第3章 80x86指令系统

3.1 指令格式

计算机是通过执行指令序列来解决问题的，因而每种计算机都有一组指令集供用户使用，这组指令集称为计算机的指令系统。指令由操作码和操作数构成，操作码说明指令的功能，用指令助记符表示；操作数由各种不同的寻址方式提供。汇编语言的指令格式如下：

```
指令助记符[操作数1[,操作数2[,操作数3]]] [;注释]
```

指令助记符体现该指令的功能，对应一条二进制编码的机器指令。指令的操作数个数由该指令来确定，可以没有操作数，也可以有一个、二个或三个操作数。绝大多数指令的操作数要显式的写出来，但也有指令的操作数是隐含的，不需要在指令中写出。

当指令含有操作数，并要求在指令中显式地写出来时，在书写时必须遵守以下原则：

- 指令助记符和操作数之间要有分隔符，分隔符可以是若干个空格或 TAB 键；
- 如果指令含有多个操作数，那么操作数之间要用英文的逗号“,”分开；
- 指令后面还可以书写注释内容，注释语句以分号“;”开始。

3.2 寻址方式

操作数是指令或程序的主要处理对象。如果某条指令或某个程序不处理任何操作数，那么该指令或程序不可能有数据处理功能。在 CPU 的指令系统中，除 NOP(空操作指令)、HLT(停机指令)等少数指令之外，大量的指令在执行过程中都会涉及操作数。所以，在指令中如何表达操作数或操作数所在位置就是正确运用汇编指令的一个重要因素。

在指令中，计算操作数或操作数存放位置的方法称为寻址方式。操作数的各种寻址方式是用汇编语言进行程序设计的基础，也是本课程学习的重点之一。

80x86 系列中，8086 和 80286 的字长是 16 位，一般只处理 8 位和 16 位数，只在乘除指令中才会有 32 位数；80386 及其后续机型的字长为 32 位，因此可以处理 8 位、16 位外，还可以处理 32 位操作数，在乘除指令中还可以产生 64 位数。在本书后面的讲述中，如果处理的是 32 位数，则适用于 80386 及其后续机型。

80x86 系统有 7 种基本的寻址方式：立即寻址方式、寄存器寻址方式、直接寻址方式、寄存器间接寻址方式、寄存器相对寻址方式、基址加变址寻址方式、相对基址加变址寻址方式。其中，后 5 种寻址方式是确定内存单元有效地址的 5 种不同的计算方法，可方便地实现对数组元素的访问。

另外，在 32 位微机系统中，为了扩大对存储单元的寻址能力，增加了一种新的寻址方式——32 位地址的寻址方式。

为了表达方便，这里用符号(X)表示 X 的值，如(AX)表示寄存器 AX 的值。

3.2.1 立即寻址方式

操作数作为指令的一部分直接写在指令中，这种操作数称为立即数，也即高级语言里面的常量。这种寻址方式称为立即数寻址方式。

立即数可以是 8 位、16 位或 32 位，该数值紧跟在操作码之后。如果立即数为 16 位或 32 位，那么，它将按“高高低低”的原则进行存储。

【例 3-1】 立即数寻址举例。

```
MOV AH,80H       ; 指令执行后(AH) = 80H
ADD AX,1234H     ; 假定指令执行前(AX) = 0014H,则执行后(AX) = 0014H + 1234H = 1248H
MOV ECX,123456H  ; 假定指令执行前(ECX) = A0014H,则执行后(ECX) = A0014H + 123456H = 1C346AH
MOV B1,12H       ; 把立即数 12H 送到名称为 B1 的字节存储单元中
MOV W1,3456H     ; 把立即数 3456H 送到名称为 W1 的字存储单元中
```

各指令执行情况的对比如表 3-1 所示。

表 3-1 立即数寻址方式

指令执行前各寄存器内容	执行的指令	指令执行后相关寄存器的内容(16 进制表示)
AX:1234H	MOV AH,80H	AX:8034H
AX:0014H	ADD AX,1234H	(AX)＝0014H＋1234H＝1248H
ECX:A0014H	MOV ECX,123456H	(ECXX)＝A0014H＋123456H＝1C346AH
	MOV B1,12H	把立即数 12H 送到名称为 B1 的字节存储单元中
	MOV W1,3456H	把立即数 3456H 送到名称为 W1 的字存储单元中

以上指令中的第二操作数都是立即数，在汇编语言中，规定立即数不能作为指令中的第一操作数。该规定与高级语言中“赋值语句的左边不能是常量”的规定是一致的。

立即数寻址方式通常用于对通用寄存器或内存单元赋初值。图 3-1 所示是指令“MOV AX,4576H”存储形式和执行示意图。

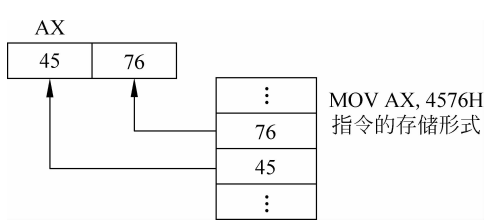

图 3-1 立即数寻址的存储和执行

3.2.2 寄存器寻址方式

寄存器寻址方式是指令所要的操作数已存储在某寄存器中，或把目标操作数存入寄存器。指令中可以引用的寄存器及其符号名称如下：

8 位寄存器有 AH、AL、BH、BL、CH、CL、DH 和 DL 等。

16 位寄存器有 AX、BX、CX、DX、SI、DI、SP、BP 和段寄存器等。

32 位寄存器有 EAX、EBX、ECX、EDX、ESI、EDI、ESP 和 EBP 等。

寄存器寻址方式是一种简单快捷的寻址方式，源和目的操作数都可以是寄存器。

1. 源操作数是寄存器寻址方式

例如：

```
ADD VARD,EAX
ADD VARW,AX
MOV VARB,BH
```

其中，VARD、VARW 和 VARB 是双字、字和字节类型的内存变量。在第 4 章将会学到如何定义它们。

2. 目的操作数是寄存器寻址方式

例如：

```
ADD BH,78H
ADD AX,1234H
MOV EBX,12345678H
```

3. 源和目的操作数都是寄存器寻址方式

例如：

```
MOV EAX,EBX
MOV AX,BX
MOV DH,BL
```

由于指令所需的操作数已存储在寄存器中，或把操作的结果存入寄存器，这样在指令执行过程中会减少读写存储器单元的次数，所以使用寄存器寻址方式的指令具有较快的执行速度。通常情况下，提倡在编写汇编语言程序时，应尽可能地使用寄存器寻址方式，但也不要把它绝对化。

3.2.3 直接寻址方式

指令所要的操作数存放在内存中，在指令中直接给出该操作数的有效地址，这种寻址方式为直接寻址方式。

在通常情况下，操作数存放在数据段中，所以其物理地址将由数据段寄存器 DS 和指令中给出的有效地址直接形成，但如果使用段超越前缀，那么操作数可存放在其他段。

【例 3-2】 假设有指令“MOV BX,[1234H]”,在执行时,(DS)=2000H,内存单元21234H的值为5D2AH。问该指令执行后,BX的值是什么?

根据直接寻址方式的寻址规则,把该指令的具体执行过程用图3-2来表示。

从图3-2中,可看出执行该指令要分三步。

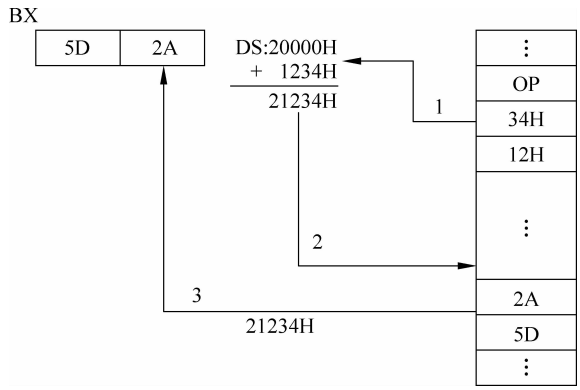

图3-2 直接寻址的存储和执行

(1) 由于1234H是一个直接地址,它紧跟在指令的操作码之后,随取指令而被读出。

(2) 访问数据段的段寄存器是DS,用DS的值和偏移量1234H相加,得存储单元的物理地址为21234H。

(3) 取单元21234H的值5D2AH,并按“高高低低”的原则存入寄存器BX中。

所以,在执行该指令后,BX的值就为5D2AH。

由于数据段的段寄存器默认为DS,如果要指定访问其他段内的数据,可在指令中用段跨越前缀显式地书写出来。下面指令的目的操作数就是带有段跨越前缀的直接寻址方式。

```
MOV  ES:[1000H],AX  ; 该指令把AX寄存器的内容送入地址为(ES)*16+1000H的存储单元
```

直接寻址方式常用于处理内存单元的数据,其操作数是内存变量的值,该寻址方式可在64KB的段内进行寻址。

注意:立即寻址方式和直接寻址方式的书写格式不同,直接寻址的地址要写在方括号[]内。在程序中,直接地址通常用内存变量名来表示。例如,“MOV BX,VARW”中的VARW是内存字变量的名称,该变量在数据段中定义。

【例 3-3】 试比较下列指令中源操作数的寻址方式(VARW是内存字变量)。

```
MOV  AX,1234H   ; 前者是立即寻址
MOV  AX,[1234H] ; 后者是直接寻址
MOV  AX,VARW
MOV  AX,[VARW]  ; 两者是等效的,均为直接寻址
```

3.2.4 寄存器间接寻址方式

操作数在存储器中,操作数的有效地址用SI、DI、BX和BP四个寄存器之一来指定,称这种寻址方式为寄存器间接寻址方式。该寻址方式物理地址的计算方法如下:

$$EA = (DI)/(SI)/(BP)/(BX)$$

$$PA = (DS) \times 16 + EA$$

寄存器间接寻址方式读取存储单元的原理如图 3-3 所示。在不使用段超越前缀的情况下，规定：

- 若有效地址用 SI、DI 和 BX 等之一来指定，则其默认的段寄存器为 DS；
- 若有效地址用 BP 来指定，则其默认的段寄存器为 SS(即堆栈段)。

【例 3-4】 假设有指令“MOV BX,[DI]”，在执行时，(DS)＝1000H，(DI)＝2345H，存储单元 12345H 的内容是 4354H。问执行指令后 BX 的值是什么？

根据寄存器间接寻址方式的规则，在执行本例指令时，寄存器 DI 的值不是操作数，而是操作数的地址。

该操作数的物理地址应由 DS 和 DI 的值形成，即 PA＝(DS)×16＋(DI)＝1000H×16＋2345H＝12345H。

所以，该指令的执行效果是把从物理地址为 12345H 开始的一个字存储单元中的值传送给 BX 寄存器。

其执行过程如图 3-4 所示。

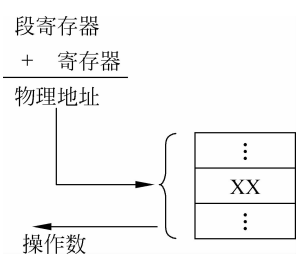

图 3-3　寄存器间接寻址

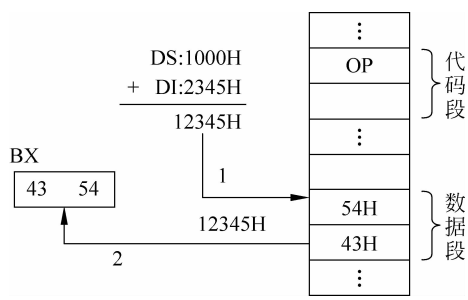

图 3-4　寄存器间接寻址

3.2.5　寄存器相对寻址方式

操作数在存储器中，其有效地址是一个基址寄存器(BX、BP)或变址寄存器(SI、DI)的内容和指令中的 8 位/16 位偏移量之和。其有效地址的计算公式如下式所示。

$$EA＝BX/(BP)＋位移量，PA＝(DS)×16＋EA$$

在不使用段跨越前缀的情况下，有下列规定：

若有效地址用 SI、DI 和 BX 等之一来指定，则其默认的段寄存器为 DS；若有效地址用 BP 来指定，则其默认的段寄存器为 SS。

指令中给出的 8 位/16 位偏移量用补码表示。在计算有效地址时，如果偏移量是 8 位，则进行符号扩展成 16 位。当所得的有效地址超过 0FFFFH，则取其 64K 的模。

【例 3-5】 假设指令“MOV BX,[SI＋100H]”，在执行它时，(DS)＝1000H，(SI)＝2345H，内存单元 12445H 的内容为 39A8H，问该指令执行后 BX 的值是什么？

根据寄存器相对寻址方式的规则，在执行本例指令时，源操作数的有效地址 EA 为

$$EA＝(SI)＋100H＝2345H＋100H＝2445H$$

该操作数的物理地址应由 DS 和 EA 的值形成，即

$$PA=(DS)\times16+EA=1000H\times16+2445H=12445H$$

所以，该指令的执行效果是，把从物理地址为 12445H 开始的一个字单元中的值 39A8H 传送给 BX。其执行过程如图 3-5 所示。执行后 BX 寄存器的值为 39A8H。

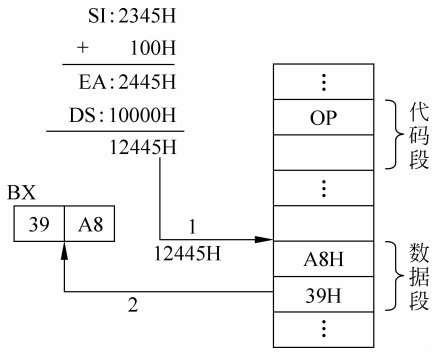

图 3-5 寄存器相对寻址

3.2.6 基址加变址寻址方式

操作数在存储器中，其有效地址是一个基址寄存器(BX、BP)和一个变址寄存器(SI、DI)的内容之和。其有效地址的计算公式为：EA=(BX)/(BP)+(SI)/(DI)；PA=(DS)×16+EA。

在不使用段超越前缀的情况下，规定：如果有效地址中含有 BP，则默认的段寄存器为 SS；否则，默认的段寄存器为 DS。

【例 3-6】 假设指令“MOV BX,[BX+SI]”，在执行时，(DS)=1000H，(BX)=2100H，(SI)=0011H，内存单元 12111H 的内容为 1234H。问该指令执行后，BX 的值是什么？

