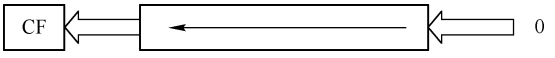

图 3.19　逻辑左移和算术左移

例如：假设(AX)=5555H，即 0101010101010101B，指行指令：

```
        SHL AX, 1
或      SAL AX, 1       ; (AX)左移一位
则      (AX)=0AAAAH，即 1010101010101010B。
```

左移 1 位相当于“乘以 2”，显然这比用乘法指令实现快得多。只有在移动 1 位数时 OF 才有效，表示移位前后最高位是否不同。当 OF=1 时，移位前后最高位不同，即有符号数“乘以 2”后溢出。

【例 3.11】 求(AX)×10 的值。

```
SHL     AX, 1     ; (AX)×2
MOV     BX,AX     ; (BX)=(AX)，即“原(AX)”×2
SHL     AX, 1     ; (AX)×2，即“原(AX)”×4
SHL     AX, 1     ; (AX)×2，即“原(AX)”×8
```

```
ADD     AX,BX     ; (AX)=(AX)+(BX) ，即“原(AX)”×8＋“原(AX)”×2
```

移位指令要注意寻址方式的规定。

例如：SHL AX, 2　　;×：用立即数表示只能是 1

```
MOV CL,2
SHL AX, CL     ; √：移动位数超过 1，要使用(CL)表示
```

（2）逻辑右移 SHR

逻辑右移的操作是：目的操作数右移，移出的最低位放入 CF 标志，空出的最高位填入 0，如图 3.20 所示。

（3）算术右移 SAR

算术右移的操作是：目的操作数右移，移出的最低位放入 CF 标志，最高位不变，如图 3.21 所示。

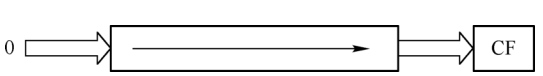

图 3.20　逻辑右移

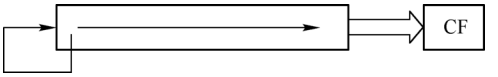

图 3.21　算术右移

右移 1 位相对于“除以 2”。逻辑右移是无符号数“除以 2”，而算术右移是有符号数“除以 2”，所以符号位保持不变。

例如：　SHR AX, 1　　；右移 1 位，无符号数(AX)除以 2

```
SAR BX, 1     ；右移 1 位，有符号数(BX)除以 2
```

（4）循环左移 ROL

循环左移的操作是：目的操作数左移，移出的最高位填回空出的最低位，并放入 CF 标志，如图 3.22 所示。

例如：假设(AX)=7AAEH，即 0111101010101110B，执行指令：

```
ROL AX, 1     ; (AX)循环左移一位
```

则(AX)=0F55CH，即 1111010101011100B，CF=0。

（5）循环右移 ROR

循环右移的操作是：目的操作数右移，移出的最低位填回空出的最高位，并放入 CF 标志，如图 3.23 所示。

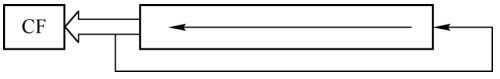

图 3.22　循环左移

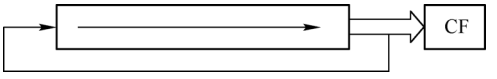

图 3.23　循环右移

（6）带进位循环左移 RCL

带进位循环左移的操作是：目的操作数左移，空出的最低位填入 CF 原值，移出的最高位放入 CF 标志，如图 3.24 所示。

例如：假设(AX)=7AAEH，即 0111101010101110B，CF=1，执行指令：

```
RCL AX, 1     ; (AX)  带进位循环左移一位
```

则(AX)=0F55DH，即 1111010101011101B，CF=0。

【例 3.12】 将(DX:AX)中的 32 位数左移 1 位。

```
SHL AX, 1     ; (AX)左移 1 位，最低位填入 0，原来的最高位被放入 CF
```

```
RCL DX, 1       ; (DX)带进位循环左移 1 位，最低位填入 CF 值，
                ; 即(AX)原来的最高位。
```

