第 3 章　8086 指令系统与汇编语言程序设计

本章学习目标

- 熟练掌握 8086 指令格式与寻址方式。
- 熟练掌握 8086 指令系统。
- 熟练掌握汇编语言编程格式。
- 掌握汇编语言程序设计。

本章首先向读者介绍 8086 指令格式与寻址方式，其次介绍 8086 指令系统，再次介绍汇编语言编程格式，最后介绍汇编语言程序设计。

3.1　8086 指令格式与寻址方式

8086 指令格式与寻址方式包含以下方面：

- 指令格式。
- 寻址方式。

计算机处理的信息存放在存储器和外部设备中。微处理器 CPU 从存储器和外部设备接收信息，对之进行加工处理，然后再将结果送到 CPU 存储器和外部设备，CPU 根据指令的要求执行相应的操作。微处理器采用什么样的寻址方式，直接决定着微型计算机系统的性能。一个高性能的微机系统，应当具有多种灵活的寻址方式，来满足数据存储格式的要求。

指令是控制计算机一步步实现某种操作的命令，指令是根据 CPU 硬件的特点研制出来的，一系列指令构成了计算机程序，用指令编写的程序能够充分开发计算机硬件资源，其目标代码短、运行速度快。指令要解决两个问题：一是要指出进行什么操作——操作码；二是要指出操作数和操作结果放在何处——寻址方式。

3.1.1　指令格式

8086 指令是构成汇编语言指令语句的基本单位，通过汇编程序将其翻译成 CPU 可识别的指令代码（目标代码）以执行某种操作。通常，指令由操作码和操作数组成，其一般格式为：

操作码　操作数，操作数

其中，操作码指明该指令能够完成的功能及操作性质，通常占指令的第一个字节，也称指令助记符。操作数是该指令所处理的数据对象，分为源操作数和目的操作数。不同指令的操作数个数不同，一般包括 1～2 个或隐含操作数。所以指令格式也可表示为：

操作码　操作数，源操作数

【例 3-1】 MOV　　AX, 8000H

这是一条数据传送指令，MOV 是指令中的操作码，指明该指令执行数据传送功能，将操作数 8000H 传送到寄存器 AX 中，该指令执行后 AX 寄存器的值为 8000H。AX 寄存器为目的操作数，8000H 为源操作数。

8086 微处理机的一条指令可以由 1～6 个字节组成（当有段超越时最多为 7 个字节），这要由指令的操作内容和采用的不同寻址方式而定。操作码在前，跟着的是操作数字段。

3.1.2　指令的寻址方式

我们已经了解到指令是由操作码和操作数构成的，CPU 执行指令必须先得到操作码，再获取操作数，操作码是通过 BIU 读取内存 CS 逻辑段中 IP 所指的单元内容，那么 CPU 通过什么途径来得到操作数呢？所谓寻址方式，就是寻找指令中操作数的值或操作数的地址进而获得操作数的方法。我们知道，8086 系统中的操作数可以在指令中直接给出，也可以保存在寄存器或存储器中，还可以从外部设备直接读取待处理数据。指令从不同地方获得的操作数的方式大体分为两类：一类与数据有关的寻址方式，一类与地址有关的寻址方式。其中，与数据有关的寻址方式可分为立即数寻址、寄存器寻址、存储器寻址、I/O 端口寻址 4 种；与地址有关的寻址方式可包括段内直接寻址、段间直接寻址、段内间接寻址、段间间接寻址，如图 3.1 所示。

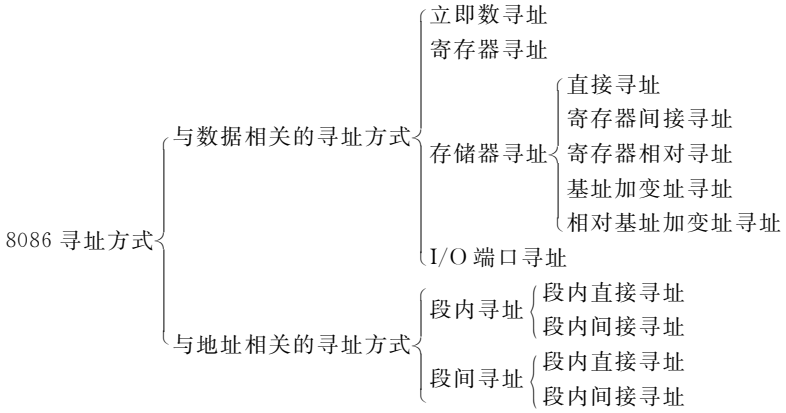

图 3.1　寻址方式示意图

1. 与数据有关的寻址方式

1）立即数寻址

操作数是一个立即数，在指令中紧跟在操作码之后作为指令的一部分存放在代码段中，CPU 在代码段取指令的同时取出操作数，直接在 CPU 内部进行操作，执行速度快。其示意图如图 3.2 所示，格式为：

操作码目的操作数，立即数

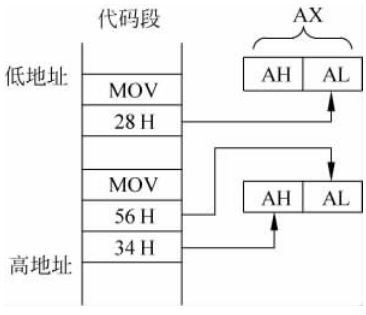

图 3.2　立即数寻址示意图

说明：

（1）立即数可以是 8 位，也可以是 16 位；

（2）立即数存放在存储器的代码段中，低字节在低

地址，高字节在高地址中；

（3）立即数可以是运算数值、地址值或字符名；

（4）立即数寻址主要应用于给寄存器或存储器赋初值。

【例 3-2】
```
MOV  AL, 28H        ;(AL) = 28H
MOV  AX, 3456H      ; (AX) = 3456H,其中(AH) = 34H,(AL) = 56H
```

2）寄存器寻址

寄存器寻址是指指令操作数存放在 CPU 内部寄存器中。在这种寻址方式下，操作数存在 16 位寄存器 AX、BX、CX、DX、SI、BP、DI、SP，也可存在 8 位寄存器 AL、AH、BL、BH、CL、CH、DL、DH 中，在整个指令执行过程中，CPU 并不需要访问存储器，执行速度比访问存储器快得多，使用累加器 AX 速度更快。其格式如下：

操作码　目的操作数寄存器,源操作数寄存器

【例 3-3】
```
MOV  AX,BX;
```

操作数存放在 BX 寄存器中，即(BX)＝1526H；则执行后(AX)＝1562H，而(BX)的值保持不变，寄存器寻址示意图如图 3.3 所示。

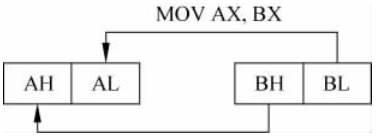

图 3.3　寄存器寻址示意图

3）存储器寻址

存储器寻址是指指令操作数存放在存储器中。在这种寻址方式下，指令首先要知道存储器存放该操作数的地址才能对其进行访问，因而指令中要给出操作数的地址信息，由 CPU 计算出操作数在存储器中的地址。

存储器寻址指令书写中的一个共同点是，通常用“[]”表示。

在 8086 系统中，20 位地址总线寻址内存空间 1MB，其地址范围是 00000H～0FFFFFH。为了便于与 8088 CPU 16 位地址总线寻址能力 64KB(216＝64KB)相兼容，8086 系统内部存储器采用分段管理，将内存空间分为若干逻辑段，每个逻辑段存储容量为 64KB，寻址范围为 0000H～0FFFFH，这些逻辑段通常包含代码段、数据段、堆栈段和附加段。

这些逻辑段可以是连续划分的，也可以是相互重叠，各逻辑段起始单元地址的标识，称为段基址(或段首地址)，一般都存放在同名的段寄存器 CS、DS、SS 和 ES 中。

在讨论存储器的寻址方式时，我们主要关心的是操作数到段基地址的距离，即偏移量，通常，将操作数相对于段基地址的偏移量称为有效地址(Effective Address，EA)或偏移地址。有效地址可以理解为在本段中的存储单元地址，通常在指令中直接给出的 16 位地址信息；而段基地址可以理解为其有效地址为 0000H。其中，有效地址 EA 与基址寄存器、变址寄存器、比例因子、位移量相关，可由下式计算而得：

$$EA＝基址＋(变址×比例因子)＋位移量$$

基址寄存器一般为 BX、SP、BP，基址寄存器一般为 SI、DI，8086 CPU 指令集中比例因子为 1，位移量可以是变量，也可以是具体的数值，后面将详细阐述公式中几个部分不同组合对应的寻址方式。

这样由段基地址和有效地址两个元素可以唯一确定一个操作数在存储器中的地址。通常内存中任意存储单元的地址有两种表达方式：逻辑地址和物理地址。

逻辑地址直接指出操作数在内存中地址的组成因素，由两个 16 位地址组成，通常用“段

基址：有效地址”表示。

物理地址为 20 比特位表示 1MB 连续存储单元的实际地址，范围为 00000H ～0FFFFFH，它由逻辑地址转化而来，根据地址信息的数制表示形式，可以有以下两种计算方法：

物理地址(20)位 = 段地址 × 16 + 有效地址(地址信息由二进制数字表示)
物理地址(5)位 = 段地址 × 10H + 有效地址(地址信息由十六进制数字表示)

例如，数据段的段基址为 0010 0000 0000 0000B(2000H)，操作数存储单元的有效地址为(048FH)0000 0100 1000 1111B，则该存储单元的逻辑地址为 2000H：048FH，物理地址为 2048FH。

8086 系统允许通过多种寻址方式求得操作数有效地址，相应的有存储器直接寻址、寄存器间接寻址、寄存器相对寻址、基址加变址寻址、相对基址加变址寻址 5 种。

(1) 直接寻址。在直接寻址方式中，存储器单元的有效地址由指令直接给出，程序直接通过紧跟在指令操作码后的内存有效地址来访问内存中的操作数。直接寻址方式默认操作数在数据段中，亦即认为段地址为 DS 寄存器中的值。直接寻址指令可访问一个内存单元数据，也可以访问两个内存单元数据。其格式如下：

操作码　目的操作数寄存器,[立即数(有效地址)]
操作码　[立即数(有效地址)],源操作数寄存器

【例 3-4】　MOV　AX,[1234H]

其中，1234H 立即数由中括号括起表示寻址内存，且为内存中数据段的有效地址。假设(DS) = 2000H，则操作数所在的段地址为 2000H，有效地址为 1234H，其逻辑地址表示为 2000H：1234H，其物理地址为：2000H × 10H + 1234H = 21234H。而其执行过程是将内存中物理地址为 21234H 单元的字节数据取到累加器 AL 中，将 21235H 地址单元内容送到 AH 中，即取出一个字数据，低地址单元数据送低 8 位寄存器 AL 中，高地址单元数据送高 8 位寄存器 AH 中。寻址过程如图 3.4 所示。

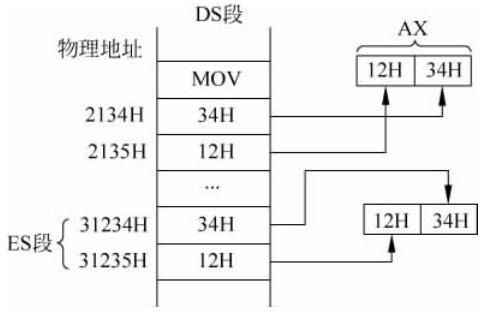

图 3.4　直接寻址过程示意图

当然，也可以人为地指定操作数所在的段(如 CS、SS 或 ES 等)，称为段超越。若操作数在附加段 ES 中，则例 3-4 可改写如下。

【例 3-5】　MOV　AX,ES:[1234H]
　　ES:　MOV　AX,[1234H]

假设(ES) = 3000H，则操作数所在的段地址为 3000H，偏移量为 1234H，其逻辑地址表示为 3000H：1234H，其物理地址为：3000H × 10H + 1234H = 31234H，指令寻址 31234H 和 31235H 两个地址单元数据。

(2) 寄存器间接寻址。寄存器间接寻址是指操作数存放在寄存器中，而其有效地址存放在基址存储器或变址寄存器中。对于 8086 CPU，基址寄存器只能是 BX 或 BP，变址寄存

器只能是 SI 或 DI。

操作码　的操作数寄存器,[基址寄存器或变址寄存器]
操作码　[基址寄存器或变址寄存器],源操作数寄存器

需要注意的是,对于 8086 来说,在不指定数据所在段的情况下,间址寄存器采用 BP; SP 时默认的段是堆栈段(段寄存器为 SS),而采用其他基址寄存器 BX 或变址寄存器 SI、DI 作为间址进行寻址时,默认的段为数据段 DS(串操作指令例外,还有附加段 ES),间址所用寄存器与默认段之间的对应关系如图 3.5 所示。另外,我们仍可以采用段超越方法来指定段寄存器。

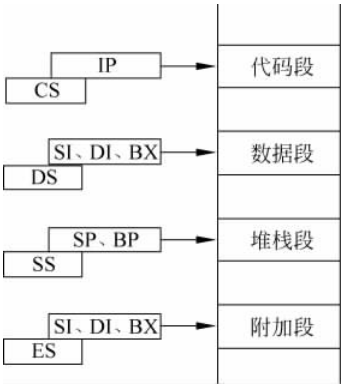

图 3.5　寄存器与默认段对应关系

【例 3-6】　MOV　AX,[BX]

同直接寻址一样,"[]"表示寻址内存,且中括号内的寄存器值为内存中的有效地址。假设(DS)=2000H,(BX)=1234H,则指令把物理地址为 2000H×10H+1234H=21234H 单元开始的一个字传送到 AX 中。若(21234H)=20H,(21235H)=1AH,则(AL)=20H,(AH)=1AH,即指令执行后,(AX)=1A20H。

(3) 寄存器相对寻址。在这种寻址方式下,操作数的有效地址为基址或变址寄存器的内容与指令中指定的一个 8 位或 16 位位移量之和。位移量可以是变量,也可以是具体的数值。除非人为指定段超越,如果采用寄存器 BX、DI、SI 进行寻址,则默认操作数存放在数据段中,如果采用寄存器 BP 进行寻址,则认为操作数在堆栈段中。其格式如下:

操作码　目的操作数寄存器,位移量[基址寄存器或变址寄存器]
操作码　[基址寄存器或变址寄存器 + 位移量],源操作数寄存器

【例 3-7】　MOV　AX,[BX + 1000H]

假设(DS)=2000H,(BX)=1234H,则指令执行将内存中的一个字或字节传送到 AX 或 AH 中,指令的寻址过程如图 3.6 所示。

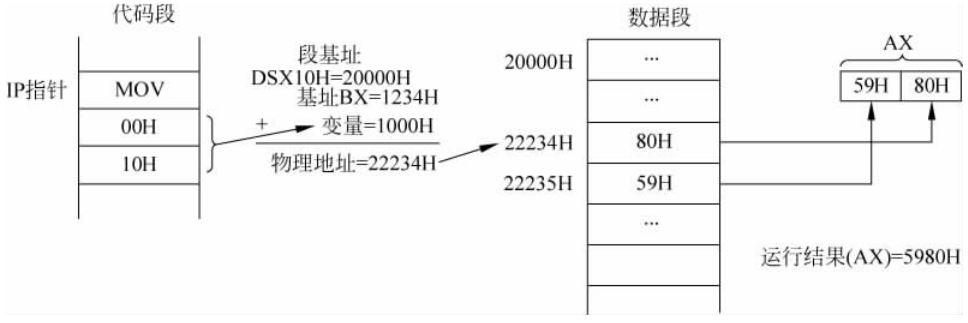

图 3.6　寄存器相对寻址过程

说明:例 3-7 还有几种等同的写法,例如:MOV AX,1000H[BX]、MOV AX,[BX]1000H。

(4) 基址变址寻址。基址变址寻址是指指令中带有基址寄存器和变址寄存器,指令操作数的有效地址为基址寄存器与变址寄存器中数据之和。其格式如下:

操作码　的操作数寄存器,[基址寄存器 + 变址寄存器]
操作码　的操作数寄存器,[基址寄存器][变址寄存器]
操作码　[基址寄存器 + 变址寄存器],源操作数寄存器
操作码　[基址寄存器][变址寄存器],源操作数寄存器

【例 3-8】 MOV　AH,[BX][SI]

设(DS)＝2000H,(BX)＝1234H,(SI)＝1000H,则操作数的有效地址为基址寄存器BX内容与变址寄存器SI内容之和,即1234H＋1000H＝2234H,操作数物理地址为22234H。基址变址寻址过程如图3.7所示。

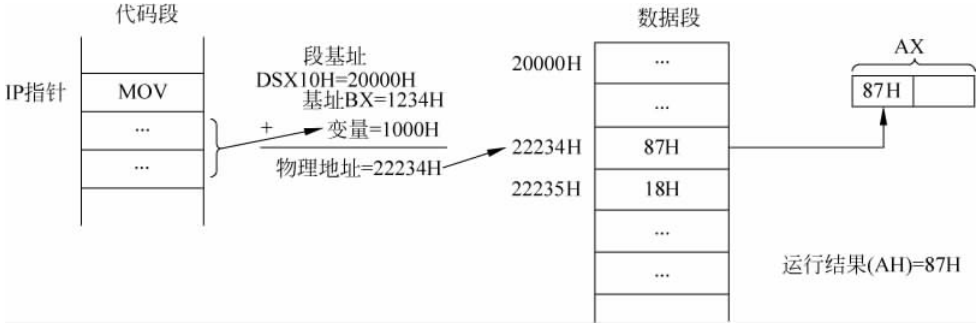

图 3.7　基址变址寻址过程

说明:例3-8的指令还可以写为MOV AH,[BX＋SI]。

注意:① 不能同时为基址寻址或同时为变址寻址,指令中SI和DI寄存器不能同时出现,BX和BP寄存器也不能同时出现;

② 若基址采用BX寄存器时,则操作数默认存放在数据段存储器中;若基址采用BP寄存器时,则操作数默认存放在堆栈段存储器中;当指令中指定段超越时,操作数存放于指令所指定的段中。

(5) 相对基址变址寻址。与存储器基址变址寻址方式相比,相对基址变址寻址操作数的有效地址由基址、变址、位移量三者之和构成,多了一个8位或16位的位移量。其格式如下:

操作码　的操作数寄存器,[基址寄存器 + 变址寄存器 + 位移量]
操作码　的操作数寄存器,位移量[基址寄存器][变址寄存器]
操作码　[基址寄存器 + 变址寄存器 + 位移量],源操作数寄存器
操作码　位移量[基址寄存器][变址寄存器],源操作数寄存器

【例 3-9】 MOV　AL,[BP + SI + 100H]或 MOV　AL,100H[BP][SI]

假设(SS)＝4000H,(BP)＝1234H,(SI)＝2540H,指令的寻址过程如图3.8所示。

说明:该指令还可以写为:

```
MOV  AL,100H[BP][SI]
MOV  AL,[BP + 100H][SI]
MOV  AL,[BP]100H[SI]
MOV  AL,[BP][SI]DAO(DAO = 100H)
MOV  AL,[SI + 100H][BP]
```

