目　　录

第一章　基础知识 1
　第一节　为什么要用汇编语言编写程序 1
　第二节　8086/8088 CPU 的功能结构 1
　第三节　8086/8088 CPU 的寄存器结构 3
　第四节　堆栈与存贮器结构 4
　第五节　数字、字符编码 6
　　习题一 7
第二章　8086/8088 的指令系统 9
　第一节　寻址方式 9
　第二节　标志寄存器 13
　第三节　指令系统 17
　　习题二 33
第三章　汇编语言与汇编程序 35
　第一节　汇编语言语句 35
　第二节　结构 51
　第三节　记录 53
　第四节　宏指令语句 55
　　习题三 60
第四章　程序设计的基本方法 62
　第一节　汇编语言程序设计的基本步骤 62
　第二节　程序的基本结构形式 66
　第三节　分支程序设计 68
　第四节　循环程序设计 79
　第五节　子程序设计 91
　第六节　DOS 系统功能调用 100
　第七节　磁盘文件管理 106
　　习题四 121
第五章　输入输出和中断 122
　第一节　输入和输出 122
　第二节　中断 131
　第三节　ROM BIOS 中断调用 137

第四节 PC之间、PC与Z80之间的通信 142
习题五 170
第六章 程序设计的一些技法 171
第一节 字符处理 171
第二节 代码转换 179
第三节 表的处理和应用 192
第四节 算术运算 203
第五节 浮点数运算 217
第六节 图形显示 219
第七节 声音与乐曲 229
习题六 233
第七章 软件开发 235
第一节 问题的定义 235
第二节 模块化结构化程序设计 237
第三节 查错和测试 241
第四节 文件编制与维护 242
习题七 244
第八章 汇编语言程序的上机过程 245
第一节 基本概念 245
第二节 汇编语言程序的上机过程 251
第三节 行编辑与全屏幕编辑程序 252
第四节 汇编与宏汇编程序 263
第五节 连接程序 264
第六节 调试程序 265
第七节 运行程序 276
习题八 278
附录 A 指令系统综述与查阅表 279
附录 B IBM PC ASCII 码字符表 290
参考资料 291

前　言

随着微型计算机事业的迅速发展，大批 IBM PC 及其兼容机进入我国各行各业，各方面的应用、开发、维修、生产正在逐步深入。高等院校中也大量采用了 IBM PC 系列机为教学服务。本书原来主要是为计算机及有关专业的学生编写的教材，经部分兄弟院校和培训班使用，在广泛听取教师及科研人员意见的基础上，考虑到不同年级教学的需要、也考虑到不同用户和科技人员学习掌握汇编语言程序设计方法的需要，对本书作了修改和补充。

本书涉及的面比较广，可以选择性地讲用或阅读。对于低年级的学生，可选择第一章、第二章、第三章的第一节、第四章的一至六节、第五章的一至二节、第六章和第八章，第七章供参考，目的是扩大学生的知识面；对于高年级学生可以选择第一章的二至四节、第二章、第三章、第四章的六至七节、第五章、第六章和第八章。第八章最好与上机实习结合讲用。培训班、用户及科技人员可根据需要随意选用。

本书编写过程中参考了周明德老师主编的《微型机 IBM PC (0520)系统原理与应用》一书(油印稿)，并征得同意，引用了其中的部分内容。编者对周明德老师的热情支持和帮助表示真诚的感谢。

在编写本书的过程中，得到了北京计算机学院、计算机技术系、微型机室的领导和老师们的大力支持和帮助；北方交通大学徐悦、浙江大学等兄弟院校的老师，航天部 23 所、煤炭部综采中心等科研单位的工程技术人员提出了宝贵意见；轻工部科学研究院李国俊、北京文献服务社徐仁尧、空军学院徐洪才、兵器部计算所谢南萍等老师和科技人员给与了很大支持和帮助。借此机会，编者向诸位表示热情的感谢。

承蒙清华大学计算机系朱家维教授在百忙之中为本书审阅，热情指导，并提出宝贵意见，深表感谢。

因时间紧迫，而且编者水平有限、经验不足，缺点错误之处定有不少，敬请读者指正，以待改进。

编者

第一章　基础知识

第一节　为什么要用汇编语言编写程序

采用高级语言（例如 BASIC,Pascal,FORTRAN 等）编写的程序，机器是不能直接执行的，需要由编译程序或解释程序将它翻译成对应的机器语言程序，机器才能接受。通过编译或解释程序生成的机器语言程序往往比较冗长，占用存贮空间较大，执行起来速度慢。高级语言的程序员无法直接利用机器硬件系统的许多特性，例如寄存器、标志位以及一些特殊指令等，影响许多程序设计技巧的发挥。而汇编语言程序是直接利用机器提供的指令系统编写的程序，它与机器语言程序一一对应，因此占用存贮空间少，执行速度快。

用汇编语言编程可以充分发挥机器硬件的功能并提高编程的质量。为此学习汇编语言程序设计首先应该熟悉机器的指令系统。而指令系统又是与具体机器的内部结构密切相关的，因此要熟悉机器的内部结构，特别是中央处理器（ CPU ）和存贮器的结构。还应熟悉机器中与编程有关的其它部分的结构，例如中断系统、视频显示、键盘接收、定时功能等等。另外我们还必须知道系统为我们提供的软件环境，如放在ROM 中的监控程序、存在磁盘上的操作系统、汇编程序、调试程序等。我们可以充分利用它们的支持，直接调用其中的子程序等等。

是否采用汇编语言编写程序，要看具体的应用场合，在软件的开发时间及软件的质量方面进行权衡和抉择。一般来说某些对执行时间和存贮器容量要求较高的程序采用汇编语言编写，如实时控制系统、智能化仪器仪表及高性能软件等方面。

第二节　8086/8088 CPU 的功能结构

本章我们将首先介绍 8086/8088 CPU的内部结构和存贮器结构，作为学习汇编语言程序设计的基础知识。

Intel 8086/8088是两种第三代微处理器。在汇编语言一级，它们与 8080/8085 是兼容的。它们有 20 条地址线，直接寻址能力达 1MB。8088 具有 8 位数据通道可以与存贮器或输入输出交换信息。而 8086 则为 16 位数据通道。其它方面，两个处理器都是相同的，为其中一个 CPU 写的软件可以不需修改地在另一个 CPU 上执行。 IBM PC系列机及兼容机上广泛地采用了 8088。

8086/8088 CPU 就功能而言分成两大部分：总线接口单元 BIU (Bus Interface Unit) 和执行单元 EU (Execution Unit)。如图 1-1 所示。

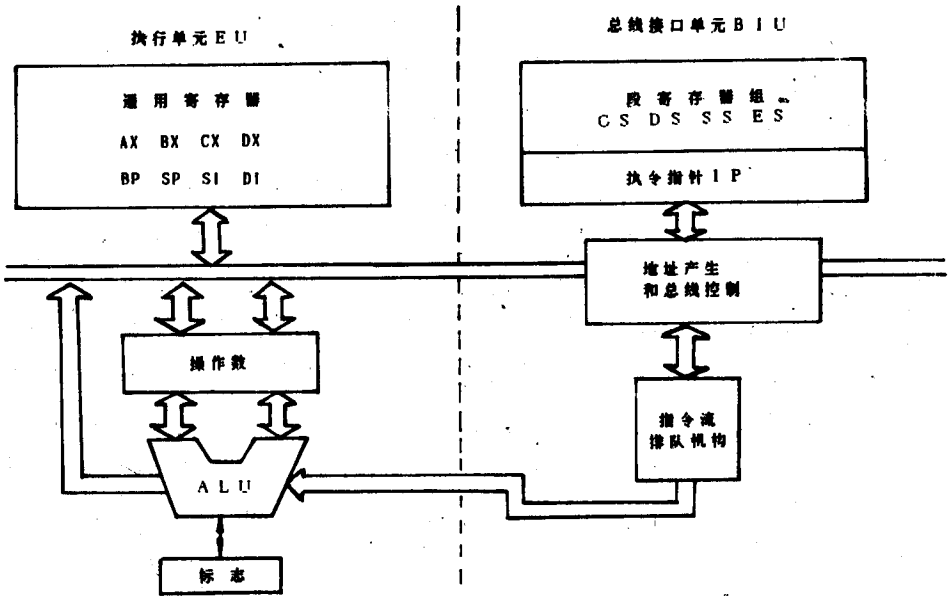


图 1-1 8086/8088 CPU的功能结构

BIU负责 8086/8088 CPU 与存贮器和外部设备之间的信息传送。具体地说，即 BIU负责从内存指定部分取出指令送至指令流排队机构中排队，在执行指令时，所需的操作数，也由 BIU 从内存的指定区域取出，传送给 EU 部分去指行。

EU负责指令的执行，并进行算术逻辑运算等。由于取指令部分和执行指令部分是分开的，所以在一条指令的执行过程中，就可以取出一条（ 或多条 ）指令，放在指令流队列中排队。当一条指令执行完以后就可以立即执行下一条指令，减少了 CPU为取指令而等待的时间，提高了CPU 的利用率、加快了系统的运行速度。另一方面又降低了与之配合的存贮器的存取速度的要求。

如前所述，在 8080/8085 等标准的 8 位微处理器中，程序的执行顺序为取第一条指令，执行第一条指令；取第二条指令，执行第二条指令；・・・ 直至取最后一条指令，执行最后一条指令。这样，在每一条指令执行完以后，CPU 必须等到下一条指令取出来以后才能执行。它的工作顺序如图 1-2 所示。

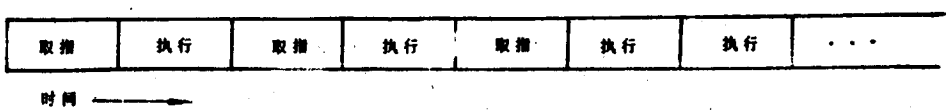


图 1-2 8 位微处理器的程序执行过程

但在 8086/8088 中，由于 BIU 和 EU 分开，所以，取指和执行可以重迭进行，它的执行顺序如图 1-3 所示。

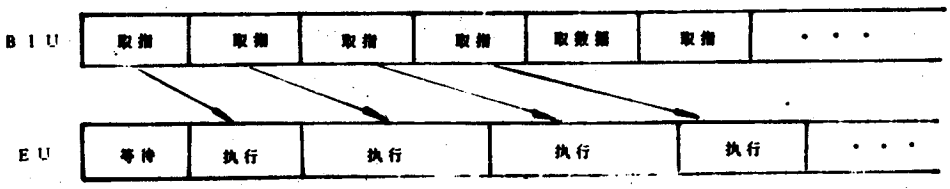


图 1-3 8086/8088 程序的执行过程

第三节 8086/8088 CPU 的寄存器结构

8086/8088 的寄存器结构如图 1-4 所示。

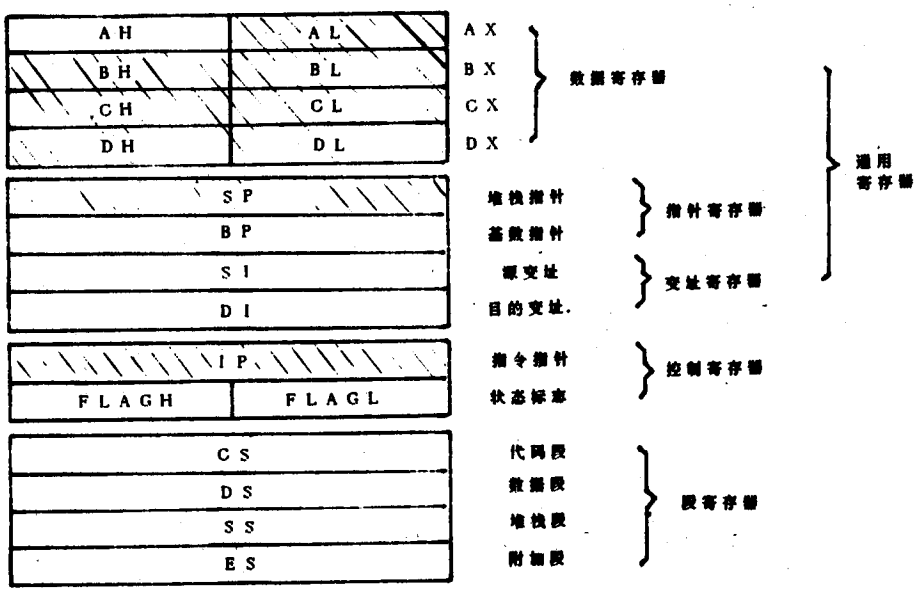


图 1-4 8086/8088 的寄存器结构

AX、BX、CX、DX 4个数据寄存器是 16 位的，其中 AX 叫累加器，它们的通常用途可以用表 1-1 说明。

表 1-1

寄存器	通 常 用 途
AX	字乘法、字除法、字 I/O
AL	字节乘法 字节除法 字节 I/O 转移 十进制算术运算
AH	字节乘法 字节除法
BX	转移
CX	串操作 循环次数
CL	变量移位或循环
DX	字乘法 字除法 间接 I/O

8086/8088 也能处理 8 位数，图 1-4 中的 4 个 16 位数据寄存器也可以作为 8个 8 位寄存器使用，图中打斜线的部分即相当于 8080/8085 中的通用寄存器。

8086/8088 中的堆栈指针 SP (Stack Pointer)类似于 8080 和 8085 中的堆栈指针，用于确定在堆栈操作时，堆栈在内存中的位置。但在 8086/8088 中 SP 还必须与 SS（堆栈段寄存器）一起才能确定堆栈的实际位置。

在 8086/8088 中增加了三个 16 位寄存器，即基数指针寄存器 BP（Base Pointer Register）、源变址寄存器 SI（Source Index Register）和目的变址寄存器 DI（Destination Index Register），还增加了几种寻址方式，从而能更灵活地寻找操作数。

8086/8088 中的指令指针 IP（Instruction Pointer）类似于 8080/8085中的程序计数器 PC。但是它们也略有不同 8080/8085 中的 PC 指向下一条即将要执行的指令，而在 8086/8088 中 IP 则指向下一次要取出的指令，另一方面，IP 要与 CS 寄存器相配合才能形成真正的物理地址。

8086/8088 中的标志寄存器占用两个字节，共有九个标志位（保留了 8080/8085 中的 5 个标志）。

此外 8086/8088 中还有 4 个 16 位的段寄存器 CS、DS、SS、ES，使 8086/8088 能在 1MB 的范围内对内存进行寻址。

有关本节所涉及到的寄存器及其使用，分别在以后有关章节中说明。

第四节 堆栈与存贮器结构

一.堆栈

堆栈是在内存RAM 中开辟的一端固定一端活动的存贮空间。活动端叫栈顶，固定端叫栈底。数据遵从先进后出的原则。所有信息的存入和取出都从栈顶进行，CPU 中有一个栈指针寄存器 SP，它始终指向堆栈的顶部(即栈顶的地址)。SP 的初值(即栈底)的设置可以由指令 MOV SP，im 来实现（im 是 16 位立即数)。堆栈主要用来进行现场数据保护，子程序与中断服务程序的调用返回等。

堆栈操作的指令分为两类，即数据进栈指令 PUSH 和出栈指令 POP。

(1) 进栈指令 PUSH

PUSH 指令可以将通用寄存器、段寄存器中的一个字推进栈顶。例如：

PUSH AX 与 PUSH BX

指令的执行分为两步：第一步先 SP-1→SP，然后把寄存器中的高位字节（如 AH）送至 SP 所指的单元。第二步再次使 SP-1→SP，把寄存器的低位字节（如 AL）送至SP所在单元。如图 1-5 所示。

随着推入内容的增加，堆栈扩展，SP 值减小，但每次操作完 SP 总指向栈顶。

(2) 出栈指令 POP

出栈指令 POP 将现行 SP 所指的栈顶的一个字传送至段寄存器或通用寄存器。

例如： POP AX

它的操作步骤是先将栈顶内容送入 AX寄存器的低位字节，再 SP+1→SP，然后再将栈顶内容送入 AX 寄存器的高位字节，再 SP+1→SP。

二．存贮器结构

8086/8088 有 20 条地址线，它的直接寻址能力为 1MB（2 的 20 次方 ）。所以在一个系统中，可以有多达 1MB 的存贮器，地址从 00000H 到 FFFFFH。任意给定的一个 20 位地址，就可以从这 1MB 中取出所需要的指令和操作数。如前所述，8086/8088 CPU 内部的 ALU 只能进行 16位运算。与地址有关的寄存器如 SP、IP ，以及 BP、SI、DI 等也都是 16 位的。因而对地址的运算也只能是 16 位。这就是说，对于 8086/8088来说，各种寻址方式，寻找操作数的范围最多可能是 64KB 。那么，它的 20位地址又是如何形成的呢？它是将整个 1MB存贮器以64KB为范围分为若干段。在寻址一个具体物理单元时，必须由一个基本地址再加上由 SP或 IP 或 BP 或 SI 或 DI 等可由 CPU 处理的 16 位偏移量来形成实际的 20 位物理地址。这个基本地址就是由 8086/8088 中的段寄存器，即代码段寄存器 CS、堆栈段寄存器 SS、数据段寄存器 DS 以及附加段寄存器 ES 中的一个来形成的，在形成 20位物理地址时，段寄存器中的 16 位数会自动左移 4 位，然后与 16位偏移量相加。如图 1-6 所示。

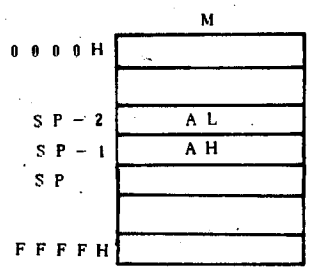


图 1-5 堆栈操作示意图

每次需要产生一个 20 位地址的时候，一个段寄存器会自动被选择。且能自动左移4位再与一个16 位地址偏移量相加，产生所需要的 20 位物理地址。

当取指令的时候，则自动选择代码段寄存器 CS ，再加上由 IP 所决定的 16 位偏移量，计算得到要取的指令的物理地址。

当涉及到一个堆栈操作时，则自动选择堆栈段寄存器 SS，再加上由SP 所决定的16位偏移量，计算得到堆栈操作所需要的 20 位物理地址。

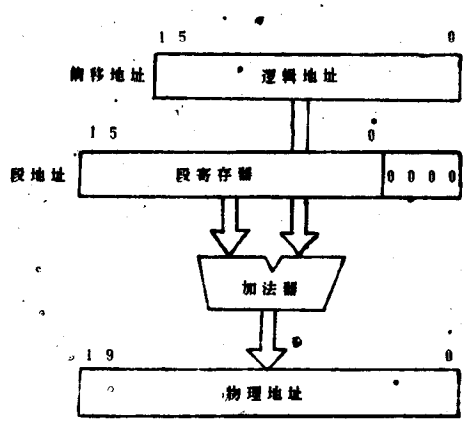


图 1-6 8086/8088 中物理地址的形成

当涉及到一个操作数时，则自动选择数据段寄存器 DS 或附加段寄存器 ES ，再加上 16 位偏移量，计算得到操作数的 20 位物理地址。 16 位偏移量，可以是包含在指令中的直接地址，也可以是某一个 16 位地址寄存器的值，也可以是指令中的偏移量加上 16 位地址寄存器中的值等等，取决于指令的寻址方式。

