宏汇编语言程序设计编程指导

毛明 编著

机械工业出版社

宏汇编语言程序设计编程指导

毛 明 编著

机械工业出版社

（京）新登字 054 号

本书主要讨论宏汇编语言程序设计的方法和技巧。它不同于一般的理论性的程序设计教程，而是一本实用的编程指导教程。全书以理论为先导，以大量的程序设计实例为主体，目的在于帮助初学 IBM PC 8088、80286 和 80386 宏汇编语言程序设计的计算机专业人员和大专院校计算机软件专业学生，系统、全面地学习宏汇编语言程序设计的编程方法和技巧。

本书适合于从事微型计算机系统开发和应用的软、硬件技术人员学习和参考，特别适合于大专院校计算机软件专业学生的学习和参考。

宏汇编语言程序设计编程指导

毛　明　编著

*

责任编辑：王中玉　　责任校对：肖新民

盛君豪

封面设计：姚　毅　　版式设计：朱淑珍

*

机械工业出版社出版（北京阜成门外百万庄南街一号）

邮政编码：100037

（北京市书刊出版业营业许可证出字第 117 号）

北京通县财联印刷厂印刷

新华书店北京发行所发行・新华书店经售

*

开本 787×1092 1/16・印张 15 3/4・字数 384千字

1993年 7 月北京第 1 版・1993年 7 月北京第 1 次印刷

印数 0 001—4 000・定价：13.50元

*

ISBN 7-111-03776-6/TP・186

前　言

由于 IBM PC、286、386 以及长城 0520 系列微机的普及和广泛应用，PC 系列及长城 0520 系列微机的高级开发和应用技术已成为微型计算机软、硬件技术人员必须学习和掌握的重要内容。以 IBM PC Intel 8088 为核心的宏汇编语言程序设计已成为高等院校计算机软件专业学生必修的专业课程之一，同时也是各行各业中广大计算机软、硬件技术人员从事微型计算机系统开发和应用的基本工具。由于汇编语言是面向机器的语言，它是机器语言的符号化表示，其程序设计与计算机系统的内部结构有着密切的联系，它不象高级语言那样接近于自然语言，容易学习和掌握，往往使初学者感到高深莫测，不易学习和掌握。

为了帮助高等院校计算机软件专业学生和广大计算机软、硬件技术人员系统而全面地掌握宏汇编语言程序设计的编程方法和技巧，作者在总结多年来从事宏汇编语言程序设计的教学和科研工作经验的基础上，编写了此书，奉献给广大读者。本书从 Intel 8088 基本指令入手，按照循序渐近、由易到难、由简单到复杂、由初级到高级的顺序，全面介绍了宏汇编语言程序设计的各种编程应用技术。全书以简明的基础理论为先导，以大量的程序设计实例为主体，着重于讲述程序设计的方法和技巧。

本书各章节的内容是这样安排的：第一章概括地介绍了宏汇编语言程序设计的基本知识。其主要内容是 Intel8088、80286 和 80386 CPU 的寄存器和存储器结构、指令系统、汇编语言语句的种类及格式、源程序的一般结构以及汇编语言程序的上机过程等。第二章是基本指令的编程实例。该章按照 CPU 指令系统的分类结构，列举了许多典型而简单的程序，分别讲述了每一类指令的一般用法、基本编程方法和技巧。第三章是程序设计的基本方法。其主要内容是讨论程序设计的三种逻辑结构，即顺序结构、分支结构和循环结构的程序设计方法。第四章主要讨论结构程序设计方法 —— 宏指令、子程序、模块化程序设计和参数传递的方法。结构程序设计可以使程序设计规范化，结构化的程序既便于阅读，又便于程序维护，是大型程序设计的重要技术。第五章主要讨论数据处理的程序设计问题。数据处理主要包括十进制算术运算、代码转换、表数据的查找、插入、删除和排序等内容。第六章主要讨论如何使用 BIOS 中断功能调用进行程序设计的问题。着重讨论了绘图与动画程序设计的基本方法并给出了程序设计实例。第七章主要讨论如何利用系统功能调用进行程序设计的问题。系统功能调用是汇编语言程序设计中的一个非常重要的组成部分，在应用程序完成 I/O 设备管理、磁盘文件管理、内存管理工作时，利用系统功能调用完成这些工作是必不可少的。第八章讨论了常用的 I/O 端口的编程应用问题。着重介绍了使计算机发声和演奏音乐的编程方法。第九章介绍了高档微机开发和综合应用的一些实例。该章的主要目的在于通过一些综合性的程序设计实例，说明汇编语言程序设计在计算机高级开发技术和应用方面的重要性和实用性。

本书的特点在于实用性强，书中所有的实例均经作者在 IBM PC、286、386 和长城 0520 系列微机上调试通过，初学者既可以从书中学到程序设计的基本方法，又可以遵循实例上机练习。将理论与实际有机地结合在一起，必然使学习效果倍增。作者衷心地希望本书能给广大

读者许多有益的启示和帮助。

在本书编写过程中，得到了北京电子科技学院计算机系系主任王贵和副教授的热情关心和帮助，本书部分章节内容的安排以及许多程序设计的方法和技巧都与王贵和副教授多年来的悉心指导是分不开的，作者在此深表谢意。孔德祥副教授、张立同志、方勇同志在本书的编写过程中，也给予了很多关心和帮助，在此一并表示感谢。

由于水平所限，时间仓促，书中缺点、错误在所难免，敬请各位读者批评指正。

毛　明

1992 年 5 月

目　录

第一章 基 本 知 识

本章概括地介绍了 8088、80286、80386 等微处理器的 CPU 结构、寄存器结构以及指令系统，重点介绍了 8088 的存储器结构、堆栈原理、8088 的寻址方式以及指令系统，其主要目的是帮助初学者从整体上全面地回顾一下 Intel 系列微处理器的性能及特点，了解这些内容是进行汇编语言程序设计的基础。另外，本章还介绍了汇编语言语句的种类及格式、源程序的一般结构以及汇编语言的上机过程等内容。

第一节 8088 CPU 结构及指令系统

一、8088 CPU 结构

8088 CPU 主要由两大部分组成，即总线接口单元(BIU)和执行单元(EU)。BIU 负责 CPU 与存储器之间的信息传送，而 EU 则负责指令的执行，以及进行算术、逻辑运算。8088 是 IBM PC 及其兼容机上广泛采用的微处理器。

Intel 公司的 8086 芯片与 8088 芯片都是 16 位的微处理器，两者的不同之处在于 8088 具有 8 位的数据通道，而 8086 则具有 16 位的数据通道，除此之外，两者的其它方面都是相同的。本书列举的所有程序都可以不加修改地在 8086 CPU 上运行。

二、8088 CPU 寄存器

(一) 8088 CPU 寄存器的组成

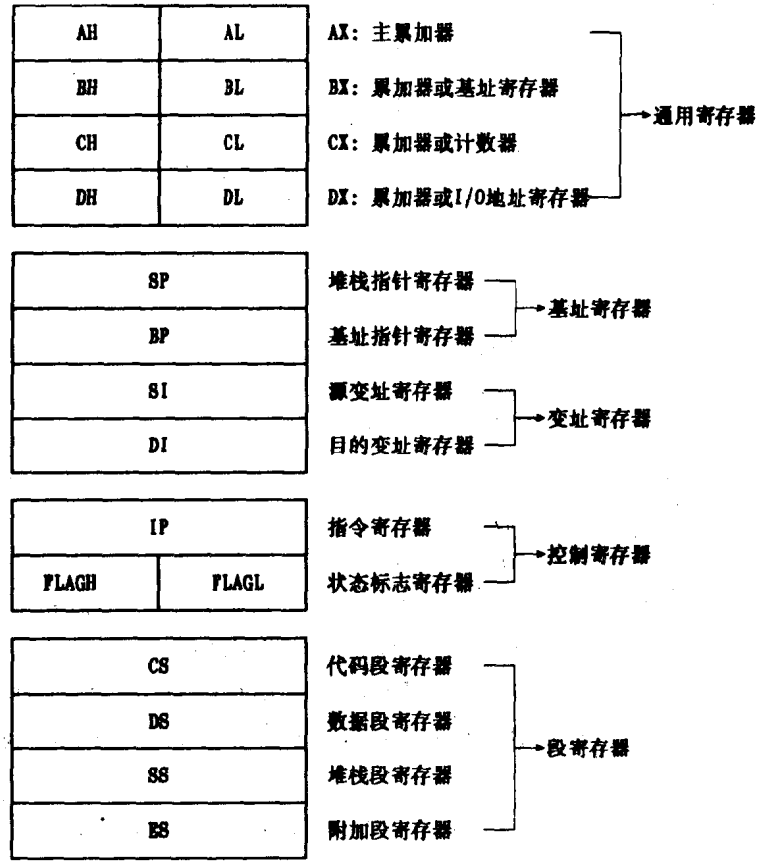


图 1-1 8088 寄存器结构

8088 CPU 共有 14 个 16 位寄存器，分为 4 组，它们的名称、结构及功能如图 1-1 所示。

通用寄存器用来存放 8 位或 16 位算术/逻辑操作数。指针寄存器用于访问堆栈中的数据，其中 SP 寄存器称为堆栈指针，它始终指向堆栈的栈顶；而 BP 则是用于堆栈中一般数据的存取。变址寄存器主要用字符串的操作。IP 指针始终指向下一条将要执行的指令。段寄存器主

要进行存储器的分段，以便使 8088 在 1MB 内存范围中寻址。

（二）8088 状态标志寄存器

8088 标志寄存器为 16 位寄存器，其中只有 9 个标志位。这 9 个标志位是按位进行定义，用以反映系统运行的状态以及运算结果的特点。9 个标志位一般被划分为两大类型：状态标志和控制标志，如图 1－2 所示。

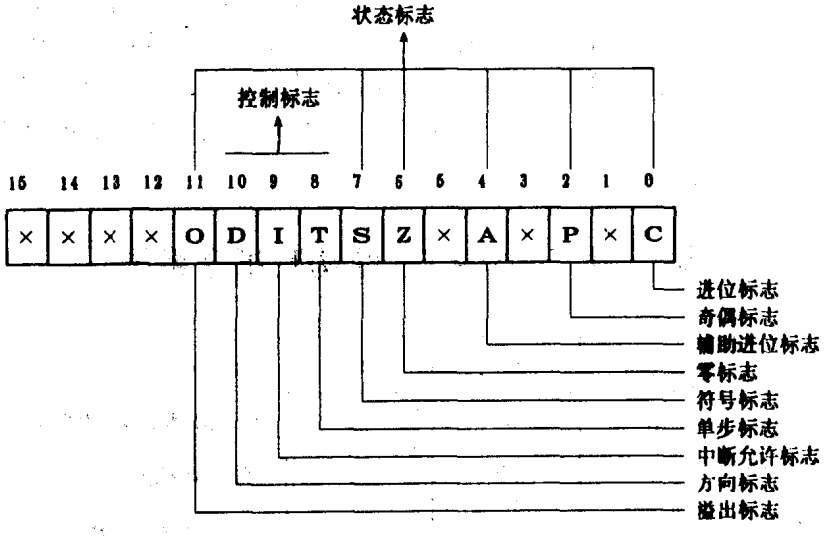


图 1－2　8088 状态寄存器结构

状态标志通常是在进行算术或逻辑运算之后，由系统根据运算结果的特点进行设置，以反映这种操作结果的某种性质。控制标志可以由指令进行设置，使处理器按照用户的意图进行操作。下面是各个标志位的具体含义。

1. 状态标志位

（1）进位标志 CF：若结果的最高位产生一个进位（加法）或一个借位（减法），则 CF＝1，否则 CF＝0。

（2）辅助进位 AF：若一字节中第 4 位上产生一个进位或借位，则 AF＝1，否则 AF＝0。

（3）奇偶标志 PF：若结果为偶数个 1，则 PF＝1，否则 PF＝0。

（4）零 标 志 ZF：若运算的结果为零，则 ZF＝1，否则 ZF＝0。

（5）符号标志 SF：若结果的最高位为 1，则 SF＝1，否则 SF＝0。

（6）溢出标志 OF：若带符号数算术运算的结果产生溢出，则 OF＝1，否则 OF＝0。

2. 控制标志位

（1）方向标志 DF：若 DF＝1，则数据串操作指令将按地址递减的方式进行；若 DF＝0，则按递增的方式进行。

（2）中断标志 IF：若 IF＝1，则 CPU 允许接受外部可屏蔽的中断请求，即开中断；反之 IF＝0，为关中断。

（3）陷井标志 TF：若 TF＝1，则处理器处于单步方式。在此方式下，每执行完一条指令自动产生一个软件陷井，执行单步中断。反之，处理器为正常工作方式。

三、8088 存储器结构

8088 有 20 位的地址总线，可以寻址 1MB 内存单元。但是，由于 8088 CPU 都是 16 的寄存器，CPU 内部的运算也只能是 16 位的，这样 8088 只能在 64KB 范围内寻找操作数。为了充分利用 20 位的地址总线以及寻址 1MB 内存单元，8088 采取了分段结构。寻址采用了分段结构后，一个段可以寻址 64KB 范围，改变段值，可以寻址不同的 64KB 范围，从而可以寻址 1MB 范围。在实际寻址时，一个段寄存器会自动被选择，并且左移 4 位与相应的寄存器中的 16 位偏移量相加形成真正的 20 位物理地址。图 1－3 表明了存储器的分段结构，图 1－4 表明了物理地址的形成过程。

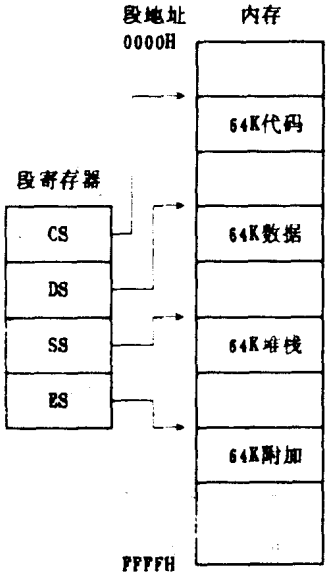


图 1－3　8088 存储器的分段结构

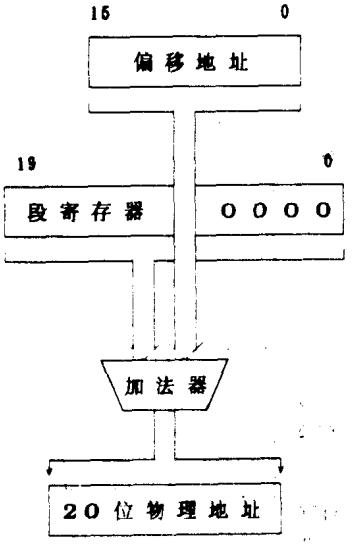


图 1－4　8088 物理地址形成过程

一般情况下，一个操作数寄存器与一个段寄存器之间都有一个缺省的组合关系，特殊情况下，可以更换段寄存器，表 1－1 列出了段更换的范围。

表 1－1　段缺省与段更换的约定

操作数寄存器	缺省段寄存器	可更换的段寄存器
IP(指令指针)	CS	无
SP(堆栈指针)	SS	无
BP(基址指针)	SS	CS、DS 或 ES
BX、SI、DI(非字符串操作)	DS	CS、ES 或 SS
SI(源串指针)	DS	ES、SS 或 CS
DI(目标串指针)	ES	无

四、堆栈和堆栈指针

堆栈是在存储器中开辟的一端固定、一端活动的，以先进后出为原则进行数据存取的数据结构。其固定端叫栈底，活动端叫栈顶，栈底一般位于高地址端，栈顶则位于低地址端。数据存

入堆栈叫做压栈(PUSH);从堆栈中取一数据叫做弹栈(POP),为了按照先进后出的原则存取堆栈中的数据,堆栈设有一指针 SP,它始终指向堆栈的栈顶。压栈和弹栈都是就 SP 的当前位置而进行的。压栈和弹栈都以字为单位。堆栈结构如图 1-5a。

堆栈指针的变化原则:

1. 压栈过程

在数据压栈时,SP 指针减 2,数据被压入 SP 指向的存储单元,低字节在低地址,高字节在高地址,堆栈空间减小。如图 1-5b 所示。

2. 弹栈过程

在数据弹栈时,数据首先被取出放入相应的寄存器,SP 指针加 2,堆栈空间扩大。如图 1-5c 所示。

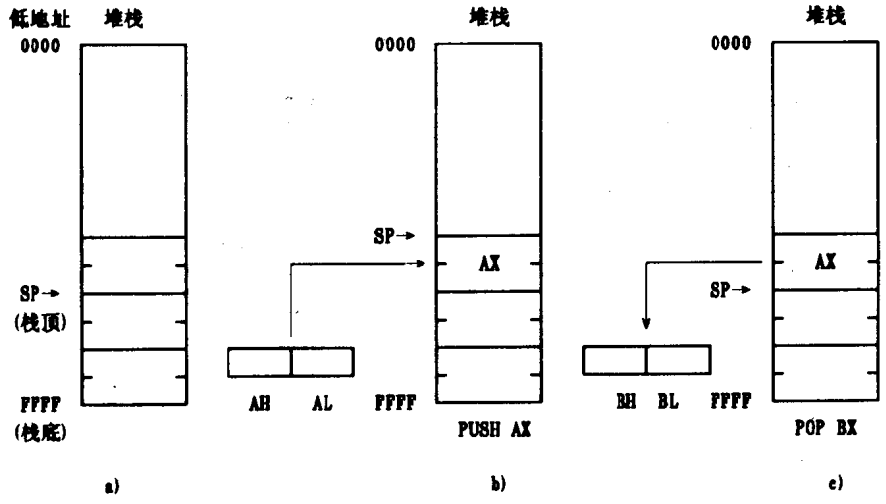


图 1-5 堆栈变化过程示意图

五、8088 的寻址方式

所谓寻址方式即寻找指令中操作数地址的方式。8088 CUP 采用了灵活多样的寻址方式,其中常见的寻址方式有下面几种:

1. 隐含寻址

指令中不指明操作数,而是隐含对规定目标的操作。

例如:DAA

其含义是对寄存器 AL 进行十进制数调整,其结果仍在 AL 中。

2. 立即寻址

指令中直接给出了常数操作数,这个常数可以是字节常数,也可以是字常数。

例如:MOV AX,1000

其含义是把常数 1000 送入 AX 寄存器。

3. 寄存器直接寻址

指令中的源操作数或目的操作数放在某一寄存器之中。

例如:INC SI

其含义是把 SI 寄存器中的数加 1。

4. 寄存器间接寻址

在指令中的寄存器包含操作数的地址,操作数本身在存储器中。

例如:ADD AX,[BX]

它的含义是把 BX 寄存器中的数据作为存储单元的地址,将该地址单元中的数取出来与 AX 相加,其结果放在 AX 中。

可以进行寄存器间接寻址的寄存器有 BX、BP、SI、DI。

5. 直接寻址

在指令中直接给出操作数的单元地址。

例如:AND AX,DS:[2000H]

其含义是把当前数据段偏移 2000H 地址中的数,与 AX 寄存器中的数进行逻辑与操作,其结果放在 AX 中。

6. 变址寻址

这种寻址方式是把变址寄存器中的数加上一个偏移量形成操作数的地址,使用的寄存器是 SI 或 DI。

例如:MOV AX,BASE[SI]

其含义是把 SI 的内容加上 BASE(常数)得到的数作为操作数的地址,将该地址单元中的数取出来放在 AX 中。

7. 基址寻址

这种寻址方式是把基址寄存器中的数加上一个偏移量形成操作数的地址,使用的寄存器是 BX 或 BP。

例如:SUB AX,[BX+100]

其含义是以 BX 寄存器的内容加上偏移量 100 得到的数为操作数地址,将 AX 减去该地址中的数,结果放在 AX 中。

8. 基址加变址寻址

这种寻址方式是把一个变址寄存器的内容,加上一个基址寄存器中的内容,再加上指令中指定的一个偏移量形成操作数的地址。

例如:MOV AX,BASE[BX][SI]

该指令的含义是把 BX 内容加上 SI 的内容,再加上 BASE 的值形成操作数的地址,将该地址中的数送入 AX 中。

9. 输入/输出端口寻址

端口寻址分为直接寻址和间接寻址两种。在直接端口寻址方式中,端口地址直接由指令提供一个 8 位的立即数,这种寻址方式只能寻址 0～255 个端口。

例如:IN AL,63H

其含义是从 63H 端口中读一字节数据送 AL。

在间接端口寻址方式中,被寻址的端口地址放在 DX 寄存器中。这样,这种寻址方式能寻址 64KB 的端口地址。

例如:OUT DX,AL

其含义是把 AL 中的一字节数送入 DX 指明的端口。

六、8088 指令系统

指令是计算机完成特定功能的最小的执行单位，又是程序设计中的最小的组成部分。熟练地掌握各种指令的格式、功能及其用法是编程的关键所在。

对于一条机器指令来说，一般包括两大部分内容，一是指令操作码，它表明指令的操作性质；另一则是指令操作数，它给出操作的对象。8088 CUP 采用了较为灵活的指令格式，一条指令的机器码由 1 到 6 个字节组成，具体指令的长度是随指令的种类及寻址方式的不同而不同的，有单字节指令、双字节指令，也有三字节指令、四字节指令甚至六字节指令。

8088 CPU 共有 110 多条指令，它们构成了 8088 的指令系统。这些指令按其功能可以划分为如下六组：

(1) 数据传送指令；

(2) 算术运算指令；

(3) 逻辑指令；

(4) 串操作指令；

(5) 程序控制指令；

(6) 处理器控制指令。

(一) 数据传送指令

数据传送是汇编程序设计中经常进行的工作之一。频繁的数据传送往往进行在寄存器与寄存器、寄存器与存储器以及存储器与存储器之间。传送的内容有数据，有地址，还有各种标志。8088 基本数据传送指令分为如下四类：

1. 通用数据传送指令

例如，数据传送指令(MOV)，压栈指令(PUSH)等。

2. I/O 端口操作指令

例如，端口输入指令(IN)，端口输出指令(OUT)等。

3. 地址传送指令

例如，传送偏移地址的指令(LEA)，传送双字地址的指令(LDS/LES)等。

4. 标志传送指令

例如，标志进栈指令(PUSHF)，标志弹栈指令(POPF)等。

(二) 算术运算指令

在 8088 指令系统中，提供了加、减、乘、除四则运算指令 20 条，利用这些指令不仅可以进行无符号数的运算，还可以进行有符号数的运算；不仅可以进行字节运算，还可以进行字运算；不仅可以进行二进制数的运算，还可以进行十进制(BCD 码)的运算。但是，所有这些运算指令仅限于对整数的运算，不能进行浮点数(实数)的运算。

1. 加法指令

例如，不带进位的加法指令(ADD)，带进位的加法指令(ADC)，加 1 指令(INC)等。

2. 减法指令

例如，不带进位的减法指令(SUB)，带进位的减法指令(SBB)，比较指令(CMP)等。

3. 乘法指令

例如，无符号数乘法指令(MUL)，带符号数乘法指令(IMUL)等。

4. 除法指令

例如，无符号数除法指令(DIV)，带符号数除法指令(IDIV)等。

(三) 逻辑指令

逻辑指令是可以对二进制位(Bit)进行操作的指令。在 8088 指令系统中，逻辑指令按其功能可以划分为如下三个类型：

1. 逻辑运算指令

例如，逻辑与运算(AND)，逻辑或运算(OR)，测试指令(TEST)等。

2. 移位指令

例如，逻辑左移/算术左移指令(SHL/SAL)，逻辑右移指令(SHR)等。

3. 循环移位指令

例如，不带进位的左循环移位指令(ROL)，带进位的左循环移位指令(RCL)等。

(四) 串操作指令

串操作指令是可以对字符串进行传送、比较、搜索、输入/输出的指令。

例如，字符串传送指令(MOVS)，字符串比较指令(CMPS)，字符串搜索指令(SCAS)等。

(五) 程序控制指令

程序控制指令是在程序运行中用以控制程序分支以及循环的指令，可分为如下几类：

1. 转移指令

(1) 无条件转移指令：例如，无条件转移指令(JMP)，过程调用指令(CALL)，过程返回指令(RET)等。

(2) 条件转移指令：条件转移指令是只有在条件满足时，才转移到目标地址的指令。如果条件不满足，程序将按顺序继续执行。

例如，高于/不低于也不等于转移指令(JA/JNBE)，有进位(或借位)转移指令(JC)，CX 寄存器等于 0 转移指令(JCXZ)等。

2. 迭代控制指令

迭代控制指令一般用在循环程序中循环体的前面或后面，根据条件的满足与否决定循环体是否继续执行。所有的迭代控制指令都以 CX 为循环计数器。

例如，LOOP 指令(CX－1→CX，若 CX≠0 转)，LOOPE(CX－1→CX，若 CX≠0 并且 ZF＝1 转)等。

3. 中断指令

中断指令是一类特殊的无条件转移指令，该类指令在执行时，首先把标志寄存器压栈，然后清除 TF 和 IF 标志，通过中断向量表查得相应的向量地址(段地址和偏移地址)进行直接调用，以达到控制转移。具体执行过程是：堆栈指针减去 2，把标志寄存器压栈。复位 TF 和 IF 标志位。SP 指针再减 2 并且把 CS 寄存器的当前值压入堆栈，然后用中断向量地址的高字取代 CS 原来的内容。SP 指针再减 2 并把指令指针 IP 的当前值压入堆栈，用中断向量的低字取代原指令指针 IP 的内容，这样就使得程序转移到中断例程中去执行。在中断例程执行完成后，IRET 指令恢复保存的返回地址及标志寄存器内容，从而使处理器继续执行主程序中的中断指令的下一条指令。

INT n 中断调用指令

IRET 中断返回指令

两个特殊的中断指令：

(1) INTO　溢出中断；

(2) INT 3　断点中断 。

(六) 处理器控制指令

处理器控制指令包括对标志位操作和直接对 CPU 进行控制的指令。它可以使 CPU 按照用户的需要进行工作。

1. 标志操作指令

例如，清进位标志(CLC)，清方向标志(CLD)，置方向标志(STD)等。

2. 处理器暂停指令

使处理器进入暂时停止状态(HLT)，这个暂停状态可以用外部中断或复位操作予以清除。

3. 处理器等待指令

使处理器处于等待状态(WAIT)，使处理器本身与外部事件同步。这时处理器的等待状态持续到外部中断发生时为止。

关于各指令的详细语法格式请见附录 A。

第二节　80286 CPU 结构及指令系统

80286 CPU 是 Intel 公司于 1984 年推出的新一代高性能的微处理器，它不论是在速度上还是在功能上都比 8088 CPU 更加强大，是 IBM AT 计算机的 CPU 芯片。80286 有两种工作方式，一种称为实地址方式，一种称为保护虚地址方式(或保护方式)。在实地址方式下，80286 相当于一个增强了的 8088，它支持 8088 的所有指令。实地址方式是 AT 机的默认方式，即计算机开机以后的方式，除非一个程序有意地将方式改变为保护方式，否则将一直保持在这一方式下。在保护方式下，80286 支持任何在实地址方式下的操作，并为数据保护和内存管理提供了许多功能，这就为多用户和多任务提供了硬件支持。80286 可以管理两种内存，一种是物理地址空间，另一种是虚地址空间，物理地址空间是 80286 当时使用的内存，而虚地址空间是 80286 可以使用到的内存空间。两个地址空间的大小不一样，物理空间可用到 16MB，而虚地址空间则可用到 1GB(2^{30}字节)。所谓虚地址空间，是指一个程序需要访问的内存部分不在物理空间中，此时，操作系统可以用 80286 的内存管理功能，将所需虚存部分转换为物理地址空间。

一、80286 CPU 结构

80286 CPU 结构可以划分为四个部分，即总线部件 BU(Bus Unit)、指令部件 IU(Instruction Unit)、执行部件 EU(Execution Unit)和地址部件 AU(Address Unit)。

(一) 总线部件 BU

总线部分是 80286 的传送载体，它的任务是完成存储器及 I/O 的读写，当 BU 不用总线进行其它操作时，它预取指令字节，并把它们放入 6 字节的预取队列中，当执行 JMP 或 CALL 指令时，BU 将冲掉该队列，且将目标地址填入其中。

(二) 指令部件 IU

指令部分从 BU 的代码队列中取出指令，解释它们，将代码放入指令队列或管道上。队列或管道上能存放三条被解释了的指令。

(三) 执行部件 EU

执行部分在内部ROM中的微码控制下执行指令。当它快要执行完当前指令时，ROM发出信号经EU从指令队列中取出下一条指令。这里要注意的是，BU、IU在EU执行指令期间也在工作，这样就保证了EU部分不等待地继续执行下一条指令的可能性。

(四) 地址部件AU

地址部分通过将虚地址转换为实地址，检查保护权限，起内存管理和保护作用。

二、80286 CPU寄存器组成

(一) 与8088相同的寄存器

80286 CPU的寄存器包含了8088 CPU所有的寄存器，如通用寄存器AX、BX、CX和DX，指针寄存器BP、SP、SI和DI，段寄存器CS、SS、DS和ES，指令指针寄存器IP，标志寄存器F。其中标志寄存器中的标志位比8088的标志位增加了两个标志，如图1-6所示。

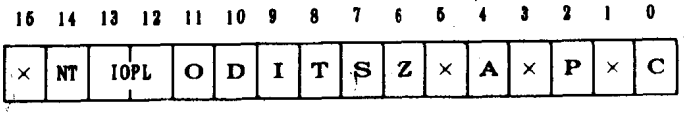


图1-6 80286标志寄存器

标志寄存器中的CF、PF、AF、ZF、SF、TF、IF、DF和OF各位的含义与8088相同，新增加的两个标志IOPL和NT用于80286的保护方式。其中IOPL标志是输入/输出特权标志，用于保证指令只完成处于0～3特权层中的I/O操作；NT标志是嵌套进程标志，表示当前执行的进程是否嵌套于另一进程中，若是，则NT=1，否则NT=0。

(二) 80286新增加的寄存器

1. 机器状态字寄存器

除以上各寄存器外，80286增加了一个机器状态字寄存器MSW。机器状态字寄存器只用了最低四位，如图1-7所示。

MSW中各位的含义是：

(1) PE——保护允许位

当PE为1时，表示80286处于保护方式，否则表示80286处于实地址方式。

(2) MP——监控协处理器位

该位与TS位一起使用，用于确定当TS=1时，WAIT指令是否产生一个协处理器不可用故障。

(3) EM——仿真协处理器位

当EM=1时，会引起所有协处理器操作码都产生一个协处理器不可用故障。若EM=0时，则所有协处理器操作码都将在实际的80286协处理器上执行。

(4) TS——任务切换位

每当任务切换操作时自动置位。此时，协处理器操作码将导致协处理器不可用陷井。

2. 80286用于保护方式的寄存器

80286为了实现保护虚地址下的运行，还增加了以下四个寄存器：

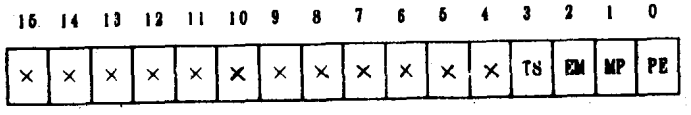


图1-7 80286机器状态字

(1) 全局描述器表寄存器GDTR；

(2) 中断描述器表寄存器IDTR；

(3) 局部描述器表寄存器 LDTR；

(4) 任务寄存器 TR。

对于这些寄存器数据的存取操作，要用 80286 提供的专用指令来进行。

三、80286 的指令系统

(一) 80286 比 8088 增强的指令

80286 还对 8088 的一些指令的功能进行了改进和功能增强，例如：

(1) 整数、带符号数的乘法指令能够让一个带符号或无符号字与一个立即数相乘而产生一个单字长度的积。

(2) 入栈指令能够使一个立即数象寄存器内容或存储单元内容一样地入栈。

(3) 移位和循环移位指令能够用 1 到 31 之间的立即数来操作，而在 8088 中则要求将非 1 的移位计数值放入寄存器 CL 中。

(二)80286 新增加的指令

80286 的指令系统包括了 8088 微处理器的全部指令，并增加了以下几种新的指令：

1. 为支持高级语言提供的指令

(1) 检查数组越界(BOUND)；

(2) 为高级过程建立堆栈区(ENTER)；

(3) 退出高级过程(LEAVE)。

2. 增加的串操作指令

(1) 输入串(INS，INSB 和 INSW)；

(2) 输出串(OUTS，OUTSB 和 OUTSW)。

3. 增加的堆栈操作指令

(1) 使所有通用寄存器入栈(PUSHA)；

(2) 使所有通用寄存器出栈(POPA)。

4. 增加的保护控制指令

(1) 装全局描述表寄存器 LGDT；

(2) 存全局描述表寄存器 SGDT；

(3) 装中断描述表寄存器 LIDT；

(4) 存中断描述表寄存器 SIDT；

(5) 装局部描述表寄存器 LLDT；

(6) 存局部描述表寄存器 SLDT；

(7) 装任务寄存器 LTR；

(8) 存任务寄存器 STR；

(9) 装机器状态字 LMSW；

(10) 存机器状态字 SMSW。

除上面所介绍的指令以外，还有装入访问权指令 LAR、调整请求特权级指令 ARPL、验证段的读/写访问指令 VERR/VERW 和装入段限指令 LSL 等。

第三节　80386 芯片结构及指令系统

80386 芯片在速度上和功能上比 80286 芯片更加强大，它不仅包含了 80286 的全部功能，而且在功能上有了重大的改进。80386 有了 32 位的寄存器与数据通路，从而突破了 8088、80286 等 16 位微处理器的性能。80386 是 32 位的微处理芯片，具有分段、分页功能，具有 32 位的地址和数据类型。80386 与 80286 一样，也有两种操作方式：实地址方式和虚地址保护方式。最大寻址能力可达 4 千兆字节的物理存储空间和 64 兆兆(2^{46})字节的虚拟存储空间，有内部集成的存储管理部件和保护机构，从而更有力地支持了多任务及多用户的操作系统。

一、80386 芯片结构

80386 芯片由中央处理器、存储管理部件和总线接口组成。

中央处理器由执行部件和指令部件组成。执行部件中包括了 8 个用于地址计算和数据操作的通用寄存器和一个用于加速移位、循环移位、乘法和除法操作的 64 位的桶形移位器。

存储管理部件包括一个段部件和一个页部件。段部件通过提供一个附加的寻址分量和有效的共享实现对逻辑地址空间的管理。该附加的寻址分量简化了代码和数据的再定位。调页机构在分段过程之下操作，并且对分段程序是透明的，以实现对物理地址空间的管理。

80386 总线接口提供地址流水线操作、动态数据总线宽度调整以及为数据总线每个字节提供直接的字节允许信号，确保了短的平均指令执行时间和很高的系统吞吐量。

二、80386 CPU 寄存器组成

(一) 通用寄存器

80386 有 8 个 32 位的寄存器：EAX、EBX、ECX、EDX、ESP、EBP、ESI 和 EDI。每个寄存器的低 16 位都可以独立地存取，它们分别是 AX、BX、CX、DX、SP、BP、SI 和 DI，其中 AX、BX、CX 和 DX 各自还可以被分成 2 个 8 位的寄存器。

(二) 段寄存器

80386 有 6 个段地址寄存器 CS、DS、SS、ES、FS 和 GS。正在执行的程序由 CS 寻址，当前活动的数据段由 DS 寻址，堆栈段由 SS 寻址。另外，程序员可以对三个并行活动的数据段 ES、FS 和 GS 进行数据存取。

(三) 控制与状态寄存器

80386 CPU 有 4 个 32 位的控制寄存器 CR0、CR1、CR2 和 CR3(其中 CR1 保留)，一个 32 位的指令指针(EIP)和一个 32 位的标志寄存器(EFLAGS)。

1. 控制寄存器

CR0 的 0～15 位为机器状态字 MSW。CR2、CR3 用于支持分页特征，CR2 包含了造成最后一次页故障的线性地址，CR3 中包含了目录表的物理基地址。

2. 指令指针寄存器

指令指针寄存器 EIP 保存着即将执行的下一条指令的偏移量，它总是与当前的 CS 段相配合，以形成指令的物理地址。EIP 允许单独使用其低 16 位，其名字为 IP，以保证与 8088、8086 和 80286 的兼容。

3. 标志寄存器

标志寄存器 EFLAGS 用于控制或标识 80386 的某些操作，或标识某些指令执行以后的结

果和特征。标志寄存器的低 16 位命名为 F，标志位与 80286 完全相同。80386 增加了两种标志，即虚拟方式标志 VM 和恢复标志 RF，如图 1－8 所示。

当 VM＝1 时，表示 80386 工作在虚拟保护方式下，否则就是工作在实地址方式下。RF 标志通常与调试寄存器中的断点和单步操作一起使用。当 RF＝1 时，则在下一条指令中的所有调试故障都将被忽略，在成功地完成每一条指令时，RF 将自动复位。

（四）调试与测试寄存器

80386 有 8 个 32 位的调试寄存器 DR0～DR7（其中 DR4、DR5 被保留）和两个 32 位的测试寄存器 TR6 和 TR7。

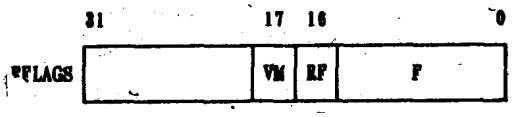


图 1－8　80386 标志寄存器

调试寄存器为程序员提供了调试程序的手段，而测试寄存器则用于对转换设备缓冲区中的 RAM 以及内容可寻址存储器的测试。

（五）保护方式下的寄存器

与 80286 一样，80386 也设立了四个专门用于保护虚地址方式下的寄存器，它们分别是 GDTR、IDTR、LDTR 和 TR。

三、80386 指令系统

80386 指令系统一共有 120 多条指令，分为数据传送、算术运算、逻辑运算、字符串处理、位处理、控制转移、高级语言支持、操作系统支持和处理机控制等九种类型。这些类型的指令除包含 80286 的所有指令外，主要增加了如下一些指令：

1. 增加的数据传送指令

(1)传送字节或字，零扩展至双字(MOVZX)；

(2)传送字节或字，符号扩展至双字(MOVSX)；

(3)字扩展成双字(CWDE)；

(4)双字扩展成四倍字(CDQ)；

(5)装入 FS 段寄存器(LFS)；

(6)装入 GS 段寄存器(LGS)；

(7)装入 SS 段寄存器(LSS)；

(8)32 位标志入栈(PUSHFD)；

(9)32 位标志出栈(POPFD)；

(10)通用寄存器入栈(PUSHAD)；

(11)通用寄存器出栈(POPAD)。

2. 增加的逻辑移位指令

(1)双精度左移(SHLD)；

(2)双精度右移(SHRD)。

3. 增加的位处理指令

(1)位测试(BT)；

(2)位测试和置位(BTS)；

(3)位测试和清除(BTR)；

(4)位测试和求反(BTC)；

(5)位正向扫描(BSF)；

(6)位反向扫描(BSR)。

4. 增加的串操作指令

(1)双字串比较(CMPSD);

(2)双字串扫描(SCASD);

(3)双字串输入(INSD);

(4)双字串输出(OUTSD);

(5)取双字串(LODSD);

(6)存双字串(STOSD);

(7)插入位串(IBTS);

(8)抽出位串(XBTS)。

第四节　语句的种类及格式

一、语句的种类

汇编语言的语句可以分为三类:指令语句、伪指令语句和宏指令语句。

指令语句是能产生目标代码的、CPU 可以执行的能完成特定功能的语句。而伪指令语句则是起说明作用的,不产生目标代码。宏指令语句是由若干条指令性语句组成的、能完成一个较完整功能的指令体。

二、语句的一般格式

(一) 指令语句的一般格式

标号:　　指令助记符　　操作数,操作数　　;注释

(二) 伪指令语句的一般格式

名字　　伪指令定义符　参数 1,参数 2,…　　;注释

从上面的格式可以看出,不论是指令性语句,还是伪指令性语句,它们都由四个部分组成。第一部分是标号或名字,它是给指令单元或伪指令起的符号名称;第二部分是操作助记符或定义符,它告诉 CUP 要执行什么样的操作或告诉汇编程序伪指令语句的伪操作功能;第三部分是操作数部分或参数部分,它指明指令操作的对象或进行伪操作时所需的参数;第四部分是以";"号为开头的注释,它主要是提供给程序员用以对程序进行功能方面的描述和说明。

标号和名字的区别在于:标号后有冒号,名字后面没有冒号;标号可以作为 JMP 和 CALL 指令的转移目标,与具体的指令地址相联系,而名字一般用作定义变量名、过程名、段名使用。

(三) 宏指令语句

1. 宏指令的定义

在汇编语言中,可以把需要重复使用的一段指令性语句,定义为一条宏指令。这样,在汇编语言需要这一段指令的地方可以写一个宏调用,从而既减少了指令的重复书写,又可以使编写的程序结构清楚。

宏定义的一般格式是:

名字 MACRO [形式参数 1 [,形式参数 2[,…]]]

ENDM

其中 MACRO 是宏指令的开始，ENDM 是宏指令的结束，在 MACRO 与 ENDM 之间的所有指令行称为宏定义体。形式参数是一个虚符号名，相当于一个变量，在进行宏调用时，要被实在参数所替代。

2. 宏调用语句

在进行了一个宏定义之后，只要在需要的地方就可以进行宏调用。宏调用语句的格式为：

宏定义名 [实在参数 1 [,实在参数 2[,…]]]

汇编程序在进行汇编时，把宏定义体插入到宏调用处，这称为宏扩展。

3. 宏定义的取消

一个宏指令名可以用伪指令 PURGE 来取消，然后就可以重新定义。PURGE 语句的一般格式为：

PURGE　宏定义名[,…]

三、伪指令语句

伪指令语句可以分为以下几类：

(1)符号定义语句；

(2)数据定义语句；

(3)段定义语句；

(4)过程定义语句；

(5)程序结束语句。

(一)符号定义语句

符号定义语句主要用于给一个符号定义一个值或把其定义为别的符号名。主要有等值(EQU)语句和等号("=")语句两种。其格式分别为：

符号名　EQU　表达式

符号名　=　表达式

(二)数据定义语句

数据定义语句主要功能是为一个数据项分配存储单元，用一个符号名与这个或这些单元相联系，并为这些单元提供一个初始值。数据定义语句的一般格式：

名字　数据定义符　初值

其中"数据定义符"是规定数据类型的符号，一般有：DB(字节)、DW(字)、DD(双字)等几个类型。初值可以是一个数，也可以是一张数据表。

(三)标号及其属性

对于每个指令单元的标号，都具有一个属性，或者是近(NEAR)属性，或者是远(FAR)属性。NEAR 属性的标号只产生偏移量，FAR 属性的标号则产生全地址(段地址和偏移量)。标号的属性可以是隐含的，也可以进行定义。在代码段中的所有指令单元的标号都隐含具有近属性，如果需要定义为远属性，可以采用下面的格式：

标号　LABEL　FAR

具有近属性的标号只能在本段内使用，而具有远属性的标号不仅可在本段内使用，还可以被其它段所使用。

不论是数据单元地址标号还是指令单元标号，都可以使用所谓的分析操作符和合成操作符对它们的地址属性进行读取和修改操作。分析操作符有 SEG、OFFSET、TYPE、SIZE 和 LENGTH 等几种，它们的使用格式和含义如下：

格　式		含　义
SEG	标号	取标号的段值
OFFSET	标号	取标号的偏移地址
TYPE	标号	取标号的类型。标号类型及其对应值见表 1－2。
SIZE	标号	取标号的长度(以所定义的类型为单位)
LENGTH	标号	取标号的字节单元数

合成运算符用于修改标号的类型，或者用于给标号规定一个新的类型，有 PTR 和 THIS 两种。

PTR 操作符的使用格式如下：

类型　PTR　标号

其中类型可以是 BYTE、WORD、DWORD、NEAR 和 FAR 等，这一伪操作使得 PTR 后面的标号具有 PTR 前面的类型，而不管该标号的原有类型是什么。

表 1－2　标号类型及其对应值

标号类型	返回值
字节	1
字	2
双字	4
近标号	－1
远标号	－2

THIS 操作符的使用格式如下：

新标号　EQU　THIS　类型

原标号　……

THIS 伪操作使得其下面的“原标号”具有新的标号名和新的类型，但是并不分配新的存储单元。

(四) 段定义语句

段定义语句可以使程序设计结构化，便于内存区域的合理使用。它的一般格式为：

段名　SEGMENT [定位类型] [组合类型] ['类别']

.

.

.

段名　ENDS

段名是识别段的标志，其后的三个选择项是段的三个属性。它们各自的含义、类型及功能如下：

1. 定位类型

PARA —— 段定位在节边界(缺省类型)；

BYTE —— 段定位在字节边界；

WORD —— 段定位在字边界；

PAGE —— 段定位在页边界。

2. 组合类型

PUBLIC —— 与其它同名的段顺序连接；

COMMON —— 与其它同名的段重迭连接；

STACK —— 该段用作堆栈段，与同名的段顺序连接；

MEMORY —— 连接时，该段定位在所有其它段之上；

AT 表达式—— 把该段装在表达式的值所指定的段地址上。

3. '类别'

类别告诉该段的类型，连接时类别相同的段被集中在一起。类别一般有 CODE、DATA 和 STACK 等。

（五）其它伪定义语句

1. ASSUME 定义语句

格式：ASSUME 段寄存器：段名[,…]

功能：ASSUME 告诉汇编程序段名所指定的段由哪一个段寄存器寻址。但只有最先使用的代码段和堆栈段的初始值在装入系统时由系统设置。其它段寄存器的实际值要由程序来设置。

2. 起始地址设定语句 ORG

格式：ORG 表达式

功能：该语句指明了其后的程序或数据，以表达式的值为起始地址进行存放。

3. 程序结束语句 END

格式：END [标号]

功能：该语句告诉汇编程序，主程序或模块程序在哪儿结束。其中只有主程序的结束语句带有标号，该标号必须是程序运行时第一条要执行的指令标号。

第五节　源程序的一般结构

汇编语言源程序一般分为两种结构形式，一种是 EXE 文件源程序结构，另一种则是 COM 文件源程序结构。EXE 文件源程序可以设有若干个堆栈段、若干个数据段和若干个代码段，在加载过程中需要进行段的重定位，一般应用在中、大型程序设计的情况下。而 COM 文件源程序只允许设有一个段，不允许设置堆栈段或数据段，适合于小型程序设计。

一、EXE 源程序文件的一般结构

一个完整的 EXE 文件源程序应该包括堆栈段、数据段和代码段，特殊情况下还可以使用附加数据段。对于每一类别的段都可以有若干个，这往往是出现在多模块程序设计的情况下。EXE 程序的入口（第一条执行的指令地址）可以设置在代码段中的任何地方，但是必须与程序的结束语句 END 后面的标号相一致。下面是一个标准的汇编语言源程序结构。

```
SSEG    SEGMENT    STACK 'STACK'
        DB         256  DUP(0)
SSEG    ENDS
DSEG    SEGMENT    PUBLIC 'DATA'
                   .
                   .

DSEG    ENDS
```

```
CSEG    SEGMENT   PUBLIC ' CODE'
        ASSUME    CS:CSEG,DS:DSEG,ES:DSEG
MAIN    PROC      FAR
        PUSH      DS                          ;保护 PSP 段地址
        XOR       AX,AX
        PUSH      AX                          ;保护偏移 0 地址
        MOV       AX,DSEG
        MOV       DS,AX                       ;建立数据段的可寻址性
        MOV       ES,AX                       ;建立附加数据段的可寻址性
                  .
                  .
                  .
        RET                                   ;返回 DOS
MAIN    ENDP
CSEG    ENDS
        END       MAIN                        ;程序结束
```

以上只是汇编语言源程序的一般书写格式，特殊情况下可以有所取舍。例如，有的源程序只设有堆栈段和代码段，不专门设置数据段，而是把需要的数据放在代码段中。有的源程序则只设有数据段和代码段甚至只设有代码段，这种情况下堆栈段由系统设定。下面是一个完整的汇编语言源程序文件，它在结构上完全采用了上述的格式。该程序完成的功能是在屏幕上当前光标处显示一个字符串："How are you?"，然后返回 DOS。

二、COM 源程序文件的一般结构

COM 文件源程序只允许设有一个段，不允许设置专门的堆栈段或数据段，程序的指令和执行过程中的数据以及堆栈操作都在一个段(代码段)中。程序中的第一条指令必须从偏移量 100H 处开始，也即在代码段的开始部分第一条指令之前要写上：ORG 100H。

许多系统程序都采用 COM 格式的文件结构，这是由于 COM 文件中的数据、堆栈、程序指令都包含在代码段中，即整个程序的长度不会超过 64K 字节，因而 COM 文件结构紧凑，不需要段的重定位，从而装入速度快。

下面是一个用 COM 结构编写的程序实例。

```
CSEG      SEGMENT   PUBLIC ' CODE'
          ORG       100H
          ASSUME    CS:CSEG,DS:CSEG,ES:CSEG
MAIN      PROC      NEAR
          JMP       START
MSG       DB        'How  are  you?  $'
START:    MOV       AH,9
          MOV       DX,OFFSET  MSG
```

```
        INT     21H
        INT     20H
MAIN    ENDP
CSEG    ENDS
        END     MAIN
```

一旦源程序按上述规定编制好后，就可按常规对其进行汇编和连接，最后用 DOS 的外部命令 EXE2BIN 将其转换为扩展名为.COM 类型的文件。

第六节　汇编语言程序的上机过程

建立并运行一个汇编语言程序，一般需要经过以下几个步骤：

一、编辑源程序

利用行编辑 EDLIN 或全屏幕编辑 WORDSTAR 建立一个汇编语言源程序。

二、汇编源程序

运用宏汇编程序 MASM 对汇编语言源程序进行汇编。MASM 最简单的一种使用方法是：

A>MASM　源程序名;<CR>

经过汇编之后，如果源程序没有错误，可以自动生成一个目标程序(.OBJ)。如果源程序中有错误，可以再回到步骤一，对错误的地方进行修改，修改完成后重新进行汇编。

三、连接目标程序

经过汇编以后得到的目标程序必须经过连接，得到可执行(.EXE)文件。连接要使用连接程序 LINK。LINK 程序最简单的一种使用方法是：

A>LINK　目标程序名;<CR>

如果在连接过程中没有发现错误，连接程序会自动生成一个可执行文件(.EXE)。注意，对于采用 COM 格式编写的汇编语言源程序在进行连接时，会产生如下错误：

```
Warning: NO stack segment
There was 1 error detected
```

这是一种必然的结果，因为 COM 源程序文件只允许设有一个代码段，这种错误并不影响 COM 文件的生成，用户不用理会这一提示。

四、运行程序

如果汇编语言源程序编写时采用了 EXE 文件格式，则在目标程序连接完成后，即可直接运行得到的 EXE 文件。运行 EXE 文件的方法是，在 DOS 状态下直接打入所执行的文件名。如果汇编语言源程序编写时采用了 COM 文件格式，则在目标程序连接完成后，所得到的 EXE 文件还不能被运行，需要使用 DOS 外部命令 EXE2BIN 将所得到的 EXE 文件转换为 COM 文件，然后才能运行得到的 COM 文件。

五、调试程序

如果在运行程序过程中，发现程序有逻辑错误，要用 DEBUG 调试程序进行动态地跟踪调试，查找出错误的原因，重新对源程序进行编辑、汇编、连接和调试，这种过程一直循环，直到得到的 EXE 文件或 COM 文件能正确运行为止。

汇编语言程序上机的全过程如图 1－9 所示。

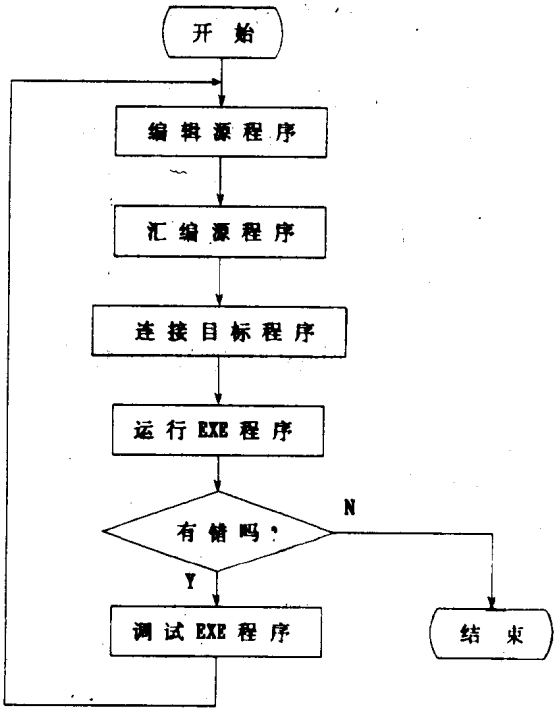


图 1－9　汇编语言程序上机的全过程

第二章　基本指令编程实例

8088 指令系统中的所有 100 多条指令从功能上分类可以分为五大类，它们分别是数据传送指令、算术运算指令、逻辑运算指令、串操作指令和程序控制指令等。每一类中都有若干条指令，每一类中的指令在宏观功能上都有其共同的特点，但每条指令又都有其具体的功能、用法和编程技巧。本章讨论 8088 指令系统中各类指令的基本用法和编程技巧，以便使初学者在学习完每一类指令以后，初步掌握每一类指令的基本用法和编程技巧。

第一节　数据传送指令编程实例

一、数据传送指令概述

基本数据传送指令可分为如下四类：

（一）通用数据传送指令

名称	助记符	操作数
传送指令	MOV	目的，源
字压栈	PUSH	16 位操作数
字弹栈	POP	16 位操作数
交换指令	XCHG	目的，源
查表指令	XLAT	（隐含）

（二）I/O 端口操作指令

名称	助记符	操作数
端口输入指令	IN	端口地址
端口输出指令	OUT	端口地址

（三）地址传送指令

名称	助记符	操作数
传送偏移地址	LEA	目的，源
传送双字地址	LDS	目的，源
传送双字地址	LES	目的，源

（四）标志传送指令

名称	助记符	操作数
标志寄存器低 8 位→AH	LAHF	（隐含）
AH→标志寄存器低 8 位	SAHF	（隐含）
标志进栈指令	PUSHF	（隐含）
标志弹栈指令	POPF	（隐含）

二、程序设计实例

【例 2.1】 寄存器之间的数据交换

使用三种不同的方法编写三段程序，将 AX 和 BX 的内容进行交换。

编程指导：

最简单的交换方法是使用一条 XCHG 指令，这样即简单又方便。第二种方法是利用数据传送指令 MOV，接助于中间寄存器 CX 进行交换，需要三条指令。第三种方法是根据堆栈操作中数据先进后出的原则，利用堆栈操作指令进行交换，需要四条指令。

(1) 利用 XCHG 指令

```
XCHG        AX，BX
```

(2) 利用 MOV 指令

```
MOV         CX，AX
MOV         AX，BX
MOV         BX，CX
```

(3) 利用堆栈操作指令

```
PUSH    AX
PUSH    BX
POP     AX
POP     BX
```

【例 2.2】 打开/关闭光标

下面的一段程序利用直接 I/O 端口操作可以把光标打开和关闭。

编程指导：

光标是由 6845 芯片产生的。6845 芯片共有 16 个数据寄存器，其中寄存器 10 的第 5 位为 1 时关闭光标，为 0 时打开光标。该寄存器还保存光标的“起始线”值，而寄存器 11 则保存光标“终止线”值，二者相配合确定了光标的高度。由于光标关闭时，光标的大小即无实际意义，因此只需将 20H 送入寄存器 10 把第 6 位置 1。若要重新打开光标，则必须置起始线的值。对于正常光标，该值为 11。光标终止线值并不受影响，因为该值存于 11 号寄存器中。要想访问 6845 的各个寄存器，首先应将所访问的寄存器号送入地址寄存器，地址寄存器在单色卡上为 3B4H 端口，在彩色卡上为 3D4H 端口。而要访问的第 10、11 号寄存器的端口地址在单色卡上是 3B5H，在彩色卡上为 3D5H。下面是在单色卡上打开、关闭光标的程序。

程序清单

(1) 关闭光标

```
MOV     DX，3B4H        ；单色卡 6845 芯片地址寄存器
MOV     AL，10          ；选择 10 号数据寄存器
OUT     DX，AL          ；送请求
INC     DX              ；DX 指向数据寄存器端口
MOV     AL，20H
OUT     DX，AL          ；关闭光标
```

(2) 打开光标

```
MOV     DX，3B4H
```

```
MOV     AL，10
OUT     DX，AL          ；选择 10 号寄存器
INC     DX
MOV     AL，00H         ；打开光标并设定光标起始线为 0
OUT     DX，AL          ；打开光标请求
```

【例 2.3】设置/清除单步标志

在 8088 汇编语言指令系统中，没有专门的对单步标志 TF 位进行置位和复位的指令，下面的程序通过堆栈进行数据传送来设置单步标志位。

编程指导：

8088 标志寄存器为 16 位的寄存器，其中只有 9 个标志位。这 9 个标志位是按位进行定义，用以反映系统运行的状态以及运算结果的特点。9 个标志位一般被划分为两大类型：状态标志和控制标志。如图 1－2 所示。

状态标志通常是在进行算术或逻辑运算之后，由系统根据运算结果的特点进行设置，以反映这种操作结果的某种性质。控制标志可以由指令进行设置，使处理器按照用户的意图进行操作。对于控制标志来说，8088 只设置了对方向标志和中断允许标志进行操作的四条指令：CLD、STD、CLI、STI，没有设置对单步标志进行操作的指令。单步标志的用途是：若 TF＝1，则 CPU 每执行一条指令即产生一个内部中断 1，允许程序检查各个寄存器的内容及标志位。若 TF＝0，则 CPU 连续执行指令，直到程序结束。

程序清单

(1) 置单步标志位

```
PUSH    AX              ；保护 AX
PUSHF                   ；标志进栈
POP     AX              ；标志寄存器内容→AX
OR      AX，0100H       ；置单步标志（单步标志在标志寄存器的第 8 位）
PUSH    AX              ；修改结果进栈
POPF                    ；置单步标志后的状态→标志寄存器
POP     AX
```

(2) 清单步标志位

```
PUSH    AX              ；保护 AX
PUSHF                   ；标志进栈
POP     AX              ；标志寄存器内容→AX
AND     AX，0FEFFH      ；清单步标志位
PUSH    AX              ；修改结果进栈
POPF                    ；处理后的状态→标志寄存器
POP     AX
```

【例 2.4】查表替换

下面一个程序完成把 AX 中的四位十六进制数通过查表分别替换成相应的 ASCII 码字符，然后显示在屏幕上。

编程指导：

十六进制数共有 16 个，它们分别是：00H，01H，02H，03H，04H，05H，06H，07H，08H，09H，0AH，0BH，0CH，0DH，0EH，0FH。要显示它们，必须把它们分别转换为对应的 ASCII 码：30H～39H（对应 00H～09H），41H～46H（对应于 0AH～0FH）。下面的程序采用查表指令 XLAT 进行十六进制到 ASCII 码的转换。XLAT 指令的功能是：把 BX 所指表中的以 AL 值为偏移地址的一字节内容送 AL 本身。为了把 AX 中的四位十六进制数分别查表、转换并显示出来，我们采用循环移位的方法，每次把一位十六进制数（4bit）移至 AL 的低四位，然后屏蔽 AL 的高四位，通过查表，转换为相应的 ASCII 码显示出来即可。