需要注意的是,BP为基址寄存器时,操作数默认为在堆栈段(SS)存储器中。

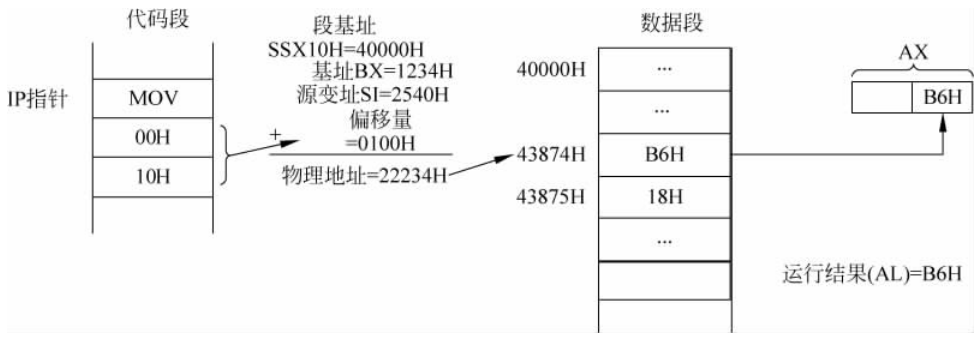

图 3.8　基址变址相对寻址过程

2. 与地址数据有关的寻址方式

计算机执行程序时，有时候需要改变程序的执行顺序，跳转到一个新的地址，BIU 再顺序取指令，指令指针 IP 自动加 1，指向代码段中下一条要执行的指令，从而在新的跳转地址上顺序执行程序，直到再来一个跳转。与地址有关的寻址方式用来确定程序跳转时下一条要执行的指令地址。与地址有关的寻址方式包括段内寻址和段间寻址。段内寻址是指在当前代码段内进行程序跳转执行的转移地址寻址，分为段内直接寻址和段内间接寻址。段间寻址是指跳转执行的程序在其他代码段，这种不同代码段之间的转移寻址即为段间寻址，可分为段间直接寻址和段间间接寻址。

1) I/O 端口寻址

CPU 与外部设备进行通信都要通过 I/O 端口，并通过专用 I/O 指令完成。根据微处理器分配给外部设备端口号不同，I/O 端口寻址方式不同，一般有两种：直接端口寻址方式和间接端口寻址方式。I/O 直接寻址方式是指端口号在指令中直接指出的寻址方式，允许寻址范围只能是 256 个端口；I/O 直间接寻址方式是指端口号通过寄存器 DX 进行传送的寻址方式，允许寻址范围为 64K(0～65 535)个端口。

微处理器分配给外部设备最多有 64K 个端口，其中前 256 个端口(端口号为 0～0FFH)为 8 位固定端口，可采用 I/O 直接寻址方式或间接寻址方式；其余端口(端口号 100H～4FFH)为 16 位可变端口，必须采用 I/O 间接寻址方式。CPU 与 I/O 端口传送信息只能通过累加器 AX 或 AL 完成。

【例 3-10】 直接端口寻址方式。

```
IN    AL,30H              ; 从 30H 号的端口读取一个 8 位数据
IN    AX,30H              ; AL←30H 端口号中的数据,AH←31H 端口中的数据
OUT   21H,320H
OUT   DX,AL
```

说明：间接寻址端口地址必须先送到 DX 寄存器，然后再用累加器 AX/AL 进行传送。

2) 段内寻址

(1) 段内直接寻址。

转向的有效地址为立即数或符号地址且确定转移地址时，不改变 CS 寄存器值，只在当前指令所在 IP 处加上一个位移量，转移地址范围是当前指令所在处前 128 到后 127(－128～＋127)个单元之间，称为段内直接转移。段内直接转移适用于条件转移，又可分为段内近转

移和段内短转移。

段内短转移是指当前位移量为 8 位。若转向地址为符号地址，则要在转移到的符号地址前加属性操作符 SHORT PTR，其指令格式为：

```
JMP  0A0H
JMP  SHORT PTR TRANS1     ;TRANS1 是要转移到的符号地址
```

段内近转移是指当前前位移量为 8 位。若转向地址为符号地址，则要在转移到的符号地址前加属性操作符 SHORT PTR，其指令格式位为：

```
JMP  1A2BH
JMP  NEAR PTRT  RANS1     ;TRANS1 是要转移到的符号地址
```

(2) 段内间接寻址。

转向的有效地址为一个寄存器或存储单元的内容且确定转移地址时，不改变 CS 寄存器值，用转向的有效地址来取代 IP，称为段内间接转移。转向的有效地址可以由除立即数之外的任何寻址方式得到。适用于无条件转移。其指令格式为：

```
JMP  BX
JMP  WORD PTR[BX + 1000]     ; 只将存储器中字单元内容送 IP
```

3) 段间寻址

(1) 段间直接寻址，当指令中直接给出转向地址的段地址和有效地址时，可以用段地址取代 CS，用有效地址取代 IP，称为段间直接寻址。其指令格式为：

```
JMP  4000H: 25AAH
```

(2) 段间间接寻址，当指令中用存储器两个字单元内容间接给出转向地址的段地址和有效地址时，可以将存储器中高地址内容取代 CS，用低地址内容取代 IP，称为段间间接寻址。存储器有效地址可以由除立即数、寄存器寻址之外的任何寻址方式得到。段间间接寻址只适用于无条件转移指令，其指令格式为：

```
JMP  WORD PTR[BX + 1000H]; 将存储器中有效地址(BX) + 1000H 和(BX) + 1001H 字节单元内容送 IP,
(BX) + 1002H 和(BX) + 1003H 字节单元内容送 CS
```

3.2　8086 指令系统

8086 指令系统包含以下方面：

- 概述。
- 执行部件(EU)。

3.2.1　概述

指令是控制计算机一步步实现某种操作的命令，是根据 CPU 硬件的特点制定的。指令对于程序而言，如同人们说话与写作时的文字与必须遵循的语法。指令要解决两个问题：第一，要指出进行什么样的操作——操作码；第二，要指出操作数和操作结果放在何处——

寻址方式。8086 的指令按功能大致分为以下 6 种：

(1) 数据传送类指令；

(2) 算术运算类指令；

(3) 逻辑操作类指令；

(4) 字符串操作类指令；

(5) 控制转移类指令；

(6) 处理机控制类指令。

为了方便表示，本书约定一些符号，如表 3.1 所示。

表 3.1　符号约定及含义

符号	含　义	符号	含　义
i8	一个 8 位的立即数	m8	一个 8 位存储器操作数
i16	一个 16 位的立即数	m16	一个 16 位存储器操作数
imm	一个 8 位或 16 位的立即数	mem	一个 8 位或 16 位存储器操作数
r8	一个 8 位通用寄存器	dest	目的操作数
r16	一个 16 位通用寄存器	src	源操作数
reg	一个 8 位或 16 位通用寄存器	port	I/O 端口号
seg	一个段寄存器		

3.2.2　数据传送指令

数据传送指令是将数据或地址传送到寄存器或存储单元中。它分为以下 4 类：

(1) 通用数据传送指令；

(2) 地址传送指令；

(3) 标志传送指令；

(4) 输入/输出指令。

在数据传送指令中，除第(3)类标志传送指令会对标志位产生影响外，其余的指令均不影响标志位。所以在下面的阐述中，对于(1)、(2)、(4)类指令，将不再说明它们对标志状态位的影响。

1. 通用数据传送指令

通用数据传送指令包括 MOV、PUSH、POP、XCHG、XLAT。

1) 数据传送指令 MOV

格式：

```
MOV  dest,src
```

功能：将源操作数的内容传送到目的地址中。

其中，src 可以为 mem、seg、imm；dest 可以为 mem、seg。

MOV 可以实现一个字节或一个字的传送，但要求 dest 与 src 相同类型，即长度相等。

需要注意的是，

(1) 代码段寄存器 CS 不能作为目的操作数被 MOV 指令修改，但可以将 CS 作为源操作数，将其内容传送给其他寄存器或存储器。

(2) 源操作数与目标操作数不能同时为内存操作数。

【例 3-11】 以下指令均为合法的传送指令,括号中为目的操作数和源操作数的寻址方式。

```
MOV  AL,5              ;(寄存器,立即数)
MOV  AX,BX             ;(寄存器,寄存器)
MOV  DS,AX             ;(段寄存器,寄存器)
MOV  ES,DS             ;(段寄存器,段寄存器)
MOV  ES: VAR,12        ;(存储器,立即数)
MOV  WORD PTR[BX],12   ;(存储器,立即数)
```

说明:(1) 有段超越,VAR 为符号地址;

(2) WORD PTR 指明存储器操作数的属性是字属性。

2) 堆栈操作指令 PUSH、POP

堆栈是计算机的一种数据结构,数据的存取原则是“先进后出,后进先出”。

(1) 进栈指令 PUSH。

格式:

```
PUSH  src
```

src 可以为 r16、seg、m16。

功能:SP←SP-2

(SP+1,SP)←(src)

该指令先将堆栈指针减 2,然后将源操作数送入栈顶。

(2) 出栈指令 POP。

格式:

```
POP dest
```

dest 可以为 r16、seg、m16。

功能:(dest)←(SP+1,SP)

SP←SP+2

该指令实现栈顶字数据送目标操作数,同时堆栈指针加 2。

【例 3-12】 实现 DS 和 ES 的内容交换。

```
PUSH    DS
PUSH    ES
POP     DS
POP     ES
```

3) 数据交换指令 XCHG

格式:

```
XCHG  dest,src
```

dest 可以为 reg、mem; src 可以为 reg、mem。

功能:源操作数与目的操作数互换。

【例 3-13】 XCHG AX,BX

假设执行命令前(AX)=1234H,(BX)=5678H,则该指令执行后,(AX)=5678H,(BX)=1234H。

说明:

指令中源操作数与目的操作数类型必须一致,或同为 8 位或同为 16 位。

XCHG 允许两个通用寄存器之间交换数据,例如,XCHG AL,BH。

XCHG 允许通用寄存器与存储器操作数之间交换数据,例如,XCHG AH,[BP]。

XCHG 不允许两个存储器之间交换数据。

XCHG 不允许段寄存器和立即数为操作数。

4) 换码指令 XLAT

格式:

```
XLAT 或 XLAT  变量名或表格首址
```

源操作数与目的操作数均隐含。

功能: 把数据段 DS 中偏移地址为 BX+AL 的内存单元的内容送到 AL 中,即 AL←(BX+AL)。

【例 3-14】 将表格中位移量为 4 的代码取到 AL 中,如图 3.9 所示。

```
LEA  BX,table
MOV  AL,4
XLAT table
```

指令执行后,(AL)=05。

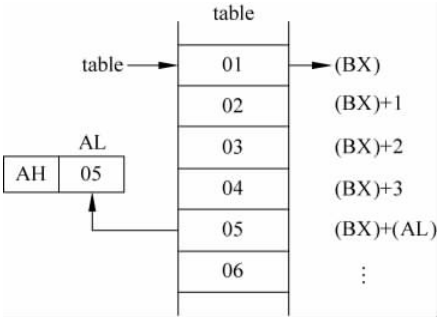

图 3.9 XLAT 指令操作示意图

2. 地址传送指令 LEA、LDS、LES

这是一类专门用于传送地址码的指令,源操作数必须是存储器地址,目标操作数必须是 16 位通用寄存器操作数,共包括以下 3 种指令。

1) 取有效地址指令 LEA

格式:

```
LEA  r16,mem
```

功能: 取内存单元 mem 的有效地址,送到 16 位寄存器 r16 中,即 r16←EA(mem)。

【例 3-15】 设 DS = 2100H,BX = 100H,SI = 10H,(DS: 110H) = 1234H

指令 LEA BX,[BX+SI]执行后,BX=BX+SI=110H。

2) 地址指针装入 DS 指令 LDS

格式:

```
LDS  r16,m32
```

功能: 把内存中的 32 位源操作数中的低 16 位送到指定寄存器 r16 中,高 16 位送到段寄存器 DS 中。即 r16←m32 低 16 位;DS←m32 高 16 位。

【例 3-16】 LDSSI,[1234H]

假设 DS：1234H～1237H 单元中存放着 1 个物理地址，1234H 和 1235H 单元存放的有效地址，送 16 位 SI 寄存器；1236H 和 1237H 单元存放的短地址，送 DS。

3）地址指针装入 ES 指令 LES

把上述指令中的 DS 换成 ES，即成为 LES 指令。

3. 标志传送指令 LAHF、SAHF、PUSHF、POPF

1）取标志指令 LAHF

格式：

```
LAHF
```

该指令源操作数隐含为标志寄存器低 8 位，目标操作数隐含为 AH。

功能：把 16 位的标志寄存器低 8 位送至寄存器 AH 中，即 AH←(FLAG)0～7。

LAHF 执行示意图如图 3.10 所示。

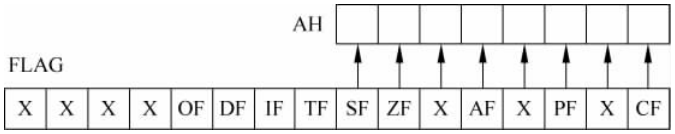

图 3.10　LAHF 执行示意图

2）置标志指令 SAHF

格式：

```
SAHF
```

源操作数隐含为 AH，目标操作数隐含为标志寄存器。

功能：把寄存器 AH 中的内容送至 16 位的标志寄存器低 8 位，即(FLAG)0～7 ←AH。

SAHF 执行示意图如图 3.11 所示。

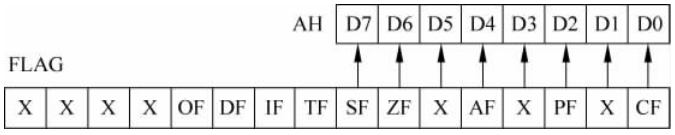

图 3.11　SAHF 执行示意图

3）标志入栈指令 PUSHF

格式：

```
PUSHF
```

源操作数隐含为标志寄存器，目标操作数隐含为堆栈区。

功能：标志寄存器入栈。

(1) SP← SP－2

(2) (SP＋1,SP)← FLAG

4）标志弹出指令 POPF

格式：

```
POPF
```

源操作数隐含为堆栈区，目标操作数隐含为标志寄存器。

功能：数据出栈到标志寄存器。

（1）FLAG←(SP+1,SP)；

（2）SP← SP+2；

此操作是 PUSHF 的逆操作。

【例 3-17】 把标志寄存器 TF 位清零，其他位不变。

```
PUSHF                          ; 标志寄存器出栈
POP    AX                      ; 取标志寄存器内容
AND    AX,0FEFFH               ; TF 清零,其他位不变
PUSH   AX                      ; 新值入栈
POPF                           ; 送入标志寄存器
```

4. I/O 指令 IN、OUT

I/O 指令用于 CPU 与输入/输出设备端口之间的数据传送。IN 用于 CPU 从外设读数据，OUT 用于 CPU 对外设写数据。对外设的寻址方式有两种：直接寻址和间接寻址。

1）直接寻址

指令中直接给出端口地址，相应的指令格式如下：

输入格式：

```
IN  AL/AX,port
```

输出格式：

```
OUT  port,AL/AX
```

说明：

（1）port 为端口地址，取值范围为：0～255(0FFH)；

（2）操作数类型为字节时，选用 AL 寄存器，类型为字时，选用 AX 寄存器。

【例 3-18】 从端口 60H 读字节，把它低 4 位清零后，再从 60H 送出。

```
IN    AL,60H                   ; 从 60H 读 8 位数
AND   AL,0F0H                  ; 低 4 位清零
OUT   60H,AL                   ; 从 60H 送出
```

2）间接寻址

端口地址存于 DX 寄存器中。相应的指令格式如下：

输入格式：

```
IN  AL/AX,DX
```

输出格式：

```
OUT  DX,AL/AX
```

说明：(1) 端口地址，取值范围为：0～65 535(0FFFFH)。

(2) 操作数类型为字节时，选用 AL 寄存器，类型为字时，选用 AX 寄存器。

【例 3-19】 使用间接寻址方式实现：从端口 60H 读字节，把它的低 4 位清零后，再从 61H 送出。

```
MOV     DX,60H              ; 源端口地址送 DX 寄存器中
IN      AL,DX               ; 从 60H 读 8 位数
AND     AL,0F0H             ; 低 4 位清零
INC     DX                  ; DX 寄存器内容为目标端口地址
OUT     DX,AL               ; 从 61H 送出数据
```

注意：

(1) 对外设端口操作时，当端口地址在 0～255(0FFH)范围内，寻址方式可选用直接寻址，也可选用间接寻址；当端口地址大于 255 时，只能选用间接寻址，并且地址寄存器智能用 DX。

(2) 数据寄存器只能用 AL(字节操作)或 AX(字操作)。

3.2.3　算术运算指令

算术运算指令是反映 CPU 运算能力的一组指令，也是编程时经常使用的一组指令。算术运算指令是在 ALU 中完成的，执行结果影响标志寄存器。8086 指令系统提供了加、减、乘、除 4 类基本算术指令以及类型转换指令。这些运算实现字节、字或多字节数的运算、支持二进制 BCD 码表示的十进制的运算、支持带符号数和无符号数两种类型数据的运算。

1. 二进制加法指令

1) 基本加法指令 ADD

ADD 指令是不带进位的加法指令，也是 8086 系统最基本的加法运算指令，该指令用来执行 2 个字或字节操作数的相加操作，并将结果存放在目的操作数中。

格式：

```
ADD  dest,src
```

dest 可以为 reg、mem；src 可以为 reg、mem、imm。

【例 3-20】

```
ADD  AH,BL              ; 通用寄存器之间相加
ADD  AX,2000H           ; 通用寄存器与立即数相加
ADD  [BX],BX            ; 内存数与通用寄存器之间相加
ADD  [BX+SI],200        ; 内存数与立即数相加
ADD  AX,[DI]            ; 通用寄存器与内存数之间相加
ADD  AL,50H             ; AL 和 50H 相加，结果放在 AL 中
ADD  DI,SI              ; DI 和 SI 的内容相加，结果放在 DI 中
```

2) 带进位的加法指令 ADC

ADC 的指令格式与 ADD 相同，指令的功能也很相似，只有一点区别就是 ADC 指令执行加法运算时，将当前的进位标志 CF 的值也加在和中。这条指令一般用于多字节的加法运算中，当低字节相加结果有进位，即 CF＝1 时，高字节除了进行两操作数相加之外，还需要加上进位 1。

【例 3-21】

```
MOV  AX,[1000H]          ; 将第一个数的低 16 位取到 AX
ADD  AX,[2000H]          ; 第一个数和第二个数的低 16 位相加
MOV  [SI],AX             ; 低 16 位相加结果送到 DS: SI 和 DS: SI + 1 单元
MOV  AX,[2002H]          ; 取第一个数的高 16 位送到 AX 中
ADC  AX,1002H            ; 两个数的高 16 位连同进位位相加
MOV  [SI + 2],AX         ; 高 16 位相加的结果送到 DS: SI + 2 和 DS: SI + 3 单元
```

3）加 1 指令 INC

INC 加 1 指令也称增量指令，该指令只有一个操作数，其格式为：

```
INC  src
```

该指令在执行时，将操作数的内容加 1，执行结果送操作数，操作数可以为通用寄存器和存储器。INC 指令一般用在循环程序中修改指针和循环次数。INC 指令影响标志位 AF、OF、PF、SF 和 ZF，但不影响进位标志 CF。

【例 3-22】

```
INC  AL                  ; 将 AL 中的内容加 1
INC  CX                  ; 将 CX 中的内容加 1
INC  BYTE PTR[BX]        ; 将 BX 所指的字节存储单元的内容加 1
```

2. 二进制减法指令

1）基本减法指令 SUB

SUB 同 ADD 指令相对应，为不带借位的基本减法指令，实现 2 个字节或 2 个字的目的操作数与源操作数相减，结果存放在目的操作数中。

格式：

```
SUB  dest,src
```

dest 可以为 reg、mem；src 可以为 reg、mem、imm。

当目的操作数是内存操作数时，源操作数只能是通用寄存器或立即数。SUB 指令执行结果影响标志位 OF、SF、ZF、AF、PF 和 CF。

【例 3-23】

```
SUB  AH,BL               ; 通用寄存器之间相减
SUB  AX,2000H            ; 通用寄存器与立即数之间相减
SUB  [BX],BX             ; 内存数与通用寄存器之间相减
SUB  [BX + SI],200       ; 内存数与立即数相减
SUB  AX,[DI]             ; 通用寄存器与内存数之间相减
SUB  AL,20               ; AL 中的数减去 20，结果在 AL 中
SUB  SI,5010H            ; SI 中的数减去 5010H，结果存在 SI 中
SUB  WORDPTR [DI],1000H  ; DI 和 DI + 1 所指的两个单元中的数减去 1000H，
                         ; 结果在 DI 和 DI + 1 所指的单元中
```