在 8086/8088 系统中，存贮器的访问，如图 1-7 所示。

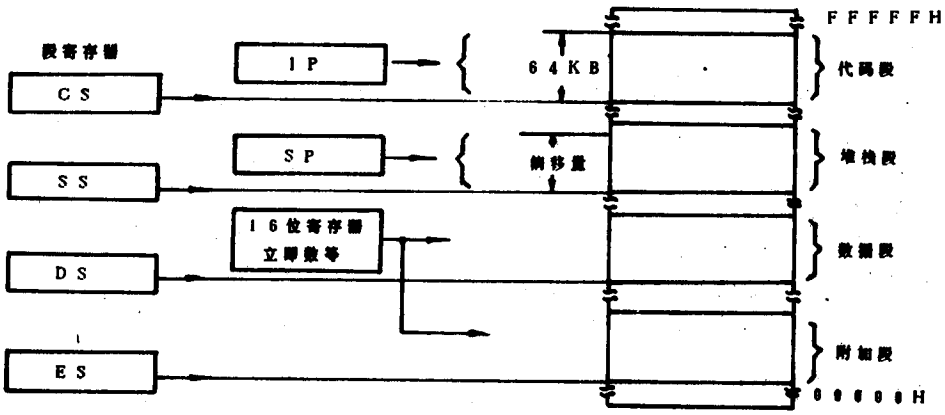


图 1-7 8086/8088 存贮器结构

所以，在不改变段寄存器值的情况下，寻址的最大范围是 64KB ，若有一个任务，它的程序长度、堆栈长度，以及数据区长度都不超过 64KB 的话，则可在程序开始时，分别给 DS、SS、CS 置值，然后在程序中就可以不再考虑这些段寄存器，程序就可以在各自的区域中正常地进行工作。若某一个任务所需的总的存贮器长度（包括程序长度、堆栈长度和数据长度等）不超过 64KB ，则可在程序开始时使 CX、SS、DS 相等，程序也能正常工作。

这种存贮器分段的方法，对于一个程序中要用的数据区超过 64KB 或要求从两个（或多个）不同的区域中存取操作数时，只要在取操作数以前，用指令给数据段寄存器重新赋值就可以了。

上述的存贮器分段方法，对于要求在程序区、堆栈区和数据区之间隔离时是非常方便的。这种分段方法也适用于程序的再定位要求。在不少情况下，要求同一个程序能在内存的不同区域中运行，而不改变程序本身，这在 8086/8088中是可行的。只要使程序中的转移指令都为相对转移指令，而在运行这个程序前设法改变各个段寄存器的值就可以了。

第五节 数字、字符编码

一.ASCII 码

在计算机中，数字是用二进制表示的。而计算机处理的问题中不仅仅是数字，还包括各种字符， 如大小写英文字母、标点符号、运算符号等等。 为了计算机能识别和处理它们，这些字符应如何表示呢？由于计算机中的基本物理器件是具有两个状态的器件，所以各种字符只能按特定的规则用若干位二进制码的组合来表示。编码可以有各种方式（即规定），但要考虑通用问题。目前在微型计算机中普遍采用的是 ASCII 码（AmericanStandard Code for Information Interchange 美国标准信息交换码），IBM PC ASCII码字符

表见附录 B。

它是用八位二进制数编码，故可以表示 256 个字符。表中最上两行是位 7 到位 4的代码。最左边的两列是位 3 到位 0 的代码。用此表，可实现字符与其 ASCII码的互查。从表中可以看出数字 0 到 9 对应的 ASCII 码是 30H 到 39H，英文大写字母 A～Z 对应的 ASCII 码是 41H 到 5AH ，英文小写字母 a～ z 的 ASCII 码是 61H 到7AH 。整个表以中间为界分为左、右两部分。左边 128 个字符的 ASCII码值的最高位为 0，右边128个字符的 ASCII 码值的最高位为 1 。256 种字符大体可以分为如下几类 ：

- 16 个专用的游戏符号
- 15 个用于文字处理编辑的符号
- 96 个常用 ASCII 字符
- 48 个外语字符
- 48 个商用图形符号
- 16 个常用希腊字母
- 15 个常用科学符号

二．BCD 码

虽然二进制数实现容易、可靠，二进制运算规律十分简单，但是二进制数不直观、书写麻烦、易错。于是在计算机输入输出时通常还是采用人们习惯的十进制数表示。这就产生了二进制编码的十进制数，称为 BCD 码（ Binary Code Decimal ）。

一位十进制数用四位二进制数编码表示，表示的方法可多种，通常用的是 8421 BCD 码。其特点是：

(1) 8421 BCD 码的十个十进制数字 0 到 9 分别用四位二进制数表示。四位二进制数有 16 种组合，取前十种，即用 0000、0001、・・・ 、1001 分别表示 0 到 9。

(2) 每组四位二进制数之间是二进制的 ，而组与组之间（即 BCD 码的十进制数之间）是十进制的。

例如：

98D=(1001 1000)BCD

16D=(0001 0000)B=(0001 0110)BCD

可以从 (0100 1001.0110 1001)BCD 方便地认出是十进制数的 49.69。很容易实现十进制数与 BCD 码的转换。关于 BCD 码与二进制间的转换，手算比较费事，可以由专门的指令或程序实现。

习题一

1．8086/8088 CPU 中有哪些寄存器？如何分组？各有什么用途？

2．设堆栈指针 SP 的初值为 2000H，AX=3000H，BX=5000H，试问：

(1) 执行指令 PUSH AX 后，SP=?

(2) 再执行 PUSH BX 及 POP AX 后，SP=? AX=? BX=?

并画出堆栈变化示意图。

3. 8086/8088系统中，存贮器的物理地址由哪两部分组成？每一个段与寄存器之间有何对应要求？

第二章　8086/8088 的指令系统

本章介绍8086/8088 的部分指令、寻址方式和有关的标志寄存器，以便进一步学习程序设计的方法。

第一节　寻址方式

寻址方式就是指令中用于说明操作数所在地址的方法。8086/8088 的基本寻址方式有六种。

一.立即寻址（Immediate Addressing）

这种寻址方式所提供的操作数直接包含在指令中。它紧跟在操作码的后面，与操作码一起放在代码段区域中。如图 2-1 所示。

例：MOV AX，im

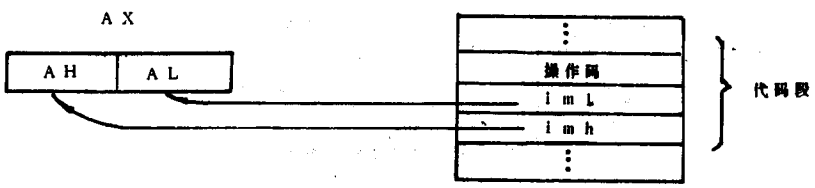


图 2-1 立即寻址示意图

立即数 im 可以是 8 位的，也可以是 16 位的。若是 16 位的，则 imL 在低地址字节，imH 在高地址字节 。若是字操作数 ，而且它的高位字节是由低位字节符号扩展的，则在指令中的立即数，只具有低位字节。

立即寻址主要是用来给寄存器或存贮器赋初值。

二.直接寻址（Direct Addressing）

操作数地址的16位偏移量直接包含在指令中，它与操作码一起存放在代码段区域。操作数一般在数据段区域中，它的地址为数据段寄存器 DS 加上这 16 位地址偏移量。如图 2-2 所示。

例如：　MOV　AX，DS:[2000H]

指令中的 16 位地址偏移量是低位字节在前，高位字节在后。

这种寻址方法，是以数据段的地址为基础，故可在多达64KB的范围内寻找操作数。

在 8086/8088中允许段超越。即对于寻找操作数来说，还允许操作数在以代码段、堆栈段或附加段为基准的区域中，即只要在指令中指明是段超越（详见第三章）的，则16位

地址偏移量可以与 CS 或 SS 或 ES 相加，作为操作数的地址。

MOV AX, DS:[2000H]

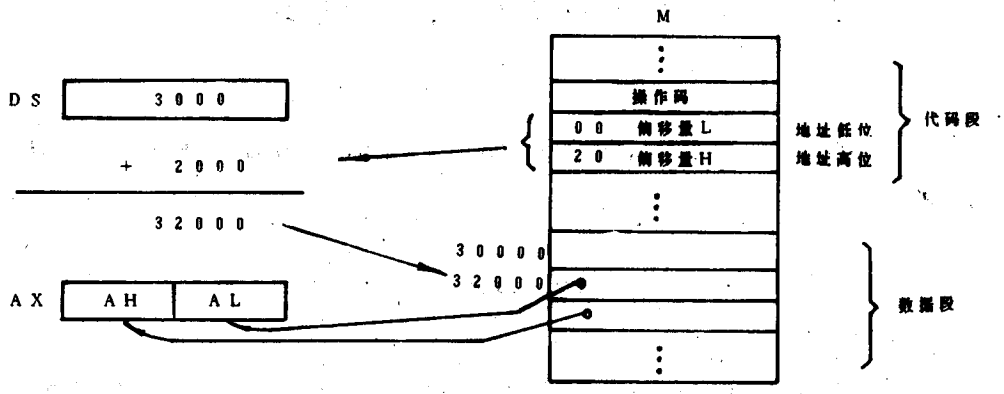


图 2-2 直接寻址示意图

三.寄存器寻址 (Register Addressing)

操作数包含在 CPU 的内部寄存器中，例如寄存器 AX、BX、CX、DX等，如图 2-3 所示。

例: MOV DS , AX

图 2-3 寄存器寻址方式示意图

虽然操作数可在CPU 内部通用寄存器的任意一个中，且它们都能参与算术或逻辑运算和存放运算结果，但是， AX 是累加器，若结果是存放在 AX 中的话，则通常指令执行时间要短些。

四.寄存器间接寻址 (Register Indirect Addressing)

在这种寻址方式中，操作数是在存贮器中。但是，操作数地址的 16 位偏移量包含在以下四个寄存器 SI、DI、BP、BX 之一中。这又可以分成两种情况:

1.若以 CI、DI、BX 间接寻址，则类似于 8080 中的 HL 间接寻址，通常操作数在现行数据段区域中，即数据段寄存器 DS 加上 SI、DI、BX 中的 16 位偏移量，为操作数的地址，如图 2-4 所示。

例: MOV AX, [SI]

2.若是以寄存器 BP 间接寻址，则操作数在堆栈段区域中，即堆栈段寄存器 SS 与 BP相加作为操作数的地址，如图 2-5 所示。

例 : MOV AX, [BP]

若在指令中规定是段超越的，则 BP 也可以与其它的段寄存器相加，形成操作数地址。

MOV AX, [SI]

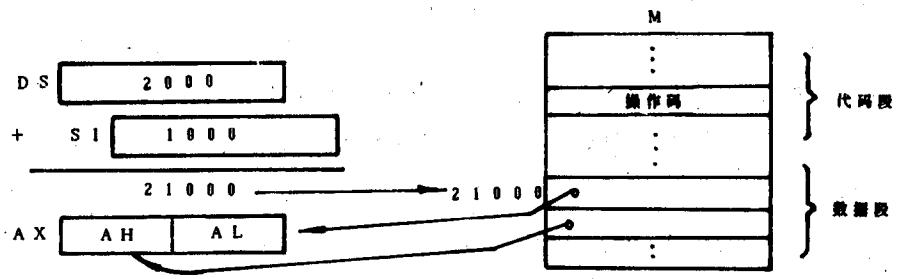


图 2-4 寄存器间接寻址示意图

MOV AX, [BP]

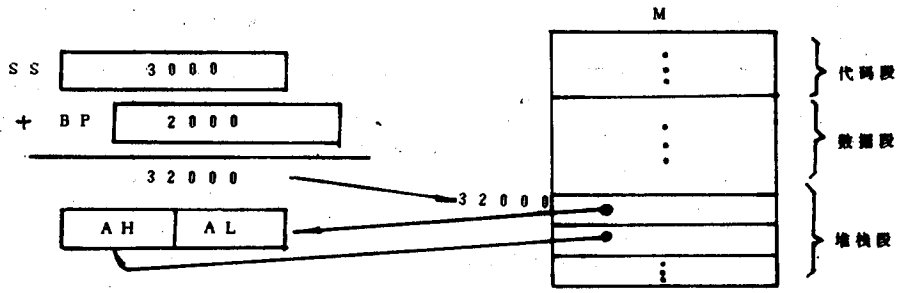


图 2-5 以 BP 作为间接寻址示意图

五.变址寻址 (Index Addressing)

所谓变址寻址就是由指定的寄存器内容，加上指令中给定的 8 位或 16 位偏移量（当然要由一个段寄存器作为地址基准)作为操作数的地址。

上述可以作为寄存器间接寻址的四个寄存器SI、DI、BX、BP也可以作为变址寻址。如图 2-6 所示。

例: MOV AX，COUNT [SI]

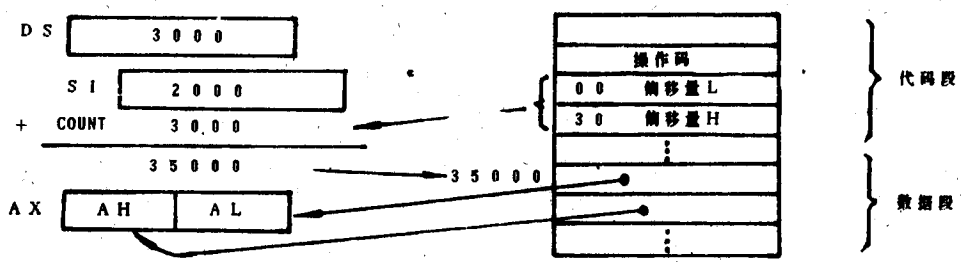


图 2-6 变址寻址示意图

在正常情况下，若用 SI、DI 和 BX 作为变址，则与数据段寄存器相加，形成操作数的地址；若用 BP 变址，则与堆栈段寄存器相加，形成操作数的地址。

但是，只要在指令中指定是段超越的，则也可以用别的段寄存器作为地址基准。

六.基址加变址寻址

在 8086/8088 中，通常把 BX和 BP看作是基址寄存器，把 SI、DI看作变址寄存器，可以把这两种寻址方式组合起来形成一种新的寻址方式。这种寻址方式是把一个基址寄存器（BX 或 BP）的内容加上一个变址寄存器（SI 或 DI）的内容，再加上指令中指定的 8 位或 16 位偏移量（ 当然要以一个段寄存器作为地址基准 ）作为操作数的地址，如图2-7 所示。

例如： MOV AX，MASK [BX] [SI]

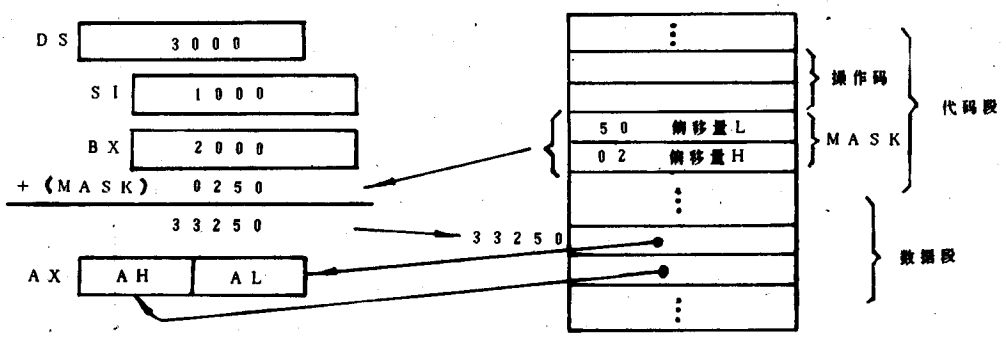


图 2-7 基址加变址寻址示意图

在正常情况下，由基地址决定哪一个段寄存器作为地址指针。即若用BX作为基地址，则操作数在数据段区域中；若用 BP 作为基地址，则操作数在堆栈段区域中，但若在指令中规定段是超越的，则可用其它段寄存器作为地址基准。

如上所述，8086/8088 中的存贮器是分段的，我们寻找一个内存操作数，只能在某一个段的 64KB 范围内寻找，以什么寄存器间址、变址与基址加变址，则操作数在什么段区域中，在 8086/8088 中有一个基本约定。 只要在指令中不特别说明要超越这个约定，则正常情况就按这个基本约定来寻找操作数，这就是所谓的 Default(默认)状态。这些基本约定和允许超越的情况如表 2-1 所示。

表 2-1

存贮器存取方式	约定段基数	可修改的段基数	逻辑地址
取指令	CS	无	IP
堆栈操作	SS	无	SP
源串	DS	CS、ES、SS	SI
目的串	ES	无	DI
用 BP 作为基寄存器	SS	CS、DS、ES	有效地址
通用数据读写	DS	CS、ES、SS	有效地址

第二节　标志寄存器

标志寄存器反映系统的状态及运算结果的特点。8086/8088 中有一个标志寄存器，占用 2 个字节，共有 9 个标志位，如图 2-8 所示。

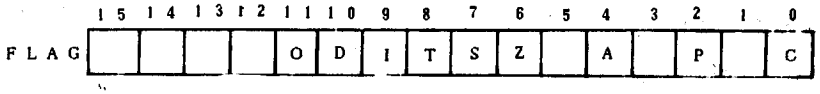


图 2-8 8086/8088 中的标志位

一.标志位

各个标志位的功能分述如下：

1.辅助进位标志 AF (Auxitiary Carry Flag)

在字节操作时若由低半字节(一个字节的低 4 位) 向高半字节有进位或借位；在字操作时，低位字节向高位字节有进位或借位，则 AF=1 ，否则为 0。这个标志用于十进制算述运算指令中。

2.进位标志 CF (Carry Flag)

当结果的最高位（字节操作时的 D7 或字操作时的 D15）产生一个进位或借位，则CF=1，否则为 0。这个标志主要用于多字节数的加、减法运算。移位和循环移位指令也能够把存贮器或寄存器中的最高位（左移时）或最低位（右移时）放入标志 CF 中。

3.溢出标志 OF (Overflow Flag)