程序清单

```
STACK    SEGMENT PARA STACK ' STACK'
         DB 128 DUP (0)
STACK    ENDS
DATA     SEGMENT
ASC_TAB  DB ' 0123456789ABCDEF'
DATA     ENDS
CODE     SEGMENT
         ASSUME   CS: CODE, DS: DATA, ES: DATA
MAIN     PROC     FAR
         PUSH     DS
         XOR      AX, AX
         PUSH     AX
         MOV      AX, DATA
         MOV      DS, AX
         LEA      BX, ASC_TAB
         MOV      AX, 3FA7H
         MOV      CX, 4              ; 处理 4 位十六进制数
RETRY:   PUSH     CX
         MOV      CL, 4
         ROL      AX, CL             ; 一个十六进制数→AL 低 4 位
         PUSH     AX                 ; 保护 AX
         AND      AL, 0FH            ; 屏蔽字节高 4 位
         XLAT                        ; 查表替换：[BX+AL] →AL
         MOV      DL, AL
         MOV      AH, 2
         INT      21H                ; 显示一位十六进制数
         POP      AX                 ; 恢复 AX
```

```
        POP     CX
        LOOP    RETRY
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.5】清屏程序

下面的程序完成图形方式下的清屏功能，相当于 DOS 的内部命令 CLS。

编程指导：

在 IBM PC 系统中，显示器有 7 种工作方式，其中方式 4、5、6 是图形方式。在这三种工作方式下，屏幕上的所有信息都是由点来组成的，这些点的信 息存放在所谓的屏幕缓冲区中。屏幕缓冲区的结构 形式如图 2－1 所示。屏幕缓冲区把屏幕上点的信息分奇、偶扫描行分别存放于从 BA00 段和 B800 段中，每 区 8000 个字节。在方式 4 和方式 5 状态下，屏幕上点 的分辨率为 320×200，也即屏幕水平方向有 320 个 点，垂直方向有 200 个点，在这种工作方式下，每 两个 BIT 位构成一个点。在方式 6 状态下，屏幕上点 的分辨率为 640×200，也即屏幕水平方向有 640 个 点，垂直方向有 200 个点，在这种工作方式下，每 一个 BIT 位即表示一个点。但是，不论那一种方式，只要对应点的 BIT 位为 0，该点在屏幕上即显示为暗 点。根据这一规律，我们要清除屏幕，只要把屏幕 缓冲区（包括奇扫描区和偶扫描区）的内容全部置为 0 即可。在下面的程序设计中，我们采用了对奇、偶缓冲区同时操作的方法，即每清除一字节的偶扫描区的点，接着清除一字节奇扫描区的点。这样重 复执行 8000 次，即可完成清屏。

程序清单

```
STACK   SEGMENT PARA STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
VDIE1   DW 0000H          ; 偶扫描区起始偏移地址
        DW 0B800H         ; 偶扫描区段地址
VDIE2   DW 0000H          ; 奇扫描区起始偏移地址
        DW 0BA00H         ; 奇扫描区段地址
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA,ES:DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
```

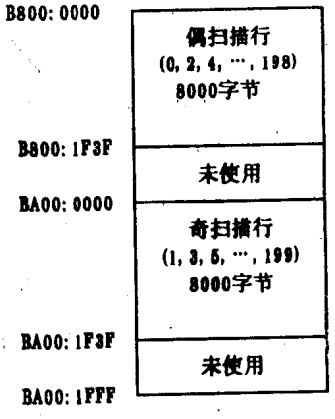


图 2－1　屏幕缓冲区信息格式

```
        MOV     DS，AX
        MOV     ES，AX
        MOV     AH，0FH
        INT     10H             ；取显示器工作方式→AL
        TEST    AL，04H         ；测试是否为图形方式?
        JZ      EXIT            ；不是图形方式 4，5，6，转 EXIT
        LEA     BX，VDIE1
        LES     DI，[BX]        ；偶扫描区的段地址→ES，偏移量→DI
        LDS     SI，[BX+4]      ；奇扫描区的段地址→DS，偏移量→SI
        MOV     AL，0           ；填充字节为 0
        MOV     CX，8000        ；每个区有 8000 个字节
RETRY：MOV      ES：[DI]，AL    ；填充偶扫描区一个字节
        MOV     DS：[SI]，AL    ；填充奇扫描区一个字节
        INC     SI
        INC     DI              ；修改地址
        DEC     CX              ；修改计数
        JNZ     RETRY           ；重复执行
EXIT：  RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

第二节　二进制算术运算编程实例

一、二进制算术运算指令

在 8088 指令系统中，提供了加、减、乘、除四则运算指令 20 条，其中用于二进制运算的指令如下所示：

（一）加法指令

名称	助记符	操作数
不带进位的加法指令	ADD	目的，源
带进位的加法指令	ADC	目的，源
加 1 指令	INC	操作数

（二）减法指令

名称	助记符	操作数
不带进位的减法指令	SUB	目的，源
带进位的减法指令	SBB	目的，源
求补指令	NEG	操作数
比较指令	CMP	目的，源
减 1 指令	DEC	操作数

（三）乘法指令

名称	助记符	操作数
无符号数乘法指令	MUL	乘数
带符号数乘法指令	IMUL	乘数

（四）除法指令

名称	助记符	操作数
无符号数除法指令	DIV	除数
带符号数除法指令	IDIV	除数
字节扩展指令（字节扩展成字）	CBW	（隐含）
字扩展指令（字扩展成双字）	CWD	（隐含）

二、二进制算术运算编程实例

【例 2.6】多字节无符号二进制数加法

下面的程序完成将 DATA1 单元的 4 字节无符号二进制数 93H、0F8H、7AH、12H 与 DATA2 单元的 4 字节无符号二进制数 6CH、0D9H、73H、24H 进行加法运算。

编程指导：

考虑到编程的方便，在定义加数和被加数时，将低位放在低地址，将高位放在高地址。进行加法运算时，按照由低地址到高地址的顺序取得数据，而每一步运算的结果也按照低位放在低地址、高位放在高地址的方式进行存储。

程序清单

```
STACK   SEGMENT PARA STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
DATA1   DW 12H, 7AH, 0F8H, 93H
COUNT   EQU ($-DATA1) / (TYPE DATA1)
DATA2   DW 24H, 73H, 0D9H, 6CH
DATA3   DW 5 DUP (0)
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA
MAIN    PROC      FAR
        PUSH      DS
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA
        MOV       DS, AX
        MOV       SI, OFFSET DATA1
        MOV       DI, OFFSET DATA2
```

```
        MOV     BX, OFFSET DATA3
        MOV     CX, COUNT
        CLC
RETRY:  MOV     AX, [SI]
        INC     SI
        INC     SI
        ADC     AX, [DI]
        INC     DI
        INC     DI
        MOV     [BX], AX
        INC     BX
        INC     BX
        LOOP    RETRY
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.7】多字节无符号二进制数减法

下面的程序完成将 DATA1 单元的 4 字节无符号二进制数 93H、0F8H、7AH、12H 与 DATA2 单元的 4 字节无符号二进制数 6CH、0D9H、73H、24H 进行减法运算。

程序清单

数据段：

```
DATA1  DW 12H, 7AH, 0F8H, 93H
COUNT  EQU ($-DATA1) / (TYPE DATA1)
DATA2  DW 24H, 73H, 0D9H, 6CH
DATA3  DW 5 DUP (0)
```

代码段：

```
        MOV     SI, OFFSET DATA1
        MOV     DI, OFFSET DATA2
        MOV     BX, OFFSET DATA3
        MOV     CX, COUNT
        CLC
RETRY:  MOV     AX, [SI]
        INC     SI
        INC     SI
        SBB     AX, [DI]
        INC     DI
```

```
        INC     DI
        MOV     [BX], AX
        INC     BX
        INC     BX
        LOOP    RETRY
```

【例 2.8】无符号二进制数的乘法

下面的程序分别说明了字节与字节、字节与字、字与字等无符号数的乘法运算。

编程指导：

在进行数的乘法运算时，8088 要求被乘数的长度必须与乘数长度的相同。即：如果乘数是字节长度，则被乘数也必须为字节长度。所以，如果在进行乘法运算时，被乘数与乘数的长度不相同，则必须在运算之前将其中较短的一个数进行长度扩展，使其与较长的一个数的长度保持一致。扩展长度的方法是利用 MOV 指令或 XOR 指令将高字节或高字清 0。

程序清单

```
STACK   SEGMENT STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
WORD1   DW      4000
WORD2   DW      3000
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        ;
        MOV     AL, 30      ; 字节 (AL) ×字节 (BL) →AX
        MOV     BL, 20
        MUL     BL
        ;
        MOV     AL, 30      ;字节(AL)扩展成字(AX),乘以字 WORD1→DX:AX
        XOR     AH,AH
        MUL     WORD1
        ;
        MOV     AX,WORD1  ;字(WORD1)×字(WORD2)→DX:AX
```

```
        MUL     WORD2
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.9】无符号二进制数的除法

下面的程序分别说明了字节与字节、字节与字、字与字等无符号数的除法运算。

编程指导：

在进行无符号数的除法运算时，8088 要求被除数的长度必须是除数长度的两倍。例如，如果除数是字节长度，则被除数必须为字长度。所以，如果在进行除法运算时，被除数与除数如果长度相同，必须在运算之前将被除数进行长度扩展，即原来是字节的被除数要扩展成字，原来是字长度的被除数要扩展成双字。扩展长度的方法是利用 MOV 指令或 XOR 指令将高字节或高字清 0。

程序清单

```
STACK   SEGMENT STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
WORD1   DW      4000
WORD2   DW      3000
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        ;
        MOV     AL, 30      ;字节(AL)扩展成字(AX)÷字节(BL)→AL(商)
        XOR     AH, AH      ;                              AH (余数).
        MOV     BL, 20
        DIV     BL
        ;
        MOV     AX, WORD1 ;字(WORD1)÷字节(BL)→AL(商)、AH(余数)
        MOV     BL,20
        DIV     BL
```

```
        ;
        MOV     AX,WORD1  ;字(WORD1)扩展成双字(DX:AX)÷字(WORD2)
        XOR     DX,DX     ;→AX(商),DX(余数)
        DIV     WORD2
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.10】有符号二进制数的乘法

下面的程序分别说明了字节与字节、字节与字、字与字等有符号数的乘法运算。

编程指导：

在进行数的乘法运算时，8088 要求被乘数的长度必须与乘数长度的相同。即：如果乘数是字节长度，则被乘数也必须为字节长度。所以，如果在进行乘法运算时，被乘数与乘数的长度不相同，则必须在运算之前将其中较短的一个数进行符号扩展，使其与较长的一个数的长度保持一致。扩展符号的方法是使用 CBW 和 CWD 指令。

程序清单

```
STACK   SEGMENT STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
WORD1   DW      -6000
WORD2   DW      3000
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        ;
        MOV     AL, 30      ; 字节 (AL) ×字节 (BL) →AX
        MOV     BL, 20
        IMUL    BL
        ;
        MOV     AL, 30
        CBW                 ; 字节 (AL) 扩展成字 (AX), 乘以 WORD1→DX: AX
```

```
        IMUL    WORD1
        ;
        MOV     AX, WORD1 ; 字 (WORD1) ×字 (WORD2) →DX: AX
        IMUL    WORD2
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.11】有符号二进制数的除法

下面的程序分别说明了字节与字节、字节与字、字与字等有符号数的除法运算。

编程指导：

在进行数的除法运算时，8088 要求被除数的长度必须是除数长度的两倍。例如，如果除数是字节长度，则被除数必须为字长度。所以，如果在进行除法运算时，被除数与除数的长度相同，必须在运算之前将被除数进行符号扩展，即原来是字节的被除数要扩展成字，原来是字长度的被除数要扩展成双字。扩展符号的方法是使用 CBW 和 CWD 指令。

程序清单

```
STACK   SEGMENT STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
WORD1   DW      -6000
WORD2   DW      3000
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        ;
        MOV     AL,30
        CBW                 ;字节(AL)扩展成字(AX)÷字节(BL)→AL(商)
        MOV     BL,20
        IDIV    BL
        ;
        MOV     AX,WORD1    ;字(WORD1)÷字节(BL)→AL(商)、AH(余数)
        MOV     BL,20
```

```
        IDIV    BL
        ;
        MOV     AX,WORD1
        CWD                     ;字(WORD1)扩展成双字(DX:AX)÷字(WORD2)
                                ;→AX(商),DX(余数)
        IDIV    WORD2
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

第三节　逻辑指令编程实例

一、逻辑指令概述

逻辑指令是可以对二进制位（Bit）进行操作的指令，按其功能可以划分为如下三个类型：

（一）逻辑运算指令

1. 逻辑与

助记符	操作数	功能
AND	目的，源	对应的两个 BIT 位同时为 1，其运算结果才为 1

与运算的特殊情况：

(1) 某一操作数，自身和自身相“与”，操作数自身不变，但可以使 CF 标志清 0。

(2) 使某数若干位维持不变、若干位置“0”。要维持不变的位和“1”相“与”；而要置“0”的位要和“0”相“与”。

2. 逻辑或

助记符	操作数	功能
OR	目的，源	对应的两个 BIT 位同时为 0，其运算结果才为 0

或运算的特殊情况：

(1) 某一操作数自身与自身相“或”，其值不变，但可以使 CF 标志清 0。

(2) 使某数若干位维持不变、若干位置“1”。维持不变的位与“0”相“或”；而要置“1”的位则要与“1”相“或”。

3. 逻辑非

助记符	操作数	功能
NOT	8/16 位操作数	字节或字的各位求反

4. 逻辑异或

助记符	操作数	功能
XOR	8/16 位操作数	对应的两个 BIT 位相同，其运算结果为 0，否则为 1

异或运算的特殊情况：

(1) 某一操作数自身与自身相“异或”，其结果为“0”并使 CF 清 0。

(2) 使某数若干位维持不变、若干位取反。要维持不变的位与“0”相“异或”；而要取反

的位则要与“1”相“异或”。

5. 测试指令

助记符	操作数	功能
TEST	目的，源	两和操作数进行与运算，结果不回送

说明：测试指令 TEST 与上述各个逻辑指令的区别在于：AND 、OR、NOT、XOR 等指令执行以后都要影响操作数本身。而测试指令执行以后，不影响被测试的操作数，只把测试结果的状态反映在标志寄存器上。

（二）移位指令

1. 逻辑左移/算术左移指令

助记符	操作数
SHL/SAL	OPRD，m

功能如图 2－2 所示。

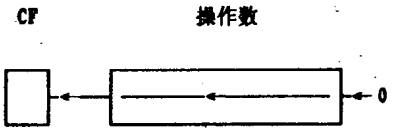


图 2－2　逻辑左移/算术左移

2. 逻辑右移指令

助记符	操作数
SHR	OPRD，m

功能如图 2－3 所示。

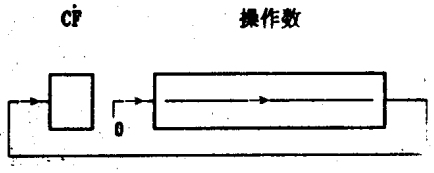


图 2－3　逻辑右移

3. 算术右移指令

助记符	操作数
SAR	OPRD，m

功能如图 2－4 所示。

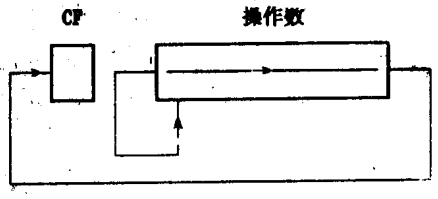


图 2－4　算术右移

（三）循环移位指令

1. 不带进位的左循环移位指令

助记符	操作数
ROL	OPRD，m

功能如图 2－5 所示。

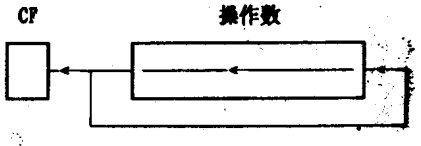


图 2－5　不带进位的左循环移位

2. 不带进位的右循环移位指令

助记符	操作数
ROR	OPRD，m

功能如图 2－6 所示。

3. 带进位的左循环移位指令

助记符	操作数
RCL	OPRD，m

功能如图 2－7 所示。

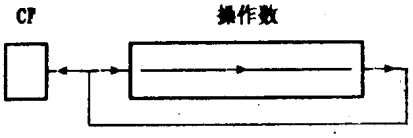


图 2－6　不带进位的右循环移位

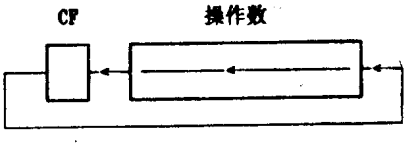


图 2－7　带进位的左循环移位

4. 带进位的右循环移位指令

助记符　　操作数

RCR　　OPRD，m

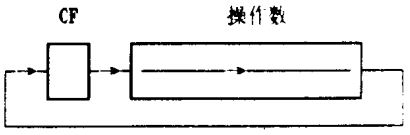


图 2-8　带进位的右循环移位

功能如图 2-8 所示。

说明：如果移位的次数为 1 时，可以直接把移位次数写在指令中，如下所示：

```
SHL AX，1
```

但是，如果移位的次数大于 1，则要把移位次数送 CL 寄存器，在移位指令中用 CL 作为移位次数。如下所示：

```
MOV CL，4
ROR BX，CL
```

二、逻辑指令编程实例

【例 2.12】字母小写变大写

从内存单元 BUF1 开始有一串大小写混合的 ASCII 码字符串，试编程将其中所有小写字母转换为大写字母，转换后的字母放在原字母所在位置。

编程指导：

在计算机中，每一个英文小写字母与其对应的大写字母的 ASCII 码在数值上相差 20H，例如，小写字母“a”的 ASCII 码为 61H（二进制表示为 01100001B），大写字母“A”的 ASCII 码则为 41H（二进制表示为 01000001B）。从二进制表示形式上来说，每一字母的小写形式的 ASCII 码第 5 位都为“1”，而相应大写字母 ASCII 码的第 5 位都为“0”。所以将小写字母转换为大写字母的方法是，用 AND 指令把小写字母 ASCII 码的第 5 位置为“0”即可。另外，由于字符串中既有大写字母，又有小写字母，还有其它的字符，为了避免对其它字符进行变换处理，在进行变换操作之前，有必要对所有字符的 ASCII 码值进行范围判别，只对其 ASCII 码在 61H～7AH 之间的字母进行处理。

程序清单

```
STACK   SEGMENT STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
BUF1    DB ' DFDdS $ @d * DJDserA&'
COUNT   EQU $ - BUF1
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
```

```
        MOV     DS，AX
        LEA     SI，BUF1
        LEA     DI，BUF1
        MOV     CX，COUNT
RETRY：MOV      AL，[SI]
        CMP     AL，61H          ；与″ a″ 字母的 ASCII 码值进行比较
        JB      NOCHG            ；不是小写字母，转
        CMP     AL，7AH          ；与″ z″ 字母的 ASCII 码值进行比较
        JA      NOCHG            ；不是小写字母，转
        AND     AL，11011111B    ；小写变大写
        MOV     [DI]，AL
NOCHG：INC      SI
        INC     DI
        DEC     CX
        JNZ     RETRY
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

【例 2.13】字母大写变小写

从内存单元 BUF1 开始有一串大小写混合的 ASCII 码字符串，试编程将其中所有大写字母转换为小写字母，转换后的字母放在原字母所在位置。

编程指导：

将大写字母转换为小写字母的方法是，用 OR 指令把大写字母 ASCII 码的第 5 位置为“1”即可。同样，为了避免对其它字符进行不必要的变换处理，在进行变换操作之前，有必要对所有字符的 ASCII 码值进行范围判别，只对其 ASCII 码在 41H～5AH 之间的字母进行处理。

程序清单

```
        LEA     SI，BUF1
        LEA     DI，BUF1
        MOV     CX，COUNT
RETRY：MOV      AL，[SI]
        CMP     AL，41H          ；与″ A″ 字母的 ASCII 码值进行比较
        JB      NOCHG            ；不是大写字母，转
        CMP     AL，5AH          ；与″ Z″ 字母的 ASCII 码值进行比较
        JA      NOCHG            ；不是大写字母，转
        OR      AL，00100000B    ；大写变小写
```

```
        MOV     [DI], AL
NOCHG:  INC     SI
        INC     DI
        DEC     CX
        JNZ     RETRY
```

【例 2.14】 字节逐位反排序

编程将 AL 中的 8 bit 二进制数进行反向排序，结果放在 AH 中，要求排序完成后不破坏 AL 中的数。

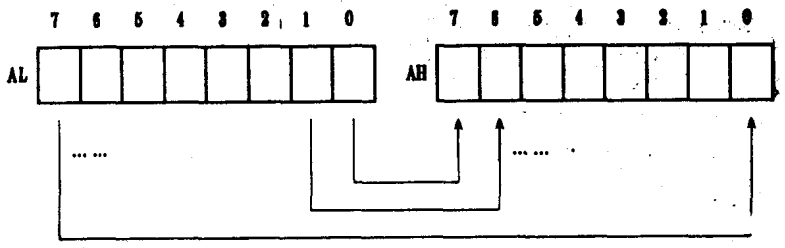


图 2-9 字节逐位反排序示意

编程指导：

反向排序过程如图 2-9 所示。

采用循环程序设计方法，循环体如下：

```
ROL     AL, 1
RCR     AH, 1
```

第一次循环：AL 第 7 位→CF→AH 第 7 位；

第二次循环：AH 第 7 位→AH 第 6 位；AL 第 6 位→CF→AH 第 7 位；

第三次循环：AH 第 6 位→AH 第 5 位；AH 第 7 位→AH 第 6 位；

AL 第 5 位→CF→AH 第 7 位；

依次类推，循环 8 次，即完成反向排序。

程序清单

入口参数：AL＝被反排的数

出口参数：AL＝被反排的数

AH＝反排的结果

```
BITXCHG     PROC    NEAR
            PUSH    CX
            MOV     CX, 8           ; 循环次数送 CX
            MOV     AH, 0           ; 结果单元清 0
RETRY:      ROL     AL, 1           ; AL 第 7 位→CF
            RCR     AH, 1           ; CF→AH 第 7 位
            LOOP    RETRY
```

```
            POP     CX
            RET
BITXCHG     ENDP
```

另　　解：

```
BITXCHG     PROC    NEAR
            PUSH    CX
            MOV     CX，8           ；循环次数送 CX
            MOV     AH，0           ；结果单元清 0
RETRY：     ROR     AL，1           ；AL 第 0 位→CF
            RCL     AH，1           ；CF→AH 第 0 位
            LOOP    RETRY
            POP     CX
            RET
BITXCHG     ENDP
```

【例 2.15】 字节逐位扩展一位

编程将 AL 中的 8 位二进制数每位重复扩展一次，所得结果放在 BX 中。

编程指导：

按题意，要进行如图 2－10 所示的操作。

因为每位都要重复扩展一次，所以首先给 AL 做一复本于 AH 中，将 AL 最高位移进 CF，再由 CF 环移到 BX 第 0 位；紧接着将 AH 最高位移进 CF，再由 CF 环移到 BX 的第 0 位。如此过程循环 8 次即可。

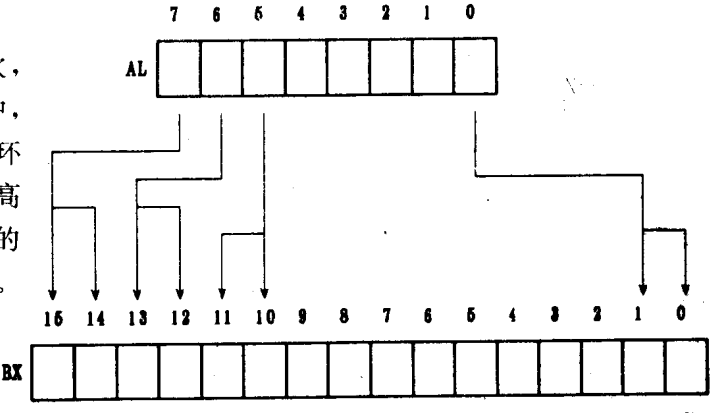


图 2－10　字节逐位扩展示意

程序清单

入口参数：AL＝被扩展的数

出口参数：BX＝扩展的结果

```
REPBIT2     PROC    NEAR
            PUSH    AX
            PUSH    CX
            MOV     AH，AL          ；AL 备份于 AH
            XOR     BX，BX          ；结果单元清 0
            MOV     CX，8           ；循环 8 次
RETRY：     RCL     AL，1
            RCL     BX，1           ；1 bit 送 BX
```

```
        RCL     AH, 1
        RCL     BX, 1           ; 同 1 bit 再送 BX
        LOOP    RETRY
        POP     CX
        POP     AX
        RET
REPBIT2 ENDP
```

【例 2.16】 多字节整体移位

编程把从 BLOCK 开始的内存单元中连续的 4 个字节统一左移 1 位，要求最低字节的第 0 位补 0；较低字节的第 7 位移到较高字节的第 0 位。

编程指导：

设立地址指针指向最低字节，对其进行带进位的左环移，使其第 7 位移至进位 CF，然后对地址较高的下一字节进行相同的移位操作，使 CF 移到其第 0 位，而其第 7 位移至 CF，……，如此进行到第 4 个字节。为了保证最低字节的第 0 位为 0，程序开始时首先对 CF 清 0。

程序清单

```
STACK   SEGMENT   PARA     STACK ' STACK'
        DB        100 DUP (0)
STACK   ENDS
DATA    SEGMENT
BLOCK   DB        23H, 43H, 98H, 10H
COUNT   EQU       $-BLOCK
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA, ES: DATA
MAIN    PROC      FAR
        PUSH      DS
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA
        MOV       DS, AX
        MOV       SI, OFFSET BLOCK
        MOV       CX, COUNT
        CLC
RETRY:  RCL       BYTE PTR [SI], 1
        INC       SI
        LOOP      RETRY
        RET
MAIN    ENDP
```

```
CODE    ENDS
        END     MAIN
```

【例 2.17】 字节奇偶比特对换

编程把 AL 中奇偶比特位（第 0 位与第 1 位、第 2 位与第 3 位，……等）两两对换。

编程指导：

相邻的比特位两两对换，即第 7 位与第 6 位对换；第 5 位与第 4 位对换；第 3 位与第 2 位对换；第 1 位与第 0 位对换。如图 2－11 所示。对换的规律是：奇数位都向左移动一位，而偶数位都向右移动一位。

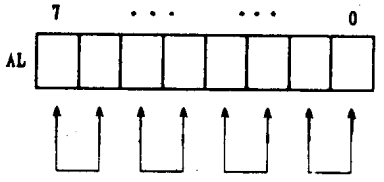


图 2－11 相邻的比特位对换

程序清单

入口参数：AL＝被对换的数

出口参数：AL＝对换结果

```
XCHGBIT PROC    NEAR
        MOV     AH，AL
        AND     AH，10101010B       ；获取奇数位 7，5，3，1
        SHR     AL，1               ；奇数位移至偶数位
        AND     AL，01010101B       ；获取偶数位 6，4，2，0
        AHL     AH，1               ；偶数位移至奇数位
        OR      AL，AH              ；组合结果
        RET
XCHGBIT ENDP
```

【例 2.18】 用移位做乘法

利用移位指令将 AL 中的无符号数乘以 20。

编程指导：

在移位指令中，SAL 每移 1 位，可以将原操作数扩大一倍，相当于原数乘 2；移两位可以使原数扩大三倍，相当于原数乘 4；移 3 位其结果相当于原数乘 8，移 4 位其结果相当于原数乘 16。而 X×20＝X×16＋X×4，所以只要求出 X×4 与 X×16 相加即可。编程中为了防止乘积超出 AL 所能表示的数据范围，首先将 AL 中的数扩展成字。

程序清单

入口参数：AL＝被乘数

出口参数：AX＝乘积

```
ALMUL20 PROC    NEAR
        PUSH    BX
        MOV     AH，0               ；AL 扩展成字
        SAL     AX，1               ；AL×2
        SAL     AX，1               ；AL×4
```

```
        MOV     BX，AX          ；保存 AL×4
        SAL     AX，1           ；AL×8
        SAL     AX，1           ；AL×16
        ADD     AX，BX          ；AL×16+AL×4
        POP     BX
        RET
ALMUL20 ENDP
```

第四节　串操作指令编程实例

一、串操作指令概述

在程序设计中，处理的数据不仅仅是单个字节或字，往往要处理成批连续的字节或字，这些连续的字节（或字）称为字符串。最常见的字符串是 ASCII 码字符串。对字符串的处理一般有这样几种情况：

（1）将从某一地址开始的一个字符串传送至另一地址；

（2）将两个字符串进行比较，看它们是否相等；

（3）在一个字符串中搜索一个特定的字符或字符串；

（4）给一个字符串中各个字节（或字）赋以相同的初值。

为了方便、快速地进行各种字符串的处理，8088 指令系统专门设置了五种基本的串操作指令，它们是：

- MOVS（MOVS 源串，目的串/MOVSB/MOVSW）字符串传送指令
- CMPS（CMPS 源串，目的串/CMPSB/CMPSW）字符串比较指令
- SCAS（SCASB/SCASW）字符串搜索指令
- LODS（LODSB/LODSW）字符串输入指令
- STOS（STOSB/STOSW）字符串输出指令

利用这些指令，我们可以进行字符串的传送、比较、搜索、将字符串中的某一元素取至累加器 AL（或 AX）中，或将累加器 AL（或 AX）中的值存入字符串中。

串操作指令的共同特点：

（1）所有的串操作指令以 DS：SI 为源寻址指针，以 ES：DI 为目的指针，而且在每一操作完成后，自动地修改指针 SI 及 DI。指针修改的规则如表 2－1 所示。

鉴于上述修改规则，在对字符串处理时，若想按照从低地址到高地址的方向（正向）来进行，必须在串操作之前用 CLD 指令将 DF 标志清零，并且将原串的首地址偏移量送 SI，将目的串的首地址偏移量送 DI。而如果想从高地址到低地址的方向（反向）来进行时，必须在串操作之前用 STD 指令将 DF 标志置 1，并且将源串的末地址偏移量送 SI，将目的串末地址偏移量送 DI。

表 2－1　串操作指令指针修改规则

条　件		对 SI/DI 的修改操作
字节串	DF＝0	＋1
字串	DF＝1	－1
字节串	DF＝0	＋2
字串	DF＝1	－2

(2) 有时串操作指令需要重复执行若干次，这时可以给串操作指令加上重复前缀，而重复的次数送 CX 中。例如：

```
MOV     CX, 30
REP     STOSW
```

在 8088 指令系统中，重复前缀有三种形式，见表 2-2。

说明：LODS 指令一般不加重复前缀，因为对该指令加重复前缀无意义。

二、串操作指令编程实例

【例 2.19】分离奇、偶数

设从内存单元 BLOCK 开始有若干个以字节为单位的奇、偶数，试编程把其中的所有奇数传送到 BUF1 开始的连续单元中，把所有的偶数传送到从 BUF2 开始的连续内存单元中。

表 2-2 重复前缀一览表

助 记 符	执 行 条 件
REP	重复串操作直至 CX=0
REPE/REPZ	等于/为 0 时重复串操作，直至 ZF=0 或 CX=0
REPNE/REPNZ	不等于/不为 0 时重复，直至 ZF=1 或 CX=0

编程指导：

判断某数是奇数还是偶数的方法是测试其第 0 位，若第 0 位为“1”，则此数为奇数，否则为偶数。

程序清单

```
STACK   SEGMENT   PARA STACK ' STACK'
        DB        128 DUP (0)
STACK   ENDS
DATA    SEGMENT
BLOCK   DB        20, 31, 63, 80, 44, 50, 23, 130, 77
COUNT   EQU       $ - BLOCK
BUF1    DB        COUNT DUP (0)
BUF2    DB        COUNT DUP (0)
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS: CODE,DS: DATA, ES: DATA
START:            MOV     AX, DATA
                  MOV     DS, AX
                  MOV     ES, AX
                  LEA     SI, BLOCK
                  LEA     DI, BUF1
```

```
                LEA     BX, BUF2
                MOV     CX, COUNT
                CLD
RETRY:          LODSB
                TEST    AL, 00000001B
                JZ      NEXT1                   ; 是偶数，转 NEXT1
                STOSB                           ; 存入奇数
                JMP     SHORT NEXT2
NEXT1:          XCHG    BX, DI
                STOSB                           ; 存入偶数
                XCHG    BX, DI
NEXT2:          LOOP    RETRY
                MOV     AH, 4CH
                INT     21H
CODE    ENDS
        END     START
```

【例 2.20】数据块的传送

编写一个数据块传送程序，能实现对一个数据段中的任意两个地址之间的数据块传送。

编程指导：

在进行数据块传送中需要注意的是：如果两个数据块地址无重叠，则即可以按地址增量方式传送，又可以按地址减量方式传送。但是如果地址有重叠，则就要分析源块首地址与目的块首地址之间的关系，若源块首地址低于目的块首地址，则只能按地址减量方式进行传送。反之，则只能按地址增量方式进行传送。

程序清单

```
                LEA     SI, S_BLOCK     ; 源块首地址→SI
                LEA     DI, O_BLOCK     ; 目的块首地址→DI
                MOV     CX, COUNT       ; 字节数→CX
                CMP     SI, DI          ; 源块首地址高于目的块首地址?
                JA      NEXT2           ; 是，转 NEXT2
                ADD     SI, CX
                ADD     DI, CX
                DEC     SI              ; 源块末地址→SI
                DEC     DI              ; 目的块末地址→DI
                STD                     ; 置减量方向
                JMP     SHORT NEXT2
NEXT1:          CLD                     ; 置增量方向
NEXT2:          LODSB
```

```
        STOSB
        LOOP        NEXT2
```

【例 2.21】字符串查找与替换

编写一个程序，把 BUFFER 缓冲区中所有的“FILENAME”字符串替换成长度相同的“Filename”字符串。

程序清单

```
STACK       SEGMENT     PARA STACK ' STACK'
            DB          128 DUP (0)
STACK       ENDS
DATA        SEGMENT
SOU _ STRING            DB ' FILENAME'
OBJ _ STRING            DB ' Filename'
STRING _ LENGTH         EQU $ - OBJ _ STRING
BUFFER      DB ' FIFILENAME5y $ ygg # G3FggFILEO@ # D? ^ FILENAME + FJI45><>'
BUFFER _ LENGTH         EQU $ - BUFFER
DATA        ENDS
CODE        SEGMENT
            ASSUME CS: CODE, DS: DATA, ES: DATA
START       PROC        FAR
                        PUSH    DS
                        XOR     AX, AX
                        PUSH    AX
                        MOV     AX, DATA
                        MOV     DS, AX
                        MOV     ES, AX
                        CLD
                        LEA     DI, BUFFER              ; 搜索区域首地址→DI
                        MOV     CX, BUFFER _ LENGTH     ; 缓冲区长度→CX
                        MOV     AL, SOU _ STRING
SCAS _ STRING: REPNE    SCASB                           ; 搜索串中的第一个字符
               JNE      EXIT                            ; 未搜索到，转出口
               PUSH     CX
               PUSH     DI
               LEA      SI, SOU _ STRING + 1
               MOV      CX, STRING _ LENGTH - 1
               REPE     CMPSB                           ; 比较后继的字符串是否相等
               POP      DI
```

```
                POP     CX
                JNE     SCAS _STRING                ；后继串不相等，转
                PUSH    CX
                LEA     SI，OBJ _STRING
                DEC     DI
                MOV     CX，STRING _LENGTH
                REP     MOVSB                       ；替换字符串
                POP     CX
                SUB     CX，STRING _LENGTH - 1      ；调整被搜索的字符个数
                JMP     SHORT SCAS _STRING          ；转 SCAS _STRING，继续搜索
EXIT：          RET
START ENDP
        CODE    ENDS
        END     START
```

第五节　程序控制指令编程实例

一、程序控制指令概述

程序控制指令是在程序运行中用以控制程序分支以及循环的指令，按照其测试条件和用途的不同可以把转移指令分为无条件转移指令、条件转移指令、迭代控制指令、中断转移指令以及处理器控制指令（主要是标志操作指令）等几类，每一类中又包括若干条指令。

（一）无条件转移指令 JMP

无条件转移指令是一类特殊的转移指令，这些指令一旦被执行，程序就必然转移至规定的目标。所有的无条件转移指令都要求一个标号作为转移的目标地址。

在 8088 汇编语言指令系统中，无条件转移指令共有下列几种格式：

1. 短无条件转移指令

格式：JMP　SHORT　短标号

功能：(IP) ＝offset　短标号

说明：要求转移指令与目的标号在同一代码段中，并且相互之间的距离在－127～＋127字节之间。

2. 段内直接无条件转移指令

格式：JMP 近标号

功能：(IP) ＝offset　近标号

说明：要求转移指令与目的标号在同一代码段中，并且相互之间的距离在－32K～＋32K字节之间。

3. 段内间接无条件转移指令

格式：JMP　寄存器/存储器（字）

功能：(IP) ＝（寄存器/存储器（字））

说明：要求转移指令与目的标号在同一代码段中，且目的标号的地址必须事先传送至某一寄存器或存储器（字）中。

4. 段间直接无条件转移指令

格式：JMP FAR PTR　　远标号

功能：（CS）＝seg　　远标号

（IP）＝offset　　远标号

说明：转移指令与目的标号一般在不同的代码段中，也可以在同一代码段中。

5. 段间间接无条件转移指令

格式：JMP DWORD PTR 存储器（双字）

功能：（CS）＝（存储器＋2）

（IP）＝（存储器＋0）

说明：转移指令与目的标号一般在不同的代码段中，也可以在同一代码段中。

（二）条件转移指令

条件转移指令是只有在条件满足时，才转移到目标地址的指令。如果条件不满足，程序将按顺序继续执行。所有的条件转移指令都需要一个短标号作为转移目标。条件转移指令一般紧跟在一个算术运算或逻辑运算指令之后，根据运算结果对标志位的影响来测试条件的满足与否。表 2－3 中列出了所有的条件转移指令及其测试的条件。

表 2－3　条件转移指令

指令助记符	测试条件	功能
JA/JNBE	（CF or ZF）＝0	高于/不低于也不等于 0 时，转
JAE/JNB	CF＝0	高于或等于/不低于 0 时，转
JB/JNAE	CF＝1	低于/不高于也不等于 1 时，转
JBE/JNA	（CF or ZF）＝1	低于或等于/不高于 1 时，转
JC	CF＝1	进位为 1 时，转
JCXZ	CX＝0	CX 寄存器等于 0 时，转
JE/JZ	ZF＝1	等于/为 0 时，转
JG/JNLE	（（SF xor OF）or ZF）＝0	大于/不小于也不等于 0 时，转
JGE/JNL	（SF xor OF）＝0	大于或等于/不小于 0 时，转
JL/JNGE	（SF xor OF）＝1	小于/不大于也不等于 1 时，转
JLE/JNG	（（SF xor OF）or ZF）＝1	小于或等于/不大于 1 时，转
JNC	CF＝0	无进位时，转
JNE/JNZ	ZF＝0	不等于/不为 0 时，转
JNO	OF＝0	无溢出时，转
JO	OF＝1	有溢出时，转
JNP/JPO	PF＝0	非奇/为偶时，转
JP/JPE	PF＝1	为奇/非偶时，转
JNS	SF＝0	符号标志为 0/是偶数时，转
JS	SF＝1	符号标志为 1/是负数时，转

（三）迭代控制指令

迭代控制指令一般用在循环程序中循环体的前面或后面，根据条件的满足与否决定循环体是否继续执行。所有的迭代控制指令都以 CX 为循环计数器。

表 2－4 列出了所有的迭代控制指令。和条件转移指令一样，它们需要一个标号用以指定条件满足时转移的目标地址，而且转移目标是一个短标号。

表 2－4 迭代控制指令

指令助记符	执行动作
LOOP	CX－1→CX，若 CX≠0 转，否则执行下一条指令
LOOPE/LOOPZ	CX－1→CX，若 CX≠0 并且 ZF＝1 转，否则执行下一条指令
LOOPNE/LOOPNZ	CX－1→CX，若 CX≠0 并且 ZF＝0 转，否则执行下一条指令

迭代控制指令的使用方法是：当需要重复执行一段程序时，把重复执行的次数送 CX 寄存器，然后执行循环体，最后在该循环体的结束处置一条迭代控制指令，迭代控制指令后面的标号是循环体中第一条指令的标号。

（四）中断指令

中断指令可分为中断调用指令和中断返回指令两种类型：

1. 中断调用指令

   INT 中断功能号

2. 中断返回指令

   IRET

（五）标志操作指令

8088 指令系统提供了对进位标志 C、方向标志 D 以及中断标志 I 等三个标志位进行操作的 7 条指令，它们分别是：

（1）CLC 清进位标志（0→CF）；

（2）CMC 进位标志取反（NOT（CF）→CF）；

（3）STC 置进位标志（1→CF）；

（4）CLD 清方向标志（0→DF）；

（5）STD 置方向标志（1→DF）；

（6）CLI 清中断允许标志（0→IF）；

（7）STI 置中断允许标志（1→IF）。

二、程序控制指令编程实例

【例 2.22】JA 与 JG、JB 与 JL 的区别

下面的程序用于说明无符号数相互之间进行比较时，应该使用的转移指令是 JA 与 JB；而有符号数相互之间进行比较时，应该使用的转移指令则是 JG 与 JL。

编程指导：

在 8088 汇编语言指令系统中，对于无符号数的比较和有符号数的比较分别设置了两组转移指令。

对于无符号数相互之间进行的比较，结果的大小只能采用 JA/JNBE（高于转/不低于也不

等于转）或JAE/JNB（高于且等于转/不低于转）或JB/JNAE（低于转/不高于也不等于转）等转移指令来判别。从另外一个角度来说，即只要对比较的结果采用这一组转移指令来实现分支，则说明是把参加比较的数据当作无符号数来处理。

对于有符号数相互之间进行的比较，结果的大小只能采用JG/JNLE（大于转/不小于也不等于转）或JGE/JNL（大于且等于转/不小于转）或JL/JNGE（小于转/不大于也不等于转）等转移指令来判别。从另外一个角度来说，即只要对比较的结果采用这一组转移指令来实现分支，则说明是把参加比较的数据当作有符号数来处理。

由于上述原因，下面的程序执行完成后，其结果是：AL=0AH；BL=0FFH；CL=01；DL=0。而不可能是另一种结果。

程序清单

```
STACK   SEGMENT PARA STACK ' STACK'
        DB 128 DUP (0)
STACK   ENDS
DATA    SEGMENT
BUF     DB 0AH, 0BH, ' A', ' B', 0FFH, 0
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
START:  MOV     AL, BUF          ; 0AH→AL
        MOV     BL, BUF+4        ; 0FFH (-1) →BL
        CMP     BL, AL
        JAE     NEXT0
        MOV     CL, 0
        JMP     NEXT1
NEXT0:  MOV     CL, 1
NEXT1:  CMP     BL, CL
        JGE     NEXT2
        MOV     DL, 0
        JMP     NEXT3
NEXT2:  MOV     DL, 1
NEXT3:  RET
MAIN    ENDP
```

```
CODE    ENDS
        END     MAIN
```

【例 2.23】CF 标志位的一般用法

下面一个程序完成把从 BUF1 开始的 4 个 ASCII 码转换为对应的十六进制数。在转换过程中，一旦发现有不合法（ASCII 码不在 30H～39H 及 41H～46H 之间）的 ASCII 码，则程序显示“数据不合法!”并返回 DOS；若转换正确，则把转换结果送 BUF2 开始的内存单元中，并显示“数据转换正确!”，然后返回 DOS。

编程指导：

在 DOS 的系统功能调用中，CF 标志常常被用来反映子程序执行结果的正确与否的状态。一般情况下，子程序调用完成后，若 CF＝1 则说明该子程序在执行时发生错误；若 CF＝0 则说明该子程序得到正确的执行。

根据这一规则，我们在设计用户自己的子程序时，也可以用 CF 标志来反映子程序执行结果的状态。下面的子程序 CHECK 完成的功能是，判别 SI 所指向的内存单元的 ASCII 码是否在规定的范围中，若在，则把它转换为对应的 16 进制数并置 CF 标志为 0，然后返回；若不在，则置 CF 标志为 1 并返回主程序。这样，主程序每次调用完子程序后，根据 CF 标志的状态即可了解当前的 ASCII 码是否合法，从而决定程序下一步的走向。

程序清单

```
STACK     SEGMENT PARA STACK ' STACK'
DB 256    DUP (0)
STACK     ENDS
DATA      SEGMENT
BUF1      DB ' 0FA4'
LENTH     EQU $ - BUF1
BUF2      DB LENTH DUP (0)
OK _ MSG  DB 13, 10, ' 数据转换正确! $'
ERR _ MSG DB 13, 10, ' 数据不合法! $'
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA, ES: DATA
MAIN      PROC      FAR
          PUSH      DS
          XOR       AX, AX
          PUSH      AX
          MOV       AX, DATA
          MOV       DS, AX
          MOV       ES, AX
          MOV       SI, OFFSET BUF1
```

```
        MOV     DI, OFFSET BUF2
        MOV     CX, LENTH
        CLD
AGAIN:  CALL    CHECK
        JC      ERR_EXIT
        STOSB
        LOOP    AGAIN
        MOV     DX, OFFSET OK_MSG
        JMP     SHORT EXIT
ERR_EXIT:       MOV     DX, OFFSET ERR_MSG
EXIT:   MOV     AH, 9
        INT     21H
        RET
MAIN    ENDP
CHECK   PROC    NEAR
        LODSB
        SUB     AL, 30H
        JB      EXIT1           ; 若 AL<0，则 1→CF 并且转 EXIT1
        CMP     AL, 0AH
        CMC
        JNB     EXIT1           ; 若 0≤AL≤9，则 0→CF 并转 EXIT1
        AND     AL, 5FH         ; 小写变大写
        SUB     AL, 07
        CMP     AL, 10
        JB      EXIT1           ; 若 AL<10，1→CF，转 EXIT1
        CMP     AL, 16
        CMC                     ; 若 AL>16，1→CF；若 10≤AL<16，0→CF
EXIT1:  RET
CHECK   ENDP
CODE    ENDS
        END     MAIN
```

第三章　程序设计的基本方法

尽管在进行计算机程序设计时所使用的语言不同，有BASIC语言、C语言、PASCAL语言、汇编语言等。但是不论何种语言，其程序设计的基本原理是相同的，程序设计中语句之间，指令之间，程序段之间都遵循其一定的逻辑组织结构，汇编语言也不例外。这种逻辑组织结构归纳起来有顺序结构、分支结构和循环结构等三种结构形式，程序设计时采用此三种结构形式编写程序的方法，分别称之为顺序程序设计方法、分支程序设计方法和循环程序设计方法。由于程序是复杂的，相应的程序设计的方法也是复杂的，特别是分支程序设计和循环程序设计都具有许多形式和技巧。本章的主要内容就是讨论分支程序设计和循环程序设计的基本方法和编程技巧。

第一节　顺序程序设计方法

顺序程序设计是最简单、最基本的程序设计方法。所谓顺序，即程序中的指令在执行过程中既没有跳跃，又没有重复，而是从头到尾连续地顺序执行。虽然，在实际程序设计中，往往很少有顺序结构的程序，但无论再大的程序，从局部上来说都是由若干个顺序结构的程序片段所组成。由于顺序结构比较简单，这里仅举一例。

【例3.1】 用加法指令做乘法

下面的一段程序可以将AL中的无符号数乘以10，其运算的结果放在AX中。

编程指导：

用加法指令将AL中的无符号数乘以10的方法是，将被加数自身与自身相加得到2倍的结果和8倍的结果，然后将这个和再进行相加即得到10倍的结果。在加法过程中，为了防止产生溢出，将AL由字节扩展到AX中。

程序清单

```
……
MOV     AH，0
ADD     AX，AX          ；AX×2
MOV     BX，AX          ；保存2倍的结果在BX中
ADD     AX，AX          ；AX×4
ADD     AX，AX          ；AX×8
ADD     AX，BX          ；AX×8+AX×2→AX
……
```

第二节　分支程序设计方法

在许多程序设计中，由于实际问题的需要，程序在执行过程中，往往要根据不同的条件执行一个大程序中的不同的程序段，程序的执行不是顺序地，而是跳跃式的进行，这就是分支程序。要设计分支程序，就必须学会分支程序设计的基本方法。

一、利用比较与转移指令实现分支

利用比较与转移指令实现分支程序设计的方法，是在需要分支的地方用两数比较指令CMP，或串比较指令CMPS，或串搜索指令SCAS等进行分支条件的比较判断，然后利用各种条件转移指令实现分支的程序设计方法。

【例3.2】 分离正、负数

设从BLOCK开始的内存单元中，有若干个以字节为单位的正、负数，试编程将正数存放在从P_DATA开始的内存单元中；而将负数存放于从M_DATA开始的内存单元中。

编程指导：

考虑到原有数据有可能全是负数或全是正数，所以在开辟正、负数的单元时，分别开辟与原数个数相同的存储单元。图3-1是分离正、负数程序的框图。

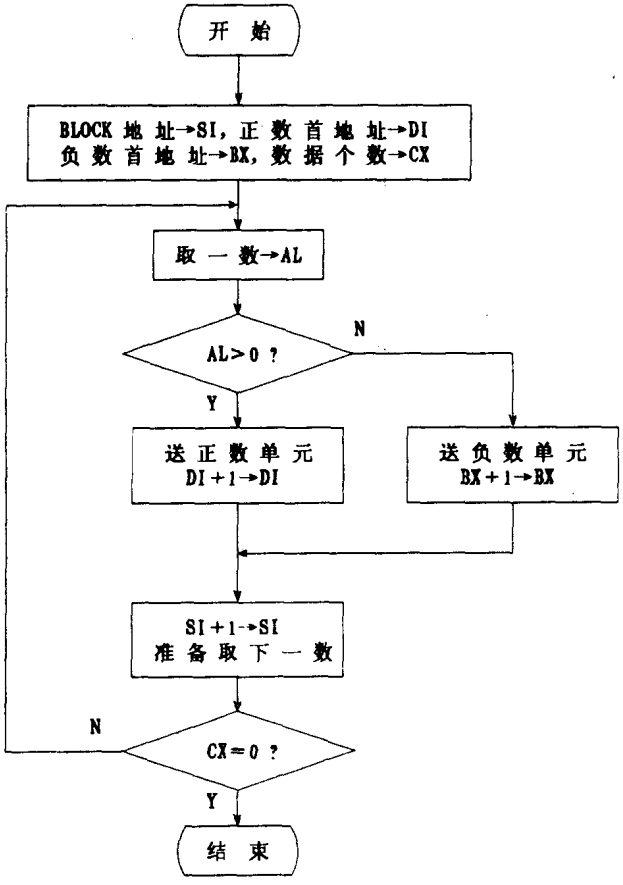


图3-1　分离正、负数算法

程序清单

```
STACK    SEGMENT PARA    STACK' STACK'
         DB      100 DUP (0)
STACK    ENDS
DATA     SEGMENT
BLOCK    DB      -70, 90, -23, -42, 60
COUNT    EQU     $-BLOCK
P_DATA   DB      COUNT DUP (0)
M_DATA   DB      COUNT DUP (0)
DATA     ENDS
CODE     SEGMENT
         ASSUME  CS: CODE, DS: DATA, ES: DATA
MAIN     PROC    FAR
         PUSH    DS
         XOR     AX, AX
         PUSH    AX
         MOV     AX, DATA
         MOV     DS, AX
         MOV     ES, AX
         MOV     SI, OFFSET BLOCK          ; 数据首地址→SI
         MOV     DI, OFFSET P_DATA         ; 正数首地址→DI
         MOV     BX, OFFSET M_DATA         ; 负数首地址→BX
         MOV     CX, COUNT                 ; 计数值→CX
RETRY:   MOV     AL, [SI]                  ; 取一数→AL
         CMP     AL, 0
         JGE     LABEL1                    ; 是正数，转 LABEL1
         MOV     [BX], AL                  ; 存负数
         INC     BX                        ; 地址加 1
         JMP     MEND
LABEL1:  MOV     [DI], AL                  ; 存正数
         INC     DI                        ; 地址加 1
MEND:    INC     SI                        ; 原数据地址加 1
         LOOP    RETRY                     ; 未处理完，转 RETRY
         RET
MAIN     ENDP
CODE     ENDS
         END     MAIN
```

二、利用测试与转移指令实现分支

利用测试与转移指令实现分支程序设计的方法，是在需要分支的地方用逻辑测试指令TEST进行分支条件的测试判断，然后利用各种条件转移指令实现分支的程序设计方法。测试指令不同于比较指令的原因是，测试指令是逻辑“与”运算，而比较指令是算术减法运算。用测试与转移指令实现分支的方法一般适合于用二进制位控制分支的程序设计，而用比较与转移指令实现分支的方法一般适合于用字节或字中的数控制分支的程序设计。

【例 3.3】 统计奇数的个数

设从BLOCK开始的内存单元中有若干以字节为单位的奇偶数，试编程统计其中奇数的个数放于M_DATA单元中。

编程指导：

判断某数是奇数还是偶数的方法是：测试其第0位是1还是0。第0位是1，表明原数是奇数；若第0位是0，则表明原数是偶数。而测试第0位是0还是1的最简单的方法是利用测试指令TEST。

程序清单

```
STACK    SEGMENT PARA     STACK' STACK'
         DB      100 DUP (0)
STACK    ENDS
DATA     SEGMENT
BLOCK    DB      51, 30, 43, 107, 89, 90, 200, 197, 70, 64
COUNT    EQU     $-BLOCK
M_DATA   DB      COUNT DUP (0)
DATA     ENDS
CODE     SEGMENT
         ASSUME  CS: CODE, DS: DATA, ES: DATA
MAIN     PROC    FAR
         PUSH    DS
         XOR     AX, AX
         PUSH    AX
         MOV     AX, DATA
         MOV     DS, AX
         MOV     ES, AX
         MOV     SI, OFFSET BLOCK          ; 数据首地址→SI
         MOV     CX, COUNT                 ; 数据个数→CX
RETRY:   MOV     AL, [SI]                  ; 取一数→AL
         TEST    AL, 00000001B             ; 测试第0位
         JZ      MEND                      ; 是偶数，转MEND
         INC     M_DATA                    ; 奇数个数加1
MEND:    INC     SI                        ; 原数据地址加1
```

```
        LOOP    RETRY                          ；未处理完，转 RETRY
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

三、利用入口地址表实现分支

利用入口地址表实现分支程序设计的方法，是将主程序中的各个子程序入口地址按照一定的顺序存放在内存区域中，这一存放子程序入口地址的区域叫做入口地址表。主程序在运行过程中，依赖于有规律的一组键盘命令来执行不同的子程序。主程序中编有一个主循环程序，它不断地检测键盘输入的命令，当接收到一个命令时，根据一定的算法计算出该命令对应的子程序地址在入口地址表中的位置，从而获得相应子程序的入口地址，然后利用过程调用指令（CALL）转入该子程序中去执行。当执行完相应的子程序后，通过过程返回指令（RET）返回到主程序，等待用户键入新的命令。

【例 3.4】 设某程序中包含有 5 个子程序 SUBP1、SUBP2、SUBP3、SUBP4、SUBP5。试编写一个程序完成如下功能：程序循环等待键盘输入，并且

（1）当按下 F1 键时，执行子程序 SUBP1，执行完成后返回循环等待状态；

（2）当按下 F2 键时，执行子程序 SUBP2，执行完成后返回循环等待状态；

（3）当按下 F3 键时，执行子程序 SUBP3，执行完成后返回循环等待状态；

（4）当按下 F4 键时，执行子程序 SUBP4，执行完成后返回循环等待状态；

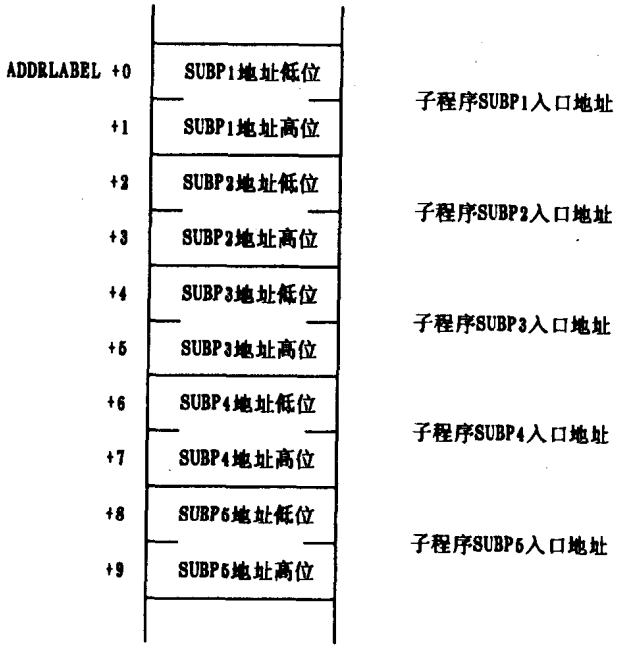


图 3-2　子程序入口地址表

（5）当按下 F5 键时，执行子程序 SUBP5，执行完成后返回循环等待状态；

（6）当按下 ESC 键时，程序运行结束。按其它任意键，程序不响应。

编程指导：

我们不妨把完成 F1～F5 键相应功能的子程序入口地址放在一起，组成一个入口地址表，如图 3-2 所示。

程序设计中，只要判断所按的键属于 F1～F5 键之一，即可想办法从该入口地址表中查出相应的入口地址，转去执行即可。问题的关键在于如何求出各个键的服务程序入口地址的地

址，由图 3-2 可以看出：

入口地址的地址＝表基地址＋偏移量

这里，表基地址是已知的（ADDRLABEL），只要求出各个键服务程序入口地址在表中相对于表基地址的偏移量即可。因每个服务程序的入口地址在表中占有两个字节，而且 F1～F5 的键盘扫描码分别为 3BH，3CH，3DH，3EH，3FH，很有规律。所以各入口地址偏移量可由：（键盘扫描码－3BH）×2 求得。这样，入口地址的地址即可求得。

程序清单

```
……
WAIT _ INPUT:
        MOV     AH, 0
        INT     16H                     ; 等待键盘输入
        CMP     AL, 1BH                 ; 判是否为 ESC 键
        JNZ     NEXT1                   ; 不是 ESC 键，转 NEXT1
EXIT:   RET                             ; 主程序出口
NEXT1:  CMP     AH, 3BH
        JB      WAIT _ INPUT            ; 比 F1 键码小，转 WAIT _ INPUT
        CMP     AH, 3FH
        JA      WAIT _ INPUT            ; 比 F5 键码大，转 WAIT _ INPUT
        SUB     AH, 3BH
        MOV     AL, AH
        XOR     AH, AH
        SHL     AX, 1                   ; (键盘扫描码－3BH) ×2
        MOV     BX, OFFSET ADDRLABEL
        ADD     BX, AX                  ; 求出入口地址的地址→BX
        CALL    WORD PTR CS:[BX]        ; 调服务程序
        JMP     WAIT _ INPUT
; 定义入口地址表
ADDRLABEL       DW SUBP1
                DW SUBP2
                DW SUBP3
                DW SUBP4
                DW SUBP5
; 各个子程序
SUBP1   PROC    NEAR
        ……
        RET
SUBP1   ENDP
SUBP2   PROC    NEAR
```

```
        ……
        RET
SUBP2   ENDP
SUBP3   PROC    NEAR
        ……
        RET
SUBP3   ENDP
SUBP4   PROC    NEAR
        ……
        RET
SUBP4   ENDP
SUBP5   PROC    NEAR
        ……
        RET
SUBP5   ENDP
……
```

说明：上述程序设计方法被广泛地应用于各种系统软件及应用软件的程序设计中，例如：系统软件 EDLIN. COM 及 DEBUG. COM 就采用此种方法实现分支。现以 DEBUG. COM 为例来说明这一点。在 DEBUG. COM 程序中，设计了 A，C，D，E，F，……，等十几个命令，DEBUG. COM 在执行这些命令时，即采用了定义入口地址表的方法来实现分支，下面是利用 U 命令对 DEBUG. COM（V2. 40 15237 字节）进行反汇编后得到的一段程序。

```
1174：0262 AC          LODSB                ；取一命令字符→AL
1174：0263 2C41        SUB     AL，41       ；减去基值 41H
1174：0265 720F        JB      0276         ；AL<′ A′，是非法命令，转 0276
1174：0267 3C19        CMP     AL，19
1174：0269 770B        JA      0276         ；AL>′ Z′，是非法命令，转 0276
1174：026B D0E0        SHL     AL，1        ；求偏移值：(ASCII 码－41H) ×2
1174：026D 98          CBW                  ；扩展到 AX 中
1174：026E 93          XCHG    BX，AX       ；AX→BX
1174：026F 2E          CS：
1174：0270 FF97C903    CALL    [BX＋03C9]   ；调用子程序
1174：0274 EBC5        JMP     023B         ；转主菜单
1174：0276 E9CF03      JMP     0648         ；转错误处理
```

从 1174：03C9 偏移地址开始，到 1174：03FC 偏移地址为止，DEBUG. COM 定义了所有的 26 个英文字母对应的服务程序入口地址表。在此表中，对于 DEBUG. COM 未定义的命令字母如 C，J 等，其入口地址所指向的程序是一段错误程序处理。如下所示：