根据基址加变址寻址方式的规则，在执行本例指令时，源操作数的有效地址 EA 为

$$EA=(BX)+(SI)=2100H+0011H=2111H$$

该操作数的物理地址应由 DS 和 EA 的值形成，即

$$PA=(DS)\times16+EA=1000H\times16+2111H=12111H$$

该指令的执行效果是，把从物理地址为 12111H 开始的一个字单元中的值传送给 BX 寄存器，执行后(BX)=1234H。执行过程如图 3-6 所示。

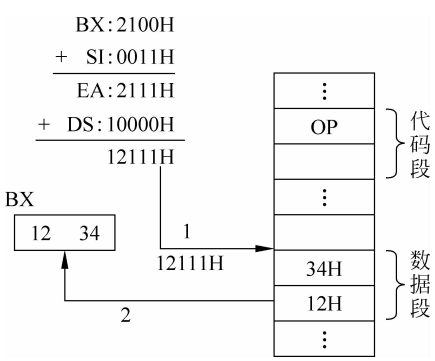

图 3-6 基址加变址寻址

3.2.7　相对基址加变址寻址方式

操作数在存储器中，其有效地址是一个基址寄存器(BX、BP)的值、一个变址寄存器(SI、DI)的值和指令中的8位/16位偏移量之和。其有效地址的计算公式为

EA＝(BX)/(BP)＋(SI)/(DI)＋位移量，　PA＝(DS)×16＋EA

在不使用段超越前缀的情况下，规定：如果有效地址中含有BP，则其默认的段寄存器为SS；否则，其默认的段寄存器为DS。

指令中给出的8位/16位偏移量用补码表示。在计算有效地址时，如果偏移量是8位，则进行符号扩展成16位。当所得的有效地址超过0FFFFH，则取其64K的模。

【例3-7】 假设指令“MOV AX,[BX＋SI＋200H]”，在执行时，(DS)＝3000H，(BX)＝2100H，(SI)＝0010H，内存单元32310H的内容为5678H。问该指令执行后AX的值是什么？

根据相对基址加变址寻址方式的规则，在执行本例指令时，源操作数的有效地址EA为

EA＝(BX)＋(SI)＋200H＝2100H＋0010H＋200H＝2310H

该操作数的物理地址应由DS和EA的值形成，即PA＝(DS)×16＋EA＝3000H×16＋2310H＝32310H。该指令的执行效果是，把从物理地址为32310H开始的一个字单元中的值传送给AX寄存器。执行后(AX)＝5678H。

从相对基址加变址这种寻址方式来看，由于它的可变因素较多，看起来就显得复杂，但正因为其可变因素多，它的灵活性也就很高，如图3-7所示。

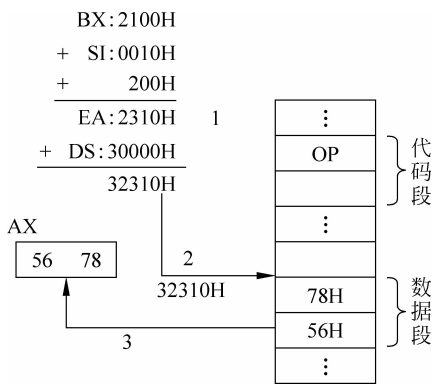

图3-7　相对基址加变址寻址

例如：

用D1[i]来访问一维数组D1的第i个元素，它的寻址有一个自由度，用D2[i][j]来访问二维数组D2的第i行、第j列的元素，其寻址有两个自由度。多一个可变的量，其寻址方式的灵活度也就相应提高了。

相对基址加变址寻址方式有多种等价的书写方式，下面的书写格式都是正确的，并且其寻址含义也是一致的。

```
MOV  AX,[BX + SI + 1000H]
MOV  AX,1000H[BX + SI]
MOV  AX,1000H[BX][SI]
MOV  AX,1000H[SI][BX]
```

但书写格式 BX [1000＋SI]和 SI[1000H＋BX]等是错误的，即所用寄存器不能在[“,”]之外，该限制对寄存器相对寻址方式的书写也同样起作用。因为[BX]表示把 BX 寄存器的内容作为地址，而 BX 表示直接使用 BX 中的内容。

相对基址加变址寻址方式是以上 7 种寻址方式中最复杂的一种寻址方式，可变形为其他类型的存储器寻址方式。表 3-2 所示是该寻址方式与其他寻址方式之间的变形关系。

表 3-2　相对基址加变址寻址方式与其他寻址方式之间的变形关系

源操作数	指令的变形	源操作数的寻址方式
只有偏移量	MOV AX,[100H]	直接寻址方式
只有一个寄存器	MOV AX,[BX]MOV AX,[SI]	寄存器间接寻址方式
有一个寄存器和偏移量	MOV AX,[BX＋100H]　MOV AX,[SI＋100H]	寄存器相对寻址方式
有两个寄存器	MOV AX,[BX＋SI]	基址加变址寻址方式
有两个寄存器和偏移量	MOV AX,[BX＋SI＋100H]	相对基址加变址寻址方式

3.3　数据传送指令

80x86 指令系统，指令按功能可分为数据传送指令、算术运算指令、控制转移指令、串操作指令、逻辑运算指令、输入输出指令、处理器控制指令、保护方式指令 8 个部分。

数据传送指令包括通用数据传送指令、地址传送指令、标志寄存器传送指令、符号扩展指令、扩展传送指令等。

3.3.1　通用数据传送指令

1. 传送指令

格式：

```
MOV DEST,SRC
```

功能：把一个字节、字或双字从源操作数 SRC 传送至目的操作数 DEST。

执行的操作：(DEST)←(SRC)。

传送指令允许的数据流方向如图 3-8 所示。

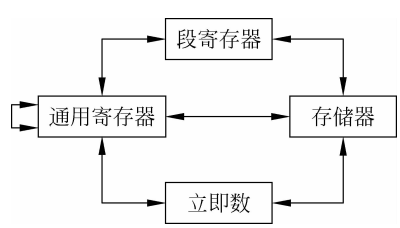

图 3-8　传送指令数据流

由图 3-8 可知，数据允许流动方向为：通用寄存器之间、通用寄存器和存储器之间、通用寄存器和段寄存器之间、段寄存器和存储器之间，另外还允许立即数传送至通用寄存器或存储器。但在上述传送过程中，段寄存器 CS 的值不能用传送指令改变。

【例 3-8】 CPU 内部寄存器之间的数据传送。

```
MOV AL,DH          ; (AL)←(DH)(8 位)
MOV DS,AX          ; (DS)←(AX)(16 位)
MOV EAX,ESI        ; (EAX)←(ESI)(32 位)
```

【例 3-9】 CPU 内部寄存器和存储器之间的数据传送。

```
MOV [BX],AX          ; 间接寻址,把 AX 寄存器的内容传送到 BX 寄存器内容为地址的单元内 (16 位)
MOV EAX,[EBX + ESI]  ; 基址变址寻址(32 位)
MOV AL,BLOCK         ; BLOCK 为变量名,直接寻址(8 位)
```

【例 3-10】 立即数送通用寄存器、存储器。

```
MOV EAX,12345678H    ; EAX←12345678H (32 位)
MOV [BX],12H         ; 间接寻址(8 位)
MOV AX,1234H         ; AX←1234H(16 位)
```

【例 3-11】 段地址必须通过寄存器(如 AX,BX 等)送到 DS 寄存器。

```
MOV AX,DATA_SEG
MOV DS,AX
```

【例 3-12】 把变量 TABLE 的偏移地址送 BX 寄存器。

```
MOV BX,OFFSET TABLE ; OFFSET 为属性操作符,表示把其后的符号地址的值(不是内容)作为操作数
```

注意:

- 源和目的操作数不允许同时为存储器操作数;
- 源和目的操作数数据类型必须一致;
- 源和目的操作数不允许同时为段寄存器;
- 目的操作数不允许为 CS 和立即数;
- 当源操作数为立即数时,目的操作数不允许为段寄存器;
- 传送操作不影响标志位。

2. 扩展传送指令

格式:

```
MOVSX  DEST,SRC
MOVZX  DEST,SRC
```

功能:将源操作数由 8 位扩展到 16 位并传送到目的操作数,或由 16 位扩展到 32 位并传送到目的操作数。其中 MOVSX 是按有符号数扩展,MOVZX 是按无符号数扩展。无符号数或有符号数的正数高位扩展为 0,有符号数的负数高位扩展为全 1。这两个指令不影响标志位。

【例 3-13】 带符号数扩展。

```
MOV BL,80H           - 128
MOVSX AX,BL          ; 将 80H 扩展为 FF80H 后送 AX 寄存器中。
```

【例 3-14】 无符号数扩展。

```
MOV BL,80H           ; 128
MOVZX AX,BL          ; 将 80H 扩展为 0080H 后送 AX 寄存器中。
```

注意:

- 目的操作数应为 16 位或 32 位通用寄存器;

- 源操作数长度须小于目的操作数长度，为 8 位或 16 位通用寄存器或存储器操作数；
- 扩展传送操作不影响标志位。

3. 交换指令

1）格式：

```
XCHG OPR1,OPR2
```

功能：交换操作数 OPR1 和 OPR2 的值，操作数数据类型为字节、字或双字。允许通用寄存器之间、通用寄存器和存储器之间交换数据，但不允许使用段寄存器。

执行的操作：

```
(OPR1)↔(OPR2)
```

【例 3-15】 交换指令举例。

```
XCHG AX,BX          ; 通用寄存器之间交换数据(16 位)
XCHG ESI,EDI        ; 通用寄存器之间交换数据(32 位)
XCHG BX,[BP + SI]   ; 通用寄存器和存储单元之间交换数据(16 位)，具体操作如表 3-3 所示。
```

表 3-3 寄存器和存储器之间的数据交换

指令执行前	执行的指令	指令执行后
(BX)＝6F30H，(BP)＝0200H，(SI)＝0046H， (SS)＝2F00H，(2F246H)＝4154H BP 寄存器默认的段寄存器是 SS	XCHG BX，[BP＋SI]	(BX)＝4154H， (2F246H)＝6F30H

注意：

- 操作数 OPR1 和 OPR2 不允许同为存储器操作数；
- 操作数数据类型必须一致；
- 交换指令不影响标志位。

如要实现存储器操作数交换，可用如下指令实现：

```
MOV AL,BLOCK1
XCHG AL,BLOCK2
MOV BLOCK1,AL
```

2）格式：

```
BSWAP REG
```

功能：将 32 位通用寄存器中，第 1 个字节和第 4 个字节交换，第 2 个字节和第 3 个字节交换。此指令适用于 486 及其之后的机型。

执行的操作如图 3-9 所示。

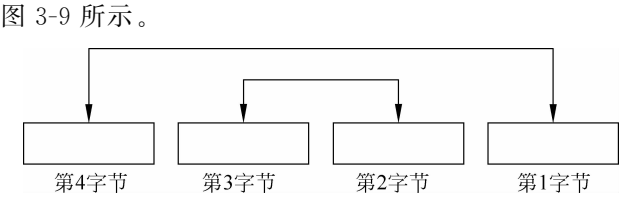

图 3-9 字节交换指令的执行过程

【例 3-16】 BSWAP 指令的使用。

```
MOV EAX,44332211H
BSWAP EAX            ; 指令执行后(EAX)  = 11223344H
```

注意：

- 操作数为 32 位通用寄存器；
- 交换指令不影响标志位。

3.3.2 堆栈操作指令

1. 入栈指令

(1) 源操作数入栈格式：

```
PUSH SRC
```

功能：将源操作数压入堆栈，源操作数允许为 16 位或 32 位通用寄存器、存储器和立即数以及 16 位段寄存器。当操作数数据类型为字类型，压栈操作使 SP 值减 2；当数据类型为双字类型，压栈操作使 SP 值减 4。

执行的操作：

```
16 位指令:(SP)←(SP) - 2
          ((SP) + 1,(SP))←(SRC)
32 位指令:(ESP)←(ESP) - 4
          ((ESP) + 3,(ESP) + 2,(ESP) + 1,(ESP))←(SRC)
```

(2) 寄存器入栈格式：

```
PUSHA
PUSHAD
```

功能：PUSHA 将 16 位通用寄存器压入堆栈，压栈顺序为 AX,CX,DX,BX,SP,BP,SI,DI。指令执行后(SP)←(SP)－16 。

PUSHAD 将 32 位通用寄存器压入堆栈，压栈顺序为 EAX,ECX,EDX,EBX,ESP,EBP,ESI,EDI。指令执行后(ESP)←(ESP)－32 。

【例 3-17】 PUSH AX 指令的执行情况，如图 3-10 所示。

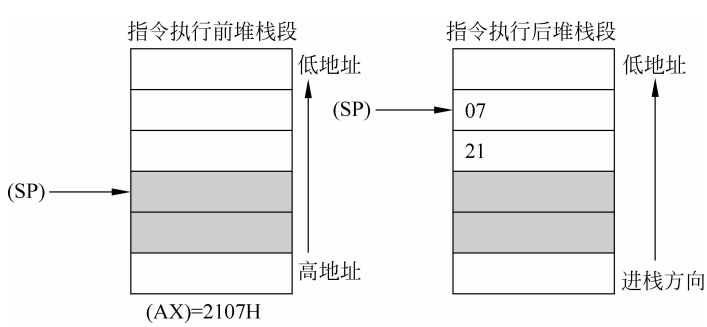

图 3-10　PUSH AX 指令的执行情况

【例 3-18】 PUSHAD 指令的执行情况，如图 3-11 所示。

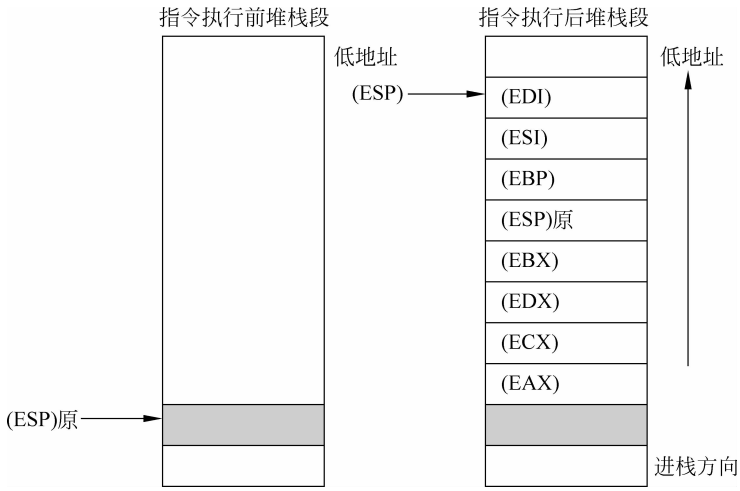

图 3-11 PUSHAD 指令的执行情况

2. 出栈指令

(1) 弹出操作数格式：

```
POP DEST
```

功能：从栈顶弹出操作数送入目的操作数。目的操作数允许为 16 或 32 位通用寄存器、存储器和 16 位段寄存器。当操作数数据类型为字类型，出栈操作使 SP 加 2；当操作数数据类型为双字类型，出栈操作使 SP 加 4。