在算术运算中，带符号数的运算结果超出了8 位或 16 位带符号数能表达的范围，即在字节运算时 >+127 或 <-128，在字运算时>32767 或 <-32768，此标志置位。一个任选的溢出中断指令，在溢出情况下能产生中断。

溢出和进位是两个不同性质的标志，千万不能混淆了。例如在字节运算时：

```
        MOV AL，64H
        ADD AL，64H
即:       01100100
        + 01100100
        ----------
          11001000
```

D7位向前无进位，故运算后CF=0；但运算结果超过了+127，此时，溢出标志 OF=1。

又例如，在字节运算时：

```
        MOV AL，0ABH
        ADD AL，0FFH
```

即:

```
    10101011   (-85)
+   11111111   (-1)
-------------------
  1 10101010
```

D7位向前有进位，故运算后 CF=1 ；但运算的结果又不小于 -128 ，此时，溢出标志 OF=0。

在字运算时，如有

```
        MOV   AX,0064H
        ADD   AX,0064H
```

即：

```
    00000000 01100100
+   00000000 01100100
-------------------------
    00000000 11001000
```

D15 位未产生进位，故 CF=0；运算结果显然未超 +32767 故 OF=0。

但若有

```
        MOV AX,6400H
        ADD AX,6400H
```

即：

```
    01100100 00000000
+   01100100 00000000
-------------------------
    11001000 00000000
```

D15 位未产生进位，故 CF=0。但运算结果不小于 +32767 溢出标志 OF=1。

又例如：

```
        MOV AX,0AB00H
        ADD AX,0FFFFH
```

即：

```
    10101011 00000000
+   11111111 11111111
-------------------------
    110101010 11111111
```

D15 位产生进位，故 CF=1，但运算结果不小于 -32768，故 OF=0。

4.符号标志 SF (Sign Flag)

它的值与运算结果的最高位相同。即结果的最高位（字节操作时为 D7，字操作时为 D15）为 1，则 SF=1；否则 SF=0。

由于在 8086/8088 中符号数是用补码表示的，所以 SF 表示了结果的符号，SF=0 为正，SF=1 为负。

5.奇偶标志 PF (Parity Flag)

若操作结果中"1" 的个数为偶数，则 PF=1，否则 PF=0。这个标志可用于检查在数据传送过程中是否发生错误。

6.零标志 ZF (Zero Flag)

若运算的结果为 0，则 ZF=1，否则 ZF=0。

8086/8088 还提供了三个控制标志，它们能由程序来置位和复位，以变更处理器的操作。

7.方向标志 DF (Direction Flag)

若用指令置DF=1，则引起串操作指令为自动减量指令，也就是从高地址到低地址处理字符串；若使 DF=0，则串操作指令就为自动增量指令。

8.中断允许标志 IF (Interrupt-enable Flag)

若指令中置 IF=1，则允许 CPU 去接收外部的可屏蔽中断请求；若使IF=0，则屏蔽上述的中断请求；对内部产生的中断不起作用。

9.追踪标志 TF (Trap Flag)

置 TF 标志，使处理进入单步方式，以便于调试。在这个方式中，CPU 在每条指令执行以后，产生一个内部中断，允许程序在每条指令执行以后进行检查。

二.标志操作指令

8086/8088 中有一部分指令是专门对标志寄存器或标志位进行操作的。包括四条标志寄存器传送指令和标志位操作指令。

1.标志寄存器传送指令

(1) LAHF (Load AH with flags)

把标志寄存器的低 8 位（包括符号标志 SF、零标志 ZF、辅助进位标志 AF、奇偶标志 PF 和进位标志CF）传送至 AH 的指定位，即相应地传送至位 7、6、4、2 和 0。（位 5、3、1 的内容没有定义），如图 2-9 所示。

这条指令本身不影响这些标志位。

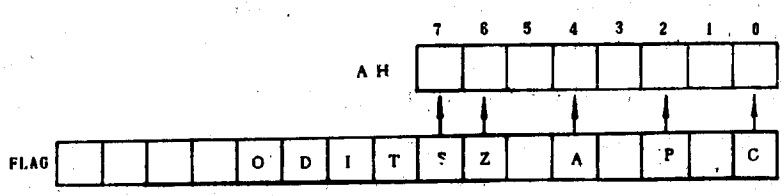


图 2-9 LAHF 指令示意图

(2) SAHF (Store AH into Flags)

这条指令与上一条的操作刚好相反，它是把寄存器 AH 的指定位，传送至标志寄存器的低 8 位的 SF、ZF、AF、PF 和 CF 标志。因而这些标志的内容就要受到影响，这取决于AH中相应位的状态，但这条指令并不影响溢出标志 OF、方向标志 DF、中断屏蔽标志IF 和追踪标志 TF，即不影响标志寄存器的高位字节。

(3) PUSHF (Push Flags)

把整个标志寄存器（包括全部九个标志）推入至堆栈指针所指的堆栈的顶部，同时修改堆栈指针，即 SP-2→SP。

这条指令不影响标志位。

(4) POPF (POP Flags)

这条指令把现行堆栈指针所指的一个字，传送给标志寄存器，同时相应地修改堆栈指针，即 SP+2→SP。

这条指令执行后，8086/8088 的标志位就取决于原堆栈顶部的内容。

PUSHF 和 POPF 这两条指令可以保存和恢复标志寄存器。在子程序调用和中断服务中可利用这两条指令来保护和恢复标志位。

另外，这两条指令也可以用来改变追踪标志TF。在8086/8088 的指令系统中，没有直接能改变TF标志的指令，故若要改变 TF标志，先用 PUSHF 指令把标志位入栈，然后设法改变栈顶存贮单元的 D8 位（把整个标志看成一个字）再用 POPF 指令恢复，这样其余的标志不受影响，而只有 TF 标志按需要改变了。

2.标志位操作指令

8086/8088 有七条直接对标志单独进行操作的指令。其中有三条是针对进位标志 CF 的，有两条是针对标志 DF 的，有两条是针对中断标志 IF 的。

(1) CLC (Clear Carry Flags)

此指令使标志 CF=0

(2) CMC (Complement Carry Flags)

此指令使标志 CF 取反，即若 CF=0，则 1→CF；若CF=1，则 0→CF。

(3) STC (Set Carry Flags)

此指令使标志 CF=1

(4) CLD (Clear direction Flag)

此指令使标志 DF=0，则在串操作指令时，使地址增量。

(5) STD (Set Direction Flag)

此指令使标志 DF=1。则在串操作指令时，使地址减量。

(6) CLI(Clear Interrupt enable Flag)

此指令使中断允许标志 IF=0，于是在 8086/8088 系统中，外部装置送至可屏蔽中断 INTR 引线上的中断请求，CPU 就不予以响应---也即中断屏蔽。但此标志对于非屏蔽中断 NMI 引线上的请求，以及软件中断都没有影响。

(7) STI(Set Interrupt-enable Flag)

此指令使标志 IF=1，则 CPU 就可以响应出现在 INTR 引线上的外部中断请求。

上述 7 条指令除对指定的标志位进行操作外，对其它标志位皆无影响。

第三节　　指令系统

8086/8088 的指令系统可以分为以下六个功能组。

1.数据传送 (Data Transter)

2.算术运算 (Arithmetic)

3.逻辑运算 (Logic)

4.串操作 (String Menipulation)

5.程序控制 (Program Control)

6.处理器控制 (Processor Control)

本节中我们就其中的一部分指令加以介绍，其余部分将结合程序设计方法的讲解分散到各章节中去介绍。为查阅指令方便，书后有附录 A（指令系统综述与查阅表 ），可由此找到各类指令的详述。

一.数据传送指令：

这里主要介绍 MOV、XCHG 和地址传送指令。

1. MOV OPRD1,OPRD2

MOV 是操作码，OPRD1 和OPRD2 分别是目的操作数和源操作数。此指令把一个字节或一个字操作数从源传送至目的。

源操作数可以是累加器、寄存器、存贮器以及立即操作数，而目的操作数可以是累加器、寄存器和存贮器。

数据传送方向的示意图，如图 2-10 所示。

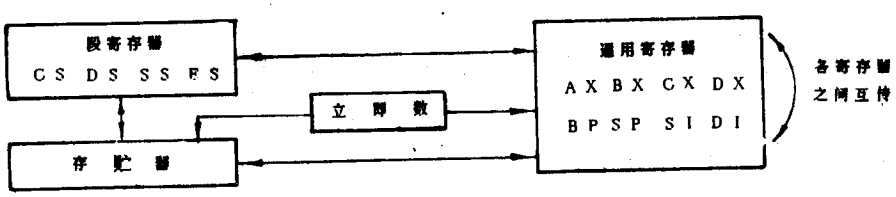


图 2-10 数据传送方向示意图

具体来说，一条数据传送指令能实现：

(1) CPU 内部寄存器之间数据的任意传送（除了代码段寄存器 CS 和指令指针 IP 以外)。例如：

```
MOV AL,BL
MOV DL,CH
MOV AX,DX
MOV CX,BX
MOV DS,BX
```

```
MOV DX,ES
MOV BX,DI
MOV SI,BP
    .
    .
    .
```

(2) 立即数传送至 CPU 内部的通用寄存器组（即 AX、BX、CX、DX、BP、SP、SI、DI），给这些寄存器赋初值。

例如：

```
MOV CL,4
MOV AX,03FFH
MOV SI,057BH
    .
    .
    .
```

(3) CPU 内部寄存器（除了 CS 和 IP 以外）与存贮器（所有寻址方式）之间的数据传送，可以实现一个字节或一个字的传送。

CPU 的通用寄存器可以与存贮器之间实现数据传送。

例如：

```
MOV   AL , BUFFER
MOV   AX , [SI]
MOV   [DI] , CX
MOV   SI , BLOCK[BP]
      .
      .
      .
```

还可以实现寄存器（除 CS 以外）与存贮器之间的数据传送。

例如：

```
MOV DS,DATA[SI+BX]
MOV DEST[BP+DI],ES
      .
      .
      .
```

但是，MOV 指令不能实现存贮单元之间的数据传送。若我们需要把地址（即段内的地址偏移量）为 AREA1 的存贮单元的内容，传送至同一段内的地址为 AREA2 的存贮单元中去。MOV 指令不能直接完成这样的传送，但我们可以CPU 的内部寄存器为桥梁来完成这样的传送。可以采用如下指令：

```
MOV AL,AREA1
MOV AREA2,AL
```

若要求这样传送的不是一个字节，而是一个数据块，例如100个数据，如何实现呢？

当然可以采用与上述类似的 200 条指令来完成 100 个数据的传送。这些指令的操作是重复的（每条指令的地址是变化的）。为了简化程序的编制，当然希望利用循环程序，其困难在于每一循环时要修改地址(源地址和目的地址)，于是就要把地址放在寄存器中，用寄存器间接寻址来寻找操作数，而修改寄存器的内容，就可以实现对地址的修改。再把循环次数的控制部分考虑进去，就可得如下程序：

```
        MOV SI,OFFSET AREA1
        MOV DI,OFFSET AREA2
        MOV CX,100
AGAIN:  MOV AL,[SI]
        MOV [DI],AL
        INC SI
        INC DI
        DEC CX
        JNZ  AGAIN
```

（这里的 AREA1 和 AREA2 是立即数还是存贮单元地址，汇编程序是根据对符号的定义来加以区别的，关于如何对符号进行定义，我们在后面汇编语言一章详细讨论 ）。

其中的增量（INC）、减量（DEC）指令以及转移指令，我们在后面详细介绍。

其中 ，OFFSET AREA1 是指地址单元 AREA1在段内的地址偏移量。如上所述，在8086/8088 中要寻找内存操作数时，必须以段地址（在某个段寄存器中）加上此单元的段内地址偏移量，才能确定某一内存单元的物理地址。

以下在说明指令功能时所举的程序例子中，我们只说明可执行指令部分---即 8086 / 8088指令（我们在程序清单中，给出了必要的伪指令，以便在机器中运行这些程序)。

2.交换指令

```
        XCHG OPRD1,OPRD2
             目的   源
```

这是一条交换指令，把一个字节或一个字的源操作数与目的操作数相交换。交换能在通用寄存器与累加器之间，通用寄存器之间，通用寄存器与存贮器之间进行，但段寄存器不能作为一个操作数。

例如：

```
        XCHG AL,CL
        XCHG AX,DI
        XCHG BX,SI
        XCHG AX,BUFFER
        XCHG BX,DATA[SI]
```

3.地址传送指令

8086/8088 中有三条地址传送指令

(1) LEA (Load Effective Address)

例: LEA OPRD1,OPRD2

此指令把源操作数 OPRD2 的偏移量传送至目的操作数OPRD1。源操作数必须是一个内存操作数，目的操作数必须是一个 16 位的通用寄存器。这条指令通常用来建立串操作指令所须的寄存器指针。例如指令:

LEA BX,BUFR

将把变量 BUFR 的地址偏移量部分送到 BX。

(2) LDS (Load pointer into DS)

此指令完成一个地址指针的传送。地址指针包括段地址部分和偏移量部分。指令将段地址送入 DS，偏移量部分送入一个 16 位的指针寄存器或变址寄存器。例如指令:

LDS SI,[BX]

将把 BX 所指的 32 位地址指针的段地址部分送入 DS，偏移量部分送入 SI。

(3) LES (Load pointer into ES)

这条指令除将地址指针的段地址部分送入 ES 外，与 LDS 类似。例如:

LES DI,[BX+CONT]

二.算术运算指令

8086/8088 提供加、减、乘、除四种基本算术操作。这些操作都可用于字节或字的运算，也可以用于带符号数与无符号数的运算。带符号数用补码表示。

8086/8088 也提供了各种校正操作，故可以进行十进制算术运算。

参与加、减运算的操作数可如图 2-11 所示。

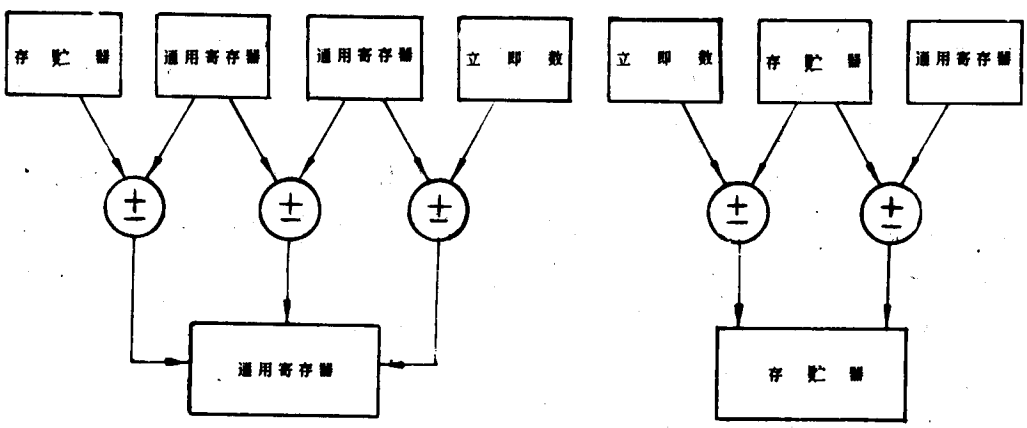


图 2-11 加、减运算的操作数

1.加法指令 (Addition)

(1) ADD OPRD1,OPRD2

这个指令完成两个操作数相加，结果送至目的操作数OPRD1，即 :

OPRD1←OPRD1+OPRD2

目的操作数可以是累加器，任一通用寄存器，以及存贮器操作数。

具体地说可以实现累加器与立即数，累加器与任一通用寄存器，累加器与存贮单元内容相加，和放在累加器中。

例如：
```
ADD AL,30
ADD AX,3000H
ADD AX,SI
ADD AL,DATA[BX]
```

可以实现任一通用寄存器与立即数相加，与累加器或别的寄存器相加，与存贮单元的内容相加，和放在寄存器中。

例如：
```
ADD BX,3FFH
ADD SI,AX
ADD DI,CX
ADD DX,DATA[BX+SI]
    .
    .
    .
```

可以实现存贮器操作数与立即数相加，与累加器或别的寄存器相加，和放至此存贮单元中。例如：
```
ADD BETA[SI],100
ADD BETA[SI],AX
ADD BETA[SI],DX
    .
    .
    .
```

这些指令对标志位 CF、OF、PF、SF、ZF 和 AF 有影响。

(2) ADC (ADd with Carry)

这条指令与上一条指令类似，只是在两个操作数相加时，要把进位标志CF的现行值加上去，结果送至目的操作数。

ADC 指令主要用于多字节运算中。在 8086/8088 中可以进行 8 位运算，也可以进行 16 位运算。 但是 16 位二进制数的表达范围仍然是很有限的，为了扩大数的范围，仍然需要多字节运算。例如，有两个四字节的数相加，加法要分两次进行，先进行低两字节相加，然后再做高两字节相加。在高两字节相加时要把低两字节相加以后的进位考虑进去，就要用到带进位的加法指令 ADC。

若此两个四字节的数，已分别放在自 FIRST 和 SECOND开始的存贮区中，每个数占四个存贮单元，存放时，最低字节在地址最低处，则可用以下程序段实现相加。
```
MOV AX,FIRST
ADD AX,SECOND
```

```
MOV THIRD,AX
MOV AX,FIRST+2
ADC AX,SECOND+2
MOV THIRD+2,AX
```

这条指令对标志位的影响与 ADD 相同。

(3) INC OPRD (INCrement)

这条指令完成对指定的操作数 OPRD 加 1，然后返回此操作数。此指令主要用于在循环程序中修改地址指针和循环次数等。

这条指令执行的结果影响标志位 AF、OF、PF、SF 和 ZF，而对进位标志没有影响。

这条指令的操作数可以是在通用寄存器中，也可以在内存中。

如： INC AL

2.减法指令 (Subtraction)

(1) SUB OPRD1,OPRD2

这条指令完成两个操作数相减，也即从OPRD1中减去 OPRD2，结果放在 OPRD1 中。

具体地说，可以从累加器中减去立即数，或在寄存器和内存操作数中减去立即数；或在寄存器中减去寄存器或内存操作数；或从寄存器或内存操作数中减去寄存器操作数等。

例如：
```
SUB CX,BX
SUB [BP+2],CL
```

(2) SBB(Subtract With Borrow)

这条指令与SUB类似，只是在两个操作数相减时，还要减去借位标志 CF 的现行值。

本指令对标志位 AF、CF、OF、PF、SF 和 ZF 都有影响。

本指令主要用于多字节操作数相减时。

(3) DEC OPRD (DECrement)