```
1174：03C9    129C    0648    1243    0482    06BC ……
```

四、利用转移指令表实现分支

利用转移指令表实现分支程序设计的方法，是在数据段中开辟一块内存区域，在此内存区域中顺序存放一系列无条件转移指令，这些转移指令分别是转向主程序中各个子程序服务程序的转移指令。主程序在运行过程中，也是依赖于有规律的一组键盘命令来执行不同的子程序。主程序中有一个主循环程序，它不断地检测键盘输入的命令，当接收到一个命令时，根据一定的算法计算出该命令对应的子程序转移指令在转移指令表中的位置，转入该位置去执行，从而执行了转移表中相应位置的无条件转移指令，于是转入到相应命令对应的子程序中去执行。当执行完相应的子程序后，子程序的最后又通过一条无条件转移指令返回到主循环程序，等待用户键入新的命令。

【例 3.5】 题目同【例 3.4】

编程指导：

在上例中，我们把各个键的服务子程序以过程的形式定义在代码段中，而把各个过程的入口地址放在一起，组成一个入口地址表。在执行时，根据各个键值，求出其入口地址在表中的地址，用过程调用的方式执行各个子程序。但是，在实际应用中，各个子程序有可能是主程序中的一个程序段，而不是以过程的形式出现，即：子程序的末尾是一条转移指令，而非 RET，这样我们就不能采用上述的方法编写程序。此时，我们可以定义一张指令跳转表，即定义一个转移到各个子程序的转移指令表，而执行每个子程序的时候，也采用转移的方式。定义的指令跳转表如图 3-3 所示。

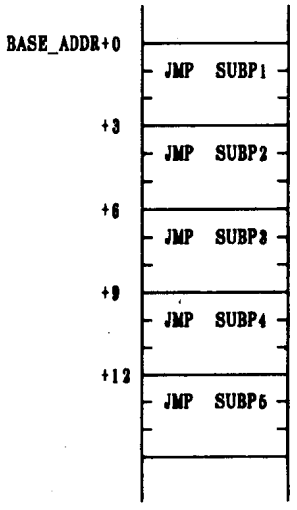


图 3-3 转移指令表

要执行每个子程序，需要计算出每个子程序的跳转指令在表中的地址。从图 3-3 可以看出：跳转指令的地址＝表基地址＋偏移量。值得注意的是，因每个转移指令占有三个字节单元，所以计算偏移量的公式应为：（键盘扫描码－3BH）×3。

程序清单

```
……
WAIT _ INPUT:
        MOV     AH, 0
        INT     16H                 ; 等待键盘输入
        CMP     AL, 1BH             ; 判是否为 ESC 键
        JNZ     NEXT1               ; 不是 ESC 键，转 NEXT1
EXIT:   RET
NEXT1:  CMP     AH, 3BH
        JB      WAIT _ INPUT        ; 比 F1 键码小，转 WAIT _ INPUT
        CMP     AH, 3FH
        JA      WAIT _ INPUT        ; 比 F5 键码大，转 WAIT _ INPUT
        SUB     AH, 3BH
```

```
        MOV     AL，AH
        XOR     AH，AH
        MOV     BX，AX
        SHL     AX，1                    ；（键盘扫描码－3BH）×2→AX
        ADD     AX，BX                   ；（键盘扫描码－3BH）×3→AX
        MOV     BX，OFFSET BASE _ ADDR
        ADD     BX，AX                   ；求出跳转指令的地址→BX
        JMP     BX                       ；调服务程序，入口地址在 BX 中
；定义转移指令表
BASE _ ADDR：
        JMP     SUBP1
        JMP     SUBP2
        JMP     SUBP3
        JMP     SUBP4
        JMP     SUBP5
；各个子程序
SUBP1：  ……
        JMP     WAIT _ INPUT
SUBP2：  ……
        JMP     WAIT _ INPUT
SUBP3：  ……
        JMP     WAIT _ INPUT
SUBP4：  ……
        JMP     WAIT _ INPUT
SUBP5：  ……
        JMP     WAIT _ INPUT
……
```

五、利用关键字-入口地址表实现分支

利用关键字-入口地址表实现分支程序设计的方法，是在数据段中开辟一块内存区域，在此内存区域中顺序存放一系列的“关键字：入口地址”等数据。“关键字”是执行每个子程序的键盘命令代码，而“入口地址”则是相应关键字所对应的子程序入口地址。主程序在运行过程中，也是依赖于一组有规律的关键字来执行不同的子程序。主程序中有一个主循环程序，它不断地检测键盘输入的关键字命令，当接收到一个关键字命令时，通过与“关键字-入口地址”表中的关键字配合比较，查得对应的关键字，然后从关键字的后面获得相应子程序的入口地址，利用过程调用或无条件转移指令转入到相应的子程序中去执行。当执行完相应的子程序后，子程序的最后又通过一条过程返回指令，或者是无条件转移指令返回到主循环程序，等待用户键入新的命令。

【例 3.6】 题目同【例 3.4】

编程指导：

在编程应用中，有时不仅仅把各个服务程序的入口地址定义为一张表，而且在此表中也放入关键字，以便进行分支程序设计。在本例中，关键字即为 F1～F5 各个键的扫描码：3BH，3CH，3DH，3EH，3FH。对于这种表来说，表的存放结构有两种。

第一种结构是：关键字 1，入口地址 1；关键字 2，入口地址 2；关键字 3，入口地址 3，……。如图 3-4 所示。

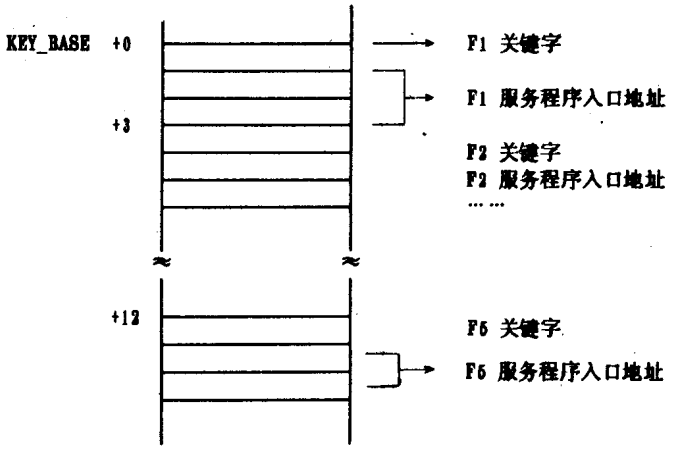


图 3-4　关键字-入口地址表结构 1

第二种结构是：先顺序存放各个关键字，再顺序存放各个入口地址。即：关键字 1，关键字 2，关键字 3，……；入口地址 1，入口地址 2，入口地址 3，……。如图 3-5 所示。

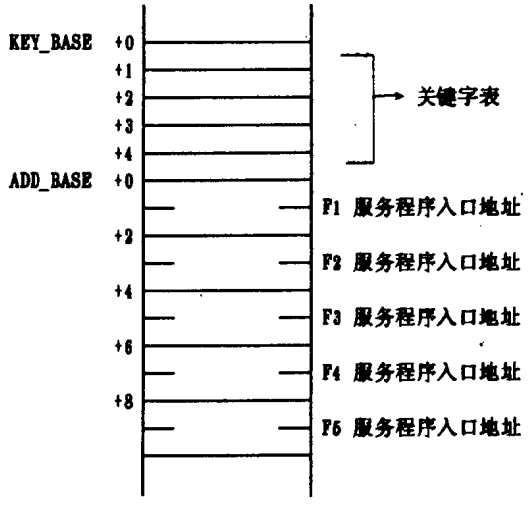


图 3-5　关键字-入口地址表结构 2

程序清单

(1) 对应于第一种数据结构的程序设计方法

……

```
WAIT _ INPUT:
            MOV     AH, 0
            INT     16H                     ; 等待键盘输入
            CMP     AL, 1BH                 ; 判是否为 ESC 键
            JNZ     NEXT1                   ; 不是 ESC 键，转 NEXT1
EXIT:       RET
NEXT1:      CMP     AH, 3BH
            JB      WAIT _ INPUT            ; 比 F1 键码小，转 WAIT _ INPUT
            CMP     AH, 3FH
            JA      WAIT _ INPUT            ; 比 F5 键码大，转 WAIT _ INPUT
            MOV     BX, OFFSET KEY _ BASE   ; 表的首地址→BX
NEXT2:      CMP     AH, CS:[BX]             ; 比较关键字
            JNZ     NEXT3                   ; 与当前关键字不等，转 NEXT3
```

```
        CALL    WORD PTR CS:[BX+1]      ;调用子程序
        JMP     WAIT_INPUT
NEXT3:  ADD     BX,3                    ;BX 指向下一关键字地址
        JMP     NEXT2                   ;转 NEXT2,与下一关键字比较
;定义关键字-入口地址表
KEY_BASE:
        DB      3BH
        DW      SUBP1
        DB      3CH
        DW      SUBP2
        DB      3DH
        DW      SUBP3
        DB      3EH
        DW      SUBP4
        DB      3FH
        DW      SUBP5
;各个子程序
SUBP1   PROC    NEAR
        ……
        RET
SUBP1   ENDP
SUBP2   PROC    NEAR
        ……
        RET
SUBP2   ENDP
SUBP3   PROC    NEAR
        ……
        RET
SUBP3   ENDP
SUBP4   PROC    NEAR
        ……
        RET
SUBP4   ENDP
SUBP5   PROC    NEAR
        ……
        RET
SUBP5   ENDP
……
```

(2) 对应于第二种数据结构的程序设计方法

```
……
WAIT _ INPUT:
        MOV     AH, 0
        INT     16H                         ; 等待键盘输入
        CMP     AL, 1BH                     ; 判是否为 ESC 键
        JNZ     NEXT1                       ; 不是 ESC 键，转 NEXT1
EXIT:   RET
NEXT1:  CMP     AH, 3BH
        JB      WAIT _ INPUT                ; 比 F1 键码小，转 WAIT _ INPUT
        CMP     AH, 3FH
        JA      WAIT _ INPUT                ; 比 F5 键码大，转 WAIT _ INPUT
        MOV     BX, OFFSET KEY _ BASE       ; 关键字表首地址→BX
        MOV     DX, OFFSET ADD _ BASE       ; 入口地址表首地址→DX
NEXT2:  CMP     AH, CS:[BX]                 ; 比较关键字
        JNZ     NEXT3                       ; 与当前关键字不等，转 NEXT3
        MOV     BX, DX                      ; 对应的入口地址→BX
        CALL    WORD PTR CS:[BX]            ; 调子程序
        JMP     WAIT _ INPUT
NEXT3:  INC     BX                          ; 关键字地址加 1
        ADD     DX, 2                       ; 入口地址加 2
        JMP     NEXT2                       ; 转 NEXT2，与下一关键字比较
; 定义关键字-入口地址表
KEY _ BASE:                                 ; 关键字表
        DB      3BH, 3CH, 3DH, 3EH, 3FH
ADD _ BASE:                                 ; 入口地址表
        DW      F1 _ PORT, F2 _ PORT, F3 _ PORT, F4 _ PORT, F5 _ PORT
; 各个子程序
SUBP1   PROC    NEAR
        ……
        RET
SUBP1   ENDP
SUBP2   PROC    NEAR
        ……
        RET
SUBP2   ENDP
SUBP3   PROC    NEAR
        ……
        RET
```

```
SUBP3    ENDP
SUBP4    PROC    NEAR
         ……
         RET
SUBP4    ENDP
SUBP5    PROC    NEAR
         ……
         RET
SUBP5    ENDP
……
```

第三节　循环程序设计方法

一、循环程序的组成部分

循环程序大体上由以下四个部分组成：

(1) 初始化部分：为进入循环作准备，如设置地址指针及循环次数等；

(2) 循环体部分：这是循环程序的主体，由若干条指令组成，循环多少次，这部分语句就执行多少次；

(3) 循环修改部分：负责修改地址指针或变量的内容，以便为下一次循环作准备；

(4) 循环控制部分：依据给定的循环次数或循环条件，判断是否继续循环。

上面只是对循环程序的宏观划分。实际问题中，这四个部分是互相联系、不可分割的，一个循环程序往往不可能很清楚地划分为这四个部分。

【例 3.7】一维数组求和

设从内存单元 WORD _ BUF 开始有若干个以字为单位的数，试编程求出它们的和存放于 SUM _ BUF 字单元中。

编程指导：

在计算数组元素个数时，利用 EQU 伪指令，用字节个数 $-WORD _ BUF 除 2 来得到。由于数组元素以字为单位，所以，地址指针修改时每次加 2。程序流程图如图 3-6。

程序清单

```
STACK       SEGMENT   PARA STACK' STACK'
            DB        128 DUP (0)
STACK       ENDS
DATA        SEGMENT
WORD _ BUF  DW        50, 278, 100, 12, 310, 28
COUNT       EQU       ($-WORD _ BUF) /2
SUM _ BUF   DW        0
DATA        ENDS
CODE        SEGMENT
```

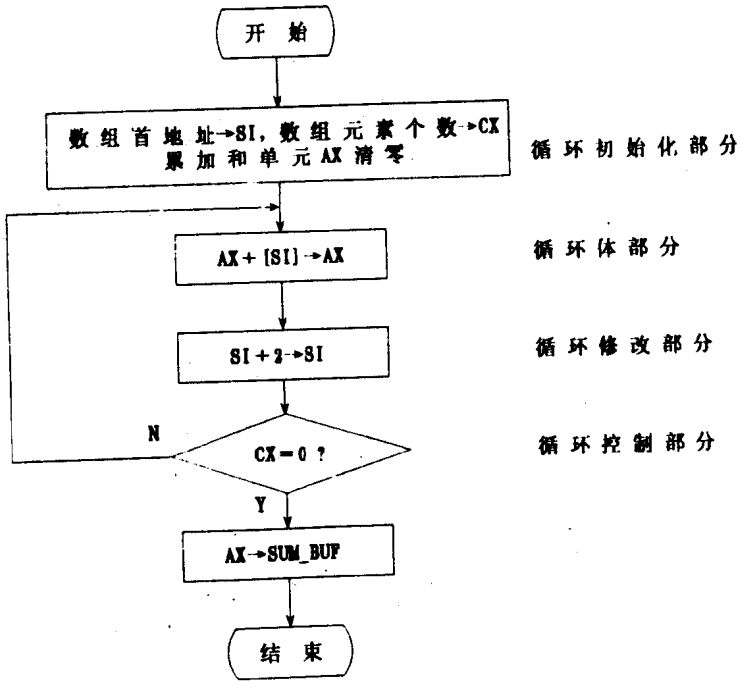


图 3-6　一维数组求和流程图

```
        ASSUME  CS: CODE, DS: DATA, ES: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        LEA     SI, WORD_BUF
        MOV     CX, COUNT          ; 初始化部分
        MOV     AX, 0
RETRY:  ADD     AX, [SI]           ; 循环体部分
        INC     SI
        INC     SI                 ; 循环修改部分
        DEC     CX
        JNZ     RETRY              ; 循环控制部分
        MOV     SUM_BUF, AX
        RET
```

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

二、循环程序的结构形式

前面讲过，循环程序可以划分为初始化、循环体、循环修改及循环控制等四个组成部分，实际问题的不同，在编写程序中，这四个部分的前后顺序就可能不同。但不论怎样，都可把它们划分为如下两种结构形式：

1.“先执行、后判断”结构

这种循环程序至少执行一次循环体，第一次循环执行完成后，再根据循环计数值或循环条件判断是否继续第二次循环。如图 3-7 a 所示。

2.“先判断、后执行”结构

这种循环程序在执行循环体之前，首先判断一下循环条件是否满足。若满足，则执行循环体；若不满足，则一次也不执行循环体。如图 3-7 b 所示。

实际问题中，究竟采用哪种结构形式进行编程，需要根据具体问题来决定。对于有可能一次也不执行循环体的循环程序来说，应采用第二种结构。而对于一个循环程序在任何情况下都不会出现循环次数为零的可能时，采用以上任一种结构形式都可以。

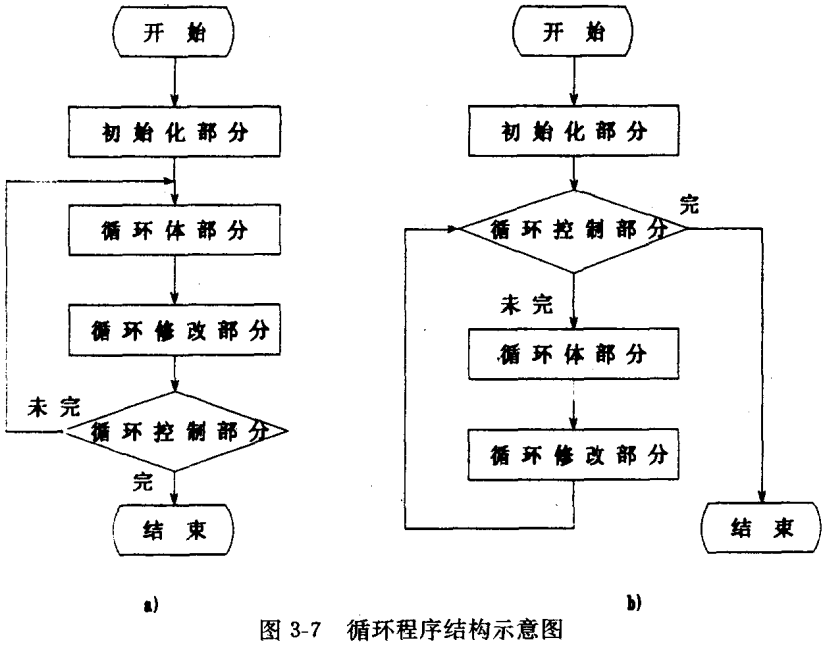


图 3-7　循环程序结构示意图

【例 3.8】统计值为“1”的 bit 位数

设 AX 寄存器中有一个 16 位的二进制数，试编程统计其中值为“1”的位数，统计结果放于 CL 寄存器中。

编程指导：

由于 AX 中的 16 位二进制数有可能全为 0，所以，程序一开始应该首先判断 AX 是否为零，若 AX＝0，则直接转结束。另外，即使 AX 初值不为 0，也有可能在统计若干位后，AX 中剩余未统计的位全为 0，这样也不需要再进入循环进行统计。所以，本程序应采用“先判断、后执行”的结构形式，以便尽量缩短程序执行的时间。

程序清单

```
        ……
        MOV     CL，0           ；计数初值置 0
RETRY：  AND     AX，AX          ；判 AX＝0 否
        JZ      EXIT _ PORT     ；若 AX＝0，转结束
        ROL     AX，1
        JNC     RETRY           ；当前位不为 1，转
        INC     CL              ；计数加 1
        JMP     RETRY
        ……
```

【例 3.9】最大次数的单循环延时程序

编写一个单循环延时程序，要求循环的次数最大。

编程指导：

8088 一次所能处理的数最长为一个字，一个字所能存放的最大的无符号数为 0FFFFH，其相当于十进制数的 65535。如果程序的循环次数设置为 0FFFFH，则延时程序只能循环 65535 次。实际上，我们可以给计数赋初值 0，而采取先执行后判断的结构形式编写一个能进行 65536 次循环的延时程序，这是单循环的最大次数。对于这样的程序只能采用先执行后判断的结构形式，否则程序一开始即跳出循环。

程序清单

```
        ……
        MOV     CX，0
DELAY：  NOP
        DEC     CX
        JNZ     DELAY
        ……
```

三、控制循环的方法

在循环程序设计中，控制循环的方法可分为：计数控制法、条件控制法以及计数条件的综合控制法三种，而计数控制法又可分为正计数控制法与倒计数控制法两种。所谓正计数控制法，即控制循环的计数值从一个小的初始值“0”或“1”开始不断地计数，直到某一给定的值为止，停止循环；所谓的倒计数法，即控制循环的计数值从一个给定的最大值开始不断

地递减，直到计数值为“0”，控制循环结束。

在实际问题中，对于循环次数事先无法确定的循环程序来说，应该采用条件控制法控制循环；而对于循环次数事先可以确定的问题来说，一般采用计数控制法。而有的问题需要综合采用以上两种方法控制循环。下面分别举例说明。

（一）计数控制法编程实例

计数控制法是按照某一计数值控制循环程序执行次数的一种方法，它适合于循环次数事先确定的情况下。

【例 3.10】奇偶比特归并排列

编程把 AL 中的八位二进制数进行重新排列，要求将源 AL 中的第 76543210 各位按 75316420 位次排列，结果放于 AH 中。其过程如图 3-8 所示。

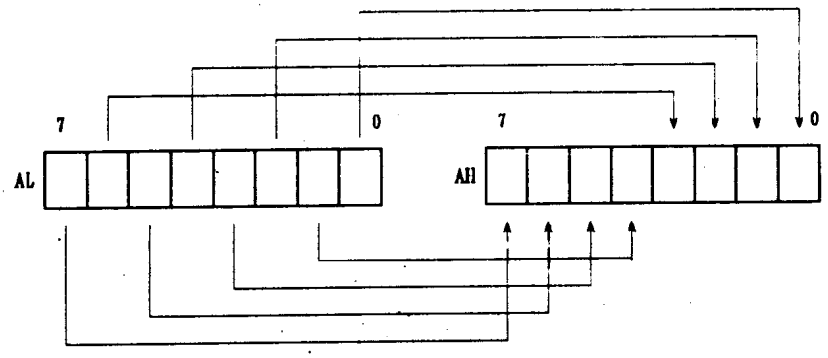


图 3-8　奇偶比特归并排列

编程指导：

从题意可以看出：奇数位依次排在 AH 的高半字节，而偶数位依次排在 AH 的低半字节。程序设计中采用两个并列的单循环，第一个循环完成把 AL 中的四个奇数位（位 7、5、3、1）顺次移到 AH 中。本循环执行四次循环体，每一次循环，把一个奇数位中的值移到 AH 中。然后，甩掉一个偶数位的值。这样，本循环执行完成后，即把 AL 中的四个奇数位移到 AH 中的低半字节。第二个循环程序完成把 AL 中的四个偶数位（位 6、4、2、0）顺次移到 AH 中。本循环也执行四次循环体，每执行一次循环体，首先甩掉一个奇数位的值。然后，把一个偶数位中的值移进 AH 中。这样，执行四次循环后，即把 AL 中的四个偶数位移进 AH 的低半字节，而原低半字节中的值移到 AH 的高半字节。

程序清单

```
        ......
        MOV     AH，0
        MOV     CX，4
RETRY1：ROL     AL，1
        RCL     AH，1          ；移进一个奇数位到 AH 中
        ROL     AL，1          ；甩掉 AL 中的一个偶数位
        LOOP    RETRY1         ；循环把四个奇数位移到 AH 中
```

```
RETRY2:   ROL    AL, 1          ; 甩掉 AL 中一个奇数位
          ROL    AL, 1
          RCL    AH, 1          ; 移进一个偶数位到 AH 中
          INC    CX
          CMP    CX, 4
          JNE    RETRY2         ; 循环把四个偶数位移到 AH 中
          ……
```

说明：本程序中的第一个循环采用倒计数法控制循环，第二个循环采用正计数法控制循环，第二个循环的计数初值直接利用第一个循环的执行结果。

（二）条件控制法

条件控制法是在循环程序执行过程中，根据某一条件是否成立控制循环程序是否继续执行的方法。

【例 3.11】字符传送

内存单元 BUF1 开始的缓冲区中有一字符串，字符串中的字符个数不定，但其中必有且仅有一个“$”字符。试编程把该字符串中“$”字符前面的所有字符，传送到 BUF2 开始的连续内存单元中。

编程指导：

由于传送的字符个数不能确定，只能根据是否遇到“$”字符来决定是否结束循环。所以，本程序应该采用条件控制法来控制循环。

程序清单

```
STACK     SEGMENT PARA STACK' STACK'
          DB      128 DUP (0)
STACK     ENDS
DATA      SEGMENT
BUF1      DB      ' bt#f$y@rrvl; n'
BUF2      DB      100 DUP (0)
DATA      ENDS
CODE      SEGMENT
          ASSUME  CS: CODE, DS: DATA, ES: DATA
MAIN      PROC    FAR
          PUSH    DS
          XOR     AX, AX
          PUSH    AX
          MOV     AX, DATA
          MOV     DS, AX
          MOV     ES, AX
          LEA     SI, BUF1
```

```
                LEA     DI, BUF2                    ; 初始化部分
RETRY1:         MOV     AL, [SI]
                CMP     AL,' $'
                JZ      EXIT _ PORT                 ; 循环控制部分
                MOV     [DI], AL                    ; 循环体部分
                INC     SI
                INC     DI                          ; 循环修改部分
                JMP     RETRY1
EXIT _ PORT:    RET
MAIN            ENDP
CODE            ENDS
                END     MAIN
```

（三）计数、条件的综合控制法

在某此应用程序中，由于实际问题的特殊性，控制循环次数的方法单独使用计数控制法或者是条件控制法都不能圆满地解决循环控制问题，此时就要采用计数、条件的综合控制法。

【例 3.12】 接收字符并打印

编写一个程序，要求接收键盘输入的汉字与字符混合的字符串，最多 100 字符，存放于从内存单元 CHAR _ BUF 开始的连续的单元中；若接收到“Esc”键（1BH）或缓冲区满，则停止接收；然后提示用户准备好打印机，并等待用户按任意键回答就绪信号，最后将字符串中的所有汉字打印出来。

编程指导：

把接收键盘输入编成单重循环形式，而控制循环结束的方法有两种可能：一种可能是缓冲区未满，但用户按了“Esc”键，这时必须结束循环；另一种可能是用户未按“Esc”键，但缓冲区已满，这时也必须结束循环。用第一种方法结束循环只能采用条件控制法，而用第二种方法控制循环只能采用计数控制法。这样，要兼顾两种可能，就必须在每次循环中采用条件计数的综合控制方法来控制循环。在两个可能的条件中，只要有一个满足，就结束循环。

程序清单

```
STACK     SEGMENT  PARA STACK' STACK'
                   DB 128 DUP (0)
STACK     ENDS
DATA      SEGMENT
INPUT _ MSG        DB 13, 10,' 请输入字符串：$'
CHAR _ BUF         DB 100 DUP (0)
READY _ MSG        DB 13, 10,' 请准备好打印机，然后按任意键……$'
DATA      ENDS
CODE      SEGMENT
```

```
        ASSUME  CS: CODE, DS: DATA, ES: DATA
START   PROC    FAR
                PUSH    DS
                XOR     AX, AX
                PUSH    AX
                MOV     AX, DATA
                MOV     DS, AX
                MOV     ES, AX
                MOV     DX, OFFSET INPUT_MSG
                MOV     AH, 9
                INT     21H
                MOV     CX, 100
                MOV     SI, OFFSET CHAR_BUF
NEXT1:          MOV     AH, 1
                INT     21H
                MOV     [SI], AL
                INC     SI
                CMP     AL, 1BH
                JE      NEXT2
                LOOP    NEXT1
NEXT2:          MOV     DX, OFFSET READY_MSG
                MOV     AH, 9
                INT     21H
                MOV     AH, 8
                INT     21H
                MOV     CX, 100
                LEA     SI, CHAR_BUF
RETRY:          MOV     DL, [SI]
                INC     SI
                CMP     DL, 0A1H
                JB      NEXT3
                MOV     AH, 5
                INT     21H
                JMP     TEST_END
NEXT3:          CMP     DL, 1BH
                JZ      EXIT
TEST_END:       LOOP    RETRY
EXIT:           MOV     DL, 0DH
                MOV     AH, 5
```

```
                INT     21H
                MOV     DL, 0AH
                INT     21H
                RET
START           ENDP
CODE            ENDS
                END     START
```

四、多重循环程序设计举例

一个循环体中又套有另外一个或几个循环体，这种程序叫做多重循环程序。最常见的多重循环程序有二重循环和三重循环。

（一）二重循环程序设计实例

【例 3.13】计算 9×9 乘法表

试编程求出 9×9 乘法表的 45 个值，放于 MULT _ BUF 开始的连续内存单元中。

编程指导：

9×9 乘法表如下所示：

```
1×1
1×2     2×2
1×3     2×3     3×3
…       …       …
1×9     2×9     3×9     …       9×9
```

本程序应采用两重循环。外层循环用计数法来控制循环，被乘数值从 1 变化到 9，每执行一次外循环计算出被乘数 n 的 9—n+1 个乘积。内循环也用计数法来控制循环，乘数的初值等于被乘数 n，循环到乘数等于 9 为止。其流程图如图 3-9 所示。

程序清单

```
STACK       SEGMENT STACK' STACK'
            DB 128  DUP (0)
STACK       ENDS
DATA        SEGMENT
MULT _ BUF  DB 45   DUP (0)
DATA        ENDS
CODE        SEGMENT
            ASSUME  CS: CODE, DS: DATA
START       PROC    FAR
            PUSH    DS
            XOR     AX, AX
            PUSH    AX
```

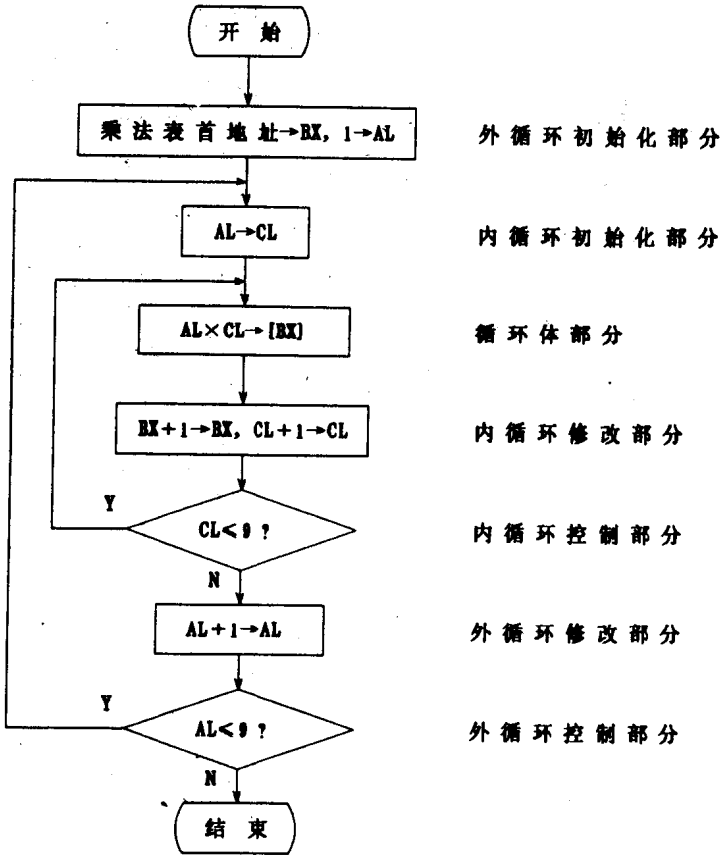


图 3-9　计算 9×9 乘法表的程序流程

```
        MOV     AX，DATA
        MOV     DS，AX
        MOV     BX，OFFSET MULT _ BUF
        MOV     AL，1
RETRY1：MOV     CL，AL
RETRY2：PUSH    AX
        MUL     CL
        MOV     [BX]，AL
        INC     BX
        INC     CL
        POP     AX
        CMP     CL，9
        JLE     RETRY2
        INC     AL
        CMP     AL，9
```

```
        JLE     RETRY1
        RET
START   ENDP
CODE    ENDS
        END     START
```

（二）三重循环程序设计实例

【例 3.14】转置矩阵法加密信息

下面是一种加密文件内容的方法。这种加密方法是将明文信息按 64bit 进行分组，按 8bit（一字节）为一行的方式组成一个 8×8 的方阵，然后将矩阵按列组成 8 字节的密文。例如，明文信息为：COMPUTER，它们的代码是：43H、4FH、4DH、50H、55H、54H、45H、52H。将它们组成 8×8 的方阵其结果如下：

$$X = \begin{bmatrix} 43H \\ 4FH \\ 4DH \\ 50H \\ 55H \\ 54H \\ 45H \\ 52H \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

按列将它们形成密文，其结果为：00H、FFH、00H、1DH、60H、6EH、C1H、EAH。

试编写一个程序，利用转置矩阵加密法，将 HCQ1 开始的 800 个字节明文信息进行加密变换，加密以后的密文信息存放在 HCQ2 开始的 800 个字节单元中。

编程指导：

我们要采用三重循环程序设计方法来编写该程序。在三重循环程序中，最内层的循环被调用一次，其自身要循环 8 次，完成将 8 个字节的同一位组成一字节的密文信息并进行存储；次外层的循环被调用一次，其自身要循环 8 次，完成将 8 个字节的 8 个比特位全部组成密文信息并进行存储；最外层的循环要重复 100 次，完成将 800 个字节的明文信息进行加密变换。

程序清单

```
STACK   SEGMENT PARA STACK' STACK'
        DB      128 DUP (0)
STACK   ENDS
DATA    SEGMENT
HCQ1    DB 800 DUP (?)
LENTH   EQU $-HCQ1
HCQ2    DB 800 DUP (?)
DATA    ENDS
CODE    SEGMENT
```

```
                ASSUME  CS: CODE, DS: DATA, ES: DATA
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     SI, OFFSET HCQ1
        MOV     DI, OFFSET HCQ2
        MOV     CX, LENTH/8         ; 组数送 CX
        CLD
RETRY1: PUSH    CX
        MOV     CL, 1
RETRY2: PUSH    SI                  ; 加密 64bit 明文
        XOR     AH, AH
        MOV     DX, 8
RETRY3: LODSB
        SHL     AL, CL
        RCL     AH, 1
        DEC     DX
        JNZ     RETRY3              ; 8 个字节的同一位未处理完，转
        MOV     AL, AH
        STOSB
        POP     SI
        INC     CL
        CMP     CL, 8
        JBE     RETRY2              ; 64bit 明文未处理完，转 RETRY2
        ADD     SI, 8               ; 地址加 8，加密下一组
        POP     CX
        LOOP    RETRY1
        RET
MAIN    ENDP
CODE    ENDS
        END     MAIN
```

第四章　结构程序设计与参数传递

结构程序设计方法不仅可以使程序结构清楚，而且便于程序的调试、易于阅读和维护。结构程序设计常用的方法有：宏指令、子程序和模块化程序设计等几种。由于在结构程序设计中，各个子程序之间、模块化程序之间，应用程序与 DOS 系统程序之间往往要进行必要的、甚至是大量的参数传递工作，所以掌握参数传递的基本方法对于结构程序设计来说也是相当必要的。本章的主要内容是讨论宏指令设计方法、子程序设计方法、模块化程序设计方法以及参数传递的基本方法。

第一节　宏指令编程实例

在程序设计中，如果有一段程序需要被重复使用若干次，为了避免重复书写这段程序并使源程序结构清楚，可以将该段程序定义为一个宏指令。也就是说，宏指令实际上相当于若干条基本指令的组合，只不过这些指令不是机械地组合，而是有机地组合，是能完成一种特定功能的程序段。

使用宏指令时要注意：

(1) 宏定义可以写在程序的任何地方，但必须在宏调用的前面；

(2) 宏调用时实在参数的个数必须不少于宏定义时形式参数的个数；

(3) 出现在宏定义中的转移标号，必须在宏定义的开始用 LOCAL 伪指令进行说明。

【例 4.1】字符串显示宏指令

程序设计中，经常需要显示一个字符串，为了不重复书写这段程序，可以将显示字符串的这一段程序定义为一个宏指令，而在程序中需要的地方调用该宏指令即可。

编程指导：

考虑到在显示字符串时，不同地方的宏调用可能要显示不同的字符串，所以宏指令必须带有地址宏参数，这样可以使一条宏指令适应于不同字符串的显示需要。

程序清单

```
; 数据段
        ……
MSG1    DB      ' Input SOURCE filename: $'
MSG2    DB      ' Input OBJECT fikename $'
        ……
; 代码段 ……
        ; 宏定义
XSSTRING        MACRO   ADDR
                MOV     AH, 9
```

```
        MOV     DX，OFFSET ADDR
        INT     21H
        ENDM
；主程序
        ……
        XSSTRING MSG1           ；宏调用显示字符串 MSG1
        ……
        XSSTRING MSG2           ；宏调用显示字符串 MSG2
        ……
```

【例 4.2】数据块移动宏指令

假设从内存单元 BUFFER1 开始有一块以字节为单位的、长度为 800 的数据块，试编写一个宏指令将该数据块传送到从 BUFFER2 开始的内存单元中（假设两块数据区不重叠）。

编程指导：

考虑到不同时刻的宏调用，可能在数据块的传送地址及数据传送个数上都有所不同，所以在进行宏定义时，设置了三个形式参数，它们分别是源数据块首地址参数 ADDR1、目标数据块首地址参数 ADDR2 和数据个数参数 COUNT。

程序清单

```
；数据段
        ……
BUFFER1 DB      800 DUP (?)
BUFFER2 DB      800 DUP (?)
        ……
；代码段
        ……
        ；宏定义
MOVSTR  MACRO   ADD1，ADD2，COUNT
        LOCAL   RETRY               ；对宏指令中的标号 RETRY 进行说明
        MOV     SI，OFFSET ADD1
        MOV     DI，OFFSET ADD2
        MOV     CX，COUNT
        CLD
RETRY:  LODSB
        STOSB
        LOOP    RETRY
        ENDM
        ；主程序
        ……
```

```
MOVSTR   BUFFER1，BUFFER2，800    ；传送字符串的宏调用
……
```

第二节　子程序设计实例

一、子程序与过程

如果在一个程序的多个地方或多个程序的多个地方用到了同一段程序，为了避免重复书写这段程序以及节省磁盘或内存空间，可以将这段程序编写为一个特殊的程序，放于程序中，对于需要执行这段程序的程序，只要采用调用指令转移到这段程序去，执行完毕后再返回到原来的程序即可。这段公用的特殊的程序就叫做“子程序”，而调用子程序的程序叫做“主程序”，在汇编语言中一般把“子程序”叫做“过程”。在本书中两种说法都采用。

二、过程的类型、定义及返回

过程具有两种类型，一种是近（NEAR）属性的过程，一种是远（FAR）属性的过程。对应于两种属性，过程的定义就分为近过程的定义、远过程的定义两种定义方式。

过程在执行完成后，需要返回调用自己的主程序，由于过程有近过程、远过程之分，所以对于返回指令就有近返回、远返回之别。在近过程中的返回指令叫近返回，而在远过程中的返回指令叫远返回。

1. 近过程的定义方法

(1) 格式：

```
过程名   PROC    [NEAR]
          …
[标号:] RET     空/n          ；返回指令（其中：n=0，2，4，…）

          …
过程名   ENDP
```

(2) 返回指令的功能：从堆栈段中当前堆栈指针 SP 指向的单元中弹出一个字送 IP，即 (IP) = ((SP)) 并且 (SP) +2→ (SP) 或 (SP) +2+n→ (SP)。n 的含义是把堆栈中的 n/2 个字的内容弹出作废，由于对于堆栈的操作必须以字为单位，所以 n 一般为偶数。例如：RET 4。

2. 远过程的定义方法

(1) 格式：

```
过程名   PROC    FAR
          …
[标号:] RET     空/n          ；返回指令（其中：n=0，2，4，…）

          …
过程名   ENDP
```

(2) 返回指令的功能：从堆栈段中当前堆栈指针 SP 指向的单元中弹出一个字送 IP，再弹

出一个字送 CS，如果返回指令中含有数 n，则再从堆栈中废除 n/2 个字。即（IP）=（(SP)）并且（SP）+2→（SP）；（CS）=（(SP)）并且（SP）+2→（SP）；（SP）+n→（SP）。n 的含义同上。

三、过程调用语句的类型及格式

过程调用分为近过程调用和远过程调用两大类型。对于每一类型又分为直接调用与间接调用两种格式。近调用只能用于调用语句与被调用的过程在同一代码段中的情况；而远调用一般则用于调用语句与被调用的过程在不同的代码段中的情况，并且被远调用的过程必须定义为远过程属性。

1. 近直调用

（1）格式：CALL 近标号

（2）功能：保留返回地址于堆栈中：　(SP)=（SP）−2

((SP))=（过程返回的偏移地址）

实现转移：　(IP)=OFFSET 近标号

注："过程返回的偏移地址"是"CALL 近标号"指令的下一条指令地址。

例：CALL SUB _ PROC（过程标号）

2. 近间调用

（1）格式：CALL 寄存器/存储器（字）

（2）功能：保留返回地址于堆栈中：　(SP)=（SP）−2

((SP))=（过程返回的偏移地址）

实现转移：　(IP)=（寄存器）/（存储器）

例：CALL BX

CALL SUB1

CALL WORD PTR [BX+SI]

3. 远直调用

（1）格式：CALL 远标号

（2）功能：保留返回地址于堆栈中：　(SP)=（SP）−2

((SP))=（过程返回的段地址）

(SP)=（SP）−2

((SP))=（过程返回的偏移地址）

实现转移：　(CS)=SEG 远标号

(IP)=OFFSET 远标号

注："过程返回段地址"是指"CALL 远标号"指令所在段的段地址；"过程返回的偏移地址"是指"CALL 远标号"指令的下一条指令地址。

例：CALL SUB _ PROC（远过程标号）

CALL FAR PTR SUB1

4. 远间调用

（1）格式：CALL 存储器地址标号（双字类型）

（2）功能：保留返回地址于堆栈中：　(SP)=（SP）−2

((SP))=（过程返回的段地址）

(SP)＝（SP）－2
((SP))＝（过程返回的偏移地址）
实现转移：　(CS)＝（存储器地址＋2）
(IP)＝（存储器地址）

例：CALL DWORD PTR SUB1
CALL DWORD PTR [BX+SI]
CALL SUB1（SUB1 是双字类型的存储器地址标号）

四、过程的并列、嵌套与递归

在程序设计中，由于实际问题的需要，一个主程序往往要调用若干个不同的过程或对一个过程要调用若干次。而且过程本身又可以调用自己或别的过程。由于这样，各个过程之间就存在着并列、嵌套和递归的关系。

所谓过程的并列，即在一个过程调用语句的后面紧接着又是一个过程调用语句。例如：

```
CALL    SUB1
CALL    SUB1
```

这里，被调用的过程可以是同一个过程，也可以是不同的过程。

所谓过程的嵌套，即在一个过程内部又要调用执行另外一个过程。只要堆栈空间允许，嵌套的层次不限。例如，下面的子程序 SUB1 中又调用另外一个子程序 SUB2。

```
SUB1    PROC    NEAR
        ...
        CALL    SUB2
        ...
        RET
SUB1    ENDP
```

所谓过程的递归，即过程在执行中又要调用自身。递归调用实际上是过程嵌套的一种特殊情况。例如，下面的子程序 SUB1 在其运行过程中又调用自身。

```
SUB1    PROC    NEAR
        ...
        CALL    SUB1
        ...
        RET
SUB1    ENDP
```

五、子程序设计举例

【例 4.3】近调用语句使用方法

下面的程序完成的功能是把屏幕设置成方式 1（40×25 彩色字符方式），然后在屏幕的中

部由左至右动画显示一个“小太阳”(ASCII 码 0FH)。程序运行时，首先以绿色显示一个“小太阳”，然后延时 50ms，再擦除该位置处的“小太阳”，最后修改光标的列坐标在下一个位置处显示“小太阳”，……，如此循环直到列坐标等于 79 为止。

编程指导：

程序的结构简单，主要目的是说明近直调用、近间调用语句的使用方法。

程序清单

```
CODE1   SEGMENT PARA PUBLIC' CODE'
        ASSUME  CS: CODE1, DS: CODE1
MAIN    PROC    FAR
        PUSH    CS
        POP     DS
        MOV     AH, 0FH
        INT     10H
        PUSH    AX                      ; 保存显示器工作方式
        MOV     AX, 0001H
        INT     10H                     ; 设置方式 1 (40×25 彩色字符方式)
        MOV     DX, 0C00H               ; 光标定位在 (12, 00)
RETRY:  PUSH    DX
        CALL    SET_GB                  ; ①近直调用：设置光标位置子程序
;       MOV     BX, 0002H               ; 选择 0 页，并选择字符颜色为绿色
        MOV     BP, OFFSET ADDR
        CALL    WORD PTR CS: [BP]       ; ②近间调用：显示 ASCII 码为 0FH 的字符
;       MOV     AX, OFFSET DELAY
        CALL    AX                      ; ③近间调用：延时程序
;       MOV     BX, 0000H               ; 选择 0 页，并选择字符颜色为黑色
        CALL    WORD PTR ADDR           ; ④近间调用：擦除当前光标处的字符
;       POP     DX
        INC     DL
        CMP     DL, 79                  ; 是否已到第 79 列
        JNZ     RETRY                   ; 没有，转
        POP     AX
        MOV     AH, 0
        INT     10H                     ; 恢复原有显示方式
        MOV     AH, 4CH
        INT     21H                     ; 返回 DOS
MAIN    ENDP
; 延时 50 毫秒的子程序
DELAY   PROC    NEAR
```

```
        PUSH    CX
        PUSH    DX
        MOV     DX, 5
DELAY1: MOV     CX, 2801
DE10MS: LOOP    DE10MS
        DEC     DX
        JNZ     DELAY1
        POP     DX
        POP     CX
        RET
DELAY   ENDP
;;;;;;;;;;;;;;;;;;;;;;;;
; 光标定位子程序
; 入口参数：DX=光标位置
;;;;;;;;;;;;;;;;;;;;;;;;
SET_GB  PROC    NEAR
        MOV     AH, 2
        MOV     BH, 0
        INT     10H
        RET
SET_GB  ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 在当前光标位置处显示字符
; 入口参数：BH=页号，BL=字符颜色
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
XS_CHAR PROC    NEAR
        MOV     AX, 090FH
        MOV     CX, 1
        INT     10H
        RET
XS_CHAR ENDP
ADDR    DW OFFSET XS_CHAR          ; 显示字符子程序入口地址保存单元
CODE1   ENDS
        END     MAIN
```

【例 4.4】远调用语句使用方法

下面的程序完成与【例 4.3】同样的功能，只是程序在结构上书写的格式不同。

编程指导：

本程序采用多个代码段的程序设计方法，把延时子程序写在 CODE1 代码段中，把光标定

位子程序、显示字符子程序写在CODE3代码段中，并且把这些子程序都定义为远属性的过程，以便不同代码段中的程序对其进行长调用。整个程序的主程序写在CODE2代码段中。程序的结束语句为“END MAIN”，这表明程序第一条执行的指令是CODE2代码段中的MAIN开始的指令。

程序清单

```
CODE1    SEGMENT   PARA PUBLIC' CODE'
         ASSUME    CS: CODE1, DS: CODE2
; 延时50毫秒的子程序
DELAY    PROC      FAR
         PUSH      CX
         PUSH      DX
         MOV       DX, 5
DELAY1:  MOV       CX, 2801
DE10MS:  LOOP      DE10MS
         DEC       DX
         JNZ       DELAY1
         POP       DX
         POP       CX
         RET
DELAY    ENDP
CODE1    ENDS
;
CODE2    SEGMENT   PARA PUBLIC' CODE'
         ASSUME    CS: CODE2, DS: CODE2
ADDR     DD        XS_CHAR              ; 显示字符子程序入口地址保存单元
;
MAIN     PROC      FAR
         PUSH      CS
         POP       DS
         MOV       AH, 0FH
         INT       10H
         PUSH      AX                   ; 保存显示器工作方式
         MOV       AX, 0001H
         INT       10H                  ; 设置方式1（40×25彩色字符方式）
         MOV       DX, 0C00H            ; 光标定位在（12, 00）
RETRY:   PUSH      DX
         CALL      FAR PTR SET_GB       ; ①远直调用：设置光标位置子程序
; 说明：由于SET_GB子程序说明在后，所以该调用语句不能写为“CALL SET_GB”
```

```
;
        MOV     BP, OFFSET ADDR
        MOV     BX, 0002H           ; 选择 0 页，并选择字符颜色为绿色
        CALL    DWORD PTR CS:[BP]; ②远间调用：显示 ASCII 码为 0FH 的字符
;
        CALL    DELAY               ; ③远直调用：延时程序
; 说明：由于 DELAY 子程序说明在前，所以该调用语句勿需写为“CALL FAR PTR DELAY”
;
        MOV     BX, 0000H           ; 选择 0 页，并选择字符颜色为黑色
        CALL    ADDR                ; ④远间调用：擦除当前光标处的字符
;
        POP     DX
        INC     DL
        CMP     DL, 79              ; 是否已到第 25 行
        JNZ     RETRY               ; 没有，转
        POP     AX
        MOV     AH, 0
        INT     10H                 ; 恢复原有显示方式
        MOV     AH, 4CH
        INT     21H                 ; 返回 DOS
MAIN    ENDP
CODE2   ENDS
;
CODE3   SEGMENT PARA PUBLIC' CODE'
        ASSUME  CS: CODE3, DS: CODE2
;;;;;;;;;;;;;;;;;;;;;;;;
; 光标定位子程序
; 入口参数：DX=光标位置
;;;;;;;;;;;;;;;;;;;;;;;;
SET_GB  PROC    FAR
        MOV     AH, 2
        MOV     BH, 0
        INT     10H
        RET
SET_GB  ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 在当前光标位置处显示字符
; 入口参数：BH=页号，BL=字符颜色
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
XS _ CHAR PROC      FAR
          MOV       AX，090FH
          MOV       CX，1
          INT       10H
          RET
XS _ CHAR ENDP
CODE3     ENDS
          END       MAIN
```

【例 4.5】求 n 的阶乘

下面的程序是采用递归调用的方法求 n 的阶乘 n! 的程序。阶乘的递归定义为：

$$n! = \begin{cases} 1 & \text{当 } n = 1 \\ n(n-1) & \text{当 } n > 1 \text{ 时} \end{cases}$$

编程指导：

程序设计中编写了一个乘法递归子程序，依次计算 4×3、(4×3) ×2、((4×3) ×2) ×1。为了便于分析程序的流程以及堆栈的变化情况，程序以求 4 的阶乘为例来进行。程序运行时堆栈的变化过程如图 4-1 所示。

程序清单

```
STACK     SEGMENT   PARA STACK′ STACK′
          DB        512 DUP (0)
STACK     ENDS
DATA      SEGMENT   PUBLIC′ DATA′
n         DW        ?                    ；数 n
n _ FACT  DW        ?                    ；数 n 的阶乘 n! 存放单元
DATA      ENDS
CODE      SEGMENT   PARA PUBLIC′ CODE′
          ASSUME    CS：CODE，DS：DATA
START：   MOV       AX，DATA
          MOV       DS，AX
          MOV       n，4                 ；给 n 赋初值
          MOV       n _ FACT，1          ；给 n 的阶乘赋初值
          CMP       n，1
          JE        EXIT                 ；如果 n=1，转结束处理
          CALL      FACT                 ；调用求阶乘子程序
EXIT：    MOV       AH，4CH
          INT       21H                  ；返回 DOS
```

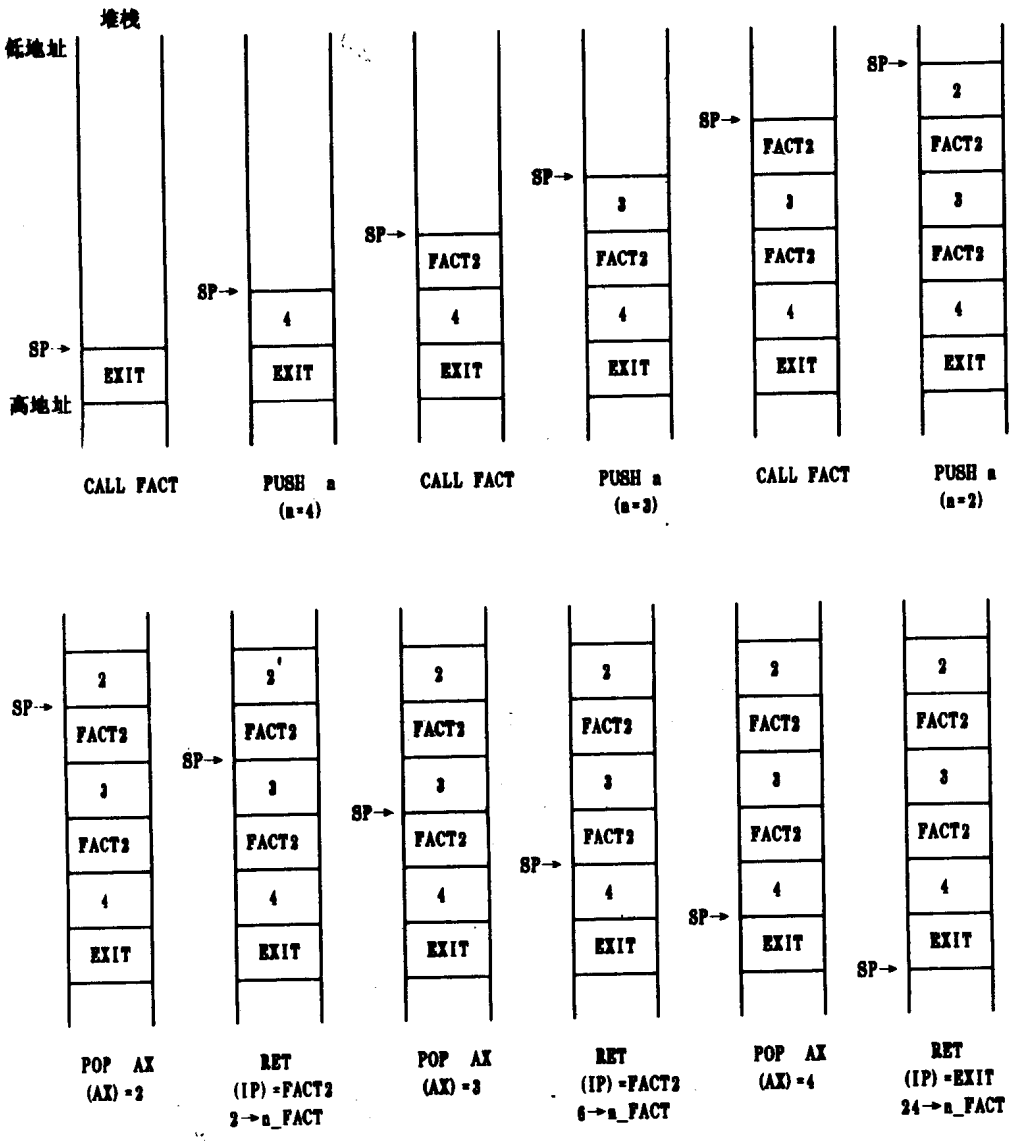


图 4-1　求 n 的阶乘堆栈变化示意

```
;
FACT    PROC    NEAR
FACT1:  PUSH    n
        DEC     n
        CMP     n, 1
        JE      FACT2
        CALL    FACT
FACT2:  POP     AX
        CWD
        MUL     n_FACT          ; 求 m (m-1) 其中：m=2, 3, 4, …, n
        MOV     n_FACT, AX
```

```
        RET
FACT    ENDP
CODE    ENDS
        END     START
```

【例 4.6】显示十六进制数

下面的程序将 DX 中的四位 BCD 码以十六进制形式进行显示。

编程指导：

程序中涉及到几个既有并列关系又有嵌套关系的子程序，特别是子程序之间在进行并列时采用了一种特殊的方式，即子程序 SUB1 本来需要被连续以 CALL 指令调用两次，但程序设计时，采用了一次 CALL 调用和一次顺序地被执行的方式，节省了一条指令。本例的目的就是用以说明这种特殊的子程序并列设计方法。

程序清单

```
STACK   SEGMENT   STACK' STACK'
        DB 256    DUP (0)
STACK   ENDS
CODE    SEGMENT
        ASSUME    CS: CODE
START   PROC      FAR
        PUSH      DS
        XOR       AX, AX
        PUSH      AX
        MOV       DX, 7A4FH
        CALL      SUB0
        RET
START   ENDP
SUB0    LABEL     NEAR
        MOV       AL, DH
        CALL      SUB1          ; 调用 SUB1 子程序
        MOV       AL, DL
SUB1    PROC      NEAR          ; SUB1 被顺序执行一次
        MOV       AH, AL
        PUSH      CX
        MOV       CL, 4
        SHR       AL, CL
        POP       CX
        CALL      SUB2
        MOV       AL, AH
SUB2    PROC      NEAR
        AND       AL, 0FH
```

```
        ADD     AL，90H
        DAA
        ADC     AL，40H
        DAA                          ；一位 BCD 码转换成 ASCII 码
        PUSH    DX
        PUSH    AX
        AND     AL，7FH
        MOV     DL，AL
        MOV     AH，2
        INT     21H                  ；显示 DL 中的 ASCII 码字符
        POP     AX
        POP     DX
        RET
SUB2    ENDP
SUB1    ENDP
CODE    ENDS
        END     START
```

第三节　模块化程序设计方法与实例

一、模块化程序设计概述

在进行大型程序设计时，往往需要将整个问题分解为若干个小问题，如有必要可以将每一个小问题再分解为更小的若干个问题。程序设计时，可以将每个小问题编写成一个单独的程序模块，然后将所有的模块连接起来组合成一个大程序。这种程序设计方法就是模块化程序设计。

模块化程序设计中的各个模块虽然从物理结构上是相互独立的，但是从逻辑结构上具有一定的内在联系。它们是一个统一的整体，共同组合起来完成一个完整的功能。

在实际程序设计中，一个程序可能被划分为几十个模块，这些模块往往又被划分为若干层。处于同一层上的所有模块，都被该层之上的一个模块所管辖，这一管辖模块叫做控制模块。位于整个程序最上一层的控制模块叫做该程序的主控模块。如图 4－2 所示。

二、模块化程序设计的伪操作符

（一）模块的定义

模块化程序设计中的各个模块都有一个名称，都有自己的数据段和代码段，还要有模块的结束标志。它的一般定义格式为：

```
[NAME       模块名]
            ……
            (模块中的程序)
            ……
END         [标号]
```

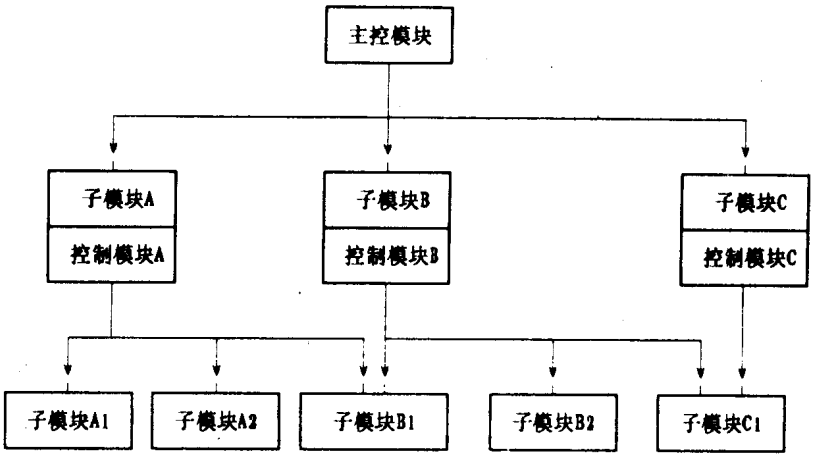


图 4-2 模块化程序的层次结构

其中，[ ] 中的项是可以省略的项。模块名是识别该模块的标志，同时也是本模块与其它模块的区分标志。如果模块名省略，则该模块中 TITLE 语句中的页标题的前六个字符就被当作该模块的名称。如果模块中没有 TITLE 语句，则该模块的磁盘文件名就被认为是该模块的名称。

END 语句是该模块的结束标志，它被用于告诉汇编程序 (MASM) 该模块的所有程序到此为止。END 语句中的标号是可选项，该标号是整个程序开始执行的第一条指令的标号。由于一个程序只可能有一个起点，所以只有整个程序的主控制模块的结尾才能使用该选择项。

(二) 公共符号的定义

一个模块中某些符号常量、变量和标号 (包括过程名)，往往需要被其它模块所引用，这样的符号常量、变量和标号被称为公共符号。对于公共符号，需要进行公共符号的定义，否则汇编程序只允许这些符号在本模块中使用，如果其它模块引用这些符号则被认为是错误的。公共符号的定义格式是：

PUBLIC ＜符号表＞

其中，＜符号表＞中的各个符号之间用逗号分隔开。

注意，如果符号表中包含有转移标号和过程名，则这些转移标号和过程名一般都属于 FAR 类型。

(三) 外部符号引用定义

当前模块中使用的某些符号 (包括符号常量、变量、转移标号和过程名)，引用的是别的模块中的符号，对于这样的符号必须使用 EXTRN 伪指令进行说明，否则汇编程序认为这些符号没有被定义。外部符号引用的定义格式是：

EXTRN ＜符号：类型＞ [，…]

其中，“类型”可以是 BYTE、WORD、DWORD、NEAR、FAR 和 ABS。ABS 表示该符号是符号常量而不是变量或标号。

注意，被引用的外部符号的类型，必须与这些外部符号在其它模块中定义时的类型保持一致。

三、模块化程序设计实例

【例 4.7】小型文件加/解密系统

下面的程序可以对任何一个源程序文件或数据文件进行加/解密，达到保护源程序或数据文件不被非法阅读和篡改的目的。

编程指导：

程序设计中采用了模块化程序设计方法，其模块划分结构如图 4－3 所示。

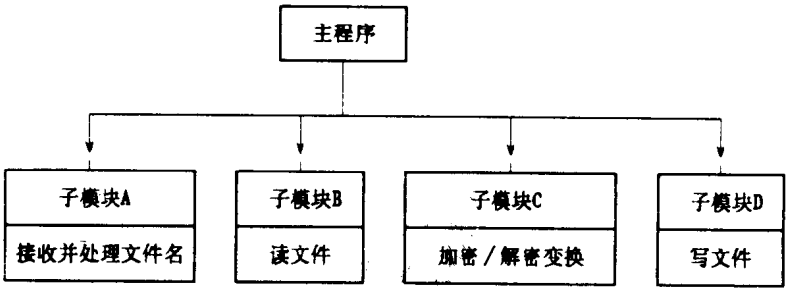


图 4－3　小型加/解密系统模块结构

该程序设计的思路是：

(1) 程序一开始运行便显示提示信息，调用“子模块 A”接收用户输入的读盘文件名；

(2) 调用“子模块 B”将对应的磁盘文件读入内存缓冲区；

(3) 调用“子模块 C”对内存缓冲区中的文件内容进行加密或解密变换；

(4) 显示提示信息，调用“子模块 A”接收用户输入的写盘文件名；

(5) 调用“子模块 D”将加密或解密的结果写入对应的磁盘文件中。

由于加密模块即“子模块 C”编写时采用了可逆运算，所以加密模块，实际上又是密模块。对于一个源程序文件或数据文件进行奇数次的处理，所得的结果是密文文件，进行偶数次的处理，所得结果是明文文件。

程序清单

(1) 主程序

由于主程序中使用了一些外部模块名，如 JJM、SRWJM、DWJ、XWJ 等，所以在主程序的开头用 EXTRN 伪操作对这些外部符号进行了定义。

在主程序中的一些变量和标号，如 WJM、HCQ、ZJS 等要被其它外部模块所引用，所以在这些变量和标号的定义之后，用 PUBLIC 伪操作对这些公共符号进行了定义。