2）带借位的减法指令 SBB

带借位的减法指令 SBB 在形式上和功能上都和 SUB 指令类似，只是 SBB 在执行减法运算时，是用被减数减去减数，再减去低位字节产生的借位。和带进位的加法指令 ADC 类

似，SBB主要用在多字节减法运算中。SBB同SUB指令一样，执行结果影响标志位OF、SF、AF、PF和CF。

格式：

```
SBB dest,src
```

【例 3-24】

```
SBBAX,2030H        ; 将AX的内容减去立即数2030H,并减去进位标志CF的值
SBB       WORD PTR [DI+2],1000H ; 将DI+2和DI+3所指的两个单元的内容减去立
                                ; 即数1000H,再减去CF的值,结果存在DI+2
                                ; 和DI+3所指的单元中
```

3）减1指令DEC

DEC减1指令也称减量指令，其执行时将实现目的操作数的值减1，功能与INC指令功能相反。减1指令只有一位操作数。DEC指令影响标志位AF、OF、PF、SF和ZF，但不影响进位标志位CF。

格式：

```
DEC src
```

【例 3-25】

```
DEC AX                  ; 将AX的内容减1,再送回AX中
DEC DL                  ; 将DL的内容减1,结果送回DL中
DECBYTE PTR[SI]         ; 将DS: SI所指的单元的内容减1,结果送回次单元
DECBYTE PTR[DI+2]       ; 将DI+2所指的单元的内容减1,结果送回此单元中
```

4）求补指令NEG

NEG求补指令能够对指令中给出的目的操作数的值取补码，再将结果送回目的操作数。NEG指令影响标志位AF、CF、OF、PF、SF和ZF。

格式：

```
NEG dest
```

【例 3-26】

```
NEG AX;             将计算0减去AX中的内容得到该数据的补码,送AX寄存器
NEG AL
```

注意：

（1）当(AX)＝(AL)＝0时，指令执行后，目的操作数的值仍为0。进位标志CF＝0，溢出标志OF＝0，此时没有借位没有溢出。

（2）如果操作数(AX)＝8000H或(AL)＝80H，NEG指令执行后，目的操作数的回送值不变，即执行后(AX)＝8000H或(AL)＝80H，而进位标志CF＝1，溢出标志OF＝1。原因是若把AX看成带符号数，8000H相当于十进制的－32 768，0－(AX)＝32 768＞32 767，所以有溢出，OF＝1。同理，80H等于－128，当(AL)为无符号数时，0－(AL)＝128。

（3）通常指令执行会使CF为1，只有当操作数为0时，才使CF为0，这是因为NEG指令执行时，是用0减去某个操作数，除非给定的操作数为0，否则均会产生借位。

5）比较指令 CMP

CMP 比较指令执行目的操作数减去源操作数，但不回送相减的结果，只是影响标志位，CPU 根据标志位判断比较结果，指令执行后，被比较的两个操作数的值保持不变。CMP 指令常与条件转移指令配合使用，完成各种条件判断和程序转移。

格式：

```
CMP dest,src
```

dest 可以为 reg、mem、src 可以为 reg、mem 和 imm。

【例 3-27】

```
CMP AX,1000H            ; 将 AX 中的数和 1000H 比较，结果影响标志位
CMP AL,80H              ; 将 AL 中的数和 80H 比较，结果影响标志位
CMP AX,[BX+100]         ; 将累加器和两个存储单元的数相比，单元地址由
                        ; BX+100 和 BX+101 指出
```

如何根据标志位来判断比较结果呢？对于无符号数比较，可根据 CF 判断大小；对于有符号的比较，要根据 OF 和 SF 两者的关系来判断结果。

ZF＝1，被比较两数相等；

CF＝0，两个被比较的无符号数，被减数大，减数小；

CF＝1，两个被比较的无符号数，被减数小，减数大；

SF＝0，两个被比较的同符号数，被减数大，减数小；

SF＝1，两个被比较的同符号数，被减数小，减数大；

OF＝1 且 SF＝1，两个有符号数比较，被减数大；

OF＝1 且 SF＝0，两个有符号数比较，被减数小。

3. 乘法指令

8086 指令系统对于乘法运算提供了无符号乘法和有符号乘法两种。执行乘法运算时，如果两个 8 位数相乘，会得到一个 16 位的乘积。如果两个 16 位数相乘，会得到一个 32 位乘积。8086 乘法指令中只有一个操作数，另一个操作数为隐含操作数 AL 或 AX。对于 32 位的乘积，将 DX 寄存器看成是 AX 寄存器的扩展，这样，当乘积为 16 位时，结果就在 AX 中，乘积为 32 位时，结果在 DX 和 AX 两个寄存器中，DX 中为乘积的高 16 位，AX 中为乘积的低 16 位。乘法指令影响标志位 OF 和 CF。

1）无符号数的乘法指令 MUL

MUL 指令实现两个 8 位或 16 位无符号数相乘，结果送 AX 或 AX 和 DX。

格式：

```
MUL src
```

src 可以为 reg8、reg16 或 mem。

【例 3-28】

```
MUL BL                  ; AL 中的 8 位数和 BL 中的 8 位数相乘，结果存放在 AX 中
MUL BX                  ; AX 中的 16 位数和 BX 中的 16 位数相乘，结果存放在 DX 和 AX 中
MUL BYTE PTR[BX]        ; AL 中的 8 位数和 BX 所指的存储单元中的 8 位数相乘，结果存放在 AX 中
MUL WORD PTR[DI]        ; AX 中的 16 位数和 DI，DI+1 所指存储单元中的 16 位数相乘，
                        ; 结果存放在 DX 和 AX 中
```

2）带符号数的乘法指令 IMUL

IMUL 带符号数的乘法指令在功能和形式上与 MUL 很类似，只是要求两个乘数必须均为带符号数。

格式：

```
IMUL src
```

src 可以为 reg8、reg16 或 mem。

【例 3-29】

```
IMUL BX             ; DX 和 BX 中的两个 16 位带符号数相乘,结果存放在 DX 和 AX 中
IMUL BYTE PTR[BX]   ; AL 中的 8 位带符号数和 BX 所指存储单元中的 8 位带符号数相乘,
                    ; 结果存放在 AX 中
IMUL BYTE PTR[BX]   ; AL 中 8 位带符号数和 BX 所指单元的 8 位带符号数相乘,结果存放在 AX 中
IMUL WORD PTR[DI]   ; AX 中的 16 位带符号数和 DI,DI + 1 所指的单元 16 位带符号相乘,
                    ; 结果存放在 DX 和 AX 中
```

注意：如果操作数是内存，则必须明确内存单元地的类型；否则，对于 MUL [BX][SI] 无法确定是进行字节乘法还是字乘法，正确的指令为 MUL BYTE PTR[BX][SI]或者 MUL WORD PTR[BX][SI]。

4. 除法指令

8086 执行除法指令时，规定被除数必须是除数的 2 倍字长，即被除数为 16 位时，除数为 8 位，被除数为 32 位时，除数为 16 位。被除数在指令中是隐含地放在 AX、DX 中，当被除数为 32 位时，则 DX 中放高 16 位，AX 中放低 16 位。指令格式中给出操作数是除数，计算机根据给定的除数形式来确定被除数在 AX 中或 DX、AX 中。

1）无符号数的除法指令 DIV

DIV 指令用来实现 16 位或 32 位无符号被除数与 8 位或 16 位无符号除数相除运算。

格式：

```
DIV src
```

src 可以为 reg8、reg16 或 mem。

【例 3-30】

```
DIV CL              ; 若(CL) = 02H,(AX) = 0129H,则指令执行后(AL) = 64H,(AH) = 01H
DIV WORD PTR[DI]    ; DX 和 AX 中的 32 位数除以 DI,DI + 1 所指的两单元中的 16 位,商在 AX 中,
                    ; 余数在 DX 中
```

2）有符号数的除法指令 IDIV

IDIV 在功能及形式上和 DIV 类似，主要区别在于 IDIV 指令在执行时，将被除数和除数都看成带符号数，因此，具体执行过程不同于 DIV 的执行过程。

格式：

```
DIV src
```

src 可以为 reg8、reg16 或 mem。

【例 3-31】

```
IDIV BX          ; 将 DX 和 AX 中的 32 位数除以 BX 中的 16 位
                 ; 数指令执行后,商在 AX 中,余数在 DX 中
```

注意：

(1) 除法指令中,源操作数不能为立即数。

(2) 除法运算后,标志位 AF、CF、OF、PF、SF 和 ZF 都是不确定的,或 0 或 1 且都没有意义。

(3) 除法运算中,8086 指令系统中规定余数的符号和被除数的符号相同。

(4) 除法运算中,凡字节运算的商超过－128～＋127 或 0～255,或者字运算的商超过－32 768～＋32 767 或 65 535 时均为溢出,0 做除数也为溢出。除法出现溢出时,将立即产生 0 号中断,程序停止执行。

(5) 除法运算时,要求被除数为除数的 2 倍,如果不是,则要把被除数变成除数的双倍长度即所说的对 AX、AL 进行扩展。对于无符号数相除来说,只将 AH 和 DX 寄存器清零来实现寄存器扩展；对于有符号数相除来说,对于 AH 和 DX 的扩展,8086 指令系统提供了专门用于带符号数扩展的指令 CBW 和 CWD,将 AL 中的数据的最高位(符号位)扩展到 AH 的 8 位中,或者把 AX 中的最高位扩展到 DX 的 16 位中。

5. 符号扩展指令

带符号扩展指令有两种：字节扩展指令 CBW 和字扩展指令 CWD。

1) 字节扩展指令 CBW

格式：

```
CBW
```

其为隐含操作数 AL、AH,功能是将一个字节的带符号数扩展成一个字,即将 AL 中的符号扩展到整个 AH 中。

(1) 如果 AL 最高位为 1(AL 值为负),则(AH)＝0FFH；

(2) 如果 AL 最高位为 0(AL 值为正),则(AH)＝00H。

2) 字扩展指令 CWD

格式：

```
CWD
```

其为隐含操作数 AX、DX,功能是一个字的带符号数扩展成双字,即将 AX 中的符号扩展到整个 DX 中。

(1) 如果 AX 最高位为 1(AX 值为负),则(DX)＝0FFFFH；

(2) 如果 AX 最高位为 0(AX 值为正),则(DX)＝0000H。

6. 十进制运算指令

8421BCD 编码用 4 位二进制表示一位十进制数,4 位二进制数表示范围是 0000～1111,对应十进制数 0～15,而一位十进制数表示范围是 0～9,所以,当某十进制位运算值超过 9 或发生进位或借位时,运算结果会出错,需要调整。BCD 调整指令用于对 BCD 数进行加、减、乘、除运算时的调整,使结果正确,并且仍然是 BCD 编码形式。

下面以具体例子说明加、减法的基本调整原理。

【例 3-32】 非组合 BCD 调整。

(1) 8+7=15

```
  0000 1000   8H
 +0000 0111   7H
-------------
0000 1111   1FH,结果错误,因为 1111 > 9,加 6 调整
 +0000 0110
-------------
  0001 0101   15H 正确结果
```

(2) 9+8=17

```
  0000 1001   9H
 +0000 1000   8H
-------------
  0001 0001   11H,结果错,因为低位向高位有进位(AF=1),需加 6 调整
 +0000 0110
-------------
  0001 0111   17H 正确结果
```

可见,运算中,若和大于 9 或有辅助进位(AF=1)就需加 6 调整。同理,减法中,若差大于 9 或有辅助借位(AF=1),就需减 6 调整。

【例 3-33】 压缩的 BCD 数调整。

(1) 18+7=25

```
  0001 1000   18H
 +0000 0111   7H
-------------
  0001 1111   1FH,结果错误,因为 1111 > 9,需加 6 调整
 +0000 0110
-------------
  0010 0101   25H 正确结果
```

(2) 19+98 = 27

```
  0001 1001   19H
 +1001 1000   98H
-------------
  1011 0001   B1H,结果错。首先,低位向高位有进位(AF=1),加 6 调整
 +0000 0110
-------------
  1011 0111   B7H 结果错。首先,高位(1011)2 > 9,需加 60 调整
 +0110 0000
-------------
1 0001 0111   117H 结果对
```

可见,在压缩的 BCD 数加法运算时,若低 4 位和大于 9 或有辅助进位(AF=1)就需在低 4 位“加 6 调整”;若高 4 位和大于 9 或有进位(CF=1)就需在高 4 位“加 6 调整”。同理,在 BCD 数减法运算时,若低 4 位差大于 9 或有辅助借位(AF=1)就需在低 4 位“减 6 调整”;若高 4 位差大于 9 或有借位(CF=1),就需要在高 4 位“减 6 调整”。

1) 加法压缩 BCD 码调整指令 DAA

DAA 加法十进制调整指令,能够校正 AL 中一个字节的压缩 BCD 码相加的结果(高 4 位表示十进制十位数,低 4 位表示十进制个位数),并将校正结果存入 AL,DAA 指令为无操作数指令,隐含操作数为 AL,该指令常跟随加法指令 ADD、ADC 之后。DAA 调整指令会影响 CF、AF、SF、ZF、PF 等标志位。

格式:

```
DAA
```

指令调整过程如下:

(1) 如果 AL 寄存器中低 4 位大于 9 或辅助进位(AF)=1,则(AL)=(AL)+6 并且(AF=1);

(2) 如果(AL)≥0A0H 或(CF)=1,则(AL)=(AL)+60H 并且(CF)=1。

【例 3-34】

```
ADD AL,BL              ; 若计算 67 与 33 的和,用 BCD 码表示则(AL) = 67H,(BL) = 33H,
                       ; 指令执行后,(AL) = 9AH
DAA                    ; 调整 AL 内容,则用压缩 BCD 码表示,则(AL) = 00H, CF = 1
```

2) 加法非压缩 BCD 码调整指令 AAA

AAA 非组合 BCD 码加法十进制调整指令,能够校正 AL 中一个字节的非压缩 BCD 码相加的结果(高 4 位表示十进制十位数,低 4 位表示十进制个位数),并将校正结果存入 AX,AAA 指令为无操作数指令,隐含操作数为 AL,该指令常跟随加法指令 ADD、ADC 之后。AAA 调整指令只会影响 CF 和 AF 标志位。其格式为:

```
AAA
```

指令调整过程如下:

(1) 如果 AL 寄存器中低 4 位大于 9 或辅助进位(AF)=1,则(AL)=(AL)+6、(AH)=(AH)+1 并且 AF=1、CF=1;

(2) (AL)=(AL) AND 0FH。

【例 3-35】

```
ADD AL,BL              ; 若计算 7 与 3 的和,用 BCD 码表示则(AL) = 07H,(BL) = 03H,
                       ; 指令执行后,(AL) = 0AH
AAA                    ; 调整 AL 内容,则用压缩 BCD 码表示,则(AL) = 10H,CF = 1
```

3) 减法调整指令 DAS

DAS 减法调整指令与 DAA 指令相似,用于修正 AL 中一个字节的压缩 BCD 码相减的结果,并将校正结果存入 AL,该指令常跟随减法指令 SUB、SBB 之后。DAS 调整指令会影响 CF、AF、SF、ZF、PF 等标志位。

格式:

```
DAS
```

指令调整过程如下:

(1) 如果(AL)中低 4 位大于 9 或辅助进位(AF)=1,则(AL)=(AL)-6 并且(AF=1);

(2) 如果(AL)≥0A0H 或(CF)=1,则(AL)=(AL)-60H 并且(CF)=1。

【例 3-36】

```
SUB AL,BL              ; 若计算 63 与 37 相减,用 BCD 码表示则(AL) = 63H,(BL) = 37H,
                       ; 指令执行后,(AL) = 2CH,AF = 1,个位有借位
DAS                    ; 调整 AL 内容,则用压缩 BCD 码表示,则(AL) = 26H,CF = 1
```

4) 减法 ASCII 调整指令 AAS

AAS 减法调整指令与 AAA 指令相似,用于修正 AL 中一个字节的非压缩 BCD 码相减的结果,并将校正结果存入 AX,该指令常跟随减法指令 SUB、SBB 之后。DAS 调整指令会

影响 CF、AF、SF、ZF、PF 等标志位。

格式：

```
AAS
```

指令调整过程如下：

(1) 如果 AL 寄存器中低 4 位大于 9 或辅助进位(AF)＝1，则(AL)＝(AL)－6(AH)＝(AH)－1 并且 AF＝1、CF＝1；

(2) (AL)＝(AL)　AND0FH。

【例 3-37】

```
SUB AL,BL              ; 用 BCD 码表示则(AL) = 63H,(BL) = 37H,指令执行后,(AL) = 27H,
                       ; AF = 1,个位有借位
DAS                    ; 调整 AL 内容,用压缩 BCD 码表示,则(AL) = 06H,CF = 1
```

5) ASCII 码乘法调整指令 AAM

AAM 非组合 BCD 码乘法十进制调整指令，能够校正 AX 中一个字节的非压缩 BCD 码直接相乘的结果，并将校正结果存入 AX，AAM 指令也为误操作数指令，隐含操作数为 AX，该指令常跟随乘法指令 MUL、IMUL 之后。该指令影响标志位 SF、ZF 和 PF。

格式：

```
AAM
```

【例 3-38】

```
MULDL              ; 如果 BCD 码(AL) = 02H,(DL) = 05H,则指令执行后(AH) = 00H,
                   ; (AL) = 0AH,该结果大于 9,不是 BCD 码
AAM                ; 指令对(AX) = 000AH 进行修正,结果为(AX) = 0100H,
                   ; 正好等于 10 的 BCD 码表示,即(AH) = 01H,(AL) = 00H
```

6) ASCII 码除法调整指令 AAD

AAD 除法调整指令用于对 BCD 码进行除法运算，校正 AX 中的非压缩 BCD 码，同 AAM 乘法调整相似，只是除法调整是在 DIV、IDIV 指令执行前进行。实际上是将 AX 中的非压缩 BCD 码被除数转换成无符号二进制数，该指令也为无操作数指令，指令只影响标志位 SF、ZF 和 PF。

格式：

```
AAD
```

AAD 指令的调整过程自动完成：将(AH)×0AH＋(AL)的结果送 AL，然后将 AH 清 0。

【例 3-39】

```
AAD              ; 如果(AX) = 0708H = 78D,(BL) = 02H = 2D,指令执行后,(AX) = 004EH = 78D
DIV BL           ; 商(AL) = 39H,余数(AH) = 00H
```

3.2.4　逻辑操作指令

逻辑操作指令可以分为逻辑运算指令和移位运算指令两大类，均可对二进制数进行操

作。逻辑运算指令包括 AND、TEST、OR、XOR 和 NOT。移位运算指令包括 SHL、SHR、SAL、SAR、ROL、ROR、RCL 和 RCR。

1. 逻辑运算指令

逻辑运算指令能够对 8 位、16 位操作数进行逻辑运算，因为逻辑运算类指令是在 ALU 中完成的，所以除 NOT 指令外，均对 PF、SF 及 ZF 有影响，而 CF＝OF＝0，AF 不确定。

1）逻辑与指令 AND

逻辑与指令 AND 实现两个操作数按位进行逻辑与运算，运算结果存放到目的操作数。AND 常用于目的操作数与源操作数 0 相对应位清 0。

格式：

```
AND dest,src
```

【例 3-40】

```
AND  AL,0FH              ; AL 中高 4 位清 0
AND  [BL],0AAH           ; 使存储单元[BL]数据第 1、3、5、7 位为 0
```