本指令对指令的操作数减 1，然后送回此操作数，所用的操作数可以是寄存器，也可以是内存操作数。

在相减时，把操作数作为一个无符号二进制数来对待。 指令执行的结果， 影响标志 AF、OF、PF、SF 和 ZF，但对 CF 标志不影响 (即保持此指令以前的值)。

例如： DEC [SI]

(4) NEG OPRD (NEGate)

这条指令对操作数取补，也即用零减去操作数，再把结果送回操作数。

例如：
```
NEG AL
NEG MULRE
```

若在字节操作时对-128，或在字操作时对-32768取补，则操作数没变化，但溢出标志 O 置位。

此指令影响标志 AF、CF、OF、PF、SF 和 ZF。此指令的结果一般总是使标志 CF=1，

除非在操作数为零时，才使 CF=0。

(5) CMP OPRD1,OPRD2 (CoMPare)

比较指令完成两个操作数相减，使结果反映在标志位上，但并不送回结果。

具体地说，比较指令，可使累加器与立即数，与任一通用寄存器或任一内存操作数相比较。例如：

```
CMP AL,100
CMP AX,SI
CMP AX,DATA[BX]
  .
  .
  .
```

也可以使任一寄存器与立即数或别的寄存器、或任一内存操作数相比较。

例如：
```
CMP BX,04FEH
CMP DX,DI
CMP CX,COUNT[BP]
  .
  .
  .
```

也可以使内存操作数与立即数，与任一寄存器相比较。

例如：
```
CMP DATA,100
CMP COUNT[SI],AX
CMP POINTER[DI],BX
```

比较指令主要用于比较两个数之间的关系，即两者是否相等，或两个中哪一个大。在比较指令之后，根据 ZF标志即可判断两者是否相等 。若两者相等，相减以后结果为零，ZF 标志为 1，否则为 0。

若两者不相等，则可在比较指令之后利用其它标志位的状态来确定两者的大小。

如果是两个无符号数进行比较，则在比较指令之后，可以根据CF标志的状态判断两数大小。例如，比较指令

```
CMP AX,BX
```

做 AX-BX 操作。若结果没有产生借位 (CF=0)，显然 AX >=BX，若产生了借位，则 AX<BX (注意，无符号数比较大小时，不能据 SF 标志判断两数的大小)。

带符号数的比较就比较麻烦。因为这时无法根据某一标志 (SF标志或 CF 标志) 判断两数的大小。例如：-2<127

比较指令使两数相减如下：

```
   11111110      (-2)
 - 01111111      (127)
 ----------
   01111111
```

结果的符号为0。若用 SF标志判别两数大小，则认为两数相减后结果为正，从而得出 -2>127 的错误结果。

那么，应如何判断两个带符号数的大小呢？现分析如下：

在 CMP AX,BX 中，若 AX 与 BX 中的数同符号，即 AX>0、BX>0 或 AX<0、BX<0，则 AX-BX不会产生溢出，此时可用 S标志判断两数大小，即 SF=0，AX>BX ；SF=1 则AX<BX。

若 AX 与 BX 中的数不同符号，即 AX>0、BX<0 或 AX<0、BX>0，那么 AX-BX 则可能会产生溢出。若 AX-BX 没有产生溢出，则仍然可用 SF 标志判断两数大小，符合上述规律；但是若 AX-BX产生了溢出，则是当 SF=1 时，AX>BX；SF=0 时 AX<BX。

总上所述，可得如下结论：

当没有溢出（OF=0）时，若 SF=0，则 A>B

若 SF=1，则 A<B

当产生溢出（OF=1）时，若 SF=0，则 A<B

若 SF=1，则 A>B

进而：若 OF "异或" SF =0，则 A>B

OF "异或" SF =1，则 A<B

在8086/8088 转移指令中，考虑到上述情况，设置了两组转移指令，分别适用于无符号数比较和有符号数的比较，详见第四章第三节。

例题：求最大值

若自 BLOCK 开始的内存缓冲区中，有100个带符号数，要我们找出其中的最大值，把它存放到 MAX 单元中。

我们先把数据块的第一个数取至 AX 中，然后从第二个存贮单元开始，依次与 AX中的内容相比较，若 AX 中的值大，接着进行下一次比较；若 AX 中的值小，则把内存单元的内容送至 AX 中。这样，经过99次比较，在 AX 中的必然是数据块中的最大值，再把它存至 MAX 单元中。

要进行 99 次比较，当然要编一个循环程序，在每一循环中要用比较指令，然后用转移指令来判断大小，循环开始前要置初值。

能满足上述要求的程序段为：

```
        MOV BX,OFFSET BLOCK
        MOV AX,[BX]
        INC BX
        INC BX
        MOV CX,99
AGAIN:  CMP AX,[BX]
        JG NEXT
        MOV AX,[BX]
```

```
NEXT:   INC BX
        INC BX
        DEC CX
        JNE AGAIN
        MOV MAX,AX
        HLT
```

三.逻辑运算和移位指令

这一组指令包括逻辑运算，移位和循环移位指令三部分。

1.逻辑运算指令

(1) NOT ,OPRD

这条指令对操作数求反，然后送回原处，操作数可以是寄存器或存贮器内容。此指令对标志无影响。例如：NOT AL

(2) AND

这条指令对两个操作数进行按位的逻辑 "与" 运算。即只有相 "与" 的两位全为 1，"与"的结果才为 1；否则"与"的结果为 0。"与"以后的结果送回目的操作数。

8086/8088 的 AND 指令可以进行字节操作，也可以进行字操作。

"与"指令的一般格式为：

```
AND OPRD1,OPRD2
```

其中目的操作数 OPRD1可以是累加器，也可以是任一通用寄存器，也可以是内存操作数(所有寻址方式)。源操作数 OPRD2 可以是立即数，是寄存器，也可以是内存操作数(所有寻址方式)。

例如：
```
AND AL,0FH
AND AX,BX
AND SI,BP
AND AX,DATA_WORD
AND DX,BUFFER[SI+BX]
AND DATA_WORD,00FFH
AND BLOCK[BP+DI],DX
```

某一个操作数，自己和自己相"与"，操作数不变，但可以使进位标志 CF 清 0。

"与"操作指令主要用在使一个操作数中的若干位维持不变，而若干位置为 0的场合。这时，要维持不变的这些位与"1"相"与"；而要置为 0 的这些位与"0"相"与"。

此指令执行以后，标志 CF=0，OF=0。标志 PF、SF、ZF 反映操作的结果，对标志 AF 未定义。

(3) TEST

本指令能完成与 AND指令相同的操作，结果反映在标志位上，但并不送回，即TEST指令不改变操作数的值。

这条指令，通常是在不希望改变原有的操作数的情况下，用来检测某一位或某几位的条件是否满足。编程时可在这条指令后面加上条件转移指令。

TEST 指令的一般格式为：

```
TEST   OPRD，im    ；im 是立即数
```

立即数中哪一位为 1，表示对哪一位进行测试。

例如，若要检测 AL 中的最低位是否为 1，为 1 则转移，可用以下指令：

```
        TEST AL,01H
        JNZ THERE
            .
            .
            .
            .
THERE:      .
            .
```

若要检测 CX 中的内容是否为 0，为 0 则转移，可用以下指令：

```
        TEST CX,0FFFFH
        JZ THERE
            .
            .
            .
            .
THERE:      .
            .
```

(4) OR

对指定的两个操作数进行逻辑 "或" 运算。即进行 "或" 运算的两位中的任一个为 1（或两个都为 1）时，则 "或"的结果为 1；否则为 0。"或"运算的结果送回目的操作数。

8086/8088 允许对字节或字进行"或"运算。

"或" 运算指令使标志位 CF=0，OF=0 ；"或" 操作以后的结果反映在标志位 PF、SF 和 ZF 上；对标志 AF 未定义。

"或" 指令的一般格式为：

```
OR OPRD1,OPRD2
```

其中，目的操作数 OPRD1，可以是累加器，可以是任一通用寄存器，也可以是一个内存操作数（所有寻址方式）。源操作数 OPRD2 ，可以是立即数，也可以是寄存器，也可以是内存操作数（所有寻址方式）。例如

```
OR AL,30H
OR AX,00FFH
OR BX,SI
OR BX,DATA_WORD
```

```
OR BUFFER[BX],SI
OR BUFFER[BX+SI],8000H
        .
        .
        .
```

一个操作数自身相"或"，不改变操作数的值，但可使进位标志 CF 清 0。

"或"运算主要应用于：如果要求使一个操作数中的若干位维持不变，而另外若干位为 1 的场合。这时，要维持不变的这些位与 "0" 相 "或"；而要置为 "1" 的这些位与 "1" 相 "或"。

利用 "或" 运算，可以对两个操作数进行组合，也可以对某些位置位。

(5) XOR

这条指令对两个指定的操作数进行 "异或" 运算，即进行 "异或" 运算的两位不同时（即一个为 1，另一个为 0），"异或" 的结果为 1；否则为 0，"异或"运算的结果送回目的操作数。

XOR 指令的一般形式为：

```
XOR OPRD1,OPRD2
```

其中，目的操作数 OPRD1 可以是累加器，可以是任一个通用寄存器，也可以是一个内存操作数（全部寻址方式）。源操作数可以是立即数，可以是寄存器，也可以是内存操作数（所有寻址方式），例如：

```
XOR AL,0FH
XOR AX,BX
XOR DX,SI
XOR CX,COUNT_WORD
XOR BUFFER[BX],DI
XOR BUFFER[BX+SI],AX
        .
        .
        .
```

当一个操作数自身做 "异或" 运算时，由于每一位都相同，则 "异或" 结果必为0，且使进位标志 CF 也为 0，这是使操作数的初值置为 0 的常用的有效的方法。如：

```
XOR AX,AX
XOR SI,SI
```

可使 AX 和 SI 清 0。

若要求一个操作数中的若干位维持不变，而若干位取反，可用"异或"运算来实现。要维持不变的这些位与 "0" 相 "异或"；而要取反的那些位与 "1" 相 "异或"。

XOR 指令执行后，标志位 CF=0，OF=0 ；标志位 PF、SF、ZF反映 "异或"操作结果；标志 AF 未定义。

2.移位指令

8086/8088 有三条移位指令:

算术左移或逻辑左移指令:

```
SAL/SHL OPRD, m    ; m 是移位次数 , 可以是 1或 寄存器 CL
```

SAL/SHL : 即 Shift Arithmetic Left / SHift logic Left

算术右移指令:

```
SAR OPRD, m        ; Shift Arithmetic Right
```

逻辑右移指令:

```
SHR OPRD, m        ; Shift logic Right
```

这些指令可以对寄存器操作数或内存操作数进行指定的移位，可以进行字节操作，也可进行字操作。

这些指令可以一次只移 1 位 ，也可以移位由寄存器 CL 中的内容规定的次数（即 m 为 1或 CL)。

(1) SAL 和 SHL 这两条指令，在物理上是完全相同的。每移位一次在右面补零，而最高位进入标志位 CF。如图 2-12 所示。

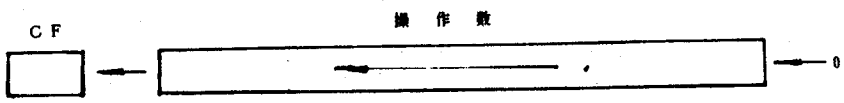


图 2-12 左移指令示意图

在移位次数为 1 的情况下 ，若移位完了以后 ，操作数的最高位与标志位C不相等，则溢出标志 OF=1 ； 否则 OF=0 。这用以表示移位以后的符号位与移位前的是否相同（若相同，则 OF=0)。

标志位 PF、.SF、ZF 表示移位以后的结果。例如:

```
SAL AH,1
SAL DI,CL
```

(2) SAR 每执行一次，使操作数右移一位，但保持符号位不变，最低位移至标志CF 。如图 2-13 所示。

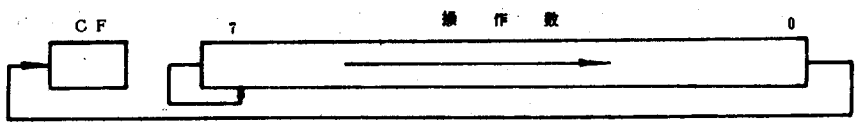


图 2-13 算术右移指令示意图

SAR 可以移位指定的次数。指令影响标志位 CF、OF、PF、SF 和 ZF。

(3) SHR 每执行一次，使操作数右移一位，最低位移至标志 CF ，最左位补 0。如图 2-14 所示。

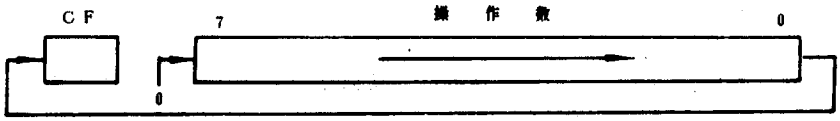


图 2-14 逻辑右移指令示意图

指令可以执行指定的次数。在指定的移位次数为 1 时，若移位以后 ，操作数的最高位与次最高位不同，则标志 OF=1，反之，OF=0 。这用以表示移位以后的符号位是否改变(OF=0，符号位未变)。

3.循环移位指令

8086/8008 有四条循环移位指令

左循环移位 ROL OPRD , m ; Rotate Left

右循环移位 ROR OPRD , m ; Rotate Right

带进位左循环移位 RCL OPRD , m ; Rotate Left through CF

带进位右循环移位 RCR OPRD , m ; Rotate Right through CF

前两条循环指令，未把标志位 CF 包含在循环的环中，后两条把标志位 CF 包含在循环的环中，作为整个循环的一部分。

循环指令可以对字节进行操作，也可以对字进行操作。操作数可以是寄存器操作数，也可以是内存操作数。可以是循环移位一次，也可以循环移位由CL的内容所决定的次数。

ROL 指令，每执行一次。把最高位一方面移入标志位CF，另一方面返回操作数的最低位。如图 2-15(a) 所示。

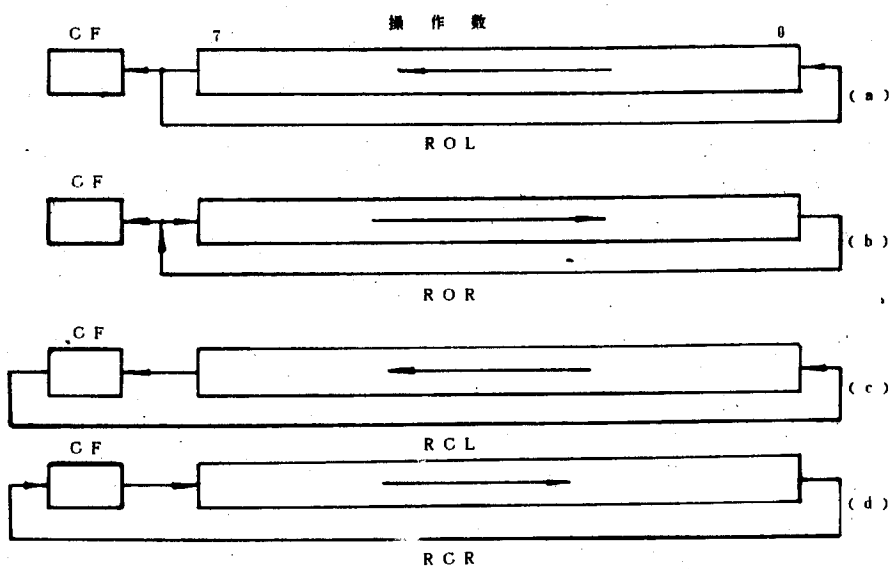


图 2-15 循环移位指令示意图

当规定的循环次数为 1 时，若循环以后的操作数的最高位不等于标志位 CF，则溢出标志 OF=1；否则 OF=0。这可以用来表示移位前后的符号位是否改变（OF=0，则表示符号

位未变)。

ROL 指令只影响标志位 CF 和 OF。

ROR 指令，每执行一次，操作数的最低位一方面传送至标志 CF，另一方面循环回操作数的最高位，如图 2-15(b) 所示。

当规定的循环次数为 1 时，若循环移位后操作数的最高位与它的次高位不相等，则标志位 OF=1；否则 OF=0。这可用以指示移位前后的符号位是否改变（OF=0，表示符号位未变)。

此指令只影响标志 CF 和 OF。

RCL指令是把标志位 CF包含在循环中的左循环移位指令。每执行一次，则操作数的最高位传送至标志 CF，而原有的标志 CF中的内容，传送到操作数的最低位。如图2-15(c) 所示。

只有在规定循环次数为 1 时，若循环以后的操作数的最高位与标志 CF不相等时，则标志 OF=1；否则 OF=0。这可以用来表示循环以后的符号位与原来的是否相同。

这个指令只影响标志位 CF 和 OF。

RCR 指令是把标志位 CF包含在循环中的右循环移位指令。每执行一次，标志位CF中的原内容传送至操作数的最高位，而操作数的最低位送至标志CF。如图 2-15(d) 所示。

只有当规定的循环次数为1 时，在循环以后，若操作数的最高位与次高位不相等，则标志 OF=1；否则 OF=0。这可以用来表示循环前后的符号位是否相同。

本指令只影响标志 CF 和 OF。

左移一位，只要左移以后的数未超出一个字节或一个字的表达范围，则原数的每一位的权增加了一倍，相当于原数乘 2。右移一位相当于除 2。

在数的输入输出过程中乘 10 的操作是经常要进行的。在 8086/8088中有乘法指令，乘 10 可采用乘法指令来做，由指令手册中可知，执行时间最短的乘法指令需要 70 个 T 状态。而 X*10=X*2+X*8，也可以采用移位和相加的办法来实现 *10。请看以下程序段，为保证结果完整，先将 AL 中的字节扩展为字。

```
                                        T 状态数
MOV AH,0      ;                          4
SAL AX,1      ;X*2                       2
MOV BX,AX     ;移至 BX 中暂存             2
SAL AX,1      ;X*4                       2
SAL AX,1      ;X*8                       2
ADD AX,BX     ;X*10                      3
```

这样，从指令的数量来说是多了，但总的执行时间（15 个 T 状态）短得多。

X*2，也可以用 X+X 来实现。所以 X*10 也可以用以下程序段实现：

T 状态数

```
MOV AH,0      ;                    4
ADD BX,AX     ;X*2                 3
MOV BX,AX     ;                    2
ADD AX,AX     ;X*4                 3
ADD AX,AX     ;X*8                 3
ADD AX,BX     ;X*10                3
```

总共的执行时间为 18 个 T 状态，所以，用移位的办法实现 *10 是比较合适的。

在 8086/8088中可以实现 16 位数的移位，但若有一个四字节数，它们或是放在两个通用寄存器中（例如 AX 和 DX 中），或是存放在连续的内存单元中 。若我们要求这个四字节数整个左移一位或整个右移一位又如何实现呢?