（7）带进位循环右移 RCR

带进位循环右移的操作是：目的操作数右移，空出的最高位填入 CF 原值，移出的最低位放入 CF 标志，如图 3.25 所示。

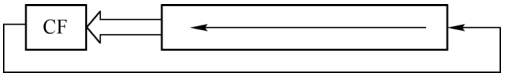

图 3.24　带进位循环左移

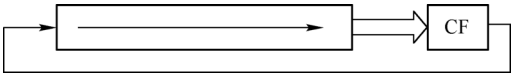

图 3.25　带进位循环右移

8086 的移位指令助记符都由三个字母组成，其命名规则如图 3.26 所示，可以参照图中方法帮助记忆。

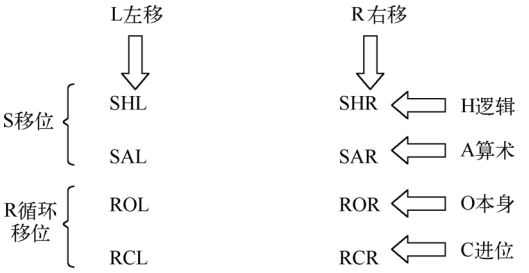

图 3.26　移位指令的命名规则

3.3.4　串指令

串指令包括串传送 MOVS、串比较 CMPS、串扫描 SCAS、串装入 LODS、串存储 STOS 等，是 8086 对数据块操作的支持。

串指令具有以下共同特点。

① DS:SI 寻址源串，ES:DI 寻址目的串。

② 每次操作后，SI、DI 自动修改，方向标志 DF 控制串的处理方向：

当 DF=0 时，SI 和 DI 加 1 或加 2；

当 DF=1 时，SI 和 DI 减 1 或减 2。

③ 串指令需要和重复前缀配合，实现重复操作。

下面以 MOVS 指令为例，介绍串指令的这些特点。

MOVS 指令的操作说明如表 3.21 所示。

表 3.21　MOVSB 和 MOVSW 指令

指　令	操　作	
	DF=0	DF=1
MOVSB	(i) ((ES:DI))←((DS:SI)) (ii) (SI)←(SI)+1，(DI)←(DI)+1	(i) ((ES:DI))←((DS:SI)) (ii) (SI)←(SI)–1，(DI)←(DI)–1
MOVSW	(i) ((ES:DI))←((DS:SI)) (ii) (SI)←(SI)+2，(DI)←(DI)+2	(i) ((ES:DI))←((DS:SI)) (ii) (SI)←(SI)–2，(DI)←(DI)–2

MOVS 指令执行两步工作：首先，将 DS:SI 确定字节或字传送到 ES:DI 指向的位置；

然后，将 SI 和 DI 加减 1 或 2。MOVS 指令有两种方式：字节方式（每次传送一个字节）和字方式（每次传送一个字），前者是 MOVSB（B 表示 BYTE）指令，MOVSB 指令自动将 SI 和 DI 加/减 1，后者是 MOVW（W 表示 WORD）指令，MOVSB 指令自动将 SI 和 DI 加/减 2。MOVS 指令还有两个处理方向：当 DF=0 时，是递增方向，MOVS 指令自动对 SI 和 DI 加 1 或 2；当 DF=1 时，是递减方向，MOVS 指令自动对 SI 和 DI 减 1 或 2。

REP 以指令前缀形式加在串指令前面，用于重复执行串指令，重复的次数由 CX 寄存器内容确定，其说明如表 3.22 所示。

将 REP 前缀和 MOVS 指令配合后，就获得了如图 3.27 所示的数据块描述。DS:SI 指向源数据块，ES:DI 指向目的数据块，CX 是数据块的长度（单位是字或字节）。

表 3.22　REP 前缀指令

格　式	操　作
REP 串指令	（1）若(CX)=0，退出重复操作； （2）若(CX)≠0，执行后面的串指令一次，并执行(CX)←(CX)−1； （3）转(1)

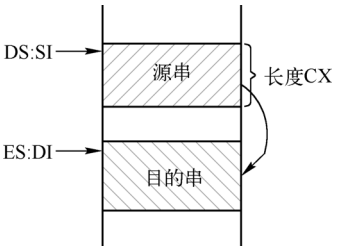

图 3.27　串寻址

【例 3.13】 在数据段中定义了两个数据块 SOURCE 和 DEST，数据块长度保存在变量 LEN 中，请用串指令实现 SOURCE 到 DEST 的传送。

本题有以下 4 种实现方法。

① 字节递增方式

```
MOV AX, DS
MOV ES, AX          ; (ES)←(DS)
LEA SI, SOURCE      ; DS:SI 指向源数据块
LEA DI, DEST        ; ES:DI 指向目的数据块
MOV CX, LEN         ; (CX)←数据块长度
CLD                 ; DF←0
REP MOVSB           ; 传送
```

② 字节递减方式

```
PUSH DS
POP ES              ; (ES)←(DS)
LEA SI, SOURCE
LEA DI, DEST
MOV CX, LEN         ; (CX)←数据块长度
ADD SI, CX
DEC SI              ; DS:SI 指向源数据块
ADD DI, CX
DEC DI              ; ES:DI 指向目的数据块
STD                 ; DF←1
REP MOVSB           ; 传送
```

如果数据长度是偶数，并且 SOURCE 和 DEST 的地址符合对齐原则（偶地址），则可以采用字传送方式，速度比字节方式更快。

③ 字递增方式

```
MOV AX, DS
MOV ES, AX          ; (ES)←(DS)
LEA SI, SOURCE      ; DS:SI 指向源数据块
LEA DI, DEST        ; ES:DI 指向目的数据块
MOV CX, LEN
SHR CX, 1           ; (CX)←数据块长度
CLD                 ; DF←0
REP MOVSW           ; 传送
```

④ 字递减方式

```
PUSH DS
POP ES              ; (ES)←(DS)
LEA SI, SOURCE
LEA DI, DEST
MOV CX, LEN
ADD SI, CX
SUB SI, 2           ; DS:SI 指向源数据块
ADD DI, CX
SUB DI, 2           ; ES:DI 指向目的数据块
SHR CX, 1           ; (CX)←数据块长度
STD                 ; DF←1
REP MOVSW           ; 传送
```

如图 3.28 所示，如果 SOURCE 和 DEST 有重叠，在图 3.28（a）的情况下应该使用递减方式传送，在图 3.28（b）的情况下应该使用递增方式传送，否则将出现冲掉数据的错误。