2）测试指令 TEST

测试指令 TEST 与 AND 指令相似，也执行两个操作数指令相与运算，不同的是，TEST 运算结果不回送，而仅仅影响标志位。通常，TEST 指令用来检测指定的值是 1 还是 0、某数的奇偶性、某数的正负。

格式：

```
TEST dest,src
```

【例 3-41】

```
TEST  AL,08H             ; 测试第 4 位是否为 1,若为 1,则 ZF = 0; 若为 0,则 ZF = 1,
                         ; 指令执行后 AL 值不变
TEST  AL,01H             ; 测试第 0 位是否为 1,若为 1,则 ZF = 0,表示该数为奇数
```

3）逻辑或指令 OR

逻辑或指令 OR 能够实现两个操作数按位进行逻辑或运算，运算结果存放到目的操作数。OR 指令常用于对目的操作数的某些位置 1。

格式：

```
OR dest,src
```

【例 3-42】

```
OR AL,1AH            ; 将 AL 数据第 1、3、4 位置 1
OR AL,01H            ; 将 AL 数据最低位置 1,其他位不变
```

4）逻辑异或指令 XOR

逻辑异或指令 XOR 能够实现两个操作数按位进行异或运算，运算结果存放到目的操作数。XOR 指令常用于对目的操作数的某些位置取反，其他位保持不变。

格式：

```
XOR dest,src
```

【例 3-43】

```
XOR  BL,08H          ; BL 的最低位取反,其他位保持不变
XOR  [AL],04H        ; [AL]内存数的第 3 位取反,其他位不变
XOR  AX,AX           ; 通过 XOR,使 AX 寄存器清 0
```

5）求反指令 NOT

求反指令 NOT 是单操作数指令，能够实现操作数的每一位按位取反运算，该指令不影响标志位。

格式：

```
NOT dest
```

dest 可以为 reg8、reg16 或 mem。

【例 3-44】

```
MOV  AX,2010H
NOT  AX          ; 指令执行完后,(AX) = 0DFEFH
```

2. 移位类指令

8086 系统提供 8 条移位指令，包括算术移位指令、逻辑移位指令、循环移位指令和带进位循环移位指令共 4 类，分左移和右移操作共 8 条。这 8 条指令有统一的指令格式，格式如下：

```
操作符 dest,src
```

其中，操作符包括 SAL 算术左移指令、SAR 算术右移指令、SHL 逻辑左移指令、SHR 逻辑右移指令、ROL 不带进位的循环左移指令、ROR 不带进位的循环右移指令、RCL 带进位的循环左移指令和 RCR 带进为的循环右移指令；目的操作数可以为 8 位或 16 位寄存器和存储器操作数。若移位次数为 1，则直接在指令中指出移位次数；若移位次数大于 1，则可先将移位次数存入 CL，指令中移位次数必须为 CL 寄存器。

指令执行操作说明：

（1）SAL 算术左移指令、SAR 算术右移指令、SHL 逻辑左移指令和 SHR 逻辑右移指令 4 个指令均为非循环移位指令。4 个指令均可对带符号数和无符号数进行运算。

（2）右移指令：算术右移时将操作数看成带符号数，保持最高位的值不变；逻辑右移时把操作数看成无符号数，执行时最高位添 0。

（3）左移指令：算数左移指令 SAL 和逻辑左移指令 SHL 的功能完全一样，每移位一次，最低位补 0，最高位进入 CF。在左移位数为 1 的情况下，影响溢出标志，如果最高位和 CF 不同，则溢出标志 OF＝1，对带符号数来说，可由此判断移位后的符号位和移位前的符号位改变；如果移位后的最高位和 CF 相同，则 OF＝0，表明移位前后符号位没有变。在左移位数大于 1 的情况下，指令对溢出标志不产生影响。

（4）向左移位指令可以看成目的操作数乘 2n（n＝1 或 CL）；向右移位指令可以看成目的操作数除 2n（n＝1 或 CL）。对无符号数乘（除）2n 时，可采用算术移位指令、对带符号数乘（除）2n 时，可采用逻辑移位指令代替相应的乘除指令。

(5) ROL 不带进位的循环左移指令、ROR 不带进位的循环右移指令、RCL 带进位的循环左移指令和 RCR 带进位的循环右移指令,均为循环移位指令,常用于多字节的移位操作中。4 个指令均可对带符号数和无符号数进行运算。

(6) ROL 和 ROR 为简单循环指令。ROL 是循环左移指令,其每移动一次,操作数的最高位同时移到进位标志 CF 及操作数的最低位中,其他位依次向左移动一位。RCR 与 RCL 移位方向相反,其每移动一次,操作数的最低位移到 CF 中,而移位前的标志位 CF 则移到操作数的最高位中,其他位依次向右移动一位。

(7) RCL 和 RCR 指令是连同 CF 值一起循环指令。RCL 是带进位 CF 循环左移指令,其每移动一次,操作数的最高位移到 CF 中,而移位前的标志位 CF 则移到操作数的最高位中,其他位依次向左移动一位。

(8) 所有的移位指令在执行时,都会影响标志位 CF、OF、PF、SF 和 ZF。

注意:

① 段寄存器不能参与逻辑操作指令;

② 立即数不能为目的操作数;

③ 不能直接对两个内存操作数进行逻辑操作,必须先把其中一个放到通用寄存器中;

④ 在 8086 系统中,当移位次数大于 1 时一定要用 CL 来存放移位次数。

【例 3-45】 完成寄存器 AX 中 16 位逻辑左移 2 位和 DX:BX 中的 32 位整数逻辑左移 1 位。

分析:由于 8086 移位运算类指令一次只能对字节和字数据进行操作,因此必须先对低位寄存器移位,再对高位寄存器移位。

```
MOV  CL,2
SHL  AX,CL
SHL  BX,1          ; BX 逻辑左移 1 位
RCL  DX,1          ; DX 带进位循环左移 1 位
```

程序中当低 16 位寄存器逻辑左移后,其最高位进入了 CF 标志寄存器中,而高 16 位寄存器逻辑左移是通过带进位循环左移指令实现的。如果要完成寄存器 DX:AX 中的 32 位整数逻辑右移 1 位,程序将如何改动?请读者思考。

3.2.5 字符串操作指令

字符串操作指令的实质是对连续的存储单元中的字串或字节串进行处理。字符串操作指令是 8086 指令系统中唯一可在指令中同时出现两个内存单元的指令,包括 MOVS、LODS、STORS、CMPS 和 SCAS,它们能分别实现字符串的传送、读取、存储、比较、扫描功能。

下面具体介绍几种常用串操作指令的功能及用法。

1. 串传送指令 MOVS

MOVS 串传送指令能将源串中指针 DS:SI 所指的字节或字传送到目的串中指针 ES:DI 所指的存储单元中,并根据 DF 方向标志值修改 SI、DI,使其指向下一存储单元。MOVS 指令对标志寄存器没有影响。

格式:

```
MOV SB          ; 字节串传送指令
MOV SW          ; 字串传送指令
```

指令执行前，必须进行如下设置：

(1) 设置隐含的寄存器 SI、DI，由 SI 和 DI 作为串操作存储单元地址指针。

(2) 设置方向标志位 DF。当 DF＝0 时，变址寄存器 SI(或 DI)增加 1 或 2；当 DF＝1 时，变址寄存器 SI(或 DI)减少 1 或 2。

(3) 设置字符串长度 CX，由 CX 进行计数。

【例 3-46】 传送 10B 的字符串程序段。

```
    CLD
    MOV  SI,1000H
    MOV  DI,2000H
    MOV  CX,10
 KKK:
    MOV  SB
    DEC  CX
    JNZ  KKK
```

另外，为进一步简化程序，MOVS 串传送指令可以与重复前缀 REP 配合使用，上述程序改写为：

```
CLD
MOV  SI,1000H
MOV  DI,2000H
MOV  CX,10
REP  MOVSB
```

REP 重复前缀在 MOVS 串传送指令之前可以自动完成以下功能：

① CX＝0 时，串操作执行完毕。否则执行②③；

② CX 的值减 1；

③ 执行一次 MOVS 串操作指令；

④ 转到①重复 REP 操作。

这样利用 REP 来自动完成 DEC CX 和 JNZ 的操作。

2. 存字符串数据命令 STOS

STOS 指令能把寄存器 AL、AX 中的数据，传送到目的串中指针 DI 所指的字节或字并相应的修改 DI，使其指向串中下一个数据单元。该指令对标志寄存器没有影响。

格式：

```
STOSB           ; 存字节串指令
STOSW           ; 存字串指令
```

存字节串指令 STOSB，隐含操作数 AL，目的操作数 ES：DI。存字串指令 STOSW，隐含源操作数 AX，目的操作数 ES：DI。

STOS 与重复前缀 REP 配合使用时，表示将 AL 或 AX 中的数据装入连续的内存单元中，使得该连续内存单元数值相等。

STOS 执行前设置与 MOVS 指令相似。

【例 3-47】 将 0FFH 存入 50 个内存字节单元。

```
CLD
MOV  DI,2000H
MOV  AL,0FFH
MOV  CX,50
REP  STOSB
```

3. 取字符串数据指令 LODS

LODS 指令能把源串中指针 SI 所指的字节或字传送到寄存器 AL 或 AX 中，并相应地修改 SI，使其指向串中下一个数据单元。该指令常与 STOS 指令配合使用而 LODS 前不加 REP，实现“从存储器读取数据—处理数据—保存到存储器中”或“STOS 存数据、LODS 数据，处理数据”的功能，对标志寄存器没有影响。

格式：

```
LODSB          ; 取字节串指令
LODSW          ; 取字串指令
```

LODSB 取出字节串指令，隐含目的操作数 AL，源操作数 DS：SI；LODSW 取字串指令，隐含目的操作数 AX，源操作数 DS：SI。

【例 3-48】 试写出利用 STOS、LODS 指令转换字母大小写的程序段。

```
    CLD
    LEA  SI,SOURCE
    LEA  DI,DESTINATE
    MOV  CX,20
 KKK:
    LODSB
    OR AL,00100000B
    STOSB
    DEC  CX
    JNZ  KKK
```

4. 比较字串指令 CMPS

CMPS 指令能对源串和目的串中的指针 SI 和 DI 所指的字节或字计算目的操作数减源操作数的值，比较两内存单元值的大小，但不保存相减结果，只改变标志寄存器中相应的标志位，并相应修改 SI、DI，使其指向串中下一个单元。CMPS 指令常用于寻找两个字符串中第 1 个相等或第 1 个不相等的元素。CMPS 对标志位都有影响。

```
CMPSB          ; 比较字节串操作指令
CMPSW          ; 比较字串操作指令
```

比较字节串指令 CMPSB，隐含源操作数 DS：SI，目的操作数 ES：DI。比较字串指令 CMPSW，隐含源操作数 DS：SI，目的操作数 ES：DI。

CMPS 可与重复前缀 REPE/REPZ（相等时重复）、REPNE/REPNZ（不等时重复）配合使用，自动完成以下功能：

(1) 如果(CX)=0,则结束串比较操作,否则执行(2)、(3)、(4)。

(2) CX 的值自动减 1。

(3) 执行一次串比较操作指令。

(4) 如果零标志位 ZF=1(REPE/REPZ)或 ZF=0(REPNE/REPNZ),则转回(1)。

(5) ZF=0 完成 REPE/REPZ 操作或 ZF=1 结束 REPNE/REPNZ 操作。

5. 扫描字符串指令 SCAS

SCAS 指令能够比较寄存器 AL 或 AX 与目的串中指针 DI 所指的内存单元 ES: DI 字节或字大小,同时修改 DI 使其指向串中下一个数据单元。SCAS 指令常用来搜索目的串中是否含有与 AL 或 AX 中数据相同或不同的某个元素。SCAS 指令对标志位有影响。

格式:

```
SCASB          ; 字节扫描字符串操作指令
SCASW          ; 字扫描字符串操作指令
```

扫描字符串指令 SCASB,隐含源操作数 AL,目的操作数 ES: DI。比较字符串指令 SCASW,隐含操作数 AX,目的操作数 ES: DI。

SCAS 指令同 CMPS 相似,可与重复前缀 REPE/REPZ (相等时重复)、REPNE/REPNZ(不等时重复)配合使用。

【例 3-49】 在 100 个字串中寻找第 1 个与 0F88H 相同的值,并送 1 给 DL。

```
CLD
LEA   SI,SOURCE
MOV   DI,2000H
MOV   CX,100          ; 串长为 100
MOV   AX,0F88H        ; 搜索的字元素
REPNE SCASW           ; REPNE 表示不相等时继续搜索下一个字
JNE   KKK             ; 如果 100 个字中都找不到(AX),则转去处理 KKK
MOV   DL,01H          ; 若果找到(AX),使(DL) = 1
```

字符串操作指令有许多相似之处,例如:

(1) 指令默认源字符串在数据段中,即段地址为 DS,且有效地址指针为 SI。源字符串可以用段超越的方法来指定段。

(2) 目的字符串只能在附加段中,即段地址为 ES,且有效地址指针为 DI。

(3) 串处理指令是隐含 SI 和 DI 为间址的间接寻址方式。

(4) 串处理的方向取决于方向标志 DF,DF=0 时,地址指针 SI 和 DI 增量(+1 或+2);DF=1 时,地址指针 SI 和 DI 减量(-1 或-2)。程序员可以使用指令 CLD 和 STD 来设置方向标志。

(5) MOVS、STOS、LODS 指令不影响条件码,CMPS、SCAS 指令根据比较的结果设置条件码。

(6) 串操作指令都能和重复前缀(REP、REPE/REPZ、REPNE/REPNZ)配合使用,以简化程序。与指令 MOVS 和 STOS 联用的重复前缀是 REP,取串指令一般不加重复前缀;与指令 CMPS 和 SCAS 联用的重复前缀是 REPE/REPZ 和 REPNE/REPNZ。重复前缀只能用于串操作指令中,在其他指令中无效。

(7) 执行串操作指令前必须设置隐含地址指针 SI 或 DI,设置方向标志位 DF,设置字符串长度 CX。

(8) 串操作一般分两步执行:第一步完成处理功能,如传送、存取、比较等;第二步进行指针修改,以指向下一个要处理的字节或字。

3.2.6 程序控制转移指令

程序控制类指令是指通过该表 CS、IP 或只改变 IP 的值以控制程序的执行顺序,实现指令转移、程序调用等功能。控制转移类指令包括 5 类指令,即无条件转移指令、条件转移指令、循环控制指令、子程序调用与返回指令、中断指令。

1. 无条件转移指令 JMP

JMP 指令控制程序无条件地跳转到 IP 或 CS:IP 所指的目的单元。

格式:

```
JMP  标号
```

其中标号有 3 种形式。

(1) 段内进转移。"JMP NEAR PTR LABLE"是指令 JMP 的默认格式,可以表示为"JMP LABLE"。它可在当前代码段内转移,位移量是 16 位的带符号补码数,其范围为 0000H～FFFFH,执行时转向地址是 IP 当前值加 16 位位移量。其寻址方式可以是直接寻址、寄存器寻址或存储器寻址。

【例 3-50】

```
JMP  LABLE              ; IP←OFFSET 标号,实现段内的转移
JMP  BX                 ; IP←(BX)
JMP  BYTE PTR [DI]      ; IP←DS: [DI]由 DS: DI 地址确定,存储器内容送 IP
```

(2) 段内短转移。"JMP SHORT PTR LABLE"是当前段内的转移,位移量是 8 位的带符号补码数,转移地址范围为(IP)－128～(IP)＋127,其范围为 00H～FFH,执行时转向地址是 IP 当前值加 8 位位移量。其寻址方式可以是直接寻址、寄存器寻址和存储器寻址。

(3) 段间远转移。"JMP FAR PTR LABLE"实现段间的跳转指令,是从当前代码段跳转到另一个代码段中,这意味着不仅改变 IP 值,还会改变 CS 的值,其寻址方式可以是除立即数寻址之外的任何寻址方式。

2. 条件转移指令

条件转移指令是段内短转移指令,即 IP 值改变的最大范围为－128～＋127。条件转移指令是一组重要的转移指令,它根据标志寄存器中的标志位来决定是否需要转移,以满足各种不同的转移需要。

格式:

```
条件转移指令标识符   标号
```

条件转移指令分为三大类:基于带符号数的条件转移指令、基于无符号数的条件转移指令和基于算术标志位的条件转移指令。如表 3.2 所示。

表 3.2　条件转移指令

分类	指令	转移条件	说　明
无符号数比较	JG/JNLE	(SF XOR OF)=0 且 ZF=0	大于/不小于或等于,转移,否则顺序执行
	JGE/JNL	(SF XOR OF)=0	大于或等于/不小于,转移,否则顺序执行
	JL/JNGE	(SF XOR OF)=1	小于/不大于或等于,转移,否则顺序执行
	JLE/JNG	(SF XOR OF)=1 或 ZF=1	小于或等于/不大于,转移,否则顺序执行
有符号数比较	JZ/JE	ZF=1	相等/比较值为零,转移,否则顺序执行
	JNZ/JNE	ZF=0	不相等/比较值不为零,转移,否则顺序执行
	JA/JNAE	CF=0 且 ZF=0	大于/不小于或等于,转移,否则顺序执行
	JAE/JNB	CF=0	大于或等于/不小于,转移,否则顺序执行
	JB/JNAE	CF=1	小于/不大于或等于,转移,否则顺序执行
	JBE/JNA	CF=1 或 ZF=1	小于或等于/不大于,转移,否则顺序执行
根据算术标志值	JC	CF=1	有进位转移,否则顺序执行
	JNC	CF=0	无进位,转移到标号处执行,否则顺序执行
	JO	OF=1	有溢出,转移到标号处执行,否则顺序执行
	JNO	OF=0	无溢出,转移到标号处执行,否则顺序执行
	JS	SF=1	结果为负,转移,否则顺序执行
	JNS	SF=0	结果为正,转移,否则顺序执行
	JP/JPE	PF=1	结果低 8 位偶数个 1,转移,否则顺序执行
	JNP/JPO	PF=0	结果低 8 位奇数个 1,转移,否则顺序执行
	JCXZ	CX=0	计数为 0,转移到标号处执行,否则顺序执行

3. 循环控制指令

循环控制指令用在循环程序中,控制一段程序的重复执行,其重复次数由计数 CX 值是否为零决定,循环指令均不影响条件码。

格式:

LOOP 标号

其中,标号用来表示在汇编指令中循环的转向地址,CPU 执行循环指令时,若满足循环条件,就计算转向地址:当前(IP)+8 位位移量→(IP),即实现循环。若不满足循环条件,则退出循环,程序继续顺序执行。循环指令都是段内短转移指令,位移量是用 8 位带符号数来表示的,转向地址在当前 IP 值的−128B～+127B 范围之内。

循环指令根据循环条件有 3 种形式。

(1) 计数循环指令 LOOP:指令执行时,(CX)减 1→(CX),测试循环次数,若(CX)≠0,则转到“(IP)=OFFSET 标号”执行,否则循环结束,执行下一条指令。

(2) 非零计数循环指令 LOOPNZ/LOOPNE:指令执行时,(CX)减 1→(CX),测试循环次数,若(CX)≠0,再查看 ZF 标志,若 ZF=1,则转到“(IP)=OFFSET 标号”执行,否则循环结束,执行下一条指令。当 ZF=1 且(CX)≠0 时循环;ZF=0 或(CX)=0 时退出循环。

(3) 零计数循环指令 LOOPZ/LOOPE:指令执行时,(CX)减 1→(CX),测试循环次数,若(CX)≠0,再查看 ZF 标志,若 ZF=0,则转到“(IP)=OFFSET 标号”执行,否则循环结束,执行下一条指令。当 ZF=0 且(CX)≠0 时循环;ZF=1 或(CX)=0 时退出循环。