```
        NAME    MK0. ASM
        EXTRN   JJM:FAR,SRWJM:FAR,DWJ:FAR,XWJ:FAR      ;外部符号引用说明
STACK   SEGMENT PARA STACK' STACK'
        DB 128  DUP (0)
STACK   ENDS
DATA    SEGMENT PARA PUBLIC' DATA'
WJM     DB 16,?, 16 DUP (0)
HCQ     DB 10000 DUP (0)
MSG1    DB 0DH, 0AH,' 请输入读盘文件名: $'
MSG2    DB 0DH, 0AH,' 请输入写盘文件名: $'
ZJS     DW ?
        PUBLIC WJM, HCQ, ZJS                            ; 公共符号说明
```

```
DATA    ENDS
CODE    SEGMENT PARA PUBLIC' CODE'
        ASSUME CS: CODE, DS: DATA
START   PROC    FAR
        PUSH    DS
        MOV     AX, 0
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
; ————主控模块————
        MOV     DX, OFFSET MSG1
        CALL    SRWJM                   ; 接收键入的读盘文件名
        CALL    DWJ                     ; 读文件
        MOV     BX, OFFSET HCQ
        MOV     CX, ZJS
        CALL    JJM                     ; 加/解密变换
        MOV     DX, OFFSET MSG2
        CALL    SRWJM                   ; 接收输入的写盘文件名
        CALL    XWJ                     ; 写文件
        RET
START   ENDP
CODE    ENDS
        END     START
```

(2) 加/解密模块

本模块要被主程序所调用，所以必须将 JJM 过程名用 PUBLIC 进行定义。

```
        NAME    MK1. ASM
        PUBLIC  JJM
CODE    SEGMENT PUBLIC' CODE'
        ASSUME  CS: CODE
; 入口参数: BX=缓冲区首地址
;          CX=字节数
JJM     PROC    FAR
        PUSH    CX
        PUSH    BX
JJM1:   PUSH    CX
        MOV     AL, [BX]
        MOV     CL, 4
        ROL     AL, CL                  ; 对 AL 中的内容进行左循环 4 次
```

```
        MOV     [BX], AL
        INC     BX
        POP     CX
        LOOP    JJM1
        POP     BX
        POP     CX
        RET
JJM     ENDP
CODE    ENDS
        END
```

(3) 接收文件名、读盘和写盘模块

为了简明起见，我们将接收文件名模块、读盘模块和写盘模块合并在一个代码段中，组成一个模块。在一模块中的程序要引用主模块中的 WJM、HCQ、ZJS 等符号，我们要用 EXTRN 对这些被引用的符号进行定义。而本模块中的符号 SRWJM、DWJ、XWJ 等又要被主模块所引用，所以在此模块的开头又要将它们用 PUBLIC 进行定义。

```
        NAME    MK2. ASM
        EXTRN   WJM: BYTE, HCQ: BYTE, ZJS: WORD
        PUBLIC  SRWJM, DWJ, XWJ
CODE    SEGMENT PUBLIC' CODE'
        ASSUME  CS: CODE
; 入口参数：DX=提示信息首地址
SRWJM   PROC    FAR
SR1:    PUSH    DX
        MOV     AH, 9
        INT     21H                     ; 显示提示信息
        MOV     DX, OFFSET WJM+2
        MOV     AH, 0AH
        INT     21H                     ; 接收输入的文件名
        POP     DX
        CMP     WJM+1, 0                ; 判是否直接回车
        JZ      SR1                     ; 是直接回车，转 SR1 重新接收
        MOV     BX, OFFSET WJM+2
        ADD     BL, WJM+1
        MOV     BYTE PTR [BX], 0        ; 文件名的尾部清 0
        RET
SRWJM   ENDP
; 读文件模块
; 入口参数：无
```

```
DWJ     PROC    FAR
        MOV     AX，3D00H
        MOV     DX，OFFSET WJM+2
        INT     21H                    ；打开文件
        MOV     BX，AX
        MOV     CX，10000
        MOV     DX，OFFSET HCQ
        MOV     AH，3FH
        INT     21H                    ；读文件
        MOV     ZJS，AX
        MOV     AH，3EH
        INT     21H                    ；关闭文件
        RET
DWJ     ENDP
；写文件模块
；入口参数：无
XWJ     PROC    FAR
        MOV     AH，3CH
        MOV     CX，0
        MOV     DX，OFFSET WJM
        INT     21H                    ；建立文件
        MOV     BX，AX
        MOV     CX，ZJS
        MOV     DX，OFFSET HCQ
        MOV     AH，40H
        INT     21H                    ；写文件
        MOV     AH，3EH
        INT     21H                    ；关闭文件
        RET
XWJ     ENDP
CODE    ENDS
        END
```

说明：上述程序的装配方法是：

(1) 汇编：

A>MASM MK0；<CR>

A>MASM MK1；<CR>

A>MASM MK2；<CR>

(2) 连接：

A>LINK MK0+MK1+MK2；<CR>

(3) 运行：

A>MK0<CR>

第四节　参数传递的方法

一、参数传递的方法

在程序设计中，经常要遇到传递参数的问题。参数传递不仅进行在主程序与子程序、子程序与子程序之间，而且还进行在系统与主程序之间。如果需要传递参数，就必须事先确定好参数传递的方法。传递参数的方法主要有以下几种：

1. 用寄存器传递参数

这种方法是利用 CPU 的寄存器传递参数，是一种常用的方法。特点是：速度快，使用方便。但是，寄存器的数目是有限的，当同时传递的参数很多时，不宜采用。所以，一般适用于传递的参数较少的情况。

2. 用存储器传递参数

这种方法是在数据段或代码段中开辟一块存储器单元，主程序事先将欲处理的内容送入这些单元中，并设置一个或几个指针指向它，然后调用子程序。而子程序通过指针对这些数据进行处理，处理的结果可能放入另外一块内存单元中，或某个寄存器中。

在需要传递的参数比较多时，采用这种方法。

3. 用堆栈传递参数

堆栈是在内存中开辟的一块具有特殊存取数据规则的存储区域，在某些程序设计中采用堆栈来传递参数往往会达到预期的效果。在使用堆栈传递参数时，用 PUSH 指令存放参数，为了读出堆栈中的参数，一般不用 POP 指令，而是根据 SP 的变化规律，推算出所放参数在堆栈中的地址，然后用 MOV 指令对参数进行读取。

这种方法在使用时一定要小心，否则会造成堆栈的混乱，导致死机。

4. 用 DOS 的 PSP 段前缀传递参数

利用 DOS 的 PSP 段前缀可以由系统向运行程序中传递参数，许多系统程序或应用程序就是通过这种方法提供运行程序所需要的参数。

如果在系统提示符状态下，键入一个运行的文件名后，DOS 确定最低可用的空闲内存地址作为该运行程序的起始地址，这个区域被称为程序段。程序段的前 256 个字节是由 EXEC 系统为调用该运行程序而建立的控制区，叫做程序段前缀控制块，简称为 PSP。如果键入运行文件名的同时，相应地输入一些参数，则前两个参数分别被格式化在程序段前缀的 5CH 和 6CH 处，在 81H 处包含了所有键入的未格式化的字符，而在 80H 字节中记录了这些字符的个数。应用程序不仅可以使用格式化区域中的参数，还可以使用未格式化区域中的参数。

二、用寄存器/存储器传递参数编程实例

【例 4.8】汉字显示字模到打印字模的转换

在 CCDOS 操作系统下，汉字的 16×16 打印点阵字模是由 16×16 显示点阵字模经横纵转换而来。设从内存单元 DISP _ DOTS 开始，存放有“大”字的 16×16 显示点阵字模（共 32 个字节），试编程将其转换成 16×16 打印点阵字模，转换的结果放在从内存单元 PRINT _ DOTS

开始的 32 个字节单元中。

编程指导：

在 CCDOS 中，汉字的 16×16 显示点阵字模是按横向的顺序排列而成的，例如，“大”字的 16×16 显示点阵字模及其点阵图如图 4－4 和表 4－1 所示。

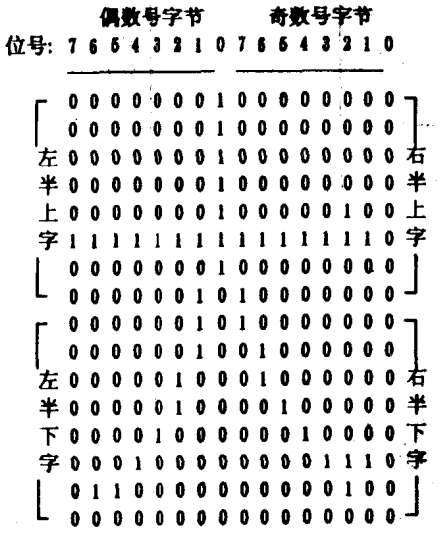


图 4－4 “大”字的点阵图

表 4－1 “大”字的点阵字模

字节编号（十进制）	字节值（十六进制）	字节值（十六进制）	字节编号（十进制）
00	01H	00H	01
02	01H	00H	03
04	01H	00H	05
06	01H	00H	07
08	01H	04H	09
10	FFH	FEH	11
12	01H	00H	13
14	02H	80H	15
16	02H	80H	17
18	02H	40H	19
20	04H	40H	21
22	04H	20H	23
24	08H	10H	25
26	10H	0EH	27
28	60H	04H	29
30	00H	00H	31

为了说明起见，把汉字 16×16 显示点阵字模划分为左半上字，左半下字，右半上字及右半下字等四个部分。其中，左半上字包括第 00，02，04，06，08，10，12，14 等 8 个字节；左半下字包括第 16，18，20，22，24，26，28，30 等 8 个字节；右半上字包括第 01，03，05，07，09，11，13，15 等 8 个字节；右半下字包括第 17，19，21，23，25，27，29，31 等 8 个字节，如图 4－4 所示。

要把汉字的 16×16 显示点阵字模转换成打印字模，即要按照如下方式把原 16×16 显示点阵字模进行重新组合：

将显示点阵字模的左半上字的第 0，2，…，14 等 8 个字节的所有第 7 位组合成一个纵向字节；将左半下字的 16，18，…，30 等 8 个字节的所有第 7 位组合成另一纵向字节。这样，得到打印点阵字模的一列 16 位数据。接着，再把左半上字、左半下字的所有第 6 位，第 5 位，第 4 位，……，第 0 位分别组合成打印点阵纵向的第 2 列，第 3 列，……，第 8 列数据。同理，对右半上字、右半下字的 16 个字节作同样的变换，得到打印点阵的第 9 列，第 10 列，第 11 列，……，第 16 列数据即可。如图 4－5 所示。

为了完成以上转换工作，程序设计采用三重循环形式。最内层循环（XCHG _ DOT 程

序）执行一次，完成把四分之一字（左半上字或左半下字或右半上字或右半下字）的 8 个字节的同一位组合成一字节打印字模；外层循环执行一次，完成把原 16×16 显示点阵的纵向 16 个字节的同一位组合成一列纵向打印字模（两个字节）；最外层循环执行一次完成 8 列纵向打印字模的生成。这样，最外层循环执行两次，即生成所有 16 列打印字模。

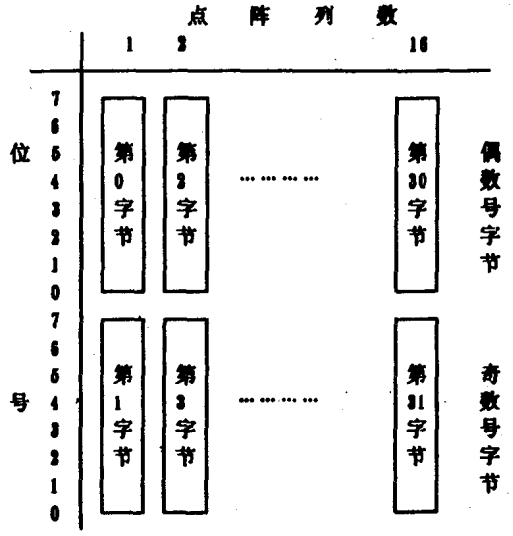


图 4-5 汉字打印字模排列结构

程序清单

```
STACK    SEGMENT   PARA STACK' STACK'
         DB        128 DUP (0)
STACK    ENDS
DATA     SEGMENT
DISP_DOTS          DB 01H, 00H, 01H, 00H, 01H, 00H, 01H, 00H
                   DB 01H, 04H, 0FFH, 0FEH, 01H, 00H, 02H, 80H
                   DB 02H, 80H, 02H, 40H, 04H, 40H, 04H, 20H
                   DB 08H, 10H, 10H, 0EH, 60H, 04H, 00H, 00H
PRINT_DOTS         DB 32 DUP (0)
DATA     ENDS
CODE     SEGMENT   PUBLIC' CODE'
         ASSUME    CS: CODE, DS: DATA, ES: DATA
START              PROC    FAR
                   PUSH    DS
                   XOR     AX, AX
                   PUSH    AX
                   MOV     AX, DATA
                   MOV     DS, AX
                   MOV     ES, AX
                   LEA     SI, DISP_DOTS
                   LEA     DI, PRINT_DOTS
                   PUSH    DI
                   MOV     CX, 2          ; 处理 2 列字节（偶数列、奇数列）
LOOP1:             PUSH    SI
```

```
                    PUSH    CX
                    MOV     CX, 8           ; 每列处理 8 位
LOOP2:              PUSH    SI
                    CALL    XCHG_DOT        ; 处理上半字的同一位
                    CALL    XCHG_DOT        ; 处理下半字的同一位
                    POP     SI
                    LOOP    LOOP2           ; 左或右半字 8 位未处理完，转
                    POP     CX
                    POP     SI
                    INC     SI
                    LOOP    LOOP1           ; 2 个 8 列打印字模未生成，转
                    POP     DI
                    RET
START               ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 子程序 XCHG_DOT
;        入口参数：SI=  要处理的 8 字节字模首地址
;                  CL=  要处理的位号
;        出口参数：SI=  要处理的 8 字节字模末地址+2
;                  DI=  下一字节打印字模存放单元
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
XCHG_DOT            PROC    NEAR
                    XOR     AH, AH
                    MOV     DH, 8           ; 处理 8 个字节的同一位
RETRY:              MOV     AL, [SI]
                    RCR     AL, CL          ; 当前字模字节的指定位移进 CF
                    RCL     AH, 1           ; CF 移进 AH 的最低位
                    ADD     SI, 2           ; 地址+2，以便取得纵向下一字节
                    DEC     DH
                    JNZ     RETRY
                    MOV     [DI], AH        ; 存储组合成的一字节打印字模
                    INC     DI
                    RET
XCHG_DOT            ENDP
CODE        ENDS
            END     START
```

说明：在上例中，主程序与子程序 XCHG_DOT 之间传递参数的方法既用到存储器，又用到寄存器。主程序把欲转换的点阵字模的首地址放于 SI，把移位次数送 CL，然后调用子程序。

而子程序执行时，从SI所指的单元中取数据进行处理，完成转换任务后，把结果放于DI所指的存储器单元中。

三、用堆栈传递参数编程实例

【例4.9】求矩阵的乘法

设有矩阵$A_{m\times k}$及矩阵$B_{k\times n}$，其矩阵中的元素按行的顺序分别放在两块连续的内存缓冲区中。试编程求出其乘积，求得的乘积矩阵$C_{m\times n}$的m×n个元素也按行的顺序存放于一个连续的内存单元中。

编程指导：

因为$C_{m\times n}$中的每一元素$C_{ij}=A_{i1}\times B_{1j}+A_{i2}\times B_{2j}+\cdots+A_{ik}\times B_{kj}$（其中：i=1，2，…，m；j=1，2，…，n）。所以，程序的核心是编写一个求$A_{iz}\times B_{zj}$（其中：z=1，2，…，k）乘法的子程序；然后，把求得的k个中间结果累加即得到C_{ij}。由于，计算一个C_{ij}需要做k次类似的乘法，所以本子程序要编写成一个循环子程序，循环的次数为k。这样，才能求出矩阵$C_{m\times n}$的一个元素。矩阵$C_{m\times n}$一行中有n个元素，要求出这n个元素，需要把以上的程序整体作为一个循环体，每次修改地址指针，循环n次。这样，才能求得矩阵$C_{m\times n}$的n个元素。矩阵$C_{m\times n}$共有m行，所以又需要把上述两重循环作为一个循环体再循环m次，这样才能求得矩阵$C_{m\times n}$的全部m×n个元素。

下面的程序用矩阵$A_{m\times k}$的行数m作为外循环的计数，用矩阵$B_{k\times n}$的列数n作为次外循环的计数。同时，这两个计数值本身又分别作为当前所求的矩阵$C_{m\times n}$中的元素C_{ij}（i=1，2，…，m；j=1，2，…，n）实际的行数及列数。传递参数的方法是：先把m及n分别送CX，然后压栈保护，再由MOV指令从堆栈中分别传送至AX及DX寄存器。然后，程序根据寄存器AX及DX中的值，求出当前所求元素的实际行数及列数，最终求出该行该列的元素值。在下面的程序设计中，由于控制外层、次外层循环的方法都是采用倒计数法，所

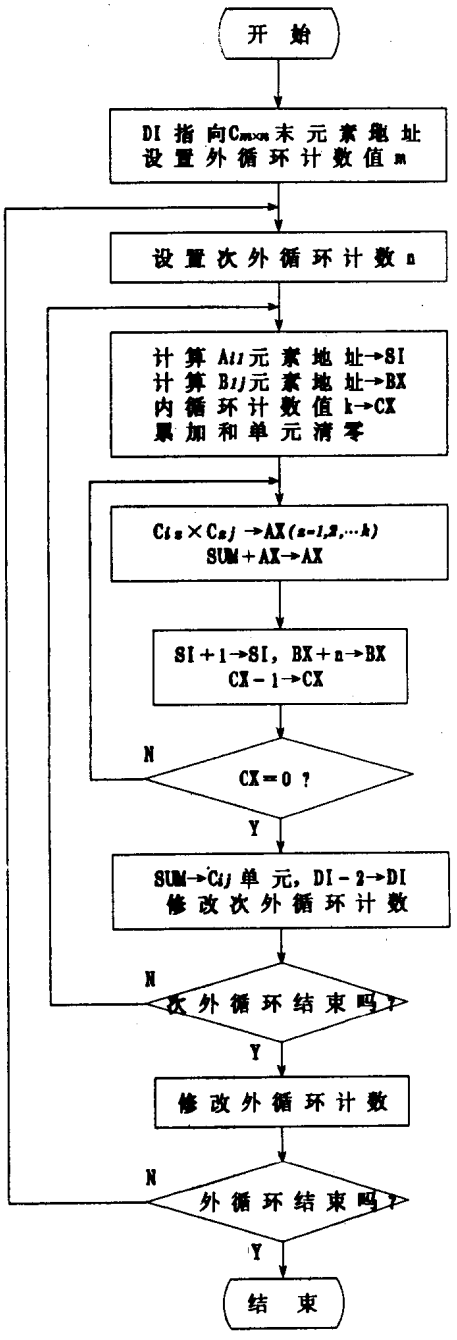


图4-6 求矩阵乘法的流程图

以求出矩阵 $C_{m\times n}$ 中的元素顺序依次是：$C_{m,n}$，$C_{m,n-1}$，…，$C_{m,1}$；$C_{m-1,n}$，$C_{m-1,n-1}$，…，$C_{m-1,1}$；$C_{1,n}$，$C_{1,n-1}$，…，$C_{1,1}$。图 4-6 是该算法的流程图。

程序清单

```
STACK   SEGMENT   PARA STACK' STACK'
        DB 128    DUP (0)
STACK   ENDS
DATA    SEGMENT
m               EQU 3              ; 矩阵 Amxk 的行数
n               EQU 3              ; 矩阵 Bkxn 的列数
k               EQU 4              ; 矩阵 Amxk 的列数或矩阵 Bkxn 的行数
Amxk            DB 1, 2, 3, 1
                DB 2, 4, 2, 0
                DB 1, 0, 3, 2
Bkxn            DB 0, 1, 2
                DB 1, 3, 1
                DB 1, 2, 1
                DB 0, 1, 1
SUM             DW 0               ; Cij 中间结果单元，其中 i=1, 2, …, m; j=1,
                                   ; 2, …, n ; Cij=Ai1×B1j+Ai2×B2j+…+Aik×Bkj
Cmxn            DW m*n DUP (0)     ; 矩阵 Cmxn 的 m×n 个元素存放单元
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,ES:DATA
START   PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     DI, OFFSET Cmxn
        ADD     DI, 2*(m*n)-2      ; DI=矩阵 Cmxn 最后一个元素存放地址
        MOV     CX, m              ; 矩阵 Amxk 的行数 m 作为外循环计数
RETRY1: PUSH    CX
        MOV     CX, n              ; 矩阵 Bkxn 的列数作为次外循环计数
RETRY2: PUSH    CX
        MOV     BP, SP
        MOV     AX, [BP+2]         ; 当前所求的元素的行号→AX
```

```
                MOV     DX，[BP]            ；当前所求的元素的列号→DX
                DEC     AX
                MOV     CL，k
                MUL     CL
                MOV     SI，OFFSET Amxk
                ADD     SI，AX              ；计算出 Cil 的地址→SI
                DEC     DX
                MOV     BX，OFFSET Bkxn
                ADD     BX，DX              ；计算出 Clj 的地址→BX
                MOV     CX，k               ；内循环计数→CX
                MOV     SUM，0              ；Cij 中间结果累加单元
RETRY3：        MOV     AL，[SI]            ；Aiz→AL
                MOV     AH，[BX]            ；Bzj→AH
                MUL     AH                  ；Aiz×Bzj （z=1，2，…，k）
                ADD     SUM，AX             ；累加中间结果
                INC     SI                  ；Am×k 中下一被乘数地址→SI
                ADD     BX，n               ；Bk×n 中下一乘数地址→BX
                LOOP    RETRY3              ；循环求出 Cij
                MOV     AX，SUM
                MOV     [DI]，AX            ；累加和送 Cij 对应的单元
                DEC     DI
                DEC     DI                  ；DI 指向同一行中的下一元素地址
                POP     CX
                LOOP    RETRY2              ；矩阵 Cm×n 的一行元素未求完，转
                POP     CX
                LOOP    RETRY1              ；矩阵 Cm×n 的 m 个行未求完，转
                RET
START           ENDP
CODE            ENDS
                END     START
```

四、用 DOS 的 PSP 传递参数编程实例

【例 4.10】设置屏幕显示方式

编写一个程序，利用 DOS 的非格式化参数区传递参数：0～7，从而把屏幕设置成与输入的参数对应的显示方式。对非法的参数应显示出错信息，然后返回 DOS。

编程指导：

程序运行时，首先从 DOS 的非格式化参数区中取得未格式化的参数，过滤掉前到空格和后继空格，然后进行参数的合法性检查，在确认参数在 0～7 之间时，调用 BIOS 中断功能设置屏幕显示方式，否则显示出错信息，然后返回 DOS。

程序清单

```
STACK     SEGMENT PARA STACK' STACK'
          DB 128 DUP (0)
STACK     ENDS
DATA      SEGMENT
PARM                DB ?
ERROR_MSG           DB 13, 10,' 输入的参数非法! $'
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE
START           PROC      FAR
                PUSH      DS
                XOR       AX, AX
                PUSH      AX
                MOV       AX, DATA
                ASSUME    ES: DATA
                MOV       ES, AX
                MOV       SI, 80H           ; 源字符串的首地址 (PSP+80H) 送 SI
                MOV       DI, OFFSET PARM   ; 屏幕方式字单元地址送 DI
                CLD
                LODSB
                CMP       AL, 0             ; 参数个数是否为 0
                JE        EXIT              ; 参数个数为 0, 转出口
NEXT1:          LODSB
                CMP       AL, 0DH
                JE        EXIT              ; 是直接回车, 转出口
                CMP       AL, 20H
                JE        NEXT1             ; 过滤掉空格, 转 NEXT1 取下一字符
                CMP       AL, 09H
                JE        NEXT1             ; 有制表符, 过滤掉
                STOSB                       ; 存储一字节参数
NEXT2:          LODSB
                CMP       AL, 0DH
                JE        NEXT3             ; 参数的下一字符为回车, 转 NEXT3
                CMP       AL, 20H
                JE        NEXT2             ; 过滤掉参数后的空格符, 转 NEXT2
                CMP       AL, 09H
                JE        NEXT2             ; 过滤掉参数后的制表符, 转 NEXT2
DISP_ERROR:     ASSUME    DS: DATA          ; 有多于一个的非空参数, 显示出错信息
```

```
                PUSH    ES
                POP     DS
                MOV     AH，9
                MOV     DX，OFFSET ERROR _MSG
                INT     21H
                RET                             ；返回 DOS
NEXT3：         CMP     ES：PARM，' 0'          ；判是否为合法的参数：0～7
                JB      DISP _ERROR             ；是非法参数，转
                CMP     ES：PARM，' 7'
                JA      DISP _ERROR             ；是非法参数，转
                SUB     ES：PARM，30H           ；方式字由 ASCII 码转二进制
                MOV     AH，0
                MOV     AL，ES：PARM            ；合法的方式字送 AL
                INT     10H                     ；设置显示方式
EXIT：          RET                             ；返回 DOS
START           ENDP
CODE    ENDS
        END START
```

【例 4.11】变换首簇号加密文件

编写一个程序，通过变换软盘根目录中文件首簇号内容的方法，达到保护文件内容不能被阅读的目的。

编程指导：

在文件目录登记项的 32 个字节中，第 26、27 字节存放的是文件的首簇号，首簇号反映了文件内容在磁盘上存放的起始位置。DOS 在进行磁盘文件的访问时，首先从文件目录表中查得文件的首簇号，然后以首簇号为起点，在文件分配表中查找文件的簇号链。如果我们改变文件的首簇号，则 DOS 在查找文件时，必然查找到一个错误的簇号链。虽然此时 DOS 仍然可以查到一个簇号链，但该簇号链已非原文件的簇号链，从而不会读取到原文件的真正内容。修改文件的首簇号有很多方法，但是为了使修改以后的首簇号便于恢复，我们采用的方法是：将首簇号的第 26、27 字节内容交换。这种修改首簇号的方法，其加密过程与解密过程是可逆的，即采用同样一种方法修改首簇号，如果修改奇数次则为加密，修改偶数次则为解密。

为了使编写的程序一次能对一批文件进行首簇号加密变换，我们利用 DOS 的格式化参数区进行参数的传递，并允许命令行中被加密的文件名中可以带有文件名通配符“?”和“*”。

程序运行后，首先读取软盘根目录区中的 112 个目录登记项的内容，将它们分别进行通配符覆盖处理后，与规定的文件名进行比较，如果两者相同，则加密该文件的首簇号，否则不进行加密。

程序清单

```
STACK   SEGMENT PARA STACK' STACK'
        DB 128 DUP (0)
```

```
STACK   ENDS
DATA    SEGMENT
HCQ     DB 32 * 112 DUP (0)
WJM     DB 11 DUP (0)
DATA    ENDS
CODE    SEGMENT PUBLIC' CODE'
        ASSUME CS: CODE, DS: DATA, ES: DATA
START   PROC   FAR
        MOV    AX, DATA
        MOV    DS, AX
        MOV    BX, OFFSET HCQ
        MOV    AL, 0
        MOV    DX, 5
        MOV    CX, 7
        INT    25H              ; 读 A 盘根目录的 7 个扇区
        MOV    CX, 112          ; 处理 112 个目录项
        MOV    SI, OFFSET HCQ   ; SI 指向第一个文件目录登记项首地址
        MOV    DI, 005DH        ; DI 指向 PSP 中文件控制块第一字符地址
        CLD
RETRY1: PUSH   CX
        PUSH   SI
        PUSH   DI
        MOV    CX, 11
        MOV    BX, OFFSET WJM
REMOVE: MOV    AL, [SI]
        MOV    [BX], AL         ; 把当前文件名传送到文件名缓冲区
        INC    SI
        INC    BX
        LOOP   REMOVE
        MOV    CX, 11
        MOV    BX, OFFSET WJM
RETRY2: MOV    AL, ES: [DI]
        CMP    AL,' ?'
        JNE    NEXT1
        MOV    [BX], AL         ; 进行通配符覆盖处理
NEXT1:  INC    DI
        INC    BX
        LOOP   RETRY2
        MOV    SI, OFFSET WJM
```

```
        MOV     DI, 005DH       ; DI 指向格式化参数区
        MOV     CX, 11
        REPE    CMPSB           ; 比较当前文件是否为被加密的文件
        POP     DI
        POP     SI
        JNZ     NEXT2           ; 不是，转 NEXT2
        MOV     AX, [SI+26]     ; 是，加密当前文件的首簇号
        XCHG    AL, AH
        MOV     [SI+26], AX
NEXT2:  ADD     SI, 32          ; SI 指向下一文件的目录项
        POP     CX
        LOOP    RETRY1
        MOV     BX, OFFSET HCQ
        MOV     AL, 0
        MOV     DX, 5
        MOV     CX, 7
        INT     26H             ; 把修改后的磁盘文件目录写盘
        MOV     AH, 4CH
        INT     21H             ; 返回 DOS
START   ENDP
CODE    ENDS
        END     START
```

说明：该程序经汇编、连接后可以用以下方式进行使用（设该程序名为 XCLUS）：

A＞XCLUS 被加密的文件名＜CR＞

其中文件名可以带有通配符“*”和“?”。该程序的加密方法是可逆的，也就是说如果要进行解密，则再运行一次该程序即可。

第五章　数据处理编程实例

本章主要讨论汇编语言设计中的数据处理问题。其内容包括十进制算术运算、代码转换、查表与排序等。这些数据处理问题是实际工作中经常遇到的问题，它们中的每一类中又包含了许多具体的小类型。例如，代码转换包括 BCD 码转二进制数、二进制数转 BCD 码，ASCII 码转 BCD 码，BCD 码转 ASCII 码，还有 ASCII 码转二进制数、二进制数转 ASCII 码等。为了使初学者掌握各种数据处理问题的编程方法，本章在介绍每一类数据处理问题时，都给出了具体的程序实例。

第一节　十进制算术运算编程实例

CPU 不仅提供了进行二进制算术运算的指令，还提供了进行十进制算术运算的指令。这里所说的十进制数就是所谓的 BCD 码，分为未组合的（非压缩的）十进制数和组合的（压缩的）十进制数。未组合的 BCD 码即用一字节中低 4 位二进制数表示一位十进制数的编码，字节的高 4 位为零。组合的 BCD 码即用一字节表示两位十进制数的编码，其中高 4 位和低 4 位分别表示一位十进制数。

进行十进制算术运算时，CPU 要求先用二进制算术运算指令对十进数进行相应的运算，然后再用十进制调整指令将运算的结果调整为正确的十进制数。

进行十进制数调整运算的指令有：

(1) 未组合的 BCD 码加法调整指令 AAA

AAA 指令紧跟在二进制加法指令之后，对在 AL 中的由两个未组合的十进制数相加的结果进行调整，产生一个未组合的十进制和在 AX 中。

(2) 组合的 BCD 码加法调整指令 DAA

DAA 紧跟在加法指令之后，对 AL 中由两个组合的十进制数相加的结果进行调整，得到正确的组合的十进制结果在 AL 中。

(3) 未组合的 BCD 码减法调整指令 AAS

AAS 指令紧跟在二进制减法指令之后，对在 AL 中的由两个未组合的十进制数相减的结果进行调整，产生一个未组合的十进制差在 AL 中。

(4) 组合的 BCD 码减法调整指令 DAS

DAS 紧跟在减法指令之后，对 AL 中由两个组合的十进制数相减的结果进行调整，得到正确的组合的十进制结果在 AL 中。

(5) 未组合的 BCD 码乘法调整指令 AAM

AAM 指令紧跟在二进制乘法指令之后，对在 AX 中的由两个未组合的十进制数相乘的结果进行调整，产生一个未组合的十进制数的乘积在 AX 中（高位在 AH 中，低位在 AL 中）。

(6) 组合的 BCD 码除法调整指令 AAD

AAD 指令能把在 AX 中的两个未组合的十进制数在两个数相除以前进行调整，使得相除

以后可以得到正确的未组合的十进制数，商在 AL 中，余数在 AH 中。

一、未组合的十进制数算术运算

【例 5.1】 多位未组合的十进制数加法

内存中从 FIRST 和 SECOND 开始的单元中分别存放着两个 4 位未组合的十进制数，数据存放的规则是，低位在低地址，高位在高地址。试编程求这两个数的和，并存放到从 THIRD 开始的内存单元中。

编程指导：

两个 4 位的十进制数进行加法运算，最高位可能产生进位，为了避免丢失最高位的进位，我们给被加数、加数以及结果存放单元都开辟了 5 个字节单元。在进行加法运算时，每进行一位加法运算，都将产生的进位加到加数的下一位上，以保证下一位加法操作的正确性。

程序清单

```
STACK     SEGMENT   STACK 'STACK'
          DB        128 DUP (0)
STACK     ENDS
DATA      SEGMENT
FIRST     DB        4, 7, 9, 2, 0                ; 十进制数 02974
SECOND    DB        6, 2, 3, 7, 0                ; 十进制数 07326
THIRD     DB        5 DUP (0)
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS:CODE,DS:DATA,ES:DATA
UNPACK    PROC      FAR
          MOV       AX, DATA
          MOV       DS, AX
          MOV       ES, AX
          LEA       SI, FIRST
          LEA       BX, SECOND
          LEA       DI, THIRD
          MOV       CX, 4+1                      ; 进行 5 次加法
          CLD
RETRY:    LODSB
          ADD       AL, [BX]                     ; 对应的一位进行加法运算
          AAA
          STOSB                                  ; 保存当前位和
          INC       BX
          ADC       BYTE PTR [BX], 0             ; 进位值加到加数的下一位
          LOOP      RETRY
          MOV       AH, 4CH
```

```
        INT     21H
UNPACK  ENDP
CODE    ENDS
        END     UNPACK
```

【例 5.2】 多位未组合的十进制数减法

内存中从 FIRST 和 SECOND 开始的单元中分别存放着两个 4 位未组合的十进制数，数据存放的规则是，低位在低地址，高位在高地址。试编程求这两个数的差，并存放到从 THIRD 开始的内存单元中。要求：如果被减数大于减数，则将“+”号存放在结果的最后，否则，用减数减去被减数，并将“-”存放在结果的最后。

编程指导：

两个十进制数进行减法运算，被减数可能大于减数，也可能小于减数。如果被减数大于减数，则计算结果得到正确的差值；而如果被减数小于减数，为了得到正确的负数值，就必须再用减数减去被减数，并在结果的后面标上负号“-”。在进行每一位的减法运算时，为了不丢掉前一次的借位值，使用了带借位的减法操作。

程序清单

```
STACK   SEGMENT STACK 'STACK'
        DB      128 DUP (0)
STACK   ENDS
DATA    SEGMENT
FIRST   DB      4, 0, 0, 0, 0           ; 十进制数 0004
SECOND  DB      1, 4, 5, 8, 0           ; 十进制数 8541
THIRD   DB      0, 0, 0, 0, '+'
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA, ES: DATA
UNPACK  PROC    FAR
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        LEA     SI, FIRST
        LEA     BX, SECOND
        LEA     DI, THIRD
GOON:   MOV     CX, 4                   ; 进行 4 次减法
        CLC
        CLD
RETRY:  LODSB
        SBB     AL, [BX]                ; 对应的一位进行减法运算
```

```
            AAS
            STOSB                       ; 保存当前位的差
            INC     BX
            LOOP    RETRY
            JC      MANUS               ; 被减数小于减数，转 MANUS
            JMP     PROGEND             ; 和为正数，转结束
MANUS:      MOV     AL, '-'             ; 传送符号标志到结果的最后一位
            STOSB
            LEA     SI, SECOND          ; 减数当作被减数
            LEA     BX, FIRST           ; 被减数当作减数
            LEA     DI, THIRD
            MOV     CX, 4
            CLC
RETRY1:     LODSB
            SBB     AL, [BX]
            AAS
            STOSB
            INC     BX
            LOOP    RETRY1
PROGEND:    MOV     AH, 4CH
            INT     21H
UNPACK      ENDP
CODE        ENDS
            END     UNPACK
```

【例 5.3】 未组合的十进制数乘法

下面的程序能够完成一个多位的未组合的十进制数与一个一位的未组合的十进制数的乘法。

编程指导：

一个多位的未组合的十进制数与一个一位的未组合的十进制的乘法应该按照如下的方法进行乘法运算：从被乘数中一次取一位（先取低位，后取高位），与乘数相乘并进行十进制乘法调整，得到一个部分积在 AX 中（其中 AH 中是部分积的高位，AL 中是部分积的低位）。调整后的低位要加上前一次的高位，再进行加法调整，然后进行存储，如果加法调整中有进位，则被自动地加到部分积的高位 AH 中，将 AH 中的数存放在结果单元中，以备下一位乘法操作以后作为部分积的高位所使用。

程序清单

```
STACK       SEGMENT     STACK 'STACK'
            DB          128 DUP (0)
```

```
STACK     ENDS
DATA      SEGMENT
DATA1     DB        6，5，3，8
DATA2     DB        9
DATA3     DB        5 DUP (0)
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS：CODE，DS：DATA，ES：DATA
UNPACK    PROC      FAR
          MOV       AX，DATA
          MOV       DS，AX
          MOV       ES，AX
          MOV       SI，OFFSET DATA1
          MOV       DI，OFFSET DATA3
          MOV       CX，4
          CLC
RETRY1：  LODSB
          MUL       DATA2           ；求部分积
          AAM                       ；十进制乘法调整
          ADD       AL，[DI]        ；加上前一次的高位
          AAA                       ；十进制加法调整
          STOSB                     ；存部分积的低位
          MOV       [DI]，AH        ；存部分积的高位
          LOOP      RETRY1
PROGEND：MOV        AH，4CH
          INT       21H
UNPACK    ENDP
CODE      ENDS
          END       UNPACK
```

【例 5.4】 未组合的十进制数除法

下面的程序能够完成一个多位的未组合的十进制数与一个一位的未组合的十进制数的除法。

编程指导：

一个多位的未组合的十进制数与一个一位的未组合的十进制的乘法应该按照如下的方法进行除法运算：从被除数中一次取一位（先取高位，后取低位）送 AL 字存器中，将上一次的余数送 AH 字存器中，然后利用 AAD 指令将 AX 中的两位未组合的被除数进行校正，接着除以除数，得到一个部分商在 AL 中，余数在 AH，最后将部分商存入相应的内存单元中。这种过程一直进行，直到除完最后一位，将最后的余数存放在商的后面。

程序清单

```
STACK     SEGMENT   STACK 'STACK'
          DB        128 DUP (0)
STACK     ENDS
DATA      SEGMENT
DATA1     DB        8, 4, 5, 2              ; 被除数 8452
DATA2     DB        5                       ; 除数
DATA3     DB        5 DUP (0)               ; 存放 4 位商和最后的余数
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA, ES: DATA
UNPACK    PROC      FAR
          MOV       AX, DATA
          MOV       DS, AX
          MOV       ES, AX
          MOV       SI, OFFSET DATA1
          MOV       DI, OFFSET DATA3
          MOV       CX, 4
          CLC
          XOR       AH, AH                  ; 被除数的高位送初值 0
RETRY1:   LODSB                             ; 取被除数中的一位
          AAD                               ; AX 中未组合的 BCD 码转换成二进制码
          DIV       DATA2
          STOSB                             ; 存部分商，余数在 AH 中
          LOOP      RETRY1
          MOV       [DI], AH                ; 存储最后的余数
          MOV       AH, 4CH
          INT       21H
UNPACK    ENDP
CODE      ENDS
          END       UNPACK
```

二、组合的十进制数算术运算

【例 5.5】 多位组合的十进制数加法

内存中从 FIRST 和 SECOND 开始的单元中分别存放着两个 3 字节组合的十进制数，数据存放的规则是，低位在低地址，高位在高地址。试编程求这两个数的和，并存放到从 THIRD 开始的内存单元中。

程序清单

```
数据段：
FIRST     DB      99H，93H，82H，0        ；十进制数 825123
SECOND    DB      99H，99H，90H，0        ；十进制数 902647
THIRD     DB      4 DUP (0)
代码段：
          MOV     SI，OFFSET FIRST
          MOV     BX，OFFSET SECOND
          MOV     DI，OFFSET THIRD
          MOV     CX，3+1
          CLC
          CLD
RETRY：   LODSB
          ADC     AL，[BX]                ；对应的两位进行加法运算
          DAA
          INC     BX
          STOSB                           ；保存当前字节的和
          LOOP    RETRY
          MOV     AH，4CH
          INT     21H
```

【例 5.6】 多位组合的十进制数减法

内存中从 FIRST 和 SECOND 开始的单元中分别存放着两个 3 字节组合的十进制数，数据存放的规则是，低位在低地址，高位在高地址。试编程求这两个数的差，并存放到从 THIRD 开始的内存单元中。要求：如果被减数大于减数，则将“+”号存放在结果的最后，否则，用减数减去被减数，并将“－”存放在结果的最后。

程序清单

```
数据段：
FIRST     DB      67H，83H，32H           ；十进制数 328367
SECOND    DB      41H，54H，59H           ；十进制数 595441
THIRD     DB      0，0，0，'+'
代码段：
          MOV     SI，OFFSET FIRST
          MOV     BX，OFFSET SECOND
          MOV     DI，OFFSET THIRD
GOON：    MOV     CX，3                   ；进行 3 次减法
          CLC
          CLD
```

```
RETRY:   LODSB
         SBB     AL,[BX]          ;对应的一位进行减法运算
         DAS
         STOSB                    ;保存当前位的差
         INC     BX
         LOOP    RETRY
         JC      MANUS            ;被减数小于减数，转 MANUS
         JMP     PROGEND          ;和为正数，转结束
MANUS:   MOV     AL,'-'           ;传送符号标志到结果的最后一位
         STOSB
         MOV     SI,OFFSET SECOND ;减数当作被减数
         MOV     BX,OFFSET FIRST  ;被减数当作减数
         MOV     DI,OFFSET THIRD
         MOV     CX,3
         CLC
RETRY1:  LODSB
         SBB     AL,[BX]
         DAS
         STOSB
         INC     BX
         LOOP    RETRY1
PROGEND: MOV     AH,4CH
         INT     21H
```

第二节　代码转换编程实例

在程序设计中，经常需要进行代码转换。常用的代码转换有 BCD 转二进制，二进制转 BCD 码；ASCII 码转 BCD 码，BCD 码转 ASCII 码；二进制转 ASCII 码，ASCII 码转二进制等。

【例 5.7】BCD 码转二进制数

编程将 AX 中的四位 BCD 码转换成二进制数，转换的结果放在 AX 中。

编程指导：

所谓 BCD 码，即用 4 位二进制数表示一个十进制数的编码。

例如：AX＝9827H 即为九千八百二十七，程序设计的方法是：

〔（千位数×10＋百位数）×10＋十位数〕×10＋个位数

程序清单

数据段：

```
W10      DW      10              ;十进制数权值
```

```
代码段：
；入口参数   AX=BCD 码
；出口参数   AX=二进制数
AXBCDTO2  PROC   NEAR
          PUSH   BX
          PUSH   CX
          PUSH   DX
          MOV    BX，AX      ；保存 AX 中的 BCD 码到 BX
          MOV    AX，0       ；结果单元清零
          MOV    CX，4       ；共处理四位 BCD 码
RETRY：   PUSH   CX
          MOV    CL，4
          ROL    BX，CL      ；一位 BCD 码移到 BX 中的低半字节
          POP    CX
          MUL    W10         ；累加和乘以权值送 AX
          PUSH   BX
          AND    BX，0FH     ；屏蔽 BX 的高半字节
          ADD    AX，BX      ；累加下一位 BCD 码
          POP    BX
          LOOP   RETRY
          POP    DX
          POP    CX
          POP    BX
          RET
AXBCDTO2  ENDP
```

【例 5.8】 二进制数转 BCD 码

编程将 AX 中的二进制数转换成四位 BCD 码，转换的结果放在 AX 中（设 AX 中的数值小于十进制数 10000）。

编程指导：

把 AX 中的二进制数除以 1000 得到的商是千位上的 BCD 码，所得余数除以 100 得到的商是百位上的 BCD 码，所得余数再除以 10 得到的商是十位上的 BCD 码，最后所得的余数是个位上的 BCD 码。

程序清单

```
数据段：
W1000     DW     1000，100，10，1        ；十进制数千，百，十，个位权值
代码段：
；入口参数   AX=二进制数
```

```
;出口参数   AX=BCD码
AX2TOBCD  PROC   NEAR
          PUSH   SI
          PUSH   BX
          PUSH   CX
          PUSH   DX
          XOR    BX, BX              ;BCD码暂存单元清零
          MOV    SI, OFFSET W1000    ;权值首地址送SI
          MOV    CX, 4               ;循环次数4送CX
RETRY:    PUSH   CX
          MOV    CL, 4
          SHL    BX, CL
          MOV    DX, 0
          DIV    WORD PTR [SI]       ;除以权值
          OR     BX, AX              ;组合BCD码
          MOV    AX, DX              ;余数送AX
          POP    CX
          INC    SI
          INC    SI                  ;地址加2，指向下一权值
          LOOP   RETRY
          MOV    AX, BX              ;BCD码由BX送AX
          POP    DX
          POP    CX
          POP    BX
          POP    SI
          RET
AX2TOBCD  ENDP
```

【例5.9】 ASCII码转BCD码

设SI所指的内存单元中有四个ASCII码，试把其转换成四位BCD码，转换的结果放在BX所指的内存单元中。

编程指导：

两个ASCII码能组合成一字节的BCD码（两个码），四个ASCII码转换成BCD码，共组合两个字节的BCD码。程序采用两重循环，内循环执行一次可以把两个ASCII码组合成一字节的BCD码。这样，外循环执行两次即可把四个ASCII码组合成两字节的BCD码。

程序清单

数据段：

```
ASCBUF    DB '3295'
```

```
BCDBUF      DB 2 DUP (0)
代码段:
; 入口参数   SI=ASCII 码首地址
;            BX=BCD 码首地址
; 出口参数   BX=BCD 码首地址
ASCTOBCD    PROC    NEAR
            PUSH    BX
            MOV     CX, 2                ; 装配两字节的 BCD 码
ATOB1:      PUSH    CX
            MOV     CX. 2                ; 内循环计数, 一字节装配两个 BCD 码
ATOB2:      PUSH    CX
            MOV     CL, 4
            SHL     BYTE PTR [BX], CL    ; 低 4 位清零, 以便组合下一 BCD 码
            MOV     AL, [SI]             ; 取一 ASCII 码送 AL
            SUB     AL, 30H              ; 转换为 BCD 码
            OR      [BX], AL             ; 组合一个 BCD 码
            INC     SI                   ; ASCII 码地址加 1
            POP     CX
            LOOP    ATOB2                ; 一字节 BCD 码未组合完. 转 ATOB2
            INC     BX
            POP     CX
            LOOP    ATOB1                ; 两字节的 BCD 码未组合完, 转 ATOB1
            POP     BX
            RET
ASCTOBCD    ENDP
```

【例 5.10】 BCD 码转 ASCII 码

编程将 AL 中的两位 BCD 码换成相应的 ASCII 码,转换的结果放在 BX 所指向的连续的两个内存单元中。

编程指导:

把 AL 中的两个 BCD 码进行分离, 变成两个十进制数 (0～9). 然后分别加上 30H。

程序清单

```
数据段:
ASCBUF      DB      2 DUP (0)
代码段:
; 入口参数   AL=BCD 码
;            BX=ASCII 码首地址
; 出口参数   BX=ASCII 码首地址
```

```
BCDTOASC  PROC   NEAR
          PUSH   CX
          PUSH   BX
          MOV    CX，2          ；一字节需处理 2 次
BTOA1：   PUSH   CX
          MOV    CL，4
          ROL    AL，CL         ；左环移 4 位
          PUSH   AX
          AND    AL，0FH        ；截取低 4 位
          OR     AL，30H        ；0～9 变 ASCII 码
          MOV    [BX]，AL
          INC    BX             ；ASCII 码缓冲区地址加 1
          POP    AX
          POP    CX
          LOOP   BTOA1
          POP    BX
          POP    CX
          RET
BCDTOASC  ENDP
```

【例 5.11】 二进制数转 ASCII 码

编程将 AX 中的二进制数转换成 ASCII 码，转换的结果放在从 ASCBUF 开始的连续的 5 个内存单元中。

编程指导：

AX 中的二进制数最大数值为 65536，转换为 ASCII 码需 5 个字节单元。程序首先将 AX 中的数除以 10，所得余数为个位上的数，加上 30H 变为相应的 ASCII 码；所得的商再作为被除数除以 10，得到的余数为十位上的数，加上 30H 变为相应的 ASCII 码；所得的商再作为被除数除以 10，得到的余数为百位上的数，……，直到被除数小于 10 时，得到最后的一位数。

程序清单

数据段：

```
ASCBUF    DB     5 DUP (0)
```

代码段：

```
；入口参数   AX＝二进制数
；出口参数   无
BINTOASC  PROC   NEAR
          MOV    CX，10              ；除数送 CX
          LEA    SI，ASCBUF＋4       ；SI 指向 ASCII 码个位数地址
BTOA1：   CMP    AX，10              ；二进制数小于 10 吗？
```

```
        JB      BTOA2           ; 小于 10 转 BTOA2
        XOR     DX, DX          ; 被除数高字清零
        DIV     CX              ; 除 10
        OR      DL, 30H         ; 余数变 ASCII 码
        MOV     [SI], DL        ; 存一字节 ASCII 码
        DEC     SI              ; ASCII 码地址减 1
        JMP     BTOA1
BTOA2:  OR      AL, 30H
        MOV     [SI], AL        ; 存最高位的 ASCII 码
        RET
BINTOASC ENDP
```

【例 5.12】 ASCII 码转二进制数

从 ASCBUF 开始的内存单元中有 4 个连续的 ASCII 码，要求把它们转换为二进制数放在 BINVAL 单元中。

编程指导：

把 4 个 ASCII 码分别转换为 0～9 的数，然后分别乘以各自的权值最后累加起来即得到相应的二进制。

程序清单

数据段：

```
ASCBUF      DB      '6472'          ; ASCII 码字符串
ASCLEN      DB      $-ASCBUF        ; ASCII 码字符个数
BINVAL      DW      0               ; 二进制数结果
MULT10      DW      1               ; 十进制权值存放单元
```

代码段：

```
ASCTOBIN    PROC    NEAR
            MOV     CX, 10          ; 乘法因子送 CX
            LEA     SI, ASCBUF-1    ; ASCII 码首地址减 1 送 SI
            MOV     BX, ASCLEN      ; ASCII 码字符个数送 BX
ATOB1:      MOV     AL, [SI+BX]     ; 取一个 ASCII 码（从后往前）
            AND     AX, 000FH       ; 转二进制数
            MUL     MULT10          ; 乘当前权值
            ADD     BINVAL, AX      ; 送累加和单元
            MOV     AX, MULT10
            MUL     CX              ; 计算下一位的权值
            MOV     MULT10, AX      ; 改变当前权值
            DEC     BX              ; 计数减 1
            JNZ     ATOB1           ; 未处理完，转 ATOB1
```

```
        RET
ASCTOBIN  ENDP
```

第三节　表的处理编程实例

表是由若干数据项目组成的集合，而项目一般包括项目编号和项目内容两部分。例如一个由学号与学生姓名组成的“学号-姓名”对照表如表 5-1。

其中学号是该表的项目编号，而姓名即为该表的项目内容。当然，不是所有的表都必须有“项目编号”和“项目内容”，在实际问题中，有的表包含这两个部分，而有的表仅包含“项目内容”。

对于一个表来说，根据表中“项目编号”或“项目内容之间的大小关系，可以把表划分为“有序表”和“无序表”两种。

表的应用是很广泛的，所以程序设计中对表的处理也是经常碰到的问题。一般来说，对表的处理包括查找、插入、删除、替换、排序操作等。下面分别举例说明。

表 5-1　学号-姓名对照表

学号	姓名
890001	王小明
890002	李　明
890003	赵丽丽
……	……

一、查表

(一) 顺序查表法

顺序查表法是指从给定的表首地址开始从前往后依次查找指定项目的方法。适合于对无序表的查找。但是，如果有序表中的数据项目比较少，则也可以采用这种查找方法进行查找。

【例 5.13】 用 CMPSB/CMPSW 进行查表

设有某班学生的“姓名-学号”对照表，其表的首地址为 NAME _ NUM _ TAB，表中的数据项按照姓氏笔划排列，试编写一程序，根据 STUDENT _ NAME 中指定的学生的姓名查表求得其对应的学号，放于从 STUDENT _ NUMBEL 开始的单元中。

程序清单

```
STACK       SEGMENT   PARA   STACK 'STACK'
            DB        128    DUP (0)
STACK       ENDS
DATA        SEGMENT
STUDENT _ NAME     DB '王浩然'
STUDENT _ NUMBEL   DB 6 DUP (0)
NAME _ NUM _ TAB   DB '王小强','890031'
                   DB '王　红','890006'
                   DB '王浩然','890010'
                   DB '张明明','890058'
                   DB '张　超','890030'
                   DB '李　强','890042'
STUDENT _ COUNT    EQU ($-NAME _ NUM _ TAB) / (6+6)
DATA        ENDS
```

```
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA, ES: DATA
START   PROC   FAR
        PUSH   DS
        XOR    AX, AX
        PUSH   AX
        MOV    AX, DATA
        MOV    DS, AX
        MOV    ES, AX
        CLD
        LEA    SI, NAME_NUM_TAB
        MOV    CX, STUDENT_COUNT
RETRY:  PUSH   CX
        LEA    DI, STUDENT_NAME
        MOV    CX, 6
        REPE   CMPSB
        JE     FOUND                ; 找到，转 FOUND
        ADD    SI, CX
        ADD    SI, 6                ; SI 指向下一个姓名地址
        POP    CX
        LOOP   RETRY                ; 表未查找完，转 RETRY
        MOV    AL, 0FFH
        MOV    CX, 6
        LEA    DI, STUDENT_NUMBEL
        REP    STOSB                ; 未找到，学号各位置 0FFH
        RET
FOUND:  POP    CX
        LEA    DI, STUDENT_NUMBEL
        MOV    CX, 6
        REP    MOVSB                ; 找到，传送学号
        RET
START   ENDP
        CODE   ENDS
        END    START
```

（二）计算查表法

计算查表法适用于有序表，而且表中的各个“项目编号”是连续递增（或递减）的关系。一般来说，此种表的“项目编号”不定义在表中，而是隐含的。查找的关键就是要找到被查项的地址，而每个“项目内容”的地址是可以根据“项目编号”用一个公式计算出来的。

【例 5.14】 查找月的英文缩写

编写一个程序根据 MONTH _ NUM 中给定的月份号，在 MONTH _ NAME _ TAB 表中查找其对应的英文缩写词，查找的结果放于 MONTH _ NAME 单元中。

编程指导：

由于每一月的英文缩写为三个字符长度，所以要查找某一月的英文缩写词所在的内存单元地址，可以通过公式：月缩写词表首地址＋（月－1）×3 计算出来。

程序清单

```
STACK     SEGMENT    PARA STACK 'STACK'
          DB         128 DUP (0)
STACK     ENDS
DATA      SEGMENT
MONTH_NUM            DB 10
MONTH_NAME           DB 4 DUP ('$')
MONTH_NAME_TAB       DB 'JAN','FEB','MAR','APR','MAY','JUN'
                     DB 'JUL','AUG','SEP','OCT','NOV','DEC'
DATA      ENDS
CODE      SEGMENT
          ASSUME     CS: CODE, DS: DATA, ES: DATA
START     PROC  FAR
          PUSH  DS
          XOR   AX, AX
          PUSH  AX
          MOV   AX, DATA
          MOV   DS, AX
          MOV   ES, AX
          MOV   AL, MONTH_NUM          ; 月送 AL
          CMP   AL, 1                  ; 月<1，转出口
          JB    EXIT
          CMP   AL, 12                 ; 月>12，转出口
          JA    EXIT
          DEC   AX                     ; 月－1
          MOV   AH, 0
          MOV   SI, AX
          ADD   AX, AX                 ; 求出该月对应的缩写词在表中
          ADD   AX, SI                 ; 的偏移：（月－1）×3→AX
          LEA   SI, MONTH_NAME_TAB     ; 表首地址→SI
          ADD   SI, AX                 ; 求该月缩写词在表中的地址
          LEA   DI, MONTH_NAME
```

```
        MOV     CX，3
        REP     MOVSB                    ；传送该月的缩写词
EXIT：  RET
START   ENDP
        CODE    ENDS
        END     START
```

实现计算查表的另一种方法是利用 XLAT 指令。但是这种查表方法要求表中的各个数据项必须以字节为单位，而且整个表的数据项不得超过 256 个。

在传统密码体制中，有一种称为代替密码的加密方法，此种加密方法的原理是：将明文中的字符按照某种规律，用新的字符加以替换，而得到密文。例如，我们可以把一个只含有英文字符的源程序文件中 ASCII 码值在 20H～7EH 之间的所有字符，按照表 5－2 中的对应关系进行逐一替换，这样即得到相应的密码。

表 5－2　基本 ASCII 码明—密对照表

明　文	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
密　文	G	F	E	D	C	B	A	g	f	e	d	c	b	a	n	m	l	k	j	i	h	N	M	L	K	J	I	H	O	P	Q	R
明　文	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
密　文	S	T	t	s	r	q	p	o	3	2	1	0	9	8	7	6	4	5	@	$	%	&	*	(	)	!	#	^	U	V	W	X
明　文	‘	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
密　文	Y	Z	x	y	z	u	v	w	[	‘	]	'	;	:	"	~	<	>	?	/	,	.	\	!	{	=	}		+	-	_	

【例 5.15】 利用 XLAT 指令查表加密

利用上面的明密替换表，对缓冲区 BUF1 中的 ASCII 码在 20H～7E 之间的字符进行替换加密，替换的结果（密文）放于缓冲区 BUF2 中。设缓冲区的长度为 2000 字节。

程序清单

```
STACK       SEGMENT    PARA STACK 'STACK'
            DB         128 DUP (0)
STACK       ENDS
DATA        SEGMENT
KEY_TAB     DB    'GFEDCBAgfedcbanmlkjihNMLKJIHOPQR'
            DB    'STtsrqpo3210987645@$%&* ()! #^UVWX'
            DB    'YZxyzuvw ['']';:"~<>?/..\|{=}+-_'
BUF1        DB    2000 DUP (0)
BUF2        DB    2000 DUP (0)
DATA        ENDS
CODE        SEGMENT
            ASSUME     CS：CODE，DS：DATA，ES：DATA
START       PROC       FAR
```