拿左移来说，可以先使低 16 位左移一位，再把高 16 位左移一位，其中的困难在于如何把低 16 位中的最高位 D15 位移至高 16 位中的最低位即 D16 位。任何一条左移指令都可以把 D15 位移至进位标志 CF中，而要把标志 C中的内容移至最低位可采用大循环（包括 CF 在内的）循环指令。即可用以下指令:

```
SAL AX,1
RCL DX,1
```

或:

```
SAL FIRST_WORD,1
RCL SECOND_WORD,1
```

例题: BCD 码转换为 ASCII 码

若在内存某一缓冲区中存放着若干个单元的用 BCD 码表示的十进制数 。每一个单元中放两位 BCD 码，要求把它们分别转换为 ASCII 码 。高位的 BCD 码转换完后放在地址较高的单元。

能满足上述要求的一个汇编语言的程序实例如下:

```
                NAME BCD TO ASCII
DATA    SEGMENT
BCDBUF  DB 34H,56H,23H,70H,96H,45H,32H,14H,81H,99H      ;10 个十进制数
COUNT   EQU $-BCDBUF
ASCBUF  DB 20 DUP(?)                                    ;需20个单元存放ASCII码
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
STAPN   DB 100 DUP(?)
STACK   ENDS
COSEG   SEGMENT
        ASSUME CS:COSEG,DS:DATA,ES:DATA,SS:STACK
```

```
START:  PUSH DS
        MOV AX,0
        PUSH AX
        MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
        MOV SI,OFFSET BCDBUF
        MOV DI,OFFSET ASCBUF
        MOV CX,COUNT
        CLD                              ;增量方向
TRANT:  LODSB
        MOV BL,AL
        AND AL,0FH
        OR AL,30H                        ;低位十进制数→ASCII码
        STOSB
        MOV AL,BL
        PUSH CX
        MOV CL,4
        SHR AL,CL                        ;右移四位
        POP CX
        OR AL,30H                        ;高位十进制数→ASCII码
        STOSB
        LOOP TRANT
        RET
COSEG   ENDS
        END START
```

这是一段实用程序，从这里可以看到 8086/8088 的一般程序结构。本段程序中引用的数据定义、段的定义等伪指令，将在下一章讲述；程序中用到的尚未学过的指令，可以查阅书后的附录 A。

小结：

以上，我们已经介绍了8086/8088 常用的部分指令，其它指令在以后的有关章节中将加以介绍。学习指令系统，应注意以下三方面的问题：

1.到何处查找某条指令。

在编程序、读程序的过程中，经常会迂到这种疑问：有没有这样一条指令？这条指令

有什么功能？为此就需要查阅指令表。为了能够尽快查到有关指令，应记住指令分类以及各类指令的特点，第一步先找到指令所在类别，再在该类指令内查找。

2.查找指令的要点。

一条指令包括的内容很多，找到一条指令后，应注重什么方面，这将因查找的目的不同而异。通常需要注意的是：

(1) 一条或一类指令的功能。

(2) 使用一条或一类指令的初始条件。例如：SHL DI,CL 要求事先将移位次数送入CL。

(3) 指令执行的结果所在。

例如：比较两个操作数的指令 CMP AX,BX

指令实现 AX 减 BX，但结果并不送回（即任一操作数都未改变。那么，若再与其它数比较时，不必重取这个数）。但结果的特性反映在标志寄存器中。当然，根据不同的查找目的，可查看指令对标志位的影响情况，指令所用的字节数，指令执行时间等内容。

3.如何使用指令。

关于如何使用一条指令，以及什么情况下使用什么指令等这些问题，将会在以后各章中陆续给与说明。

习题二

1. 分别指出下列指令中源操作数和目的操作数的寻址方式。

(1) MOV SI,100　　(2) MOV CX,DATP[SI]　　(3) MOV [SI],AX

(4) ADD AX,[BX][DI]　　(5) AND AX, BX　　(6) PUSHF

2. 试述指令 MOV AX,2000H 和 MOV AX,DS:[2000H] 的区别。

3. 标志寄存器 Flag 中包括几个标志位，各位的状态含义及用途如何？

4. 寄存器 BX 作地址指针，自 BX 所指的内存单元开始连续存放着三个无符号数（字），编程序求它们的和，并将结果存放在这三个数之后。

5. 读下述程序段，请问：什么情况下，本段程序的执行结果是 AH=0？

```
BEGIN:    IN      AL, 5FH
          TEST    AL, 80H
          JZ      BRCH1
          MOV     AH, 0
          JMP     STOP
BRCH1:    MOV     AH, 0FFH
STOP:     HLT
```

6. 读如下程序：

```
START:    IN      AL,20H
```

```
        MOV     BL, AL
        IN      AL, 30H
        MOV     CL, AL
        MOV     AX, 0
ADLOP:  ADD     AL, BL
        ADC     AH, 0
        DEC     CL
        JNZ     ADLOP
        HLT
```

请问：(1) 本程序实现什么功能 ？　　(2) 结果在哪里 ？

7. 自 BX 寄存器所指的内存单元开始，连续存放着两个四字节的有符号数（低位字节在前），编程序求它们的差（第一个数减第二个数），并将结果存放在此两数之后，若有溢出，将 BX 寄存器清零。

8. 编程序使：

(1) AX 寄存器的低四位清零。

(2) BX 寄存器的低四位置 1。

(3) CX 寄存器的低四位变反。

(4) 用 TEST 指令测试 DL 寄存器的位 3 位 6 是否同时为 0，若是，将 0 送 DL 寄存器；否则，将 1 送 DH。要求用两种方法。

9. 编程序使 AL 寄存器中的无符号数乘 20。

10. 数据定义语句如下所示：

```
FIRST   DB      90H,5FH,6EH,69H
SECOND  DB      5 DUP ( ? )
THIRD   DB      5 DUP ( ? )
FORTH   DB      5 DUP ( ? )
```

自 FIRST 单元开始存放的是一个四字节的十六进制数（低位字节在前），要求：

(1) 编一段程序将这个数左移两位后存放到自 SECOND 开始的单元。（注意保留移出部分，下题 (2) 也如此）。

(2) 编一段程序将这个数右移两位后存放到自 THIRD 开始的单元。

(3) 编一段程序将这个数求补以后存放到自 FORTH 开始的单元。

第三章　汇编语言与汇编程序

用指令的助记符、符号地址、标号等符号书写程序的语言，称为汇编语言。用汇编语言编写的程序称为汇编语言源程序（简称源程序）。把源程序翻释成机器语言程序（目标程序）的过程叫做汇编。完成汇编任务的程序叫做汇编程序。

汇编程序是最早也是最成熟的一种系统软件，它最主要的功能是将汇编语言源程序翻译成机器语言程序，还具有一些其它功能，如：据用户指定自动分配存贮区域（包括程序区、数据区、暂存区等）；自动地把各种进位制数转换成二进制数，把字符转换成 ASCII 码，计算表达式的值等；自动对源程序进行检查，给出错误信息（如非法格式，未定义的助记符、标号，漏掉操作数等）等。

汇编程序可以用汇编语言编写，也可以用各种高级语言写成，汇编程序的类型很多，但主要功能都一致，附加功能各有不同。

汇编程序的主要类型有：

交叉汇编程序：运行这种汇编程序的计算机与该汇编程序为之汇编成目标程序的机器是不同的。例如汇编程序在大机器 HP3000 上运行，而汇编成的目标程序是在 Z80系列机上执行的。

驻留汇编程序：运行这种汇编程序的计算机就是执行目标程序的计算机。

宏汇编程序：它允许把一指令序列定义为一条宏指令，有宏汇编功能。

为了完成汇编工作，汇编程序一般采取两遍扫描方式。第一遍扫描源程序产生符号表、处理伪指令等，第二遍扫描产生机器指令代码、确定数据等。

关于汇编与宏汇编程序的使用方法详见第八章。

第一节　汇编语言语句

一、语句的种类和格式：

1．三种基本语句：指令语句、伪指令语句和宏指令语句。

每一个指令语句在汇编时产生一个目标代码，对应着机器的一种操作。

如：　MOV　　BX，0　　与　　ADD　SI，AX　　等。

伪指令语句没有目标代码与之对应，主要是为汇编程序服务的。

如：　　PLACE　　DB　　?

将告诉汇编程序PLACE 定义为一个字节，所以汇编程序要为它分配一个存贮器地址。以后，当汇编程序迂到指令语句 INC PLACE 时，它将产生一个使 PLACE 单元内容增量的

目标码指令。

宏指令语句将在本章第四节讲述。

2．语句格式：

前两种语句的格式是类似的。指令语句的格式为：

标号：　　助记符　　参数，参数　　；注释

伪指令语句的格式为：

名字　　定义符　　参数，……，参数　　；注释

两种语句都由四部分组成：

第一部分是标号、名字：

在指令语句中的标号后面跟有冒号(：)，而在伪指令语句中的名字后面没有冒号，这就是两种语句在格式上的主要不同。

一个标号与一条指令的地址的符号名相联系，标号可以作为 JMP 指令和 CALL 指令的一个操作数。伪指令语句中的名字一般不作为 JMP、CALL 指令的操作数，但在间接寻址时可以。

在指令语句中的标号，可以是任选的或省略；伪指令语句中的名字可以是变量名、段名、过程名、符号名等等，可以是强制的、任选的或省略，取决于实际的定义符。

第二部分是助记符、定义符：

在指令语句中的助记符，规定这个指令语句的操作类型，如 ADD (加)、SUB(减)等。

在伪指令语句中的定义符规定这个语句的伪操作功能 。指令语句中的助记符相应于 8086/8088 指令系统中大约 100 个可能的操作码，而伪指令语句中的定义符相应于 IBM 宏汇编中提供的大约数10种伪操作功能。

第三部分是参数

助记符和定义符可以要求一系列参数，使之更能实现它们的目的。参数可分为三类：常数、操作数和表达式。

(1)常数：

IBM 宏汇编中，允许以下几种常数：

①二进制常数：以字母 B 结尾的由若干个“0”和“1”组成的序列，例如：

00101100B

②十进制常数：

由若干个 0 到 9 的数字组成的序列，可以以字母 D 结尾，或省略字母 D 。例如：1234D或1234。

③十六进制常数：

以字母 H结尾，由若干0～9的数字或字母 A～F 所组成的序列。为了避免与标识符相混淆，十六进制数必须以数字打头。所以凡以字母 A～F 开始的十六进制数，必须在前面加上 0 。例如： 56H，0BA3FH 等。

④八进制常数：

以字母Q结尾的，由若干个 0～7的数字组成的序列。例如 255Q、377Q等。

⑤十进制科学表示法：

即十进制浮点表示法。例如 2.725E-2。

⑥十六进制实数

由若干位十六进制数码（0～9）和（A～F）组成，以字母 R 结尾 ，第一位必须是 0～9 之间的数码，一个十六进制实数的总的数字位数必须是 8，16 或 20，而由字母打头的数前面必须加 0，这时数字位数多一位。

⑦串常数：

一个串常数是用引号括起来的一个或多个字符。串常数的值是包括在引号中的字符的 ASCII代码值。例如 'A' 的值是 41H ，而 'AB' 的值为 4142H。因此串常数与整数常数可以交替使用。

(2)操作数：

操作数可以是常数操作数、寄存器名或存贮器操作数。

①常数操作数：可以是具有数字值的常数或表示常数的标号和名字。 如 100 和前面提到过的名字 PLACE 等。也可以是寄存器名和输入输出端口地址。如 AX，SI和 5FH等。

②存贮器操作数：

可以分成标号和变量两种。 标号是可执行的指令语句的地址符号。这是转移指令JMP和调用指令CALL可以转向的目标操作数。变量通常是指存放在某些存贮单元中的值，这些值显然是可变的。可以用直接寻址，基址寻址，变址寻址、基址变址寻址方式对其存取。

作为存贮器操作数的标号或变量都有三种特性：

·段值(SEGMENT)：即标号或变量所在段的起始地址的前16位，而它的低四位总是0。

·段内的地址偏移量（OFFSET)。即标号或变量所指的地址与所在段的起始地址（ 段地址 ）之间的地址的偏移量。

·类型（TYPE)，对于变量来说，类型主要指是字节（BYTE）还是字（WORD)或者是双字（DWORD）等。对于标号来说，类型主要指是 NEAR 还是 FAR ，即当标号作为转移指令JMP及调用指令 CALL 的目的操作数时，是段内转移和调用（NEAR)，还是段间的转移和调用（FAR)。

(3)表达式

由某些常数、操作数、操作符和运算符组合而成表达式。

IBM 宏汇编中有三种运算符：算术运算符、逻辑运算符和关系运算符；两种操作符：分析操作符和合成操作符。

①算术运算符：

这是一些熟悉的运算符：加法(+)、减法(-)、乘法(*)、除法（/)，还有 MOD，MOD产生除法以后的余数。如：19MOD7是5，即19/7得余数为5。算术运算符总可以应用于数字操

作数，结果也是数字的。而应用于存贮器地址操作数时，有意义的运算符是加、减。

②逻辑运算符：

逻辑运算符是按位操作的与（AND）、或（OR）、异或（XOR）和非（NOT）。逻辑运算符的操作数只能是数字的，且结果也是数字的。存贮器地址操作数不能进行逻辑运算。

注意：AND、OR、XOR 和 NOT 也是指令助记符。作为 IBM 宏汇编的运算符时，它们是在程序汇编时计算的。而作为指令助记符时，则是在程序执行时计算的。例如：

```
AND     DX,     PORT AND 0FEH
```

其中第二个 AND 是逻辑运算符．在汇编时，计算 PORT AND 0FEH 后，产生一个立即数作为指令的操作数。而第一个 AND 是指令助记符，在汇编以后，执行 AND 指令时，DX 的内容与上述立即数相“与”，结果放在 DX 中。

③关系运算符：

关系运算符有相等（EQ）、不等（NE）、小于（LT）、大于（GT）、小于或等于(LE)、大于或等于（GE），关系运算符连接两个操作数，必须都是数字的或是在同一段内的存贮器地址。运算结果始终是一个数字值。若关系是假（关系不成立），则结果为 0 ，若关系为真，则结果为 0FFFFH。例如：

```
MOV     BX，PORT LT 5
```

若 PORT 的值小于 5，则汇编程序将把这条指令汇编为：

```
MOV     BX，0FFFFH
```

否则，若 PROT 的值不小于 5，则汇编为

```
MOV     BX，0
```

一般不单独使用关系运算符。因为运算的结果不是 0 就是 0FFFFH，没有别的选择，所以，常与其它运算符组合起来使用。例如：

```
MOV     BX，((PORT LT 5) AND 20) OR ((PORT GE 5) AND 30)
```

当 PORT 的值小于 5 时，上述指令将汇编为

```
MOV     BX，  20
```

否则为 `MOV     BX，30`

④分析操作符和合成操作符

分析操作符可以把存贮器操作数分解为它的组成部分；而合成操作符是把组成部分综合为存贮器操作数。（详见下节）。

第四部分是注释

注释由分号(；)开始，用来对语句的功能加以说明，使程序更容易阅读。注释要简明扼要。

二．指令语句：

IBM 宏汇编中的指令语句，必须包括一个指令助记符，以及充分的寻址信息以允许汇编程序产生一条指令。 8086/8088 指令系统的每一条指令都属于指令语句，每一条指令

语句（除个别外）都对应着 8086/8088 CPU 的一种操作，对应着相应的指令代码。

有两条特殊指令，他们是 NOP和 NIL。NOP 指令是一字节空操作指令，它不做任何存贮器询问，但是它要占用存贮单元。这有什么用途呢？ NOP 指令可以保留一些单元为以后填入指令用。另外当需要精确的时间关系时，可利用 NOP 实现延时作用。

NIL 是不使汇编程序产生任何指令代码的唯一指令。与 NOP 指令相比， NOP 使汇编程序产生一条不做任何操作的指令，而 NIL 甚至连指令都不产生。

NIL 在汇编语言程序中是为标号保留空格的。如：

```
CYCLE:    NIL
          INC    AX
```

虽然它与以下语句等效

```
CYCLE:    INC    AX
```

但有了 NIL ，若以后需要的话，便于在 INC 指令前插入指令。

三．伪指令语句：

IBM 宏汇编中包括如下伪指令语句：

- 符号定义语句 (SymboL definition)
- 数据定义语句 (Data definition)
- 段定义语句 (Segmentation definition)
- 过程定义语句 (Procedure definition)
- 结束语句 (Termination)

后面介绍的条件汇编语句，以及结构、记录也属于伪指令语句：

下面分别加以介绍。

1.符号定义语句

(1)等值语句 EQU

EQU 语句给符号名定义一个值，或定义为别的符号名、甚至是一条可执行的命令、表达式的值等。

EQU 指令的格式为：

符号名 EQU 表达式

例如：

```
PORT1        EQU    212
BUF_SIZE     EQU     32
PORT2        EQU    PORT1 + 1
COUNT        EQU     CX
CBD          EQU    DAA
```

第四个语句与前三个不同， COUNT 不是代表一值，而是寄存器 CX 的同义语， 最后一个语名把符号 CBD 定义为一条指令的助记符 DAA。

EQU 语句不能重新定义，即在同一个源程序中，用 EQU 语句定义的符号，不能再赋于不同的值。

(2) 等号语句 (=)

此语句的功能与 EQU 语句类似，最大的特点是能对符号进行再定义。例如：

```
EMP    =    6
EMP    =    9
EMP    =    EMP + 1
```

2. 数据定义语句

(1) 数据定义语句，为一个数据项分配存贮单元，用一个符号名与这个或这些存贮单元相联系，并为这个数据项提供一个任选的初始值。

例如：

```
THING            DB          ?          ; 定义一个字节
BIG_THING        DW          ?          ; 定义一个字(两个字节)
BIGGEST_THING    DD          ?          ; 定义一个双字(四个字节)
```

THING 是一个变量名，它与一个存贮器中的字节相联系，BIG_THING 与存贮器中两个连续的字节相联系，而 BIGGEST_THING 与存贮器中四个连续的字节相联系。

在讨论上述语句中的符号“?”的意义以前，要介绍一下数据项的初值的概念。

由汇编程序产生的目标码，包括指令代码和存放这些指令的地址，在目标码已经产生以后，指令已经放在指定地址的存贮器中，然后可以执行。在指令送存贮器的时候，数据项的初值也可以送至存贮器中、这意味着目标码除了包含指令代码和它们的地址以外，也可以包括数据项的起始值和它们的地址。这些初始值是由数据定义语句所规定的。例如：

```
THING      DB      25
```