对条件循环指令 LOOPZ(LOOPE)和 LOOPNZ(LOOPNE),除测试 CX 中的循环次数

外，还将 ZF 的值作为循环的必要条件，因此，运用该指令时要注意将条件循环指令紧接在形成 ZF 的指令之后。

需要注意的是，循环指令中隐含 INC CX 操作，因此在使用循环指令时，不要再增加 INC CX 指令。

【例 3-51】 将内存数据段 STRING1 单元开始存放的一个长度为 20 的字符串，复制到附加段 STRING2 开始的单元。

字符串复制的过程很简单，首先设置两个地址指针，使它们分别指向源字符和目的字符串的首地址，然后按照一定的顺序将字符串中的字符一个一个送入目标单元，每送完一个字符，源字符串和目的字符串的地址指针都要指向下一个字符。编写的程序段如下：

```
MOV  SI,OFFSET STRING1      ; OFFSET 为取 STRING1 的有效地址
MOV  DI,OFFSET STRING2
MOV  AL,[SI]
MOV  ES: [DI],AL
INC  SI
INC  DI
MOV  AL,[SI]
MOV  ES: [DI],AL
INC  SI
INC  DI
```

观察上面的程序段，发现第 3～6 行指令序列与第 7～10 行指令序列完全相同，这段指令序列实际上完成的是复制一个字符，由于有 20 个字符，故它应该重复执行 20 次。但每次执行时，地址指针 SI 和 DI 的值都已经变化，比上一次的值大 1。

对于这类“处理过程相同，只是每次处理的数据有所不同，而数据的变化是有规律”的问题，如果仍然采用顺序结构编写程序，程序将会变得十分冗长。

如果把程序中需要反复执行的相同程序段只书写一次，然后给出条件，通过条件来控制是否可以重复执行，就可以解决用顺序结构程序无法实现的问题。

按照这种程序设计思想，把上面第 3～6 行指令序列写成一个程序段，用 CX 寄存器作为计数器，初始值设为 20，每执行一次指令序列，CX 减 1，通过判断 CX 是否等于 0，来控制该指令序列执行 20 次。该程序可简化为：

```
    MOV  SI,OFFSET STRING1
    MOV  DI,OFFSET STRING2
    MOV  CX,20
AGAIN:
    MOV  AL,[SI]
    MOV  ES: [DI],AL
    INC  SI
    INC  DI
    DEC  CX
    JNZ  AGAIN
```

前、后两段程序进行比较，不难看出，按照“控制程序段重复执行一定次数”的程序设计思想编写程序，程序代码大大减少。

4. 子程序调用与返回指令

子程序是计算机程序设计中一种非常重要的编程结构，如果某程序在源程序中反复出现，那么就可以把该程序定义为子程序。汇编语言提供了定义子程序的方法，并将子程序存储在存储器中，可供一个或多个主程序反复调用，主程序和子程序可以在同一段内，也可以不在同一段内。主程序调用子程序时使用 CALL 指令，子程序返回主程序时使用 RET 指令，CALL 指令和 RET 指令有近调用、近返回及远调用、远返回两类格式。

(1) 定义子程序，格式为：

```
子程序名 PROC [NEAR/FAR]
…; 子程序体
子程序名 ENDP
```

对子程序定义的具体规定如下：

① 子程序必须是一个合法的标识符，并且要前后一致。

② 子程序名有段值、偏移量和类型 3 个属性。段值和偏移量对应于子程序的入口地址，类型就是该子程序的类型。

③ PROC 和 ENDP 必须成对出现，它们分别表示子程序定义的开始和结束。

④ 子程序的类型有近(NEAR)、远(FAR)之分，其默认的类型是近调用 NEAR 型。

(2) 子程序调用指令 CALL，格式为：

```
CALL FAR PTR SUBROUT;          ; 段间直接调用
```

上述指令执行：

① (SP)←(SP)－2，((SP))←(CS)当前

(SP)←(SP)－2，((SP))←(IP)当前

② (IP)←偏移地址(在指令的第 2、3 字节中)

(CS)←段地址(在指令的第 4、5 字节中)

```
CALL WORD PTR DESTIN           ; 段间间接调用
```

上述指令执行：

① (SP)←(SP)－2，((SP))←(CS)当前

(SP)←(SP)－2，((SP))←(IP)当前

② (IP)←(EA)，(CS)←(EA＋2)；EA 为指令寻址方式所确定的有效地址

从 CALL 指令执行的操作中看出，首先是把当前执行指令的下一条指令即子程序返回调用程序的地址保存在堆栈中。对段内调用，只需保存 IP 当前值，即 CALL 指令的下一条指令的地址存入 SP 所指示的堆栈单元中。对段间调用，保存返回地址则意味着要将 CS 和 IP 的当前值分别存入堆栈的两个字单元中。

CALL 指令的第二步操作是转子程序，即把子程序的入口地址交给 IP(段内调用)或 CS：IP(段间调用)。对段内直接方式，转移的位移量，即子程序的入口地址和返回地址之间的差值就在机器指令的第 2、3 个字节中。对段间直接方式，子程序的偏移地址和段地址就在操作码之后的两个字节中。对间接方式，子程序的入口地址就从寻址方式所确定的有效地址中获得。

(3) 子程序返回指令 RET

RET 指令执行的操作是将保存在堆栈中的主程序断点地址返回到 IP。其格式为：

RET 段内返回(近返回)

上述执行指令：

(IP)←((SP)),(SP)←(SP)+2

RET 段间返回(远返回)

上述指令执行：

① (IP)←((SP)),(SP)←(SP)+2

② (CS)←((SP)),(SP)←(SP)+2

RET　　N(带立即数返回；N 表示压入堆栈中的字节数)

上述指令执行：

① 返回地址出栈(操作同段内或段间返回)

② 修改堆栈指针(SP)←(SP)+N

RET 指令一定出现在子程序的最后，以返回到主程序。子程序的调用和返回是一对互逆操作，如果是段内返回，则只需把保存在堆栈中的偏移地址取出存入 IP；如果是段间返回，则要把偏移地址和段地址都从堆栈中取出送到 IP 和 CS 寄存器中。

子程序调用和返回指令是一种特殊的转移指令。一方面，当执行 CALL 指令时，程序的执行顺序被改变，CPU 将转而执行子程序。因此说子程序调用含有转移指令的功能。同样，子程序的返回指令 RET 使 CPU 从执行子程序转而跳转去执行 CALL 指令处的后续指令，因而返回指令 RET 也具有转移特性。另一方面，子程序调用和返回指令不同于转移指令。转移指令是一种“一去不复返”的操作，而子程序调用当子程序执行结束时，还要求 CPU 能转而执行调用指令处的后续指令，它是一种“有去有回”的操作。

5. 中断及中断返回指令

中断和中断返回指令为程序员提供了软件中断的手段，中断指令用于调用中断服务程序，中断返回指令与中断指令相反，用于结束中断服务程序返回被中断的程序。

其格式为：

INT n

指令执行：

(SP)←(SP)-2,((SP))←(FLAGS)
(SP)←(SP)-2,((SP))←(CS)
(SP)←(SP)-2,((SP))←(IP)
(IP)←(n×4),(CS)←(n×4+2)

在 INT 指令中，n 为中断类型号，用于寻找中断程序入口地址。8086 系统给每个中断程序一个编号(即中断类型编号)，各种中断程序的入口地址按中断类型号的顺序存储在中断向量表中，每个中断程序的入口地址占用 4 个字节，在中断向量中的地址由中断类型号乘 4 得到，因此执行中断指令时，可分为 3 步：

第一步，保存调用程序的现场。将断点处调用程序的标志寄存器和断点的地址入栈；

第二步，取中断程序入口地址。根据 n×4 在中断向量表中查找中断程序入口地址；

第三步，调用中断程序。将中断向量表中有效地址为 n×4 和 n×4＋1 的内存内容送 IP，n×4＋2、n×4＋3 两单元的内容送 CS，转而执行中断程序。

中断类型号可以为 0～255，常用的有 8 种：20H～27H，其中，INT 21H 是系统功能调用，本身含有 80 多个子程序，每个子程序对应一个功能号，编号范围为 00H～57H。调用系统中的子程序具有统一格式，只需 3 个语句：

(1) 传送入口参数到指定寄存器；

(2) 功能号送 AH 或 AX 寄存器；

(3) 执行 INT 21H。

3.2.7　处理器控制指令

8086 系统专门用于处理器控制的指令，包括标志位操作指令及处理器指令。

1. 标志位操作指令

标志位操作指令是一组对标志位 CF、DF、IF 置位、复位和求反操作的指令。包括 CLC、STC、CMC、CLD、STD、CLI 及 STI，分别用于修改进位标志 CF、方向标志 DF 和中断允许标志 IF。标志位操作指令，是无操作数指令，且只影响本指令指定的标志位，而不影响其他标志位。标志位操作指令及功能如表 3.3 所示。

表 3.3　标志位操作指令及功能表

类　型	指　令	功　能	说　明
修改进位标志	CLC	CF＝0	将 CF 清 0
	STC	CF＝1	将 CF 置 1
	CMC	CF 取反	
修改方向标志	CLD	DF＝0	将 DF 清 0
	STD	DF＝1	将 DF 置 1
修改中断允许标志	CLI	IF＝0	将 IF 清 0
	STI	IF＝1	将 IF 置 1

2. 处理器指令

处理器指令是一组控制 CPU 工作方式的指令。这组指令的使用频率不高。

1) 空操作指令 NOP

CPU 执行该指令不完成任何具体功能，只占用 3 个时钟周期。该指令可使相邻两条指令的执行有一点间隔。

2) 等待指令 WAIT

该指令用于测试 CPU 的 TEST/BUSY 引线。当 TEST/BUSY 线为高电平时，CPU 进入等待状态，且每隔 3 个时钟周期对 TEST/BUSY 的状态进行一次测试，直到 TEST 引线出现低电平，CPU 退出等待，顺序执行下一条指令。在等待期间，系统可以对外部中断源中断，以避免无休止的等待。

3）暂停指令 HLT

该指令使 CPU 进入暂停状态。只有当 CPU 的复位输入端 RESET 有效、不可屏蔽中断产生请求、IF＝1 且可屏蔽中断产生请求 3 种情况之一发生时，CPU 才退出暂停状态。

4）封锁总线前缀指令 LOCK

它是一条总线锁定指令，可放在任何指令的前面，使得相应的指令在执行时，总线被锁定，以防止其他主设备使用总线。

5）交权指令 ESC

交权指令用于控制协处理器完成规定的功能。

汇编语言中常用的指令如表 3.4 所示。

表 3.4　常用指令表

分　类	指令助记符	功 能 简 介	指 令 格 式
数据传送类指令	MOV	传送指令	MOV 目标，源
	PUSH	进栈指令	PUSH 源
	POP	出栈指令	POP 目标
	XCHG	数据交换指令	XCHG 目标，源
	XLAT	换码指令	XLAT/XLAT table
	LEA	传送有效地址指令	LEA r16，mem
	LDS	传送数据段地址指令	LDS r16，m32
	LES	传送附加段地址指针指令	LES 目标，源
	LAHF	读取标志指令	LAHF
	SAHF	设置标志指令	SAHF
	PUSHF	对标志寄存器的值推入堆栈指令	PUSHF
	POPF	对标志寄存器的值弹出堆栈指令	POPF
	IN	输入指令	IN AL/AX，port
	OUT	输出指令	OUT AL/AX
算数运算类指令	ADD	基本加法指令	ADD 目标，源
	ADC	带进位的加法指令	ADC 目标，源
	INC	加 1 指令	INC 目标
	AAA	加法非压缩 BCD 码调整指令	AAA
	DAA	加法压缩 BCD 码调整指令	DAA
	SUB	基本减法指令	SUB 目标，源
	SBB	带借位的减法指令	SBB 目标，源
	CMP	比较指令	CMP 目标，源
	DEC	减 1 指令	DEC 目标
	NEG	求补指令	NEG 目标
	DAS	减法调整指令	DAS
	MUL	无符号数的乘法指令	MUL 源
	IMUL	带符号数的乘法指令	IMUL 源
	AAM	ASCII 码乘法调整指令	AAM
	DIV	无符号数的除法指令	DIV 源
	IDIV	带符号数的除法指令	IDIV 源
	AAD	ASCII 码除法调整指令	AAD
	CBW	字节扩展指令	
	CWD	字扩展指令	

续表

分　类	指令助记符	功 能 简 介		指 令 格 式
逻辑操作类指令	AND	逻辑与指令		AND 目标,源
	OR	逻辑或指令		OR 目标,源
	XOR	逻辑异或指令		XOR 目标,源
	NOT	求反指令		NOT 目标
	TEST	测试指令		TEST 目标,源
	SHL	逻辑左移指令		SHL 目标, 1/CL
	SHR	逻辑右移指令		SHR 目标,1/CL
	SAL	算数左移指令		SAL 目标,1/CL
	SAR	算数右移指令		SAR 目标,1/CL
	ROL/R	不带进位的循环左/右移指令		ROL/R 目标,1/CL
	RCL/R	带进位的循环左/右移指令		RCL/R 目标,1/CL
字符串类操作指令	MOVS	串传送指令		MOVS 目标,源
	LODS	取字符串数据指令		LODS 源
	STOS	存字符串数据指令		STOS 目标
	CMPS	比较字符串指令		CMPS 源,目标
	SCAS	扫描字符串指令		SCAS 目标
控制转移类指令	JMP	无条件转移指令		JMP 目标
	JG,JNLE,JGE,JNL,JL,JNGE,JLE,JNG,JZ,JE,JNZ,JNE,JA,JNAE,JAE,JNB,JB,JBE,JNA,JC,JNC,JO,JNO,JS,JNS,JP,JPE,JNP,JPO,JCXZ		条件转移指令	Jcc short-label
	LOOP	循环控制指令		LOOP short-label
	LOOPNZ/NE	非零计数循环指令		LOOPNZ/NE short-label
	LOOPZ/E	零计数循环指令		LOOPZ/E short-label
	CALL	子程序调用指令		CALL proc-name
	RET	子程序返回指令		RET/RET exp
	INT	中断指令		INT n
	IRET	中断返回指令		IRET
	INTO	溢出中断指令		INTO
处理器控制类指令	CLC	进位标志复位		CLC
	STC	进位标志置位		STC
	CMC	进位标志求反		CMC
	CLD	方向标志复位		CLD
	STD	方向标志置位		STD
	CLI	中断允许		CLI
	STI	禁止中断		STI
	NOP	空操作指令		NOP
	WAIT	等待指令		WAIT
	HLT	暂停指令		HLT
	LOCK	封锁总线前缀指令		LOCK
	ESC	交权指令		ESC ext-op,mem

3.3 汇编语言编程格式

3.3.1 汇编语言程序结构

先看一个简单的 8086 系统，有 X、Y 两个 16 位数 3456H 和 0ABCDH，求两数之和，并将结果送到 Z 单元。用汇编语言 ASM-86 编写的程序如下：

```
SSEG  SEGMENT
    Stack DB  100DUP(?)          ;定义堆栈段
SSEG  ENDS
DSEG  SEGMENT                    ;定义数据段
    X  DW,3456H
    Y  DW,0ABCDH
    Z  DW,0
DSEG  ENDS
CSEG  SEGMENT                    ;定义代码段
    ASSUME DS:DSEG,CS:CSEG,SS:SSEG
START:
    MOV  AX,DSEG                 ;指令程序开始
    MOV  DS,AX
    LEA  SI,X
    LEA  DI,Y
    MOV  X,[SI]
    ADD  AX,[DI]
    MOV  Z,AX
    MOV  AH,4CH
    INT  21H
CSEG  ENDS                       ;代码段结束
END  START                       ;程序结束
```

汇编语言程序有如下特点：

(1) 汇编语言程序通常由若干段组成，就好像一篇文章由若干段组成一样。8086 系统的存储器都是分段编址的，相应地，汇编语言的源程序通常也分段包括数据段、代码段、堆栈段和附加数据段，其中数据段和代码段不可缺少；段的数目依据需要而定，各段顺序任意，通常数据段在代码段之前定义，段由指示性语句 SEGMENT 和 ENDS 定义。程序代码段部分开始要设置段寄存器、初始化 DS。

(2) 每个段由若干语句组成。语句又分为指示性语句和指令性语句，分别由伪指令和指令构成，一条语句通常写成一行。

(3) 汇编语言程序中至少要有一个启动标号，作为程序执行时的入口地址。启动标号常用 START、BEGIN、MAIN 等命名。

(4) 汇编语言不区分字母大小写。

(5) 为增加源程序的可读性，可在分号“;”后面加上适当的注释。

3.3.2 汇编语言程序语句

汇编语言程序中的语句分别为指令性语句(指令语句)、指示性语句(伪指令语句)和宏

指令语句3类。指令性语句是可执行语句，由指令系统中的指令构成，在汇编中要产生对应的目标代码，CPU根据这些代码才能执行相应的操作。指示性语句是不可执行语句，是由伪指令构成的，汇编时不产生目标代码，用于指示汇编程序如何编译源程序，进行诸如定义数据、分配存储区、指示程序开始和结束等服务性工作。宏指令语句是用户按照宏定义格式编写的一段程序，其中，语句可以是指令、伪指令和已定义的宏指令。宏指令的作用主要是简化汇编程序。

1. 指令性汇编语句

指令性汇编语句的格式为：

[标号：]指令助记符[操作数],[操作数][；注释]

【例3-52】

```
START: MOV AX,DSEG      ；将段地址送入AX寄存器
```

指令性汇编语句格式是由4项组成，其中，中括号中的内容表示可选项。

(1) 标号：标号是程序设计人员自己定义的一个字符串符号，用于代表内存单元的地址，表示本条语句的符号地址。它不能与汇编语言中的保留字和CPU内部寄存器同名，也不允许由数字开头，字符数不得超过31个。指令性语句中标号后面紧跟“：”，如“START：”。

(2) 指令助记符：也称为操作码或操作符。对于指令性语句，汇编程序指令助记符翻译成机器语言。例如，指令助记符MOV相对应的机器码为10001110B。

(3) 操作数：操作数是操作符的操作对象。操作数可以是一个、两个、三个或一个都没有，多个操作数之间用逗号分开。操作数可以是寄存器、存储单元、常数、变量名、表达式等，通常利用操作数给出指令待处理数据的存放地址。如“AX,DSEG”。

(4)[；注释]：注释是以“；”开头的说明部分，用来说明一段程序、一条或几条指令在程序中的功能和作用，增加源程序的可读性。注释部分是语句的非执行部分，不会生成机器代码。如“；将段地址送入AX寄存器”。

说明：MOV传送指令是8086系统中使用频率最高的指令。该指令的作用是将一个字节、字或双字从源地址传送到目的地址中。其格式为：

MOV 目的操作数、源操作数

2. 指示性汇编语句

指示性汇编语句的格式为：

[标识符]伪指令定义符[操作数][；注释]

【例3-53】 ABC DB 100

(1) 标识符：同指令性语句中的标号一样，是程序设计人员自己定义的符号，用于代表内存单元的地址，或表示本条语句的符号地址。在指示性语句中，标识符后面不需要“：”，这一点与标号在指令性语句中的应用不同。例如，“ABC”为地址标号。

(2) 伪指令定义符：和指令助记符一样统称为操作码，对于指示性语句，只是给汇编程序提供一些控制信息，帮助汇编程序正确汇编指令性语句，在汇编程序执行时，没有对应的机器码。如“DB”。

3. 宏指令语句

1）宏指令定义

使用宏指令首先要对其进行定义，它的定义是用一组伪操作来实现的。其格式为：

```
宏指令名 MACRO [形式参数 1,形式参数 2,...]
...
宏指令名 ENDM
```

其中，MACRO 和 ENDM 是一对伪操作。这对伪操作之间是宏定义体，即一组具有独立功能的程序代码。宏指令名定义后可以像一般指令那样在程序中使用。宏指令名为宏定义的名字，不可省略，宏调用时要使用它，第一个符号必须是字母，其后可以是字母或数字。