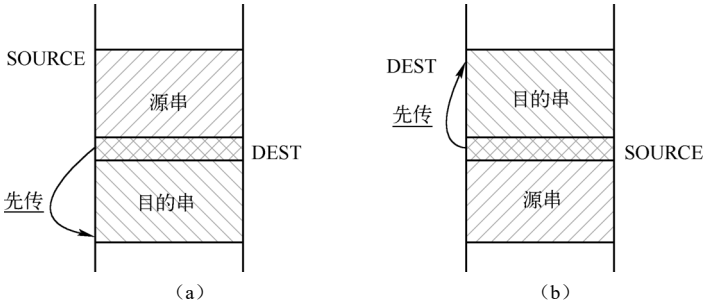

图 3.28　数据块重叠时的传送

3.3.5 控制转移指令

1．有关转移的概念

8086 的指令执行顺序是由代码段段寄存器 CS 和指令指针寄存器 IP 控制的。CS 指明下一条指令所处的段，IP 指定下一条指令的段内地址。CPU 取出指令执行后，会自动修改 IP 内容，将其加上所取指令的字节数，保证 IP 始终指向下一条指令。这就是 CPU 默认的顺序执行机制。

一般情况下，指令按照空间顺序逐条执行，但实际上程序不可能全部顺序执行，有时需要改变程序的执行流程，这就还需要有转移机制。所谓的转移就是，在执行一条控制转移指令后，即将执行的下一条指令将不再是空间顺序上的下一条指令，而是该控制转移指令指定位置上的指令。在 8086 中，各种转移指令不管功能如何，最终都是通过修改 IP 内容或 IP 及 CS 的内容实现转移的。

在 8086 系统中，根据转移指令和转移指令要转移到的目标位置之间的关系，把转移分为段内转移和段间转移。

在段内转移中，转移的目标地址在当前代码段内，CS 的内容不变，只需要将 IP 的内容修改为目标地址，其转移如图 3.29 所示。

在段间转移中，转移的目标地址不在本段，此时 CS 和 IP 的内容都需要修改，其转移如图 3.30 所示。

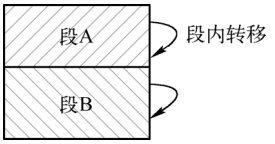

图 3.29　段内转移

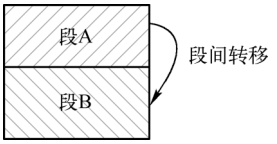

图 3.30　段间转移

转移指令的格式基本上都是“转移指令 操作数”的形式，转移指令需要根据操作数提供的数值来修改 IP 内容或 IP 及 CS 的内容。因此，又可根据转移指令中操作数的形式将转移分为直接转移和间接转移。直接转移是在指令中直接给出转移目标地址，属于立即寻址，在编程时直接用标号给出转移目标位置；而间接转移的目标地址是通过寄存器操作数或存储器操作数间接给出的。

因此，转移指令根据转移功能、转移形式的划分就可组合分为段内直接转移、段内间接转移、段间直接转移和段间间接转移 4 种类型，如表 3.23 所示。

表 3.23　转移指令的分类

转移功能 / 转移形式	段内转移	段间转移
直接转移	段内直接转移：通过立即数直接给出或形成一个 16 位地址用来修改 IP	段间直接转移：通过立即数直接给出 32 位地址用来修改 CS 和 IP
间接转移	段内间接转移：使用寄存器操作数或内存操作数给出一个 16 位地址用来修改 IP	段间间接转移：使用内存操作数给出 32 位地址用来修改 CS 和 IP

（1）段内直接转移

表 3.24 所示的两条指令都属于段内直接转移。第一条指令中给出了一个 16 位相对量，第二条指令中给出了一个 8 位的相对量，相对量表示转移的目标地址和当前地址（实际上是转移指令后面一条指令的地址）之间的差值，正值表示向后转移，负值表示向前转移，8 位相对量表示范围是–128～127，而 16 位的相对量表示的范围是–32 768～32 767。转移指令形成新的(IP)值的方法是：(IP)←（IP）+ 8 位（16 位）相对量。这种方法，如图 3.31 所示，可以说是一种特殊的寻址方式，称为相对寻址。

表 3.24　段内直接转移指令示例

指　令	编　码
JMP NEXT	JMP + 16 位相对量
JNZ NEXT	JNZ + 8 位相对量

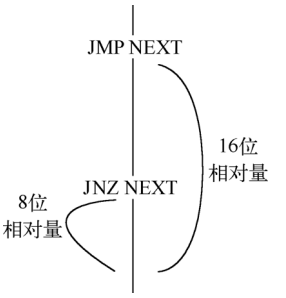

图 3.31　段内直接转移示例

（2）段内间接转移

段内间接转移可以用除立即寻址以外的其他任意一种寻址方式获得一个 16 位地址，用来修改(IP)。

例如：　JMP BX　　；寄存器寻址，(IP) ← (BX)

　　JMP WORD PTR [BX]　；基址寻址，(IP) ← ((BX))

（3）段间直接转移

段间直接转移使用指令直接给出的 16 位段址修改(CS)、16 位偏移地址修改(IP)。

例如：　JMP FAR PTR NEXT　; NEXT 是另一个段内的标号

这条指令属于段间直接转移，其编码形式是：JMP + 16 位偏移地址 + 16 位段址。

（4）段间间接转移

段间间接转移指令可以使用存储器寻址方式获得保存在存储器中的目标地址，目标地址包括 16 位段址和 16 位偏移地址、16 位段址修改(CS)、16 位偏移地址修改(IP)。如图 3.32 所示是段间间接转移。