```
        PUSH    DS
        XOR     AX，AX
        PUSH    AX
        MOV     AX，DATA
        MOV     DS，AX
        MOV     ES，AX
        LEA     SI，BUF1
        LEA     DI，BUF2
        MOV     CX，2000
        LEA     BX，KEY _TAB
        CLD
RETRY:  LODSB                   ；取一字节明文
        CMP     AL，20H
        JB      NEXT
        CMP     AL，7EH
        JA      NEXT
        SUB     AL，20H         ；字符的 ASCII 码减 20H
        XLAT                    ；查表替换
NEXT:   STOSB                   ；存密文
        LOOP    RETRY
EXIT:   RET
START   ENDP
        CODE    ENDS
        END     START
```

（三）折半查表法

折半查找的具体方法见下例。此种查表方法要求表中的“项目编号”是有序的但可以是不连续的，适合于表中的数据项较多的情况，是一种快速查找有序表的方法。

【例 5.16】 折半查找

从内存单元 TAB 开始有若干个以字为单位的有序数据，编写一个程序用折半查找的方法，查找该有序表中是否有关键字 key。若查找到，则把与关键字 key 相等的元素的地址送 addr 单元，并且把进位标志置 0；否则，把进位标志置 1。

编程指导：

折半查找的过程是：先确定待查记录的范围（区间），然后逐步缩小范围直到找到或找不到该记录为止。例如：已知如下 9 个数据元素的有序表：

(07，15，38，43，55，64，77，89，102)

现在所要查找的关键字为 38。为了进行查找，用两个指针 low 和 hig 分别指示待查元素所在范围的下界和上界，并用指针 mid 指示中间元素，即 mid＝［(low＋hig) /2］。在开始进行查找时，low 和 hig 的初值分别指向第一个和第 9 (n) 个元素。下面是查找 key＝38 的过程。由于

low＝1，hig＝9，所以求得 mid＝［（low＋hig）/2］＝［（1＋9）/2］＝5。

序号：	1	2	3	4	5	6	7	8	9
元素：	07	15	38	43	55	64	77	89	102
	↑				↑				↑
	low				mid				hig

先用 r［mid］. key 和 key 相比，因为 r［mid］. key＞key，说明待查的元素若存在，则必在 low 和 mid－1 范围内，所以令指针 hig 指向第 mid－1 个元素，重新求得 mid＝［（1＋4）/2］＝2。

07	15	38	43	55	64	77	89	102
↑	↑		↑					
low	mid		hig					

仍用 r［mid］. key 和 key 进行比较，因为 r［mid］. key＜key，说明待查元素若存在，则必在 mid 和 hig 之间，所以令指针 low 指向第 mid＋1 个元素，求得 mid 的新值［（3＋4）/2］＝3。

07	15	38	43	55	64	77	89	102
		↑	↑					
		low	hig					
		mid						

比较 r［mid］. key 和 key，因为相等，则查找成功，所查找到的元素在表中的序号等于 mid 指针的值。

在下面的程序设计中，用 SI 和 DI 分别表示 low 和 hig 指针，而用 BX 表示 mid 指针。

程序清单

```
STACK     SEGMENT    PARA STACK ' STACK '
          DB         128 DUP (0)
STACK     ENDS
DATA      SEGMENT
key       DW         38          ; 要查找的关键字
addr      DW         ?
TAB       DW         07, 15, 38, 43, 55, 64, 77, 89, 102
BYTE _ COUNT   EQU    $ - TAB                    ; 表字节计数
TAB _ LENGTH   DW   BYTE _ COUNT/ (TYPE TAB)     ; 表长
DATA      ENDS
CODE      SEGMENT
          ASSUME     CS: CODE, DS: DATA, ES: DATA
START     PROC   FAR
          PUSH   DS
          XOR    AX, AX
          PUSH   AX
          MOV    AX, DATA
```

```
        MOV     DS, AX
        MOV     ES, AX
        MOV     AX, key                              ;要查找的关键字→AX
        MOV     SI, OFFSET TAB+0                     ; 表的首地址→SI
        MOV     DI, OFFSET TAB+BYTE_COUNT-2          ; 表的末地址→DI
        CALL    binsrch                              ; 调用折半查找子程序
        RET
START   ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; binsrch 入口参数：SI=表的首地址
;                   DI=表的末地址
;                   AX=要查找的关键字
;         出口参数：CY=0 查找成功，SI=与关键字相等的元素的地址
;                   CY=1 查找失败，SI=与关键字最接近的元素的地址
;    使用的寄存器：BX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
binsrch    PROC    NEAR
RETRY:     CMP     SI, DI
           JB      NEXT3             ; low<hig, 转 NEXT3
           JE      NEXT1             ; low=hig, 转 NEXT1
EXIT:      STC                       ; 查找不成功，进位置 1
           RET                       ; 返回主程序
NEXT1:     CMP     AX, [SI]
           JZ      NEXT2             ; 已查找到，转 NEXT2
           JMP     SHORT EXIT        ; 未查找到，转 EXIT
NEXT2:     CLC                       ; 查找成功，进位置 0
           MOV     addr, SI          ; 传送元素地址
           RET                       ; 返回主程序
NEXT3:     MOV     BX, DI
           SUB     BX, SI
           SHR     BX, 1
           AND     BX, 0FFFEH        ; 保证 BX 中的值为偶数
           ADD     BX, SI            ; 求得 mid→BX
           CMP     AX, [BX]
           JA      NEXT4             ; r [mid]. key<key, 转 NEXT4
           JB      NEXT5             ; r [mid]. key>key, 转 NEXT5
           MOV     SI, BX            ; r [mid]. key=key, mid→low
           JMP     SHORT NEXT2
NEXT4:     MOV     SI, BX
```

```
            ADD     SI, 2              ; mid+1→low
            JMP     SHORT RETRY
NEXT5:      MOV     DI, BX
            SUB     DI, 2              ; mid-1→hig
            JMP     SHORT RETRY
binsrch     ENDP
CODE        ENDS
            END     START
```

二、给表中插入一个元素

【例 5.17】 有序表中插入一个元素

给有序表 TAB 中插入一个元素 key，如果该元素不在表中的话。

编程指导：

首先用 binsrch 子程序查找一下表 TAB，根据其出口参数判断表中是否有该元素，若有此元素，则返回 DOS；若无此元素，则把表中比关键字大的所有元素向后移动以便留出一个字的单元，然后把关键字插入到其中，最后修改表长（加 1）返回 DOS。

程序清单

```
STACK       SEGMENT   PARA STACK 'STACK'
            DB        128 DUP (0)
STACK       ENDS
DATA        SEGMENT
key             DW    40                          ; 要插入的关键字
addr            DW    ?
TAB             DW    07, 15, 38, 43, 55, 64, 77, 89, 102
BYTE_COUNT      EQU   $-TAB                       ; 表的字节计数
                DB    10 DUP (0)                  ; 预留 10 个字节
TAB_LENGTH      DW    BYTE_COUNT/(TYPE TAB)       ; 表长
DATA        ENDS
CODE        SEGMENT
            ASSUME    CS: CODE, DS: DATA, ES: DATA
START       PROC      FAR
            PUSH      DS
            XOR       AX, AX
            PUSH      AX
            MOV       AX, DATA
            MOV       DS, AX
            MOV       ES, AX
            MOV       AX, key                   ; 要查找的关键字→AX
```

```
                MOV     SI, OFFSET TAB+0                      ; 表的首地址→SI
                MOV     DI; OFFSET TAB+BYTE_COUNT-2           ; 表的末地址→DI
                CALL    binsrch                               ; 调用折半查找子程序
                JC      INSERT_WORD                           ; 未找到，转插入操作
                RET                                           ; 找到，返回 DOS
INSERT_WORD:    CMP     AX, [SI]                              ; 关键字与当前元素比较
                JB      INSERT1                               ; 关键字小于当前元素，转
                ADD     SI, 2                                 ; 当前地址指针+2
INSERT1:        PUSH    SI                                    ; 保存当前地址指针
                LEA     CX, TAB+BYTE_COUNT
                SUB     CX, SI
                SHR     CX, 1                                 ; 求出往后搬移的字计数
                LEA     DI, TAB+BYTE_COUNT                    ; 设置目的地址指针
                LEA     SI, TAB+ (BYTE_COUNT-2)               ; 设置源地址指针
                STD                                           ; 置方向标志为逆向
                REP     MOVSW                                 ; 搬移
                POP     SI                                    ; 恢复 SI
                MOV     [SI], AX                              ; 插入关键字
                INC     TAB_LENGTH                            ; 表长度+1
                RET                                           ; 返回 DOS
START           ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; binsrch 入口参数：SI=表的首地址
;                  DI=表的末地址
;                  AX=要查找的关键字
;        出口参数：CY=0 查找成功，SI=与关键字相等的元素的地址
;                  CY=1 查找失败，SI=与关键字最接近的元素的地址
;    使用的寄存器：BX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
binsrch      PROC    NEAR
             ……
             ……
binsrch      ENDP
CODE         ENDS
             END     START
```

三、删除表中的一个元素

【例 5.18】 有序表中删除一个元素

从有序表 TAB 中删除一个元素 key，如果该元素在表中的话。

编程指导：

首先用 binsrch 子程序查找一下表 TAB，根据其出口参数判断表中是否有该元素，若无此元素，则返回 DOS；若有此元素，则把表中比关键字大的所有元素向前移动一个字的单元，从而把关键字从其中删除，最后修改表长（减 1）返回 DOS。

程序清单

```
STACK       SEGMENT     PARA STACK 'STACK'
            DB          128 DUP (0)
STACK       ENDS
DATA        SEGMENT
key             DW      43                                  ; 要删除的关键字
addr            DW      ?
TAB             DW      07, 15, 38, 43, 55, 64, 77, 89, 102
BYTE_COUNT      EQU     $-TAB                               ; 表的字节数
                DB      10 DUP (0)                          ; 预留 10 个字节
TAB_LENGTH      DW      BYTE_COUNT/ (TYPE TAB)              ; 表长度
DATA        ENDS
CODE        SEGMENT
            ASSUME      CS: CODE, DS: DATA, ES: DATA
START       PROC    FAR
            PUSH    DS
            XOR     AX, AX
            PUSH    AX
            MOV     AX, DATA
            MOV     DS, AX
            MOV     ES, AX
            MOV     AX, key                             ; 要查找的关键字→AX
            MOV     SI, OFFSET TAB+0                    ; 表的首地址→SI
            MOV     DI, OFFSET TAB+BYTE_COUNT-2         ; 表的末地址→DI
            CALL    binsrch                             ; 调用折半查找子程序
            JNC     DELETE_WORD                         ; 找到，转删除操作
            RET                                         ; 未找到，返回 DOS
DELETE_WORD: LEA    CX, TAB+BYTE_COUNT-2
            SUB     CX, SI
            SHR     CX, 1                               ; 求出搬移的字计数
            MOV     DI, SI                              ; 设置目的地址
            ADD     SI, 2                               ; 设置源地址
            CLD                                         ; 置正向传送标志
            REP     MOVSW                               ; 搬移
```

```
        DEC     TAB_LENGTH                          ; 表计数值－1
        RET                                         ; 返回 DOS
START   ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; binsrch 入口参数：SI＝表的首地址
;                  DI＝表的末地址
;                  AX＝要查找的关键字
;        出口参数：CY＝0 查找成功，SI＝与关键字相等的元素的地址
;                  CY＝1 查找失败，SI＝与关键字最接近的元素的地址
;   使用的寄存器：BX
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
binsrch PROC    NEAR
        ……
        ……
binsrch ENDP
CODE    ENDS
        END     START
```

四、求表中的最大元素

【例 5.19】 求无序表中的最大元素

从 BLOCK 开始的内存缓冲区中有若干个以字节为单位的无符号数，试编程求它们之中的最大值放在 MAX 单元中。

编程指导：

将第一个数放入 AL 中，依次取第二个、第三个数、……，与 AL 中的数进行比较，在每一次比较之后，都将其中最大的数放在 AL 中。这样比较完所有的数以后，即在 AL 中得到它们当中的最大数。

程序清单

```
STACK   SEGMENT   STACK 'STACK'
        DB        128 DUP (0)
STACK   ENDS
DATA    SEGMENT
BLOCK   DB        34, 23, 9, 145, 210, 198, 46, 87, 16, 182
COUNT   EQU       $-BLOCK
MAX     DB        ?
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA
MAXM    PROC    FAR
```

```
        MOV     AX，DATA
        MOV     DS，AX
        MOV     BX，OFFSET BLOCK
        MOV     AL，[BX]
        INC     BX
        MOV     CX，COUNT－1
AGAIN：  CMP     AL，[BX]
        JA      NEXT                ；AL 中的数大，直接转 NEXT
        MOV     AL，[BX]            ；AL 中的数小，将大数放在 AL 中
NEXT：   INC     BX
        LOOP    AGAIN
        MOV     MAX，AL             ；将最大的一个无符号数送 MAX 单元
        MOV     AH，4CH
        INT     21H
MAXM    ENDP
CODE    ENDS
        END     MAXM
```

五、对无序表中的元素排序

【例 5.20】 冒泡排序

利用冒泡排序方法对内存单元 ARRAY 开始的以字为单位的数据进行排序。

编程指导：

冒泡排序的方法是：首先将无序表中的第一个记录的关键字和第二个记录的关键字进行比较，若 r[1].key>r[2].key，则将这两个记录内容进行交换，否则不交换。然后比较第二个和第三个记录的关键字，依此类推，直到第 n－1 记录和第 n 个记录进行比较、交换后为止。经过这样一趟排序，即把表中最大的关键字查找出来并放于最后一个记录的位置上。然后，对前 n－1 个记录进行同样的比较、交换，操作完成后则把具有次大的关键字记录放于第 n－1 个记录的位置上。重复以上过程直至没有记录交换为止。这样即得到一个数据按由小到大排序的表。下列是对有 7 个元素的无序表进行冒泡排序的过程。

表的初始状态：	[49	38	65	97	76	13	27]
第一趟排序之后：	[38	49	65	76	13	27]	97
第二趟排序之后：	[38	49	65	13	27]	76	97
第三趟排序之后：	[38	49	13	27]	65	76	97
第四趟排序之后：	[38	13	27]	49	65	76	97
第五趟排序之后：	[13	27]	38	49	65	76	97
第六趟排序之后：	13	27	38	49	65	76	97

下面是冒泡程序的具体实现。在排序过程中，为了避免不必要的排序，程序设置了一个标志单元 XCHG _FLAG，其初值被置为 0FFH，表示被排序的表是一个无序的表。在每一趟排序开始时，首先检测此标志，若此标志为 0，则结束排序；若此标志为 0FFH，则准备进行排

序，并预置此标志为 0，然后进行本趟的排序比较，若在本趟两两比较后，发生了交换，则重新置标志为 0FFH，以便在下一趟排序时检测此标志，继续排序。若两两比较后，本趟未发生交换，则置标志为 0，表示数据已排好序，从而使下一次排序不再进行。

程序清单

```
STACK     SEGMENT    PARA STACK 'STACK'
          DB         128 DUP (0)
STACK     ENDS
DATA      SEGMENT
ARRAY          DW    49, 38, 65, 97, 76, 13, 27
WORD_COUNT     DW    ($-ARRAY)/2                ; 数据个数（以字为单位）
XCHG_FLAG      DB    0FFH                       ; 交换标志单元
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA, ES: DATA
bubble_sort    PROC  FAR
               PUSH  DS
               XOR   AX, AX
               PUSH  AX
               MOV   AX, DATA
               MOV   DS, AX
               MOV   ES, AX
RETRY:         CMP   XCHG_FLAG, 0
               JE    EXIT                     ; 上次未发生交换，转 EXIT
               MOV   XCHG_FLAG, 0             ; 预置交换标志单元为 0
               DEC   WORD_COUNT
               MOV   CX, WORD_COUNT           ; 本趟比较次数→CX
               MOV   SI, OFFSET ARRAY         ; 数据首地址→SI
               CLD
NEXT1:         LODSW                          ; 取一个数据→AX
               CMP   AX, [SI]                 ; 与下一个数据进行比较
               JLE   NEXT2                    ; 前一数据小，转 NEXT2
               XCHG  [SI], AX
               XCHG  [SI-2], AX               ; 后一数据小，进行交换
               MOV   XCHG_FLAG, 0FFH          ; 置交换标志为 0FFH
NEXT2:         LOOP  NEXT1                    ; 循环进行两两数据的比较
               JMP   SHORT RETRY              ; 转 RETRY，继续进行下一趟排序
EXIT:          RET                            ; 排序完成，返回 DOS
bubble_sort    ENDP
```

```
CODE        ENDS
            END     bubble _ sort
```

【例 5.21】 快速排序

利用数据结构中的快速排序算法编写一个程序，对内存单元 BUF 开始的以字节为单位的若干个无符号数进行排序。

编程指导：

快速排序的基本思想是通过一趟排序将文件分成两部分，然后分别对这两部分进行排序以达到最后整个文件的排序。它的具体做法是：任意选取文件中的一个记录（通常是文件中的第一个记录），以它为关键字和文件中的所有其余记录的关键字进行比较，将所有关键字较它小的记录都存放于它的前面；而将所有关键字较它大的记录都存放于它的后面，这样经过一趟排序之后，可按该记录所在位置为分界，将文件分为两部分。为了进行比较，可设两个指针 i 和 j（程序设计中用 SI 作为 i 指针，用 DI 作为 j 指针，关键字放在 AL 寄存器中），其初始状态分别指向文件中的第一个记录和最后一个记录。首先将第一个记录移至辅助变量 x 中，然后 j 自 n 起逐渐减小进行 r[j].key 和 x.key 的比较，直至找到满足 r[j].key＜x.key 的记录时，将 r [j] 移至 r [i] 的位置；再令 i 自 i+1 起逐渐增大进行 r[i].key 和 x.key 的比较，直至找到满足 r[i].key＞x.key 的记录时将 r [i] 移至 r [j] 的位置；之后 j 自 j－1 起重复上述过程，直至 i=j，此时 i 便是记录 x 所应存放的位置。至此，一趟快速排序完成，将文件分为 [1，i－1] 和 [i+1，n] 两部分。具体过程如图 5－1 所示。在一趟排序完成后，将所有记录分为两部分，再分别对这两部分重复上述的排序过程，直到每一部分中只剩下一个记录为止。其整体排序过程如图 5－2 所示。

由于排序过程是一层套一层进行的，所以，程序设计中，整个排序的过程采用了递归程序设计方法。

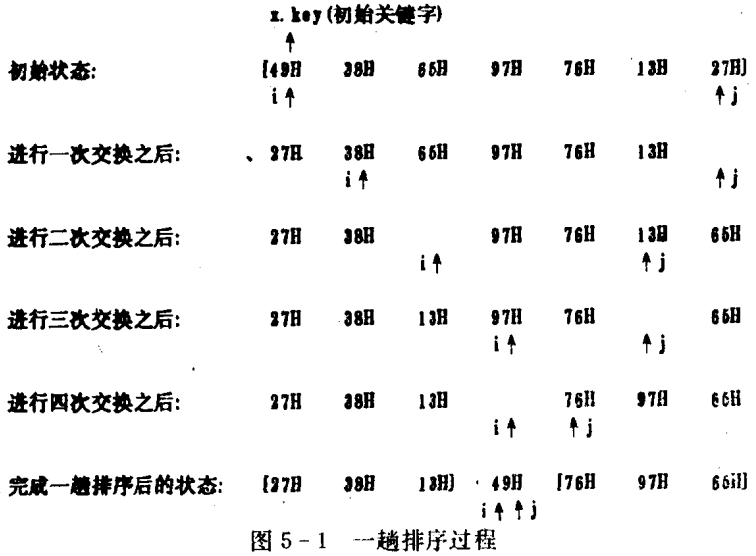


图 5－1 一趟排序过程

初始状态:	[49H	38H	66H	97H	76H	13H	27H]
一趟排序之后的状态:	[27H	38H	13H]	49H	[76H	97H	66H]
二趟前半部排序后的状态:	[13H]	27H	[38H]	49H	[76H	97H	66H]
二趟后半部排序后的状态:	[13H]	27H	[38H]	49H	[66H]	76H	[97H]
最后形成的有序文件:	[13H	27H	38H	49H	66H	76H	97H]

图 5-2　整体排序过程

程序清单

```
STACK   SEGMENT   PARA STACK 'STACK'
        DB        1024 DUP (0)
STACK   ENDS
DATA    SEGMENT
BUF     DB        49H, 38H, 65H, 97H, 76H, 13H, 27H
COUNT   EQU       $-BUF                       ; 数据个数
POINT   DW        ?                           ; 存放前后两半分隔指针的地址
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA, ES: DATA
START   PROC      FAR
        PUSH      DS
        XOR       AX, AX
        PUSH      AX
        MOV       AX, DATA
        MOV       DS, AX
        MOV       ES, AX
        MOV       SI, OFFSET BUF               ; 第一个数据地址→SI
        MOV       DI, OFFSET BUF+COUNT-1       ; 最后一个数据地址→DI
        MOV       AL, [SI]                     ; 关键字→AL
        CALL      QKSORT                       ; 调用快速排序子程序
        RET
START   ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; QKSORT 子程序注释:
;          入口参数: SI=缓冲区首地址
;                    DI=缓冲区末地址
;                    AL=关键字
```

```
;            出口参数：无
;            说    明：程序破坏 AH 中的内容
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
QKSORT       PROC     NEAR
             CMP      SI, DI
             JB       NEXT0                 ; 若首指针 i<尾指针 j, 转 NEXT0
             RET                            ; i=j, 一趟排序完成, 返回调用者
NEXT0:       PUSH     DI                    ; 尾指针进栈保护
             PUSH     SI                    ; 首指针进栈保护
NEXT1:       CMP      SI, DI
             JB       NEXT2                 ; 首指针 i<尾指针 j, 转 NEXT2
             JMP      SHORT SUBSORT         ; 若 i=j, 转 SUBSORT, 分别排序前后两半
NEXT2:       CMP      AL, [DI]
             JA       NEXT3                 ; x. key>r [j]. key, 转 NEXT3
             DEC      DI                    ; j=j-1
             JMP      SHORT NEXT1
NEXT3:       MOV      AH, [DI]              ; r [j]. key→AH
             MOV      [SI], AH              ; AH→r [i]. key
             INC      SI                    ; i=i+1
NEXT4:       CMP      SI, DI
             JB       NEXT5                 ; i<j, 转 NEXT5
             JMP      SHORT SUBSORT         ; 若 i=j, 转 SUBSORT, 分别排序前后两半
NEXT5:       CMP      [SI], AL
             JA       NEXT6                 ; r [i]. key>x. key, 转 NEXT6
             INC      SI                    ; i=i+1
             JMP      SHORT NEXT4
NEXT6:       MOV      AH, [SI]              ; r [i]. key→AH
             MOV      [DI], AH              ; AH→r [j]. key
             DEC      DI                    ; j=j-1
             JMP      SHORT NEXT1           ; 转 NEXT1, 直至一趟排序完成
SUBSORT:     MOV      [SI], AL              ; 一趟排序完, 关键字 x. key 送入正确位置
             MOV      POINT, SI             ; 保存指针 i (i=前后两半分隔指针的地址)
             CMP      SI, OFFSET BUF+0
             JE       SORT_NEXT_HALF        ; 分隔指针 i 等于表首地址, 表明前半部分
                                            ; 排序完成, 转 SORT_NEXT_HALF
             MOV      DI, SI
             DEC      DI                    ; i-1→j
             POP      SI                    ; 1→i
             CMP      SI, DI
```

```
                JNB     SUBSORT2            ; i>j，转 SUBSORT2
                MOV     AL，[SI]            ; x. key→AL
                CALL    QKSORT              ; 递归调用 QKSORT 子程序，排序前半部分
                JMP     SHORT SUBSORT2
SORT_NEXT_HALF: POP     SI                  ; 废除堆栈中的首指针
SUBSORT2:       MOV     SI，POINT           ; 分隔指针 i 的地址→SI
                POP     DI                  ; 取尾指针→j (DI 作为后半部分的尾指针 j)
                CMP     SI，OFFSET BUF+COUNT - 1
                JNB     EXIT                ; 后半部分的首指针>缓冲区尾指针，转
                INC     SI                  ; i+1→i
                CMP     SI，DI
                JNB     EXIT                ; i>j，不再排序，转出口
                MOV     AL，[SI]            ; x. key→AL
                CALL    QKSORT              ; 递归调用 QKSORT 子程序，排序后半部分
EXIT:           RET                         ; 排序完成，返回主程序
QKSORT          ENDP
CODE            ENDS
                END     START
```

第六章　BIOS 中断功能调用编程实例

本章主要讨论中断功能调用的基本编程方法及实际应用问题。中断功能调用是系统提供给用户的用于进行输入/输出设备管理的子程序包，它们被固化在主机的 ROM 存储器中，其中主要包括键盘管理中断功能调用、显示器管理中断功能调用、磁盘管理中断功能调用以及打印机管理中断功能调用等子程序，它们是直接驱动计算机硬件设备的子程序，是提供给用户进行设备管理的子程序，是汇编语言程序设计中经常使用的、不可缺少的一个子程序包。

第一节　常用的 BIOS 功能调用

在 IBM PC 及其兼容机上的 BIOS 是基本的输入/输出系统，它被固化在系统板的 ROM 芯片中，长为 8K 字节。BIOS 不仅提供了系统进行自检和初始化的参数，还可以支持系统的硬件设备，提供了对硬件设备的管理程序并提供给用户调用入口，如：显示器管理程序、键盘输入管理程序、磁盘输入/输出管理程序以及打印机管理程序等。这样，软件开发人员可以不必了解 I/O 接口的硬件特性，只要直接调用 BIOS 中的相应中断服务程序，就可以实现对 I/O 设备的控制管理。

BIOS 中提供的各种 I/O 设备管理程序，都可以采用中断调用的方式来使用，这些中断使用了中断类型号 08～1FH。对于每一类型的中断服务程序都包含了若干个子功能模块，要调用某一中断不仅要把入口参数送相应的寄存器，还要把子功能号送 AH 寄存器，然后发出中断功能调用 INT n。本节仅介绍 BIOS 中常用的几种中断功能调用。

（一）键盘管理中断功能调用

键盘管理中断功能调用号为 16H，ROM BIOS 对这一中断服务程序提供了如下三个子功能：

1. 子功能 0

入口参数：AH＝0

功　　能：从键盘读取输入的字符

出口参数：AL 中为输入字符的 ASCII 码

2. 子功能 1

入口参数：AH＝1

功　　能：检测（不等待）键盘是否有按键

出口参数：若键盘有字符输入，则字符的 ASCII 码送 AL 并置 ZF＝0；否则 ZF＝1。

3. 子功能 2

入口参数：AH＝2

功　　能：读取特殊功能键的状态

出口参数：AL 中的相应位表示了特殊键的状态。见图 6－1。

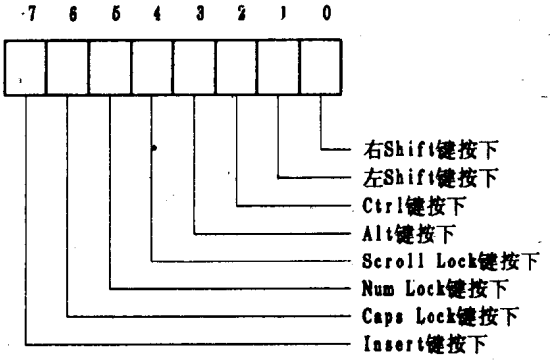


图 6－1　键盘状态字节

（二）显示器管理中断功能调用

IBM PC 显示器管理中断功能调用提供了对显示器进行各种操作的 16 个子功能。这些功能调用有：清除屏幕、设置显示器工作方式、设置光标大小和位置、实现屏幕滚动、显示字符、设置屏幕颜色等，如表 6－1 所示。

表 6－1　显示器管理中断功能调用（INT 10H）一览表

调 用 号	功　能	入口参数	出口参数
AH＝0	设置屏幕显示方式	AL＝0 40×25 黑白字符方式 AL＝1 40×25 彩色字符方式 AL＝2 80×25 黑白字符方式 AL＝3 80×25 彩色字符方式 AL＝4 320×200 彩色图形方式 AL＝5 320×200 黑白图形方式 AL＝6 640×200 黑白图形方式 AL＝7 单色板黑白字符方式	
AH＝1	设置光标类型	CH＝光标起始扫描线 CL＝光标结束扫描线	
AH＝2	设置光标位置	DH＝新光标行值 DL＝新光标列值 BX＝新光标所在的页	
AH＝3	读当前光标位置	BH＝页号	DH＝当前字符行号 DL＝当前字符列号 CX＝光标类型
AH＝4	保留		
AH＝5	选择当前显示页	AL＝页号	
AH＝6	屏幕窗口向上滚动	AL＝滚动行数 （AL＝0 全窗口滚动） CX＝滚动窗口起始行列值 DX＝滚动窗口结束行列值 BH＝空留区填充字符的属性	

（续）

调 用 号	功 能	入口参数	出口参数
AH=7	屏幕窗口向下滚动	同 AH=6 的入口参数	
AH=8	在所选择的显示页当前光标处读字符及属性	BH=显示页	AL=读出的字符 AH=读出的属性
AH=9	在所选择的显示页当前光标处写字符及属性	BH=显示页 CX=要写的字符数 AL=要写的字符 BL=要写的字符属性	
AH=0AH	在所选择的显示页当前光标处以原有属性写字符	BH=要写的显示页 CX=要写的字符数 AL=要写的字符	
AH=0BH	设置颜色（改变颜色寄存器的设置）	①若 BH=0，BL 低 5 位的值用来设置背景、前景或边框颜色 ②若 BH=1，为选择色组，BL 的第 0 位存放有色组号（0 或 1）	
AH=0CH	图形方式下在指定的坐标处写点	DX=象素点的行值 CX=象素点的列值 AL=要写的象素点值（若 AL 的最高位为 1，异或写点）	
AH=0DH	图形方式下在指定的坐标处读点	DX=象素点的行值 CX=象素点的列值	AL=指定象素点的读出值
AH=0EH	TTY 方式写字符	AL=要写的字符码 BH=页号，BL=前景颜色	
AH=0FH	取当前显示方式	无	AH=屏幕列数 AL=当前工作模式 BH=当前正在显示的页

（三）打印机管理中断功能调用

ROM BIOS 打印机管理中断调用，提供给用户如下三个子功能调用：

1. 子功能 0

入口参数：AH=0

DX=打印机号（0～2），如果只连接了一台打印机，则 DX=0

功　　能：打印字符，同时返回打印机状态

出口参数：AH=打印机状态，见图 6－2

2. 子功能 1

入口参数：AH=1

功　　能：打印机初始化，并返回打印机状态

出口参数：AH=打印机状态，见图 6－2

3. 子功能 2

入口参数：AH=2

功　　能：读打印机状态

出口参数：AH＝打印机状态，见图 6－2

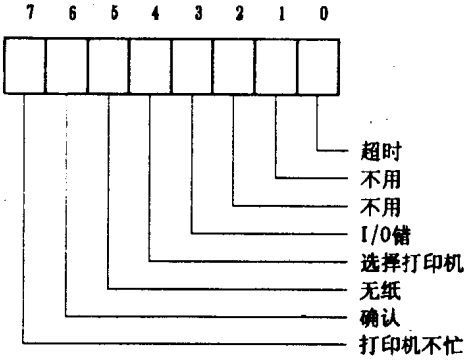


图 6－2　打印机状态字节

（四）磁盘输入/输出中断功能调用

磁盘输入/输出中断 INT 13H 提供了对磁盘进行管理的 6 个子功能，可以完成磁盘复位、格式化、读写等操作，如表 6－2 所示。

表 6－2　磁盘输入/输出中断功能调用（INT 13H）一览表

功能号	功　能	入口参数	出口参数
AH＝0	磁盘复位	无	AH＝磁盘状态（见图 6－3）
AH＝1	读取磁盘状态	无	AH＝磁盘状态（见图 6－3）
AH＝2	读磁盘扇区到内存缓冲区	DL＝驱动器号（0～3） DH＝面号（0～1） CH＝磁道号（0～39） CL＝扇区号（1～9） AL＝扇区数（0～9） ES：BX＝缓冲区首址	CF＝0 操作成功： AH＝0 AL＝读出的扇区数 CF＝1 操作失败： AH＝出错状态（见图 6－3）
AH＝3	内存缓冲区 内容写入磁盘	同读盘	同读盘
AH＝4	检验指定扇区	不要求 ES：BX，其余同读盘	同读盘
AH＝5	格式化磁盘	ES：BX＝格式化参数首址其余同读盘	同读盘

说明：在 5 号子功能中的格式化参数，是提供进行扇区格式化的扇区标识部 ID 参数，每一扇区都对应有 C、H、R 和 N 等 4 个以字节为单位的参数，它们分别表示该扇区所在的磁道号、磁头号、扇区号以及扇区长度指数。对于双面 360KB 的软盘来说，在正常情况下，C 的范围在 00～39 之间，H 的范围在 0 和 1 之间，R 的范围在 01～09 之间，N 值为 2（N＝0 表示每一扇区为 128 字节，N＝1 表示每一扇区是 256 字节，N＝2 表示每扇区是 512 字节）。例如，如果要按正常格式将第 0 面、20 磁道的 9 个扇区进行格式化，则其相应的 ID 参数与程序如下：

```
ID20   DB   20,00,01,02,20,00,02,02,20,00,03,02,20,00,04,02,20,00,05,02
       DB   20,00,06,02,20,00,07,02,20,00,08,02,20,00,09,02
```

格式化的程序片段如下：

```
MOV DX，0000H          ；对 0 磁头、A 驱动器上的磁盘进行
MOV CX，1401H          ；从 20 磁道、1 扇区开始
MOV BX，OFFSET ID20    ；ID 参数表首址→BX
MOV AX，0509H          ；格式化 9 个扇区
INT 13H                ；中断功能调用
```

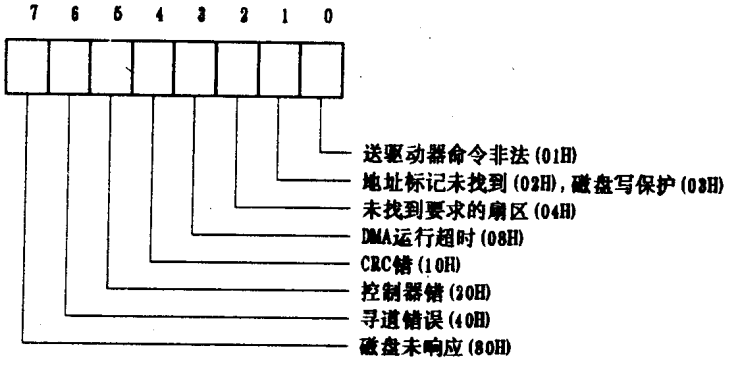


图 6-3　磁盘输入/输出状态字节

程序如下：

```
ID20   DB   20,00,01,02,20,00,02,02,20,00,03,02,20,00,04,02,20,00,05,02
       DB   20,00,06,02,20,00,07,02,20,00,08,02,20,00,09,02
```

格式化的程序片段如下：

```
MOV     DX，0000H          ；对 0 磁头、A 驱动器上的磁盘进行
MOV     CX，1401H          ；从 20 磁道、1 扇区开始
MOV     BX，OFFSET ID20    ；ID 参数表首址→BX
MOV     AX，0509H          ；格式化 9 个扇区
INT     13H                ；中断功能调用
```

第二节　BIOS 功能调用编程实例

下面就常见的几类中断功能调用的使用方法例举一些编程实例，以便说明这些中断功能调用的基本用法。

【例 6.1】 简单的输入、显示和打印字符串程序

利用 BIOS 中相应的功能调用编制一个程序，完成从键盘接收任意输入的字符串，以回车键（0DH）为结束。在输入过程中边输入、边显示和打印。

程序清单

```
SSEG        SEGMENT    STACK ' STACK '
```

```
        DB          256 DUP (0)
SSEG    ENDS
CSEG    SEGMENT     PUBLIC 'CODE'
        ASSUME      CS: CSEG
MAIN    PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
RETRY:  MOV     AH, 0
        INT     16H             ; 接收键盘输入
        MOV     AH, 0EH
        MOV     BX, 1
        INT     10H             ; 显示字符
        PUSH    AX
        MOV     AH, 0
        MOV     DX, 0
        INT     17H             ; 打印字符
        POP     AX
        CMP     AL, 0DH
        JNZ     RETRY           ; 不是回车键，转 RETRY
        RET                     ; 是回车键，返回 DOS
MAIN    ENDP
CSEG    ENDS
        END     MAIN
```

【例 6.2】 硬盘主引导记录的保存与恢复

在微机硬盘上的 0 面、0 磁道、第 1 扇区驻留有硬盘主引导记录，它的功能是引导和启动操作系统。所以当主引导记录损坏后，硬盘便不能启动。为了使硬盘主引导记录在损坏后被能正确地恢复，可用以下方法对硬盘主引导记录进行保护。

编程指导：

由于硬盘主引导记录是一个独立的扇区，不属于任何一个分区。所以用 DEBUG 命令是读不到主引导记录的。为此需要编写专门的程序对其进行保护和恢复工作。下例的程序 1 完成的功能是：首先将硬盘 0 面、0 磁道 1 扇区的硬盘主引导记录读到内存中的 BUFFER 缓冲区，然后将缓冲区中的内容写到 A 驱动器中软盘的 0 面、20H 道、1 扇区。从而把硬盘的主引导记录保存到软盘上。程序 2 完成的功能是：将 A 驱动器中软盘上 0 面、20H 道、1 扇区中事先保存的硬盘主引导记录读入内存缓冲区中，然后再将其写入硬盘的 0 面、0 磁道、1 扇区，从而完成硬盘主引导记录的修复工作。

程序清单

(1) 保存主引导记录

```
STACK     SEGMENT   PARA STACK 'STACK'
          DB        128 DUP (0)
STACK     ENDS
DATA      SEGMENT
BUFFER    DB        512 DUP (0)
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA, ES: DATA
BEGIN:    MOV       AX, DATA
          MOV       DS, AX
          MOV       ES, AX
          MOV       AX, 0201H
          MOV       BX, OFFSET BUFFER
          MOV       CX, 0001H
          MOV       DX, 80H
          INT       13H
          MOV       AX, 0301H
          MOV       BX, OFFSET BUFFER
          MOV       CX, 2001H
          MOV       DX, 0
          INT       13H
          MOV       AX, 4C00H
          INT       21H
CODE      ENDS
          END       BEGIN
```

(2) 恢复主引导记录

```
          MOV       AX, 0201H
          MOV       BX, OFFSET BUFFER
          MOV       CX, 2001H
          MOV       DX, 0
          INT       13H
          MOV       AX, 0301H
          MOV       BX, OFFSET BUFFER
          MOV       CX, 0001H
          MOV       DX, 80H
          INT       13H
```

【例 6.3】 单色显示器各种显示属性的验证

下面的程序采用直接写显示缓冲区的方法，对单色显示器文本方式下字符的各种属性进行验证。程序运行时，在屏幕的中部以下以 8 个不同的属性显示字符“A”，相邻的两个字符之间有一个空行。显示完成后，只要敲任意键程序就返回 DOS。

编程指导：

在 IBM PC 系统中，字符方式下的文本显示可以分为 80×25 和 40×25 两种，每个显示的字符在显示缓冲区中都占有两个字节，其中一个字节存放字符的 ASCII 码，另一个字节存放字符的显示属性。以字符“A”为例，其字符字节与属性字节如图 6－4 所示。其中属性字节中 RGB 的允许值如表 6－3 所示。

对于单色显示卡来说，从内存 B000H 段开始的 4KB 空间用于显示器屏幕缓冲区信息的存储，屏幕缓冲区的段地址及其信息存放格式如图 6－5 所示。

表 6－3　RGB 允许值及其含义

背景值	前景值	显示属性
000	000	不显示
111	111	显示白方块
000	111	正常显示（黑底白字）
000	001	正常显示，有下划线
111	000	反常显示（白底黑字）

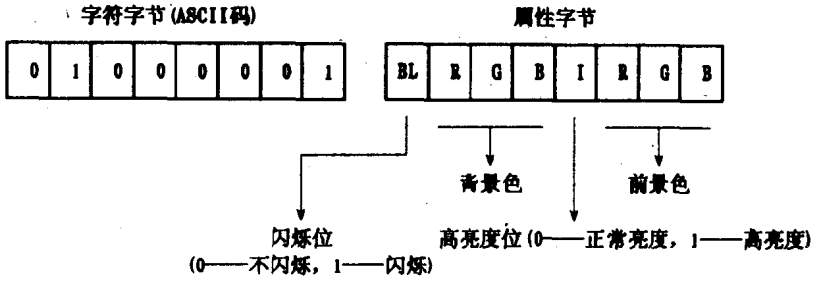


图 6－4　单色显示器的字符及其属性

程序清单

```
STACK    SEGMENT   PARA STACK 'STACK'
         DB        256 DUP (0)
STACK    ENDS
DATA     SEGMENT   PARA PUBLIC 'DATA'
ATTRIB   DB 07H                    ; 正常属性（黑底白字）
         DB 70H                    ; 反常显示（白底黑字）
         DB 87H                    ; 黑底白字并闪烁
         DB 01H                    ; 下划线
         DB 0FH                    ; 黑底白字高亮度
         DB 08FH                   ; 高亮度/闪烁
         DB 0F0H                   ; 反常显示/闪烁
         DB 0F8H                   ; 反常显示/高亮度/闪烁
DATA     ENDS
```

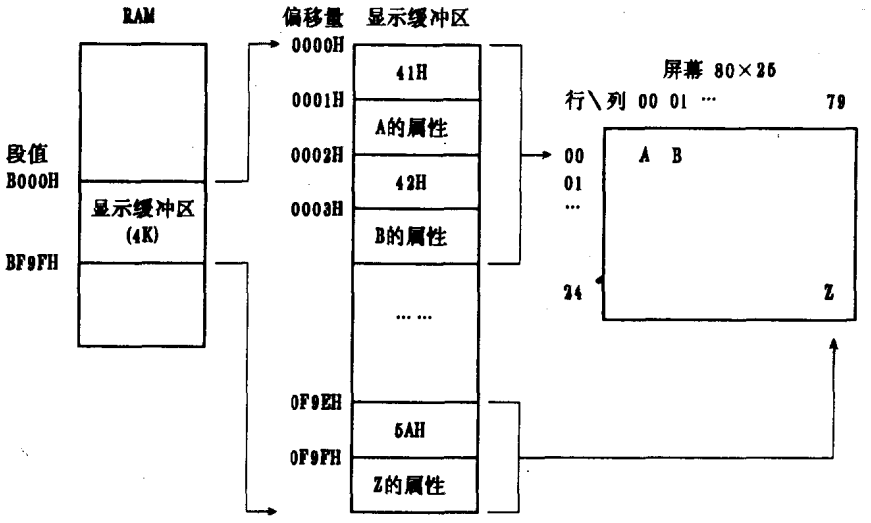


图 6-5 单色显示器屏幕缓冲区的结构

```
CODE    SEGMENT  PARA PUBLIC ' CODE '
START   PROC     FAR
        ASSUME   CS: CODE, DS: DATA
        PUSH     DS
        XOR      AX, AX
        PUSH     AX
        MOV      AX, DATA
        MOV      DS, AX
; 清除屏幕
        MOV      AX, 0B000H
        MOV      ES, AX
        MOV      DI, 0
        MOV      AL, 20H          ; 空格符→AL
        MOV      AH, 07H          ; 正常属性
        MOV      CX, 2000
        CLD
        REP      STOSW            ; 清屏
; 在屏幕的下半部以 8 个不同的属性写 "A"
        MOV      CX, 8            ; 8 个属性字节的循环计数
        MOV      DI, 240          ; 显示缓冲区的偏移地址初值
        MOV      BX, OFFSET ATTRIB ; 属性字节首地址→BX
        MOV      AL, ' A '
DISP_A: MOV      AH, [BX]
        MOV      ES: [DI], AX
```

```
            ADD     DI, 2 * (80 * 2)        ; 指向下一显示地址
            INC     BX                      ; 改变属性字节地址
            LOOP    DISP _ A
;
            MOV     AH, 0
            INT     16H                     ; 等待键盘输入
;
            MOV     DI, 0
            MOV     AL, 20H
            MOV     AH, 07H
            MOV     CX, 2000
            REP     STOSW                   ; 清屏
            RET                             ; 返回 DOS
START       ENDP
CODE        ENDS
            END     START
```

【例 6.4】彩色文本方式的字符属性验证

下面的程序用于验证彩色文本方式下字符的属性。程序运行时首先设置显示器工作方式为：80×25 彩色字符方式，然后选择边缘颜色为绿色，接着是设定光标在第 12 行、38 列，以此为起始坐标，以不同的属性显示字符串“Good!”，最后返回 DOS。

在彩色文本方式下，屏幕缓冲区的段地址从 B800H 开始，有 16KB 的内存空间，字符属性字节各位的含义同图 6－5。属性字与颜色对照表见表 6－4。

表 6－4　彩色属性字与颜色对照表

I R G B	颜　色	I R G B	颜　色
0 0 0 0	黑	1 0 0 0	暗灰
0 0 0 1	蓝	1 0 0 1	亮蓝
0 0 1 0	绿	1 0 1 0	亮绿
0 0 1 1	青（深蓝）	1 0 1 1	亮青
0 1 0 0	红	1 1 0 0	亮红
0 1 0 1	品红	1 1 0 1	亮品红
0 1 1 0	棕	1 1 1 0	黄
0 1 1 1	亮灰	1 1 1 1	白

程序清单

```
STACK       SEGMENT     PARA STACK ' STACK '
            DB          256 DUP (0)
STACK       ENDS
DATA        SEGMENT
STRING          DB                          ' Good! '
```

```
CHAR _ COUNT    EQU                     $ - STRING
CHAR _ ATRIB    DB   10H                ; 蓝底色、黑字符
                DB   25H                ; 绿底色、品红字符
                DB   39H                ; 青底色、亮蓝色字符
                DB   0A6H               ; 绿底色、棕色闪烁字符
                DB   0FEH               ; 亮灰底色、黄色闪烁字符
DATA            ENDS
CODE            SEGMENT
                ASSUME                  CS: CODE, DS: DATA
START       PROC    FAR
            MOV     AX, DATA
            MOV     DS, AX
            MOV     AX, 0003H
            INT     10H                 ; 设置显示方式为: 80×25 彩色字符
;
            MOV     AH, 0BH
            MOV     BH, 0
            MOV     BL, 2
            INT     10H                 ; 选择边缘颜色为绿色
;
            MOV     SI, OFFSET STRING
            MOV     DI, OFFSET CHAR _ ATRIB
            MOV     CX, CHAR _ COUNT
            MOV     DX, 0C26H           ; 光标起始位置 (12, 38)
RETRY:      PUSH    CX
            MOV     AH, 2
            MOV     BH, 0
            INT     10H                 ; 设置光标
            MOV     AH, 9
            MOV     BH, 0               ; 选择 0 页
            MOV     AL, [SI]            ; 当前显示的字符→AL
            MOV     BL, [DI]            ; 当前字符的属性值→BL
            MOV     CX, 1               ; 显示一个字符
            INT     10H
            POP     CX
            INC     SI                  ; 修改字符地址
            INC     DI                  ; 修改属性地址
            INC     DL                  ; 修改显示坐标
            LOOP    RETRY
```

```
        MOV     AH，4CH
        INT     21H
START   ENDP
CODE    ENDS
        END     START
```

【例 6.5】 彩色图形方式的色彩验证

下面的程序用于说明如何在彩色图形方式下设置图形颜色。程序在运行时，先将屏幕设置为 320×200 彩色图形方式，然后设置屏幕底色为绿色并选择 1 号调色板，最后以青色显示 A 字符的 8×8 点阵图形；以品红色显示 B 字符的 8×8 点阵图形；以白色显示 C 字符的 8×8 点阵图形。为了使程序返回系统时保持原有的显示方式，在程序的开始部分首先取系统当前的显示方式并进栈保护，在返回 DOS 之前用重新设置显示方式的方法进行恢复。

编程指导：

在 320×200 彩色图形方式下，一个字节能表示屏幕上的 4 个象素点，象素点的颜色可以从两种彩色组中的任意一组中选取，但同一时刻只能选择一个彩色组中的颜色。在设定的彩色组中，一个象素点可有 4 种颜色供选用。象素字节及其彩色组的颜色定义见表 6-5 和表 6-6。

表 6-5 图形信息字节

7	6	5	4	3	2	1	0
C1	C0	C1	C0	C1	C0	C1	C0
第 1 点色值		第 2 点色值		第 3 点色值		第 4 点色值	

表 6-6 色值与颜色对照表

C1	C0	第 1 彩色组	第 2 彩色组
0	0	同背景色	同背景色
0	1	绿色	青色
1	0	红色	品红
1	1	棕色	白色

程序清单

```
STACK     SEGMENT   PARA STACK 'STACK'
          DB        256 DUP (0)
STACK     ENDS
DATA      SEGMENT
LINE      DW  0     ；点的行坐标
CORLUM    DW  0     ；点的列坐标
COUNT1    DB  0
COUNT2    DB  0
COUNT3    DB  0
ATRIBUTE  DB  0
;;;;; A 字符的 8×8 点阵字模;;;;;
A_8X8     DB  00110000B    ; 30H
          DB  01111000B    ; 78H
```

```
            DB    11001100B     ; 0CCH
            DB    11001100B     ; 0CCH
            DB    11111100B     ; 0FCH
            DB    11001100B     ; 0CCH
            DB    11001100B     ; 0CCH
            DB    00000000B     ; 00H
;;;;; B 字符的 8×8 点阵字模;;;;;
B_8X8       DB 11111100B        ; 0FCH
            DB    01100110B     ; 66H
            DB    01100110B     ; 66H
            DB    01111100B     ; 7CH
            DB    01100110B     ; 66H
            DB    01100110B     ; 66H
            DB    11111100B     ; 0FCH
            DB    00000000B     ; 00H
;;;;; C 字符的 8×8 点阵字模;;;;;
C_8X8       DB    00111100B     ; 3CH
            DB    01100110B     ; 66H
            DB    11000000B     ; 0C0H
            DB    11000000B     ; 0C0H
            DB    11000000B     ; 0C0H
            DB    01100110B     ; 66H
            DB    00111100B     ; 3CH
            DB    00000000B     ; 00H
DATA        ENDS
CODE        SEGMENT
            ASSUME    CS: CODE, DS: DATA
START       PROC      FAR
            MOV       AX, DATA
            MOV       DS, AX
            MOV       AH, 0FH
            INT       10H
            PUSH      AX
            MOV       AX, 0004H
            INT       10H                ; 设置显示方式为:
                                             320×200 彩色图形方式
;
            MOV       AH, 0BH
            MOV       BH, 0
```

```
        MOV     BL, 2
        INT     10H                     ; 选择底色为绿色
;
        MOV     AH, 0BH
        MOV     BH, 1
        MOV     BL, 1                   ; 选择 1 号调色板
        INT     10H
;
        MOV     SI, OFFSET A_8X8        ; 显示 A 字符的 8×8 点阵
        MOV     ATRIBUTE, 1             ; 选择 1 号颜色 (青色)
        MOV     LINE, 80
        MOV     CORLUM, 50
        CALL    DISP_8X8DOT
;
        MOV     SI, OFFSET B_8X8
        MOV     ATRIBUTE, 2             ; 选择 2 号颜色 (品红色)
        MOV     LINE, 80
        MOV     CORLUM, 60
        CALL    DISP_8X8DOT
;
        MOV     SI, OFFSET C_8X8
        MOV     ATRIBUTE, 3             ; 选择 3 号颜色 (白色)
        MOV     LINE, 80
        MOV     CORLUM, 70
        CALL    DISP_8X8DOT
;
        MOV     AH, 7
        INT     21H                     ; 等待键盘按键.
;
        POP     AX                      ; 取显示方式
        MOV     AH, 0
        INT     10H                     ; 恢复原有显示方式
        MOV     AH, 4CH
        INT     21H                     ; 返回 DOS
START   ENDP
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 8×8 点阵显示子程序: DISP_8X8DOT
; 入口参数: SI=点阵字模首地址
; 出口参数: 无
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DISP_8X8DOT  PROC    NEAR
             MOV     AH, 0CH
             MOV     AL, ATRIBUTE
             MOV     DX, LINE
             MOV     CX, CORLUM
             MOV     COUNT1, 8              ; 一个字符显示8个扫描行
RETRY1:      MOV     COUNT2, 1              ; 一个扫描行显示一字节字模
RETRY2:      MOV     COUNT3, 8              ; 一字节字模显示8个点
RETRY3:      SHL     BYTE PTR [SI], 1
             JNC     NEXT                   ; 当前位为0，不显示，转NEXT
             PUSH    AX
             PUSH    CX
             INT     10H
             POP     CX
             POP     AX
NEXT:        INC     CX
             DEC     COUNT3
             JNZ     RETRY3                 ; 一字节的8个点未显示完，转
             INC     SI                     ; 取下一字节字模
             DEC     COUNT2
             JNZ     RETRY2                 ; 一个扫描行未显示完，转
             INC     DX                     ; 行坐标加1
             MOV     CX, CORLUM             ; 列坐标置初值
             DEC     COUNT1
             JNZ     RETRY1                 ; 8个扫描行未显示完，转
             RET
DISP_8X8DOT  ENDP
CODE         ENDS
             END     START
```

第三节　绘图程序编程实例

绘图是计算机程序设计中经常碰到的一个问题。在程序设计中最基本的绘图工作包括点、直线、斜线和圆，本节仅举两例简单的绘图程序以供读者参考。

【例 6.6】 利用键盘绘图

这个程序能运行在装有 VGA 显示卡的 286 和 386 微机上，它将屏幕置成 640×480 的彩色图形方式，并把光标的初始位置设置在屏幕的中心。程序可以接收上移键“↑”、下移键“↓”、右移键“→”、左移键“←”然后改变当前点的位置，根据当前的彩色值进行画点。在

移动过程中可以按数字键“0～9”改变点的颜色；若按“ESC”键则退出程序返回 DOS。

编程指导：

表 6－1 列出了 IBM PC 机 INT 10H 中断的有关功能调用，这些功能调用仅适用于 CGA 和单色显示卡，对于当前流行的各种 286、386 微机来说，一般都配有 EGA 卡和 VGA 卡，相应的 INT 10H 中断功能也大为增强。单就显示方式来讲，配有 EGA 显示卡的 INT 10H 的显示方式不仅包括了 CGA 卡所具有的显示方式外，还增加了方式 0DH、0EH、0FH、10H。而对于 VGA 显示卡的显示方式，则不仅包括了 CGA 和 EGA 的所有显示方式外，又增加了方式 11H 和方式 12H，如表 6－7 所示。

表 6－7　EGA/VGA 增强的图象方式

方　式	显 示 卡	分 辨 率	颜　色
0DH	EGA/VGA	320×200	16
0EH	EGA/VGA	640×200	16
0FH	EGA/VGA	640×350	单色
10H	EGA/VGA	640×350	16
11H	VGA	640×480	2
12H	VGA	640×480	16

由于增强的图象方式提高了显示器的分辨率，为开发完善的汉字操作系统提供了保证，目前不少汉字操作系统都充分利用了 EGA/VGA 卡的彩色高分辨率图象方式，使汉字操作系统的功能大为增强。本例就是利用 VGA 的方式 12H 进行简单绘图的程序。

程序清单

```
CODE        SEGMENT
            ASSUME    CS: CODE
START       PROC      FAR
            MOV       AX, 0012H
            INT       10H              ; 设置 640×480 彩色图形方式
;
            MOV       AH, 0BH
            MOV       BX, 0001H
            INT       10H              ; 选择背景为蓝色
;
            MOV       CX, 320
            MOV       DX, 240          ; 设置光标起始位置为 (240, 320)
get_char:   PUSH      BX
            MOV       AH, 0
            INT       16H              ; 读键盘
```

```
                POP     BX
                CMP     AL，1BH
                JZ      EXIT                ；是“ESC”键，转EXIT返回DOS
                CMP     AL，3AH
                JGE     plot _ dot
                CMP     AL，30H
                JL      plot _ dot
                SUB     AL，30H             ；数字键减30H转换为数值作为颜色值
                MOV     BL，AL              ；保存颜色值
                JMP     SHORT get _ char
plot _ dot：    MOV     AL，AH
                CMP     AL，48H
                JNZ     not _ up _ KEY
                DEC     DX                  ；光标上移一个坐标单位
                JMP     SHORT draw _ dot
not _ up _ KEY：CMP     AL，50H
                JNZ     not _ down _ KEY
                INC     DX                  ；光标下移一个坐标单位
                JMP     SHORT draw _ dot
not _ down _ KEY：CMP   AL，4DH
                JNZ     not _ right _ KEY
                INC     CX                  ；光标右移一个坐标单位
                JMP     draw _ dot
not _ right _ KEY：CMP  AL，4BH
                JNZ     get _ char
                DEC     CX                  ；光标左移一个坐标单位
draw _ dot：    MOV     AL，BL
                MOV     AH，0CH
                PUSH    BX
                INT     10H                 ；画一点
                POP     BX
                JMP     SHORT get _ char
exit：          MOV     AH，4CH
                INT     21H
START           ENDP
CODE            ENDS
                END     START
```

【例 6.7】 画线程序

下面的程序采用了 Bresenham 算法能在屏幕上任意两点之间画一直线。

编程指导：

Bresenham 是一种快速而有效的绘制斜线的方法。它的基本思想是：建立一个误差项(error term)，其初值为 0，并假设 X (x1，y1) 和 Y (x1，y2) 分别为所要绘制的斜线的起点和终点，且 x2>x1，y2>y1。将 x2－x1 的差值表示为 delta _ x，将 y2－y1 的差值表示为 delta _ y；并且将 (x2－x1) /2 的结果取整用 half _ x 表示，将 (y2－y1) /2 的结果取整用 half _ y 表示，然后采用图 6－6 所示的方法绘制斜线。

上面的算法是一种简单的情况，实际问题中，x1 可能大于 x2，y1 也可能大于 y2，在这种情况下，每画一个点应使行坐标或列坐标减 1。所以程序设计中，首先判断 x2－x1 的值的正负，并相应地将 DI 寄存器置成＋1 或－1。同样 SI 寄存器的值根据 y2－y1 值的正负置成＋1 或－1。在画完一个点以后，如果需要变换点的坐标，则将行坐标加上 DI，而将列坐标加上 SI。

另外，前面的讨论都是针对于斜率小于 1 的情况，在这种情况下，列坐标的变化率小于行坐标的变化率。每画一个点，行坐标必加 1，而列坐标则有时加 1、有时不加 1。如果斜率大于 1，则列坐标的变化率大于行坐标的变化率，此时每画一个点，列坐标必加 1，而行坐标则有时加 1、有时不加 1。为此，程序设计中根据斜率的不同，分别执行 EASY（斜率小于 1）和 STEEP（斜率大于 1）两个过程。

程序清单

```
DATA        SEGMENT
delta _ x   DW   ?              ; 存放 x2－x1 的差值
delta _ y   DW   ?              ; 存放 y2－y1 的差值
half _ y    LABEL WORD
half _ x    DW   ?              ; 存放 (x2－x1) /2 或 (y2－y1) /2
COUNT       DW   ?              ; 循环计数
x1          DW   10             ; 起始点的行坐标
y1          DW   10             ; 起始点的列坐标
x2          DW   320            ; 终止点的行坐标
y2          DW   200            ; 终止点的列坐标
COLOR       DB   07H            ; 颜色
DATA        ENDS
CODE        SEGMENT
MAIN        PROC                FAR
            ASSUME              CS: CODE, DS: DATA
            MOV    AX, DATA
            MOV    DS, AX
            MOV    AX, 0600H
            MOV    BH, 07
            MOV    CX, 0
            MOV    DX, 184FH
```

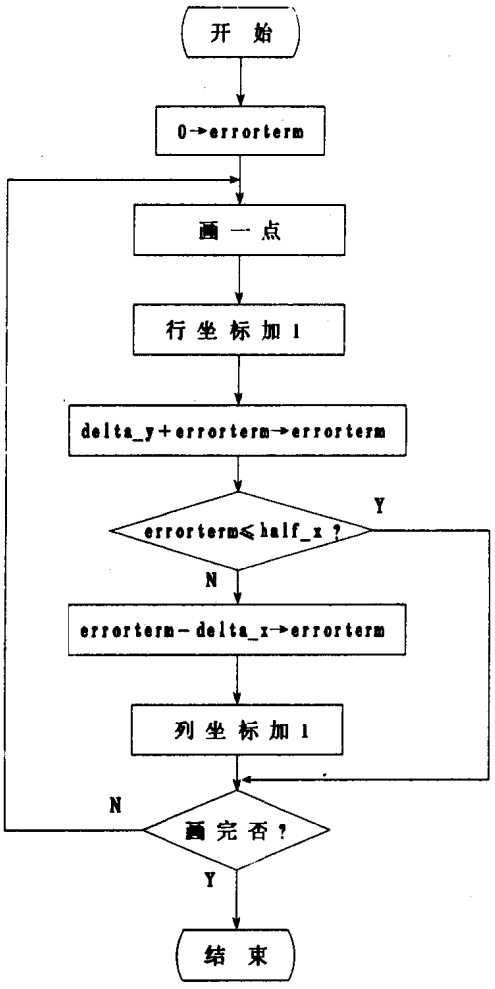


图 6-6　Bresenham 算法流程图