形式参数是可选项，它对宏定义体中的可变部分进行说明，在汇编时形式参数由实际参数代替。

2）宏指令调用

宏指令定义后，可以像使用其他汇编语言指令一样使用宏指令。使用宏指令称为宏调用，其格式为：

```
宏指令名[实际参数 1,实际参数 2,...]
```

实际参数是与形式参数一一对应的，如果实际参数多于形式参数，则多余的实际参数无效；如果实际参数少于形参数，则多余的形式参数所对应的实际参数为空白。

3）局域符号定义

同一宏指令在源程序中往往出现多次，宏定义体中的符号地址如果不加以说明，则目标程序中就会出现多个重名的符号地址。因此宏定义体中的符号地址必须在宏定义中的开始加以说明。其格式为：

```
LOCAL 符号地址 1,符号地址 2,...
这些符号地址在汇编后，由汇编程序重新命名
```

4. 汇编语言程序操作数

操作数是汇编语言的组成部分，通常包括寄存器、存储单元、常量、变量、标号和表达式等。

1）常量

凡是出现在汇编源程序中的固定值，都可称为常量，如立即数。常量在程序运行期间不会变化，包括数字常量和字符串常量两种。

（1）数字常量。

- 二进制常量：以字母 B 结尾的数为二进制数，如 01111110B。
- 八进制常量：以字母 Q 结尾的数为八进制数，如 236Q 等。
- 十进制常量：以字母 D 结尾的数为十进制数，如 100D 等。
- 十六进制常量：以字母 H 结尾的数为十六进制数，如 0ABCH 等。

需要注意的是，在汇编源程序中，凡是以字母 A～F 开始的十六进制数，必须在前面加上数字 0，以免和标识符相混淆。

(2) 字符串常量。

以单引号内一个或多个 ASCII 字符构成常量,为字符串常量。汇编程序把它们翻译成对应的 ASCII 码值,一个字符对应一个字节。例如,字符串常量“AB”和“61237”在汇编时分别被翻译为 41 42H 和 36 31 32 33 37H。

2) 变量

变量是存放在存储器单元中的数据,它的数值在程序运行过程中随时可以被修改,通常以变量名的形式出现,可以认为是存放数据的存储单元符号地址。

变量在程序中作为存储器操作数来引用。我们可以用数据定义伪指令 DB、DW、DD 等来定义变量。每个变量都具有 3 种属性。

(1) 段属性(SEG):变量所在段的段基址。为了确保汇编程序能找到该变量,应在伪指令 ASSUME 中加以说明,并把变量所在逻辑段的段基址存放在段寄存器 CS、DS、ES 或 SS 中,其中 CS 段寄存器由 CPU 自动完成初值装入,SS 段定义如果有参数 STACK 时,也自动装入,否则像 DS、ES 一样需要用指令强制装入。

(2) 偏移量属性(OFFSET):变量所在存储单元的地址相对于变量所在段的起始地址之间的距离。

(3) 类型属性(TYPE):定义在变量存储区内每个数据所占内存单元的字节数,包括 BYTE(字节)、WORD(字)、DWORD(双字)、FWORD(6 字节)、QWORD(四字)及 TBYTE(10 字节)等。

注意:

① 变量需要事先定义才能使用。

② 变量类型应与指令要求的操作数类型相符。例如,“MOV BL,V1”指令要求 V1 应该是字节类型与 BL 类型匹配。“MOV BX,V2”指令要求 V2 应该定义成字类型的变量。

③ 变量定义后,变量名仅仅是对应这个数据区的首地址。若这个数据区中有若干个数据项,在对第 2 个数及其后面的数据项进行操作时,其地址应根据类型属性相应地在第 1 个数地址的基础上加 1、2、3、4、6、8 等。

3) 标号

标号是存放某条指令的存储单元的符号地址。通常它用作条件转移指令、无条件转移指令、循环指令和调用指令的目的操作数。

标号是由标识符后面跟一个冒号来定义的。例如,“START: MOV BX,1234H”这条语句定义了标号 START,它可以供转移指令、循环指令或调用指令当作目的操作数使用。

同变量名一样,标号作为存储单元的符号地址,它也具有 3 种属性:段、偏移量和类型。同变量的段、偏移属性一致,标号的段、偏移量属性也是指它的段基址和偏移地址,而标号的类型为 NEAR 和 FAR 两种。NEAR 类型的标号所在的语句和调用指令或转移指令在同一代码段中,即段内调用或转移。FAR 类型的标号所在的语句与其调用的指令或转移指令不在同一代码段中,即段间调用或转移。

4) 表达式

表达式是用一个运算符和一个或多个操作数组成的,从而汇编成一个值。常用的构成表达式的运算符有 5 种:算术运算符、逻辑运算符、关系运算符、分析运算符、综合运算符。

(1) 算术运算符。

算术运算符包括加(+)、减(-)、乘(×)、除(/)、取模运算符 MOD 以及 SHL(左移)、SHR(右移)。算术运算符都可以对数据进行运算,得到的结果也是数据;对地址的运算是在标号上加、减某一个数字量,例如,可用 START+2、MOVE-3 等表达式来表示一个存储单元的地址。

算术运算的格式为:

```
运算符  标号或变量
用 MOD 运算符取得的是两个数相除的余数,如(11 MOD 2 = 13 MOD 2 = 1)
```

【例 3-54】
```
MOV BH,11 MOD 2        ; (BH) = 1
= MOV BH,1
```

【例 3-55】
```
MOV AL,32 MOD 5        ; (AL) = 2
= MOV AL,2
```

SHL 左移运算符,相当于 $*2^n$,n 为左移次数

【例 3-56】
```
MOV BL,01001111B SHL 1; (BL) = 10011110B = 9EH
= MOV BL,9EH
```

SHR 右移操作符,相当于/2^n,n 为左移次数

【例 3-57】
```
MOV AL,10011110B SHR 2; (AL) = 00100111B = 27H
= MOV AL,27H
```

4) +、-、*、/运算符

【例 3-58】
```
MOV AL,(8 + 1) * 2      ; (8 + 1) * 2 = 18 = 12H,所以(AL) = 12H
```

(2) 逻辑运算符。

逻辑运算符包括与(AND)、或(OR)、非(NOT)和异或(XOR)运算。逻辑运算符只能对常数进行运算,比如,NOT OFFH=00,而 77H AND 84H=04H,得到的结果也是常数。

① 逻辑"与"AND 运算符。

【例 3-59】
```
MOV BH,11 AND 0FH; (BH) = 0BH
= MOV BH,0BH
```

【例 3-60】
```
AND AX,77HAND84H       ; 第 1 个 AND 为与指令,第 2 个 AND 为逻辑运算符
= AND AX,04H
```

【例 3-61】
```
AND AX,PORT AND 80H
```

说明:PORT AND 80H 中 AND 为逻辑运算符,若 PORT=81H,则逻辑运算符结果为 80H,第 1 个 AND 为指令助记符,此时将 AX 寄存器内容与 0FEH 进行运算,结果放在 AX 中。

② 逻辑"或"OR 运算符。

【例 3-62】
```
MOV BH,24 OR 0FH       ; (BH) = 2BH
= MOV BH,2FH,
```

③ 逻辑"异或"XOR 运算符。

【例 3-63】
```
MOV BH,24 XOR 0FH      ; (BH) = 2BH
= MOV BH,2BH
```

④ 逻辑“非”NOT 运算符。

【例 3-64】
```
MOV BH,NOT 24H          ; (BH) = 0DBH
= MOV BH,0DBH
```

(3) 关系运算符。

关系运算符有：相等 EQ、不等 NE、小于 LT、小于等于 LE、大于 GT、大于或等于 GE。关系运算符为数值型的，参与运算符的两个操作数必须都是数据，或者是同一段中的存储单元地址，而结果总是一个数值。关系运算符相当于逻辑判断式，如果关系式为真，则汇编结果为 0FFFFH 或 0FFH；如果关系式为假，则汇编结果为 0H。

【例 3-65】 若 X＝30，则有

```
MOV     AX,30 EQ 30          ; (AX) = 0FFFFH
MOV     AL,30 EQ 30          ; (AL) = 0FFH
MOV     AL,10 NE 30          ; (AL) = 0FFH
MOV     AX,10 LT 30          ; (AX) = 0FFFFH
MOV     AL,35 LE 30          ; (AL) = 0H
MOV     AL,24 GT X           ; (AL) = 0H
MOV     AL,40 GE X           ; (AL) = 0FFH
```

(4) 分析运算符。

分析运算符可以把一个存储单元地址分解为段地址和偏移量，并把分析运算后的数值回送到指令中的寄存器，因而也称为数值回送运算符，包括 TYPE、OFFSET、SEG、LENGTH 和 SIZE 5 种。

求变量和标号类型值——TYPE 运算符，其功能是求变量/标号的类型值。当运算符 TYPE 后为标号，程序回送该标号的类型值：NEAR 类型为－1，FAR 类型为－2。若运算符 TYPE 后为变量，则回送该变量的类型：类型 DB 为 1，DW 类型为 2，DD 类型为 4，DQ 类型为 8，DT 类型为 10。

【例 3-66】

```
KKK   DB20DUP(?)
TYPE  KKK                ; = 1
```

说明：DUP 为重复操作符，用于同样的操作数重复多次，其格式为：

```
n DUP(初值[,初值...])
```

小括号内为重复的操作数内容，n 为重复次数。如果初值不确定，可用“?”代替，如 DUP(?)。DUP 操作符可以嵌套，即在 DUP 小括号内还可用 DUP 操作符，如 DUP(2DUP(0EEH))。

取地址偏移量——OFFSET 的功能是返回标号或变量所在段的段内偏移地址(有效地址)。

【例 3-67】
```
MOV SI,OFFSET LABEL
```

这条指令与下面的“LEA SI,LABEL”指令效果相同，均将变量 DATA 的偏移地址送 SI 寄存器。

【例 3-68】 MOV DX,OFFSET LABEL

将标号 LABEL 处的偏移地址量取到 DX 中。

取段基地址——SEG 的功能是返回变量或标号的段基址。

【例 3-69】

```
MOV  AX,SEG DATA     ; 该指令将变量 DATA 的段基地址送 AX 寄存器
MOV  DX,AX
```

取变量单元个数——LENGTH 的功能是用来计算存储单元的数目。如果一个变量用重复操作符 DUP 来指明其单元个数,则利用该运算符可得到该变量的单元个数。如果未用 DUP,则得到的结果总是 1。

【例 3-70】

```
ABC  DW4421H
MOV  SI,LENGTH ABC       ; 指令执行结果,(SI) = 1
```

取变量字节数——SIZE 的功能是返回变量所占的字节总数,等于 LENGTH 与 TYPE 的乘积。

【例 3-71】

```
ABC  DW 50DUP(?)
MOV  AL,TYPE ABC     ; (AL) = 2
MOV  AH,LENGTH ABC   ; (AH) = 50
MOV  AX,SIZE ABC     ; (AX) = 100,SIZE 变量 = LENGTH 变量 * TYPE 变量
```

(5) 综合运算符。

综合运算符可以修改变量或标号的属性,也称属性运算符,包括 PTR、THIS、SHORT、HIGH、LOW 6 种。

PTR 运算符用于为操作数指定类型属性,赋予的新类型可以是 BYTE、WORD、DWORD、NEAR、FAR,它们只在当前指令内有效。其格式为:

```
<类型> PTR 标号或变量
```

【例 3-72】

```
MOV  AL,BYTE PTR ABC     ; 指定 ABC 标识的地址单元为字节类型
MOV  AX,WORD PTR[1000H] ; 指定有效地址[1000H]起的单元为字类型
MOV  BYTE PTR[1000H],0  ; 用 BYTE 和 PTR 规定[1000H]单元为字节单
                         ; 元,结果使偏移地址 1000H 字节单元清 0
```

和 PTR 类似,运算符 THIS 也可以用来改变存储区的类型或用来把后面指定的属性赋给当前的变量、标号或地址表达式。

【例 3-73】 希望存放数据的存储区 DATA1 既可以作为字节类型,也可以作为字类型来使用,则可用以下语句:

```
DATA2  EQU THIS WORD
DATA1  DB 100DUP(?)
```

相当于:

```
DATA1   DB 100DUP(?)
DATA2   EQU WORD TYPE DATA1
```

短转移运算符 SHORT 用来指定一个转移指令的目标地址属性为短转移，转移地址之间的距离在－128～＋127 个字符范围内。

分离运算符 HIGH 和 LOW 用来将其后的表达式分离出高字节和低字节。

【例 3-74】

```
MOV   AX,HIGH 0ABCDH        ; (AX) = 00ABH
MOV   AX,LOW 0ABCDH         ; (AX) = 00CDH
```

(6) 运算符的优先规则。

表 3.5 列出了运算符的优先级别。如果一个表达式中同时具有多个运算符，则按照以下规则进行运算：

① 表达式中小括号内的运算符总是在其任何相邻的运算之前进行；

② 优先级相同时，按表达式中从左到右的顺序运算；

③ 优先级不同时，先运算高优先级的，后运算低优先级的。

表 3.5　运算符优先规则

优先级	运算符(操作符)
高	括号中表达式
↓	LENGTH、SIZE
↓	段超越运算符 CS：、DS：、SS：、ES：
↓	PTR、THIS、OFFSET、SEG、TYPE
↓	＊、/、MOD、SHL、SHR
↓	EQ、NE、LT、LE、GT、GE
↓	NOT
↓	AND
↓	OR、XOR
低	SHORT

3.3.3　伪指令

伪指令可在汇编过程中完成数据定义、符号定义、存储区间分配、指示程序结束等管理控制功能，在汇编语言中构成指示性语句。伪指令不产生任何目标代码，只是用来指示汇编程序应该如何处理汇编语言程序，用来完成汇编的辅助性工作，如变量定义、符号赋值等。不同的汇编伪指令的符号意义往往会有差别，但多数是类似的。

汇编语言提供以下几类伪指令：符号定义伪指令、数据定义伪指令、段定义伪指令、过程定义伪指令、模块定义与连接伪指令、宏定义伪指令、条件定义伪指令、列表伪指令等。

8086 系统有 20 多种伪指令，本节主要讨论以下几种常用的伪指令：

(1) 符号定义伪指令 EQU、＝；

(2) 数据定义伪指令 DB、DW、DD；

(3) 存储单元类型伪指令 BYTE、WORD、DWORD；

(4) 段定义伪指令 SEGMENT、EDNS、ASSUME、ORG;

(5) 过程定义伪指令 PROC、ENDP、NEAR、FAR;

(6) 程序结束伪指令 END。

1. 符号定义伪指令

符号定义伪指令的用途是给一个符号命名,或定义类型属性等。它为程序的编写带来了许多方便。

1) 等值伪指令 EQU

等值伪指令 EQU 将表达式的值赋予一个符号名,定义以后可用这个符号名来代替表达式。表达式可以是一个常数、符号、数值表达式或地址表达式等。其格式为:

```
符号名  EQU  表达式
```

【例 3-75】

```
ABC     EQU 2009                ; 常数赋予符号名,即 ABC = 2009
COUNT   EQU 8 * (9 + 3)         ; 数值表达式赋予符号名,即 COUNT = 96
ADDR    EQU ES: [SI + 2]        ; 地址表达式赋予符号名,即 ADDR 为 ES: [SI + 2]
AD      EQU  ADD                ; 指令助记符赋予符号名,即 AD = ADD
JIA     EQU[SI + BX + 1000H]    ; JIA 值与偏移地址为 SI + BX + 1000H 的内存
```

注意:

(1) 在同一程序中,EQU 不允许对同一个符号重复定义。

(2) 解除对 EQU 赋值的伪指令为 PURGE,如 PURGE ABC。

2) 等号伪指令=

等号伪指令=的功能是用等号对一个符号赋值,并用该符号代替表达式。与 EQU 不同之处在于:等号伪指令可以对同一个符号重复定义,当用符号对同一符号重复定义时,以最后一次定义为准。其格式为:

```
名字  =  表达式
```

【例 3-76】

```
VAL = 123               ; 定义 VAL 为 24H
VAL = VAL + 1000H       ; 重定义 VAL 为 1024H
COUNT = VAL + 1         ; 定义 COUNT 为 1025H
```

2. 数据定义伪指令

数据定义伪指令用来定义一个存储单元符号地址(变量)及由该单元开始的若干连续存储单元的类型,同时还可给定义的存储单元赋初值。常见的数据定义伪指令有 DB、DW、DD、DQ、DT。其格式为:

```
[变量名]伪指令操作数[,操作数...]
```

其中,中括号中的参数为可选项;操作数可以是常数、表达式、字符串或问号"?",但每项操作数的值不能超过数据类型限定的范围。操作数可以不止一个,多个操作数之间用","号分开。数据定义伪指令的功能包括:定义类型、分配内存单元、赋值标号地址开始存放的连续单元。

DB 定义字节 BYTE 类型，每个操作数占有 1 个字节。

DW 定义字 WORD 类型，每个操作数占 1 个字，即 2 个字节，低位字节在低地址，高位字节在高地址。

DD 定义双字 DWORD 类型，每个操作数占 2 个字，即 4 个字节。

DQ 定义 8 字节 QWORD 类型，每个操作数占 4 个字，即 8 个字节。

DT 定义 10 字节 TBYTE 类型，每个操作数为 10 个字节的组合 BCD 码。

(1) 操作数为数值表达式。

【例 3-77】

```
DAR   DB 0DH              ; DAR 单元处放数值 0DH
CDR   DB 0AH              ; 单元 CDR 处放置 0AH
DAT   DB 50,50H           ; 变量 DAT 的内容为 32H(50),DAT + 1 单元放置 50H
CAR   DB 01A3H,4789H      ; 变量 CAR 内容为 A3H,CAR + 1 内存单元为 01H,
                          ; CAR + 2 内存单元为 89H,CAR + 3 内存单元为 47H
```

(2) 操作数为字符串表达式。

【例 3-78】

```
DAR   DB 'ABC'            ; 字符串引号括起来，自左向右以字符的 ASCII 码形
                          ; 式按地址递增排序，给每个字符分配一个字节单元
```

(3) 操作数为“?”和 DUP 表达式。

【例 3-79】

```
DA_1   DB ?               ; 用“?”表示 DA_1 单元中没有存放初值，在汇
                          ; 编过程中对应 DA_1 地址处留 1 个字节单元，用
                          ; 户可用于存放中间数据、标志、运算结果等
DA_1   DB ?,?             ; 表示要求汇编程序留出 2 个字节单元，即 DA_1
                          ; 和 DA_1 + 1
DA_1   DB 4DUP(?)         ; 表示汇编时保留 4 个字节单元，且每个字节可预
                          ; 置任何内容
```

【例 3-80】 操作数是数值表达式的数据定义伪指令如下：

```
VAR1   DB 'CD'
VAR2   DW -12H
VAR2   DD 12345678H
STR1   DB 4 DUP(01H)
```

说明：伪指令 ORG 通常用于源程序的第 1 条指令前，用来规定目标程序存放在存储单元的偏移量，即起始地址。若省略 ORG，则从本段起始地址连续存放。其格式为：

```
ORG  表达式
```

3. 段定义伪指令

存储器是分段管理的，常分为数据段、代码段、堆栈段、附加段。汇编语言源程序的所有指令、变量均分别放在各个逻辑段中。段定义伪指令的功能是在汇编语言程序中定义 4 个逻辑段的段名、段地址分配。段定义伪指令有 SEGMENT、ENDS、ASSUME。

(1) SEGMENT 和 ENDS。

其格式为：

```
段名 SEGMENT
  …
段名 ENDS
```

其中每个段都有一个名字——段名，段定义的段名在开始和结尾的名称必须一致。

SEGMENT 伪指令为段定义开始，用来定义一个逻辑段的特性。

ENDS 则表示一个逻辑段的结束。在汇编语言源程序中，SEGMENT 和 ENDS 伪指令总是成对出现，且段名一致。

SEGMENT 和 ENDS 伪指令之间的部分即是该逻辑段的内容，也称段体。对于数据段、堆栈段、附加段，段体是存储单元的定义、分配等伪操作；对于代码段，段体则是指令和部分伪操作。