例如：　JMP DWORD PTR [BX]

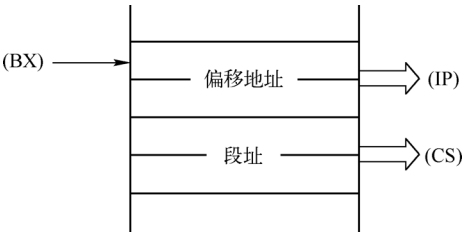

图 3.32　段间间接转移

2．无条件转移指令

无条件转移指令的格式、操作如表 3.25 所示。

表 3.25　JMP 指令

指　令	格　式	操　作
JMP	JMP OPR	程序无条件转移到 OPR 指定的位置继续执行

无条件转移指令对段内直接转移、段内间接转移、段间直接转移和段间间接转移均可以实现，例如：

（1）段内直接转移

```
JMP SHORT NEXT          ; NEXT，段内标号；SHORT，操作符，指明 8 位相对转移；
JMP NEAR PTR NEXT       ; NEXT，段内标号；NEAR PTR，操作符，指明 16 位相对转移；
```

（2）段内间接转移

```
JMP WORDR PTR [BX]      ; WORD PTR，操作符，指明字操作数
```

（3）段间直接转移

```
JMP FAR PTR ROUTE       ; ROUTE，段间标号；FAR PTR，操作符，指明段间转移；
```

（4）段间间接转移

```
JMP DWORD PTR [BX]      ; DWORD PTR，操作符，指明双字操作数
```

实际在编程时，最常用的是直接使用标号的直接转移，一般也不需要指定操作符的修饰，即直接用“JMP 标号”的形式即可，汇编程序会根据标号的性质自动选择合适的转移方式。

3．条件转移指令

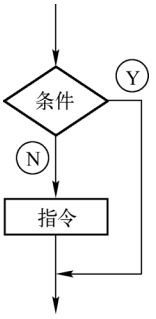

图 3.33　分支结构

条件转移指令用于支持程序最基本的结构——分支结构。

首先，所有条件转移指令的基本功能都是实现如图 3.33 所示的流程，当条件成立时进行转移，当条件不成立时不转移继续执行下条指令。

其次，所有条件转移指令的形式都是“转移指令 OPR（标号）”的形式，所有条件转移指令都用 8 位相对寻址实现段内直接转移，转移范围是–128～127。

所以，条件转移指令的差异就在于“条件”的差异。所谓的条件就是某个标志位或某几个标志位的状态。

（1）单一条件

单一条件是单个标志位所表示状态条件。5 个标志位 ZF、CF、OF、SF 和 PF，再加上它们的否定，可以构成 10 条条件转移指令，如表 3.26 所示。

表 3.26　单一条件的条件转移指令

标志位	条件			转移指令	
	标识	意义		指令	备注
ZF	Z	ZF=1	结果为零时条件成立	JZ	等价于 JE
	NZ	ZF=0	结果不为零时条件成立	JNZ	等价于 JNE
CF	C	CF=1	结果有进位时条件成立	JC	
	NC	CF=0	结果没有进位时条件成立	JNC	
OF	O	OF=1	结果溢出时条件成立	JO	
	NO	OF=0	结果不溢出时条件成立	JNO	
SF	S	SF=1	结果为负时条件成立	JS	
	NS	SF=0	结果非负时条件成立	JNS	
PF	P	PF=1	结果奇偶位为偶时条件成立	JP	
	NP	PF=0	结果奇偶位为奇时条件成立	JNP	

例如：　CMP AX, 1　　；执行(AX)–1 不保存结果
　　JZ Equal　　；如果结果为零(ZF=1)则转移到 Equal
　　……
　　Equal:

如果 JZ 的条件成立，即(AX)–1 的结果为零，就说明(AX)和 1 是相等关系。相等关系用字母“E”来标识，这样又产生了两条指令“JE”和“JNE”分别表示“相等则转移”和“不相等则转移”。“JE”和“JZ”、“JNE”和“JNZ”是同样的指令，只是使用不同的助记符而已。

在微机中，对两个数的关系判断总是通过检查两个数相减后结果的状态（状态标志位）获得的，这是计算机的一个基本方法和原理。

（2）复合条件

有时候，某些状态条件是单个标志位无法表示的，而需要多个标志位联合表示。常见的就是两个数之间的小于或大于关系。

如何得到两个数之间的“小于”或“大于”关系呢？这需要分为无符号数和有符号数来讨论。

对于无符号数，两个无符号数相减，“不够减”就是“小于”，“够减”就是“大于”。“不够减”可以用“CF=1”（产生了借位）表示；而“够减”就需要用“CF=0 并且 ZF=0”来表示。无符号数的“小于”用字母“B”标识，“大于”用字母“A”标识。