在汇编时，会把 25 放入与 THING 相联系的存贮单元中。

当汇编程序迁到“？”号时，它仍然为数据项分配存贮器，但并不产生一个目的码去初始化这个存贮单元。

通常，初始值能用一个表达式来规定，因为表达式是在汇编时计算的，所以能写如下语句：

```
IN_PORT          DB          PORT_VAL
OUT_PORT         DB          PORT_VAL + 1
```

其中，PORT _VAL 已由 EQU 语句赋了值。

(2) 可以用数值去初始化一个字节、字或双字。同样在存贮单元中可以存放存贮器地址值，但它不能是一个字节，可以是一个字（如地址偏移部分），也可以是一个双字（地址的两部分），例如：

```
LITTLE_CYCLE    DW    OFFSET CYCLE    ;  CYCLE 的地址偏移量
BIG_CYCLE       DD    CYCLE           ;  CYCLE 的段地址和地址偏移量。
```

```
CYCLE:      MOV     BX，AX
```

(3) 可以用数据定义语句定义一个表或一个字符串。到目前为止，我们已经用数据定义语句一次定义一个字节、字或双字，即定义简单变量。有时我们会用到由字节、字或双字构成的表，例如： 8086/8088 的 XLAT 指令用一个由字节组成的表，转换一个编码值为另一种编码；8086/8088 的串操作指令对包含串元素的由字节或字组成的表进行操作；8086/8088 的中断机构用一个由双字组成的表去指向中断服务程序的起始地址等。

一个表可以由一个数据定义语句的若干个初值定义，例如下列语句定义了一个包含 1～5 的平方值的表：

```
POWERS_2    DB      1，4，9，16，25
```

在目标码输入存贮器时，这个语句就把 POWERS_2 这个存贮地址的字节初始化为 1，下面四个字节分别初始化为 4，9，16，25。可以用下列语句把一个表的所有字节全部初始化为 0。

```
ALL_ZERO    DB      0，0，0，0，0
```

或缩记为：

```
ALL_ZERO    DB      5 DUP (0)
```

一个未初始化的数组，可由下列等效语句之一定义：

```
DONT_CARE   DB      ?，?，?，?，?，?
```

或

```
DONT_CARE   DB      6 DUP(?)
```

DB 伪指令也可以定义用引号括起来的字符串，它把字符串中的各个字符的 ASCII 码值相继地放在相应的存贮单元中，例如：

```
MUCT_CHAR   DB      "How    Are    you ? "
```

即在 MULT_CHAR 所指的单元中存放字符 H 的 ASCII 码值(48H)，在 MULT_CHAR + 1单元中放字符 O 的 ASCII 码值（4FH），依次类推。

(4) IBM 宏汇编对在程序中涉及到的每个存贮单元与一种类型联系起来，这样能使对访问存贮器的指令产生正确的目标码。例如：

```
XUM         DB      ?
```

告诉汇编程序， SUM 是字节类型的。当汇编程序遇到这样一个指令语句 INC SUM 时，汇编程序就知道产生一个字节增量指令，而不是字增量指令。

一个存贮单元，可以是下列类型中的一个

① 字节数据

```
如：XUM         DB      ?       ；定义一个字节
```

② 字数据 （两个连续的字节，低位字节在前 ）

```
如：BIG         DW      ?       ；定义一个字
```

③ 双字数据(四个连续的字节)

如：BIGGEST　　DD　　?　　；定义一个双字

④ NEAR 指令单元

如在一段程序内的指令：

```
CYCLE:　　CMP　　SUM，100
```

⑤ FAR　指令单元

一个指令单元能出现在一个 JMP或CALL 指令语句中。若这个单元的类型是 NEAR，汇编程序将产生一个段内 JMP或 CALL指令；若单元的类型是 FAR，则产生一个段间 JMP 或 CALL 指令。例如：

```
CYCLE:　　CMP　　SUM，100
```

告诉汇编程序，存贮单元 CYCLE 的类型是 NEAR。以后，当汇编程序迂到指令

```
JMP　　CYCLE
```

时，就产生一个段内 JMP 指令。

一个 NEAR 指令单元规定了一个长度为两个字节的指针，即此指令单元在段内的地址偏移量。获取了此地址偏移量，就可以实现段内的转移和调用。

一个 FAR 指令单元规定了一个长度为四个字节的指针，即此指令单元所在段的段地址和段内的地址偏移量。只有获取了这四个字节，才能得到一个 FAR指令单元的全地址，才能实现段间的调用和转移。

从一个存贮单元加或减一个数字值形成的新的存贮器地址与原存贮单元有相同的类型。如 SUM + 2 是字节型，BIG - 3 是字型，CYCLE + 1 是一个 NEAR 型指令单元（参见前述定义)。

(5) 分析和合成操作符。

分析操作符把存贮器地址操作数分解成它们的组成部分。这些操作符是 SEG、OFFSET、TYPE、SIZE、LENGTH。

SEG 操作符返回存贮器地址操作数所在段的段地址部分。OFFSET操作符返回地址的段内偏移量部分。

TYPE 操作符返回一个数字值，它表示存贮器操作数的类型部分。各种存贮器地址操作数类型部分的值如下：

存贮器操作数	类型部分
字节数据	1
字数据	2
双字数据	4
NEAR 指令单元	-1
FAR 指令单元	-2

注意：字节、字和双字的类型部分，分别是它们所占有的字节数，而指令单元的类型部分的值没有实际的物理意义。若 TYPE 放在一个结构名前，则返回此结构所占用的字节数。

LENGTH 和 SIZE 操作符只应用于数据存贮器地址操作数。LENGTH 返回一个与存贮器地址操作数相联系的单元数，SIZE 操作符返回一个为存贮器地址操作数分配的字节数。例如：若 MULT_WORDS 定义为

```
MULT_WORDS      DW      50 DUP(0)
```

于是 LENGTH MULT_WORDS 是 50，而 SIZE MULT_WORDS 是 100。注意：SIZE X 等于 (LENGTH X) * (TYPE X)。

合成操作符由地址部分建立存贮器地址操作数，这些操作符是 PTR 和 THIS。

PTR 的格式：

类型 PTR 表达式

类型可以是 BYTE、WORD、DWORD、NEAR、FAR。PTR 的意思是，仍按PTR后面的表达式去寻址，不管它原来有无类型、或是什么类型，均以 PTR 前的类型为准。

PTR 操作符建立一个存贮器地址操作数，它与其后的存贮器地址操作数有相同的段地址偏移量，但有不同的类型。不象一个数据定义语句，PTR 操作符并不分配存贮器，它可以给已分配的存贮器一个另外的意义。

例 1.若 TWO_BYTE 定义为：

```
TWO_BYTE        DW      ?
```

于是我们能给 TWO_BYTE 的第一个字节定义为：

```
ONE_BYTE    EQU    BYTE    PTR    TWO_BYTE
```

在这个例子中，PTR 操作符建立了一个与 TWO_BYTE 有相同的段和偏移量的新的存贮器地址操作数，但它的类型部分是字节。可以给第二个字节定名为：

```
OTHER_BYTE    EQU    BYTE    PTR  TWO_BYTE+1
```

或简单地定义为：

```
OTHER_BYTE    EQU    ONE_BYTE+1
```

例 2. PTR 操作符可以建立字和双字，解释如下：

```
MANY_BYTES      DB      100   DUP(?)    ; 一个 100个字节的矩阵
FIRST_WORD      EQU     WORD    PTR   MANY_BYTES    ; 50个字
SECOND_DOUBLE   EQU     DWORD   PTR   MANY_BYTES    ; 25个双字
```

例 3. PTR 运算符也可以建立指令单元：

```
INCHES:     CMP     SUM,    100         ; INCHES 的类型是 NEAR
            JMP     INCHES              ; 段内转移
;
MILES       EQU     FAR     PTR  INCHES   ; MILES 类型是 FAR
```

```
        JMP     MILES                   ; 段间转移
```

合成操作符 THIS，象 PTR 一样可用来建立一个特殊类型的存贮器地址操作数，而没有为它分配存贮器。新的存贮器地址操作数的段和偏移量部分，就是下一个能分配的存贮单元的段和偏移量。例如：

```
MY_BYTE     EQU     THIS     BYTE
MY_WORD     DW      ?
```

将建立 MY_BYTE 具有字节类型，且与 MY_WORD 具有相同的段和偏移量部分。在这个例子中，MY_BYTE 也能用 PTR 操作符建立：

```
MY_BYTE     EQU     BYTE    PTR     MY_WORD
```

THIS操作符对于建立FAR 指令是方便的：

```
MILES       EQU         THIS    FAR
            CMP         SUM , 100
             .
             .
             .
            JMP         MILES
```

注意：在上例中 THIS 操作符的使用，并不需要有一个与 MILES 具有相同的段和偏移量的 NEAR 指令单元。但若用 PTR 操作符代替 THIS，这样的NEAR 指令是需要的。

在 IBM 宏汇编中，除了可以用 DB 定义字节，用 DW 定义字，用 DD定义双字以外，还有 DQ 和 DT 伪指令。

DQ 用以定义四个字（8个字节），DT 用于定义 10 个字节。其它功能与上述类似。

8086/8088 宏汇编语言中，操作符及运算符的优先权等级按如下顺序，从高到低排列为：

① 圆括号，尖括号（记录中使用），方括号，圆点符（结构中使用），LENGTH，SIZE，WIDTH，MASK。

② PTR，OFFSET，SEG，TYPE，THIS，段寄存器名：（加段前缀）。

③ *，/，MOD(求模)，SHL，SHR。

④ HIGH，LOW(操作数高、低字节)。

⑤ +，-。

⑥ EQ，NE，LT，LE，GT，GE。

⑦ NOT。

⑧ AND。

⑨ OR，XOR。

⑩ SHORT 。

3. 段定义语句

段定义语句使我们按段来组织程序和利用存贮器。这些命令是：

SEGMENT、ENDS、ASSUME、ORG、PAGE、PUBLIC、TITLE、SUBTIL等。

(1)SEGMENT 和 ENDS语句把汇编语言源程序分成段，这些段就相应于存贮器段。在这些存贮器段中，存放相应段的目的码。汇编程序为什么要关心存贮器段呢？ 这是由于1MB的存储空间是按段使用的。若有一个段内的转移和调用指令，在指令中包含新的单元的16位偏移量；而一个段间的转移和调用指令，必须包括段地址和偏移量。再者，使用当前（即现行）数据段和当前堆栈段的数据访问指令，对于 8086/8088的结构来说是最优的，因为它只包含数据单元的 16 位偏移量。访问当前可寻址段之外的数据单元的指令必须加一个段超越前缀或修改段寄存器的内容。

段定义格式：

```
段名    SEGMENT [定位类型] [组合类型] [' 类别 ']
          .
          .
          .
段名    ENDS
```

段名是自己指定的，定位类型、组合类型和类别是赋给段名的属性。用方括号括起来的项表示此项可以省略（这一约定适合于其它格式定义），若不省略，各项顺序不能错，且用空格分隔(有的汇编程序不支持可选项的省略，这时就要把段定义写完整)。

定位类型表示此段的起始边界要求，可以是PAGE、PARA、WORD或BYTE，它们分别表示如下的边界地址要求：

PAGE=XXXX XXXX XXXX 0000 0000 B

PARA=XXXX XXXX XXXX XXXX 0000 B(隐含值)

WORD=XXXX XXXX XXXX XXXX XXX0 B

BYTE=XXXX XXXX XXXX XXXX XXXX B

分别称它们为以页、节、字、字节为边界。若省略此项，即隐含值为 PARA。

组合类型用来告诉连接程序本段与其它段的关系，分别为 NONE、PUBLIC、COMMOM、AT 表达式、STACK 和 MEMORY。

NONE：表示本段与其它段逻辑上不发生关系，每段都有自己的基地址。这是隐含的组合类型。

PUBLIC：连接程序首先把本段与同名同类别的其它段相邻地连接在一起，然后为所有这些 PUBLIC 段指定一个共同的段基地址，也就是连接成一个物理段。

STACK：与 PUBLIC 同样处理，但此段作为堆栈段，被连接程序中必须至少有一个 STACK 段，如果有多个，则初始化时，SS 指向第一个所遇到的 STACK 段。

CONMON：连接程序为本段和同名同类别的其它段指定相同的基地址，因而本段将与同名同类别的其它段相覆盖。段的长度取决于最长的 COMMON 段的长度。

AT表达式：连接程序把本段装在表达式的值所指定的段地址上。这个类型使得可以在某一固定的存贮区内的某一固定偏移地址处定义标号或变量，以便程序以标号或变量形式

存取这些存贮单元。

MEMORY：连接程序将把本段定位在被连接在一起的其它所有段之上。若有多个MEMORY字段，汇编程序认为所遇到的第一个为 MEMORY，其余为 COMMON。

类别：可以是任何合法的名称，必须用单引号括起来，连接程序只使同类别段发生关连。典型类别如：'STACK'、'CODE'。

(2) 为了汇编正确的段码，汇编程序必须知道程序的段结构和在各种指令执行时访问哪一段，这个信息是由 ASSUME 命令提供的。

下列程序表示如何使用 SEGMENT、ENDS和ASSUME命令定义代码段、数据段、堆栈段和附加段。

```
DATA    SEGMENT                    ;  数据段
   X    DB          ?
   Y    DW          ?
   Z    DD          ?
DATA    ENDS
ESTRA   SEGMENT                    ; 附加段
ALPHA   DB          ?
BETA    DW          ?
GAMA    DD          ?
ESTRA   ENDS
STACK   SEGMENT    PARA    STACK  'STACK'   ; 堆栈段
STAPN   DB         100 DUP(?)                ;  100个字节栈空间
TOP     EQU       LENGTH STAPN
STACK   ENDS
CODE    SEGMENT                              ;  代码段
        ASSUME     CS: CODE , ES: ESTRA , DS: DATA , SS: STACK
MAIN    PROC          FAR
        PUSH      DS                  }
        SUB    AX , AX                }  **
        PUSH    AX                    }
        MOV      AX , DATA
        MOV     DS ,  AX
        MOV     AX , EXTRA
        MOV     ES , AX
        MOV     AX , STACK
        MOV     SS , AX
```

```
        MOV     AX，TOP
        MOV     SP，AX
        ORG     0009H
          ·
          ·
          ·
        RET
MAIN    ENDP，
CODE    ENDS
        END     MAIN
```

上述举例中，在数据段和附加段中定义了一些数据，在堆栈段定义了100 个字节的堆栈空间。ASSUME语句告诉汇编程序可以假定一个特定的段寄存器指向一个特定的段，不如此汇编程序无法生成目的码程序、上例中由 ASSUME 指明 CS 寄存器指向命名为CODE的段，ES指向 EXTRA 段，DS 指向 DATA 段，堆栈段可不要求 ASSUME 语句，此时利用系统设置的堆栈。ASSUME语句可进一步解释如下：

若在上述程序中要求把字节单元 X 的内容传送至字节单元 ALPHA，这需要在码段中增加一些指令。先把 X 的内容传送给一个寄存器（例如 BX ），然后再从这个寄存器传送给 ALPHA，所以需要如下命令：

```
MOV     BX ，X
MOV     ALPHA，BX
```

在指令执行时，8086/8088 CPU 将用寄存器 DS 去寻找所指定的项（X 或 ALPHA)所在段的起始地址，当执行第一条指令时工作正常，因为 X 确实是在由 DS 的内容DATA作为起始地址的段内。但在执行第二条指令时，如何能正常工作呢？因为包含有ALPHA的这一段的起始地址 EXTRA 并不在 DS 中，对这样的指令，在汇编时，由 ASSUME语句告诉汇编程序，EXTRA不在DS 中，而在另一个段寄存器 ES 中。所以要正确执行第二条指令，必须要有一个段超越前缀，虽然指令中没有，汇编程序会产生这样的前缀，加到第二条指令中去，以便将来能正确执行。

有些情况下，为了更明确地指明段寄存器，或代替 ASSUME 语句的作用，我们可以在有关的指令中增加段超越前缀，用以告诉汇编程序，在这一条指令执行时，应使用哪一个段寄存器，例如上面提到的把X的内容移至ALPHA中可以写成：

```
MOV             BX ，       DS：X
MOV             ES：ACPHA，BX
```

这表示当访问X时，应用DS，而访问ACPHA时，应该用ES。

现在的问题是为什么要有标有 ** 的一段程序。因为当 DOS 把控制权转给我们的程序时，在程序段前缀的开始（偏移地址 00H)处安排了一条中断返回指令 INT 20H 。为了在完成我们的程序后正常返回，必须先将 DS 进栈，同时将段内偏移量（ AX 中的零)进

栈 。(关于程序段前缀问题，详见第八章第七节)。然后再把数据段的地址传送给 DS，由 ASSUME 语句告诉汇编程序，DS 寄存器指向数据段。

(3) ORG 伪指令的一般格式是：

ORG　　<表达式>

此语句指定了在它以后的程序段或数据块存放的起始地址的偏移量。也即以语句中表达式的值作为起始地址，连续存放程序或数据，除非迁到一个新的 ORG 语句。 省略ORG，则从本段的起始连续存放。

(4)PUBLIC 和 EXTRN

格式：PUBLIC　名字[，...]

EXTRN　名字：类型[，...]