```
        INT     10H             ; 清屏
        MOV     AX, 0006H
        INT     10H             ; 设置显示器为图象工作方式 6
;
        MOV     AX, y2
        SUB     AX, y1
        MOV     SI, 1           ; y2>y1, 置列坐标为增量
        JGE     STORE_Y
        MOV     SI, -1          ; y2<y1, 置列坐标为减量
        NEG     AX              ; AX 中为 y2-y1 的绝对值
STORE_Y: MOV    delta_y, AX
        MOV     AX, x2
        SUB     AX, x1
        MOV     DI, 1           ; x2>x1, 置行坐标为增量
        JGE     STORE_X
```

```
                MOV     DI, -1          ; x2<x1，置行坐标为增量
                NEG     AX              ; AX 中为 x2-x1 的绝对值
STORE_X:        MOV     delta_x, AX
                MOV     AX, delta_x
                CMP     AX, delta_y
                JL      CSTEEP          ; 斜率大于 1，转
                CALL    EASY            ; 斜率小于 1，执行 EASY 过程
                JMP     FINISH
CSTEEP:         CALL    STEEP           ; 斜率大于 1，执行 STEEP 过程
FINISH:         MOV     AH, 4CH
                INT     21H             ; 返回 DOS
MAIN            ENDP
EASY            PROC    NEAR
                MOV     AX, delta_x
                SHR     AX, 1           ; 求 (x2-x1) /2→half_x
                MOV     half_x, AX
                MOV     CX, x1
                MOV     DX, y1
                MOV     BX, 0           ; 误差项 BX 置初值 0
                MOV     AX, delta_x
                MOV     COUNT, AX       ; 以 x2-x1 的绝对值为循环计数
NEWDOT:         CALL    DOTPLOT         ; 调用画点子程序
                ADD     CX, DI          ; 行坐标加 1
                ADD     BX, delta_y
                CMP     BX, half_x
                JLE     DCOUNT
                SUB     BX, delta_x
                ADD     DX, SI
DCOUNT:         DEC     COUNT
                JGE     NEWDOT
                RET
EASY            ENDP
STEEP           PROC    NEAR
                MOV     AX, delta_y
                SHR     AX, 1
                MOV     half_y, AX
                MOV     CX, x1
                MOV     DX, y1
                MOV     BX, 0
```

```
                MOV     AX, delta _ y
                MOV     COUNT, AX
NEWDOT2:        CALL    DOTPLOT
                ADD     DX, SI
                ADD     BX, half _ x
                CMP     BX, half _ y
                JLE     DCOUNT2
                SUB     BX, delta _ y
                ADD     CX, DI
DCOUNT2:        DEC     COUNT
                JGE     NEWDOT2
                RET
STEEP           ENDP
DOTPLOT         PROC    NEAR
                PUSH    AX
                MOV     AL, COLOR
                MOV     AH, 0CH
                INT     10H                 ; 画一点
                POP     AX
                RET
DOTPLOT         ENDP
CODE            ENDS
                END     MAIN
```

第四节　动画显示编程实例

动画显示是将显示在屏幕上的图象产生运动效果的一种技术。它不仅要求程序能完成绘图的功能，而且要求程序能完成将描绘出来的图象产生运动的效果。动画显示技术被广泛地应用于计算机游戏程序、电视字幕系统以及高级绘图系统的程序设计中，在这些程序设计中包含了许多程序设计的技巧。

【例 6.8】 字幕的显示与移动

在电视字幕系统中，字幕的制作是用计算机来完成的，当一幅字幕制作好以后，一边从计算机显示器显示出来，一边通过接口板送往字幕与图象的合成系统。下面给出一个简单的字幕显示与移动的程序实例。程序把显示器设置为 320×200 图象方式，以打字的形式在屏幕的中央显示 24×24 点阵的宋体字“欢迎”；然后等待用户按任意键，接着由下向上移动字幕直至消失。

编程指导：

在 300×200 图形方式下，屏幕缓冲区的一个字节对应于屏幕上的 4 个象素点，所以 80 个字节对应于屏幕上一个扫描行的 320 个点，200 个扫描行共需要 200×80＝16000 个字节，即

屏幕缓冲区的16KB空间只能存放一屏点的信息。在这种显示方式下，系统把屏幕缓冲区的16KB空间分为偶扫描区和奇扫描区，每个区分别为8KB空间。偶扫描区的段地址为B800H，奇扫描区的段地址为BA00H。屏幕上所有偶扫描行点的信息按顺序从偶扫描区的偏移0地址开始存放；所有奇扫描行点的信息也按顺序从奇扫描区的偏移0地址开始存放。

由于屏幕缓冲区的这种特殊格式，如果要直接对其进行操作达到字幕的移动，就必须按照上述的格式，同时对偶扫描区和奇扫描区进行移动，下面的程序就采用这种方法进行字幕的上移操作。

程序清单

```
STACK       SEGMENT   PARA STACK 'STACK'
            DB        256 DUP (0)
STACK       ENDS
DATA        SEGMENT   PARA PUBLIC 'DATA'
ATTRIB  DB  02H                          ; 属性字节（选2号颜色）
; 下面是“欢”的24×24点阵字模（共3×24字节）
ZM1     DB  000H,008H,000H,000H,00EH,000H,000H,00CH,000H,000H,00CH,000H
        DB  001H,08CH,000H,07FH,0D8H,00CH,001H,09FH,0FEH,001H,090H,00CH
        DB  001H,0A2H,018H,021H,0A3H,090H,013H,043H,000H,01BH,003H,000H
        DB  00BH,003H,000H,006H,003H,000H,006H,006H,080H,007H,006H,080H
        DB  00DH,086H,080H,019H,08CH,040H,010H,0CCH,060H,020H,098H,030H
        DB  040H,030H,038H,000H,060H,01EH,001H,080H,008H,002H,000H,000H
; 下面是“迎”的24×24点阵字模
ZM2     DB  000H,008H,000H,030H,008H,000H,018H,05EH,018H,00CH,063H,0FCH
        DB  00CH,063H,018H,004H,063H,018H,000H,063H,018H,000H,063H,018H
        DB  00CH,063H,018H,07EH,063H,018H,00CH,063H,018H,00CH,063H,018H
        DB  00CH,067H,018H,00CH,07BH,018H,00CH,073H,018H,00CH,0E3H,078H
        DB  00CH,043H,030H,00CH,003H,000H,00CH,003H,000H,00EH,002H,000H
        DB  019H,000H,000H,070H,0FFH,0FEH,020H,07FH,0F8H,000H,000H,000H
DATA        ENDS
CODE        SEGMENT
START       PROC      FAR
            ASSUME    CS: CODE, DS: DATA
            PUSH  DS
            XOR   AX, AX
            PUSH  AX
            MOV   AX, DATA
            MOV   DS, AX
            MOV   AH, 0FH
            INT   10H
```

```
        PUSH    AX                  ; 保存系统原有的显示方式
;
        MOV     AX, 0004H
        INT     10H                 ; 设置显示器为 320×200 图形方式
;
        MOV     SI, OFFSET ZM1      ; 字模首地址→SI
        MOV     DX, 60              ; 从 (60, 80) 坐标开始显示
        MOV     CX, 80
        MOV     DI, 2               ; 显示两个字
RETRY1: MOV     BP, 24              ; 一个字显示 24 个扫描行
        PUSH    DX
RETRY2: MOV     BL, 3               ; 一个扫描行显示 3 字节点阵字模
        CALL    DISP_8BIT           ; 显示一个扫描行
        INC     DX                  ; 行号加 1
        DEC     BP
        JNZ     RETRY2              ; 24 个扫描行未显示完，转
        POP     DX                  ; 恢复初始行号
        ADD     CX, 120             ; 列坐标加 120
        DEC     DI                  ; 修改字计数单元
        JNZ     RETRY1              ; 转 RETRY1，显示下一个字
;
        MOV     AH, 0
        INT     16H                 ; 等待按键
;
        MOV     CX, (60+24) /2      ; 屏幕 (偶扫描与奇扫描) 上移 42 个扫描行
        CLD
SCORLL: PUSH    CX
        MOV     DI, 0
        MOV     SI, 80
        MOV     BL, 100-1           ; 偶/奇扫描区的 99 个扫描行上移一次
REMOV:  PUSH    SI
        PUSH    DI
        MOV     AX, 0B800H
        MOV     DS, AX
        MOV     ES, AX
        MOV     CX, 80
        REP     MOVSB               ; 偶扫描区上移一个扫描行
        POP     DI
        POP     SI
```

```
                MOV     AX, 0BA00H
                MOV     DS, AX
                MOV     ES, AX
                MOV     CX, 80
                REP     MOVSB               ; 奇扫描区上移一个扫描行
                DEC     BL
                JNZ     REMOV               ; 99 个扫描行未移完，转
                POP     CX
                LOOP    SCORLL              ; 整个屏幕未移完 42 次，转
;
EXIT:           POP     AX
                MOV     AH, 0
                INT     10H                 ; 恢复原有显示方式
                RET                         ; 返回 DOS
START           ENDP
;
DISP_8BIT       PROC    NEAR
                PUSH    CX
RETRY3:         MOV     BH, 8               ; 1 字节有 8 个点
RETRY4:         SHL     BYTE PTR [SI], 1
                JNC     NEXT1               ; 当前点为 0，不显示
                MOV     AL, ATTRIB
                MOV     AH, 0CH
                INT     10H                 ; 显示一个点
NEXT1:          INC     CX
                DEC     BH
                JNZ     RETRY4
                INC     SI
                DEC     BL                  ; 字节计数减 1
                JNZ     RETRY3              ; 3 个字节未显示完，转
                POP     CX
                RET
DISP_8BIT       ENDP
CODE            ENDS
                END     START
```

【例 6.9】 缓慢行驶的小汽车

下面是一个动画显示程序。程序运行时，首先用黄色在屏幕的中部画出一条道路，然后在道路的左边出现一个红色的小汽车，缓缓地从左向右行驶。如果在运行过程中按了任意键，

则程序将中止运行返回 DOS；否则小汽车将一直行驶直到屏幕的右边边缘为止。

编程指导：

程序首先调用 10H 的画点功能，在屏幕左中部画出一个小汽车，然后再调 10H 相应的功能移动图象，使小汽车由左至右移动。由于移动过程采用了 10H 中断功能调用，所以执行速度比较慢。

程序清单

```
STACK       SEGMENT    PARA STACK 'STACK'
            DB         256 DUP (0)
STACK       ENDS
DATA        SEGMENT
LINE        DW   80                ; 小汽车图形起始点的行坐标
CORLUM      DW   2                 ; 小汽车图形起始点的列坐标
COUNT1      DW   0
COUNT2      DW   0
COUNT3      DW   0
; * * * * * * * * 小汽车的 16×8 点阵字模 * * * * * * * *
CAR_8X16DOT  DB   00000011B, 11100000B
             DB   00000010B, 01010000B
             DB   00000110B, 01011000B
             DB   01111111B, 11111110B
             DB   11111111B, 11111111B
             DB   11111011B, 11101111B
             DB   00001010B, 00101000B
             DB   00000100B, 00010000B
DATA        ENDS
CODE        SEGMENT
            ASSUME    CS: CODE, DS: DATA
START       PROC   FAR
            MOV    AX, DATA
            MOV    DS, AX
            MOV    AH, 0FH
            INT    10H
            PUSH   AX                ; 保存系统当前显示方式
            MOV    AX, 0004H
            INT    10H               ; 设置显示方式为：320×200 彩色图形方式
;
            MOV    AH, 0BH
            MOV    BH, 1
```

```
        MOV     BL, 0               ; 选择 0 号调色板
        INT     10H
;
        MOV     AH, 0CH
        MOV     DX, LINE
        ADD     DX, 8
        MOV     CX, 0
        MOV     AL, 3               ; 选 3 号（黄）颜色
REPET:  PUSH    AX
        INT     10H                 ; 在屏幕的中部画一条黄色的道路
        POP     AX
        INC     CX
        CMP     CX, 319
        JNZ     REPET
; 下面是画一个小汽车
        MOV     AH, 0CH
        MOV     DX, LINE
        MOV     CX, CORLUM
        MOV     SI, OFFSET CAR _ 8X16DOT
        MOV     COUNT1, 8           ; 显示 8 个扫描行
RETRY1: MOV     COUNT2, 2           ; 一个扫描行显示两字节字模
RETRY2: MOV     COUNT3, 8           ; 一字节字模显示 8 个点
RETRY3: ROL     BYTE PTR [SI], 1
        MOV     AL, 0               ; 预置黑颜色
        JNC     NEXT                ; 当前位为 0，显示黑色，转 NEXT
        MOV     AL, 2               ; 置 2 号（红）颜色
NEXT:   PUSH    AX
        PUSH    CX
        INT     10H
        POP     CX
        POP     AX
        INC     CX
        DEC     COUNT3
        JNZ     RETRY3              ; 一字节的 8 个点未显示完，转
        INC     SI                  ; 取下一字节字模
        DEC     COUNT2
        JNZ     RETRY2              ; 一个扫描行未显示完，转
        INC     DX                  ; 行坐标加 1
        MOV     CX, CORLUM          ; 列坐标置初值
```

```
        DEC     COUNT1
        JNZ     RETRY1          ; 8个扫描行未显示完，转
; 下面是采用移动图素的方法使小汽车从左向右移动
        MOV     COUNT1, 320-2   ; 移动318次（小汽车起始位置在第2列）
        MOV     CX, CORLUM
        ADD     CX, 15          ; 小汽车最右边点的行坐标→CX
REPET1: PUSH    CX
        MOV     COUNT2, 16+1    ; 小汽车横向为16个点，外加一个空白点
REPET2: MOV     DX, LINE
        MOV     COUNT3, 8       ; 准备移动8个扫描行纵向同一位的点
REPET3: MOV     AH, 0DH
        INT     10H             ; 读当前点
        MOV     AH, 0CH
        PUSH    CX
        INC     CX
        INT     10H             ; 写当前点右边的点（移动一个点）
        POP     CX
        INC     DX              ; 行坐标加1
        DEC     COUNT3
        JNZ     REPET3          ; 纵向8个扫描行的同一位未移完，转
        DEC     CX              ; 修改列坐标CX
        DEC     COUNT2
        JNZ     REPET2          ; 小汽车整体移动一个位置，未完转
        POP     CX
        INC     CX              ; 修改行坐标
        MOV     AH, 1
        INT     16H             ; 判键盘是否有按键
        JNZ     EXIT            ; 有按键，转EXIT
        DEC     COUNT1
        JNZ     REPET1          ; 未完，继续
EXIT:   POP     AX              ; 取显示方式
        MOV     AH, 0
        INT     10H             ; 恢复原有显示方式
        MOV     AH, 4CH
        INT     21H             ; 返回DOS
START   ENDP
CODE    ENDS
        END     START
```

【例 6.10】 快速动画显示

题目同 【例 6.9】。

编程指导：

上面的程序在进行动画显示时，小汽车的前进速度太慢，其原因是移动小汽车时采用了BIOS 的功能调用对图素进行移动来实现的。下面的程序完成同样的功能，但是在进行动画显示时，由于采用了直接对屏幕缓冲区的图素进行操作，所以使小汽车的前进速度很快。

程序清单

```
STACK     SEGMENT   PARA STACK 'STACK'
          DB        256 DUP (0)
STACK     ENDS
DATA      SEGMENT
LINE      DW   80                ; 小汽车图形起始点的行坐标
CORLUM    DW   0                 ; 小汽车图形起始点的列坐标
COUNT1    DW   0
COUNT2    DW   0
COUNT3    DW   0
; * * * * * * * * 小汽车的 16×8 点阵字模 * * * * * * * *
CAR_8X16DOT  DB   00000011B, 11100000B
             DB   00000010B, 01010000B
             DB   00000110B, 01011000B
             DB   01111111B, 11111110B
             DB   11111111B, 11111111B
             DB   11111011B, 11101111B
             DB   00001010B, 00101000B
             DB   00000100B, 00010000B
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA
START     PROC   FAR
          MOV    AX, DATA
          MOV    DS, AX
          MOV    AH, 0FH
          INT    10H
          PUSH   AX                ; 保存系统当前显示方式
          MOV    AX, 0004H
          INT    10H               ; 设置显示方式为：320×200 彩色图形方式
;
          MOV    AH, 0BH
```

```
        MOV     BH, 1
        MOV     BL, 0           ; 选择 0 号调色板
        INT     10H
;
        MOV     AH, 0CH
        MOV     DX, LINE
        ADD     DX, 8
        MOV     CX, 0
        MOV     AL, 3           ; 选 3 号（黄）颜色
REPET:  PUSH    AX
        INT     10H             ; 在屏幕的中部画一条黄色的道路
        POP     AX
        INC     CX
        CMP     CX, 319
        JNZ     REPET
; 下面是画一个小汽车
        MOV     AH, 0CH
        MOV     DX, LINE
        MOV     CX, CORLUM
        MOV     SI, OFFSET CAR _ 8X16DOT
        MOV     COUNT1, 8       ; 显示 8 个扫描行
RETRY1: MOV     COUNT2, 2       ; 一个扫描行显示两字节字模
RETRY2: MOV     COUNT3, 8       ; 一字节字模显示 8 个点
RETRY3: ROL     BYTE PTR [SI], 1
        MOV     AL, 0           ; 预置黑颜色
        JNC     NEXT            ; 当前位为 0，显示黑色，转 NEXT
        MOV     AL, 2           ; 置 2 号（红）颜色
NEXT:   PUSH    AX
        PUSH    CX
        INT     10H
        POP     CX
        POP     AX
        INC     CX
        DEC     COUNT3
        JNZ     RETRY3          ; 一字节的 8 个点未显示完，转
        INC     SI              ; 取下一字节字模
        DEC     COUNT2
        JNZ     RETRY2          ; 一个扫描行未显示完，转
        INC     DX              ; 行坐标加 1
```

```
        MOV     CX，CORLUM      ；列坐标置初值
        DEC     COUNT1
        JNZ     RETRY1          ；8 个扫描行未显示完，转
；下面利用直接对屏幕缓冲区操作进行快速动画显示
        MOV     SI，80 * (80/2) ；小汽车图素在屏幕缓冲区是偶地址→SI
        MOV     COUNT2，80      ；移动 80 个字节
REPET2：MOV     COUNT3，4       ；一字节移 4 次（一字节含有 4 个图素）
REPET3：PUSH    SI
        MOV     AX，0B800H
        MOV     ES，AX          ；偶扫描行段地址→ES
        CALL    MOV2BIT         ；小汽车偶地址的图素右移一位（2 比特）
        POP     SI
        PUSH    SI
        MOV     AX，0BA00H
        MOV     ES，AX          ；奇扫描行段地址→ES
        CALL    MOV2BIT         ；小汽车奇地址的图素右移一位（2 比特）
        POP     SI
        DEC     COUNT3
        JNZ     REPET3
        CALL    DELAY           ；延时
        INC     SI              ；地址加 1
        DEC     COUNT2
        JNZ     REPET2
EXIT：  POP     AX              ；取显示方式
        MOV     AH，0
        INT     10H             ；恢复原有显示方式
        MOV     AH，4CH
        INT     21H             ；返回 DOS
START   ENDP
DELAY   PROC    NEAR
        PUSH    CX
        MOV     CX，2801
DELAY1：LOOP    DELAY1          ；延时 10ms
        POP     CX
        RET
DELAY   ENDP
MOV2BIT PROC    NEAR
        PUSH    CX
        MOV     CX，4
```

```
REPMOV:  CLC
         RCR     BYTE PTR ES:[SI+0], 1
         RCR     BYTE PTR ES:[SI+1], 1
         RCR     BYTE PTR ES:[SI+2], 1
         RCR     BYTE PTR ES:[SI+3], 1
         RCR     BYTE PTR ES:[SI+4], 1
;
         CLC
         RCR     BYTE PTR ES:[SI+0], 1
         RCR     BYTE PTR ES:[SI+1], 1
         RCR     BYTE PTR ES:[SI+2], 1
         RCR     BYTE PTR ES:[SI+3], 1
         RCR     BYTE PTR ES:[SI+4], 1
         ADD     SI, 80
         LOOP    REPMOV
         POP     CX
         RET
MOV2BIT  ENDP
CODE     ENDS
         END     START
```

第七章　DOS 系统功能调用与文件管理编程实例

DOS 系统功能调用是 PC－DOS 为用户程序提供的一组功能性的子程序，其主要内容是为用户提供了进行磁盘文件管理的子程序包。另外还包括了用于设备管理、目录管理、内存管理以及其它管理的子程序。本章中，我们首先讨论一下 DOS 管理下的磁盘文件结构以及 DOS 提供的两套用于磁盘文件管理的功能调用——FCB 功能和把柄功能，然后按照系统功能调用的分类原则，列举了每一类系统功能调用中常用的系统功能调用的程序设计实例。

第一节　DOS 系统功能调用与文件管理

一、DOS 系统功能调用及其分类

在 DOS 2.10 版本中的系统功能调用共有 87 个子程序，编号从 00H～57H。这些子程序按功能分为设备管理、文件管理、目录管理、内存管理、以及其它管理等五大类，见表 7－1。

表 7－1　DOS 系统功能调用分类表

按功能分类		功能号（十六进制）
设备管理	字符设备	1，2，3，4，5，6，7，8，9，0A，0B，0C
	磁盘设备	0D，0E，19，1A，1B，1C，1F，2F，32，36
文件管理	传统文件操作	0F，10，13，14，15，16，21，22，24，27，28，29
	新增文件管理	3C，3D，3E，3F，40，41，42，43，44，45，46
目录管理	目录查找	11，12，4E，4F
	目录更改	17，23，56
	子目录操作	39，3A，3B，47
内存管理		48，49，4A，4B
其它管理	程序处理与中断	0，25，26，31，33，34，35，37，4C，4D，50，51，52，53，55
	日历和状态	2A，2B，2C，2D，2E，30，38，54，57

各功能调用详细说明请查阅附录 B。在这些系统功能调用中，最常用的是设备管理和文件管理功能调用。例如，设备管理中的功能调用 1 号（键盘输入）、2 号（显示字符）、5 号（打印字符）、9 号（显示字符串）及 0AH 号（接收键盘输入的字符串）等。在磁盘文件管理中，常用的是新增的文件管理功能调用 3CH～42H 号。

对于所有的系统功能调用，使用时一般需要经过以下三个步骤：

(1) 子程序编号送 AH；

(2) 子程序的入口参数送相应的寄存器；

(3) 发出中断请求：INT 21H。

例如，显示一个字符串："Good morning!"

```
MSG     DB " Good morning!  $"
...
MOV     AH, 9
MOV     DX, OFFSET MSG
INT     21H
```

调用结束后，一般都有出口参数，通过出口参数可以知道功能调用的成功与否。

二、DOS 磁盘文件管理

在对文件进行读写时，DOS 提供了两套功能相同的系统调用，一套是和 CP/M 兼容的，称作 FCB 功能；另一套的和 UNIX 兼容的，叫作文件把柄功能。

(一) DOS 管理下的磁盘文件结构

DOS 管理下的文件由若干个记录块所组成，如图 7-1。通常情况下，每个记录块含有 128 个记录，其编号为 0～127；一个记录内含有的字节数称为记录，这个记录长度对给定文件来说是一个常数，由 FCB 中的一个字段指定。另外，文件把记录块 0 内的记录号 0 称为相对记录 0，随后的记录相对号递增 1；当记录块 0 内所有记录编完号后，紧接着对记录块 1 中的记录进行顺序编号，直到所有的记录块中的所有记录编号完毕。这样在磁盘文件读写时，要定位某一个记录，有以下两种办法：

(1) 顺序读写：给定记录块号和块内记录号，前者称为当前记录块，后者称为当前记录号。在指定的记录上读写后，自动修改指针使之指向下一个记录单元。

(2) 随机读写：指定相对记录号，以任意的次序读写任一指定的记录。在随机读写之前，DOS 把相对记录号转换成等价的当前块号和当前记录号，其计算方法是：

当前块号＝［相对记录号÷每块含有的记录个数］取整

当前记录号＝［相对记录号÷每块含有的记录个数］取余＋1

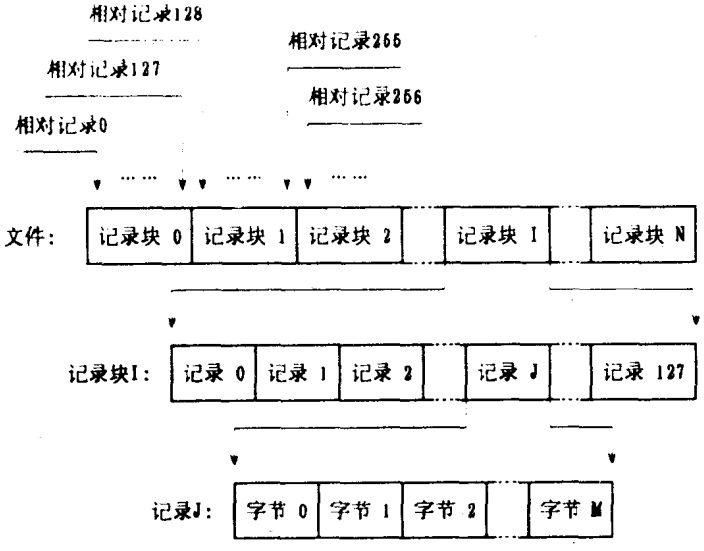


图 7-1　DOS 管理下的磁盘文件结构

（二）文件控制块 FCB 的结构

在使用 FCB 功能时，总要在内存中建立一个所谓的文件控制块 FCB（File Control Block）的数据结构来记录有关被打开文件的一些信息。这样，用户在程序中用串操作指令将格式化的 FCB 传递到自己设置的 FCB 区域，再根据 FCB 中的文件名和其它有关信息去访问目录，找到文件名相符合的文件，接着进行磁盘与内存之间的文件传送。为此 DOS 提供的功能调用子程序有：文件的打开与关闭，文件的建立与删除，目录项的查找以及对磁盘的顺序或随机读写。这些都需要有一文件控制块来记录、管理和控制文件的资料信息。这就是说，FCB 是用户程序与 DOS 文件管理子程序之间的调用接口。

为了适应上述文件管理的需要，DOS 使用的文件控制块 FCB 的标准格式如图 7－2 所示。

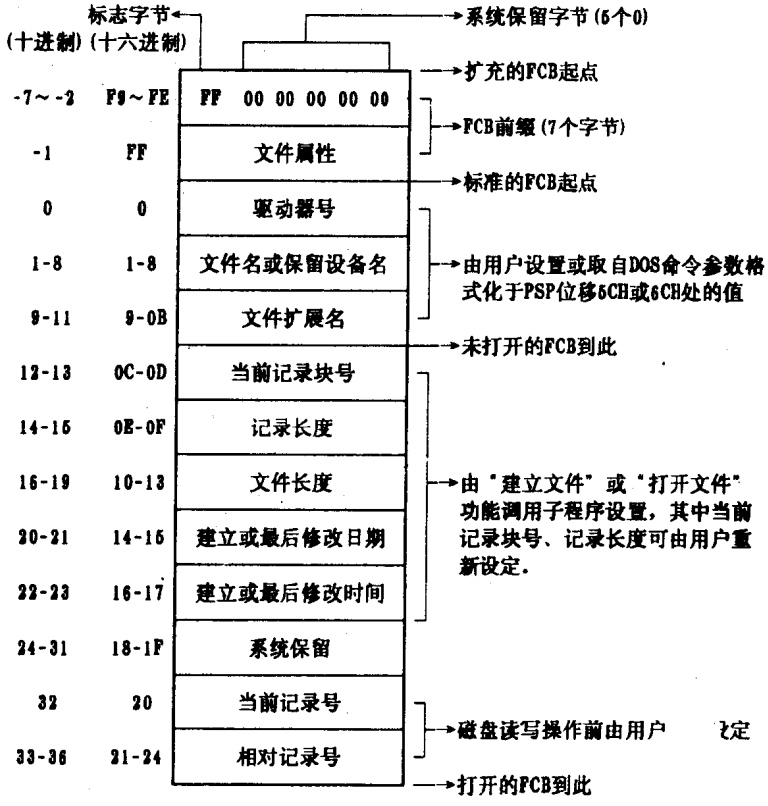


图 7－2　FCB 的标准格式

FCB 的格式有标准的和扩充的两种，前者用于常规的文件管理操作，有 37 个字节；后者则用于对具有特殊属性文件的管理操作，它是在标准格式的 FCB 前增加了 7 个字节的前缀，即扩充的 FCB 为 44 个字节，如图 7－2 所示。

对标准的或扩充的 FCB，都存在着打开和未打开两种状态。对文件进行建立、打开、删除和查找目录登记项时，由指针 DS：DX 指向一个未打开的 FCB；而对文件进行顺序或随机读写以及关闭文件时，指针 DS：DX 必须指向一个已打开的 FCB。一个未打开的 FCB 是由 FCB 的前缀（如果使用前缀的话）、驱动器号、文件名和扩展名组成（图 7－2 中已注明）。

一个打开的 FCB 是在一个未打开的 FCB 上，通过建立文件（功能调用 16H）或打开文件

(功能调用 0FH) 的功能调用填写 0CH～0FH 字段的内容而形成。而且一个 FCB 被打开时，当前记录块号字段被初始化为 0，记录长度字段被初始化为 128。用户可根据自己的需要改变记录长度。一旦设定，磁盘的读写操作即按此值为记录的大小而进行。

1. 使用 FCB 进行磁盘文件操作时的注意事项：

(1) 在对磁盘顺序读写前，要对打开的 FCB 设定当前记录块号和当前记录号（后者 DOS 并未初始化）；而在随机读写前，要对打开的 FCB 设定相对记录号（它也未被 DOS 初始化）；

(2) 驱动器字段在 FCB 打开前后的表示是不同的；

打开前：0 表示约定的驱动器号，1 表示驱动器 A，2 表示驱动器 B 等。

打开后：1 表示驱动器 A，2 表示驱动器 B。换句话说，打开后 0 被实际驱动器号取代。

(3) 字段 14H～15H 为文件建立或最后修改的日期，它和文件目录中的日期字段相匹配，可看作一个 16 位的字，其中 00～04 位表示“日”；05～08 位表示“月”；09～15 位表示“年”。同样，字段 16H～17H 表示时间，其中 00～04 以两秒为增量表示“秒”，而位 05～10 表示“分”；位 11～15 表示“小时”；

(4) 所有字段的存储方式是：低字节存于低地址，高字节存于高地址。例如，一个长为 128 字节的记录，在存放这个记录长度时，使用位移地址 0EH 和 0FH 两个字节单元，其存放安排是 (0EH) =80H，(0FH) =00；

(5) 当用户在程序中设置一存储区来存放 FCB 时，字节 0～15 和 32～36 必须由用户自行设置（其中字节 12～15 也可以使用 DOS 的约定值）；而字节 16～ 31 是由 DOS 设置，用户程序不能改变它们。

2. 扩充的 FCB 使用方法：

扩充的 FCB 比标准的 FCB 增加了 7 个字节，其中 FFH 为标志字节，它表明该 FCB 为扩充的 FCB，其后的 5 个字节 0 是系统保留的，在目前的 DOS 版本下不使用。属性字节的含义同文件目录登记项中的第 11 字节一样，它指出文件是否为普通、只读、系统、隐含、归档等文件或是否为卷标、子目录等。因此，扩充的 FCB 一般用于建立和查找有特定属性的文件。

(1) 当建立一个文件时，使用一个未打开的 FCB，其属性字节被设置为所需的只读或隐含文件，然后调用“建立文件”子功能。这样，该文件属性标志为只读、隐含或者不能被打开修改，或者在目录显示中不出现；

(2) 为查看磁盘上所存放的每一文件的目录登记项，可使用一个未打开的扩充的 FCB。同样，在属性字节中可设置为“隐含＋系统＋子目录”(即相应的 3 位都置 1)，然后，调用“查找登记项”子功能，能在设定的磁盘传输地址处，得到与所查找属性相同的一个文件目录登记项的 32 个字节。

3. 使用 FCB 访问文件的常规序列

(1) 清除将要使用的文件控制块；

(2) 从用户定义的文件名缓冲区、默认的文件控制块或程序段前缀中的 5CH 中获得文件名；

(3) 如果没能从默认的文件控制块中获得文件名，便使用功能 29H 将文件名缓冲区中的文件名送入新的文件控制块；

(4) 打开文件 (0FH 号) 或建立文件 (16H 号)；

(5) 如不使用默认的记录大小，应在 FCB 中建立记录大小域·

(6) 如果执行随机记录的 I/O，在 FCB 中要建立记录号域；

(7) 使用功能 1AH 建立磁盘传输区，除非自上次调用此功能后缓冲区地址未发生变化。如果不执行该功能调用，则磁盘传输区的默认地址是 PSP 中的偏移量 0080H，传输区的大小为 128 个字节；

(8) 请求读、写记录的操作（功能 14H ——顺序读，15H ——顺序写，21H ——随机读，22H ——随机写，27H ——随机块读，28H ——随机块写）；

(9) 如果未完成读、写操作，转步骤⑥，否则关闭文件（功能 10H 号）。

(三) DOS 的文件句柄功能

在 DOS 2.00 以上的版本中，磁盘文件目录采用了树型目录结构，由于 FCB 功能不支持树型目录结构，所以对于子目录下的文件 FCB 无法对其进行操作。于是 DOS 2.00 以上的版本中增加了所谓的文件句柄功能调用。这种存取文件和记录的方法与 UNIX 操作系统下存取文件的方法是类似的。使用这些功能调用进行磁盘文件操作时，需要一个以零为结尾的 ASCII 码字符串，该字符串中包括驱动器标识符、路径名、文件名及扩展名。

例如，下面的一个文件说明：

A：\SUBDIR\ABC.ASM

若要打开或创建一个磁盘文件时，这个 ASCII 码字符串的首地址必须由 DS:DX 寄存器传送给 DOS。若 DOS 打开或创建操作是成功的，则 DOS 回送一个文件句柄到 AX 中，此后对该文件的所有操作只需要利用这一句柄即可。通常是把句柄置于 BX 中，然后调用 DOS 的相应功能。若调用操作成功，则返回时 CF 被清 0；若操作失败，则返回时 CF 标志被置位，并且 AX 中返回一个错误代码。

利用文件句柄功能进行文件读写操作时，不再分为顺序读/写和随机读写功能调用，只有一组读/写功能调用（3FH 和 40H）。文件也没有分块的概念，也没有当前记录号和相对记录号的概念。当打开或创建一个文件后，DOS 便建立了一个文件指针（双字）指向这一文件的开头，以后所有的读/写功能调用都是从当前指针所指的位置开始读写。要想实现随机读/写，可以借助于 42H 号功能调用事先移动文件指针到要求的字节位置。每次读写的字节数可以在 1 到 64KB 之间任意选择。

利用文件句柄功能存取磁盘文件的一般过程如下：

(1) 利用缓冲区输入功能（系统功能调用 0AH 号）或由程序段前缀中的 DOS 提供的命令行尾取用户给出的文件名；

(2) 在文件说明字符串后加一个零；

(3) 利用系统功能调调用 3DH（一般用方式 2）打开文件或利用功能 3CH（一般情况下 CX 置 0，以生成一个普通属性的文件）创建文件，在打开或创建成功以后保存 AX 中返回的文件句柄；

(4) 利用系统 42H 号功能调用设置文件读写指针。用户可以相对于三种位置设置文件指针：从文件的开始位置、从当前指针位置以及从文件末尾位置。如果是顺序读/写文件，则可以跳过这一步，因为系统会自动管理文件指针；

(5) 读文件（功能 3FH）或写文件（功能 40H）。这两个功能都要求 BX 中包含文件句柄，CX 中包含记录的长度，DS：DX 指向数据缓冲区首地址。在读的过程中，若实际读的字节数小于请求读的字节数，则已到了文件末尾；写的过程中，若实际写的字节数小于请求写的字

节数，则表示文件所在的磁盘满；以上两种情况出现时都不返回错误代码，即不置进位标志，用户需要根据 AX 中的返回值进行判断处理；

（6）若文件读/写操作未完成，回到（4）继续。反之，关闭文件（功能 3EH）。

第二节　设备/文件管理功能调用编程实例

【例 7.1】 文件属性的查询与修改

编制一个程序，接收键盘输入的任意一个文件名，查询并显示出该文件的属性，然后接收用户输入的新的属性，对文件的属性进行修改。

编程指导：

文件属性反映了文件的存取权限。常用的文件属性有普通属性、只读属性、隐藏属性和系统属性等。普通属性的文件既可以被阅读，又可以被修改和删除；只读属性的文件只允许阅读，不能被修改和删除；隐藏属性的文件在通常的 DOS 目录列表时，不被显示出来；系统属性的文件在通常情况下既不能被列表显示，又不能被修改和删除。不同的文件属性使用不同的字节数据来表示，普通属性用 00H 表示，只读属性用 01H 表示，隐藏属性用 02H 表示，系统属性用 04H 表示，另外还可以将几个属性组合起来形成一个组合型的属性，例如 07H 表示该文件兼有只读、隐藏和系统等三个属性。在 DOS 系统功能调用中的 43H 号功能可以对任何文件的属性进行读取和设置，调用该功能时，AL 等于 0 表示读取文件属性，返回的属性值在 CL 中；AL 等于 1 表示设置文件属性，要设置的文件属性事先送入 CX 中。

程序清单

```
STACK       SEGMENT    PARA STACK 'STACK'
            DB         128 DUP (0)
STACK       ENDS
DATA        SEGMENT
INPUT_MSG1      DB  '****查询/改变文件属性****', 13, 10
                DB  '请输入文件标识符：$'
FILE_NAME       DB  30
                DB  ?
                DB  30 DUP (0)
DISP_ATRIBUTE   DB  13, 10,'该文件的属性为：$'
INPUT_MSG2      DB  13, 10
                DB  '文件属性对照表：0——普通，1——只读', 13, 10
                DB  '                2——隐藏，4——系统', 13, 10
                DB  '请输入新的属性值：$'
DATA        ENDS
CODE        SEGMENT
            ASSUME     CS: CODE, DS: DATA, ES: DATA
START       PROC    FAR
```

```
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     DX, OFFSET INPUT _ MSG1        ; 显示提示信息
        MOV     AH, 9
        INT     21H
        MOV     DX, OFFSET FILE _ NAME
        MOV     AH, 0AH                        ; 接收键入的文件名
        INT     21H
        MOV     BX, OFFSET FILE _ NAME+2
        MOV     AH, 0
        MOV     AL, FILE _ NAME+1
        ADD     BX, AX
        MOV     BYTE PTR [BX], 0               ; 文件名尾部置 0
        MOV     DX, OFFSET DISP _ ATRIBUTE
        MOV     AH, 9
        INT     21H
        MOV     DX, OFFSET FILE _ NAME+2
        MOV     AX, 4300H                      ; 取文件属性
        INT     21H
        MOV     DL, CL
        OR      DL, 30H                        ; 属性值转 ASCII 码
        MOV     AH, 2                          ; 显示文件属性
        INT     21H
        MOV     DX, OFFSET INPUT _ MSG2
        MOV     AH, 9
        INT     21H
        MOV     AH, 1
        INT     21H                            ; 接收键入的新属性值
        MOV     CL, AL
        AND     CX, 000FH                      ; ASCII 码转二进制数
        MOV     DX, OFFSET FILE _ NAME+2
        MOV     AX, 4301H                      ; 修改属性
        INT     21H
        MOV     AH, 4C00H
        INT     21H                            ; 返回 DOS
START   ENDP
CODE    ENDS
        END     START
```

【例 7.2】 分页显示文本文件

从 DOS 的格式化参数块中取得所需的文件名，再利用 DOS 的“打开文件”以及“顺序读”两个功能调用从磁盘上读取文件内容，按每页 23 行在屏幕上分页显示，显示完一页后暂停，提示用户：“是否继续显示？(Y/N):”，并等待用户按“Y”或“N”键进行选择。若用户按“Y”，则继续显示下一页；按“N”键结束程序返回 DOS，按其它键不响应。

程序清单

```
STACK       SEGMENT   PARA STACK 'STACK'
            DB        128 DUP (0)
STACK       ENDS
DATA        SEGMENT
FCB         DB  37 DUP (0)                    ; 文件控制块
LCNT        DB  0                             ; 行计数单元
DTA         DB  0                             ; 磁盘缓冲区
GOONMSG     DB  '是否继续显示? (Y/N): $'
ERRMSG1     DB  13, 10,'文件打开错! $'
ERRMSG2     DB  13, 10,'读文件错! $'
CRLF        DB  'Y', 0DH, 0AH,' $ '
DATA        ENDS
DISP_STRING   MACRO     STRING                ; 显示字符串的宏定义
              MOV       DX, OFFSET STRING
              MOV AH, 9
              INT       21H
              ENDM
CODE          SEGMENT   PUBLIC 'CODE'
              ASSUME    CS: CODE
START       PROC    FAR
            PUSH    DS
            XOR     AX, AX
            PUSH    AX
            MOV     AX, DATA
            MOV     ES, AX
            ASSUME  ES: DATA
            MOV     SI, 05CH
            MOV     DI, OFFSET FCB
            MOV     CX, 12
            CLD
            REP     MOVSB                     ; 从 PSP 中取格式化的文件名送 FCB
```

```
        MOV     DS, AX
        ASSUME  DS: DATA
        MOV     DX, OFFSET DTA          ; 设置磁盘缓冲区
        MOV     AH, 1AH
        INT     21H
        MOV     DX, OFFSET FCB
        MOV     AH, 0FH                 ; 打开文件
        INT     21H
        CMP     AL, 0
        JZ      NEXT1                   ; 打开正确，转
        DISP_STRING ERRMSG1             ; 打开错，显示提示信息
        RET                             ; 返回 DOS
NEXT1:  MOV     WORD PTR FCB+0CH, 0     ; 当前块号置 0
        MOV     WORD PTR FCB+0EH, 1     ; 记录长度置 1
        MOV     WORD PTR FCB+20H, 0     ; 当前记录号置 0
AGAIN:  MOV     DX, OFFSET FCB
        MOV     AH, 14H                 ; 读一个记录
        INT     21H
        CMP     AL, 0
        JZ      NEXT2                   ; 读正确，转
        CMP     AL, 03
        JZ      NEXT2                   ; 缓冲区不满，转
        DISP_STRING ERRMSG2             ; 显示读盘错
        JMP     READ_END
NEXT2:  MOV     DL, DTA
        CMP     DL, 1AH                 ; 是文件尾吗?
        JZ      READ_END                ; 是文件尾，转程序结束
        MOV     AH, 2                   ; 不是文件尾，显示该字符
        INT     21H
        CMP     DTA, 0AH                ; 是行尾吗?
        JNZ     AGAIN                   ; 不是行尾，转
        INC     LCNT                    ; 是行尾，行计数加 1
        CMP     LCNT, 24                ; 显示满一页吗?
        JNZ     AGAIN                   ; 不满一页，转
        MOV     LCNT, 0                 ; 满一页，行计数单元清 0
        DISP_STRING GOONMSG             ; 显示:" 是否继续?"
RETRY:  MOV     AH, 7                   ; 接收键盘输入，不回显
        INT     21H
        AND     AL, 0DFH                ; 字母小写变大写
```

```
                CMP     AL,'Y'
                JNZ     NEXT3                   ; 不是" Y" 键，转
                DISP_STRING CRLF                ; 是" Y" 键，显示" Y" 及回车换行
                JMP     AGAIN                   ; 转 AGAIN，继续
NEXT3:          CMP     AL,'N'
                JZ      READ_END                ; 是:" N"，转程序结束
                MOV     DL，07
                MOV     AH，2                   ; 是非法键，响铃
                INT     21H
                JMP     RETRY                   ; 转 RETRY，重新接收
READ_END:       MOV     DX，OFFSET FCB          ; 关闭文件
                MOV     AH，10H
                INT     21H
                RET                             ; 返回 DOS
START           ENDP
CODE            ENDS
                END     START
```

【例 7.3】 获取汉字 24×24 点阵打印字模

编写一个程序，根据输入的汉字内码，从 C 盘上 24×24 点阵打印字库中获取该汉字的点阵字模，以文件形式存放于当前盘上。

编程指导：

在 24×24 打印点阵字库中，汉字共分为 87 个区，编号从 01～87（其中第 10～15 区为空白区）。每个区中存放有 94 个汉字的打印字模，而每个汉字的打印字模占有 72 个字节。所以，整个汉字打印字库长度为 87×94×72＝588816 bytes。由于所有汉字的打印字模都是 72 个字节，这样我们可以把一个汉字的 72 字节字模看作为一个记录。要获取某个汉字的打印字模，首先将该汉字的内码转换成国标码，再由国标码转换成区位码，最后根据公式“区号×94＋位号”计算出该汉字在打印字库中的相对记录号。然后，填写好文件控制块中的记录长度、相对记录号，采用“随机读”功能调用读一个记录即可。要把取出的 72 字节汉字打印字模写盘保存时，把欲写的相对记录号置为 0，记录长度置为 72 字节，采用“随机写”功能调用写盘即可。

程序清单

```
STACK       SEGMENT     PARA STACK 'STACK'
            DB          128 DUP (0)
STACK       ENDS
DATA        SEGMENT
CCLIB_FCB   DB  03,'CLIB24',20H,20H,20H,20H,20H,25 DUP(0)   ;文件控制块 1
DOT24X24_FCB        DB  ?,11 DUP(20H),25 DUP(0)             ;文件控制块 2
```

```
INPUT_MSG1          DB  13, 10,'请输入汉字: $'
INPUT_MSG2          DB  13, 10,'请输入点阵写盘文件名: $'
OUTPUT_MSG1         DB  13, 10,'打开字库出错! $'
OUTPUT_MSG2         DB  13, 10,'读字库出错! $'
OUTPUT_MSG3         DB  13, 10,'建立点阵文件出错! $'
OUTPUT_MSG4         DB  13, 10,'写点阵文件出错! $'
ERR_NAME            DB  13, 10,'文件名错! $'
INPUT_WORD          DB  3
                    DB  ?
                    DB  3 DUP (0)          ; 汉字内码存放单元
INPUT_DOT_NAME      DB  12
                    DB  ?
                    DB  12 DUP (0)         ; 点阵字模写盘文件名暂存单元
HCQ                 DB  80 DUP (' $')      ; 点阵字模缓冲区
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA, ES: DATA
START   PROC    FAR
        PUSH    DS
        XOR     AX, AX
        PUSH    AX
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        MOV     DX, OFFSET HCQ             ; 设置磁盘传送地址
        MOV     AH, 1AH
        INT     21H
        MOV     AH, 19H
        INT     21H                        ; 取当前驱动器号
        MOV     DOT24X24_FCB, AL           ; 当前驱动器号送 FCB2 相应单元
        MOV     DX, OFFSET CCLIB_FCB
        MOV     AH, 0FH
        INT     21H                        ; 打开字库
        OR      AL, AL
        JZ      NEXT1                      ; 打开正确, 转
        MOV     DX, OFFSET OUTPUT_MSG1
        MOV     AH, 9
        INT     21H
        RET
```

```
NEXT1:  MOV   WORD PTR CCLIB_FCB+14,72   ;记录长度设置为72字节
        MOV   DX, OFFSET INPUT_MSG1
        MOV   AH, 9                      ;屏幕提示
        INT   21H
        MOV   DX, OFFSET INPUT_WORD
        MOV   AH, 0AH                    ;接收输入的汉字内码
        INT   21H
        MOV   DX, WORD PTR INPUT_WORD+2  ;汉字内码送DX
        XCHG  DH, DL                     ;校正内码高低字节
        AND   DX, 7F7FH                  ;内码转国标码
        SUB   DX, 2121H                  ;国标码转区位码
        MOV   AL, 94
        MUL   DH
        XOR   DH, DH
        ADD   AX, DX                ;计算该汉字字模在字库中的相对记录号
        MOV   BX, OFFSET CCLIB_FCB
        MOV   [BX+33], AX                ;相对记录号送FCB1相应单元
        XCHG  DX, BX
        MOV   AH, 21H                    ;随机读一个记录
        INT   21H
        OR    AL, AL
        JZ    NEXT2                      ;读盘正确，转
        MOV   DX, OFFSET OUTPUT_MSG2
        MOV   AH, 9
        INT   21H
        RET
NEXT2:  MOV   AH, 10H                    ;关闭字库文件
        INT   21H
        MOV   DX, OFFSET INPUT_MSG2
        MOV   AH, 9                      ;屏幕提示
        INT   21H
        MOV   DX, OFFSET INPUT_DOT_NAME
        MOV   AH, 0AH                    ;接收输入的写盘文件名
        INT   21H
        MOV   SI, OFFSET INPUT_DOT_NAME+2
        MOV   DI, OFFSET DOT24X24_FCB
        MOV   AX, 2901H
        INT   21H                        ;分析文件名
        OR    AL, AL
```

```
        JZ      OK                              ; 是有效的文件名，转 OK
        MOV     DX，OFFSET ERR _ NAME           ; 显示“文件名错!”提示信息
        JMP     SHORT ERRDISP
OK：    MOV     DX，OFFSET DOT24X24 _ FCB
        MOV     AH，16H                         ; 在当前盘上建立点阵写盘文件
        INT     21H
        OR      AL，AL
        JZ      NEXT3                               ; 文件建立正确，转
        MOV     DX，OFFSET OUTPUT _ MSG3
ERRDISP：MOV    AH，9
        INT     21H
        RET
NEXT3：  MOV    WORD PTR DOT24X24 _ FCB+14，72  ; 记录长度送 FCB2 相应单元
        MOV     WORD PTR DOT24X24 _ FCB+33，0   ; 相对记录号送 FCB2 相应单元
        MOV     DX，OFFSET DOT24X24 _ FCB
        MOV     AH，22H                             ; 随机写一个记录
        INT     21H
        OR      AL，AL
        JZ      NEXT4                               ; 写文件正确，转
        MOV     DX，OFFSET OUTPUT _ MSG4
        MOV     AH，9
        INT     21H
        RET
NEXT4：  MOV    DX，OFFSET DOT24X24 _ FCB
        MOV     AH，10H                             ; 关闭写盘文件
        INT     21H
        RET
START   ENDP
CODE    ENDS
        END     START
```

【例 7.4】 文件拷贝程序

编写一个程序，实现 DOS 命令：COPY MY1. ASM MY2. ASM 的功能。

编程指导：

我们采用新增的文件管理系统功能调用，首先打开文件 MY1. ASM，接着读该文件到内存缓冲区 HCQ，最后关闭该文件。为了将缓冲区 HCQ 中的内容写入 MY2. ASM 中，首先要建立一个 MY2. ASM 文件，然后执行写操作，最后关闭文件 MY2. ASM。这样就完成了将 MY1. ASM 文件拷贝到 MY2. ASM 文件的功能。

程序清单

```
STACK       SEGMENT STACK ' STACK '
            DB   128 DUP (0)
STACK       ENDS
DATA        SEGMENT
WJM1        DB   ' MY1. ASM ' , 0
WJM2        DB   ' MY2. ASM ' , 0
COUNT       EQU  10000
HCQ         DB   COUNT DUP (0)
ZJS         DW   ?
DATA        ENDS
CODE        SEGMENT
            ASSUME  CS: CODE, DS: DATA
COPYBEG:    MOV     AX, DATA
            MOV.    DS, AX
            MOV     AX, 3D00H
            MOV     DX, OFFSET WJM1    ; 打开文件 MY1. ASM
            INT     21H
            MOV     BX, AX             ; 文件号送 BX
            MOV     CX, COUNT
            MOV     DX, OFFSET HCQ
            MOV     AH, 3FH
            INT     21H                ; 读文件 MY1. ASM 到 HCQ 中
            MOV     ZJS, AX
            MOV     AH, 3EH
            INT     21H                ; 关闭文件 MY1. ASM
            MOV     AH, 3CH
            MOV     CX, 0
            MOV     DX, OFFSET WJM2
            INT     21H                ; 以普通文件属性建立文件 MY2. ASM
            MOV     BX, AX
            MOV     CX, ZJS
            MOV     DX, OFFSET HCQ
            MOV     AH, 40H            ; 将 HCQ 中的内容写入文件 MY2. ASM
            INT     21H
            MOV     AH, 3EH            ; 关闭文件 MY2. ASM
            INT     21H
            MOV     AH, 4CH
            INT     21H
```

```
CODE        ENDS
            END     COPYBEG
```

【例 7.5】 过滤文本文件行尾多余的空格

在 A 盘上有一源程序文件 ABC. ASM，字节长度不超过 20000。在该文件的某些文本行，最后一个非空字符与回车符（0DH）之间有连续的空格符（20H）与制表符（09H），试编程将这样的空格符及制表符过滤掉，而对于行中其它两个非空字符之间的空格符及制表符不过滤。过滤完成后，把结果存入 A 盘上另一文件 XYZ. ASM 中。

编程指导：

按照题意，要编程解决这样的问题，假设该源程序文件有这样一行内容：

NEXT: MOV AL, 30H

其处理的过程如图 7-3 所示。

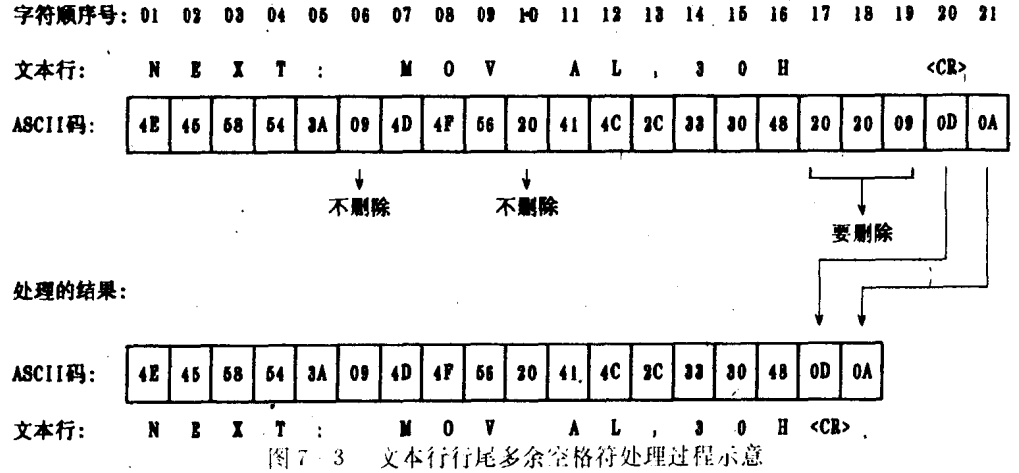


图 7-3 文本行行尾多余空格符处理过程示意

从图 7-3 中可以看到，字符“H”和“回车符（0DH）”之间有连续的空格串：20H，20H，09H20H，09H，程序就是要过滤掉这样的空格符和制表符。而对于“:”与“M”之间以及“V”与“A”之间的制表符或空格符不能过滤。下面以框图方式给出过滤的方法，见图 7-4。

程序清单

```
STACK       SEGMENT STACK 'STACK'
            DB   128 DUP (0)
STACK       ENDS
DATA        SEGMENT
BUFFER2       EQU THIS BYTE                    ; 缓冲区 2（写盘缓冲区）
BUFFER1       DB  20000 DUP (0)                ; 缓冲区 1（读盘缓冲区）
FILENAME1     DB  'A: ABC. ASM', 0
FILENAME2     DB  'A: XYZ. ASM', 0
ERR_MSG1      DB  '打开文件错!', 0DH, 0AH, 24H
ERR_MSG2      DB  '建立文件错!', 0DH, 0AH, 24H
```

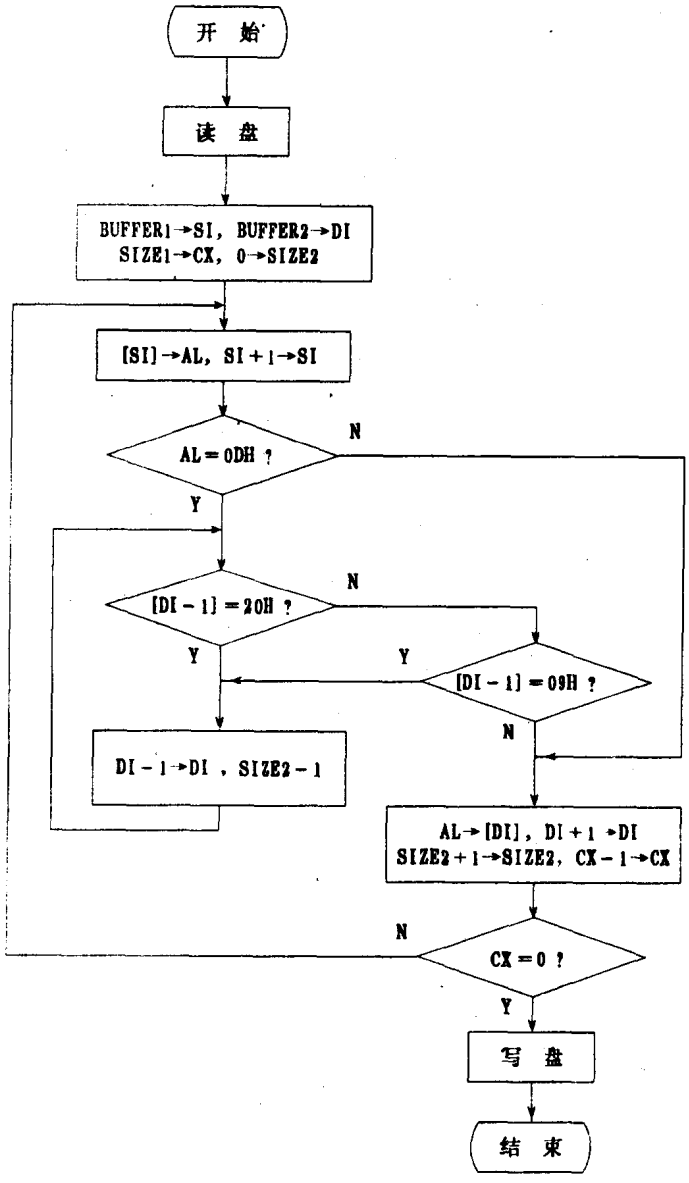


图 7-4 过滤行尾空格符的流程图

```
SIZE1       DW   0                    ; ABC. ASM 文件实际字节数
SIZE2       DW   0                    ; 写盘文件 XYZ. ASM 字节数
DATA        ENDS
CODE        SEGMENT
            ASSUME    CS: CODE, DS: DATA, ES: DATA
START       PROC      FAR
            PUSH      DS
            XOR       AX, AX
            PUSH      AX
            MOV       AX, DATA
```