对于有符号数，两个有符号数相减，“结果为负”就是“小于”，“结果为正”就是“大于”。“结果为负”时 SF=1，但这还不够，有可能会出现一个负数减去一个正数产生溢出（下溢）而使结果为正的情况，所以“结果为负”使用“SF=1 且 OF=0”（结果为负，未溢出）或者“SF=0 且 OF=1”（结果为正，但下溢）表示；类似地，“结果为正”使用“SF=0 且 OF=0”（结果为正，未溢出）或“SF=1 且 OF=1”（结果为负但上溢），而且“ZF=0”（结果非 0）表示。有符号数的“小于”用字母“L”标识，“大于”用字母“G”标识。

复合条件的条件转移指令如表 3.27 所示。

表 3.27　复合条件的条件转移指令

类　型	条　件			转 移 指 令	
无符号数	小于	B	CF=1	JB	JNAE
	小于等于	BE	CF∨ZF=1	JBE	JNA
	大于	A	CF∨ZF=0	JA	JNBE
	大于等于	AE	CF=0	JAE	JNB
有符号数	小于	L	SF ⊕ OF=1	JL	JNGE
	小于等于	LE	（SF ⊕ OF）∨ZF=1	JLE	JNG
	大于	G	（SF ⊕ OF）∨ZF=0	JG	JNLE
	大于等于	GE	SF ⊕ OF=0	JGE	HNL

条件转移指令是使用标志位的最基本和最常用的方法。需要注意，它只是使用标志位，在应用中，它必须与产生标志位的指令相配合才有算法意义。产生标志位的指令主要是算术运算和逻辑运算指令，例如比较 CMP、测试 TEST 等。

【例 3.14】 将(AX)求绝对值。

```
        CMP AX, 0       ; 比较(AX)和 0，产生标志位
        JGE Done        ; 大于等于 0 则转移，使用标志位
        NEG AX          ; 小于 0 则求补
    Done:
```

例中“CMP AX, 0”常用“OR AX, AX”替代，甚至“AND AX, AX”、“SUB AX, 0”、“OR AX, 0”，“AND AX, 0FFFFH”等指令也可使用，结果是一样的。

【例 3.15】 如果字变量 VAR 的最低 4 个 Bit 全为 0 则将(AL)设为 1，否则设为 0。

```
        TEST VAR, 0FH   ; 将 VAR“与”上 000FH，产生标志位
        MOV AL, 1       ; (AL)=1
        JZ Done         ; 结果为 0 则转移，使用标志位
        MOV AL, 0       ; 不转移，(AL)=0
    Done:
```

例中，“MOV AL, 1”对标志位没有影响，所以条件转移指令 JZ 使用的是“TEST VAR, 0FH”所产生的标志位。

4．循环指令

循环指令的格式、操作如表 3.28 所示。

表 3.28　LOOP 指令

指　令	格　式	操　作
LOOP	LOOP OPR(标号)	① (CX)←(CX) – 1 ② 若条件(CX)≠0 成立，则转移，(IP)←(IP) + 8 位相对量

循环指令 LOOP 执行两步工作，先将(CX)减 1，再判断(CX)是否为 0，如果不等于 0 则转移。LOOP 指令的转移方式也是 8 位的相对寻址，这和条件转移指令是一样的。实际上，LOOP 指令就可以看作是将“(CX)←(CX) – 1”和判断“(CX)≠0”的条件转移结合在一起的特殊的条件转移指令。

循环指令 LOOP 是对程序循环结构的支持。

【例 3.16】 在数据段中定义的 100 字的数组 ARY 逆序传送到另一数组 DEST。

```
        LEA   SI, ARY       ; SI 指向 ARY
        LEA   DI, DEST
        ADD   DI, 100–2     ; DI 指向 DEST 的尾
        MOV   CX, 100       ; (CX)=数组长度，初始化循环次数
    L1: MOV   AX, [SI]
        MOV   [DI], AX      ; (DI) ← (SI)
        ADD   SI,    2      ; SI 指向下一个字
        DEC   DI,    2      ; DI 指向下一个位置
        LOOP L1             ; 循环到 L1
```

LOOP 指令使用 CX 作为循环次数的控制，这是隐含的。需要注意的是，在循环体内不恰当地使用 CX 会对循环结构造成破坏。

5．子程序调用与返回

调用指令 CALL 和返回指令 RET 是对子程序结构的支持。

如图 3.34 所示，子程序结构的特点是：主程序调用子程序时转移到子程序执行，子程序结束后返回到主程序调用位置继续执行下面的指令。一方面，CALL 指令和 RET 指令本质上都是转移的。从转移的功能角度看，CALL 指令也分为段内调用和段间调用，即调用位置和子程序是在同一个段内或不同段间，与 CALL 相对应，RET 指令也分为段内返回和段间返回。从转移的形式上看，CALL 指令又分为直接调用和间接调用。另一方面，为了保证子程序结束后能够返回到调用位置，调用时需要将位置保存起来，称为保存断点，而返回到调用位置就称为恢复断点。保存断点和恢复断点是通过堆栈进行的。