【例 3-81】

```
DASEG SEGMENT
  ABC   DB 100DUP(?)
DASEG ENDS
```

(2) ASSUME。

ASSUME 伪指令在段定义之后，用于告诉汇编程序段寄存器与逻辑段之间的对应关系，即某一个段寄存器存放哪一个逻辑段的段基址。其格式为：

```
ASSUME 段寄存器: 段名[,段寄存器: 段名[,...]]
```

其中，对于 8086 CPU 来说，段寄存器名为 CS、DS、ES 或 SS；段名是 SEGMENT 定义过的段名。多个“段寄存器：段名”之间用“,”隔开。

使用 ASSUME 伪指令，仅仅确定了段名与段寄存器之间的关系，并不能把各个段的段基址装入相应的段寄存器中。DS、ES 和 SS 的装入可以通过给寄存器赋初值的指令来完成。CS 和 IP 的装入是 CPU 自动完成的。

【例 3-82】

```
DASEG SEGMENT
  ABC   DB 100DUP(?)
DASEG ENDS
CDSEG SEGMENT
  ASSUMECDSEG: CS,DASEG: DS
START:...
  …
CDSEG ENDS
ENDS   START
```

4. 过程定义伪指令

过程是汇编语言程序的一部分。在程序设计中，汇编程序可用“过程”来构造子程序，供主程序调用。过程定义伪指令包括 PROC、ENDP、NEAR、FAR，其格式如下：

```
过程名 PROC [NEAR/FAR]
  …
过程名 ENDP
```

其中，过程名是程序员自己定义的，不能是系统的保留字；PROC 定义一个过程，并指出该过程的类型属性为 NEAR 或 FAR，类型为 NEAR 的过程可以在段内被调用，类型为 FAR 的过程还可以被其他段调用，默认的类型是 NEAR；ENDP 标志过程的结束；PROC 和 ENDP 必须成对出现，且前面的过程名必须一致。

在一个过程中，可以有多条返回指令 RET。执行 RET 指令后，控制返回到原来调用指令的下一条指令。

当一个程序块被定义为过程后，程序可以使用 CALL 指令调用这个过程，或用转移指令转向一个过程。

5. 汇编结束伪指令 END

伪指令 END 标志着整个源程序的结束，它使汇编程序停止汇编操作，是汇编语言源程序中的最后一条指令。其格式为：

```
END 表达式(标号)
```

其中，表达式(标号)与源程序中的第一条可执行指令的标号相同。

3.3.4　上机过程

汇编语言程序实现过程如图 3.12 所示。

1. 编辑源程序

用字处理软件创建源程序。常用编辑工具有 EDIT、记事本、Word 等。无论采用何种编辑工具，生成的文件必须是纯文本文件，所有字符为半角。

2. 汇编

用汇编工具对上述源文件进行汇编，产生目标文件。汇编程序的主要功能是检查源程序的语法，并给出错误信息；产生目标程序文件；展开宏指令。

3. 连接

汇编产生的二进制目标文件并不是可执行的程序，还要用连接程序把它转换为可执行的 EXE 文件。

4. 程序运行

在建立了 EXE 文件后，只需在提示符下输入文件名即可运行程序。若程序能够运行但不能得到预期结果，则需要静态或动态查错。静态差错即检查源程序，并用文本编辑器进行修改，然后再汇编、连接、运行。

5. 程序调试及结果查看

有时静态检查不容易发现问题，尤其是碰到复杂的程序更是如此，这时就需要使用调试工具动态查错。当程序结果不能在屏幕显示时也需要使用调试工具查看结果。常用的动态调试工具为 DEBUG。

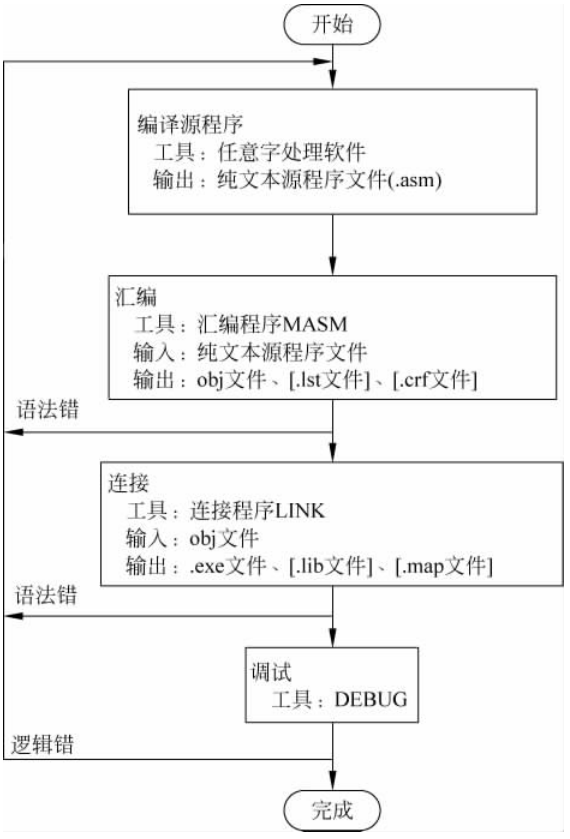

图 3.12　汇编语言程序实现流程图

3.4　汇编语言程序设计

3.4.1　顺序程序设计

1. 顺序程序的结构形式

顺序结构是最简单的一种程序结构。在程序流程图中表示为一个个处理框的串行连接，即一个语句紧跟一个语句，如图 3.13 所示。计算机执行顺序结构程序时，按指令书写的先后次序执行，从第一个语句开始顺序执行到最后一个语句，因而这种程序也称为直线程序或简单程序。顺序结构程序是最基本的程序结构，只有一个入口和一个出口，主要由数据传送指令、算术运算指令和逻辑运算指令组成。

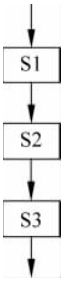

图 3.13　顺序结构流程图

2. 顺序程序设计

【例 3-83】 有 X、Y 两个 16 位数 3456H 和 0ABCDH，求两数之和，并将结果送到 Z 单元。或：X＋Y＝Z。

```
DSEG  SEGMENT
```

```
   X   DW 3456H
   Y   DW 0ABCDH
   Z   DW 0
DSEG   ENDS
CSEG   SEGMENT
   ASSUME DS: DSEG,CS: CSEG
START:
   MOV   AX,DSEG          ; 设置数据段地址
   MOV   DS,AX
   LEA   SI,X
   LEA   DI,Y
   MOV   AX,[SI]
   ADD   AX,[DI]
   MOV   Z,AX
   MOV   AX,4C00H
   INT   21H
CSEG   ENDS
END   START
```

3.4.2　分支程序设计

1. 分支程序的结构形式

分支程序设计是用一个条件判断语句，在执行时，根据判断结果来执行几个分支中的某个分支，分支结构又称为选择结构。分支程序结构可以有两种形式，如图 3.14 所示。它们分别相当于高级语言中的 IF-THEN-ELSE 语句和 CASE 语句，适用于要求根据不同条件做出不同处理的情况。IF-THEN-ELSE 语句可以引出两个分支，CASE 语句则可以引出多个分支，不论哪一种形式，它们的共同点是：运行方向是向前的，在某一种特定条件下，只能执行多个分支中的一个分支。

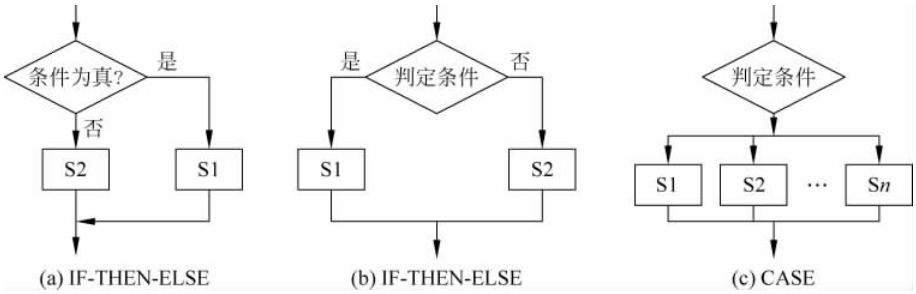

图 3.14　分支程序流程图

2. 分支程序的设计

汇编语言中分支程序设计的基本规律是利用某种操作来影响标志寄存器中相应标志位的状态，然后利用条件转移指令测试这些标志位的值，进而决定是否执行转移操作。通过对分支程序的两大要素的多次组合应用，可以实现任何复杂的分支程序。

【例 3-84】　计算函数值

$$Y=\begin{cases} X+1, & X>0 \\ X, & X=0 \\ X, & X<0 \end{cases}$$

源程序设计如下：

```
DSEG  SEGMENT
   X  DB?
   Y  DB?
DSEG  ENDS
CSEGS  EGMENT
  ASSUME CS: CSEG,DS: DSEG
START:
   MOV  AX,DSEG
   MOV  DS,AX
   LEA  SI,X
   MOV  AL,[SI]        ; 取 X 的值
   AND  AL,AL          ; AL 与自身相与,影响标志位
   JNS  LP1            ; SF = 0 转移到分支 LP1
   SUB  AL,1           ; SF = 1,X < 0
   MOV  Y,AL
   JMP  END1
LP1:
   JNZ  LP2            ; ZF = 0,则 X > 0 转移到 LP2
   MOV  Y,00H          ; ZF = 1,则 X = 0
   JMP  END1
LP2:
   ADD  AL,1
   MOV  Y,AL           ; 保存结果
END1:
   MOV  AH,4CH
   INT  21H
CSEG ENDS
END START
```

3.4.3 循环程序设计

1. 循环程序结构形式

在处理实际问题时，有时需要重复执行一组相同的操作，例如连加运算，需要进行若干次加法运算，程序冗长，如果采用循环程序设计则可以大大简化程序。循环程序结构根据条件是否满足，来决定一组相同语句是否重复执行，每执行一遍，都要对条件进行判断，直至条件不满足时，重复执行停止。循环结构程序又称迭代结构或重复程序结构，一般由 4 部分组成，即循环初始化部分、循环体、修改部分、控制部分。

（1）初始化部分：负责为循环体正常运行做好准备，完成循环初始值的设置，一般置于程序循环开始部分，且只执行一遍。初始化部分包括以下 3 个内容：

① 设置地址指针初始值。例如，初始化 SI、DI 地址指针，使它们分别指向源字符串和目的字符串的首地址。

② 设置控制部分循环结束条件的初始值。需要说明的是，如果循环结束条件是固定不变的，则可以不做这些工作。如果用重复次数作为结束条件，则需要把预定的循环次数送入相应的计数器 CX 中。

③ 设置寄存器和内存单元初始值。例如，使循环体用到的某些寄存器清 0，设置某些标

志位的状态或设置内存单元的初值等。

（2）循环工作部分：负责完成循环程序所要实现的功能，即需要重复进行的工作，它是循环程序的主体和核心。循环程序的工作部分可以是顺序结构、分支结构、循环结构。当工作部分是顺序结构或分支结构时，这样的程序称为单重循环程序；当工作部分是循环结构时，称这样的程序为多重循环程序。

（3）修改部分：负责配合循环工作部分工作，对参加运算的数据、数据的地址指针及结果单元的地址指针进行适当的修改，以保证每次循环时，参加运算的数据都是正确的，同时能正确地存放运算结果，为下一次的重复执行做准备。

（4）控制部分：保证循环程序按预定的循环次数或某种预定的条件正常循环，且能控制循环程序正常退出，是循环程序的关键部分。

根据循环程序结构组成中，循环工作部分与控制部分执行先后顺序的不同，循环程序有两种结构：DO-UNTIL 结构和 WHILE-DO 结构，如图 3.15 所示。

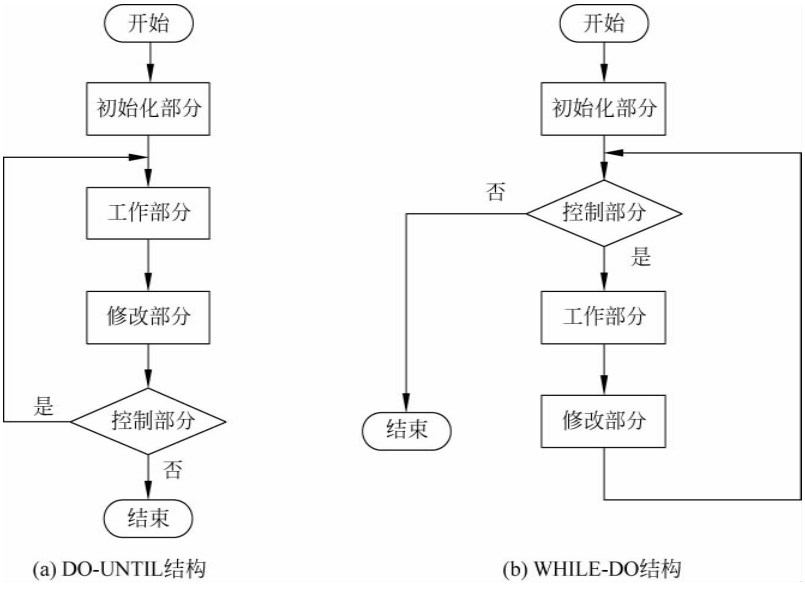

图 3.15　循环程序流程图

从图 3.15 中可以看出：DO-UNTIL 结构是先执行后判断，至少执行一次；而 WHILE-DO 结构是先判断后执行，有可能一次也没有执行。无论哪种结构，循环结构的初始化部分都只执行一次，而工作部分、修改部分、控制部分则有可能重复执行多次，通常把这 3 个部分合称为循环体。

2. 循环程序设计

在进行循环程序结构设计时，要注意下面 5 个问题：

（1）必须保证在循环程序的循环体内没有转向初始化部分的语句，否则会造成死循环，或者得不到预期结果。

（2）循环程序结构中的循环体可以是单个循环体，也可以是多个循环体。如果在循环程序的循环体中还包含一个或多个循环程序结构，那么就把这个程序称为双重或多重循环结构程序，这种情况也被称作循环嵌套。多重循环程序结构要比单重循环程序结构复杂得

多。通常设计多重循环时，应从外层循环到内层循环一层一层地进行。在设计外层循环时，先暂时把内层循环看成一个处理粗框，只有当把外层循环的初始化、工作、修改和控制 4 个部分设计完成后，再将粗框细化出内层循环的 4 个组成部分。

(3) 要合理设置循环控制条件，以保证在循环了一定次数后，能退出循环。

(4) 根据实际问题，选择结构 DO-UNTIL 或 WHILE-DO 结构。当循环次数可能为 0 时，必须用 WHILE-DO 结构。

(5) 选择好的算法并在循环体内尽量采用一些指令周期短、占用字节数少的指令，从而提高循环程序的执行效率。

【例 3-85】 求 1＋2＋3＋…＋50，并将结果送入 SUM 字单元中。

分析：求 1＋2＋3＋…＋50 的运算实际上就是要进行 49 次加法运算，但每次进行加法操作时的两个加数是不同的，它们的变化规律是一个加数每次都递增 1，而另一个加数则是上一次相加后的和，所以这种有规律的重复操作可以用循环结构实现。

源程序设计如下：

```
DATA  SEGMENT
   SUM  DW?
DATA  ENDS
CODE SEGMENT
   ASSUME CS: CODE,DS: DATA
START:
   MOV  AX,DATA
   MOV  DS,AX          ; 设置数据段寄存器
   MOV  CX,49          ; 设置循环次数赋给 CX 寄存器
   MOV  AX,1
   MOV  BX,2           ; 循环初始化
LP1:
   ADD  AX,BX          ; 工作部分
   INC  BX             ; 修改部分
   LOOP  LP1           ; 控制部分
   MOV  SUM,AX         ; 结果送入 SUM
   MOV  AH,4CH
   INT  21H
CODE ENDS
END START
```

3.4.4 子程序设计

子程序是指可被主程序多次调用的独立程序段，又称为过程，它相当于高级语言中的过程和函数。使子程序能够把具有类似的或相对独立的程序分离出来，简化源程序结构，使程序结构更加清晰，简化了程序设计和调试工作，极大地提高了编程效率。子程序一般包括子程序定义、现场保护、子程序调用、现场恢复和子程序返回 5 部分组成。

1. 子程序结构形式以及与主程序的关系

为了便于主程序调用，每个子程序都会有自己的程序名和相应的程序说明。程序名由过程定义伪指令定义的过程给出。子程序说明是为了使子程序便于阅读、维护、使用，为了

明确主程序和子程序之间的联系，明确子程序功能，应包含以下几项内容：

（1）子程序功能。用自然语言或数学语言等形式简单清楚地描述子程序完成的任务。

（2）子程序技术指标。

（3）子程序入口的符号地址。用过程定义伪指令定义该子程序的名字，这时子程序中第一条语句必须是子程序的入口指令；否则应写子程序入口指令的标号或地址。

（4）入口条件。说明子程序要求几个入口参数，这些参数表示的意义及存放位置。

（5）出口条件。说明子程序有几个输出参数，这些参数表示的含义及存放位置。

（6）受影响的寄存器。说明子程序运行后，哪些寄存器的内容被破坏了，以便使用者在调用该子程序之前注意保护现场。

子程序的结构与主程序类似，可以采用顺序、分支或循环程序结构。子程序由子程序定义伪指令定义，其格式为：

```
过程名　PROC　属性
  …
过程名　ENDP
```

其中，过程名是子程序入口的符号地址，与标号的作用相同。属性是指类型属性，包括NEAR和FAR。使用NEAR属性时，主程序和子程序在同一个代码段中；当主程序和子程序不在同一个代码段中时，应使用FAR属性。

主程序负责调用子程序，汇编语言中为子程序调用设置了专门的指令CALL，当子程序执行完时，返回到主程序调用的位置继续执行，实现“子程序返回”。同子程序调用指令一样，汇编语言中为子程序调用设置了专门的指令CALL，当子程序执行完时，返回到主程序调用的位置继续执行，实现“子程序返回”。同子程序调用指令一样，汇编语言中为子程序返回设置了专门的指令RET来实现。主程序与子程序之间的关系如图3.16所示。

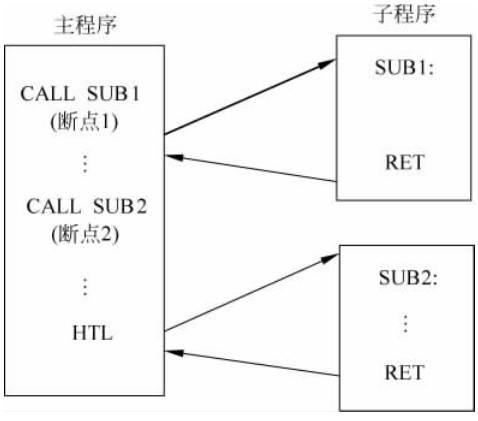

图3.16　子程序结构图

2. 子程序设计方法

在利用子程序的概念设计程序时，除了要进行子程序定义、调用、子程序返回主程序外，还应考虑现场的保护和恢复。

由于主程序和子程序通常都是分别编写的，在主程序调用子程序前，可能在寄存器中存

有一些有用的数据，而在子程序中需要使用这些寄存器，因而当子程序使用这些寄存器时，会将主程序中所需的数据破坏，造成冲突。这些寄存器的值或所需的标志位的值称为现场。显然，子程序执行前需要将现场保护起来，我们称之为保护现场；子程序执行结束返回主程序时需要将现场恢复，这个过程称为恢复现场。