其中，名字可以是数、变量或标号 (包括过程名)。 在一个模块内或者一段内由 PUBLIC 定义过的名字可在别的段直接引用。EXTRN 说明本段中或本模块中使用的名字已在别的段定义过。类型可以是 BYTE、WORD、DWORD、NEAR、FAR、ABS或 EQU定义的符号。

(5)PAGE、TITLE和SUBTTL

PAGE　参数1，参数2

PAGE一般为程序的第一语句，它指定汇编程序所产生的列表文件应该每页多少行（参数 1) 和每行多少个字符 (参数2)。例如：

PAGE 60，132

将设置每页 60 行，每行 132 个字符。每页行数可为 10～255（隐含 66)，每行字符数可以是 60～132（隐含 80)。

TITLE 正文

TITLE 伪指令用来为程序指定一个标题（不超过 60 个字符)，以后的列表文件会在每页的第一行打印这个标题。

SUBTTL　正文

SUBTTL 为程序指定一个小标题，打印在每一页的标题之后。

4．过程定义语句

过程是程序的一部分，它们可被程序调用。每次可调用一个过程，当过程中的指令执行完后，控制返回调用它的地方。

在 8086/8088 中能调用过程和从过程返回的指令是 CALL 和 RET。 这些指令分为两种情况：段内的和段间的指令。

段间的指令把过程应返回的地址的段地址和偏移量两者同时入栈 (CALL) 和退栈 (RET)。

段内的调用与返回指令只入栈和退栈地址偏移量。

过程定义语句的格式为：

PROCEDURE_NAME　　PROC　　[NEAR]/FAR

```
                    .
                    .
                    .
                  RET
                    .
                    .
                    .
PROCEDURE_NAME    ENDP
```

伪指令 PROC 与 ENDP 必须成对出现，利用过程定义语句可以把程序分段，以便于理解、调试和修改。

若整个程序由主程序和若干子程序组成，则主程序和这些子程序都应包含在代码段中，而主程序及子程序都可以作为一个过程，用过程定义语句定义。

过程定义语句 PROC 和 ENDP 限定一个过程，并说明它是 NEAR 或 FAR过程。汇编程序根据过程定义语句，当汇编到 CALL 过程时，知道汇编的是什么样的过程调用，当汇编到从这个过程返回时，知道是什么样的返回。例如：

```
MY_CODE     SEGMENT
UP_COUNT    PROC      NEAR
            ADD       CX , 1
            RET
UP_COUNT    ENDP
START:        .
              .
              .
            CALL      UP_COUNT
              .
              .
              .
            CALL      UP_COUNT
              .
              .
              .
            HLT
MY_CODE     ENDS
            END       START
```

因为 UP_COUNT 标明是 NEAR 过程，所以对它的调用，都汇编成段内调用，所有在它中的 RET 指令，都汇编为段内返回。

在一个过程中可以有多于一个的RET指令，并且过程中的最后一条指令可以不是RET指令，但必须是一条转移到过程中某处的转移指令。

END（ ENDP ）告诉汇编程序，程序（或过程）在哪儿结束了，但它不会使汇编程序产生一条 HLT（ 或 RET ）指令。

5．结束语句

除了结束语句 END 外，每个结束语句与对应的开始语句成对出现，例如，SEGMENT和ENDS， PROC 和 ENDP。

END语句标志整个源程序的结束，它告诉汇编程序，汇编工作至此结束，END语句的格式为：

```
END          <表达式>
```

其中表达式必须产生一个存贮器地址，这个地址是当程序执行时，第一条要执行的指令的地址。上面例子中。

```
START:          .
                .
                .
              END    START
```

就说明了END语句的使用。

四、条件汇编语句

IBM 宏汇编，提供条件汇编功能。各种条件汇编语句的一般格式为：

```
  IFXX    ARGUMENT
      .
      .
      .          (语句体1)
      .
      .
  [ELSE]  (任选)
      .
      .
      .          (语句体2)
      .
      .
  ENDIF
```

其中 ARGUMENT 表示条件，其值只有两个：不是真（True）就是假（False）。当条件为真时，汇编程序就汇编语句体 1 中所包含的汇编语句部分； 若条件是假，而语句中如果有 ELSE 以及语句体 2的话，则汇编程序就跳过语句体 1，对语句体 2 中的语句进行汇编；但若条件是假，而语句中没有 ELSE 及语句体 2 ， 则汇编程序就跳过这一组条件汇编语句，往下进行。ENDIF 是任一种条件语句的结束符。

IBM 宏汇编提供如下的条件汇编语句：

(1) IF　<表达式>

若表达式的值不为0，条件为真。

(2) IFE　<表达式>

若表达式的值为0，条件为真。

(3) IF1

若汇编程序正处于对源程序的第一遍扫描的过程中，条件为真。

(4) IF2

若汇编程序正处于对源程序的第二遍扫描的过程中，条件为真。

(5) IFDEF <符号>

若指定的符号已被定义或已由 EXTRN 伪指令外部说明，则条件为真。

(6) IFNDEF <符号>

若指定的符号未被定义或没经 EXTRN 伪指令外部说明，则条件为真。

(7) IFB <参量>

若参量是空格，则条件为真。

(8) IFNB <参量>

若参量不是空格，则条件为真。

(9) IFIDN <参量1>，<参量2>

若参量1的串与参量2的串相同，则条件为真。

(10) IFDIF <参量1>，<参量2>

若参量1的串与参量2的串不同，则条件为真。

第二节 结构

上一节介绍的数据定义语句，可以为变量定义多个字节的数据。如果要存取其中某一个数据，就需要程序员预先计算好这个数据所在地址，然后写出对应的地址表达式。对于比较复杂的问题就显得很繁琐。例如，学生简历一般要占据许多字节，其中又可能包括许多项。如学员姓名，年龄，系别等。如果先为表中各项分配以不同的符号名称，使得程序员在找到表头以后，直接利用符号名称存取表中各项，则将为编写程序带来很大的方便。IBM PC 宏汇编语言中的结构就提供了这种功能，这是一般的汇编语言所不具有的。

一．结构的定义

用STURC和ENDS把一系列数据定义语句括起来就成了一个结构。其格式为：

```
结构名      STRUC
        <数据定义语句序列>
结构名      ENDS
```

结构内数据定义语句中的变量名，现叫做结构字段名。

例如：

```
SAFY STRUC
NO   DB  ?
NAME DB "ABCDEFG"
SAGE DB  ?
SAFY ENDS
```

二、结构的存贮分配和预置

结构定义仅仅是告诉汇编程序存在这样一种形式的变量。只有进行结构的存贮分配和预置后，才使结构变量真正占有内存。

格式为：

结构变量名　　结构名　　<字段值表>

其中，结构名是结构定义时用的名字。结构变量名与具体的存贮空间及数据相联系，程序中可以直接引用它。字段值表用来给结构变量赋初值，其中各字段值的排列顺序及类型应与结构定义时相一致，中间以逗号分隔。如果采用在结构定义时的初值，仅写一个逗号即可；若所有字段都如此，仅写一个尖括号即可。尖括号何时也不可省略。只有在结构定义中具有一项数据的字段(包括字符串字段)才可重新赋初值。

例如：A1　SAFY <1，"ZHANG"，>

　　A2　SAFY < >

已经定义了一个结构之后，就可以通过结构的存贮分配和预置得到多个结构变量。如前所述，可将学生简历定义为一个结构，根据每个学生的情况组成多个字段值表，再用多条结构预置语句就可以产生一个学生登记表了。

三、结构及其字段的引用

要引用结构变量，可以直接写结构变量名。要引用结构变量中某一字段，则要采用如下形式：

结构变量名·结构字段名

也可以预先将结构变量的起始地址，偏移量送往某个地址寄存器，再用［地址寄存器名］代替结构变量名。例如：

```
    MOV AL，A1.NO
或  MOV BX，OFFSET A1
    MOV AL，[BX].NO
```

例题：引用结构，在数据段定义和预置结构，在代码段引用结构。

```
；定义结构
LIKE  STRUC
TO    DW   0
FRM   DW   ?
INO   DB   99 DUP(?)
LIKE  ENDS
；存贮分配和预置结构
MAS   LIKE <，5>
TXN   LIKE<，9，100>
CHA   LIKE 500 DUP(<  >)；预置500个结构变量
```

```
        ; 如下程序将CHA中所有FRM字段预置0
        MOV   BX , OFFSET CHA
        MOV   [BX].FRM , 0
        MOV   SI ,  TYPE CHA        ; SI←一个结构的字节数
        MOV   CX , LENGTH CHA-1     ; CX←结构变量个数-1
PLG:    MOV   [BX+SI].FRM , 0
        ADD   BX , SI
        LOOP  PLG
        ; 将MAS中FRM字段改为TXN中TO字段的值
        MOV AX , TXN.TO
        MOV MAS.FRM , AX
```

结构在多次定位同样的存贮格式，单个文件多重缓冲，列表处理及栈寻址等方面特别有用。

第三节　记录

到目前为止，我们能以变量名存取的最小存贮单位是字节。但实际应用中，某些量不需要一个字节，如开关量数据只需一位。如果也让它占用一个字节，就会造成存贮浪费。如果为某些量分配一位或几位，那么程序如何方便地存取这一位或几位呢？记录为解决这个问题提供了方便。记录与结构是相似的，不过记录处理按位计算的信息。

一·记录的定义

格式：

记录名 RECORD 字段名：宽度[=表达式] [，...]

其中记录名和字段名必须具有唯一性。宽度表示相应字段所占的位数（1～16位），如果各字段的宽度和大于 8位，那么汇编程序按字处理，否则按字节处理。若所定义的总位数少于 8 位或 16 位，那么所有字段就靠右对齐到字节或字的最低有效位位置。表达式是赋给相应字段的初值，可以省略。

若某字段的位数为7时，可定义为一字符，如 FID：7='A'。

例如：某工作人员情况，工龄占 6 位，性别占一位（ 0 表示男，1 表示女 ），健康状况占一位（ 0 表示健康，1 表示不健康 ），现在将其定义为记录如下：

```
BTRC RECORD YER: 6 , SEX: 1 , STAU: 1=0
```

二、记录的存贮分配和预置

与结构定义一样，记录定义只提供一个记录的样板，只有经过存贮分配和预置后，才真正占有内存。

格式：

记录变量名　记录名　〈字段值表〉

其中字段值表是赋给各字段的初值，用尖括号括起，各项间用逗号分隔，各项的顺序应与记录定义时相一致，若某一或某几个字段都采用记录定义时的初值，对应项可省略，仅写逗号即可，若仅写尖括号，表示全部采用定义时的初值。例如：

```
ZHA BTRC        <001000B，1B，>
WAN BTRC        <010000B，1B，1B>
```

将在存贮器中分配两个记录变量 ZHA（其值为 22H）和 WAN（其值 43H）。

三·记录操作符

对记录进行操作的专用操作符有三个：WIDTH、记录字段名、MASK。

1.WIDTH

格式：WIDTH 记录名或记录字段名

WIDTH 返回记录或记录字段所占的位数。例如：

```
MOV DH，WIDTH BTRC ；DH←8
MOV AL，WIDTH YER  ；AL←6
```

2.记录字段名

这个操作符直接引用，不带操作数，它返回一个立即数，表示该字段移到所在记录的最右边所需的移位次数。

例如：

```
MOV CL，YER        ；CL←2
```

3. MASK

格式：　MASK　记录字段名

MASK 返回一个 8位或16位的二进制数，这个二进制数中相应于该字段的各位为1，其余各位为0。

```
MOV BL，MASK SEX ；BL←00000010B
MOV BH，MASK YER ；BH←11111100B
```

记录操作符可以与运算符 NOT、OR、AND 及操作符 SHL（左移）、SHR（右移）配合使用。例如：

```
MOV CH，NOT MASK SEX ；CH←11111101B
MOV CL，12 SHL SEX   ；CL←12左移1位
```

四、记录及其字段的引用

利用各种记录操作符可对记录及其字段进行操作。

例如，下述程序可根据工作人员的条件做不同的安排，条件是：

工龄≥10年，女性和健康。

```
WAN DB ?
BTRC RECORD YER：6，SEX：1，STAU：1
```

```
        ; 判断工作人员是否满足指定的条件
        MOV   AL , WAN
        TEST  AL , MASK SEX
        JZ    RJT
        TEST  AL , MASK STAU
        JNZ   RJT
        MOV   CL , YER
        SHR   AL , CL
        CMP   AL , 10
        JL    RJT
        ; 满足条件时的安排
              .
              .
              .
        ; 不满足条件时的安排
RJT:          .
              .
              .
      HLT
```

第四节　宏指令语句

一．宏指令的用途

1.在汇编语言源程序中，若有的程序段要多次使用，为了使在源程序中不重复书写这个程序段，可以用一条宏指令来代替。由汇编程序在汇编时产生所需要的代码。

例如，为了实现 ASCII码与 BCD码之间的相互转换，往往需要把 AL 中的内容左移四位或右移四位，这当然可以用 8086/8088 的指令来实现。若要左移四位，可用：

```
MOV   CL , 4
SAL   AL , CL
```

若要多次使用，就可以用一条宏指令来代替，宏定义如下所示：

```
SHIFT   MACRO
        MOV     CL , 4
        SAL     AL , CL
        ENDM
```

这样，以后凡要使 AL 中的内容左移四位，就可以用一条宏指令 SHIFT来代替。

宏定义中，SHTFT 是宏指令名，MACRO 是宏定义的定义符，ENDM 是宏定义的结束符，这两者必须成对出现。

在 MACRO 与 ENDM 之间的指令序列是宏体，即要用宏指令来代替的程序段。它是由 IBM 宏汇编的指令语句（可执行语句）和管理语句（即由伪指令构成的语句）所构成。

经宏定义后，就可以引用宏指令，这称作宏调用。宏汇编程序遇到这样的调用时，就用对应的宏体来代替这条宏指令，以产生目的代码，这称作宏展开。

2.宏定义不但能使源程序的书写简洁，而且由于宏指令具有接收参量的能力，所以功能就更灵活。

例如上述的宏指令只能使 AL 中的内容左移四位。若每次使用时，要移位的次数不同；或要使不同的寄存器移位，就不方便了。但是，若在宏定义中引入参量，就可以满足上述要求。

```
SHIFT    MACRO    X
         MOV      CL，X
         SAL      CL，CL
         ENDM
```

其中，X 是一个形式参量，(这儿用来代表移位次数。)在调用时可把实际要求的移位次数作为实在参量代入，如

```
SHIFT    4
```

就可以用实在参量 4 代替在宏定义体中出现的形式参量 X ，而实现移位4次。又如：

```
SHIFT    6
```

就可以左移 6次。这样，就可以由调用时的实在参数来规定任意的移位次数。若我们再引入一个形式参量

```
SHIFT    MACRO    X，Y
         MOV      CL，X
         SAL      Y，CL
         ENDM
```

用形式参量 Y 来代替需要移位的寄存器。只要在调用时，把要移位的寄存器作为实在参量代入，就可以对任一个寄存器实现指定的左移次数。

```
SHIFT    4，AL
SHIFT    4，BX
SHIFT    6，DI
          ⋮
```

在汇编这些宏指令时，分别产生以下指令语句，宏汇编程序在每一条由宏展开产生的指令前冠以加号 " + "：

```
+ MOV    CL，4
+ SAL    AL，CL
```

```
+ MOV     CL , 4
+ SAL     BX , CL
+ MOV     CL , 6
+ SAL     DI , CL
```

第一条宏指令使 AL 左移4位；第二条宏指令使 BX (16位寄存器)左移4位；第三条宏指令使 DI 左移6位。

3.形式参量不仅可以出现在操作数部分，也可以出现在操作码部分。如：

```
SHIFT     MACRO     X , Y , Z
          MOV       CL , X
          S&Z       Y , CL
          ENDM
```

其中第三个形式参量 Z 代替操作码中的一部分。在 IBM 宏汇编中规定，若在宏定义体中的形式参量没有适当的分隔符，则不被看作为形式参量，调用时也不被实在参量所代替。例如上例中的操作码部分 S&Z 中 ，若 Z 与 S 之间没有分隔，则此处的 Z 就不被看作形式参量。 要定义它为形式参量，必须在前面加上符号 & 。所以 S&Z 中 Z 就被看作形式参量。若有以下调用：

```
SHIFT     4 , AL , AL
SHIFT     6 , BX , AR
SHIFT     8 , SI , HR
```

在汇编这些宏指令时，分别产生以下指令语句。

```
+ MOV     CL , 4
+ SAL     AL , CL
+ MOV     CL , 6
+ SAR     BX , CL
+ MOV     CL , 8
+ SHR     SI , CL
```

现在的 SHIFT 宏指令可以对任一个寄存器、 进行任意的移位操作 (算术左移，算术右移、逻辑右移，移位任意指定的位数。

由此可见宏指令的使用是十分灵活的。

二．IBM 宏汇编中的主要宏操作伪指令

1. MACRO

其一般格式为：

```
宏指令名     MACRO    <形式参量表>
                .
                .
                .     宏体
```

```
        :
        ENDM
```

宏指令名是一个宏定义调用的依据，也是不同宏定义相互区分的标志，是必须要有的。对于宏指令名的规定与对标识符的规定是一致的。

宏定义中的形式参量表是任选的，可以没有形式参量，也可以有若干形式参量。若有一个以上的形式参量时，它们之间必须用逗号分隔。对形式参量的规定与对标识符的规定是一致的。形式参量的个数没有限制，只要一行限制在 132 个字符以内就行。

在调用时的实在参量多于 1个时，也要用逗号分隔，它们与形式参量在顺序上一一相对应。但 IBM 宏汇编并不要求它们在数量上必须一致。若调用时的实在参量多于形式参量，则多余的部分被忽略；若实在参量少于形式参量，则多余的形式参量变为NULL(空)。

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 伪指令的格式为：

```
        PURGE     宏指令名[,.....]
```

即一个 PURGE 可以取消多个宏定义。

2. REPT

其一般格式为：

```
        REPT      <表达式>
          :
          :       } 指令体
          :
          :
        ENDM
```

这个伪指令可以重复执行在它的指令体部分所包含的语句。重复执行的次数，由表达式的值所决定。

例：

```
        X = 0
          REPT 10
        X = X+1
          DB X
          ENDM
```

上例把1到10分配给十个连续的内存单元。

3. IRP

其一般格式为：

```
    IRP     形式参量，<参数表>
      :
      :     指令体
```

```
ENDM
```

此伪指令能重复执行指令体部分所包含的语句，重复的次数由参数表中的参数的个数所决定(参数表中的参数必须用两个三角括号括起来，参数间用逗号分隔)。而且每重复一次，依次用参数表中的参数来代替形式参量。

例：

```
IRP   X  ,  <1,2,3,4,5,6,7,8,9,10>
DB    X
ENDM
```

因为参数表中的参数个数为10，故指令体部分重复执行10次。上例中指令体部分只有一条伪指令 DB X，其中 X 为形式参量。在第一次执行时 用参数表中的第一个参数1代替形式参量，就为 DB 1；第二次执行时，用参数表中的第二个参数 2 代替形式参量，就为 DB 2；.....。所以上例也是把 1 到 10 分配给 10 个连续的内存单元。

4.IRPC

其一般格式为：

```
IRPC    形式参量，字符串(或<字符串>)
        :
        :   } 指令体
        :
ENDM
```

此伪指令也能重复执行指令体部分所包含的语句。重复执行的次数，取决于字符串中的字符个数。而且在每次重复时，依次用字符串的字符代替形式参量。

所以，IRPC 伪指令与 IRP 伪指令很类似，只是用字符串(此字符串可以包括在两个三角括号中，也可以不包括)代替了 IRP指令中的参数表。例：

```
IRPC      X , <ABCDEF>
DB        X
ENDM
```