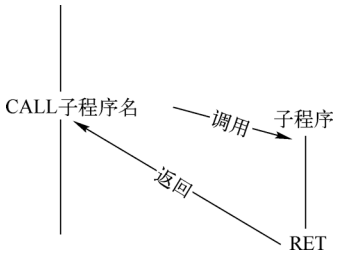

图 3.34　子程序结构的调用与返回示意图

CALL 指令和 RET 指令的格式、操作如表 3.29 所示。

表 3.29　CALL 指令和 RET 指令

指　令			CALL	RET	
格　式			CALL OPR	RET	RET n
操作	段内	段内直接	OPR 是子程序名 ① (IP)入栈 ② (IP)←(IP) + 8 位或 16 位相对量	(IP)←出栈	① (IP)←出栈 ② (sp)←(sp)+n
		段内间接	OPR 是寄存器或存储操作数 ① (IP)入栈 ② (IP)←(OPR)		
	段间	段间直接	OPR 是子程序名 ① (CS)入栈 (IP)入栈 ② (CS)←OPR 指定的 16 位段址 (IP)←OPR 指定的 16 位偏移地址	(CS)←出栈 (IP)←出栈	① (CS)←出栈 (IP)←出栈 ② (sp)←(sp)+n
		段间间接	OPR 是存储操作数 ① (CS)入栈 (IP)入栈 ② (CS)←(OPR)中的 16 位段址 (IP)←(OPR)中的 16 位偏移地址		

CALL 指令在转移形式上与 JMP 指令是一样的，例如：

（1）段内直接调用

```
CALL SHORT SUBPROC        ; SUBPROC，段内子程序；SHORT，指明 8 位相对转移；
CALL NEAR PTR SUBPROC     ; SUBPROC，段内子程序；NEAR PTR，指明 16 位相对转移；
```

（2）段内间接转移

```
CALL WORDR PTR [BX]       ; WORD PTR，指明字操作数
```

（3）段间直接转移

```
CALL FAR PTR ROUTE        ; ROUTE，段间子程序；FAR PTR，指明段间转移；
```

（4）段间间接转移

```
CALL DWORD PTR [BX]          ; DWORD PTR，指明双字操作数
```

在实际编程时，最常用的是直接使用子程序名的直接调用，一般也不需要指定操作符的修饰，即直接用“CALL 子程序名”的形式即可，汇编程序会根据子程序的性质自动选择合适的调用方式。

【例 3.17】 画出下面程序段的堆栈变化示意图。

```
100H:  CALL SHORT SUBPROC
102H:
       …
200H:  PUSH AX      ;SUBPROC
       …
       POP AX
       RET
```

堆栈变化如图 3.35 所示。

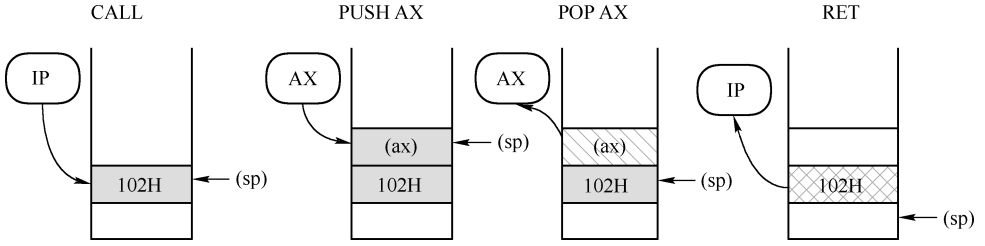

图 3.35　堆栈变化示意图

CALL 指令和 RET 指令依赖于堆栈的支持，如例 3.17 所示。如果在子程序中不能保证堆栈的平衡，则 RET 指令从堆栈中恢复到(IP)或恢复到(CS)和(IP)的地址就不是调用保存的断点，这将对程序造成致命的危害。

6．中断指令

CPU 在正常执行程序的过程中，由于某个外部或内部事件的作用，强迫 CPU 停止当前正在执行的程序，转去执行专门为该事件服务的程序（称为中断服务程序），待服务结束后返回被中断的程序继续执行，这一过程称为中断，如图 3.36 所示。

8086 将事件按种类进行编号，称为中断号。并按照中断号对中断服务程序进行统一管理，具体的做法是：将中断服务程序的地址（称为中断向量）按照中断号排列组织成中断向量表。如图 3.37 所示，8086 使用存储器最低端的 1KB（0～3FFH）存储中断向量表，每个中断向量占 4 字节（16 位偏移地址和 16 位段址），共 256 个中断向量，中断号 0～255。

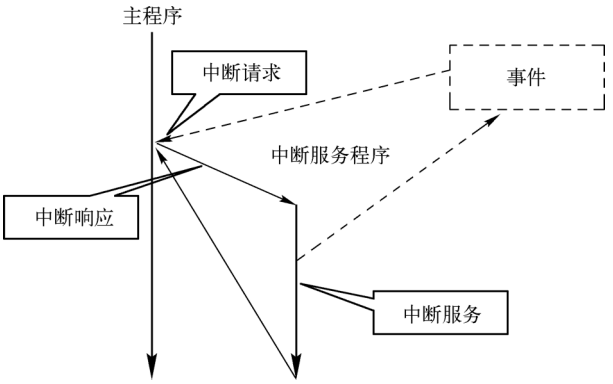

图 3.36　中断

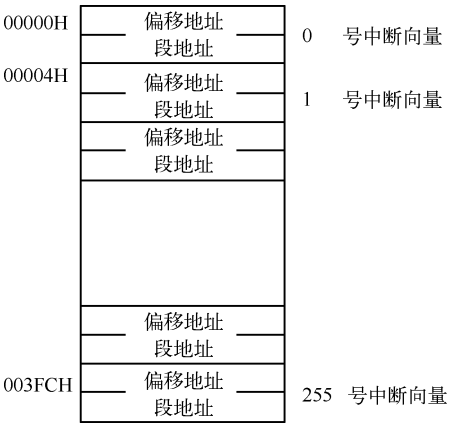

图 3.37　中断向量表

8086 可以用软中断指令 INT 产生一个指定中断号的内部中断，它的中断响应过程与其他内部中断和外部中断是一样的，其指令的格式、操作如表 3.30 所示。

在中断响应过程中，CPU 使用堆栈保存标志寄存器内容和保存断点（(CS)和(IP)）并转移到中断号指定的中断服务程序，转移的方法是用中断号查询中断向量表（即中断号×4）得到中断向量并用中断向量中的偏移地址修改(IP)、段址修改(CS)。