【例 3-19】 POP 指令的使用。

```
POP AX              ; 操作数出栈送寄存器(16 位)
POP ECX             ; 操作数出栈送寄存器(32 位)
POP [BX]            ; 操作数出栈送存储器(16 位)
POP DWORD PTR [SI]  ; 操作数出栈送存储器(32 位)
```

(2) 弹出寄存器数据格式：

```
POPA
POPAD
```

功能：POPA 从堆栈移出 16 字节数据，并且按顺序存入寄存器 DI、SI、BP、SP、BX、DX、CX、AX 中。

POPAD 从堆栈移出 32 字节数据，并且按顺序存入寄存器 EDI、ESI、EBP、ESP、EBX、EDX、ECX、EAX 中。

注意：

- 目的操作数不允许为 CS 以及立即数；
- 堆栈操作指令不影响标志位。

【例 3-20】 POP AX 指令的执行情况，如图 3-12 所示。

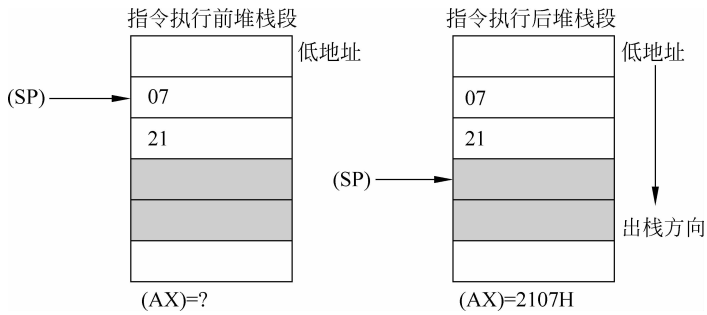

图 3-12　POP AX 指令的执行情况

3.3.3　地址传送指令

1. 取有效地址指令格式

```
LEA REG,MEM
```

功能：将源操作数的有效地址传送到通用寄存器，操作数 REG 为 16 位或 32 位通用寄存器，源操作数为 16 位或 32 位存储器操作数。

执行的操作

```
(REG)←(MEM)
```

【例 3-21】 LEA 指令的使用。

```
LEA BX,BLOCK          ; 将 BLOCK 的有效地址传送到 BX 中(16 位)
LEA EAX,[\EBX\]       ; 将 EBX 内容(有效地址)传送到 EAX 中(32 位)
```

2. 有效地址送寄存器指令格式

```
LDS(ES,FS,GS,SS)REG,MEM
```

功能：根据源操作数指定的偏移地址，在数据段中取出段地址和偏移地址分别送指定的段寄存器和指定的通用寄存器。

执行的操作：

```
(REG)←(MEM)
(SREG)←(MEM + 2)/ (MEM + 4)
```

【例 3-22】 LES 指令的使用。

```
LES BX,[SI]           ; 将 32 位地址指针分别送 ES 和 BX 即(BX)←(([SI]),(ES))←([SI + 2])
```

假定指令执行前(DS)＝B0000H，(SI)＝080AH，(B080AH)＝05AEH，(B080CH)＝4000H，则指令执行后，(BX)＝05AEH，(ES)＝4000H。

【例 3-23】 LSS 指令的使用。

```
LSS ESP,MEM           ; 将 48 位地址指针分别送 ESP 和 SS 寄存器。
```

地址传送指令对标志位无影响。

3.3.4 标志寄存器传送指令

1. 标志寄存器送 AH 指令

格式：

```
LAHF(LOAD AH WITH FLAGS)
SAHF(STORE AH INTO FLAGS)
```

功能：LAHF 将标志寄存器中低 8 位送 AH 中。SAHF 将 AH 中内容送标志寄存器中低 8 位。

执行的操作：

```
LAHF:(AH)←(FLAGS 的低字节)。
SAHF:(FLAGS 的低字节)←(AH)。
```

2. 标志寄存器内容入/出栈指令

格式：

```
PUSHF(PUSH THE FLAGS)
POPF(POP THE FLAGS)
```

功能：PUSHF 将标志寄存器低 16 位内容压入堆栈。POPF 将当前栈顶一个字传送到标志寄存器低 16 位中。

执行的操作：

```
PUSHF:  (SP)←(SP) - 2
        ((SP) + 1,(SP))←(FLAGS)
POPF:   (FLAGS)←((SP) + 1,(SP))
        (SP)←(SP) + 2
```

3. 32 位标志寄存器内容入/出栈

格式：

```
PUSHFD(PUSH THE EFLAGS)
POPFD(POP THE EFLAGS)
```

功能：PUSHFD 将标志寄存器 32 位内容压入堆栈，SP←SP－4。POPFD 将当前栈顶一个双字传送到 32 位标志寄存器中，SP←SP＋4。

执行的操作：

```
PUSHFD: (ESP)←(ESP) - 4
((ESP) + 3,(ESP) + 2,(ESP) + 1,(ESP))←(EFLAGS AND 0FCFFFFH)
; 进行与操作是为了清除 VM 和 RF 位
POPFD:  (EFLAGS)←((ESP) + 3,(ESP) + 2,(ESP) + 1,(ESP))
        (ESP)←(ESP) + 4
```

上述 SAHF，POPF，POPFD 均影响相应的标志寄存器内容。

3.3.5　查表指令

格式：

```
XLAT
```

功能：将寄存器 AL 中的内容转换成存储器表格中的对应值。实现直接查表功能。

XLAT 指令规定：BX 寄存器存放表的首地址，AL 寄存器中存放表内偏移量，执行 XLAT 指令，以段寄存器 DS 的内容为段基址，有效地址为 BX 和 AL 内容之和，取出表中一个字节内容送 AL 中。

【例 3-24】 内存中有一起始地址为 TABLE 的编码表，试编程将表中顺序号为 4 的存储单元内容送寄存器 AL

```
          ·MODEL SMALL
          ·DATA
TABLE     DB 11H,22H,33H,44H,55H     ; 某编码表
          ·CODE
          ·STARTUP
          MOV AL,4                   ; AL←4
          MOV BX,OFFSET TABLE        ; BX←TABLE 表首地址
          XLAT                       ; 结果在 AL 中,AL = 55H
          ·EXIT
          END
```

查表指令不影响标志位。

3.3.6　类型转换指令

1. 字节扩展

格式：

```
CBW
```

功能：将 AL 中 8 位带符号数，进行带符号扩展为 16 位，送 AX 中。带符号扩展是指对正数高位扩展为全 0，对负数高位扩展为全 1。

【例 3-25】 AL＝55H，经 CBW 扩展后，AX＝0055H；AL＝A5H，经 CBW 扩展后，AX＝FFA5H。

2. 字扩展

格式：

```
CWD
```

功能：将 AX 中 16 位带符号数，进行带符号扩展为 32 位，送 DX 和 AX 中。高 16 位送 DX 中，低 16 位送 AX 中。

3. 双字扩展

格式：

```
CWDE
```

功能：将 AX 中 16 位带符号数，进行带符号扩展为 32 位，送 EAX 中。

4. 4 字扩展

格式：

```
CDQ
```

功能：将 EAX 中 32 位带符号数，进行带符号扩展为 64 位，送 EDX 和 EAX 中。低 32 位送 EAX 中，高 32 位送 EDX 中。

符号扩展指令对标志位无影响。

3.4 算术运算指令

80x86 指令包括加、减、乘、除 4 种基本算术运算操作及十进制算术运算调整指令。二进制加、减法指令，带符号操作数采用补码表示时，无符号数和带符号数据运算可以使用相同的指令。二进制乘、除法指令分带符号数和无符号数运算指令。

3.4.1 加法指令

格式：

```
ADD  DEST,SRC  ;执行的操作：(DEST)←(DESG) + (SRC)
ADC  DEST,SRC  ;执行的操作：(DEST)←(DESG) + (SRC) + CF
```

功能：ADD 是将源操作数与目的操作数相加，结果传送到目的操作数。ADC 是将源操作数与目的操作数以及 CF(低位进位)值相加，结果传送到目的操作数。

源操作数可以是通用寄存器、存储器或立即数。目的操作数可以是通用寄存器或存储器操作数。

ADD，ADC 指令影响标志位为 OF、SF、ZF、AF、PF、CF。

【例 3-26】 加法指令的使用。

```
MOV AX,9876H
ADD AH,AL    ; AX = 0E76H  CF = 1  SF = 0  OF = 0  ZF = 0  AF = 0  PF = 0
ADC AH,AL    ; AX = 8576H  CF = 0  SF = 1  OF = 1  ZF = 0  AF = 1  PF = 0
```

3.4.2 减法指令

格式：

```
SUB DEST,SRC  ; 执行的操作：(DEST)←(DESG) - (SRC)
SBB DEST,SRC  ; 执行的操作：(DEST)←(DESG) - (SRC) - CF
```

功能：SUB 将目的操作数减源操作数，结果送目的操作数。SBB 将目的操作数减源操作数，还要减 CF(低位借位)值，结果送目的操作数。

源操作数可以是通用寄存器、存储器或立即数。目的操作数可以是通用寄存器或存储器操作数。

SUB，SBB 指令影响的标志位有 OF、SF、ZF、AF、PF、CF。

【例 3-27】　减法指令的使用。

```
MOV AX,9966H          ;AX = 9966H
SUB AL,80H            ;AL = E6H   CF = 1   SF = 1   OF = 1   ZF = 0   AF = 0   PF = 0
SBB AH,80H            ;AH = 18H   CF = 0   SF = 0   OF = 0   ZF = 0   AF = 0   PF = 1
```

3.4.3　加 1 减 1 指令

格式：

```
INC DEST
DEC DEST
```

功能：INC 指令将目的操作数加 1，结果送目的操作数。DEC 指令将目的操作数减 1，结果送目的操作数。目的操作数为通用寄存器或存储器操作数。

INC 和 DEC 指令影响的标志位有 OF、SF、ZF、AF、PF，不影响 CF 位。

【例 3-28】　加减 1 指令的使用。

```
INC BL                ;(BL)←(BL) + 1
INC AX                ;(AX)←(AX) + 1
INC WORD  PTR [BX]    ;存储器中的字操作数加 1
DEC BYTE PTR [SI]     ;存储器中的字节操作数减 1
DEC EAX               ;(EAX)←(EAX) - 1
```

3.4.4　比较指令

1. 格式

```
CMP DEST,SRC      执行的操作：(DEST) - (SRC)
```

功能：目的操作数减源操作数，结果不回送，只是根据结果设置条件标志位。CMP 指令后往往跟着一个条件转移指令，根据比较结果产生不同的程序分支。源操作数为通用寄存器、存储器和立即数。目的操作数为通用寄存器、存储器操作数。

CMP 指令影响的标志位有 OF、SF、ZF、AF、PF、CF。

【例 3-29】　比较指令的使用。

```
CMP CX,3
CMP WORD PTR [SI],3
CMP AX,BLOCK
```

比较指令 CMP AX,BX 执行后，对状态标志位影响见表 3-4。

表 3-4　CMP 指令对标志位的影响

数据类型	关系	CF	ZF	SF	OF
带符号数	Dest＝src	0	1	0	0
	Dest＜src	—	0	1	0
		—	0	0	1
	Dest＞src	—	0	0	0
		—	0	1	1
无符号数	Dest＝src	0	1	0	0
	Dest＜src	1	0	—	—
	Dest＞src	0	0	—	—

对于两个数的比较(AX－BX)有以下 4 种情况。

(1) 两个正数比较,使用 SF 标志位判断。

SF＝0,则 AX≥BX,若 ZF＝1,则 AX＝BX; SF＝1,则 AX＜BX。

(2) 两个无符号数比较,使用 CF 标志位判断。

CF＝0,则 AX≥BX,若 ZF＝1,则 AX＝BX; CF＝1,则 AX＜BX。

(3) 两个负数比较,使用 SF 标志位判断。

SF＝0,则 AX≥BX,若 ZF＝1,则 AX＝BX; SF＝1,则 AX＜BX。

(4) 个异符号数比较。

如果 OF＝0,仍可用 SF 标志判断大小。如果 OF＝1,说明结果的符号位发生错误,所以 SF＝0,则 AX＜BX; SF＝1,则 AX＞BX。

综上所述,两个异号数比较:

当 OF＝0,SF＝0,则 AX＞BX; SF＝1,则 AX＜BX。当 OF＝1,SF＝0,则 AX＜BX; SF＝1,则 AX＞BX。

用逻辑表达式表示为:

若 OF ∀ SF＝0,则 AX＞BX; 若 OF ∀ SF＝1,则 AX＜BX。

2. 比较并交换指令

格式:

```
CMPXCHG  DEST,REG
```

功能:比较并交换目的操作数和源操作数。

执行的操作:累加器 AC 与 DEST 比较。

如果(AC)＝(DEST),则 ZF←1 (DEST)←(SRC)。

否则,ZF←0,(AC)←(DEST)。

源操作数允许为通用寄存器,目的操作数可以为通用寄存器,存储器操作数。

CMPXCHG 影响标志位为 OF,SF,ZF,AF,PF,CF。

3. 比较并部分交换指令

格式:

```
CMPXCHG8B  MEM
```

功能:比较并交换 8 字节。

执行的操作:“EDX”:“EAX”中值与 MEM 比较。

如果(EDX: EAX)＝MEM,则 ZF←1,(MEM)←(EDX: EAX); 否则,则 ZF←0,(EDX: EAX)←(MEM)。

该指令为 64 位比较交换指令,影响 ZF 标志位。

3.4.5 交换相加指令

格式:

```
XADD  DEST,SRC
```

功能：目的操作数加源操作数，结果送目的操作数，原目的操作数内容送源操作数。源操作数允许为通用寄存器。目的操作数允许为通用寄存器、存储器操作数。

执行的操作：TEMP←(SRC)＋(DEST)，(SRC)←(DEST)，(DEST)←TEMP。

XADD 指令影响标志位有 OF、SF、ZF、AF、PF、CF。

【例 3-30】 编写程序实现 w←(x＋y＋24－z)。其中，x，y，z，w 是双精度数，分别存放在地址 x，x＋2；y，y＋2；z，z＋2；w，w＋2 的存储单元中。

```
;EX330.ASM 加减运算举例   W←(X+Y+24-Z)
DATA SEGMENT                        ; 定义数据段
  X DW 1122H,3344H
  Y DW 5566H,7788H
  Z DW 1122H,3344H
  W DW ?,?
DATA ENDS

CODE SEGMENT                        ; 定义代码段
MAIN PROC FAR                       ; 代码段中的主过程
     ASSUME CS:CODE,DS:DATA         ; 段寄存器与对应段定义的指定
START:
     PUSH DS                        ; 保护原有数据
     XOR AX,AX
     PUSH AX

     MOV AX,DATA                    ; 具体的数据段与段寄存器的对应
     MOV DS,AX

     MOV AX,X                       ; 取出数据段中各存储单元中的值
     MOV DX,X+2
     ADD AX,Y                       ;(AX)←(AX)+(Y)
     ADC DX,Y+2                     ;(DX)←(DX)+(Y+2)+CF
     ADD AX,24
     ADC DX,0
     SUB AX,Z                       ;(AX)←(AX)-(Z)
     SBB DX,Z+2                     ;(DX)←(DX)-(Z+2)-CF
     MOV W,AX
     MOV W+2,DX                     ;运算结果存放在 W 单元中
     RET

MAIN ENDP                           ; 主过程结束
CODE ENDS                           ; 代码段结束
     END START                      ; 汇编程序结束
```