```
            MOV     DS, AX
            MOV     ES, AX
            MOV     AX, 3D00H                  ; 打开文件 ABC. ASM 供读
            LEA     DX, FILENAME1
            INT     21H
            JNC     READ_FILE                  ; 打开正确，转
            LEA     DX, ERR_MSG1
ERR_EXIT:   MOV     AH, 9                      ; 显示出错信息
            INT     21H
            RET                                ; 错误返回
READ_FILE:  MOV     BX, AX                     ; 文件 ABC. ASM 的句柄送 BX
            MOV     CX, 20000
            MOV     DX, OFFSET BUFFER1
            MOV     AH, 3FH                    ; 读 20000 字节
            INT     21H
            MOV     SIZE1, AX                  ; 实际读入的字节数送 SIZE1
            MOV     AH, 3EH                    ; 关闭文件 ABC. ASM
            INT     21H
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
            LEA     SI, BUFFER1                ; 循环初始化部分
            LEA     DI, BUFFER2
            MOV     CX, SIZE1
            MOV     SIZE2, 0
RETRY:      MOV     AL, [SI]                   ; 取一字符→AL
            INC     SI                         ; 修改地址指针
            CMP     AL, 0DH                    ; 是回车符吗?
            JNZ     NEXT4                      ; 不是回车符，转
NEXT1:      CMP     BYTE PTR [DI-1], 20H       ; 前一字符是空格符吗?
            JNZ     NEXT2                      ; 不是空格符，转 NEXT2
            JMP     SHORT NEXT3                ; 是空格符，转 NEXT3
NEXT2:      CMP     BYTE PTR [DI-1], 09H       ; 前一字符是制表符吗?
            JNZ     NEXT4                      ; 不是制表符，转 NEXT4
NEXT3:      DEC     DI                         ; 目的指针减 1，滤掉一个空格符
            DEC     SIZE2                      ; 写盘字节数减 1
            JMP     SHORT NEXT1                ; 转 NEXT1，判前面是否还有空格符
NEXT4:      MOV     [DI], AL                   ; 当前字符存目的缓冲区
            INC     DI                         ; 目的地址指针加 1
            INC     SIZE2                      ; 写盘字节数加 1
            LOOP    RETRY                      ; 循环直至 CX=0
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
              MOV    AH, 3CH                 ;建立文件
              MOV    CX, 0                   ;属性为普通文件
              LEA    DX, FILENAME2
              INT    21H
              MOV    BX, AX                  ;文件句柄送 BX
              JNC    WRITE_FILE              ;建立正确，转
              LEA    DX, ERR_MSG2
              JMP    ERR_EXIT
WRITE_FILE:   MOV    AH, 40H                 ;写文件
              MOV    DX, OFFSET BUFFER2
              MOV    CX, SIZE2               ;字节数送 CX
              INT    21H
              MOV    AH, 3EH                 ;关闭文件 XYZ.ASM
              INT    21H
              RET                            ;返回 DOS
START         ENDP
CODE          ENDS
              END    START
```

第三节 内存管理功能调用编程实例

【例 7.6】 加载并执行一个子进程

编写一个应用程序，用于加载执行 DOS 的外部命令 CHKDSK．COM。要求：加载运行 CHKDSK 时，检验 B 驱磁盘上文件分配表的完好性以及扩展名为．EXE 的所有文件的磁盘分配特征，并显示文件目录及文件分配表的错误，然后在提出检验状态报告之后终止返回应用程序，再由应用程序返回 DOS。出现错误时，应显示出错信息，再返回 DOS。

编程指导：

一个应用程序调用 EXEC 功能加载执行一个子进程时，大致要经历以下步骤：

(1) 在数据段内设置一个调用参数块。格式见例中的定义；

(2) 计算应用程序本身实际使用的节长度；

(3) 释放不必要的内存空间；

(4) 判断有否足够的内存空间容纳被装入程序；

(5) 进栈保护重要的寄存器；

(6) 将堆栈入口保存于代码段中可寻址的单元中；

(7) 建立调用 EXEC 的入口指针；

(8) 调用 EXEC 功能加载并执行子进程；

(9) 子进程执行完毕，返回父进程；

(10) 恢复堆栈入口指针及重要的寄存器内容；

(11) 父进程返回 DOS。

程序清单

```
STACK     SEGMENT   STACK 'STACK'
          DW        64 DUP (0)
STACK     ENDS
DATA      SEGMENT   PUBLIC 'DATA'
Pgm_name          DB  'A:\CHKDSK.COM', 0      ; 被装载的程序文件名字符串
Par_blk           DW  Envir                   ; 环境块的段址
                  DW  OFFSET    Cmd_line      ; 命令参数的位移
                  DW  SEG       Cmd_line      ; 命令参数的段址
                  DW  OFFSET    Fcb1          ; FCB1 的位移
                  DW  SEG       Fcb1          ; FCB1 的段址
                  DW  OFFSET    Fcb2          ; FCB2 的位移
                  DW  SEG       Fcb2          ; FCB2 的段址
Cmd_line          DB  10,'B:*.EXE/F', 0DH     ; 命令参数 (F 表示检验 FAT)
Fcb1              DB  2                       ; B 驱动器号
                  DB  8 DUP ('?')             ; 文件名
                  DB  'EXE'                   ; 扩展名
                  DB  25 DUP (0)              ; FCB 其余的空间
Fcb2              DB  0                       ; 文件控制块 2 (未用)
                  DB  11 DUP (' ')
                  DB  25 DUP (0)
ERR_MSG1          DB  '内存分配错误!','0DH, 0AH
MSG1_LENGTH       EQU $-ERR_MSG1
ERR_MSG2          DB  '调用 EXEC 失败!', 0DH, 0AH
MSG2_LENGTH       EQU $-ERR_MSG2
DATA              ENDS
Envir  SEGMENT    PARA 'ENVIR'                ; 环境段
                  DB  'PATH=', 0
                  DB  'COMSPEC=A:\COMMAND.COM', 0
                  DB  0                       ; 环境串结束
Envir  ENDS
CODE   SEGMENT    PUBLIC 'CODE'
       ASSUME     CS:CODE, DS:DATA
START:      MOV   AX, DATA
            MOV   DS, AX
            MOV   BX, 4096                    ; 为父进程保留 4KB 空间
            MOV   AH, 4AH                     ; 修改内存分配块
```

```
        INT     21H                 ; 释放多余的空间
        JC      Error1              ; 修改失败，转
        MOV     AH，48H             ; 申请内存分配块
        MOV     BX，0FFFFH          ; 返回当前可用空间
        INT     21H
        CMP     BX，400             ; 能容纳下 CHKDSK 程序吗?
                                    ; CHKDSK 长度为 6KB（=400 个节）
        JB      Error2              ; 内存不够，转
        MOV     CS：Mem _size，BX   ; 保存当前可用空间
        PUSH    ES                  ; ES 进栈保护
        PUSH    DS                  ; DS 进栈保护
        ASSUME  ES：DATA            ; ES 段指向 DATA 数据段
        PUSH    DS
        POP     ES
        MOV     CS：Stk _seg，SS    ; 保存 SS
        MOV     CS：Stk _ptr，SP    ; 保存 SP
        MOV     DX，OFFSET Pgm _name ; DS：DX 指向被装入程序的字符串
        MOV     BX，OFFSET Par _blk ; ES：BX 指向参数块
        MOV     AL，0               ; 加载并执行程序
        MOV     AH，4BH             ; 调用 EXEC
        INT     21H
        MOV     SS，CS：Stk _seg    ; 恢复 SS
        MOV     SP，CS：Stk _ptr    ; 恢复 SP
        POP     DS                  ; 恢复 DS
        POP     ES                  ; 恢复 ES
        JNC     Exit                ; 加载成功，转
        JMP     Error2              ; 加载失败，转
Error1：MOV     DX，OFFSET ERR _MSG1 ; 指向显示字符串 1
        MOV     CX，MSG1 _LENGTH    ; 字符个数送 CX
        JMP     DISP
Error2：MOV     DX，OFFSET ERR _MSG2 ; 指向显示字符串 2
        MOV     CX，MSG2 _LENGTH    ; 字符个数送 CX
DISP：  MOV     BX，1               ; 句柄送 BX
        MOV     AH，40H             ; 写设备
        INT     21H
Exit：  MOV     AX，4C00H           ; 带返回码 00H 的退出
        INT     21H                 ; 返回 DOS
Stk _seg DW 0                       ; SS 保存单元
Stk _ptr DW 0                       ; SP 保存单元
```

```
Mem _ size    DW 0                          ; 存放当前可用节数
CODE          ENDS
              END     START
```

第四节　目录管理功能调用编程实例

【例 7.7】 利用扩展的 FCB 进行子目录改名

在 DOS 的内部命令中只有对子目录的建立、删除、指定等操作，没有对子目录名的修改操作，这往往使用户感到不便。本程序将利用 DOS 的扩展 FCB 格式，编写一个能对子目录名进行修改的程序。由于程序比较小，这里以．COM 文件的格式进行编写。

程序清单

```
CODE          SEGMENT
              ORG         0100H
              ASSUME      CS：CODE，DS：CODE，ES：CODE
START         PROC        NEAR
              JMP         SHORT BEGIN
ERRMSG1       DB '非法的参数！ $'
ERRMSG2       DB '不能进行改名！ $'
EXT _ FCB     DB 0FFH                       ; 扩展 FCB 标志字节
              DB 5 DUP (0)
              DB 10H                        ; 子目录属性
O _ NAME      DB 12 DUP (?)                 ; 源子目录名
              DB 05 DUP (0)
N _ NAME      DB 20 DUP (0)                 ; 新的子目录名
BEGIN：       MOV     SI，005CH
              MOV     DI，OFFSET O _ NAME
              MOV     CX，12
              CLD
              REPZ    MOVSB                 ; 传送源子目录名到 FCB
              MOV     SI，006DH
              MOV     DI，OFFSET N _ NAME
              CMP     BYTE PTR [SI]，20H    ; 判是否有新的子目录名
              JNZ     NEXT                  ; 有，转 NEXT
              MOV     AH，9
              MOV     DX，OFFSET ERRMSG1    ; 没有新子目录名，显示提示信息
              INT     21H
              JMP     SHORT EXIT
NEXT：        MOV     CX，11
```

```
        CLD
        REP     MOVSB
        MOV     DX, OFFSET EXT_FCB
        MOV     AH, 17H
        INT     21H                     ; 对子目录进行改名
        CMP     AL, 0
        JZ      EXIT                    ; 改名成功，转 EXIT
        MOV     AH, 9
        MOV     DX, OFFSET ERRMSG2
        INT     21H                     ; 显示改名出错提示信息
EXIT:   INT     20H                     ; 返回 DOS
START   ENDP
        CODE    ENDS
        END     START
```

【例 7.8】子目录维护

下面的程序利用 DOS 提供的对子目录操作的系统功能调用，编写了一个子目录维护程序，可以完成对子目录进行建立、指定和删除等操作。

程序清单

```
STACK   SEGMENT   PARA STACK 'STACK'
        DB        128 DUP (0)
STACK   ENDS
DATA    SEGMENT
MESS1   DB        13, 10,'---1=建立子目录    2=改变当前目录---'
        DB        13, 10,'---3=删除子目录    0=返回 DOS     ---'
        DB        13, 10,'请选择: $'
MESS2   DB        13, 10,'请输入子目录名: $'
MESS3   DB        13, 10,'子目录操作失败! $'
BUFF    DB        50,?, 50 DUP (0)
DATA    ENDS
CODE    SEGMENT
        ASSUME    CS: CODE, DS: DATA
START   PROC  FAR
        MOV   AX, DATA
        MOV   DS, AX
        MOV   DX, OFFSET MESS1
        MOV   AH, 9
        INT   21H
```

```
RETRY:  MOV     AH, 1
        INT     21H
        SUB     AL, 30H
        JZ      EXIT
        CMP     AL, 1
        JB      RETRY
        CMP     AL, 3
        JA      RETRY
        PUSH    AX
        MOV     DX, OFFSET MESS2
        MOV     AH, 9
        INT     21H
        MOV     DX, OFFSET BUFF
        MOV     AH, 0AH
        INT     21H
        MOV     BL, BUFF+1
        MOV     BH, 0
        MOV     SI, OFFSET BUFF+2
        MOV     BYTE PTR [SI+BX], 0
        POP     AX
        CMP     AL, 1
        JNE     NEXT1
        MOV     AH, 39H
        CALL    PUBLIC _ PROC
        JMP     NEXT3
NEXT1:  CMP     AL, 2
        JNE     NEXT2
        MOV     AH, 3BH
        CALL    PUBLIC _ PROC
        JMP     NEXT3
NEXT2:  MOV     AH, 3AH
        CALL    PUBLIC _ PROC
NEXT3:  JNC     EXIT
        MOV     DX, OFFSET MESS3
        MOV     AH, 9
        INT     21H
EXIT:   MOV     AH, 4CH
        INT     21H
START   ENDP
```

```
PUBLIC _ PROC   PROC    NEAR
                MOV     DX，OFFSET BUFF+2
                INT     21H
                RET
PUBLIC _ PROC   ENDP
CODE            ENDS
                END     START
```

第五节　中断管理及其它系统功能调用编程实例

【例 7.9】 实时时钟显示程序

下面的程序能够在不具有实时时钟的微机上，显示实时的时间。

编程指导：

本程序采用了修改中断 1CH 驻留内存的方法来显示实时的时间，修改中断利用了系统功能调用 25H，驻留内存利用了系统功能调用 31H。INT 1CH 是一个特殊的中断，系统开机以后，中断 1CH 不断地被系统以每秒大约 18. 2 次的速度来执行，原有的中断服务程序仅有一个 IRET 指令，即不执行任何操作即返回。修改以后的 1CH 中断每次被系统执行时，根据累计的计数值自动计算当前的时间，并显示出来。由于 1CH 中断是由系统自动执行的，它独立于应用程序，不影响应用程序的正常执行。

程序清单

```
; ————————————————————————————————————————
;                信息保护课程设计实验报告                              ;
;                ==========                                            ;
;              课程设计名称：实时时钟显示程序                          ;
;                  设计者：饶太新    李兴凯        日期：05/26/92     ;
; ————————————————————————————————————————
CODE        SEGMENT
            ASSUME              CS：CODE，DS：CODE
TIMES       DB    '00：00：00. 00'    ；时间显示格式
DATA        DB    90                  ；百分秒计数值定义
SR          DB    0                   ；秒值修正计数
RL          DB    0                   ；百分秒计数值
RH          DB    0                   ；秒计数值
RD          DB    0                   ；分钟计数值
HO          DB    0                   ；小时计数值
A _ ES      DW    0B800h              ；显示区内存地址
ATTRI       DB    7                   ；显示字符属性
TAI         DW    16 DUP (0)
INT1CH      PROC    NEAR
```

```
        MOV     cs: tai, ds
        MOV     cs: tai+2, es
        MOV     cs: tai+4, di
        MOV     cs: tai+6, si
        MOV     cs: tai+8, ax
        MOV     cs: tai+10, bx
        MOV     cs: tai+12, cx
        MOV     cs: tai+14, dx      ; 以上为现场保护
        MOV     AX, CS
        MOV     DS, AX
        MOV     CL, DATA
        CMP     RL, CL              ; 百分秒计数校验
        JA      NEXT1               ; 如果大于 DATA 值转秒值校验
        ADD     RL, 5
        JMP     EXIT
NEXT1:  MOV     RL, 0
        ADD     RH, 1
        CMP     SR, 5               ; 秒值校正
        JA      RAO1                ; 如果大于 5，便将 DATA 值置为 95
        MOV     DATA, 90
        ADD     SR, 1
        JMP     RAO2
RAO1:   MOV     SR, 0
        MOV     DATA, 95
RAO2:   CMP     RH, 60              ; 秒值计数校验
        JAE     NEXT2
        JMP     EXIT
NEXT2:  ADD     RD, 1
        MOV     RH, 0
        CMP     RD, 60              ; 分计数校验
        JAE     NEXT3
        JMP     EXIT
NEXT3:  MOV     RD, 0
        ADD     HO, 1               ; 小时计数
; ------------------------------------------------------------
;       显示时间调整:
;           将时间计数各值转为 ASCII 码
; ------------------------------------------------------------
EXIT:   MOV     BL, HO
```

```
            CALL    B_ASC
            MOV     WORD PTR TIMES, CX
            MOV     BL, RD
            CALL    B_ASC
            MOV     WORD PTR TIMES+3, CX
            MOV     BL, RH
            CALL    B_ASC
            MOV     WORD PTR TIMES+6, CX
            MOV     BL, RL
            CALL    B_ASC
;-----------------------------------------------
;       将可显示的 ASCII 码写入 CRT 显示区中
;-----------------------------------------------
            MOV     WORD PTR TIMES+9, CX
            LEA     SI, TIMES
            MOV     DI, 136
            MOV     AX, A_ES
            MOV     ES, AX
            MOV     AH, ATTRI
            MOV     CX, 11
RTX:        LODSB
            STOSW
            LOOP    RTX
;-----------------------------------------------
;             中断现场恢复，并退出
;-----------------------------------------------
            MOV     DS, cs: tai
            MOV     ES, cs: tai+2
            MOV     DI, cs: tai+4
            MOV     SI, cs: tai+6
            MOV     AX, cs: tai+8
            MOV     BX, cs: tai+10
            MOV     CX, cs: tai+12
            MOV     DX, cs: tai+14
            IRET
INT1CH      ENDP
;-----------------------------------------------
;       子过程：
;             将输入的十进制数转为 ASCII 码
```

```
;-------------------------------------------------
B_ASC   PROC  NEAR
        MOV   CX, 0
BEGIN:  CMP   BL, 10
        JB    STOP1
        SUB   BL, 10
        ADD   CL, 1
        JMP   BEGIN
STOP1:  MOV   CH, BL
        ADD   CX, 3030H
        RET
B_ASC   ENDP
;-------------------------------------------------
;       主过程:
;           完成将 INT 1CH 的入口变为本程序所设的入
;           口，并实现驻留．其 中设置显示值的起始值
;-------------------------------------------------
START   PROC  FAR
        MOV   AX, CS
        MOV   DS, AX
        MOV   AH, 2CH
        INT   21H                ; 取系统当前的时间
        MOV   HO, CH
        MOV   RD, CL
        MOV   RH, DH
        MOV   RL, DL
        LEA   DX, INT 1CH
        MOV   AX, 25 1CH         ; 修改 INT 1CH
        INT   21H
        MOV   DX, OFFSET START
        MOV   CL, 4
        SHR   DX, CL
        MOV   AX, CS
        ADD   DX, AX
        MOV   AX, 3100H          ; 驻留内存
        INT   21H
START   ENDP
CODE    ENDS
        END   START
```

第八章　I/O 端口及其编程实例

I/O 端口是 CPU 与计算机外设之间进行信息交换的媒介，8088 CPU 用 IN 和 OUT 指令进行 CPU 与外设之间的信息交换。ROM BIOS 中断功能中的大多数子程序都是通过对端口的读写操作来完成对计算机 I/O 设备的管理。在汇编语言程序设计中，由于实际问题的需要，用户程序也可以直接对 I/O 端口进行编程，以便完成对 I/O 设备进行特殊的控制和管理工作。

第一节　常用的 I/O 端口

一、8259 中断控制寄存器端口

（一）20H 口：中断命令寄存器

在每次外部中断结束以后，要给该口发送中断结束信号，其指令为：

```
MOV    AL，20H
OUT    20H，AL
```

（二）21H 口：中断屏蔽寄存器

8259 芯片可以接收 8 个外部可屏蔽中断。这些中断有两种状态，即允许和屏蔽状态。21H 口的 8 位分别对应于 8 个外部中断源，某位为 0，则允许接收相应的中断，否则不接收相应的中断。其对应关系如图 8－1 所示。

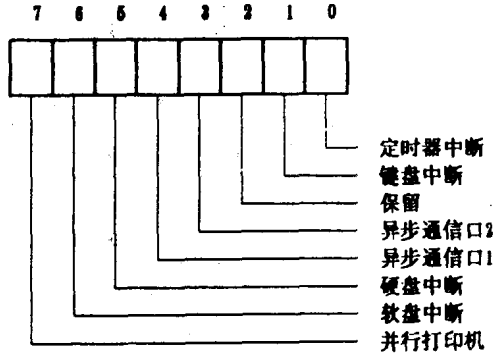


图 8－1　中断屏蔽寄存器位定义

二、8253 定时器端口

8253 共有三个独立的通道寄存器，其口地址分别为 40H、41H 和 42H，另外还有一个命令寄存器 43H。每个通道都可以单独编程，编程时先给命令寄存器送一个通道编程方式字，再给相应通道的口地址送计数值。命令寄存器的各位含义如图 8－2 所示。

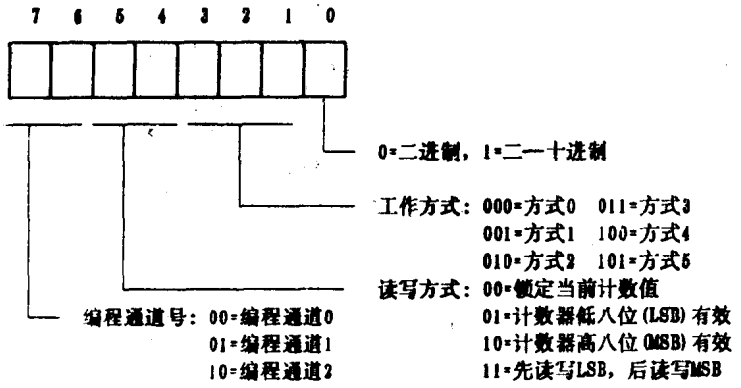


图 8－2　8253 命令寄存器位定义

三、8255 可编程外围接口

8255 是通用的 I/O 接口芯片，有很多配置方法。它可以支持许多设备和信号，包括键盘、扬声器、配置开关等。这个芯片包括三个口，分别叫做 PA 口、PB 口和 PC 口，三个口分配的口地址分别为 60H、61H 和 62H。另外，在这个芯片上还有一个命令寄存器，口地址为 63H，对于三个口的编程必须通过设置命令寄存器来进行。系统在启动时向 63H 口发送 99H，把该芯片设置成 PA 和 PC 为输入口，PB 为输出口。63H 口的各位含义如图 8－3 所示。

PA 口有两种用途，一是当 PB 口的位 7＝0 时，PA 口返回的是键盘扫描码；二是当 PB 口的位 7＝1 时，PA 口返回系统板上的设备配置情况。PA 口、PB 口和 PC 口的各位含义分别如图 8－4、图 8－5 和图 8－6 所示。

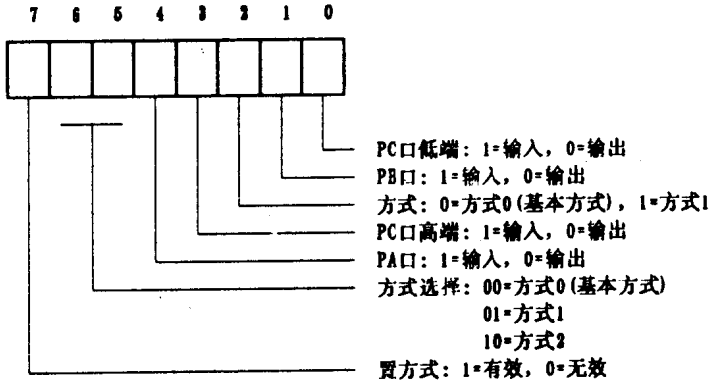


图 8－3　8255 命令寄存器（63H 口）位定义

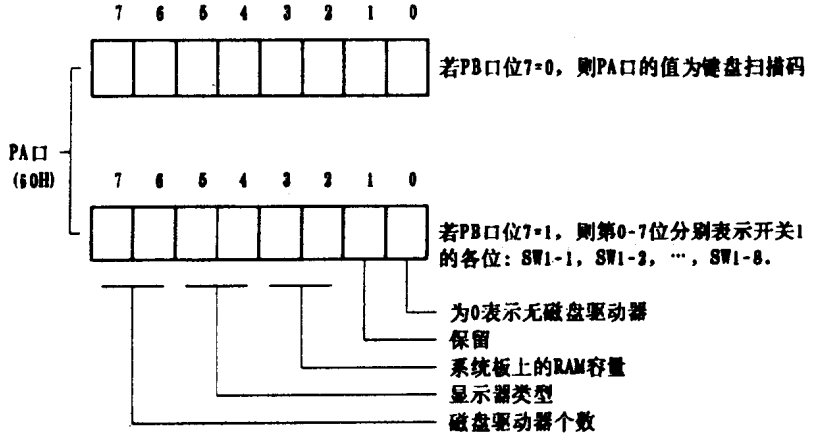


图 8－4　8255 PA 口的位定义

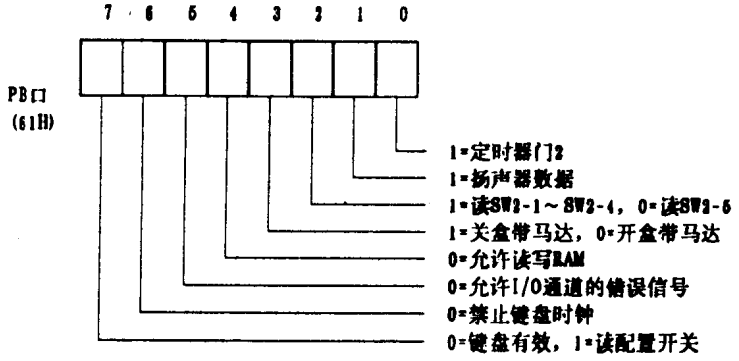


图 8－5　8255 PB 口位定义

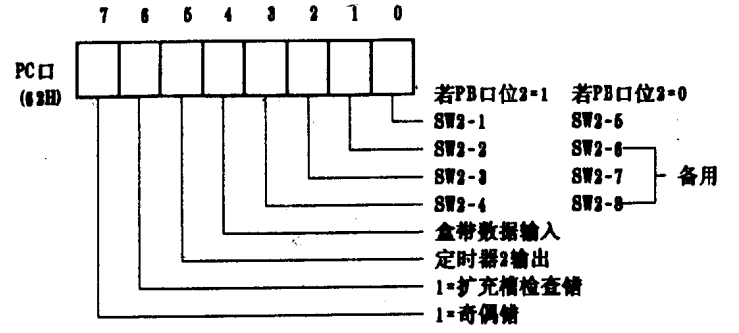


图 8－6　8255 PC 口的位定义

四、6845 CRT 寄存器端口

在 IBM PC 系统中，彩色显示器控制器采用 Motorola 公司的 6845 芯片，该芯片有一组寄存器，其相应访问口地址如表 8－1 和表 8－2 所示。

表 8－1　6845 口地址说明

口地址	寄存器名称
3D4H	索引寄存器
3D5H	数据寄存器
3D8H	方式控制寄存器
3D9H	颜色选择寄存器
3DAH	状态寄存器

表 8－2　数据寄存器（3D5H）说明

寄存器地址	寄存器号	功　能
0～3	R0～R3	水平特性参数寄存器
4～9	R4～R9	垂直特性参数寄存器
A	R10	光标起始行寄存器
B	R11	光标终止行寄存器
C	R12	起始地址（高）寄存器
D	R13	起始地址（低）寄存器

在系统起动时对该芯片进行了初始化。但是，在具体应用中如有特殊需要可以重新对其进行参数设置。参数设置的方法是：先通过地址索引寄存器（3D4H 口）送出将要进行参数设置的数据寄存器的相应地址，以指向相应的寄存器，然后从 3D5H 口送出相应数据寄存器的参数。主要寄存器的各位定义如图 8－7、图 8－8 和图 8－9 所示。

五、打印机端口

打印机有三个端口，它们分别是输出数据端口 378H、输入状态端口 379H 和输出控制端口 37AH。输出数据端口是字符编码送出的端口，输入状态端口是反映打印机当前状态的端口，而输出控制端口则是进行打印控制的端口。图 8－10 和图 8－11 分别是输出控制和输入状

态端口的位定义。

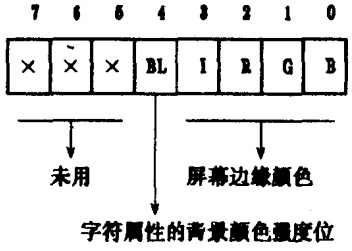


图 8－7　选色寄存器 3D9H

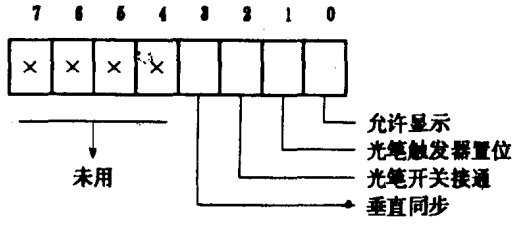


图 8－8　状态寄存器 3DAH

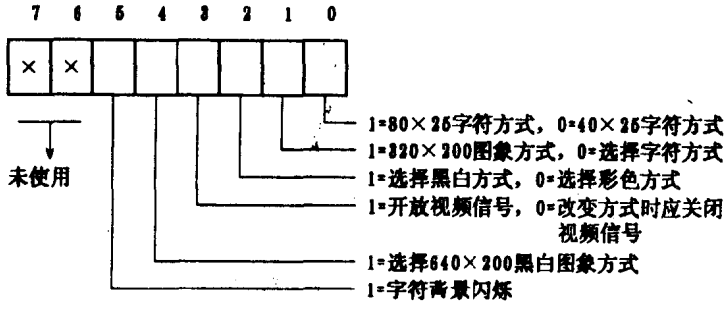


图 8－9　方式选择寄存器 3D8H

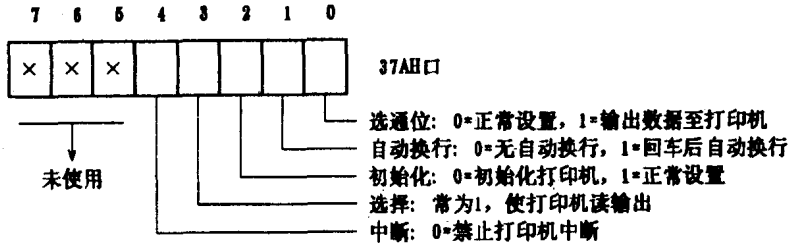


图 8－10　打印机输出控制口的位定义

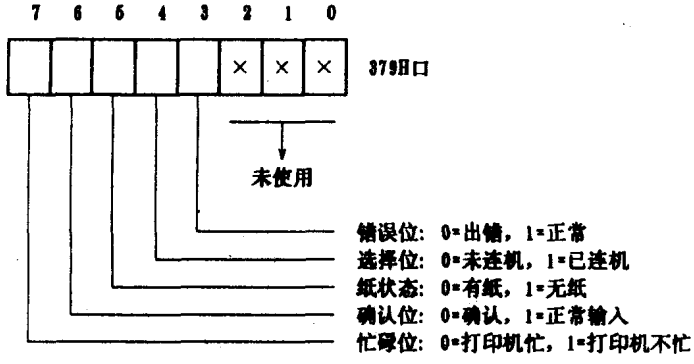


图 8－11　打印机输入状态口的位定义

第二节　8255 端口编程实例

【例 8.1】 检测并报告微机中的驱动器个数

下面的程序能根据 8255 端口芯片中 DIP 开关的设置，检测并报告微机中的驱动器个数。

程序清单

```
CSEG        SEGMENT
            ORG         100H
            ASSUME      CS：CSEG，DS：CSEG
START：     JMP         SHORT BEGIN
            MESSAGE1    DB 13，10，'DIP 开关被设置为：$ '
MESSAGE2                DB '个磁盘驱动器！$'
BEGIN：     MOV     AH，9
            MOV     DX，OFFSET MESSAGE1
            INT     21H                     ；显示提示信息
；从 8255 芯片的端口 A 取 DIP 开关设置
            IN      AL，61H                 ；从 PB 口取一字节数据
            OR      AL，10000000B           ；强置位 7
            OUT     61H，AL                 ；PB 口位 7 置 1，为 PA 口开门
            IN      AL，60H                 ；从 PA 口取开关设置
            AND     AL，11000000B           ；保留最高两位（驱动器号）
            MOV     CL，6
            SHR     AL，CL                  ；将最高两位移到寄存器底
            XOR     AL，00000011B           ；将最低两位求反，得驱动器个数于 AL 中
            OR      AL，30H                 ；二进制数转 ASCII 码
            MOV     DL，AL
            IN      AL，61H
            AND     AL，01111111B           ；位 7 强置为 0
            OUT     61H，AL                 ；字节代换
            MOV     AH，2
            INT     21H                     ；显示驱动器个数
            MOV     AH，9
            MOV     DX，OFFSET MESSAGE2
            INT     21H                     ；显示提示信息 2
            INT     20H                     ；程序结束
CSEG        ENDS
            END     START
```

【例 8.2】 使计算机发声

下面的一段程序是通过编程 8253 端口使计算机发声的程序片段。

编程指导：

使计算机发声的最简单的方法是利用系统功能调用显示 07 字符，从表面上看好象是显示一个字符，实际上该字符不显示在屏幕上，而是使计算机发出“笛—”声。使计算机发声的另外一种方法是，编程 8255 外围接口芯片使扬声器发出“嘟嘟”声。

程序清单

```
        MOV     DX，10          ；循环计数值
        IN      AL，61H         ；读 8255 PB 口
        AND     AL，0FEH        ；8253 定时器位置 0
NEXT1:  OR      AL，2           ；置扬声器位为 1
        OUT     61H，AL         ；写 PB 口
        MOV     CX，140
RETRY:  LOOP    RETRY           ；延时
        AND     AL，0FDH        ；置扬声器位为 0
        OUT     61H，AL         ；写 PB 口
NEXT2:  LOOP    NEXT2
        DEC     DX
        JNZ     NEXT1
```

第三节　6845 CRT 端口编程实例

【例 8.3】 屏幕滚动

下面的程序通过直接对显示器 I/O 端口进行操作，以及直接写屏幕缓冲区的技术，在屏幕的中间纵向顺次显示 ASCII 码 0～127 的字符，每行以白底红字兰边框显示一个字符。字符不断地上滚，约 0.5 秒滚动一行，当最后一个字符滚到屏幕的中间后改为反向滚动。在滚动过程中键入回车键，程序将结束运行返回 DOS。

编程指导：

下面的程序通过写颜色寄存器 3D9H 来设置屏幕的边缘颜色，通过改变起始地址寄存器的值来使屏幕滚动。

程序清单

```
STACK   SEGMENT   PARA STACK ' STACK'
        DB        256 DUP (0)
STACK   ENDS
DATA    SEGMENT
COUNT   DB 0
BASE    DW 0                    ；开始地址
```

```
DIRECTN   DB 0                    ; 滚动方向标志单元（0——上滚，1——下滚）
KMODE     DB ?                    ; 显示器工作方式暂存单元
DATA      ENDS
CODE      SEGMENT
          ASSUME    CS: CODE, DS: DATA
START     PROC   FAR
          PUSH   DS
          MOV    AX, 0
          PUSH   AX
          MOV    AX, DATA
          MOV    DS, AX
;
          MOV    AX, 0040H
          MOV    ES, AX
          MOV    AL, ES: [0049H]
          MOV    KMODE, AL          ; 保存系统显示方式
;
          MOV    AX, 0001           ; 设置 40×25 彩色文本显示方式
          INT    10H
          MOV    DX, 3D9H           ; 颜色选择寄存器
          MOV    AL, 01
          OUT    DX, AL             ; 设置边缘颜色为蓝色
          MOV    AX, 0B800H
          MOV    ES, AX             ; 屏幕缓冲区段地址→ES
          MOV    DI, 0              ; 开始地址→SI
          MOV    AL, 20H            ; 字符 20H→AL
          MOV    AH, 74H            ; 白底红字属性
          MOV    CX, 8000
          CLD
          REP    STOSW              ; 屏幕初始化
; 以 0 到 127 的字符填写每行第 40 列的字符
          MOV    AL, 0              ; ASCII 码 0→AL
          MOV    DI, 40             ; 第一个字符的显示位置
FILL:     MOV    ES: [DI], AL
          ADD    DI, 80             ; 地址加 80，指向下一行
          INC    AL                 ; 形成下一个 ASCII 码
          CMP    AL, 128
          JB     FILL
; 延时 0.5s 子程序
```

```
DELAY:  MOV   AL, 50
LOOP1:  MOV   CX, 2801
LOOP2:  LOOP  LOOP2
        DEC   AL
        CMP   AL, 0
        JNZ   LOOP1
;
        MOV   AH, 1
        INT   16H
        JZ    SCROLL          ; 判是否有键盘输入? 没有输入, 转
;
        MOV   AH, 0
        INT   16H
        CMP   AL, 0DH         ; 键盘有输入, 判是否为回车符
        JE    EXIT            ; 是回车符, 转 EXIT
SCROLL: MOV   BX, BASE
        CMP   DIRECTN, 0      ; 滚动方向为向上吗?
        JNE   BACK            ; 不是, 转
        INC   COUNT           ; 滚动行计数加 1
        CMP   COUNT, 116      ; 是否已达到滚动的最大行数
        JB    GOON1           ; 否, 转
        MOV   DIRECTN, 1      ; 改变滚动方向为向下
        JMP   BACK
GOON1:  ADD   BX, 40          ; 地址新值
        MOV   BASE, BX
        JMP   UPDATE          ; 转硬件滚动
BACK:   DEC   COUNT           ; 计数减 1
        CMP   COUNT, 1
        JNE   GOON2
        MOV   DIRECTN, 0      ; 计数减为 1, 重新设置向上滚动
;
GOON2:  SUB   BX, 40
        MOV   BASE, BX
;
UPDATE: MOV   DX, 3D4H        ; CRT 索引寄存器
        MOV   AL, 12          ; 选择起始地址 (高) 寄存器
        OUT   DX, AL
        MOV   DX, 3D5H
        MOV   AL, BH
```

```
        OUT     DX, AL          ; 写起始地址高字节
        MOV     DX, 3D4H
        MOV     AL, 13          ; 选择起始地址（低）寄存器
        OUT     DX, AL
        MOV     DX, 3D5H
        MOV     AL, BL
        OUT     DX, AL          ; 写起始地址低字节
        JMP     DELAY
;
EXIT:   MOV     AL, KMODE
        MOV     AH, 0
        INT     10H             ; 恢复显示方式
        RET                     ; 返回 DOS
START   ENDP
CODE    ENDS
        END     START
```

第四节　打印机端口编程实例

【例 8.4】 打印一个字符

下面的程序实现打印 AL 中的一个字符，在打印之前首先检查打印是否忙碌，只有在打印机不忙的状态下，才选通打印机进行打印。程序设计中用到了 3 个 I/O 端口地址，它们分别是打印机“输出数据”端口地址、“输入状态”端口地址和“输出控制”端口地址。

程序清单

```
; 入口参数：AL=被打印的字符
PRINT_CHAR   PROC   NEAR
        PUSH    AX
        PUSH    DX
        MOV     DX, 378H        ; “输出数据”口地址→DX
        OUT     DX, AL          ; 输出要打印的字符
        MOV     DX, 379H        ; “输入状态”口地址→DX
WAIT:   IN      AL, DX          ; 读打印机状态
        TEST    AL, 80H         ; 检查忙碌位
        JE      WAIT            ; 打印机忙，转 WAIT
        MOV     DX, 37AH        ; “输出控制”口地址→DX
        MOV     AL, 0DH         ; 置选通位为“1”
        OUT     DX, AL          ; 选通打印机
        MOV     AL, 0CH         ; 置选通位为“0”
```

```
        OUT     DX, AL                  ; 关打印机
        POP     DX
        POP     AX
        RET
PRINT_CHAR  ENDP
```

第五节　计算机发声及音乐演奏编程实例

【例 8.5】 通用发声子程序

下面的子程序是一个通用的发声子程序，它可以产生从 19Hz 到 65535Hz 之间的任一频率的声音。

编程指导：

在 IBM PC 系统中，声音是由定时器 8253 和并行接口芯片 8255 来控制的。在 8255 中通过置位 PB 口（61H）的第 0 位、第 1 位可以启动扬声器发声，通过复位此两位可以关闭扬声器的发声。扬声器发声的频率则要写入 8253 的通道 2（42H 口）。需要注意的是，在进行 8253 通道 2 编程之前，必须设定 8253 的命令寄存器（43H 口）以适当的定时器工作模式。

程序清单

```
; SOUND 子程序调用参数
;       入口参数：DI=频率（19～65535Hz）
;                 BX=发声的长短（单位为百分之一秒）
;
SOUND   PROC    NEAR
        PUSH    AX
        PUSH    BX
        PUSH    CX
        PUSH    DX
        MOV     AL, 0B6H
        OUT     43H, AL         ; 写定时器模式寄存器
        MOV     DX, 12H
        MOV     AX, 34DCH
        DIV     DI
        OUT     42H, AL         ; 写定时器 2 计数的低位
        MOV     AL, AH
        OUT     42H, AL         ; 写定时器 2 计数的高位
        IN      AL, 61H
        MOV     AH, AL          ; 保存 PB 口的当前值
        OR      AL, 3
        OUT     61H, AL         ; 启动扬声器发声
```

```
DELAY:    MOV    CX, 2801
DL10MS:   LOOP   DL10MS        ; 延时 10ms
          DEC    BX
          JNZ    DELAY         ; 延时 BX * 10ms
          MOV    AL, AH
          OUT    61H, AL       ; 关闭扬声器
          POP    DX
          POP    CX
          POP    BX
          POP    AX
          RET
SOUND     ENDP
```

【例 8.6】演奏一首乐曲

下面的程序通过调用发声子程序来演奏《蜗牛和黄鹂鸟》歌曲，在演奏过程中随时只要按下任意键，程序即停止运行返回 DOS。

编程指导：

利用上述的 SOUND 子程序可以编写演奏歌曲的程序。要编写这样的程序，首先需要把歌曲的简谱一对一地翻译为相应的频率值，而且还要建立每个音符的演奏时间表。表 8-3 中列出了部分常用的音符及其对应的频率值。《蜗牛和黄鹂鸟》歌谱翻译出来的频率及其演奏时间表见程序中的数据段。

表 8-3　部分音符及其频率对照表

音名	g	a	b	1 c	1 d	1 e	1 f	1 g	1 a	1 b	2 c	2 d	2 e
唱名	5	6	7	1	2	3	4	5	6	7	1	2	3
频率	196	220	247	262	294	330	349	392	440	494	523	287	659

《蜗牛和黄鹂鸟》歌谱

$1=E\ \frac{2}{4}$　　台湾校园歌曲

555 5 35 | 1 6 5 | 555 5 32 | 1 3 2 |
2. 3 5 55 | 3 32 1 1 | 2. 3 1 16 | 5 6 5 |
555 5 35 | 1 6 5 | 555 5 32 | 1 3 2 |
2. 3 5 5 | 3 32 1 1 | 2. 3 1 16 | 5 6 5 |
555 5 32 | 1 6 $5^{\vee}$ 56 | 1 2 1 2 | 3 2 |
1 —— | 1 —— ‖

编写歌曲程序时要注意以下几点：

1. 频率的规定

(1) 最低频率不能小于 19Hz，否则在 SOUND 子程序做除法时会产生溢出；

(2) 把歌谱中休止符的频率可以规定为 20Hz，程序在演奏休止符时只延时、不发声；

(3) 歌谱中若有连续的相同音符时，为了使发声效果有变化，可以把其中相间隔的音符的频率做适当的调整，即适当地改变频率值；

(4) 歌曲的结束规定用频率“0”作标志。

2. 演奏速度的规定

演奏的速度一般以执行 10ms 延时的次数为基准。以执行四分音符为单位，速度与执行次数的对应关系如表 8－4 所示。

表 8－4

速　度	慢速	较慢	中速	较快	快速
执行次数	250	200	100	50	25

程序清单

```
STACK    SEGMENT  PARA STACK 'STACK'
         DW 256 DUP (?)
STACK    ENDS
DATA     SEGMENT
         ; 歌曲频率表
FREQ     DW 1 DUP (392, 396, 392, 395, 330, 392, 262, 440, 392, 395)
         DW 1 DUP (392, 395, 392, 330, 294, 262, 330, 294, 297, 330)
         DW 1 DUP (392, 395, 392, 330, 335, 294, 262, 265, 294, 330)
         DW 1 DUP (262, 265, 220, 196, 220, 196, 392, 395, 392, 395)
         DW 1 DUP (330, 392, 262, 440, 392, 392, 395, 392, 395, 330)
         DW 1 DUP (294, 262, 330, 294, 294, 330, 392, 395, 330, 335)
         DW 1 DUP (294, 262, 265, 294, 330, 262, 265, 220, 196, 220)
         DW 1 DUP (196, 392, 395, 392, 395, 330, 294, 262, 440, 392)
         DW 1 DUP (395, 440, 262, 294, 262, 294, 330, 294, 262)
         DW 1 DUP (0)
         ; 持续时间表
TIME     DW 1 DUP (25, 50, 25, 50, 25, 25, 50, 50, 100, 25)
         DW 1 DUP (50, 25, 50, 25, 25, 50, 50, 100, 75, 25)
         DW 1 DUP (50, 25, 25, 50, 25, 25, 50, 50, 75, 25)
         DW 1 DUP (50, 25, 25, 50, 50, 100, 25, 50, 25, 50)
         DW 1 DUP (25, 25, 50, 50, 100, 25, 50, 25, 50, 25)
         DW 1 DUP (25, 50, 50, 100, 75, 25, 50, 50, 50, 25)
         DW 1 DUP (25, 50, 50, 75, 25, 50, 25, 25, 50, 50)
         DW 1 DUP (100, 25, 50, 25, 50, 25, 25, 50, 50, 50)
         DW 1 DUP (25, 25, 50, 50, 50, 50, 100, 100, 200)
DATA     ENDS
```

```
CODE        SEGMENT
            ASSUME  CS: CODE, DS: DATA
START:      MOV     AX, DATA
            MOV     DS, AX
            MOV     SI, OFFSET FREQ
            MOV     BP, OFFSET TIME
            CALL    PLAY
            MOV     AH, 4CH
            INT     21H
;
PLAY        PROC    NEAR
            PUSH    DI
            PUSH    SI
            PUSH    BP
            PUSH    BX
REPT0:      MOV     DI, [SI]            ; 取下一音符的频率值
            CMP     DI, 0
            JE      END_PLAY            ; 频率值为结束标志，转结束处理
            MOV     AH, 1
            INT     16H                 ; 判键盘是否有按键
            JNZ     END_PLAY            ; 键盘有按键，转结束处理
            MOV     BX, DS: [BP]        ; 取当前音符的演奏时间→BX
            CMP     DI, 20              ; 判当前音符是否为休止符
            JNE     REPT3               ; 不是休止符，转 REPT3
REPT1:      MOV     CX, 2801
REPT2:      LOOP    REPT2               ; 是休止符，执行延时、不开喇叭
            DEC     BX
            JNZ     REPT1
            JMP     REPT4
REPT3:      CALL    SOUND               ; 调用发声子程序
REPT4:      ADD     SI, 2
            ADD     BP, 2
            JMP     REPT0
END_PLAY:   POP     BX
            POP     BP
            POP     SI
            POP     DI
            RET
PLAY        ENDP
```

```
SOUND     PROC   NEAR
          ......
SOUND     ENDP
CODE      ENDS
          END    START
```

【例 8.7】用键盘弹奏乐曲

下面的程序在运行时，您可以按键盘上的数字键：1～8 弹出 8 个音符 1（刀），2（来），3（米），……。从而可以用此程序弹奏出简单的歌曲。

编程指导：

程序设计中，根据所按的键值，转换为其对应音的频率值，通过设置 8255 的 PB 口启动扬声器，然后再设置 8253 的命令寄存器，以便初始化通道 2 和使 8253 以模式 3 进行工作，最后把频率值送 8253 的通道 2 使扬声器发出声音。

程序清单

```
CODE          SEGMENT
              ASSUME  CS: CODE
MAIN          PROC    FAR
READ_KEY:     MOV     AH, 7
              INT     21H
              CMP     AL, 03              ; 是 Ctrl+C 键，转出口
              JZ      EXIT
              CMP     AL, 31H
              JB      READ_KEY
              CMP     AL, 38H
              JA      READ_KEY
              SUB     AL, 31H
              SHL     AL, 1
              CBW
              MOV     BX, AX
              MOV     AX, 34DCH
              MOV     DX, 12H
              DIV     CS: [BX+FRIQ]
              MOV     BX, AX
              MOV     AL, 0B6H
              OUT     43H, AL             ; 初始化通道 2、置模式 3 工作方式
              MOV     AX, BX
              OUT     42H, AL             ; 输出频率低字节
              MOV     AL, AH
```

```
                OUT     42H, AL             ; 输出频率高字节
                IN      AL, 61H
                OR      AL, 3
                OUT     61H, AL             ; 启动扬声器
                MOV     CX, 0FFFFH
DELAY:          LOOP    DELAY
                IN      AL, 61H
                AND     AL, 11111100B
                OUT     61H, AL             ; 关闭扬声器
                JMP     READ_KEY
EXIT:           MOV     AH, 4CH
                INT     21H
FRIQ    DW      262,    294,    330,    349,    392,    440,    494,    523
        ; 音调表 C      D       E       F       G       A       B       高音C
MAIN            ENDP
CODE            ENDS
                END     MAIN
```

【例 8.8】 在其它操作的同时进行演奏

下面的程序是一个演奏与显示操作同时进行的程序实例。程序运行时，一边在屏幕的中央以不同的颜色循环显示“Love me again, Mother !”，一边演奏《妈妈再爱我一次》歌曲。如果在演奏过程中按任意键，程序将终止运行、返回 DOS。否则，一直循环显示和演奏。

编程指导：

在许多程序设计中，都需要把演奏和其它操作同时进行。那么，如何才能实现这一功能呢？我们知道，在 IBM PC 机系统中，定时器中断 1CH 是独立于 CPU 而工作的。这一中断在系统初始化后，中断程序仅仅有一条 IRET 指令，也就是说该中断不起任何作用。但是它是每秒发生 18.2 次的硬件中断。我们可以改变这一中断服务程序，使其指向我们自编的进程。这样，我们的进程就可以独立于 CPU 以每秒 18.2 次的速度去工作。在下面的程序设计中，我们把演奏歌曲的程序指向 1CH 中断向量，每当发生 1CH 中断后，该中断例程取出当前时钟的计数值，与当前所演奏的音符的计数值进行比较，如果两者不等，则中断返回；否则，重新设置下一音符的频率和演奏时间，然后再中断返回。设置音符演奏时间的方法是，把当前时钟计数值加上音符的基本计数并送 END_NOTE 单元。

程序清单

```
STACK           SEGMENT  PARA STACK ''STACK'
                DB  256  DUP (0)
STACK           ENDS
DATA            SEGMENT
ATTRI           DB  1
```

```
MSG1    DB  'Love me again, Mother!'
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, DS: DATA
START:  MOV     AX, SEG NEW_1CH
        MOV     DS, AX
        MOV     DX, OFFSET NEW_1CH
        MOV     AX, 251CH
        INT     21H                     ; 设置 1CH 中断，指向演奏进程
        MOV     AX, DATA
        MOV     DS, AX
;
        MOV     AX, 0003
        INT     10H                     ; 设置屏幕显示方式
        CLD
        MOV     BL, ATTRI               ; 字符颜色值→BL
AGAIN:  MOV     SI, OFFSET MSG1         ; 字符串首地址→SI
        INC     BL                      ; 改变当前颜色
        AND     BL, 0FH
        MOV     DX, 0A20H               ; 光标初始位置 (10, 32)
RETRY:  MOV     AH, 2
        MOV     BH, 0
        INT     10H                     ; 光标定位
        LODSB
        MOV     AH, 9
        MOV     BH, 0
        MOV     CX, 1
        PUSH    AX
        INT     10H                     ; 显示一个字符
        POP     AX
        CMP     AL,'!'
        JE      AGAIN                   ; 已显示到字符，转 AGAIN
        MOV     CX, 0FFFFH
DELAY:  LOOP    DELAY                   ; 延时
        MOV     AH, 1
        INT     16H                     ; 键盘按键测试
        JNZ     EXIT                    ; 有按键，转出口
        INC     DL                      ; 光标列坐标加 1
        JMP     SHORT RETRY             ; 继续循环
```

```
EXIT:       MOV     DX, 0FF53H
            MOV     AX, 0F000H
            MOV     DS, AX
            MOV     AX, 251CH
            INT     21H                 ; 恢复 1CH 中断向量
            IN      AL, 61H
            AND     AL, 0FCH
            OUT     61H, AL             ; 关闭扬声器
            MOV     AH, 4CH
            INT     21H                 ; 主程序返回
CODE        ENDS
CODE1       SEGMENT
            ASSUME  CS: CODE1, DS: CODE1
NEW_1CH     PROC    FAR
            PUSH    AX
            PUSH    BX
            PUSH    CX
            PUSH    DX
            PUSH    DI
            PUSH    SI
            PUSH    DS
            PUSH    CS
            POP     DS
            CMP     SOUND_NOW, 1        ; 需要演奏吗
            JE      PLAY_IT             ; 是，开始演奏
            JMP     NOT_NOW             ; 否则，中断返回
PLAY_IT:    CMP     FIRST_NOTE, 1       ; 是演奏第一个音符吗
            JNE     TIME_CHECK          ; 不是，转
            IN      AL, 61H
            OR      AL, 3
            OUT     61H, AL             ; 启动扬声器和定时器通道 2
            MOV     AL, 0B6H
            OUT     43H, AL             ; 通道 2 初始化，设置为模式 3
            MOV     FIRST_NOTE, 0       ; 置演奏开始标志
NEXT_NOTE:  LEA     BX, FREQ
            MOV     SI, WHICH_NOTE      ; 当前音符在音符表中的偏移指针→SI
            MOV     DI, [BX][SI]        ; 取音符的频率值
            CMP     DI, 0
            JE      NO_MORE             ; 是音符结束标志，转 NO_MORE
```

```
                MOV     DX, 12H
                MOV     AX, 34DCH
                DIV     DI
                OUT     42H, AL
                MOV     AL, AH
                OUT     42H, AL
TIME_IT:        MOV     AH, 0
                INT     1AH                     ; 读取当前时钟的计数值→DX（低位）
                MOV     BX, OFFSET DELAY_TAB
                MOV     AX, [BX] [SI]           ; 取当前音符的演奏时间基值→AX
                DIV     FM
                XOR     AH, AH
                ADD     AX, DX                  ; 基值＋当前时钟计数→AX
                MOV     END_NOTE, AX            ; 存尾音符的值
TIME_CHECK:     MOV     AH, 0
                INT     1AH
                CMP     DX, END_NOTE
                JNE     NOT_NOW                 ; 当前音符延时未完，转
                MOV     SI, WHICH_NOTE
                ADD     SI, 2                   ; 计算下一音符的偏移量
                MOV     WHICH_NOTE, SI
                JMP     NEXT_NOTE               ; 转 NEXT_NOTE，设置下一音符
NO_MORE:        IN      AL, 61H
                AND     AL, 0FCH
                OUT     61H, AL                 ; 关闭扬声器
                MOV     FIRST_NOTE, 1
                MOV     END_NOTE, 0
                MOV     WHICH_NOTE, 0           ; 初始化演奏参数，为下一次演奏作准备
NOT_NOW:        POP     DS
                POP     SI
                POP     DI
                POP     DX
                POP     CX
                POP     BX
                POP     AX
                IRET                            ; 中断返回
NEW_1CH         ENDP
                ;《妈妈再爱我一次》歌曲频率表
FREQ            DW 1 DUP(440,392,330,392,523,440,392,440)
```

```
                DW 1 DUP(330,392,440,392,330,294,262,220,392,330,294)
                DW 1 DUP(296,294,330,392,395,440,330,294,330,294,262)
                DW 1 DUP(392,330,294,262,220,262,196)
                DW 1 DUP(0)
FM              DB 5
                ;演奏时间基值表
DELAY _TAB      DW 1 DUP(75,25,50,50,50,25,25,100)
                DW 1 DUP(50,25,25,50,25,25,25,25,25,25,100)
                DW 1 DUP(50,25,25,50,25,25,50,17,17,17,100)
                DW 1 DUP(75,25,25,25,25,25,100)
SOUND _NOW      DB   1                ;(1=需要演奏，0=不进行演奏)
FIRST _NOTE     DB   1                ;(1=未开始演奏，0=正在演奏)
END _NOTE       DW   0                ;音符演奏的时钟脉冲计数
WHICH _NOTE     DW   0                ;当前音符的偏移值暂存单元
CODE1           ENDS
                END    START
```

第九章　高档微机开发与综合应用实例

本章将举一些实例，用于说明汇编语言程序设计在80286、80386等高档微型计算机系统开发和应用方面的具体应用，如显示卡类型的识别、扩展内存的使用方法、CCDOS系统文件的功能扩充、软件汉化、计算机病毒解毒软件的编制等。要解决这些问题，编写相应的程序，不仅要求程序员学会汇编语言的程序设计方法，而且还要掌握一些有关的知识，如CCDOS系统文件结构和原理，汉字输入、输出和处理过程，计算机病毒的原理等，对于这些内容，读者可以参考有关书籍和资料。

【例9.1】 微机内存容量的检测

下面的程序能够检测各种微机的基本内存容量和扩展、扩充内存容量。

编程指导：

目前，各种286微机上一般都配有1MB的RAM存储器，而大多数386微机则配有1MB以上的RAM，通常情况下，人们将1M字节的RAM划分为两个部分：基本内存和扩展内存。基本内存是指RAM区最低端的640KB，扩展内存则指的是基本内存上面的384KB内存，而将1MB以上的内存称为扩充内存。

在286、386微机系统中，检测内存容量有两种类型的中断功能调用，INT 12H能够检测出基本内存容量，而INT 15H的88H子功能则能检测到扩展内存和扩充内存的容量，此两种中断功能调用以后，都在AX中返回以KB为单位的内存容量。

程序清单

```
STACK     SEGMENT   STACK 'STACK'
          DB        64 DUP (0)
STACK     ENDS
DATA      SEGMENT
BASICMEM  DW ?                          ; 基本内存容量单元
EXTMEM    DW ?                          ; 扩展内存容量单元
DATA      ENDS
CODE      SEGMENT   'CODE'
TESTMEM   PROC      FAR
          ASSUME    CS: CODE, DS: DATA
          PUSH      DS
          XOR       AX, AX
          PUSH      AX
          INT       12H                 ; 确定基本内存容量
          MOV       BASICMEM, AX        ; 保存基本内存容量
          MOV       AH, 88H
```

```
            INT     15H                 ; 确定扩展的内存容量
            MOV     EXTMEM, AX          ; 保存扩展内存容量
            RET
TESTMEM     ENDP
CODE        ENDS
            END     TESTMEM
```

【例 9.2】 微机显示卡类型的识别

下面的程序能够完成对标准显示卡类型的识别，并显示识别的结果。

编程指导：

INT 10H 功能调用的 1AH 子功能是 VGA 显示卡所特有的，所以利用此功能可以测定 VGA 显示卡的存在与否。其使用方法是：

```
MOV     BX, 0
MOV     AX, 1A00H
INT     10H
```

调用以后将返回一个显示器组合码表(DCC 表)，返回的参数 AL=1AH 表示操作功能，AL=0 表示操作失败。在操作成功的情况下，BH 和 BL 寄存器中的值作为返回的 DCC 表。其中 BH 中的值表示第二显示器，BL 中的值表示激活的显示器，两个值代表了那两种视频系统可以混合使用。如果计算机内有两个视频系统，则一个必须是单色，另一个则为彩色，BIOS 数据区的 0040：0010 字节的位 4 和位 5 表示目前正在使用的是那一个系统。DCC 表的主要特征值有 BL=4（表示使用标准彩色的 EGA），BL=5（表示使用单色显示器 EGA），BL=7（表示使用单色显示器 VGA），BL=8（表示使用彩色 VGA）。

EGA 显示卡类型的识别可以通过 INT 10H 的 12H 子功能调用来识别，12H 子功能是 EGA 卡所特有的，其调用方法是：

```
MOV     AH, 12H
MOV     AL, 0
MOV     BL, 10H
INT     10H
```

调用以后，如果 BL 寄存器的值仍为 10H，说明没有 EGA；如果 AL 寄存器中的值为 12H，则说明配置了 VGA。如果 BL 寄存器中的值不为 10H，则可以进一步通过 BIOS 数据区 0040：0087 处存放的一个信息字节进行识别，具体识别方法请阅读程序。

CGA 卡和 HGC 卡的识别方法可以通过 BIOS 数据区的 0040：0010H 字节的内容来进行，具体方法请阅读程序。

程序清单

```
CODE        SEGMENT
            ASSUME  CS: CODE, DS: CODE, ES: CODE
START       PROC    FAR
            JMP     BEGIN
```

```
NO_SUPPORT      DB    '不识别的显示卡! $', 13, 10,' $'
COLOR_CGA_MSG   DB    '彩色 CGA 图象显示卡$', 13, 10,' $'
COLOR_EGA_MSG   DB    '彩色 EGA 图象显示卡$', 13, 10,' $'
COLOR_VGA_MSG   DB    '彩色 VGA 图象显示卡$', 13, 10,' $'
MONO_HGA_MSG    DB    'Hercules 图象显示卡$', 13, 10,' $'
MONO_EGA_MSG    DB    '单色 EGA 图象显示卡$', 13, 10,' $'
MONO_VGA_MSG    DB    '彩色 VGA 图象显示卡$', 13, 10,' $'
BEGIN:          PUSH  CS
                POP   DS
                PUSH  CS
                POP   ES
                MOV   BX, 0
                MOV   AX, 1A00H
                INT   10H
                CMP   AL, 1AH
                JNE   NO_DC
                CMP   BL, 07H
                JE    MONO_V
                CMP   BL, 08H
                JE    COLOR_V
                CMP   BL, 05H
                JE    MONO_E
                CMP   BL, 04H
                JE'   COLOR_E
NO_DC:          MOV   AH, 12H
                MOV   BL, 10H
                INT   10H
                CMP   BL, 10H
                JE    INVALID
                PUSH  DS
                MOV   AX, 0040H
                MOV   DS, AX
                MOV   BL, DS:[0087H]
                POP   DS
                TEST  BL, 08H
                JZ    VALID
INVALID:        PUSH  DS
                MOV   AX, 0040H
                MOV   DS, AX
```