保护现场与恢复现场的工作既可在调用程序中完成，也可在子程序中完成。在主程序中保护现场，则一定在主程序中恢复现场；在子程序中保护现场，则一定在子程序中恢复现场。这样可以增强主程序和子程序之间的相对独立性，减少相互依赖，使程序结构清楚，减少错误。

保护现场和恢复现场可以采用以下方法：

（1）利用堆栈。利用 PUSH 压栈指令，将寄存器内容或状态标识位内容保存在堆栈中，完成保护现场操作；利用 POP 出栈指令将已保存在堆栈中的现场取出恢复，完成恢复现场操作。这种方法是最为方便，尤其是在嵌套子程序设计应用中。

（2）利用内存单元。用传送指令将寄存器内容或状态标志内容保存在内存单元中，恢复时再用传送指令从内存单元中取出。这种方法使用不方便，因此较少采用。

3. 子程序的参数传递方法

主程序调用子程序时，需要把执行的结果送回主程序，这些数据称为入口参数；子程序完成后，要把执行的结果送给主程序，这些数据称为出口参数。常用的参数传递方法有 3 种：通过寄存器传递、通过堆栈传递和通过地址表传递。

（1）通过寄存器传递参数。用这种方法传递参数，是指调用程序和子程序之间，事先约定一些存放参数的通用寄存器，当调用程序转向子程序时，先把入口参数放到约定好的寄存器中，然后调用子程序，子程序工作时，直接从约定的寄存器中取出参数，当子程序处理完数据后，再将执行结果作为出口参数放入事先约定好的寄存器中。返回调用程序后，调用程序再从约定的寄存器中取结果。

（2）通过堆栈传递参数。用这种方法传递参数，是指主程序先把入口参数压入堆栈，然后调用子程序，子程序从堆栈中弹出入口参数进行处理；子程序处理完数据后再将执行结果作为出口参数压入堆栈，返回主程序后，主程序从堆栈中弹出出口参数进行相应的处理。

（3）通过地址表传递参数。当主程序与子程序之间传递的参数较多时，可以在主程序中建立一个地址表，把要传递给子程序的参数存放在地址表中，然后调用子程序，子程序通过地址表取得参数，并把结果存入指定的存储单元中。

【例 3-86】 采用子程序设计求 N2 程序。

解：这是一个简单的程序，子程序可采用顺序程序设计，该程序是一个完整的段定义、主程序、子程序、调用、返回，由于子程序和主程序所用的寄存器较少，所以没有现场保护和恢复。通过本例，读者可以体会到主程序和子程序之间的工作关系以及程序设计方法。

源程序设计如下：

```
DSEG   SEGMENT
   N   DB 21H
   RE DW?
DSEG ENDS
CSEG SEGMENT
```

```
    ASSUME CS: CSEG,DS: DSEG
START:
    MOV   AX,DSEG
    MOV   DS,AX
    MOV   AL,N
    CALL   SUB1
    MOV   RE,AX
    MOV   AH,4CH
    INT   21H
    SUB1   PROC
    MOV   BL,AL
    MUL   BL
    RET
    SUB1   ENDS
CSEG   ENDS
END   START
```

3.4.5　应用举例

1. 折半查找法

【例 3-87】 编写程序实现。在数据段中，有一个从小到大顺序排列的无符号数组，其首地址存放在 SI 寄存器中，数组中的第一个单元存放着数组长度。在 key 单元中有一个无符号数，要求在数组中查找是否存在[key]这个数，如果找到，则使 CF＝0，并在 DI 中给出该单元在数组中的偏移地址；如果未找到，则使 CF＝1。

分析：对于这个表格查找，可以使用顺序查找或折半查找的算法思想。顺序查找简单，效率不高；而折半查找程序复杂，效率高，但被查找数据必须是有序的。本例采用折半查找方式。

在一个长度为 n 的有序数组 r 中，查找元素 k 的折半查找算法可描述如下：

(1) 初始化被查找数组的尾下标，low＝1，high＝n。

(2) 若 low > high，则查找失败，CF＝1，退出程序；否则，计算中点：mid＝(low＋high)/2。

(3) k 与中点元素 r[mid]比较。若 $k=r$[mid]，则查找成功，结束程序；若 $k<r$[mid]，则转步骤(4)；若 $k>r$[mid]，则转步骤(5)。

(4) 低半部分查找(lower)，high＝mid－1，返回步骤(2)，继续执行。

(5) 高半部分查找(higher)，low＝mid＋1，返回步骤(2)，继续执行。

```
DSEG   SEGMENT
    ARRAY DB 13, 45, 49, 54, 66, 78, 83, 85, 89, 94, 99, 123, 233, 245
    KEY    DB 12
    CG1    DB 'cha zhao cheng gong', '$'
    ERROR1DB 'cha zhao shi bai', '$'
DSEG   ENDS
CSEG   SEGMENT
ASSUME CS: CSEG, DS: DSEG
START:
    MOV   AX,DSEG
```

```
    MOV   DS,AX
    MOV   AL,KEY
    LEA   SI,ARRAY
    MOV   BL,ARRAY
    MOV   BH,0
    INC   SI
MAIN:
    CMP   BX,0
    GL    ERROR
    SHR   BX,1
    CMP   AL,[BX+SI]
    JA    HIGHER
    JB    LOWER
    JMP   CG
HIGHER:
    ADD   SI,BX
    INC   SI
    JMP   MAIN
LOWER:
    ADD   SI,0
    DEC   BX
    JMP   MAIN
ERROR:
    LEA   DX,ERROR1
    MOV   AH,09H
    INT   21H
    CLC
    JMP   EXIT
CG:
    LEA   DX,CG1
    MOV   AH,09H
    INT   21H
    CLC
    JMP   EXIT
    LEA   DX,CG1
    MOV   AH,09H
    INT   21H
    STC
    MOV   DI,[BX+SI]
    JMP   EXIT
EXIT:
    MOV   AX,4C00H
    INT   21H
CSEG  ENDS
END   START
```

2. 冒泡排序法

【例 3-88】 存储器数据段从 BUF 开始存放一个字数组，数组中第一个字中存放该数组的长度 N，编制一个程序使此数组中的数据按照从小到大的次序排列。

分析：采用冒泡排序算法。从第一个数据开始，相邻的数进行比较，若次序不对，则两

数交换位置。第一遍比较 $N-1$ 次后，最大的数据已到了数组尾，第二遍仅需比较 $N-2$ 次就够了，共比较 $N-1$ 遍就完成了排序，这样共有两重循环。

源程序：

```
DATA  SEGMENT
   BUF  DW N,15,37,8600, A768H,3412H,1256H,76H
DATA  ENDS
STACK SEGMENT STACK 'STACK'
   SA DB 100DUP(?)
   TOPLABEL WORD
STACK ENDS
CODE SEGMENT
   ASSUMECS: CODE, DS: DATA, SS: STACK
MAIN: PROC FAR
START:
   MOV  AX,STACK
   MOV  SS,AX
   MOV  SP,OFFSET TOP
   PUSH DS
   SUB  AX,AX
   PUSH AX
   MOV  AX,DATA
   MOV  DS,AX
   MOV  BX,0
   MOV  CX,BUF[BX]
   DEC  CX
L1:
   MOV  DX,CX
L2:
   ADD  BX,2
   MOV  AX,BUF[BX]
   CMP  AX,BUF[BX+2]
   JBE  CONTI
   XCHG AX,BUF[BX+2]
   MOV  BUF[BX],AX
CONT1:
   LOOP L2
   MOV  CX,DX
   MOV  BX,0
   LOOP L1
   RET
MAIN  ENDP
CODE  ENDS
END  START
```

3. 功能调用

【例 3-89】 编写一个程序，它先接收一个字符串，然后显示其中数字字符的个数、英文字母的个数和字符串的长度。

分析： 先利用 0AH 号功能调用接收一个字符串，然后分别统计其中数字字符的个数和

英文字母的个数,最后用十进制数的形式显示它们。整个字符串的长度可从 0AH 号功能调用的出口参数中取得。

源程序:

```
MLENGTH = 128              ; 缓冲区长度
DSEG  SEGMENT              ; 数据段
   BUFF  DB MLENGTH        ; 符合 0AH 号功能调用所需的缓冲区
   DB ?                    ; 实际输入的字符数
   DB MLENGTH DUP (0)
   MESS0 DB 'Please input $ '
   MESS1 DB 'Length =  $ '
   MESS2 DB 'X =  $ '
   MESS3 DB 'Y =  $ '
DSEG  ENDS
CSEG  SEGMENT              ; 代码段
   ASSUMECS: CSEG,DS: DSEG
START:
   MOV  AX,DSEG
   MOV  DS,AX
   MOV  DX,OFFSET MESS0
   CALL DISPMESS
   MOV  DX,OFFSET BUFF
   MOV  AH,0AH
   INT  21H
   CALL NEW LINE
   MOV  BH,0
   MOV  BL,0
   MOV  CL,BUFF + 1
   MOV  CH,0
   JCXZ COK
   MOV  SI,OFFSET BUFF + 2
AGAIN:
   MOV  AL,[SI]
   INC  SI
   CMP  AL,'0'
   JB   NEXT
   CMP  AL,'9'
   JA   NODEC
   INC  BH
   JMP  SHORT NEXT
NODEC:
   OR   AL,20H
   CMP  AL,'a'
   JB   NEXT
   CMP  AL,'z'
   JA   NEXT
   INC  BL
NEXT:
   LOOP AGAIN
COK:
```

```
    MOV   DX,OFFSET MESS1
    CALL DISPMESS
    MOV   AL,BUFF + 1
    XOR   AH,AH
    CALL DISPAL
    CALL NEWLINE
    MOV   DX,OFFSET MESS2
    CALL DISPMESS
    MOV   AL,BH
    XOR   AH,AH
    CALL DISPAL
    CALL NEWLINE
    MOV   DX OFFSET MESS3
    CALL DISPMESS
    MOV   AL,BL
    XOR   AH,AH
    CALL DISPAL
    CALL NEWLINE
    MOV   AX,4C00H
   INT     21H
;-------------------------------
   ; 子程序名: DISPAL
   ; 功能: 用十进制数的形式显示 8 位二进制数
   ; 入口参数: AL = 8 位二进制数
   ; 出口参数: 无
;---------------------------------
   DISPAL PROC NEAR
   MOV   CX,3
   MOV   DL,10
   DISP1:
      DIV   DL
   XCHG AH,AL
   ADD   AL,'0'
   PUSH AX
   XCHG AH,AL
   MOV   AH,0
   LOOP DISP1
   MOV   CX,3
   DISP2:
      POP   DX
   CALL ECHOCH
   LOOP DISP2
   RET
   DISPAL   ENDP

   DISPMESS   PROC NEAR
   MOV   AH,9
   INT   21H
   RET
   DISPMESS   ENDP
;---------------------------------------
```

```
; 子程序名: ECHOCH
; 功能: 调用 DOS2 号功能,显示一个字符
; 入口参数: DL = 显示字符
; 出口参数: 无
; 说明: 通过显示回车符形成回车,通过显示换行符形成换行
; ----------------------------------------
  ECHOCH  PROC NEAR
  MOV  AH,2
  INT  21H
  RET
  ECHOCH ENDP
; ------------------------------------------------
; 子程序名: NEWLINE
; 功能: 形成回车或换行
; 入口参数: 无
; 出口参数: 无
; 说明: 通过显示回车符形成回车,通过显示换行符形成换行
; ------------------------------------------------
  NEWLINE  PROC NEAR
  PUSH AX
  PUSH DX
  MOV  DL,0DH
  MOV  AH,2
  INT  21H
  MOV  DL,0AH
  MOV  AH,2
  INT  21H
  POP  DX
  POP  AX
  RET
  NEWLINE  ENDP
  CSEG ENDS
  END  START
```

4. 分组统计

【例 3-90】 设有 4 个学生参加 5 门课程考试,试计算每个学生的平均成绩和每门课的平均成绩。

分析: 根据题意,我们把 4 名学生的成绩依次存放在一个字数组中,把每个学生的平均成绩和每门课的平均成绩保存在两个字数组中。

```
DATA  SEGMENT
   Chengji DW...
           DW...
           DW...
           DW...
   Aver    DW 4DUP(?)
   Mean    DW 5DUP(?)
DATA  ENDS
CODE  SEGMENT
ASSUME CS: CODE, DS: DATA
```

```
MAIN: PROC  FAR
START:
   PUSH  DS
   MOV   AX,0
   PUSH  AX
   MOV   AX,DATA
   MOV   DS,AX
   MOV   CX,4
   LEA   BX,Chengji
   LEA   SI,Aver
   SUB   BX,2
LOOP1:
   PUSH  CX
   MOV   AX,0
   MOV   CX,5
LOOP2:
   ADD   BX,2
   ADD   AX,[BX]
   LOOP  LOOP2
   MOV   DL,5
   DIV   DL
   MOV   [SI],AL
   ADD   SI,2
   POP   CX
   LOOP  LOOP1
   MOV   BX,OFFSET Chengji
   MOV   SI,OFFSET Mean
   MOV   DI,BX
   MOV   CX,5
LOOP3:
   MOV   BX,DI
   SUB   BX,10
   PUSH  CX
   MOV   AX,0
   MOV   CX,4
LOOP4:
   ADD   BX,10
   ADD   AX,[BX]
LOOP  LOOP4
   MOV   DL,4
   DIV   DL
   MOV   [SI],AL
   ADD   SI,2
   POP   CX
LOOP  LOOP3
   RET
   MAIN  ENDP
   CODE  ENDS
   ENDS  TART
```

习　题

1. 8086CPU 的寻址方式有哪些？

2. 8086CPU 的指令系统分哪几类？

3. 带位移量的基址变址寻址方式的有效地址如何计算？

4. 数据传送类指令分哪几类？各类指令有何特点？

5. 算术运算类指令分哪几类？

6. 逻辑运算类指令分哪几类？

7. 数据串操作类指令的特点是什么？

8. 在条件转移指令中，哪些适用于无符号数？哪些适用于带符号数？

9. 如果条件转移指令的转移目标超出了其所要求的转移范围时，该怎样处理？

10. 什么是堆栈？堆栈操作的经常用法是什么？使用时应注意什么？

11. 在什么条件下采用段内、段间调用？

12. 什么是中断？什么是中断向量？什么是中断向量表？

13. 写出 4 条指令，使它们分别把 AX 的内容清零。

14. 写出 5 条指令，使它们分别能检测累加器 AX 中的内容是否为零，而不改变 AX 中的原有内容。

15. 写出 6 条指令，使它们分别能清除进位标志位 CF。

16. 假定(DS)＝2000H，(ES)＝2100H，(SS)＝1500H，(SI)＝00A0H，(BX)＝0100H，(BP)＝0010H，数据变量 VAL 的偏移地址为 0050H，请指出下列指令的源操作数字段是什么寻址方式？它的物理地址是多少？

(1) MOV AX,0ABH　　(2) MOV AX,BX　　(3) MOV AX,[100H]

(4) MOV AX,VAL　　(5) MOV AX,[BX]　　(6) MOV AX,ES:[BX]

(7) MOV AX,[BP]　　(8) MOV AX,[SI]　　(9) MOV AX,[BX＋10]

(10) MOV AX,VAL[BX]　　(11) MOV AX,[BX][S]　　(12) MOV AX,VAL[BX][SI]

17. 设(DS)＝2000H，(BX)＝0100H，(SI)＝0002H，(20100)＝12H，(20101)＝34H，(20102)＝56H，(20103)＝78H，(21200)＝2AH，(21201)＝4CH，(21202)＝0B7H，(21203)＝65H，试说明下列各条指令执行完后 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]

18. 在 ARRAY 数组中依次存储了 7 个字数据，紧接着是名为 ZERO 的字单元，表示如下：

```
ARRAY  DW  23,36,2,100,32000,54,0
ZERO   DW  ?
```

如果 BX 包含数组 ARRAY 的初始地址，请编写指令将数据 0 传送给 ZERO 单元；

如果 BX 包含数据 0 在数组中的位移量，请编写指令将数据 0 传送给 ZERO 单元。

19. 指出下列指令的错误：

(1) MOV　AH,BX　　(2) MOV　[BX],[SI]

(3) MOV　AH,[SI][DI]　　(4) MOV　MYDAT[BX][SI],ES：AX

(5) MOV　BYTE PTR [BX],1000　　(6) MOV　BX,OFFSET MYDAT[SI]

(7) MOV　CS,AX　　(8) MOV　DS,BP

20. 执行下列指令后，AX 寄存器中的内容是什么？

```
TABLE  DW  10,20,30,40,50
ENTRY  DW  3
       ⋮
MOV    BX,OFFSET TABLE
ADD    BX,ENTRY
MOV    AX,[BX]
```

21. 写出执行以下计算的指令序列，其中 X、Y、Z、R 和 W 均为存放 16 位带符号数单元的地址。

```
Z←W+(Z-X)
Z←W-(X+6)-(R+9)
Z←(W*X)/(Y+6),R←余数
Z←(W-X)/(5*Y)*2,R←余数
```

22. 已知程序段如下：

```
MOV  AX,1234H
MOV  CL,4
ROL  AX,CL
DEC  AX
MOV  CX,4
MUL  CX
INT  20H
```

试问：

每条指令执行完后，AX 寄存器的内容是什么？

每条指令执行完后，SF、ZF、CF、OF 的值是什么？

程序结束时，AX 和 DX 的内容是什么？

23. 试编写一个程序求出双字长数的绝对值，双字长数在 A 和 A+2 单元中，结果存放在 B 和 B+2 单元中。

24. 假设(BX)=0E3H，变量 VALUE 中存放的内容为 70H，确定下列各条指令单独执行后的结果。

```
XOR  BX,VALUE
AND  BX,VALUE
OR   BX,VALUE
XOR  BX,0FFH
AND  BX,0
TEST BX,01H
```

25. 试分析下面的程序段完成什么功能。

```
MOV  CL,04
SHL  DX,CL
MOV  BL,AH
SHL  AX,CL
SHR  BL,CL
OR   DL,BL
```

26. 假设数据定义如下：

```
CONAME  DB  'SPACE  EXPLORERS INC.'
PRLINE  DB  20 DUP('?')
```

用串指令编写程序段分别完成以下功能：

从左到右把 CONAME 中的字符串传送到 PRLINE；

从右到左把 CONAME 中的字符串传送到 PRLINE；

把 CONAME 中的第 3 个和第 4 个字节装入 AX；

把 AX 寄存器的内容装入 PRLINE+5 开始的字节中；

检查 CONAME 字符串中有无空格字符，如有，则把它传送给 BH 寄存器。

27. 指令 CMP　AX,BX 后面跟着一条格式为 J…L1 的条件转移指令，其中…可以是 B、BE、NB、NBE、L、NL、LE 和 NLE 中的任一个，如果 AX 和 BX 的内容给定如下：

```
     AX    BX
(1) 1F52  1F52
(2) 88C9  88C9
(3) FF82  007E
(4) 58BA  020E
(5) FFC6  FF8B
(6) 09A0  1E97
(7) 8AEA  FC29
(8) D367  32A6
```

28. 假设 X 和 X+2 单元的内容为双精度数 P，Y 和 Y+2 单元的内容为双精度数 Q（X、Y 为低位字），试说明下列程序段做什么工作。

```
   MOV  DX, X + 2
   MOV  AX, X
   ADD  AX, Y
   ADC  DX, X + 2
   CMP  DX, Y + 2
   JL   L2
   JG   L1
   CMP  AX, Y
   JBE  L2
L1:
   MOV  AX,1
   INT  20H
L2:
```

```
MOV  AX, 2
INT  20H
```

29. 从内存 3000H 开始的单元中顺序存放着 40 个同学某门课的考试成绩，试编写程序段求该班该课程的总成绩和平均成绩。

30. 判断 STRING1 和 STRING2 两个等长字符串是否相等，如相等，则在 RESULT 单元置 1，否则置 0。