3.4.6　求补指令

格式：

```
NEG  DEST
```

功能：对目的操作数求补，用 0 减去目的操作数，结果送目的操作数。目的操作数为通

用寄存器、存储器操作数。

执行的操作：(DEST)←－(DEST)。

对十进制数据而言就是求相反数，对二进制数据而言就是对操作数求反末位加 1。

NEG 指令影响标志位有 OF、SF、ZF、AF、PF、CF。

3.4.7 乘法指令

1. 单操作数乘法指令

格式：

```
MUL  SRC
IMUL  SRC
```

功能：MUL 为无符号数乘法指令，IMUL 为带符号数乘法指令。源操作数为通用寄存器或存储器操作数。目的操作数默认存放在 ACC(AL，AX，EAX)中，乘积存 AX、DX：AX 或 EDX：EAX 中。

执行的操作：无符号数乘法指令 MUL。

字节乘：(AX)←(AL)×(SRC)。

字乘：(DX：AX)←(AX)×(SRC)。

双字乘：(EDX：EAX)←(EAX)×(SRC)。

有符号数乘法指令 IMUL 的执行过程与无符号数乘法指令 MUL 的执行操作一样，但要求操作数是带符号数。

“MUL，IMUL”指令执行后，CF＝OF＝0，表示乘积高位无有效数据；CF＝OF＝1 表示乘积高位含有效数据，对其他标志位无定义。

【例 3-31】 乘法指令的使用。

```
MUL BL                          ; 字节乘
MUL WORD PTR [SI]               ; 字乘
IMUL BYTE PTR [DI]              ; 字节乘
IMUL DWORD PTR [ECX]            ; 双字乘
```

如果使用 IMUL 指令，积采用补码形式表示。

2. 双操作数乘法指令

格式：

```
IMUL  DEST,SRC
```

功能：将目的操作数乘以源操作数，结果送目的操作数。目的操作数为 16 位或 32 位通用寄存器或存储器操作数。源操作数为 16 位或 32 位通用寄存器、存储器或立即数。

源操作数和目的操作数数据长度要求一致。乘积仅取和目的操作数相同的位数，高位部分将被舍去，并且 CF＝OF＝1。其他标志位无定义。

执行的操作：(DEST)←(DEST)×(SRC)。

3. 三操作数乘法指令

格式：

```
IMUL DEST,SRC1,SRC2
```

功能：将源操作数 SRC1 与源操作数 SRC2 相乘，结果送目的操作数 DEST。目的操作数 DEST 为 16 位或 32 位，允许为通用寄存器。源操作数 SRC1 为 16 位或 32 位通用寄存器或存储器操作数。源操作数 SRC2 允许为立即数。

执行的操作：(DEST)←(SRC1)×(SRC2)。

要求目的操作数和源操作数 SRC1 类型相同，当乘积超出目的操作数部分，将被舍去，并且使 CF＝OF＝1，在使用这类指令时，需在 IMUL 指令后加一条判断溢出的指令，溢出时转错误处理执行程序。

3.4.8　除法指令

格式：

```
DIV SRC
IDIV SRC
```

功能：DIV 为无符号数除法，IDIV 为带符号数除法。源操作数作为除数，为通用寄存器或存储器操作数。被除数默认在目的操作数 AX，DX：AX，EDX：EAX 中。

执行的操作：

字节除法：(AL)←(AX)/(SRC)的商，(AH)←余数。

字除法：(AX)←(DX・AX)/(SRC)的商，(DX)←余数。

双字除法：(EAX)←(EDX・EAX)/(SRC)商，(EDX)←余数。

由于被除数必须是除数的双倍字长，一般应使用扩展指令进行高位扩展。当进行无符号数除法时，被除数高位按 0 扩展为双倍除数字长。当进行有符号数除法时，被除数以补码表示。可使用扩展指令 CBW，CWD，CWDE，CDQ 进行高位扩展。

例如：

```
MOV AX,BLOCK
CWD                                ; 被除数高位扩展
MOV BX,1000H
IDIV BX
```

对于带符号除法，其商和余数均采用补码形式表示，余数与被除数同符号。当除数为 0 或商超过了规定数据类型所能表示的范围时，将会出现溢出现象，产生一个中断类型码为 0 的中断。执行除法指令后标志位无定义。

【例 3-32】 编写程序实现(V－(X×Y＋Z－540)) /X，其中，X，Y，Z，V 均为 16 位带符号数，分别存放在 X、Y、Z、V 单元中，要求计算结果存放在 AX 寄存器，余数存放在 DX 寄存器中。编写程序如下：

```
;EX332.ASM 算术运算举例    (V-(X*Y+Z-540)) /X
```

```
DATA SEGMENT                    ; 定义数据段
  X   DW 1111H
  Y   DW 2222H
  Z   DW 3333H
  V   DW 9999H
DATA ENDS

CODE SEGMENT                    ; 定义代码段
MAIN PROC FAR                   ; 代码段中的主过程
     ASSUME CS:CODE,DS:DATA     ; 段寄存器与对应段定义的指定
START:
     PUSH DS                    ; 保护原有数据
     XOR AX,AX
     PUSH AX

     MOV AX,DATA                ; 具体的数据段与段寄存器的对应
     MOV DS,AX

     MOV AX,X                   ; 取出数据段中各存储单元中的值
     IMUL Y                     ; (AX)←(AX) × Y
     MOV CX,AX
     MOV BX,DX
     MOV AX,Z
     CWD                        ; 将 W 存储单元中的值扩展为双字
     ADD CX,AX
     ADC BX,DX                  ;X × Y + Z
     SUB CX,540
     SBB BX,0
     MOV AX,V
     CWD
     SUB AX,CX                  ;(AX)←(AX) - (CX)
     SBB DX,BX                  ;(DX)←(DX) - (BX) - CF
     IDIV X                     ;(AX)←(AX,DX) / (V)
     RET                        ;返回 DOS
     MAIN ENDP                  ; 主过程结束
CODE ENDS                       ; 代码段结束
     END START                  ; 汇编程序结束
```

3.4.9 BCD 算术运算

十进制数在机器中采用 BCD 码表示，以压缩格式存放，即一个字节存储 2 位 BCD 码，BCD 加减法是在二进制加减运算的基础上，对其二进制结果进行调整，将结果调整成 BCD 码表示形式。

1. 加法运算的调整指令

格式：

```
DAA
```

功能：将存放在 AL 中的二进制和数，调整为压缩格式的 BCD 码表示形式。

调整方法：若 AL 中低 4 位大于 9 或标志 AF＝1(表示低 4 位向高 4 位有进位)，则 AL＋6→AL，1→AF；若 AL 中高 4 位大于 9，或标志 CF＝1(表示高 4 位有进位)，则 AL＋60H→AL，1→CF。

DAA 指令一般紧跟在 ADD 或 ADC 指令之后使用，影响标志位为 SF、ZF、AF、PF、CF，OF 无定义。

2. 减法运算的调整指令

格式：

```
DAS
```

功能：将存放在 AL 中的二进制差数，调整为压缩的 BCD 码表示形式。

调整方法：若 AL 中低 4 位大于 9 或标志 AF＝1(表示低 4 位向高位借位)，则 AL－6→AL，1→AF；若 AL 中高 4 位大于 9 或标志 CF＝1(表示高 4 位向高位借位)，则 AL－60H→AL，1→CF。

DAS 指令一般紧跟在 SUB 或 SBB 指令之后使用，影响标志位为 SF、ZF、AF、PF、CF，OF 无定义。

3.4.10　ASCII 算术运算

数字 0～9 的 ASCII 码为 30H～39H，机器采用一个字节存放一位 ASCII 码，对于 ASCII 码的算术运算是在二进制运算基础上进行调整。调整指令有加、减、乘、除 4 种调整指令。

1. 加法运算的 ASCII 调整指令

格式：

```
AAA
```

功能：将存放在 AL 中的二进制和数，调整为 ASCII 码表示的结果。

调整方法：若 AL 中低 4 位小于或等于 9，仅 AL 中高 4 位清 0，AF→CF；若 AL 中低 4 位大于 9 或标志 AF＝1(进位)，则 AL＋6→AL，AH＋1→AH，1→AF，AF→CF，AL 中高 4 位清 0。

AAA 指令一般紧跟在 ADD 或 ADC 指令之后使用，影响标志位为 AF、CF，其他标志位无定义。

【例 3-33】 AAA 指令的使用。

```
MOV AX,0036H
ADD,AL,35H
AAA        ; 指令执行后(AX) = 0101H
```

2. 减法运算的 ASCII 调整指令

格式：

```
AAS
```

功能：将存放在 AL 中的二进制差数，调整为 ASCII 码表示形式。

调整方法：若 AL 中低 4 位小于等于 9，仅 AL 中高 4 位清 0，AF→CF；若 AL 中低4 位大于 9 或标志 AF＝1，则 AL－6→AL，AH－1→AH，1→AF，AF→CF，AL 中高 4 位清 0。

AAS 指令一般紧跟在 SUB、SBB 指令之后使用，影响的标志位有 AF、CF，其他标志位无定义。

【例 3-34】 AAS 指令的使用。

```
MOV AX,0132H
SUB AL,35H
AAS                              ; 指令执行后(AX) = 0007H
```

3. 乘法运算的 ASCII 调整指令

格式：

```
AAM
```

功能：将存放在 AL 中的二进制积数，调整为 ASCII 码表示形式。

调整方法：AL/10 商→AH，余数→AL。

AAM 指令一般紧跟在 MUL 指令之后使用，影响的标志位有 SF、ZF、PF，其他标志位无定义。

【例 3-35】 AAM 指令的使用。

```
MOV AL,07H
MOV BL,09H
MUL BL                           ; (AX) = 003FH
AAM                              ; 指令执行后(AX) = 0603H
```

4. 除法运算的 ASCII 调整指令

格式：

```
AAD
```

功能：将 AX 中两位非压缩 BCD 码（一个字节存放一位 BCD 码），转换为二进制数的表示形式。

调整方法：AH 10＋AL→AL0→AH。

AAD 指令用于二进制除法 DIV 操作之前，影响的标志位有 SF、ZF、PF，其他标志位无定义。

【例 3-36】 AAD 指令的使用。

```
MOV AX,0605H
MOV BL,09H
AAD                              ; (AX) = 0041H
DIV BL                           ; 指令执行后(AX) = 0207H
```

使用该类指令应注意，加法、减法和乘法调整指令都是紧跟在算术运算指令之后，将二

进制的运算结果调整为非压缩 BCD 码表示形式，而除法调整指令必须放在除法指令之前进行，以避免除法出现错误的结果。

注意：

- 如果没有特别规定，参与运算的两个操作数数据类型必须一致，且只允许一个为存储器操作数；
- 如果参与运算的操作数只有一个，且为存储器操作数，必须使用 PTR 伪指令说明数据类型；
- 操作数不允许为段寄存器；
- 目的操作数不允许为立即数；
- 如果是存储器寻址，则存储器各种寻址方式均可使用。

3.5　控制转移类指令

计算机执行程序一般是顺序地逐条执行指令，经常需要根据不同条件做不同的处理，有时需要跳过几条指令，有时需要重复执行某段程序，或转移到另一个程序段去执行。用于控制程序流程的指令包括转移、循环、过程调用和中断调用。

3.5.1　转移指令

1. 无条件转移指令

格式：

```
JMP TARGET
```

功能：使程序无条件地转移到指令规定的目的地址 TARGET 去执行指令。转移分为短转移、段内转移(近程转移)和段间转移(远程转移)。

1) 段内直接转移

格式：

```
JMP SHORT TARGET     ; 段内直接短转移
```

执行的操作：(IP)←(IP)＋8 位位移量。

```
JMP NEAR PTR TARGET  ; 段内直接近转移
```

执行的操作：(IP)←(IP)＋16 位位移量。

功能：采用相对寻址将当前 IP 值(即 JMP 指令下一条指令的地址)与 JMP 指令中给出的偏移量之和送 IP 中。段内短转移(SHORT)指令偏移量为 8 位，允许转移偏移值的范围为－128～＋127。段内近程转移(NEAR)指令在 16 位指令模式下，偏移量为 16 位，允许转移偏移值范围为-2^{15}～$+2^{15}-1$。在 32 位指令模式下，偏移值范围为-2^{31}～$+2^{31}-1$。

【例 3-37】 JMP 指令的使用。

```
JMP NEXT
  ⋮
```

```
NEXT: MOV AL,BL
```

本例为无条件转移到本段内标号为 NEXT 的地址去执行指令，汇编程序可以确定目的地址与 JMP 指令的距离。

2）段内间接转移

格式：

```
JMP REG
JMP NEAR PTR [REG]
```

功能：段内间接转移，其中 JMP REG 指令地址在通用寄存器中，将其内容直接送 IP 实现程序转移。JMP NEAR PTR [REG]指令地址在存储器中，默认段寄存器根据参与寻址的通用寄存器来确定，将指定存储单元的字取出直接送 IP 实现程序转移。在 16 位指令模式，转移偏移值范围为$-2^{15}\sim2^{15}-1$。在 32 位指令模式，转移偏移值范围为$-2^{31}\sim+2^{31}-1$。

【例 3-38】 段内转移指令的使用。设 DS＝1000H，EBX＝00002000H。

```
JMP BX                          ; 将 2000H 送 IP
JMP NEAR PTR [BX]               ; 将地址 1000∶2000 单元存放的一个字送 IP
JMP NEAR PTR [EBX]              ; 将段选择符为 1000H，偏移地址为 00002000H 单元存放的双字送 EIP
```