表 3.30　INT 指令

指　令	格　式	操　作	备　注
INT	INT TYPE(中断号)	① (FLAGS)入栈	标志寄存器入栈
		② IF, TF ← 0	清除 IF 和 TF 标志位
		③ (CS)入栈 (IP)入栈	保存断点
		④ (IP)←(TYPE×4) (CS)←(TYPE×4+2)	转中断服务程序

【例 3.18】 取出中断号 5AH 的中断向量。

```
MOV AX, 0                ;
MOV ES, AX               ; ES←0，中断向量表存放在内存最低端，段址为 0
LDS SI, ES:[5AH*4]       ; 中断向量放在中断向量表“中断号×4”位置，占 4 字节，
                         ; 低字是偏移地址，高字是段址。所以使用 LDS 取长指针。
```

中断服务程序结束时需要使用 IRET 指令返回被中断的程序，其格式、操作如表 3.31 所示。

表 3.31　IRET 指令

指　令	格　式	操　作	备　注
IRET	IRET	① (IP)←出栈	恢复断点
		② (CS)←出栈	
		③ (FLAGS)←出栈	恢复标志寄存器

操作系统（如 DOS）通常使用软中断指令 INT 提供系统功能调用，可以让程序员在不

了解操作系统程序分布的情况下透明地使用操作系统的功能服务。

【例 3.19】 列举常用的 DOS 系统功能调用。

```
MOV AH, 1          ; 1 号系统功能，接受键盘输入并回显
INT 21H            ; 系统功能调用，AL 将返回键盘按键的 ASCII 码
MOV KEY, AL        ; 保存按键到变量 KEY 中
MOV AH, 2          ; 2 号系统功能，显示字符
MOV DL,'H'         ; DL 中存放要显示字符的 ASCII 码
INT 21H            ; 系统功能调用，在屏幕上显示字母“H”
MOV AH, 9          ; 9 号系统功能，显示字符串
LEA DX, STR        ; DS:DX 指向要显示的字符串，字符串以'$'字符结束
INT 21H            ; 系统功能调用，在屏幕上显示字符串
MOV AH, 4CH        ; 4CH 号系统功能，结束程序
INT 21H            ; 系统功能调用，结束程序，返回 DOS
```

3.3.6 处理器控制指令

1．标志位控制指令

标志位控制指令用于对 CF、DF 和 IF 标志位的清除和设置，其指令规则说明如表 3.32 所示。

表 3.32 标志位控制指令

标 志 位	指 令	格 式	操 作
CF	CLC	CLC	CF←0
	CMC	CMC	CF← $\overline{CF}$
	STC	STC	CF←1
DF	CLD	CLD	DF←0
	STD	STD	DF←1
IF	CLI	CLI	IF←0
	STI	STI	IF←1

2．其他处理器控制指令

（1）NOP

格式：NOP

功能：空操作指令。不执行任何操作，但要花费 CPU 一个总线周期的时间。

（2）HLT

格式：HLT

功能：停机指令。CPU 不执行任何操作，一直处于等待中断的暂停状态，响应中断后继续执行下一条指令。

3.4 80x86 寻址方式及指令的扩充

3.4.1 寻址方式的扩充

与 8086 相比，80x86 的 32 位寻址方式主要扩充有：

① 在 32 位寻址方式下，基址寻址可使用任何一个 32 位通用寄存器；变址寻址可使用除堆栈指针寄存器 ESP 外的任何一个 32 位通用寄存器。

例如：MOV EAX, [EDX]　；寄存器间接寻址不再局限于 EBX、EBP、ESI、EDI。

② 使用变址寄存器进行寻址时，可以选取一个比例因子，取值 1、2、4、8。因此，变址寻址方式下的有效地址计算式为：EA = 变址寄存器的值×比例因子＋位移量。基址加变址寻址方式依此类推。

例如：MOV　ECX, [EAX][EDX*8]　; EA=(EAX)+(EDX)×8

3.4.2 指令系统的扩充

80386、80486 等 32 位微处理器的指令系统除了保留前代 CPU 原有的功能并进行增加外，在实模式功能、保护模式管理、模式间的转换及管理等方面也逐步进行了扩充，扩充的基本指令如表 3.33 所示。