```
                MOV     AL, DS:[0010H]
                POP     DS
                AND     AL, 30H
                CMP     AL, 30H
                JE      MONO_C
                CMP     AL, 20H
                JE      COLOR_C
                MOV     DX, OFFSET NO_SUPPORT
                JMP     FINISH
VALID:          CMP     BH, 01H
                JE      MONO_E
                JMP     COLOR_E
MONO_V:         MOV     DX, OFFSET MONO_VGA_MSG
                JMP     FINISH
COLOR_V:        MOV     DX, OFFSET COLOR_VGA_MSG
                JMP     FINISH
MONO_E:         MOV     DX, OFFSET MONO_EGA_MSG
                JMP     FINISH
COLOR_E:        MOV     DX, OFFSET COLOR_EGA_MSG
                JMP     FINISH
MONO_C:         MOV     DX, OFFSET MONO_CGA_MSG
                JMP     FINISH
COLOR_C:        MOV     DX, OFFSET COLOR_CGA_MSG
FINISH:         MOV     AH, 9
                INT     21H
                MOV     AH, 4CH
                INT     21H
START           ENDP
CODE            ENDS
                END     START
```

【例 9.3】 高档微机扩展内存的使用方法

本例程序完成将汉字操作系统的显示字库装入到 286 或 386 微机的扩展内存中，同时设置了获取汉字显示字模信息的中断功能调用 INT DFH，其主要目的在于介绍高档微机扩展内存的使用方法。

编程指导：

80286、80386 都有两种工作方式，一种称为实方式，另一种则称之为保护方式。在实方式下，80286、80386 相当于高速的 8086/8088，只能寻址 1MB 内存，而在保护方式下，80286 则能寻址 16MB 的内存空间，而 80386 则能寻址更大的内存空间。

由于国内广泛使用的汉字操作系统都配有一个 250KB 左右的显示字库，在汉字操作系统运行时，一般都将汉字显示字库装在基本内存中，这样使得留给用户程序的空间大大减小，无法运行一些较大的应用程序。如果能将汉字显示字库装入到扩展内存或扩充内存中，这将给用户程序留出较大的基本内存空间，以便能够运行大型的应用软件。使用扩展内存和扩充内存都要进入保护工作方式，即使用 24 位的实地址进行寻址，而不像 8086 那样，用段地址和偏移量来寻址。为了方便用户对扩展内存和扩充内存的使用，286、386 微机上的 INT 15H 提供了有关的功能调用。

INT 15H 的 87H 号子功能使得用户可以在 16MB 空间内的任何两块存储区之间传送数据，不过使用这一功能之前，需要首先建立一个描述符表 GDT，以便指出要传送的源地址、目标地址、段长度和访问权限等信息。描述符表共有 6 个描述符项，每个描述符项占用 8 个字节长度，结构如图 9－1 所示。其中：第一个描述符是空的（由用户初始化为全 0）；第二个描述符描述 GDT 本身作为数据段（由用户初始化为全 0）；第三个描述符是源缓冲区描述符（由用户初始化）；第四个描述符是目标缓冲区描述符（由用户初始化）；第五个描述符是用来创建保护方式段的描述符（由用户初始化为全 0）；第六个描述符是用来创建保护方式堆栈段的描述符（由用户初始化为全 0）。在六个描述符中最重要的描述符是源缓冲区和目标缓冲区描述符，其格式如图 9－2 所示。

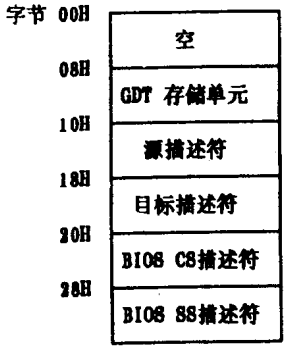


图 9－1　描述符表结构

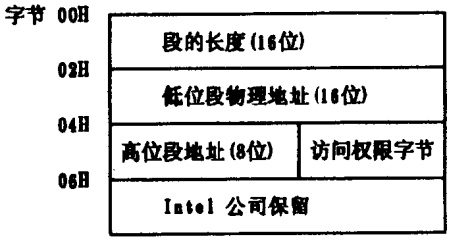


图 9－2　源、目标描述符格式

需要说明的是，由于缓冲区必须适合单段，因此缓冲区的长度必须不能大于 32KB 字，缓冲区的起始地址为 3 字节长的物理地址（24 位），并且传送时，由最低地址开始从源缓冲区传送字到目标缓冲区。

使用 87H 号子程序时，除了建立一个描述符表 GDT 以外，还要设置如下的一些参数：

（1）ES：SI 指向描述符表 GDT；

（2）源和目标缓冲区描述符必须有段长度，其大小为 2×CX－1 字节或更大一些，但不能大于 32K 字，数据存取权限字节应设置为 93H，源和目标都应设置为 24 位的物理地址（字节高，字低），Intel 公司保留字节设置为全 0；

（3）CX 中为要传送的数据块的长度，以字为单位，最大长度为 8000H（32K 字）。

程序清单

```
CODE        SEGMENT
            ORG         100H
```

```
            ASSUME    CS: CODE, DS: CODE, ES: CODE
START:      JMP       LOADLIB
GDT_BUF     DB 8      DUP (0)     ; 全局描述符表
POINTER     DB 8      DUP (0)
SOURCE      DW 2      DUP (0)     ; 源描述符
SOURCE1     DB 4      DUP (0)
TARGET      DW 2      DUP (0)     ; 目标描述符
TARGET1     DB 4      DUP (0)
CS_SEG      DB 8      DUP (0)     ; BIOS 代码段描述符
SS_SEG      DB 8      DUP (0)     ; 堆栈段描述符
ADDR        DW 0
INTDF       PROC   NEAR
            STI
            PUSH   AX
            PUSH   CX
            PUSH   ES
            PUSH   SI
            PUSH   DX
            MOV    CX, 0020H
            MOV    AX, DX    ; 计算当前汉字字模首地址在字库文件中的偏移
            MUL    CX
            ADD    DL, 10H              ; 加上 1MB 地址
            MOV    CS: SOURCE+2, AX     ; 设置源段描述符中的 24 位基地址
            MOV    CS: SOURCE1, DL
            CMP    SI, CS: ADDR         ; 本次地址与上一次地址相同吗?
            JZ     L016E                ; 相同, 转
            MOV    CS: ADDR, SI
            MOV    AX, DS               ; 以下计算目标段描述符中的 24 位基地址
            MOV    DX, 16
            MUL    DX
            ADD    AX, SI
            ADC    DL, 0
            MOV    CS: TARGET+2, AX     ; 设置目标段描述符中的 24 位基地址
            MOV    CS: TARGET1, DL
L016E:      PUSH   CS
            POP    ES
            LEA    SI,GDT_BUF           ;ES:SI 指向全局描述符表(GDT)的段偏移地址
            MOV    CX, 16               ; 传送 16 个字 (一个 16×16 点阵的汉字字模)
            MOV    AH, 87H
```

```
        INT     15H
        POP     DX
        POP     SI
        POP     ES
        POP     CX
        POP     AX
        IRET
INTDF   ENDP
LOADLIB: PUSH   CS
        POP     DS
        PUSH    CS
        POP     ES
        LEA     DX, CCLIB_FILE_NAME
        MOV     AX, 3D00H           ; 打开字库文件
        INT     21H
        JB      L01DF
        MOV     BX, AX
        XOR     DX, DX
        XOR     CX, CX
        MOV     AX, 4202H           ; 指针移到文件尾，返回文件长度
        INT     21H
        JB      L01DF
        MOV     CX, 1024
        DIV     CX                  ; 计算字库文件以K为单位的长度
        INC     AX
        MOV     BP, AX              ; 保存文件长度
        MOV     AX, 4200H           ; 指针移到文件头
        XOR     DX, DX
        XOR     CX, CX
        INT     21H
        JB      L01DF
        MOV     AL, 93H
        MOV     SOURCE1+1, AL       ; 设置源和目标描述符中的访问权限字节值
        MOV     TARGET1+1, AL
        MOV     AX, 0800H
        MOV     SOURCE, AX          ; 设置源和目标描述符的段限字值
        MOV     TARGET, AX
        MOV     AX, DS
        MOV     DX, 16              ; 计算源段的24位基地址
```

```
        MUL     DX
        ADD     AX，OFFSET BUFFER
        ADC     DL，0
        MOV     SOURCE+2，AX        ；设置源段描述符的基地址
        MOV     SOURCE1，DL
L01D5：  MOV     CX，1024
        MOV     DX，OFFSET BUFFER
        MOV     AH，3FH             ；读 1024 字节到 DS：BUFFER
        INT     21H
L01DF：  JB      L0228
        MOV     AX，LENG
        MOV     CX，1024
        MUL     CX                  ；计算目标段描述符的基地址
        ADD     DL，10H
        MOV     TARGET+2，AX
        MOV     TARGET1，DL
        LEA     SI，GDT_BUF
        MOV     CX，512             ；传送 512 个字（1024 字节）
        MOV     AH，87H
        INT     15H
        JB      L0228
        PUSH    AX
        MOV     AX，CS：LENG
        INC     AX
        CMP     AX，BP              ；是否已装入完毕？
        MOV     CS：LENG，AX
        POP     AX
        JNZ     L01D5               ；未装入完字库，转
        MOV     AH，3EH
        INT     21H
        MOV     DX，OFFSET INTDF
        MOV     AX，25DFH           ；设置 INT DFH 中断
        INT     21H
        MOV     DX，0029H
        MOV     AX，3100H           ；驻留退出
        INT     21H
        MOV     AX，4C00H
        INT     21H
L0228：  MOV     AH，9
```

```
        LEA     DX, CCLIB_FILE_NAME
        INT     21H
        MOV     AX, 4C01H           ; 错误返回
        INT     21H
CCLIB_FILE_NAME     DB    'CCLIB', 0
                    DB    'Error! $'
LENG    DW      0                   ; 保存已装入的字库文件长度（以 KB 为单位）
BUFFER  DB      1024  DUP (0)       ; 缓冲区
CODE    ENDS
        END     START
```

【例 9.4】 汉字制表符快速输入程序

在 CCDOS 2.10 版本中，输入中文制表的方法只能在区位码方式下进行，这种方法既不方便，又影响输入速度。下例中的程序通过修改 CCDOS 的键盘输入模块 INT 16H 为其增加快速输入制表符的功能。

编程指导：

本程序首先保存系统中原有的 INT 16H 中断向量地址，再把 16H 中断入口地址指向自编的 INT 16H 附加模块。当有键盘输入时，本程序首先对键入的键盘扫描码进行判别，如果输入的是 Shift 键与有关键的组合，则本程序把扫描码进行转换，返回对应的制表符在 AX 中；如不是，则执行系统原有的键盘管理程序。

用本程序输入制表符的方法是：同时按左 Shift 键与小键盘的数字键输入的是细线条的制表符；同时按右 shift 键与小键盘的数字键输入的是粗线条的制表符。组合键与制表的具体对应关系如图 9-3a 和图 9-3b 所示。

a) 小键盘与左shift键组合　　b) 小键盘与右shift键组合

图 9-3　制表符键位示意图

程序清单

```
CODE        SEGMENT
            ASSUME    CS: CODE, DS: CODE, ES: CODE
New_INT16   PROC      FAR
            OR        AH, AH
            JZ        NEXT2             ; 是 0 号功能，转 NEXT2
```

```
                CMP     AH, 1
                JNZ     NEXT1                   ; 非 0 和 1 号功能，转 NEXT1
                CMP     CS: TAB _ Flag, 0       ; 判暂存单元是否有制表符的低字节内码
                JZ      NEXT1                   ; 没有，转 NEXT1
                MOV     AX, CS: TAB _ CODE      ; 有，返回制表符的低字节内码
                RET     2                       ; 废除堆栈中的标志字并返回系统
NEXT1:          JMP     CS: Old _ INT16         ; 调用原 16H 中断向量
NEXT2:          CMP     CS: TAB _ Flag, 0       ; 判暂存单元是否有制表符的低字节内码
                JZ      NEXT3                   ; 没有，转 NEXT3
                MOV     AX, CS: TAB _ CODE      ; 有，返回制表符的低字节内码
                MOV     CS: TAB _ Flag, 0       ; 置暂存单元为空标志
EXIT:           IRET                            ; 中断返回
NEXT3:          XOR     AH, AH
                CLI
                PUSHF
                CALL    CS: Old _ INT16         ; 执行原 16H 的 0 号功能，从键盘读一字符
                CMP     AH, 47H                 ; 字符扫描码小与 47H，转中断返回
                JB      EXIT
                CMP     AH, 51H
                JA      EXIT                    ; 字符扫描码大与 51H，转中断返回
                MOV     CS: BUFF, AX            ; 暂存键盘扫描码
                PUSHF
                MOV     AH, 2
                CALL    CS: Old _ INT16         ; 读键盘状态
                TEST    AL, 00000001B           ; 是右 Shift 键吗?
                JZ      Test _ left _ Shift     ; 不是，转
                PUSH    BX
                MOV     BX, OFFSET TAB1         ; 是右 Shift 键，粗线条制表符首地址→BX
                JMP     NEXT4
Test _ left _ Shift:    TEST    AL, 00000010B   ; 是左 Shift 键吗?
                MOV     AX, CS: BUFF            ; 恢复 AX 中的键盘扫描码
                JZ      EXIT                    ; 没有按 Shift 键，转中断返回
                PUSH    BX
                MOV     BX, OFFSET TAB2         ; 是左 Shift 键，细线条制表符首地址→BX
NEXT4:          MOV     AX, CS: BUFF            ; 恢复 AX 中的键盘扫描码
                MOV     AL, AH
                SUB     AL, 47H
                MOV     AH, 0
                SHL     AX, 1
```

```
                ADD     BX，AX                  ；计算组合键对应的制表符首地址→BX
                MOV     AX，CS：[BX]            ；制表符的内码→AX
                MOV     CS：BYTE PTR TAB _ CODE，AH  ；内码低字节送暂存单元
                MOV     CS：TAB _ Flag，1       ；置暂存单元为非空标志
                XOR     AH，AH                  ；AH＝0，AL＝制表符内码高字节
                POP     BX
                IRET                            ；中断返回
BUFF            DW      0                       ；键盘扫描码暂存单元
Old _ INT16     DD      ?                       ；系统原有的 16H 中断向量地址保存单元
TAB _ Flag      DB      0                       ；制表符低字节暂存标志字节
TAB _ CODE      DW      0                       ；制表符低字节码暂存单元

TAB1    DB      '┌','┬','┐','─','├','┼','┤','│','└','┴','┘'
TAB2    DB      '┌','┬','┐','─','├','┼','┤','│','└','┴','┘'
Set _ INT16：   PUSH    CS
                POP     DS
                MOV     AX，3516H
                INT     21H
                MOV     WORD PTR Old _ INT16，BX  ；保存系统原有的 16H 中断向量地址
                MOV     WORD PTR Old _ INT16＋2，ES
                MOV     DX，OFFSET New _ INT16
                MOV     AX，2516H
                INT     21H                     ；设置新的 16H 中断向量
                MOV     DX，OFFSET Set _ INT16
                ADD     DX，104H
                INT     27H                     ；程序驻留退出
New _ INT16     ENDP
CODE            ENDS
                END     Set _ INT16
```

【例 9.5】 汉字编码查找

在进行软件汉化工作时，经常需要将西文提示信息修改为相应的汉字提示信息，修改操作往往是针对 EXE 或 COM 等二进制文件进行的，这时就需要知道汉字的内码。试编写一个程序完成：从键盘接收汉字，然后显示汉字的国标码、机内码与区位码。

编程指导：

在计算机上输入汉字时，从键盘键入的字符叫做汉字的输入码，输入码对于不同的汉字来说，其长度不同，但一般来说不超过 4 个字符。计算机接收到输入码后即自动将其转换为汉字的机内码形式进行存储，每一汉字的机内码占用两个字节长度。把键盘接收的汉字信息

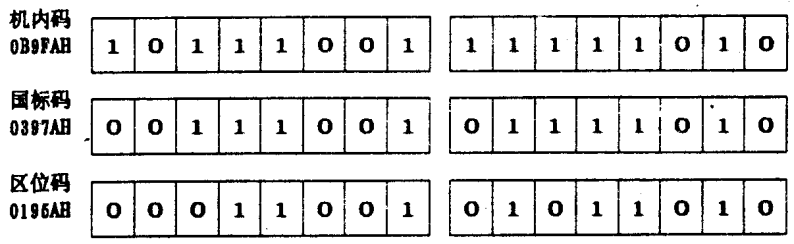


图 9-4 “国”字的机内码、国标码和区位码

直接以十六进制形式显示出来就是该汉字的机内码；将机内码每个字节的最高位屏蔽掉，再以十六进制形式显示出来，则为国标码；将机内码的每个字节各减去 0A0H 再以十进制形式显示出来，即为该汉字的区位码。例如“国”字的机内码、国标码与区位码如图 9-4 所示。

程序清单

```
STACK       SEGMENT   PARA STACK 'STACK'
            DB        128 DUP (0)
STACK       ENDS
DATA        SEGMENT   PARA PUBLIC 'CODE'
BUFFER      DB        128,?, 128 DUP (0)
DISP_MSG1   DB        13, 10,'请输入待查汉字：$'
DISP_MSG2   DB        13, 10,'国标码是：$'
DISP_MSG3   DB        13, 10,'机内码是：$'
DISP_MSG4   DB        13, 10,'区位码是：$'
DATA        ENDS
CODE        SEGMENT   PARA PUBLIC 'CODE'
            ASSUME    CS: CODE, DS: DATA
START       PROC  FAR
            PUSH  DS
            XOR   AX, AX                    ; 异或指令的用法
            PUSH  AX
            MOV   AX, DATA
            MOV   DS, AX
            MOV   DX, OFFSET DISP_MSG1
            CALL  DISP_STRING               ; 显示提示信息
            MOV   DX, OFFSET BUFFER
            MOV   AH, 0AH
            INT   21H                       ; 接收键盘输入的汉字内码
            CALL  CRAF
            MOV   DX, OFFSET DISP_MSG2
            CALL  DISP_STRING
```

```
        MOV   BX, OFFSET BUFFER+1
        XOR   CH, CH
        MOV   CL, [BX]
        INC   BX
NEXT1:  MOV   AL, [BX]
        AND   AL, 07FH                ; 屏蔽最高位
        PUSH  CX
        MOV   CL, 4
        PUSH  BX
        MOV   BL, AL
        ROL   AL, CL
        MOV   CX, 2
NEXT2:  AND   AL, 0FH
        CMP   AL, 09H
        JBE   NEXT3
        ADD   AL, 7
NEXT3:  ADD   AL, 30H
        MOV   DL, AL
        MOV   AH, 2
        INT   21H                     ; 显示对应的国标码
        MOV   AL, BL
        LOOP  NEXT2
NEXT4:  MOV   DL, 20H
        MOV   AH, 2
        INT   21H
        POP   BX
        INC   BX
        POP   CX
        LOOP  NEXT1
        CALL  CRAF
        MOV   DX, OFFSET DISP_MSG3
        CALL  DISP_STRING
        MOV   BX, OFFSET BUFFER+1
        XOR   CH, CH
        MOV   CL, [BX]
        INC   BX
NEXT5:  MOV   AL, [BX]
        PUSH  CX
        MOV   CL, 4
```

```
        PUSH    BX
        MOV     BL, AL
        ROL     AL, CL
        MOV     CX, 2
NEXT6:  AND     AL, 0FH
        CMP     AL, 09H
        JBE     NEXT7
        ADD     AL, 7
NEXT7:  ADD     AL, 30H
        MOV     DL, AL
        MOV     AH, 2                   ; 显示对应的机内码
        INT     21H
        MOV     AL, BL
        LOOP    NEXT6
NEXT8:  MOV     DL, 20H
        MOV     AH, 2
        INT     21H
        POP     BX
        INC     BX
        POP     CX
        LOOP    NEXT5
        CALL    CRAF
        MOV     DX, OFFSET DISP _ MSG4
        CALL    DISP _ STRING
        MOV     BX, OFFSET BUFFER+1
        XOR     CH, CH
        MOV     CL, [BX]
        INC     BX
NEXT9:  MOV     AL, [BX]
        MOV     AH, 0
        SUB     AL, 0A0H
        PUSH    CX
        MOV     CL, 0AH
        DIV     CL
        OR      AL, 30H
        PUSH    AX
        MOV     AH, 2
        MOV     DL, AL
        INT     21H
```

```
            POP     AX
            MOV     DX, AX
            OR      AH, 30H
            MOV     DL, AH
            MOV     AH, 2                    ; 显示对应的区位码
            INT     21H
            MOV     DL, 20H
            INT     21H
            POP     CX
            INC     BX
            LOOP    NEXT9
            CALL    CRAF
            RET
START       ENDP
DISP_STRING PROC    NEAR
            MOV     AH, 9
            INT     21H
            RET
DISP_STRING ENDP
CRAF        PROC    NEAR
            MOV     AH, 2
            MOV     DL, 0DH
            INT     21H
            MOV     DL, 0AH
            INT     21H
            RET
CRAF        ENDP
CODE        ENDS
            END     START
```

【例 9.6】“小球”病毒解毒程序

由于计算机病毒的广泛传播和蔓延，编写消除计算机病毒的解毒程序也是微型计算机系统技术人员必不可少的一项工作。因为计算机病毒的分析和消除技术要涉及到系统内部的许多深层次的技术，而汇编语言程序设计能充分地发挥微机系统的所有功能，所以要编写计算机病毒的解毒程序，汇编语言程序设计是最有力的工具。下面的程序能够解除软盘和硬盘上的“小球”病毒。

编程指导：

1. 病毒简介

“小球”病毒又叫做“圆点”病毒、“乒乓”病毒、“弹球”病毒，它是我国最早发现的一

种计算机病毒，属于操作系统型的良性病毒。病毒发作时，屏幕上出现一个四处弹跳的圆点，干扰正常的屏幕显示，并且随着时间的推移，机器速度不断地变慢，直到使用户无法进行工作，但是它不破坏磁盘文件。

2. 病毒的结构及其特征

“小球”病毒的病毒程序大约为1K字节，占用软盘的两个扇区，分两个部分存放于磁盘上，第一部分占用了磁盘的引导扇区，第二部分则占用了磁盘上另外一个空闲簇。由于一个簇为两个扇区，所以在该簇中，病毒程序只占用了第一个扇区，另外一个扇区则存放的是原来真正的磁盘引导记录。病毒程序在传染成功以后，即将文件分配表中病毒程序第二部分所占的空闲簇登记项的内容修改为FF7，以示为坏簇，防止其它程序使用该簇。对于被感染的不同的磁盘来说，病毒程序第二部分所占的空闲簇位置一般来说是不同的，病毒程序两部分之间为了取得联系，特将第二部分所在簇的第一扇区的逻辑扇区号，存放在第一部分所在扇区（逻辑0扇区）的偏移地址1F9H～1FAH。

一个被感染“小球”病毒的磁盘，一般有下面三个主要特征：

(1) 磁盘引导扇区的开头指令代码为“EB 1C”，对应的指令为“JMP 011E”；

(2) 磁盘引导扇区的结尾有字符串“57 13 55 AA”；

(3) 用PCTOOLS的M命令显示磁盘映象时，发现原来正常的磁盘扇区被标为坏扇区。

3. 病毒消除方法

根据上面对“小球”病毒工作原理的分析，消除“小球”病毒的方法主要分为两步来进行：

(1) 恢复磁盘引导扇区的原始内容；

(2) 收回病毒程序第二部分所占的簇。

根据上述两条原则，解毒程序首先将引导扇读入内存，然后根据病毒的程序代码来判断有无病毒。若无，则返回；若有，则保存病毒第二部分逻辑扇区号，并且据此计算出对应的簇号。然后恢复原引导扇区（将病毒第二部分所占簇的第二扇区内容写入引导扇区），最后根据前面的计算结果收回被病毒程序第二部分所占用的簇（将FAT表对应登记项的内容由7FF改为000）。至此，解毒结束。本消除程序可以解除硬盘和软盘上的“小球”病毒。

程序清单

```
STACK       SEGMENT   STACK 'STACK'
            DB        64 dup (0)
STACK       ENDS
DATA        SEGMENT
;;;;;;;;;;;;;;;;;;;;;;;;;;“小球”病毒解毒程序;;;;;;;;;;;;;;;;;;;;;;
; 北京电子科技学院一系        王成玖  彭继东        1991. 6. 13.          ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
BPB         DB 20H DUP (0)                    ; 磁盘基数暂存区
MES         DB 1024 DUP (0)
ERR         DB 0DH,0AH,20H,20H,'COMMAND LINE ERROR!',0DH,0AH,'$'
SUC1        DB 0DH,0AH,20H,20H,'NO FOUND!',0DH,0AH,'$'
```

```
SUC2    DB 0DH, 0AH, 20H, 20H,'FIND VIRUS!', 0DH, 0AH,'$'
SUC3    DB 0DH, 0AH, 20H, 20H,'SUCCESS IN KILLING!', 0DH, 0AH,'$'
FIRST   DW 0
FA      DW 0
HAND    DW 0
DRIV    DB  0
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE, SS: STACK
DISP    MACRO   ADDR                    ; 宏定义（显示提示信息）
        LEA     DX, ADDR
        MOV     AH, 9
        INT     21H
        ENDM
START   PROC    FAR
        MOV     SI, 0081H               ; 取 DOS 命令行参数（即驱动器标识符）
        MOV     CL, [SI-1]
        XOR     CH, CH
        CMP     CX, 0
        JNZ     RETRY1
ERR1:   DISP    ERR                     ; 驱动器码错，返回 DOS
        JMP     EXIT
RETRY1: LODSB
        CMP     AL, 20H
        JZ      SKIP1
        CMP     AL, 09H
        JZ      SKIP1
        JMP     NEXT
SKIP1:  LOOP    RETRY1
NEXT:   PUSH    AX
        ASSUME  DS: DATA, ES: DATA
        MOV     AX, DATA
        MOV     DS, AX
        MOV     ES, AX
        POP     AX
        AND     AL, 0DFH
        CMP     AL,'A'                  ; 判断驱动器标识符
        JZ      M1
        CMP     AL,'B'
```

```
        JZ      M2
        CMP     AL,'C'
        JZ      M3
        JMP     ERR1
M1:     MOV     BYTE PTR DRIV, 0
        JMP     RBOOT
M2:     MOV     BYTE PTR DRIV, 1
        JMP     RBOOT
M3:     MOV     BYTE PTR DRIV , 2
RBOOT:  LEA     BX, MES                 ; 读引导扇区
        MOV     CX, 1
        MOV     AL, DRIV
        MOV     DX, 0
        INT     25H
        POPF
        MOV     AX, [BX+1FCH]           ; 保存病毒第二部分逻辑扇区号
        CMP     AX, 1357H               ; 具体判别是否有病毒
        JNZ     NOVIRUS
        MOV     AX, [BX+27H]
        CMP     AX, 013A1H
        JNZ     NOVIRUS
        MOV     AX, [BX+4EH]
        CMP     AX, 2680H
        JNZ     NOVIRUS
        MOV     AX, [BX+66H]
        CMP     AX, 0C0B8H
        JNZ     NOVIRUS
        ADD     BX, 27H
        MOV     AX, 0
        MOV     CX, 2FH
        MOV     AX, [BX]
MB:     ADD     BX, 2
        XOR     AX, [BX]
        LOOP    MB
        CMP     AX, 07FEBH
        JNZ     NOVIRUS
        LEA     BX, MES +1F9H
        MOV     AX, [BX]
        MOV     WORD PTR HAND, AX
```

```
        DISP    SUC2            ; 发现病毒显示提示信息
        CALL    BEGIN           ; 调用解毒程序
NOVIRUS: DISP   SUC1            ; 无病毒提示
EXIT:   MOV     AX, 4C00H       ; 返回 DOS
        INT     21H
START   ENDP
BEGIN   PROC    NEAR            ; 解毒程序
        LEA     DI, BPB         ; 保存 BPB 参数至 BPB 单元
        LEA     SI, MES
        MOV     CX, 10H
        CLD
        REP     MOVSW
        MOV     DX, HAND        ; 读病毒第二部分第二扇区（原引导扇）
        INC     DX
        MOV     AL, DRIV
        MOV     CX, 1
        LEA     BX, MES
        INT     25H
        POPF
        MOV     AL, DRIV        ; 恢复原引导扇
        LEA     BX, MES
        MOV     CX, 1
        MOV     DX, 0
        INT     26H
        POPF
        MOV     CX, [BX+0EH]    ; 计算最大簇号放 AX 中
        MOV     AL, [BX+10H]
        CBW
        MUL     WORD PTR [BX+16H]
        ADD     CX, AX
        MOV     AX, 020H
        MUL     WORD PTR [BX+11H]
        ADD     AX, 1FFH
        MOV     BX, 0200H
        DIV     BX
        ADD     CX, AX
        LEA     SI, BPB
        MOV     AX, [SI+13H]
        MOV     BL, [SI+0DH]
```

```
        SUB     AX, CX
        MOV     FIRST, CX           ; 保存文件区起始逻辑扇区号
        XOR     DX, DX
        XOR     BH, BH
        DIV     BX
        INC     AX
        CALL    FB                  ; 调用子程序
        CMP     AX, 0FF0H           ;判断 FAT 每项应占多少位(16 或 12 位)
        JA      WANG                ; 16 位转
        MOV     AX, FA              ; 12 位×1. 5=FAT 中位置
        SHL     AX, 1
        ADD     AX, FA
        SHR     AX, 1
        JMP     MBB
WANG:   MOV     AX, FA              ; 16 位×2=FAT 中位置
        SHL     AX, 1
        MOV     WORD PTR BPB, 0FFH  ; 16 位标志
MBB:    LEA     BX, BPB             ; 计算在 FAT 中第几扇区放 DI 中
        MOV     DI, [BX+0EH]
        MOV     SI, [BX+16H]
        MOV     BX, 0400H
        INC     AX
MY:     CMP     AX, BX
        JB      RFAT1
        SUB     AX, 0200H
        INC     DI
        JMP     MY
RFAT1:  DEC     AX
        PUSH    AX
        MOV     AL, DRIV            ; 读第一份 FAT 中要修改的扇区
        LEA     BX, MES
        MOV     CX, 2
        MOV     DX, DI
        INT     25H
        POPF
        POP     AX
        ADD     BX, AX
        CMP     WORD PTR BPB, 0FFH  ; 修改 FAT 表中内容 (置 0)
        JNZ     MW
```

```
        MOV     AX, 0
        JMP     WFAT1
MW:     MOV     AX, [BX]
        TEST    WORD PTR FA, 01H
        JZ      WFAT
        AND     AX, 0FH
        JMP     WFAT1
WFAT:   AND     AX, 0F000H
WFAT1:  MOV     [BX], AX
        MOV     CX, 2                   ; 写第一份 FAT
        MOV     AL, DRIV
        LEA     BX, MES
        INT     26H
        POPF                            ; 写第二份 FAT
        MOV     CX, 2
        LEA     BX, MES
        MOV     AL, DRIV
        ADD     DX, SI
        INT     26H
        POPF
        DISP    SUC3                    ; 提示解毒成功
        MOV     AX, 4C00H               ; 直接返回 DOS
        INT     21H
        RET
BEGIN   ENDP
FB      PROC    NEAR                    ; 将病毒对应扇区号转变成簇号
        PUSH    AX
        PUSH    BX
        PUSH    DX
        PUSH    SI
        MOV     AX, HAND                ; 取扇区号
        SUB     AX, FIRST               ; 扇区号-文件区起始扇区号
        LEA     SI, BPB
        MOV     BL , [SI+0DH]           ; 取每簇扇区数
        MOV     DX, 0
        MOV     BH, 0
        DIV     BX
        ADD     AX, 2
        MOV     FA, AX                  ; 保存在 FA 中（对应簇号）
```

```
        POP     SI
        POP     DX
        POP     BX
        POP     AX
        RET
FB      ENDP
CODE    ENDS
        END     START
```

附　录

附录 A　8088 指令系统一览表

一、符号说明

（一）标志位名称及其对应的符号

AF＝辅助进位标志　　PF＝奇偶标志　　CF＝进位标志

SF＝符号标志　　DF＝方向标志　　TF＝陷井标志

IF＝中断标志　　ZF＝零标志　　OF＝溢出标志

（二）标志位状态及其对应的表示符号

－＝不影响标志位

√＝影响标志位

R＝恢复标志位的原值

×＝不确定

0＝无条件地设置成零

1＝无条件地设置成 1

（三）操作数符号

SRC＝源操作数

DEST＝目的操作数

寄＝寄存器

存＝存储器

数＝立即数

二、指令一览表

名　称	指令助记符	操作数	功　能　说　明	标志位状态 O D I T S Z A P C
ASCII 加法调整	AAA	无	若（AL&0FH）＞9或 AF＝1，则 AL←AL＋6，AH←AH＋1 AF←1，CF←AF，AL←AL&0FH	×－－－××√×√
ASCII 除法调整	AAD	无	AL←AH×10＋AL AH←0	×－－－√√×√×
ASCII 乘法调整	AAM	无	AH←AL÷10的商 AL←AL÷10的余数	×－－－√√××√×
ASCII 减法调整	AAS	无	若（AL&0FH）＞9或 AF＝1，则 AL←AL－6，AH←AH－1，AF←1 CF←AF，AL←AL&0FH	×－－－××√×√

（续）

名　称	指令助记符	操作数	功 能 说 明	标志位状态 O D I T S Z A P C
带进位加法	ADC	寄，寄 寄，存 存，寄 寄，数 存，数	若 CF＝1，则 (DEST)←(DEST)＋(SRC)＋1 否则 (DEST)←(DEST)＋(SRC)	√－－－√√√√√
加　法	ADD	同 ADC	(DEST) ← (DEST) ＋ (SRC)	√－－－√√√√√
逻 辑 与	AND	同 ADC	(DEST) ← (DEST) AND (SRC) CF←0，OF←0	0－－－√√×√ 0
段内直接调用	CALL	近标号	SP←SP－2，(SP＋1，SP) ←IP IP←偏移地址	－－－－－－－－－
段内间接调用	CALL	寄/存 （字）	SP←SP－2，(SP＋1，SP) ←IP IP← （寄/存）	－－－－－－－－－
段间直接调用	CALL	远标号	SP←SP－2，(SP＋1，SP) ←CS SP←SP－2，(SP＋1，SP) ←IP CS←段地址，IP←偏移地址	－－－－－－－－－
段间间接调用	CALL	存 （双字）	SP←SP－2，(SP＋1，SP) ←CS SP←SP－2，(SP＋1，SP) ←IP CS← （存＋2），IP← （存）	－－－－－－－－－
字节转为字	CBW	无	若 AL＜80H，则 AH←0 否则 AH←0FFH	－－－－－－－－－
清进位标志	CLC	无	CF←0	－－－－－－－－ 0
清方向标志	CLD	无	DF←0	－ 0 －－－－－－－
清中断标志	CLI	无	IF←0	－－ 0 －－－－－－
进 位 求 反	CMC	无	若 CF＝0，则 CF←1；否则 CF←0	－－－－－－－－√
比　较	CMP	寄，寄 寄，存 存，寄 寄，数 存，数	(DEST) － (SRC) 的结果反映 在标志寄存器中，但不影响 操作数本身	√－－－√√√√√
字节串比较	CMPS CMPSB	存，存	(DEST) － (SRC)；若 DF＝0，则 SI←SI＋1，DI←DI＋1；若 DF＝1 则 SI←SI－1，DI←DI－1	√－－－√√√√√
字 串 比 较	CMPS CMPSW	存，存	(DEST) － (SRC)；若 DF＝0，则 SI←SI＋2，DI←DI＋2，若 DF＝1 则 SI←SI－2，DI←DI－2	√－－－√√√√√
字转为双字	CWD	无	若 AX＜8000H，则 DX←0， 否则 DX←0FFFFH	－－－－－－－－－

（续）

名　称	指令助记符	操作数	功　能　说　明	标志位状态 O D I T S Z A P C
组合的十进制数加法调整	DAA	无	若（AL&0FH）>9或 AF=1，则 AL←AL+6，AF←1 若（AL）>9FH 或 CF=1，则 AL←AL+60H，CF←1	×－－－√√√√√
组合的十进制数减法调整	DAS	无	若（AL&0FH）>9或 AF←1，则 AL←AL-6，AF←1 AL←AL-60H，CF←1	×－－－√√√√√
减 1	DEC	寄/存	（寄/存）←（寄/存）－1	√－－－√√√√－
无符号数的除法	DIV	寄/存	字节除法：AL←AX÷（寄/存）的商；AH←余数 字除法：AX←（DX，AX）÷（寄/存）的商；DX←余数	×－－－××××××
向其它处理器提供指令	ESC	数，寄 数，存	向其它处理器提供一条指令（立即数为外操作码0～63）	－－－－－－－－－
暂　停	HLT	无	使处理器暂停	－－－－－－－－－－
带符号数的整数除法	IDIV	寄/存	与 DIV 相同，但用于有符号数的除法，余数与被除数同号	×－－－××××××
带符号数的整数乘法	IMUL	寄/存	若为字节乘法，则 AX←AL×（寄/存） 若为字乘法，则 DX，AX←AX×（寄/存）	√－－－××××√
输　入	IN	AL，DX AL，数 AX，DX AX，数	从 DX 或“数”所指明的端口中取一个字节→AL 从 DX 或“数”所指明的端口中取一个字→AX	－－－－－－－－－
加 1	INC	寄/存	（寄/存）←（寄/存）+1	√－－－√√√√－
中断调用	INT	功能号	SP←SP-2，（SP+1，SP）←Flags，IF←0，TF←0； SP←SP-2，（SP+1，SP）←CS SP←SP-2，（SP+1，SP）←IP CS←（功能号×4+2） IP←（功能号×4）	－－0 0－－－－－
溢出中断	INTO	无	若 OF=0，则不操作 若 OF=1，则执行 INT 4	－－0 0－－－－－
中断返回	IRET	无	IP←（SP+1，SP），SP←SP+2 CS←（SP+1，SP），SP←SP+2 Flag←（SP+1，SP），SP←SP+2	R R R R R R R R R

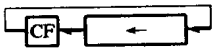
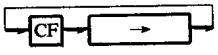
（续）

名　称	指令助记符	操作数	功 能 说 明	标志位状态 O D I T S Z A P C
高于或不低于等于时转移	JA/JNBE	短标号	CF＝0且 ZF＝0时转移	————————
高于等于或不低于或无进位时转移	JAE/JNB /JNC	短标号	CF＝0时转移	————————
低于或不高于等于或有进位时转移	JB/JNAE /JC	短标号	CF＝1时转移	————————
低于等于或不高于时转移	JBE/JNA	短标号	CF＝1或 ZF＝1时转移	————————
CX 为零时转移	JCXZ	短标号	CX＝0时转移	————————
等于或为零时转移	JE/JZ	短标号	ZF＝1时转移	————————
大于或不小于等于时转移	JG/JNLE	短标号	ZF＝0且 SF＝OF 时转移	————————
大于等于或不小于时转移	JGE/JNL	短标号	SF＝OF 时转移	————————
小于或不大于等于时转移	JL/JNGE	短标号	SF≠OF 时转移	————————
小于等于或不大于时转移	JLE/JNG	短标号	ZF＝1或 SF≠OF 时转移	————————
无条件转移	JMP	近标号	IP←OFFSET 标号	————————
		远标号	CS←SEG 标号 IP←OFFSET 标号	————————
		寄/存（字）	IP←（寄/存）	————————
		存（双字）	CS←（存＋2），IP←（存）	————————
不等于或非零时转移	JNE/JNZ	短标号	ZF＝0时转移	————————
无溢出转移	JNO	短标号	OF＝0时转移	————————
无符号转移	JNS	短标号	SF＝0时转移	————————
溢出时转移	JO	短标号	OF＝1时转移	————————
偶校验转移	JP/JPE	短标号	PF＝1时转移	————————
奇校验转移	JPO/JNP	短标号	PF＝0时转移	————————
有符号转移	JS	短标号	SF＝1时转移	————————
装入标志位低字节到 AH	LAHF	无	SF、ZF、AF、PF 和 CF 装入到 AH 寄存器的第7，6，4，2，0位	————————

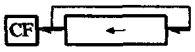
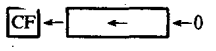
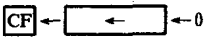
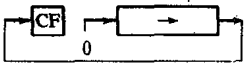
（续）

名　称	指令助记符	操作数	功 能 说 明	标志位状态 O D I T S Z A P C
装入 DS	LDS	寄，存 （双字）	DS←（存+2） 寄←（存）	————————
装有效地址	LEA	寄，存	寄←存储器的偏移地址	————————
装入 ES	LES	寄，存 （双字）	ES←（存+2） 寄←（存）	————————
封锁总线	LOCK	指令	本指令用作其它指令的 前面，使指令执行期间发出 总线封锁信号	————————
取字节串	LODS LODSB	存	AL←（存），若 DF=0，则 SI←SI+1，否则 SI←SI-1	————————
取字串	LODS LODSW	存	AX←（存），若 DF=0，则 SI←SI+2，否则 SI←SI-2	————————
用 CX 计数器 控制循环	LOOP	短标号	CX←CX-1，若 CX≠0，则 IP←标号偏移地址 否则 IP 不变	————————
	LOOPE/ LOOPZ	短标号	CX←CX-1，若 CX≠0且 ZF=1 则 IP←OFFSET 标号	————————
	LOOPNE/ LOOPNZ	短标号	CX←CX-1，若 CX≠0且 ZF=0 则 IP←标号偏移地址 否则 IP 不变	————————
传　送	MOV	(DEST)， (SRC)	(DEST) ← (SRC)，其中 DEST 可 以是寄（IP 除外）/存；SRC 可 以是寄（IP 除外）/存/数	————————
字符串传送	MOVS MOVSB	存，存	(ES：DI) ← (DS：SI)；若 DF=0， 则 SI←SI+1，DI←DI+1；若 DF =1，则 SI←SI-1，DI←DI-1	————————
字串传送	MOVS MOVSW	存，存	(ES：DI) ← (DS：SI)，若 DF=0， 则 SI←SI+2，DI←DI+2，若 DF =1，则 SI←SI-2，DI←DI-2	————————
无符号数乘法	MUL	寄/存	若为字节乘法： AX←（寄/存）×AL 若为字乘法： DS：AX←（寄/存）×AX	√———————√
求　补	NEG	寄/存	(寄/存) ←0－（寄/存）	√———√√√√ 1
空操作	NOP	无		————————

（续）

名　称	指令助记符	操作数	功　能　说　明	标志位状态 O D I T S Z A P C
求　反	NOT	寄/存	字节： （寄/存）←0FFH－（寄/存） 字： （寄/存）←0FFFFH－（寄/存）	－－－－－－－－－
逻辑或	OR	寄，寄 寄，存 存，寄 寄，数 存，数	（DEST）←（DEST）OR（SRC）	0－－－√√×√ 0
输　出	OUT	DX，AL 数，AL DX，AX 数，AX	把AL或AX中的值送向指定的端口	－－－－－－－－－
出　栈	POP	寄/存	（寄/存）←（SP＋1，SP） SP←SP＋2	－－－－－－－－－
标志寄存器出栈	POPF	无	Flags←（SP＋1，SP） SP←SP＋2	－－－－－－－－－
入　栈	PUSH	寄/存	SP←SP－2 （SP＋1，SP）←（寄/存）	－－－－－－－－－
标志寄存器入栈	PUSHF	无	SP←SP－2 （SP＋1，SP）←Flags	－－－－－－－－－
带进位循环左移	RCL	寄/存， 1/CL		√－－－－－－－√
带进位循环右移	RCR	寄/存， 1/CL		√－－－－－－－√
重复串操作	REP	串指令	当CX≠0时，重复串操作	
	REPE/ REPZ	串指令	当CX≠0或ZF＝1时，重复串操作	
	REPNE/ REPNZ	串指令	当CX≠0或ZF＝0时，重复串操作	
过程返回	RET	空/数	IP←（SP＋1，SP），SP←SP＋2 若为段间返回，则还要执行 CS←（SP＋1，SP） SP←SP＋2［＋字节数］	－－－－－－－－－

（续）

名　称	指令助记符	操作数	功 能 说 明	标志位状态 O D I T S Z A P C
小循环左移	ROL	寄/存， 1/CL		－－－－－－－－√
小循环右移	ROR	寄/存， 1/CL		√－－－－－－－√
AH 送标志寄存器低字节	SAHF	无	Flags 的低字节←AH	－－－－√√√√√
算术左移	SAL	寄/存， 1/CL		√－－－－－－－√
算术右移	SAR	寄/存， 1/CL		0－－－√√×√√
带进位减法	SBB	寄，寄 存，寄 寄，存 寄，数 存，数	若 CF＝1，则 (DEST) ← (DEST) － (SRC) － 1 否则 (DEST) ← (DEST) － (SRC)	√－－－√√√√√
字符串扫描	SCAS SCASB	存	将 AL 与（存）进行比较， 若 DF＝0，则 DI←DI＋1 否则 DI←DI－1	√－－－√√√√√
字串扫描	SCAS SCASW	存	将 AX 与（存）进行比较， 若 DF＝0，则 DI←DI＋2 否则 DI←DI－2	√－－－√√√√√
逻辑左移	SHL	存		√－－－－－－－√
逻辑右移	SHR	存		√－－－ 0√×√√

（续）

名　称	指令助记符	操作数	功　能　说　明	标志位状态 O D I T S Z A P C
置进位标志	STC	无	CF←1	———————— 1
置方向标志	STD	无	DF←1	— 1 ———————
置中断标志	STI	无	IF←1	—— 1 ——————
存字符串	STOS STOSB	存	(存) ←AL；若 DF=0，则 DI←DI+1，否则 DI←DI -1	————————
存字串	STOS STOSW	存	(存) ←AX；若 DF=0，则 DI←DI+2，否则 DI←DI -2	————————
减　法	SUB	同 SBB	(DEST) ← (DEST) — (SRC)	√———√√×√√
测　试	TEST	同 SBB	(DEST) AND (SRC) 的结果反映在标志位上，但不送存	0 ———√√×√ 0
等　待	WAIT	无	使处理器处于等待状态，直到产生一个外部中断为止	————————
交　换	XCHG	寄，寄 寄，存 存，寄	(DEST) 与 (SRC) 交换	————————
查　表	XLAT	表名	AL← (BX+AL)	————————
异　或	XOR	同 SBB	(DEST) ← (DEST) XOR (SRC)	0 ———√√×√ 0

附录 B　DOS 软中断及系统功能调用一览表

一、DOS 的软中断

DOS 2.10软中断主要使用了中断20H～27H，其功能定义如表 B -1所示。

表 B -1　DOS 软中断一览表

中　断	功　能	入口参数	出口参数
INT 20	程序正常退出	CS=PSP 段地址	
INT 21	系统功能调用	AH=调用号 功能调用入口参数	功能调用出口参数
INT 22	程序结束处理		
INT 23	Ctrl _ Break 退出		
INT 24	严重错误处理		
INT 25	绝对磁盘读	AL=驱动器号 CX=读入扇区数 DX=起始逻辑扇区号 DS：BX=缓冲区地址	AL=0（忽略） AL=1（重试） AL=2（通过 INT 23H 终止）

（续）

中　断	功　能	入口参数	出口参数
INT 26	绝对磁盘写	AL＝驱动器号 CX＝写盘扇区数 DX＝起始逻辑扇区号 DS：BX＝缓冲区地址	Cy＝1　出错 Cy＝0　写盘正确
INT 27	驻留退出	CS＝PSP段地址 DX＝程序末地址＋1	

二、DOS的系统功能调用

系统功能调用是DOS为用户程序提供的一组常用的子程序，如表B-2。程序员可通过下面的方法使用相应的系统功能调用：把功能调用编号送入寄存器AH中，设置好入口参数，然后向DOS发出INT 21H命令，最后获取出口参数。在表B-2中，带“＊”的调用号是DOS未公开的系统功能调用，这些系统功能调用是DOS提供给其系统软件使用的。

表B-2　DOS 2.10系统功能调用一览表

调用号	功　能	入口参数	出口参数
00H	程序终止	CS＝PSP段地址	
01H	键盘输入字符		AL＝输入的字符
02H	显示输出	DL＝显示的字符	
03H	串行设备输入		AL＝串行口输入的字符
04H	串行设备输出	DL＝输出的字符	
05H	打印输出	DL＝输出的字符	
06H	直接控制台I/O	DL＝FF（输入请求）	AL＝输入的字符
		DL＝字符（输出请求）	
07H	直接控制台输入 （不显示输入）		AL＝输入的字符
08H	键盘输入字符 （不显示输入）		AL＝输入的字符
09H	显示字符串	DS：DX＝缓冲区首址	
0AH	输入字符串	DS：DX＝缓冲区首址	
0BH	检查标准输入状态		AL＝00，无按键 AL＝FF，有按键
0CH	清除输入缓冲区， 并执行指定的标准 输入功能	AL＝功能号 （01/06/07/08/0AH） DS：DX＝缓冲区（功能0AH）	AL＝输入的数据 （功能01/06/07/08）
0DH	初始化磁盘状态		
0EH	选择缺省的驱动器	DL＝驱动器号（0＝A，1＝B..）	AL＝逻辑驱动器数
0FH	打开文件	DS：DX＝未打开的FCB首址	AL＝00：成功，FF：失败
10H	关闭文件	DS：DX＝打开的FCB首址	AL＝00：成功，FF：失败

（续）

调 用 号	功 能	入 口 参 数	出 口 参 数
11H	查找第一匹配目录	DS：DX＝未打开的 FCB 首址	AL＝00：成功，FF：失败
12H	查找下一匹配目录	DS：DX＝未打开的 FCB 首址	AL＝00：成功，FF：失败
13H	删除文件	DS：DX＝未打开的 FCB 首址	AL＝00：成功，FF：失败
14H	顺序读	DS：DX＝打开的 FCB 首址	AL＝00：成功 AL＝01：文件结束 AL＝02：缓冲区太小 AL＝03：缓冲区不满
15H	顺序写	DS：DX＝打开的 FCB 首址	AL＝00：成功，01：盘满 AL＝02：缓冲区太小
16H	创建文件	DS：DX＝未打开的 FCB 首址	AL＝00：成功 AL＝FF：目录区满
17H	文件换名	DS：DX＝被修改的 FCB 首址	AL＝00：成功，AL＝FF 未找到目录项或文件名已有
＊18H	保留未用		
19H	取缺省驱动器号		AL＝驱动器号（0＝A，1＝B）
1AH	设置磁盘缓冲区：DTA	DS：DX＝磁盘缓冲区首址	
＊1BH	取缺省驱动器的磁盘格式信息		AL＝每簇的扇区数 CX＝每扇区的字节数 DX＝数据区总簇数-1 DS：BX＝介质描述字节
＊1CH	取指定驱动器的磁盘格式信息	DL＝驱动器号 （0＝缺省，1＝A，…）	AL＝每簇的扇区数 CX＝每扇区的字节数 DX＝数据区总簇数-1 DS：BX＝介质描述字节
＊1DH	保留未用		
＊1EH	保留未用		
＊1FH	取缺省驱动器的 DPB		DS：BX＝DPB
＊20H	保留未用		
21H	随机读一个记录	DS：DX＝打开的 FCB 首址	AL＝00　读操作成功 AL＝01　文件结束 AL＝02　缓冲区太小 AL＝03　缓冲区不满
22H	随机写一个记录	DS：DX＝打开的 FCB 首址	AL＝00　写操作成功 AL＝01　磁盘满 AL＝02　缓冲区太小
23H	取文件大小	DS：DX＝未打开的 FCB 首址	AL＝00：成功，FF：失败
24H	设置随机记录号	DS：DX＝打开的 FCB 首址	

（续）

调 用 号	功 能	入 口 参 数	出 口 参 数
25H	设置中断向量	AL＝中断号 DS：DX＝中断程序入口地址	
* 26H	创建新的 PSP	DS：DX＝新的 PSP 段地址	
27H	随机读若干记录	DS：DX＝打开的 FCB 首址 CX＝要读的记录数	AL＝00 读操作成功 AL＝01 文件结束 AL＝02 缓冲区太小 AL＝03 缓冲区不满 CX＝已读入的块数
28H	随机写若干记录	DS：DX＝打开的 FCB 首址 CX＝要写的记录数	AL＝00 写成功，01 盘满 AL＝02 缓冲区太小 CX＝已写的记录个数
29H	分析文件名	AL＝控制分析标志 DS：SI＝要分析的字符串 ES：DI＝未打开的 FCB	AL＝00 无通配符 AL＝01 有通配符 AL＝FF 驱动器字母无效 ES：DI＝未打开的 FCB
2AH	取系统日期		CX＝年（1980～2099） DH＝月（1～12） DL＝日（1～31） AL＝星期（0～6）
2BH	置系统日期	CX＝年，DH＝月，DL＝日	AL＝0：成功，FF：失败
2CH	取系统时间		CH＝时（0～23） CL＝分（0～59） DH＝秒（0～59） DL＝百分之几秒（0～99）
2DH	置系统时间	CX＝时，分；DX＝秒，百分秒	AL＝0：成功，FF：失败
2EH	设置/复位检验开关	AL＝0：关闭，1：打开	
2FH	取磁盘传输地址：DTA		ES：BX＝DTA 首地址
30H	取 DOS 版本号		AL、AH＝主，次版本号
31H	结束并驻留	AL＝返回码，DX＝驻留节数	
* 32H	取指定驱动器的 DPB	DL＝驱动器号（0＝缺省）	DS：BX＝DPB 首地址
33H	取或置 Ctrl－Break 标志	取标志：AL＝0 置标志：AL＝1，DL＝标志	DL＝标志状态（功能0） （0：关，1：开）
* 34H	取 DOS 中断标志		ES：BX＝DOS 中断标志
35H	取中断向量地址	AL＝中断号	ES：BX＝中断程序入口

（续）

调 用 号	功 能	入 口 参 数	出 口 参 数
36H	取磁盘的自由空间	DL＝驱动器号：0＝缺省 1＝A 驱动器	AX＝FFFF（驱动器无效） 否则，AX＝每簇扇区数 BX＝自由簇数 CX＝每扇区字节数 DX＝文件区所占簇数
* 37H	取/置参数分隔符 取/置设备名许可标志	取分隔符：AL＝0 置分隔符：AL＝1 DL＝分隔符 取许可标志：AL＝2 置许可标志：AL＝3， DL＝许可标志	DL＝分隔符（功能0） 注：缺省值为：2FH（" /"） DL＝许可标志（功能2）
38H	取国家信息	AL＝0，DS：DX＝缓冲区	
39H	创建子目录	DS：DX＝路径字符串	成功：CF＝0 失败：CF＝1，AX＝错误码
3AH	删除子目录	DS：DX＝路径字符串	成功：CF＝0 失败：CF＝1，AX＝错误码
3BH	设置当前目录	DS：DX＝路径字符串	成功：CF＝0 失败：CF＝1，AX＝错误码
3CH	创建文件	DS：DX＝带路径的文件名 CX＝属性： 1—— 只读， 2—— 隐藏，4—— 系统	成功：CF＝0，AX＝文件号 失败：CF＝1，AX＝错误码
3DH	打开文件	DS：DX＝带路径的文件名 AL＝方式： 0—— 读， 1—— 写， 2—— 读/写	成功：CF＝0，AX＝文件号 失败：CF＝1，AX＝错误码
3EH	关闭文件	BX＝文件号	成功：CF＝0 失败：CF＝1，AX＝错误码
3FH	读文件或设备	BX＝文件号 CX＝欲读/写的字节数 DS：DX＝缓冲区	成功：CF＝0 AX＝实际读写的字节数 失败：CF＝1，AX＝错误码
40H	写文件或设备		
41H	删除文件	DS：DX＝带路径的文件名	成功：CF＝0 失败：CF＝1，AX＝错误码
42H	移动文件指针	AL＝方式： 0—— 正向， 1—— 相对，2—— 反向 BX＝文件号 CX：DX＝移动的位移量	成功：CF＝0 DX：AX＝新的指针位置 失败：CF＝1，AX＝错误码
43H	取/置文件属性	取属性：AL＝0 置属性：AL＝1，CX＝新属性 DS：DX＝带路径的文件名	CX＝属性（功能0）： 01H — 只读，02H — 隐藏 04H —系统，20H —归档

（续）

调用号	功能	入口参数	出口参数
44H	设备输入/输出控制： 设置或取得与打 开设备的把柄相关连 信息，或发送/接收控 字符串至设备把柄	取设备信息：AL＝0 置设备信息：AL＝1 读设备控制通道：AL＝2 写设备控制通道：AL＝3 AL＝4（同功能2）AL＝5（同功能3） 取输入状态：AL＝6 取输出状态：AL＝7 BX＝文件号 （功能0～3，6～7） BL＝驱动器号（功能4～5） CX＝字节数（功能2～5） DS：DX＝缓冲区（功能2～5）	失败：CF＝1，AX＝错误码 成功： DX＝设备信息（功能0） AL＝状态（功能6～7） 0：未准备，1：准备 AX＝传送的字节数 （功能2～5）
45H	复制文件号（对于一个 打开的文件返回一个 新的文件号）	BX＝文件号	成功：CF＝0， AX＝新的文件号 失败：CF＝1，AX＝错误码
46H	强行复制文件号	BX＝现存的文件号 CX＝第2文件号	成功：CF＝0 失败：CF＝1，AX＝错误码
47H	取当前目录	DL＝驱动器号 DS：SI＝缓冲区首地址	成功：CF＝0 失败：CF＝1，AX＝错误码
48H	分配内存	BX＝所需内存的节数	成功：CF＝0 AX＝分配的段数 失败：CF＝1，AX＝错误码 BX＝最大可用块大小
49H	释放内存	ES＝释放块的段值	成功：CF＝0 失败：CF＝1，AX＝错误码
4AH	修改内存分配	ES＝修改块的段值 BX＝新长度（以节为单位）	成功：CF＝0 失败：CF＝1，AX＝错误码 BX＝最大可用块的大小
4BH	装载程序 执行程序	AL＝0（装载并执行） AL＝1（获得执行信息） AL＝3（装载但不执行） DS：DX＝带路径的文件名 ES：BX＝装载用的参数块	成功：CF＝0 失败：CF＝1，AX＝错误码
4CH	带返回码的结束	AL＝进程返回码	
4DH	取返回码： 取得由另一进程， 通过功能调用4CH/31H 指定的返回码		AL＝进程返回码 AH＝返回码 1—— Ctrl _ Break 结束 2—— 严重设备错结束 3—— 调用31H 而结束 0—— 正常结束

（续）

调 用 号	功 能	入 口 参 数	出 口 参 数
4EH	查找第 1 个匹配项	DS：DX＝带路径的文件名 CX＝属性	成功：CF＝0 失败：CF＝1，AX＝错误码
4FH	查找下 1 个匹配项		成功：CF＝0 失败：CF＝1，AX＝错误码
* 50H	建立当前的 PSP 段地址	BX＝PSP 段地址	
* 51H	读当前的 PSP 段地址		BX＝PSP 段地址
* 52H	取 DOS 系统数据块首址		ES：BX＝系统数据块首址
* 53H	为块设备建立 DPB	DS：SI＝BPB，ES：BP＝DPB	
54H	取检验开关设定值		AL＝标志值（0：关，1：开）
* 55H	由当前 PSP 创建新 PSP	DX＝PSP 段地址	
56H	文件换名	DS：DX＝带路径的旧文件名 ES：DI＝带路径的新文件名	成功：CF＝0 失败：CF＝1，AX＝错误码
57H	取/置文件日期和时间	取：AL＝0，BX＝文件号 置：AL＝1，BX＝文件号， CX＝时间，DX＝日期	成功：CF＝0 CX＝时间，DX＝日期（取） 失败：CF＝1，AX＝错误码

参 考 文 献

1 张怀莲编．IBM PC(Intel 8086/8088)宏汇编语言程序设计．北京：电子工业出版 社，1987

2 朱慧真编．汇编语言教程．北京：国防工业出版社，1988

3 张载鸿编．局部网操作系统 DOS 高级技术分析．北京：国防工业出版社，1988

4 张钟恩编著．IBM — PC/XT、长城0520微型机实用程序设计．北京：中国计量出版社，1988

5 张福炎等编著．微型计算机 IBM PC 的原理与应用．南京：南京大学出版社，1984

6 唐先余编著．实用 DOS 技术．成都：四川大学出版社，1989

7 毛 明编著．实用软件技术．北京：电子工业出版社，1991

[General Information]
书名=宏汇编语言程序设计编程指导
作者=毛明
页数=245
SS号=10204970
出版日期=1993年7月第1版