3）段间直接转移

格式：

```
JMP FAR PTR TARGET
```

功能：段间直接转移，FAR PTR 说明标号 TARGET 具有远程属性。将指令中由 TARGET 指定的段值送 CS，偏移地址送 IP。

执行的操作：(IP)←TARGET 的段内偏移地址，(CS)←TARGET 所在段的段地址。

386 及后继机型，偏移地址送 EIP。

在 16 位指令模式下，段地址送 CS，偏移地址为 16 位，转移偏移值范围$-2^{15}\sim+2^{15}-1$；在 32 位指令模式下，段地址送 CS，偏移地址为 32 位，转移偏移值范围为$-2^{31}\sim+2^{31}-1$。

4）段间间接转移

格式：

```
JMP FAR PTR [Reg]
```

功能：段间间接转移，由 FAR PTR [Reg]指定的存储器操作数作为转移地址。

在 16 位指令模式下，存储器操作数为 32 位，包括 16 位段基址和 16 位偏移地址。

【例 3-39】 段间转移指令的使用。

```
JMP FAR PTR [BX]                ; 数据段双字存储单元低字内容送 IP
                                ; 数据段双字存储单元高字内容送 CS
```

在 32 位指令模式下，存储器操作数包括 16 位选择符。

2. 条件转移指令

该类指令是根据上一条指令对标志寄存器中标志位的影响来决定程序执行的流程，若

满足指令规定的条件，则程序转移；否则，程序顺序执行。这类似于高级语言中选择语句，只有 IF 子句，无 ELSE 子句。

条件转移指令的转移范围为段内短转移或段内近程转移，不允许段间转移。段内短转移(short)的转移偏移值范围为－128～＋127。段内近程转移，在 16 位指令模式下转移偏移值范围为-2^{15}～$+2^{15}-1$，在 32 位指令模式下转移偏移值范围为-2^{31}～$+2^{31}-1$。

条件转移指令包括单标志位条件转移、无符号数比较条件转移、带符号数比较条件转移和测试 CX 条件转移 4 类。

格式：

```
JCC TARGET
```

功能：若测试条件‘CC’为真，则转移到目标地址 TARGET 处执行程序，否则顺序执行。

(1) 单标志位条件转移指令，如表 3-5 所示。

【例 3-40】

```
JZ NEXT                          ; 若标志 ZF = 1 则转移到标号 NEXT 处执行
```

(2) 无符号数比较条件转移，如表 3-6 所示。

【例 3-41】

```
JA NEXT                          ; 无符号数 A 与 B 比较,若 A>B 则转移到标号 NEXT 处执行程序
```

(3) 带符号数比较条件转移，如表 3-7 所示。

【例 3-42】

```
JG NEXT                          ; 带符号数 A 与 B 比较,若 A>B 则转移到标号 NEXT
```

(4) 测试 CX 条件转移，如表 3-8 所示。

表 3-5　单标志位条件转移指令

助记符	判断条件	说　明	助　记　符	判断条件	说　明
JO	OF＝1	溢出则转移	JNZ/JNE	ZF＝0	非零/不等于则转移
JNO	OF＝0	无溢出转移	JP/JPE	PF＝1	奇偶位为 1 则转移
JS	SF＝1	负数则转移	JNP/JPO	PF＝0	奇偶位为 0 则转移
JNS	SF＝0	正数则转移	JC/JB/JNAE	CF＝1	进位/低于/不高于或等于转移
JZ/JE	ZF＝1	0/等于则转移	JNC/JNB/JAE	CF＝0	无进位/不低于/高于或等于转移

表 3-6　无符号数比较条件转移指令

助　记　符	判 断 条 件	说　明
JA/JNBE	CF∨ZF＝0	高于/不低于且不等于转移
JAE/JNB	CF＝0	高于等于/不低于转移
JB/JNAE	CF＝1	低于/不高于且不等于转移
JBE/JNA	CF∨ZF＝1	低于等于/不高于转移

表 3-7 带符号数比较条件转移指令

助 记 符	判 断 条 件	说 明
JL/JNGE	SF ∀OF=1	< 即小于/不大于或等于则转移
JNL/JGE	SF ∀OF=0	≥ 即不小于/大于或等于则转移
JLE/JNG	(SF ∀OF) ∨ ZF=1	≤ 即小于或等于/不大于则转移
JNLE/JG	(SF ∀OF) ∨ ZF=0	> 即不小于或等于/大于则转移

表 3-8 测试 CX 条件转移指令

助 记 符	判 断 条 件	说 明
JCXZ	(CX)=0	CX 内容为 0 则转移,转移范围-128～+127
JECXZ	(ECX)=0	ECX 内容为 0 则转移,转移范围-128～+127

【例 3-43】

```
JCXZ TARGET                    ; CX = 0 转移到标号 TARGET 处
JECXZ TARGET                   ; ECX = 0 转移到标号 TARGET 处
```

条件转移指令一般紧跟在 CMP 或 TEST 指令之后,判断执行 CMP 或 TEST 指令对标志位的影响来决定是否转移。

【例 3-44】 符号函数

$$f(X)=\begin{cases}-1, & X<0\\ 0, & X=0\\ 1, & X>0\end{cases}$$

假设 x 为某值且存放在寄存器 AL 中,试编程将求出的函数值 f(x)存放在 AH 中。程序如下:

```
; EX344.ASM  跳转指令的使用 f(X) = { -1, X<0 ; 0, X=0 ; 1, X>0
    .MODEL TINY                ; 精简模式的编程模式
    .DATA                      ; 数据段的定义
        X DB -8
    .CODE                      ; 代码段的定义
    STARTUP:
        MOV AX,@DATA           ; 预定义符@DATA 取出数据段的段地址
        MOV DS,AX
        MOV AL,X               ;(AL)←(X)
        CMP AL,0               ;X 与 0 比较大小
        JGE BIG                ; 若 X>=0,则跳到标号为 BIG 的语句处执行
        MOV AL,0FFH            ; 若 X<0,则(AL)←-1,并跳到 DONE 处执行
        JMP DONE
       BIG:  JE DONE           ; 若 X==0,则跳到 DONE 处执行
        MOV AL,1               ; X>0,(AL)←1
       DONE: MOV AH,AL
        MOV AX,4C00H           ; 返回 DOS 环境
        INT 21H
    END  STARTUP
```

【例 3-45】 编程实现把 BX 寄存器内的二进制数用十六进制数的形式在屏幕上显示出来。

程序如下：

```
;EX345.ASM 跳转指令实现二到十六进制的转换
  .MODEL TINY
  .CODE
  STARTUP:
         MOV BX,1001010101000111B ;为了简化程序,直接在 BX 寄存器中放入要转换的数据
         MOV CH,4                 ;设置每 4 位为一组
   AGAIN:  MOV CL,4
           ROL BX,CL              ;循环左移,把 BX 中的最高 4 位移动到最右边
           MOV AL,BL
           AND  AL,0FH            ;屏蔽掉 AL 的高 4 位,只取原有数据的最高 4 位
           ADD  AL,30H            ;数字数据转换为对应的数据字符,便于输出
           CMP  AL,3AH
           JB NEXT                ;字符值比 3AH 小,直接输出
           ADD  AL,07H            ;若转换为的字符比 9 大,则转换为字符 A—F
    NEXT:  MOV DL,AL              ;输出 DL 中的字符
           MOV AH,2
           INT 21H
           DEC CH
           JNZ AGAIN

           MOV AX,4C00H
           INT 21H
  END STARTUP
```

3.5.2 循环控制指令

这类指令用(E)CX 计数器中的内容控制循环次数，先将循环计数值存放在(E)CX 中，每循环一次(E)CX 内容减 1，直到(E)CX 为 0 时循环结束。循环指令不影响条件码。

格式：

```
LOOPcc TARGET
```

功能：将(E)CX 内容减 1，不影响标志位，若(E)CX 不等于 0，则转移到目标地址 TARGET 处执行程序。转移范围为－128～＋127。

执行的操作：(COUNT Reg)←(COUNT Reg)－1。检查是否满足测试条件，若满足，则转移到目标地址 TARGET 处执行程序；若不满足，则顺序执行下一条指令。

具体的三种格式为：

格式 1：

```
LOOP TARGET
```

测试条件：计数器(E)CX 的内容不为 0。

格式 2：

```
LOOPZ(LOOPE) TARGET
```

测试条件：ZF＝1 且计数器(E)CX 的内容不为 0。

格式 3：

```
LOOPNZ(LOOPNE) TARGET
```

测试条件：ZF＝0 且计数器(E)CX 的内容不为 0。

【例 3-46】 计算 $\sum_{i=0}^{10} i$。

```
; EX346.ASM 循环指令的使用 ∑(i=0,10) i
   .MODEL TINY
   .CODE
      START:
            XOR AX,AX
            MOV DX,1
            MOV CX,10              ;设置计数器 CX,从 1 累加到 10
      SUM:
            ADC AX,DX              ;部分累加和放在 AX 寄存器中
            INC DX                 ;判断测试条件 CX ≠ 0 是否成立,若成立则继续累加
            LOOP SUM               ;不成立,退出循环,顺序执行下一条指令
            MOV AX,4C00H
            INT 21H
      END START
```

【例 3-47】 找出以 ARRAY 为首地址的 100 个字数组中的第一个非 0 项，送 DX 寄存器中。

```
;EX347.ASM   用循环指令在数组中查找非 0 项
.MODEL SMALL
.DATA
   ARRAY DW 0,0,23,21,32,23,90,0,67,54
.STACK  100H
.CODE
   START:
       PUSH DS
       SUB AX,AX
        PUSH AX
        MOV AX,@DATA
        MOV DS,AX
        MOV CX,10                  ;数组长度存入计数器
        LEA BX,ARRAY
        MOV SI,-2
    SEARCH:
        INC SI
        INC SI
        CMP WORD PTR [BX+SI],0     ;各元素依次与 0 比较,遇到非 0 项或比较次数超过数组长度
                                   ;停止比较
        LOOPZ    SEARCH
        MOV DX,[BX+SI]             ;非 0 值存入 DX 寄存器
        MOV AX,4C00H
        INT 21H
 END START
```

3.6　串操作指令

80x86 提供字符串的操作。串指连续存放在存储器中的一些数据字节、字或双字。串操作允许程序对连续存放的数据块进行操作。

串操作通常以 DS:(E)SI 来寻址源串,以 ES:(E)DI 来寻址目的串,对于源串允许用段跨越前缀来修改数据段寄存器名。(E)SI 或(E)DI 这两个地址指针在每次串操作后,都自动进行修改,以指向串中下一个元素。地址指针修改是增量还是减量由方向标志来规定。当 DF=0,(E)SI 及(E)DI 的修改为增量;当 DF=1,(E)SI 及(E)DI 的修改为减量。根据串元素类型不同,地址指针增减量也不同,在串操作时,字节类型 SI,DI 加、减 1;字类型 SI,DI 加、减 2;双字类型 ESI,EDI 加、减 4。

3.6.1　重复前缀指令

如果需要连续进行串操作,通常加重复前缀。重复前缀可以和任何串操作指令组合,形成复合指令,如表 3-9 所示。

表 3-9　重复前缀指令

指令格式	判断条件	执行的操作
REP	(Count Reg) ≠ 0	计数器减 1,当计数器的值不为 0 时重复操作
REPE/REPZ	(Count Reg) ≠ 0 ∧ ZF=1	计数器减 1,当计数器的值不为 0 并且比较的结果相等时重复操作
REPNE/REPNZ	(Count Reg) ≠ 0 ∧ ZF=0	计数器减 1,当计数器的值不为 0 并且比较的结果不相等时重复操作

3.6.2　方向标志指令

格式:

```
CLD/STD
```

功能:CLD 为清除方向标志,即将 DF 置 0,地址从低到高。STD 为设置方向标志,即将 DF 置 1,地址方向从高到低。

3.6.3　串传送指令

基本格式:

```
[REP] MOVS DESTS,SRCS     ; 在操作数中指明是字节、字、双字
[REP] MOVSB  (字节)       ; 指令表明是字节操作
[REP] MOVSW  (字)         ; 指令表明是字操作
[REP] MOVSD  (双字)       ; 指令表明是双字操作
```

功能:将 DS:(E)SI 存储单元中的源串复制到 ES:(E)DI 存储单元的目的串中,同时

根据方向标志及数据格式对源变址寄存器和目的变址寄存器进行修改。MOVS 指令的寻址方式是隐含的，源串必须在数据段中，目的串必须在附加段中，但是源串可以用段跨越前缀来修改。在与 REP 指令连用时，还必须先把数据串的长度值送到计数寄存器中，以便结束指令，所以，在执行该指令前先做好以下准备工作：

(1) 把存放在数据段中的源串首地址(若是反向传递则是末地址)放入源变址寄存器中。

(2) 把要存放数据串附加段中的目的串首地址(若是反向传递则是末地址)放入目的变址寄存器中。

(3) 把数据段长度放入计数寄存器。

(4) 建立方向标志。

(5) 使用 MOVS 等指令实现数据串的复制操作。

该指令对标志位无影响。

如果加重复前缀 REP，则可以实现连续存放的数据块的传送，直到(E)CX＝0 为止。

在 16 位指令模式下，使用 SI、DI、CX 寄存器；在 32 位指令模式下，使用 ESI、EDI、ECX 寄存器。

【例 3-48】 在数据段中有一字符串，要求把它们传送到附加段的缓冲区中。

程序如下：

```
;EX348.ASM  把字符串从源缓冲区复制到目的缓冲区,用完整的段定义方式书写程序
DATAREA SEGMENT    ;定义数据段作为源缓冲区、字符串以 $ 结束,可以用 21H 号中断的 9 号功能输出
   MESS1  DB 'PERSONAL COMPUTER','$'
DATAREA ENDS
EXTRA   SEGMENT                          ;定义附加段作为目的缓冲区
   MESS2  DB 18 DUP(?)
EXTRA   ENDS
CODE SEGMENT                             ;定义代码段
MAIN PROC FAR                            ;定义主过程
   ASSUME CS:CODE,DS:DATAREA,ES:EXTRA    ;把段寄存器和定义的各个段对应起来
START:
     PUSH DS                             ;保护原有的数据段寄存器内容
     XOR AX,AX                           ;存储 0 作为新的堆栈段的开始
     PUSH AX
     MOV AX,DATAREA                      ;取出数据段 DATAREA 的首地址存入 DS 寄存器
     MOV DS,AX
     MOV AX,EXTRA                        ;取出附加段 EXTRA 的首地址存入 ES 寄存器
     MOV ES,AX
     CLD                                 ;设置字符串传送方向
     MOV CX,18                           ; 设置字符串传送个数
     LEA SI,MESS1                        ;源串地址送入 SI 寄存器中
     LEA DI,MESS2                        ;目的串地址送入 DI 寄存器中
     REP MOVSB                           ;循环传送字符直至计数器 CX 的值为 0
     MOV AX,EXTRA                        ;输出目的串的内容
     MOV DS,AX
     MOV DX,OFFSET MESS2
     MOV AH,9
     INT 21H
```

```
    RET                              ;返回 DOS
MAIN ENDP                            ;主过程结束
CODE ENDS                            ;代码段结束
    END START                        ;汇编程序结束
```