表 3.33　80386 扩充的指令

类　名	指　令	说　明
传送类	MOVSX	带符号扩展传送

续表

类　名	指　令	说　明
传送类	MOVZX	带零扩展传送
	PUSHA	将全部 16 位通用寄存器进栈
	PUSHAD	将全部 32 位通用寄存器进栈
	PUSHFD	32 位标志寄存器进栈
	POPA	将全部 16 位通用寄存器出栈
	POPAD	将全部 32 位通用寄存器出栈
	POPFD	32 位标志寄存器出栈
	BSWAP	使指定的 32 位寄存器的字节次序变反
算术类	XADD	加法，并将原目操作数的交换到源操作数
逻辑类	SHLD	双精度左移
	SHRD	双精度右移
	BT	位测试
	BTS	位测试并置 1
	BTR	位测试并置 0
	BTC	位测试并求反
	BSF	位扫描，自低位至高位
	BSR	位扫描，自高位至低位

处理机控制类	BOUND	数组边界检查
	ENTER	建立堆栈帧指令
	LEAVE	释放堆栈帧指令

习题 3

1．指令由哪两部分构成？分别起什么作用？

2．存储器寻址方式提供的是什么地址？它最终如何映射到实际的物理地址？

3．简述立即寻址和直接寻址、寄存器寻址和寄存器间接寻址的区别。

4．指出下列各种操作数的寻址方式。

（1）MOV AX, 420H

（2）MOV [BX], SI

（3）MOV AX, [90]

（4）MOV [DI+90], AH

（5）MOV AL, [BP+SI+20]

5．指出下列寻址方式中源操作数所使用的段寄存器。

（1）MOV AX, [SI+20h]

（2）MOV AL, [1000H]

（3）MOV AX, ES:[BX][SI]

（4）MOV AX, [BP+1234H]

6．已知寄存器(BX)=2010H、(DI)=0FFF0H 和(BP)=420H，试分别计算出下列寻址方式中源操作数的有效地址。

（1）MOV AX, [2345H]

（2）MOV AX, [BX]

（3）MOV AX, ES:[DI+200]

（4）MOV AX, DS:[BP+DI]

（5）MOV AX, [BX+DI+114H]

7．使用类似 MOV 指令的双操作数指令需要注意什么？

8．8086 系统的堆栈是如何组织的？如何判断堆栈满或空？

9．已知(SS)=0FF00H、(SP)=00B0H，先执行两条将 8086H 和 0F20H 入栈的 PUSH 指令，再执行一条 POP 指令。请画出堆栈内容变化过程示意图（请标出存储单元的物理地址）。

10．用一条指令实现将 BX 与 SI 之和传送给 CX 的功能。

11．写出下列指令序列中每条指令的执行结果，并指出标志位 CF、ZF、OF、SF 的变化情况。

```
MOV     BX,     40ABH
ADD     BL,     09CH
MOV     AL,     0E5H
CBW
ADD     BH,     AL
```

```
SBB     BX,     AX
ADC     AX,     20H
SUB     BH,     –9
```

12．简述乘法指令和除法指令寻址方式的特点。

13．完成一个计算 DL（无符号数）三次方的指令序列。

14．按下列要求编写指令序列。

（1）清除 DH 中的最低 3 位而不改变其他位，结果存入 BH 中；

（2）把 DI 中的最高 5 位置 1 而不改变其他位；

（3）把 AX 中的 0～3 位置 1，7～9 位取反，13～15 位置 0；

（4）检查 BX 中的第 2、5 和 9 位中是否有一位为 1；

（5）检查 CX 中的第 1、6 和 11 位中是否同时为 1；

（6）检查 AX 中的第 0、2、9 和 13 位中是否有一位为 0；

（7）检查 DX 中的第 1、4、11 和 14 位中是否同时为 0；

15．NOT 指令和 NEG 指令有和不同？编写指令序列将(DX:AX)中的双字算术求反。

16．使用移位指令将 40 和–49 分别乘 2 和除 2，请注意选择合适的移位指令。

17．分析下面指令序列完成的功能。

```
MOV     CL,     4
SHL     DX,     CL
MOV     BL,     AH
SHL     AX,     CL
SHR     BL,     CL
OR      DL,     BL
```

18．方向标志 DF 的作用是什么？用于设置或消除该标志位的指令是什么？

19．已知数据段中保存有 100 个字的数组，起始地址为 0B00H。编写指令序列实现将–1 插入到第一个字。

20．什么是段间转移？什么是段内转移？它们的实现机制有何不同？

21．条件转移指令中，所谓的“条件”是什么，如何得到？如何使用？请举例说明。

22．假设 AX 和 BX 中的内容是有符号数，CX 和 DX 中的内容是无符号数，试实现：

（1）如果(DX)>(CX)，则转到 EXCEED；

（2）如果(BX)>(AX)，则转到 EXCEED；

（3）如果(CX)=0，则转到 ZERO；

（4）如果(BX)–(AX)将产生溢出，则转到 OVERFLOW；

（5）如果(BX)≤(AX)，则转到 NOTBIG；

（6）如果(DX)≤(CX)，则转到 NOTBIG。

23．已知数据段中保存有 100 个字的数组，起始地址为 0B00H。编写指令序列将数组中每个字加 1。

24．简述段内调用和段间调用的主要区别。

25．已知指令 CALL SUB1 实现段内调用子程序 SUB1，请用 JMP 指令序列实现此调用功能。

26．子程序 SUB1 的最后一条指令是 RET 指令实现段内返回，它能否用 JMP 指令

模拟？

27．如何得到中断 60H 的中断服务程序的首地址？

28．编写指令序列完成将中断 1CH 指向新的中断服务程序的功能，新的中断服务程序首地址为 0FA00:100H。