指令 REP MOVSB 的执行情况如图 3-13 所示。

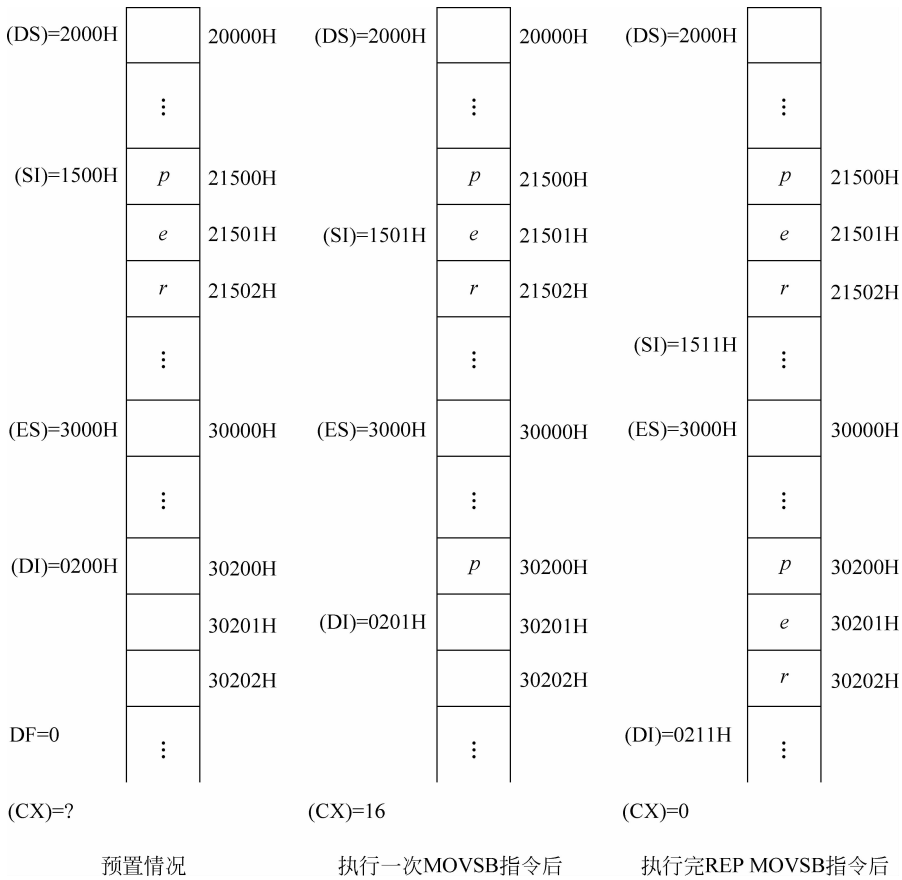

图 3-13　指令 REP MOVSB 的执行情况

3.6.4　串比较指令

基本格式：

```
[REPE/Z]     CMPS DESTS,SRCS          ; 字符相同且(CX)≠0 则继续比较
[REPNZ/NE]   CMPS DESTS,SRCS          ; 字符不相同且(CX)≠0 则继续比较
[REPE/Z]     CMPSB/CMPSW/CMPSD        ; 字符相同且(CX)≠0 则继续比较
[REPNZ/NE]   CMPSB/CMPSW/CMPSD        ; 字符不相同且(CX)≠0 则继续比较
```

功能：由 DS：(E)SI 指定的源串元素减去 ES：(E)DI 指定的目的串元素，结果不回送，仅根据比较的结果设置标志位 CF、AF、PF、OF、ZF、SF。当源串元素与目的串元素值相同时，ZF＝1；否则，ZF＝0。每执行一次串比较指令，根据 DF 的值和串元素数据类型自动修改(E)SI 和(E)DI。

在串比较指令前加重复前缀 REPE/Z，则表示重复比较两个字符串，若两个字符串的元素相同则继续比较直到(E)CX＝0 为止，否则结束比较。在串比较指令前加重复前缀 REPNE/NZ，则表示若两个字符串元素不相同时，重复比较直到(E)CX＝0 为止，否则结束比较。

【例 3-49】 编程实现两个串元素比较，如相同则输出“match!”信息，否则输出目的串中第一个不同的字符。

```
;EX349.ASM 两个字符串元素比较。可以更改 MESS1 和 MESS2 的内容,查看不同的运行结果
DATAREA SEGMENT                          ;定义数据段
   MESS1   DB 'PERSONAL COMPUTER',' $ '  ;定义源串的内容
   MESS3   DB 'MATCH!',' $ '
DATAREA ENDS

EXTRA   SEGMENT
   MESS2   DB 'PERSONAL COMPUTER',' $ '  ;定义目的串的内容
EXTRA   ENDS

CODE SEGMENT
MAIN PROC FAR
     ASSUME CS:CODE,DS:DATAREA,ES:EXTRA
START:
     PUSH DS
     XOR AX,AX
     PUSH AX

     MOV AX,DATAREA                      ;源串所在段的段地址存入 DS 寄存器
     MOV DS,AX

     MOV AX,EXTRA                        ;目的串所在段的段地址存入 ES 寄存器
     MOV ES,AX
                                         ;------- REP CMPSB ---------------
     CLD
     MOV CX,19
     LEA SI,MESS1                        ;源串所在段内偏移地址存入 SI 寄存器
     LEA DI,MESS2                        ;目的串所在段内偏移地址存入 DI 寄存器
     REPE CMPSB                          ;两个串对应字节内容相同则继续比较

     CMP CX,0                            ;是否比较到串结尾
     JNZ CON                             ;(CX)≠0,说明有不同的字符出现
     MOV DX,OFFSET MESS3                 ;比较到结尾,说明两个字符串相同
     MOV AH,9                            ;输出 MESS3 中的字符串
     INT 21H

 CON: DEC DI
     MOV AH,2                            ;输出目的串中第一个不同的字符
     MOV DL,ES:[DI]
     INT 21H

     RET
```

```
MAIN ENDP
CODE ENDS
     END START
```

3.6.5　串扫描指令

格式：

```
[REPE/Z]   SCAS DESTS          ; 字符相同且(CX)≠0 则继续扫描(AX)和 DESTS
[REPNE/NZ] SCAS DESTS          ; 字符不相同且(CX)≠0 则继续扫描(AX)和 DESTS
[REPE/Z] SCASB/SCASW/SCASD
                       ; 字符相同且(CX)≠0 则继续扫描(Ac)和 DESTS,Ac 指 AL、AX 或 EAX 寄存器
[REPNE/NZ] SCASB/SCASW/SCASD
                       ; 字符不相同且(CX)≠0 则继续扫描(Ac)和 DESTS,Ac 指 AL、AX 或 EAX 寄存器
```

功能：把 AL,AX 或 EAX 的内容与 ES:(E)DI 规定的目的串元素进行比较，由 AL、AX 或 EAX 的内容减去 ES:(E)DI 规定的目的串元素，结果不回送，仅影响标志位 CF、AF、PF、SF、OF,ZF。当 AL、AX 或 EAX 的值与目的串元素值相同时，ZF＝1；否则 ZF＝0。每执行一次串扫描指令，根据 DF 的值和串元素数据类型自动修改(E)DI。

在串扫描指令前加重复前缀 REPE/Z,则表示目的串元素值和累加器值相同时重复扫描，直到 CX/ECX＝0 为止，否则结束扫描。若加重复前缀 REPNE/NZ,则表示当目的串元素值与累加器值不相等时，重复扫描直到 CX/ECX＝0 时为止，否则结束扫描。

该指令影响标志位为 CF、AF、PF、SF、OF、ZF。

【例 3-50】 在内存 DEST 开始的 6 个单元寻找字符‘C’，如找到将字符‘C’的地址送 ADDR 单元，否则 0 送 ADDR 单元。

```
;EX350.ASM 在字符串中查找一指定元素,并将找到的地址送入某单元
DATAREA SEGMENT                ;定义数据段
   MESS1   DB   'ABCDEF'       ;定义要查找的源字符串
   ADDR    DW   ?
DATAREA ENDS
CODE SEGMENT
MAIN PROC FAR
     ASSUME CS:CODE,DS:DATAREA
START:
     PUSH DS
     XOR AX,AX
     PUSH AX

     MOV AX,DATAREA
     MOV ES,AX                 ;源串所在段的段地址存入 ES 寄存器
     MOV DS,AX
                               ;------ REPNE SCASB ---------------
     CLD
     MOV CX,6                  ;查找的字符串的长度
     LEA DI,MESS1              ;源串所在段内偏移地址存入 DI 寄存器
     MOV AL,'C'                ;要查找的字符存入 AL 寄存器
     REPNE SCASB               ;扫描(AL)与(ES:DI) 的内容,对应字节内容不相同则继续查找
```

```
    JZ QUIT                ;ZF = 1 即查找到,则跳转到 QUIT 处记录字符位置
    MOV DI,0               ; ZF = 0,说明未查找到相应的字符,则 DI 置 0
    JMP DONE

QUIT:DEC DI
DONE: MOV ADDR,DI          ;把位置标志送入 ADDR 存储单元中

    RET
MAIN ENDP
CODE ENDS
    END START
```

3.6.6 串装入指令

格式：

```
LODS SRCS
LODSB/LODSW/LODSD
```

功能：将 DS：SI/ESI 所指的源串元素装入累加器(AL、AX,EAX)中,并根据方向标志和数据类型修改源变址寄存器 SI/ESI 的内容。该指令一般不和 REP 指令连用,但可以和 LOOP 指令连用实现重复操作,并且不影响标志位。

3.6.7 串存储指令

格式：

```
[REP] STOS DESTS
[REP] STOSB/STOSW/STOSD
```

功能：将累加器(AL,AX,EAX)中的值存入 ES：DI/EDI 所指的目的串存储单元中,并且每存储一次,根据方向标志和数据类型修改源变址寄存器 SI/ESI 的内容。若加重复前缀 REP,则表示将累加器的值连续送目的串存储单元,直到 CX/ECX=0 时为止。

该指令不影响标志位。

3.7 逻辑运算指令

3.7.1 逻辑指令

1. 逻辑与指令 AND

格式：

```
AND DEST,SRC
```

功能：目的操作数和源操作数按位进行逻辑与运算,结果存目的操作数中。源操作数可以是通用寄存器、存储器或立即数。目的操作数可以是通用寄存器或存储器操作数。

AND 指令常用于将操作数中某位清 0(称屏蔽),只需将要清 0 的位与 0,其他不变的位与 1 进行逻辑与操作即可。

【例 3-51】 AND 指令的使用。

```
AND AL,0FH  ; 将 AL 中高 4 位清 0,低 4 位保持不变
```

AND 指令影响标志位为 SF、ZF、PF,并且使 OF＝CF＝0。

2. 逻辑或指令 OR

格式:

```
OR DEST,SRC
```

功能:目的操作数和源操作数按位进行逻辑或运算,结果存目的操作数中。源操作数可以是通用寄存器、存储器或立即数。目的操作数可以是通用寄存器或存储器操作数。

OR 指令常用于将操作数中某位置 1,只需将要置 1 的位与 1 进行或运算,其他不改变的位与 0 进行或运算。

【例 3-52】 OR 指令的使用。

```
OR AL,80H                    ; 将 AL 中最高位置 1
```

OR 指令影响标志位为 SF、ZF、PF,并且使 OF＝CF＝0。

3. 逻辑异或指令 XOR

格式:

```
XOR DEST,SRC
```

功能:目的操作数和源操作数按位进行逻辑异或运算,结果送目的操作数。源操作数可以是通用寄存器、存储器或立即数。目的操作数可以是通用寄存器或存储器操作数。

XOR 指令常用于将操作数中某些位取反,只需将要取反的位异或 1,其他不改变的位异或 0 即可。

【例 3-53】 XOR 指令的使用。

```
XOR AL,0FH                   ; 将 AL 中低 4 位取反,高 4 位保持不变
```

XOR 指令影响标志位为 SF、ZF、PF,并且使 OF＝CF＝0。

4. 逻辑非指令 NOT

格式:

```
NOT DEST
```

功能:对目的操作数按位取反,结果回送目的操作数。目的操作数可以为通用寄存器或存储器。

【例 3-54】 NOT 指令的使用。

```
NOT EAX
```

```
NOT BYTE PTR [BX]
```

NOT 指令对标志位无影响。

5. 测试指令 TEST

格式：

```
TEST DEST,SRC
```

功能：目的操作数和源操作数按位进行逻辑与操作，结果不回送目的操作数。源操作数可以为通用寄存器、存储器或立即数。目的操作数可以为通用寄存器或存储器操作数。

【例 3-55】 TEST 指令的使用。

```
TEST DWORD PTR [BX],80000000H
TEST AL,CL
```

TEST 指令常用于测试操作数中某位是否为 1，而且不会影响目的操作数。如果测试某位的状态，对某位进行逻辑与 1 的运算，其他位逻辑与 0，然后判断标志位。运算结果为 0，ZF＝1，表示被测试位为 0；否则 ZF＝0，表示被测试位为 1。

TEST 指令影响标志位为 SF、ZF、PF，并且使 OF＝CF＝0。

3.7.2 移位指令

移位指令对操作数按某种方式左移或右移，移位位数可以由立即数直接给出，或由 CL 间接给出。移位指令分一般移位指令和循环移位指令，如表 3-10 所示，每个指令执行的操作如图 3-14 所示。

表 3-10 移位指令

指 令	说 明
SHL(shift logical left)	逻辑左移
SHR(shift logical right)	逻辑右移
SAL(shift arithmetric left)	算术左移
SAR(shift arithnmetric right)	算术右移
ROL(rotat left)	循环左移
ROR(rotat right)	循环右移
RCL(rotat left through carry)	带进位循环左移
RCR(rotat right through carry)	带进位循环右移
SHLD(shift left double)	双精度左移
SHRD(shift right double)	双精度右移

1. 一般移位指令

1) 算术/逻辑左移指令

格式：

```
SAL DEST,OPRD
SHL DEST,OPRD
```

功能：按照操作数 OPRD 规定的移位位数，对目的操作数进行左移操作，最高位移入 CF 中。每移动一位，右边补一位 0。目的操作数可以为通用寄存器或存储器操作数。

SAL，SHL 指令影响标志位 OF、SF、ZF、PF、CF。左移 n 位，相当于原数据乘以 2^n。

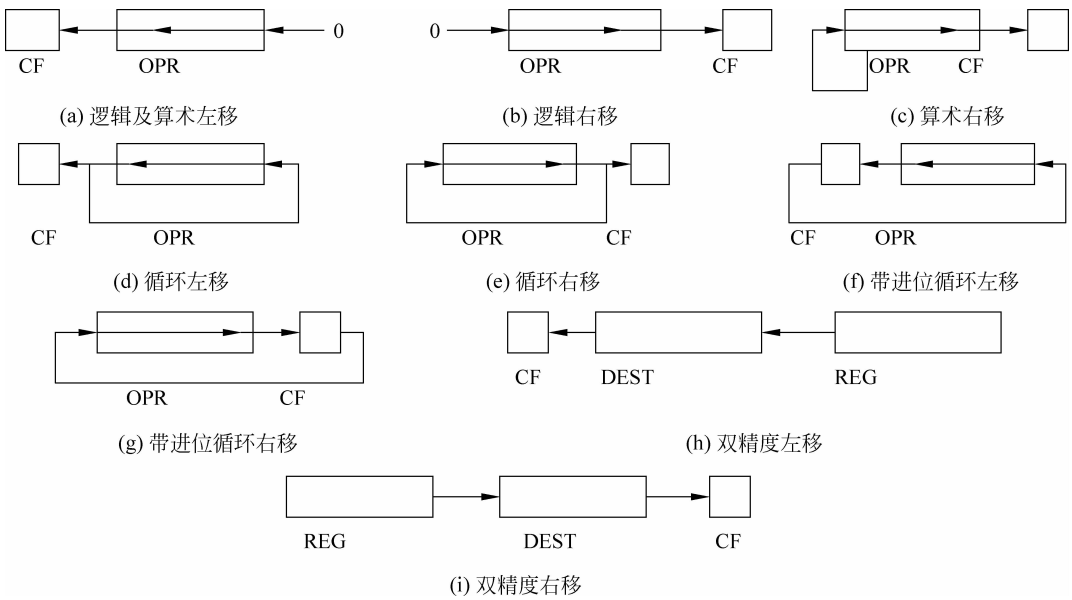

图 3-14　移位指令的操作

【例 3-56】 逻辑左移指令的使用。

```
MOV CL,2
SHLWORD PTR [DI],CL
```

指令执行前，(DS)＝0F800H，(DI)＝18OAH，(0F980A)＝0064H。

指令执行后，(0F980A)＝0190H。等价于 0064H×2^2d＝100d×4d＝400d。

```
SAL BX,CL                    ;指令执行前(BX) = 0064H,指令执行后(BX) = 0190H
```

2）算术右移指令

格式：

```
SAR DEST,OPRD
```

功能：按照操作数 OPRD 规定的移位次数，对目的操作数进行右移操作，最低位移至 CF 中，最高位(即符号位)保持不变。目的操作数可以为通用寄存器或存储器操作数。

SAR 指令影响标志位有 OF、SF、ZF、PF、CF。

3）逻辑右移指令

格式：

```
SHR DEST,SRC
```

功能：按照操作数 OPRD 规定的移位位数，对目的操作数进行右移操作，最低位移至 CF 中。每移动一位，左边补一位 0。目的操作数可以为通用寄存器或存储器操作数。

SHR 指令影响标志位有 OF、SF、ZF、PF、CF。

【例 3-57】 逻辑右移指令的使用。

```
MOV CL,2
SAR BX,CL                ;指令执行前(BX) = 0024H,指令执行后(BX) = 0009H
SHR BYTE PTR [SI],CL
                ;指令执行前(DS) = 13E5H,(SI) = 0010H,(13E60H) = 0024H,指令执行后(BX) = 0009H
```

算术/逻辑左移,只要结果未超出目的操作数所能表达的范围,每左移一次相当于原数乘 2。算术右移只要无溢出,每右移一次相当于原数除以 2。

2. 循环移位指令

格式:

```
ROL DEST,OPRD
ROR DEST,OPRD
RCL DEST,OPRD
RCR DEST,OPRD
```

功能:循环左移指令 ROL,目的操作数左移,每移位一次,其最高位移入最低位,同时最高位也移入进位标志 CF。循环右移指令 ROR,目的操作数右移,每移位一次,其最低位移入最高位,同时最低位也移入进位标志 CF。

带进位循环左移指令 RCL,目的操作数左移,每移动一次,其最高位移入进位标志 CF,CF 移入最低位。带进位循环右移指令 RCR,目的操作数右移,每移动一次,其最低位移入进位标志 CF,CF 移入最高位。

目的操作数可以为通用寄存器或存储器操作数。循环移位指令影响的标志位有 CF、OF,其他标志位无定义。

【例 3-58】 循环移位指令的使用。

```
MOV CL,4
ROL AL,CL                ;指令执行前(AL) = 1DH,指令执行后(AL) = D1H
ROR BX,CL                ;指令执行前(BX) = 001DH,指令执行后(BX) = D001H
RCR BYTE PTR [SI],CL
;指令执行前(DS) = 13E5H,(SI) = 0000H,(DS:SI) = (13E50H) = 1234H,BYTE PTR [SI] = 34H
;指令执行后(DS:SI) = (13E50H) = 1283H,BYTE PTR [SI] = 83H
```

【例 3-59】 将一个两位数压缩的 BCD 码转换成二进制数。

```
;EX359.ASM  将一个两位数压缩的 BCD 码转换成二进制数
    .MODEL SMALL
    .DATA
       X1 DB 01011101B
       X2 DB ?
    .CODE
       START:
       PUSH DS
       MOV AX,0
       PUSH AX
```

```
    MOV AX,@DATA
    MOV DS,AX

    MOV AL,X1
    MOV BL,AL
    AND BL,0FH        ;把 X1 的高 4 位屏蔽掉,只保留低 4 位
    AND AL,0F0H       ;把 X1 的低 4 位屏蔽掉,只保留高 4 位
    MOV CL,4
    ROR AL,CL         ;把 AL 中的高 4 位移动到最低 4 位
    MOV BH,0AH
    MUL BH
    ADD AL,BL
    MOV X2,AL
    MOV AX,4C00H
    INT 21H
END   START
```

3. 双精度移位指令

格式：

```
SHLD DEST,SRC,OPRD
SHRD DEST,SRC,OPRD
```

功能：对于由目的操作数 DEST 和源操作数 SRC 构成的双精度数，按照操作数 OPRD 给出的移位位数，进行移位。SHLD 是对目的操作数进行左移，SHRD 是对目的操作数进行右移。先移出位送标志位 CF，另一端空出位由 SRC 移入 DEST 中，而 SRC 内容保持不变。目的操作数可以是 16 位或 32 位通用寄存器或存储器操作数。源操作数 SRC 允许为 16 位或 32 位通用寄存器。操作数 OPRD 可以为立即数或 CL。目的操作数和源操作数 SRC 数据类型必须一致。

SHLD、SHRD 指令常用于位串的快速移位、嵌入和删除等操作，影响标志位为 SF、ZF、PF、CF，其他标志位无定义。

3.7.3　位操作指令

位操作指令包括位测试和位扫描指令，可以直接对一个二进制位进行测试、设置和扫描。

1. 位测试和设置指令

格式：

```
BT DEST,SRC  ; 将 SRC 指定的 DEST 中一位的数值复制到 CF
BTC DEST,SRC ; 将 SRC 指定的 DEST 中的一位数值复制到 CF,且将 DEST 中该位取反
BTR DEST,SRC ; 将 SRC 指定的 DEST 中的一位数值复制到 CF,且将 DEST 中该位复位
BTS DEST,SRC ; 将 SRC 指定的 DEST 中一位数值复制到 CF,且将 DEST 中该位置位
```

功能：按照源操作数指定的位号，测试目的操作数，当指令执行时，被测试位的状态被

复制到进位标志 CF。

目的操作数为 16 位或 32 位通用寄存器或存储器，源操作数为 16 位或 32 位通用寄存器，以及 8 位立即数。当源操作数为通用寄存器时，必须同目的操作数类型一致。源操作数 SRC 以两种方式给出目的操作数的位号：

- SRC 为 8 位立即数，以二进制形式直接给出要操作的位号；
- SRC 为通用寄存器，如果 DEST 为通用寄存器，则 SRC 中二进制值直接给出要操作的位号。如果 DEST 为存储器操作数，通用寄存器 SRC 为带符号整数，SRC 的值除以 DEST 的长度所得到的商作为 DEST 的相对偏移量，余数直接作为要操作的位号。DEST 的有效地址为 DEST 给出的偏移地址和 DEST 相对偏移量之和。

BT、BTC、BTR、BTS 指令影响 CF 标志位，其他标志位无定义。

【例 3-60】 位测试指令的使用。

```
MOV AX,1234H ;(AX) = 0001 0010 0011 0100,最右边第 0 位
MOV CX,5
BT AX,CX      ; CF = 1,(AX) = 1234H
BTC AX,CX     ; CF = 1,(AX) = 1214H = 0001 0010 0001 0100
BTS AX,CX;    ; CF = 0,(AX) = 1234H
BTR AX,CX     ; CF = 1,(AX) = 1214H
```

2. 位扫描指令

格式：

```
BSF  DEST,SRC
BSR  DEST,SRC
```

功能：BSF 从低位开始自右向左扫描源操作数，目的是检索第一个为 1 的位。若所有位都是 0，则 ZF＝1，目的寄存器无定义；否则 ZF＝1，并且将第一个出现 1 的位号存入目的操作数。BSR 从高位开始自左向右扫描源操作数，若所有位都是 0，则 ZF＝1，目的寄存器无定义；否则 ZF＝0，并且将第一个出现 1 的位号存入目的操作数。

源操作数可以为 16 位、32 位通用寄存器或存储器。目的操作数为 16 位或 32 位通用寄存器。源操作数和目的操作数类型必须一致。

BSF，BSR 指令影响 ZF 标志位，其他标志位无定义。

【例 3-61】 位扫描指令的使用。

```
MOV EBX,0F333EE00H
BSR EAX,EBX  ; ZF = 0,EAX = 0000001FH = 31
BSF EDX,EBX  ; ZF = 0,EDX = 00000009H
```

3. 进位标志指令

(1) 格式为 CLC。功能是，清除进位标志，CF←0。

(2) 格式为 STC。功能是，设置进位标志，CF←1。

(3) 格式为 CMC。功能是，进位标志取反，CF←CF 求反。

3.8　输入输出指令

3.8.1　IN 输入指令

功能：根据源操作数 SRC 给出的端口地址，将操作数从指定端口传送到目的操作数 DEST 处，其中 DEST 为 AL，AX 或 EAX，端口地址 SRC 可以直接给出 8 位端口地址，或由 DX 寄存器以间接形式给出。

1. 长格式：

```
IN  AL,PORT                                  ;(字节)
IN  AX,PORT                                  ;(字)
IN  EAX,PORT                                 ;(双字)
```

执行的操作：

```
(AL)←(PORT)                                  ;(字节)
(AX)←(PORT+1,PORT)                           ;(字)
(EAX)←(PORT+3,PORT+2,PORT+1,PORT)            ;(双字)
```

2. 短格式

```
IN  AL,DX                                    ;(字节)
IN  AX,DX                                    ;(字)
IN  EAX,DX                                   ;(双字)
```

执行的操作：

```
(AL)←((DX))                                  ;(字节)
(AX)←((DX)+1,(DX))                           ;(字)
(EAX)←((DX)+3,(DX)+2,(DX)+1,(DX))            ;(双字)
```

【例 3-62】 IN 指令的使用。

```
IN AL,10H       ;取出端口 10 的一个字节的内容送到 AL 寄存器
IN AX,20H       ;取出端口 20 的一个字的内容送到 AX 寄存器
IN EAX,30H      ;取出端口 30 的一个双字的内容送到 EAX 寄存器
IN AL,DX        ;以 DX 的内容作为端口号,从该端口中取一个字节的内容送到 AL 寄存器
IN AX,DX        ;以 DX 的内容作为端口号,从该端口中取一个字的内容送到 AX 寄存器
IN EAX,DX       ;以 DX 的内容作为端口号,从该端口中取一个双字的内容送到 EAX 寄存器
```

3.8.2　OUT 输出指令

功能：将源操作数 SRC 送到目的操作数 DEST 所指定的端口。其中，源操作数 SRC 为 AL，AX 或 EAX，目的操作数可以 8 位端口地址方式直接给出或以 DX 寄存器间接方式给出。

1. 长格式

```
OUT  PORT,AL                              ;(字节)
OUT  PORT,AX                              ;(字)
OUT  PORT,EAX                             ;(双字)
```

执行的操作：

```
(PORT)←(AL)                               ;(字节)
(PORT+1,PORT)←(AX)                        ;(字)
(PORT+3,PORT+2,PORT+1,PORT)←(EAX)         ;双字)
```

2. 短格式

```
OUT  DX,AL                                ;(字节)
OUT  DX,AX                                ;(字)
OUT  DX,EAX                               ;(双字)
```

执行的操作：

```
((DX))←  (AL)                             ;(字节)
((DX)+1,(DX))←(AX)                        ;(字)
((DX)+3,(DX)+2,(DX)+1,(DX))←(EAX)         ;(双字)
```

注意：

- 所有的 I/O 端口和 CPU 之间的通信都有 IN 和 OUT 指令来完成，IN 完成从 I/O 到 CPU 的信息传送，OUT 完成从 CPU 到 I/O 的信息传送；
- 直接寻址方式端口地址为 8 位，共有 0～255 个端口地址；
- 间接寻址方式，只能使用短格式，先把端口号放入 DX 寄存器中，然后从以 DX 寄存器的内容为端口号的端口中来传送信息。寻址范围为 64KB；
- 每个 I/O 地址对应的端口的数据长度为 8 位，传送 8 位数据占用一个端口地址，传送 16 位数据占用 2 个端口地址，传送 32 位数据占用 4 个端口地址。

3.8.3 串输入指令

格式：

```
[REP] INS DEST,DX
[REP] INSB、INSW、INSD
```

功能：根据 DX 给出的端口地址，从外设读入数据送入以 ES：DI/EDI 为地址的目的串存储单元中，每输入一次，均根据 DF 的值和串元素类型自动修改 DI/EDI 的值。若加重复前缀 REP，则表示连续从外设输入串元素存入目的串存储单元中，直到 CX/ECX＝0 为止。

执行的操作：

字节的操作：

```
((DEST))←((DX))        ;(字节)
(DEST)←(DEST)±1
```

字的操作：

```
((DEST))←((DX))        ;(字)
(DEST)←(DEST)±2
```

双字的操作：

```
((DEST))←((DX))        ;(双字)
(DEST)←(DEST)±4
```

3.8.4　串输出指令

格式：

```
[REP] OUTS DX,SRCS
[REP] OUTSB/OUTSW/OUTSD
```

功能：将 DS：SI/ESI 所指的源串元素，按照 DX 寄存器指定的端口地址送往外设，每输出一次，均根据 DF 的值和串元素类型自动修改 SI/ESI 的值，若加重复前缀 REP，则表示连续向外设输出串元素，直到 CX/ECX＝0 时为止。

在使用带重复前缀的串输入输出指令时，必须考虑端口的数据准备或接收状态。

所有输入输出指令均不影响标志位。

3.9　处理器控制

3.9.1　总线封锁前缀

格式：

```
LOCK 指令
```

功能：LOCK 为指令前缀，可以使 LOCK 引脚变成逻辑 0，在 LOCK 引脚有效期间，禁止外部总线上的其他处理器存取带有 LOCK 前缀指令的存储器操作数。

可加 LOCK 前缀的指令：

(1) ADD/ADC/SUB/SBB/OR/XOR/AND Mem，Reg/imm。

(2) NOT/NEG/INC/NEC Mem。

(3) XCHG Reg，Mem 或 XCHG Mem，Reg。

(4) BTS/BRT/BTC Mem，Reg/imm。

(5) CMPXCHG，XADD。

Mem 为存储器操作数，Reg 为通用寄存器，imm 为立即数。

3.9.2　空操作

格式：

```
NOP
```

功能：空操作，除使 IP/EIP 增 1 外，不做任何工作。该指令不影响标志位。在调试程序时往往用这条指令占有一定的存储单元，以便在正式运行时用其他指令取代。

3.9.3 处理器等待指令

格式：

```
WAIT
```

功能：检查 BUSY 引脚状态，等待协处理器完成当前工作。

3.9.4 处理器暂停指令

格式：

```
HLT
```

功能：暂停程序的执行。当产生一个外部中断或非屏蔽中断时，才继续执行下一条指令。

3.10 新增指令

3.10.1 80286 新增指令

1. 普通指令

普通指令共 4 条：

PUSHA：把 8 个寄存器值压入堆栈。

POPA：从堆栈中弹出数据恢复 8 个寄存器的值。

INS：字符串输入指令。

OUTS：字符串输出指令。

INS 指令的功能是从指定的端口输入一字符串到指定内存地址中去。可以使用 REP 前缀。

2. 高级指令

高级指令，共 3 条：

ENTER：进入过程，为过程保留堆栈空间和确定过程嵌套级。

LEAVE：退出过程，释放过程所占堆栈空间。

BOUND：检查地址寄存器的值是否在数组边界内。

3. 保护方式指令

这类指令用于实地址模式和保护虚地址模式的切换，并完成保护模式下的一些专门操作，共 16 条指令。

ARPL：调整请求和特权级别。

CLTS：清除任务切换标志位。

LAR：将段描述符中的存取权限装入寄存器。

LGDT：将从指定地址开始的 6 字节装入全局描述符表寄存器中。

LIDT：将从指定地址开始的 6 字节装入中断描述符表寄存器中。

LLDT：将 16 位值装入局部描述符表寄存器。

LMSW：装入机器状态字寄存器。

LSL：将段描述符中的段限值装入寄存器。

LTR：将 16 位值装入任务寄存器。

SGDT：把全局描述符表寄存器的内容存放到内存 6 字节单元。

SIDT：把中断描述符表寄存器的内容存放到内存 6 字节单元。

SLDT：将局部描述符表的 16 位值存入内存或寄存器中。

SMSW：存储机器状态字寄存器的值。

VERR：校验读访问。

VERW：校验写访问。

STR：存储任务寄存器(与 LTR 方向相反)。

3.10.2　80386 新增指令

测试与置位类指令。

该部分指令在 3.7.3 节已经详细介绍过，现总结如下。

格式 1：

```
BT 寄存器或存储器地址,寄存器或立即数
```

功能：位测试指令，用于检查指定位，并将该位复制到进位标志位中。

格式 2：

```
BTR  寄存器或存储器地址,寄存器或立即数
```

功能：位测试且取反指令，用于检查指定位，将该位复制到进位标志位中，并将原指定位取反再置入位。

格式 3：

```
BTR  寄存器或存储器地址,寄存器或立即数
```

功能：位测试且复位指令，用于检查指定位，将该位复制到进位标志位中，并将原指定位复位。

格式 4：

```
BTS  寄存器或存储器地址,寄存器或立即数
```

功能：位测试且置位指令，用于检查指定位，将该位复制到进位标志位中，并将原指定位置位。

3.10.3 80486 新增指令

1. 位扫描指令

格式 1：

BSF 寄存器,寄存器或存储器地址

功能：位扫描指令,它从源寄存器或存储器地址中的数的最低位(第 0 位)开始扫描直到置位位为止,并将该置位位的索引(位号)送入目的寄存器中。

格式 2：

BSR 寄存器,寄存器或存储器地址

功能：位扫描指令,它从源寄存器或存储器地址中的数的最高位(第 31.15 位)开始扫描直到置位位为止,并将该置位位的索引(位号)送入目的寄存器中。

2. 数的传送与扩展指令

格式 1：

MOVSX 寄存器,寄存器或存储器地址

功能：传送有符号数到目的寄存器中,并将符号扩展到操作数的所有位。

格式 2：

MOVZX 寄存器,寄存器或存储器地址

功能：传送无符号数到目的寄存器中,并用 0 进行扩展。

3. 双精度移位指令

格式 1：

SHLD 寄存器或存储器地址,寄存器,CL 或立即数

功能：双精度左移指令,第一个操作数左移 N(第三个操作数指出)位,其右边空出位由第二个操作数的左边 N 位填补,CF 保持第一个操作数最后一次的移出位。

格式 2：

SHRD 寄存器或存储器地址,寄存器,CL 或立即数

功能：双精度右移指令,参见 SHLD。

4. 条件设置类指令

这类指令用于测试指定的标志位所处的状态,根据测试结果,将指定的一个 8 位寄存器或内存单元置 1 或 0：结果为真,8 位寄存器或内存单元置 1；结果为假,8 位寄存器或内存单元置 0。

这类指令有 SETA. SETNBE,还有 SETAE. SETNC、SETB. SETNAE. SETC、SETNA.

SETBE、SETE. SETZ、SETG. SETNLE、SETGE. SETNL、SETL. SETNGE、SETLE. SETNG、SETNE. SETNZ、SETNO、SETNS、SETO、SETP. SETPE、SETPO. SETNP、SETS 等。

5. 字节交换指令 BSWAP

本指令用于将 32 位通用寄存器的双字以字节为单位，高位字节与低位字节进行交换。

6. 比较与交换指令 CMPXCHG

本指令将存放在 8 位、16 位、32 位寄存器或存储器中的第一操作数与累加器 AL、AX、EAX 的内容进行比较：相等，则 ZF＝1，并将存放 8 位、16 位、32 位寄存器中的第二操作数送第一操作数的存储单元；不相等，则 ZF＝0，并将第一操作数送相应累加器。

7. 交换与相加指令 XADD

本指令将存放在 8 位、16 位、32 位寄存器或存储器中的第一操作数与存放在 8 位、16 位、32 位寄存器中的第二操作数相加，结果存入到第一操作数，而将第一操作数存入到第二操作数。

8. cache 管理指令

(1) INVD：清洗 cache 指令。
(2) WBINVD：回写和清洗 cache 指令。
(3) INVLPG：作废 TLB 项指令。

3.10.4　增强功能的指令

1. 转换指令

CDQ：将 EAX 中带符号的双字转换为 EDX，EAX 中带符号的 4 字，它把 EAX 中的符号位扩展到 EDX 中的所有位来实现转换。

CWDE：把字转换为双字，即把 AX 中的符号位扩展到 EAX 中的其他位，实现把 AX 中的字转换成 EAX 中的双字的功能。

2. 字符串操作指令

字符串操作指令 CMPSD、INSD、LODSD、MOVSD、OUTSD、SCASD、STOSD。

该组指令是在字符串操作指令 CMPS、INS、LODS、MOVS、OUTS、SCAS、STOS 后加 D，变成了对双字的操作，类似于加 B 对字节操作、加 W 对字操作。寄存器 DS：SI 指向源串，ES:DI 指向目标串。

3. 整数乘指令 IMUL

在 80286 中，IMUL 有两种格式：

```
IMUL  16 位寄存器,立即数
```

IMUL 16 位寄存器,16 位存储器,立即数

在 80386 中,新增一种格式:

```
IMUL  寄存器,寄存器,存储器      ;注意源和目标操作数位数必须相同
```

4. 堆栈操作指令

在 80286 中,PUSHA:把 8 个寄存器值压入堆栈,POPA:从堆栈中弹出数据恢复 8 个寄存器的值。

上述两个指令对 8 个 16 位的寄存器进行堆栈操作。

在 80386 中要对相应的 8 个 32 位寄存器进行堆栈操作,需使用 PUSHAD、POPAD 指令。在 80386 中要对 32 位的标志寄存器 EFLAG 寄存器进行堆栈操作,需使用 PUSHFD、POPFD 指令。

5. 中断返回指令 IRETD

从堆栈中弹出 32 位的指针,使用 IRETD,以从堆栈中弹出一个双字的指针。

上机实验 2:算术运算符的使用

【实验目的】

(1) 深入理解 Debug 命令中的常用指令,理解指令的含义。

(2) 熟悉常用的各种指令,如算术运算,跳转指令等。

【试验内容】

(1) 参照例 3-30 程序,把以下字符串'20073485HuangAn'(前面是学号后面是名字的拼音)存入数据段中并显示出来,然后实现 w←(x−y+59−z)。

数据段定义如下:

```
DATA SEGMENT
      MYNAME DB ' 20073485HUANGAN'
      X   DW 1111H,2222H
      Y   DW 3333H,4444H
      Z   DW 5555H,6666H
      W   DW ?,?
DATA ENDS
```

(2) 参照例 3-32 程序,实现 w←(x * y+560−z)/ v。

数据段定义如下:

```
DATA SEGMENT
  X   DW 1111H
  Y   DW 2222H
  Z   DW 3333H
  V   DW 9999H
 DATA ENDS
```

(3) 符号函数

$$f(x)=\begin{cases}-1, & x<0\\ 0, & x=0\\ 1, & x>0\end{cases}$$

假设 x 为某值且存放在寄存器 AL 中,试编程将求出的函数值 f(x)存放在 AH 中。

习题 3

3-1　指出下列各种操作数的寻址方式。

(1) [BX]　　(2) SI

(3) 435H　　(4) [BP+DI+123]

(5) [23]　　(6) data (data 是一个内存变量名)

(7) [DI+32]　　(8) [BX+SI]

(9) [EAX+90]　　(10) [BP+4]

3-2　已知寄存器 BX、DI 和 BP 的值分别为 1234H、012F0H 和 42H,试分别计算下列各操作数的有效地址。

(1) [BX]　　(2) [DI+123H]

(3) [BP+DI]　　(4) [BX+DI+200H]

(5) [1234H]　　(6) [BX×2+345H]

3-3　假定 DS=1123H, SS=1400H, BX=0200H, BP=1050H, DI=0400H, SI=0500H, LIST 的偏移量为 250H,试确定下面各指令访问内存单元的地址。

(1) MOV　AL,[1234H]　　(2) MOV　AX,[BX]

(3) MOV　[DI],AL　　(4) MOV　[2000H],AL

(5) MOV　AL,[BP+DI]　　(6) MOV　CX,[DI]

(7) MOV　EDX,[BP]　　(8) MOV　LIST[SI],EDX

(9) MOV　CL,LIST[BX+SI]　　(10) MOV　CH,[BX+SI]

(11) MOV　EAX,[BP+200H]　　(12) MOV　AL,[BP+SI+200H]

(13) MOV　AL,[SI-0100H]　　(14) MOV　BX,[BX+4]

3-4　按下列要求编写指令序列。

(1) 清除 DH 中的最低三位而不改变其他位,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、第 5 和第 9 位中是否有一位为 1。

(5) 检查 CX 中的第 1、第 6 和第 11 位中是否同时为 1。

(6) 检查 AX 中的第 0、第 2、第 9 和第 13 位中是否有一位为 0。

(7) 检查 DX 中的第 1、第 4、第 11 和第 14 位中是否同时为 0。

3-5　选择适当的指令实现下列功能。

(1) 右移 DI 三位,并把 0 移入最高位。

(2) 把 AL 左移一位,使 0 移入最低一位。

(3) AL 循环左移三位。

(4) EDX 带进位位循环右移 4 位。

(5) DX 右移 6 位,且移位前后的正负性质不变。

3-6 假设(DS)=2000H,(BX)=0100H,(SI)=0002H,(20100)=12H,(20101)=34H,(20102)=56H,(20103)=78H,(21200)=2AH,(21201)=4CH,(21202)=B7H,(21203)=65H,试写出下列各指令执行完后 AX 寄存器的内容。

(1) MOV AX,1200H　　(2) MOV AX,BX

(3) MOV AX,[1200H]　　(4) MOV AX,[BX]

(5) MOV AX,1100[BX]　　(6) MOV AX,[BX][SI]

(7) MOV AX,1100[BX][SI]

3-7 变量 DATAX 和 DATAY 的定义如下:

```
DATAX DW  0148H
      DW  2316H
DATAY DW  0237H
      DW  4052H
```

请按下列要求写出指令序列:

(1) DATAX 和 DATAY 两个字数据相加,和存放在 DATAY 单元中。

(2) DATAX 和 DATAY 两个双字数据相加,和存放在 DATAY 开始的字单元中。

(3) DATAX 和 DATAY 两个字数据相乘,积存放在 DATAY 开始的单元中。

(4) DATAX 和 DATAY 两个双字数据相加,积存放在 DATAY 开始的单元中。

(5) DATAX 和 DATAY 两个字数据相除。

(6) DATAX 双字和 DATAY 字数据相除。

3-8 假定(DX)=0B9H,(CL)=3,(CF)=1,下列各指令单独执行后 DX 的值是多少?

(1) SHR DX,1　　(2) SAR DX,CL

(3) SHL DX,1　　(4) ROR DX,CL

(5) ROL DX,CL　　(6) RCL DX,CL

3-9 假定 AX 和 BX 中的内容为带符号数,CX 和 DX 中的内容为无符号数,请用比较指令和条件转移指令实现以下功能。

(1) 若 DX 的内容超过 CX 的内容,则转到 J1 处执行。

(2) 若 BX 的内容大于 AX 的内容,则转到 J2 处执行。

(3) 若 CX 的内容等于 0,则转到 J3 处执行。

(4) 若 BX 的内容小于 AX 的内容,则转到 J4 处执行。

(5) 若 DX 的内容低于 CX 的内容,则转到 J5 处执行。