把字符 A 到 F 分配给内存中的 6 个连续单元。

以上 MACRO，REPI、IRP和IRPC四个宏定义的伪指令都必须以伪指令 ENDM 作为它的结束符。

(三)宏指令与子程序的区别

宏指令是用一条宏指令来代替一段程序，以简化源程序，子程序也有类似的功能。那么，这两者之间有什么区别。

1.宏指令是为了简化源程序的书写，在汇编时，汇编程序处理宏指令，把宏定义体插入到宏调用处。所以，宏指令并没有简化目标程序。有多少次宏调用，在目标程序中仍需

要有同样多次的目标代码插入。所以，宏指令不节省目标程序所占的内存单元。

子程序或过程是在执行时，由 CPU 处理的 。若在一个源程序中多次调用同一个子程序，在目标程序的代码中，主程序中只有调用指令的目标代码，子程序的代码只是一段。

2.把上述两者的特点加以比较，可以看出：若在一个源程序中多次调用一段程序，则可用子程序， 也可以用宏指令来简化源程序。 用子程序的方法，汇编后产生的目标代码少，也即目标程序占用的内存空间少，节约了内存空间。但是，子程序在执行时，每调用一次都要先保护断点，通常在程序中还要保护现场；在返回时，先要恢复现场，然后恢复断点(返回)这些操作都额外增加了时间，因而执行时间长点，速度慢点。而宏指令恰好相反，它的目标程序长，占用的内存单元多；但在执行时不需要保护断点、现场以及恢复、返回等这些额外操作，因而执行时间短，速度快。

所以，当要代替的程序段不长，速度是主要矛盾时，通常用宏指令；而当要代替的程序段较长时，额外操作所附加的时间就不明显了，而节省存贮空间是主要矛盾，通常采用子程序。

另外，宏指令是机器的指令系统中没有的，但又可以作为一条指令使用。所以，从形式上看，宏指令扩充了机器的指令系统。

习题三

1. 指令 AND AX，OPD1 AND OPD2 中， OPD1 和 OPD2 是两个已赋值的变量，问两个 AND 操作分别在什么时间进行 ？有什么区别 ？

2. 如下指令或程序是否有错 ？错在哪里 ？

(1) K1 EQU 1024

```
        .
        .
        .
    MOV K1，AX
```

(2) MOV DS，100

```
    MOV [1000]，[2000]
```

(3) IMP DB ?

```
        .
        .
        .
    MOV  IMP，AX
```

(4) A1 DB ?

```
    A2  DB 10
        .
        .
        .
```

```
CMP   A1，A2
```

(5)将 1000 送入 X1 单元，用如下程序：

```
X1   DB ?
      :
      :
      :
MOV  BX，X1
MOV  [BX]，1000
```

3.下列语句在存贮器中分别为变量分配多少字节?

```
VR1    DW    9
VR2    DW    4 DUP(?)，2
CONT   EQU   10
VR3    DD    CONT DUP(?)
VR4    DB    2 DUP ( ?, CONT DUP (0))
VR5    DB    'HOW  ARE YOU ?'
```

4.下面语句有何区别 ?

```
X1    EQU 1000H
X2    =   1000H
```

5．分析操作符和合成操作符各有哪几种，分别举例并加以解释说明。

6．8086/8088 汇编语言程序中段的类型有几种，各段如何定义?段定义中，定位类型，组合类型、类别各起什么作用，各有什么含义?

7．代码段中，开始的一段程序有通用性，试将此段程序定义为一条宏指令。

8．利用结构的概念定义并分配一个通信录。

9．请定义一条宏指令，它可以实现任一数据块的传送(假设无地重迭)，只要给出源和目的数据块的首地址以及数据块的长度即可。

如果一条宏指令既可以从数据块的首地址又可从其末地址开始传送这个数据块，应如何定义这条宏指令 ?

第四章　程序设计的基本方法

第一节　汇编语言程序设计的基本步骤

设计一个好的程序不仅应该满足设计要求，能正常运转，实现预定的功能。还应满足：

(1)程序要结构化，简明、易读、易调试。

(2)执行速度快。

(3)占用存贮空间少(即存贮容量小)。

在初期的计算机中，由于存贮设备昂贵，容量有限，一般要尽量少占用存贮空间。随着工艺技术的发展，存贮容量和存贮设备的价格下降，存贮空间和密度大大提高，有时也就不特别计较容量问题了。但是在存贮空间一定时，还是要设法降低程序的存贮容量。

程序的执行速度问题，在某些实时控制、跟踪等程序中显得特别突出。例如：计算机控制的地对空射击，如程序运行太慢，就不能跟踪目标，失去意义。

速度和容量问题有时是矛盾的，在许多情况下要加以权衡，看哪一方面最主要。随着计算机事业的发展，程序的日渐庞大和复杂，使程序结构化，显得越来越重要，要写好程序文件(包括注释、说明)以便于阅读、调试，以及与其它程序的连接，以便于共享等。

一．汇编语言程序设计的基本步骤

(1)分析问题，抽象出描述问题的数学模型。

(2)确定解决问题的算法思想。

(3)绘制流程图或结构图。

(4)分配存贮空间及工作单元(包括寄存器)。

(5)逐条编写程序。

(6)静态检查。

例：编一程序查找高考生的最高英文分数，假设所有考生分数已存入计算机内。

1．分析问题

拿到一个问题后，首先要全面分析它，看它给出了什么条件，有什么特点，找出其规律性，归纳出数学模型。就本例而言，每个学生的分数已给出，成绩分数是正数，一般在0～100之间，若有加试分，一般也越不出0～200范围，那么学生分数是0～200之间的正数集合，记为 {S}。找出最高分数，亦即在集合中找出最大数，记为 max {S}。有些很简单的问题，不一定非要写出数学模型不可，但有了数学模型以后，《计算方法》等课程中的许多行之有效的算法可直接利用。

2．确定解决问题的算法思想。

如果已有了数学模型，能够直接利用或间接利用一些计算方法和程序设计方法是最好的。若没有现成方法可用，也可从人类社会实践中寻找启发，总结出算法思想。

本例中，若是人工的方法找最高分数，则是拿到分数单，从第一个分数开始向下看，一边看一边将两个分数比较，看到较高的分数就记在脑中，较低的就置之不理，再有更高的分数时，脑子中就记住这个更高的分数，而遗弃前面较高者。以次类推，直到所有分数看完，最高分数就在脑子中形成了。从这个人工过程我们可以得到启发，计算机如何做这件事呢？

现在可以归纳算法思想：建立一个数据指针指向数据区的首地址，将第一个数取入某个寄存器(如AL)中，与下一个数比较，若下一个数较大，就将它取入寄存器中，否则不理它，然后调整数据指针，再与下一个数比较，・・・直至比较完毕，结果，寄存器中就为最高分数了。

读取一个个分数，计算机用MOV指令，比较数的大小用CMP指令，分析判断当前的分数是否比较大，用条件转移指令等等。

3．绘制流程图或结构图

流程图一般是利用一些带方向的线段，框图等把解决问题的先后次序等直观地描述出来。流程图种类比较多，如逻辑流程图，算法流程图，程序流程图等等，复杂问题可以画几级流程图，先画出粗框图，再逐步求精，对于复杂问题，程序就要分成一个个模块，画出模块间结构图。

这个例题的程序流程图容易画，根据算法思想将解决问题的顺序描述出来即可，见图4-1。

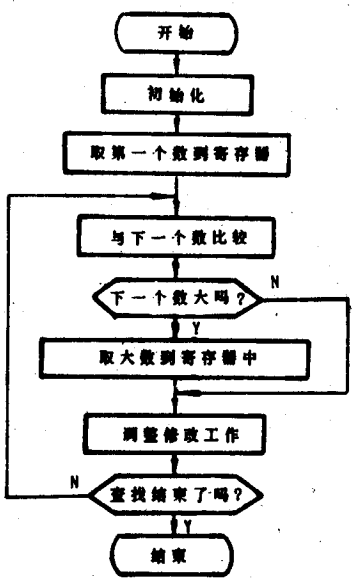


图 4-1 程序流程图

为了判断查找工作是否结束，可以在初始化部分设计一个计数器，将学生人数（分数个数）减 1 后送入计数器，每查看比较一次，计数器减 1，减到零时，查找工作就结束了。初始化部分还应包括建立一个指针指向数据区。

4．分配存贮空间和工作单元

8086/8088 存贮器结构要求存贮空间分段使用，因此要分别定义数据段，堆栈段、代码段以及附加段。工作单元可以设置在数据段中的某些存贮单元，但最好是 CPU的内部寄存器。例如：我们把学生英文分数分配在数据区，定义 100个字节的堆栈空间，利用 BX 寄存器作数据指针，用 CX 作计数器，

用 AL 存最高分数。

5．逐条编写程序

一个问题，有了算法思想和程序流图以后就简单了。可以在不同机器上编程序实现（不同机器有不同的指令系统）。我们利用 8086/8088 的指令系统，按流程图的要求，编写出程序如下：

```
DATA SEGMENT
FEN   DB 85，90，．．．．                ；定义学生分数
DATA  ENDS
STACK SEGMENT PARA STACK ' STACK '
      DB 100 DUP (?)                     ；定义100个字节堆栈空间
STACK ENDS
CODE  SEGMENT
      ASSUME CS: CODE，DS: DATA，SS: STACK
START PROC FAR
      PUSH DS
      MOV AX，0
      PUSH AX                            ；保护返回地址
      MOV AX，DATA
      MOV DS，AX                         ；设置数据段寄存器
      MOV BX，OFFSET FEN                 ；设置数据指针
      MOV CX，LENGTH FEN                 ；设置计数器
      DEC CX
      MOV AL，[BX]                       ；取第一个分数
LOP:  INC BX                             ；调整指针
      CMP AL，[BX]                       ；与下一个数比较
      JAE NEXT                           ；下一个数不大转移
      MOV AL，[BX]                       ；取这数送入AL
NEXT: DEC CX                             ；计数器减1
      JNZ LOP                            ；计数器≠0，转移到LOP
      RET
START ENDP
CODE  ENDS
      END START
```

6．静态检查

程序编好以后，首先要进行静态检查，看程序是否具有所要求的功能，程序是否清晰

易读，选用的指令尽量字节数少，执行速度快，易错的地方要重点检查，如比较次数应是分数个数减 1，因为第一个数已先取入寄存器；转移条件是否正确等。有时要从头检查，查看每一步的正确性。静态检查，认为无错后，才可上机调试。当然上机调试发现问题后，仍可能从某一步或从第一步检查。

二、流程图的画法

利用流程图进行程序的设计是一种基本方法。对于复杂程序，可以先将其分成一个个模块，即把大事化小，然后再对每个模块进行设计，逐步求精，对一个个具体模块仍可使用流程图法设计。

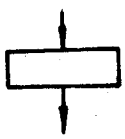
流程图的作用是明显的，象写文章一样，先列一个提纲，先对程序的结构作全局性的安排。先做什么，后做什么，把整个程序的先后次序，执行步骤用框图直观而清晰地表示出来。

流程图一般由四部分组成：

1．执行框 （矩形框）

执行框中写明某一段程序或某一模块的功能。其特点是有一个入口一个出口。

形如：

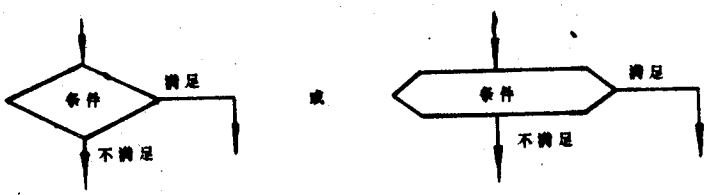


2．判别框

可用菱形或尖角形框表示，框内写明比较、判断的条件。条件较长时用尖角框合适。它可有一个入口两个出口，在各个出口处要写明条件，若条件成立则写入“是”或“Y”或“1”，条件不成立则写“否”或“N”或“0”。

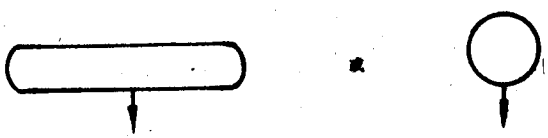
形式如：

菱形框　　尖角框

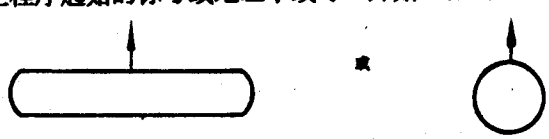


3．起始框和终止框：

表示程序段的起始和终止。起始框有一个出口：形如：



框内可以写上程序起始的标号或地址，或写“开始”。终止框有一个入口。形如：



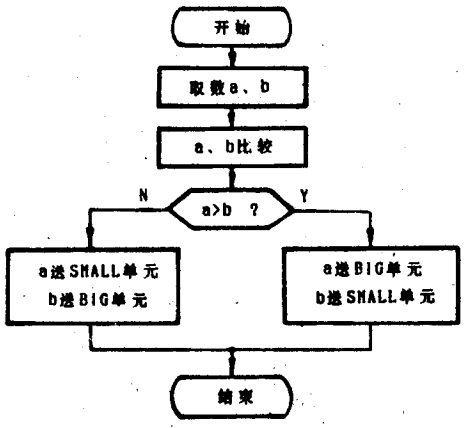


图 4-2 程序流程图

框内可写入“暂停”、“结束”、“返回”等。

4．指向线：

指向线由带箭头的线段组成，它表示程序执行的顺序。例如：比较a、b两数的大小，将大者送入 BIG 单元，小者送入SMALL单元。其程序流程图如图4-2。

第二节 程序的基本结构形式

程序的基本结构形式有三种：顺序结构、分支结构和循环结构。

1．顺序结构

从流程图上看，顺序结构的程序只有一个起始框、终止框和一至几个执行框。程序将顺序执行，无分支、无循环、无转移。顺序结构的程序一般是简单程序。

例1．编程序：按下列公式计算Y的值，X是存放在FIRST单元的一字节数。

Y = X*X-50

程序可如下：

```
MOV AL，FIRST
MUL AL                    ；AX中为X*X
SUB AX，50
HLT
```

程序执行结果：AX中为 Y 的值。

例2：内存中自 TABLE 开始的七个单元连续存放着自然数 0至 6 的立方值(称作立方表)。任给一数 X (0≤X≤6) 在 XX 单元，查表求 X 的立方值，并把结果存入YY单元。

求某数 X 的立方值可用连乘方法，即 X*X*X 。这里已给出一立方表，可用查表方法求得 X 的立方值。

分析一下表的存放规律，可知表的起始地址与数X的和，正是X的立方值所在单元的地址。由此可得程序如下：

```
DATA  SEGMENT
TABLE DB 0，1，8，27，64，125，216
XX    DB ?
```

```
YY    DB ?
DATA  ENDS
STACK SEGMENT PAYA STACK  'STACK'
      DB    50 DUP(?)
STACK ENDS
COSEG SEGMENT
      ASSUME CS: COSEG , DS: DATA , SS: STACK
START PROC FAR
      PUSH DS
      MOV AX , 0
      PUSH AX
      MOV AX , DATA
      MOV DS , AX
      MOV BX , OFFSET TABLE
      MOV AH , 0
      MOV AL , XX
      ADD BX , AX
      MOV AL , [BX]
      MOV YY , AL
      RET
START ENDP
COSEG ENDS
      END START
```

2．分支结构

一般情况下，程序按顺序方式执行，要机器自动根据不同情况做出判断、选择，以执行不同的程序，这就需要分支结构程序。计算机的分析、判断能力就是这样实现的。

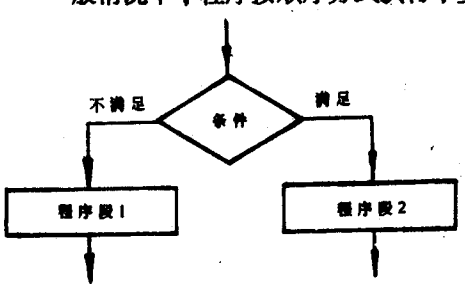


图 4-3 分支结构图

分支结构的基本形式如图4-3 。

当分支条件满足时执行程序段 1，不满足时执行程序段 2。当程序段 2为空时，分支结构为简单形式。

3．循环结构

在日常生活中，需要重复地做某些事的情况是很多的。凡需要重复做的工作，在计算机中可用循环结构程序实现。顺序结构程序中的指令只执行一次就结束了，分支结构程序则要根据条件的不同，跳过一些指令转向执行另一些指令，最多也执行一次，而循环结构

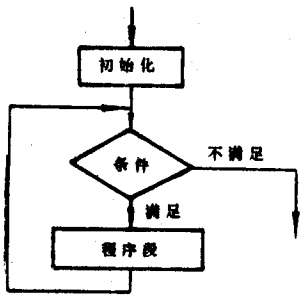


图 4-4 循环结构图

程序中的某些指令则要重复执行多次，这使程序长度大大缩短。

循环结构的基本形式如图4-4：

每次测试循环条件，当条件满足时，重复执行程序段，否则结束循环，程序顺序向下执行。

第三节 分支程序设计

一、转移指令

转移指令分为条件转移和无条件转移指令，这类指令的特点是改变程序执行的顺序。

1．无条件转移指令： JMP 标号

JMP指令无条件地转移到目标单元。

一个段内直接JMP指令，用从JMP指令得到的目标的相对偏移量加到指令指针上以改变指令指针。

若汇编程序能确定目标是在距JMP指令的±127字节之内时，就自动产生一个叫做短（SHORT）JMP 的两字节指令；否则，就产生一个能在±32K 范围内寻址目标的 NEAR JMP。

例 JMP WIN

WIN 是一个 NEAR 型或 SHORT 型标号。

一个段内间接 JMP指令，其目标地址或是由一个16位通用寄存器，或是由内存单元来寻址。在前一种情况下，IP取自指令中指定的寄存器，后一种情况下，IP的值由指令中指定的内存字单元的内容代替。

例如： JMP CX

JMP WORD PTR [BX]

在一个段间直接JMP指令中，由包含在指令中的值代替CS和IP。

例： JMP FAR PTR TABL

一个段间间接JMP指令，由指令指定的双字指针的第一个字单元的内容代替 IP，第二个字单元的内容代替 CS。

例：JMP DWORD PTR [BP][DI]

所寻址的单元是定义为双字的单元。

2．条件转移指令

8086/8088 有一连串的条件转移指令，它以某些标志位的逻辑运算作为依据，若满足指令所规定的条件，则程序转移，否则顺序执行。在这类指令中，转向的目的地址必须在

转移指令的+127或-128个字节之内，这些指令对标志位无影响，指令的助记符也不唯一。但是归纳一下主要可分成三类：

(1)根据单个标志位的条件转移指令。

(2)用于无符号数的条件转移指令。

(3)用于有符号数的条件转移指令。

下面我们分类加以介绍：

(1)根据单个标志位的条件转移指令。

这类指令共有五种十条

①JC或JNC

这两条指令是根据标志位 CF 的值进行转移。JC 是当 CF=1（即有进/借位时）时转移到目标地址，而JNC是当 CF=0 时转移到目标地址的条件转移指令。

②JE/JZ或JNE/JNZ

这两条指令是根据标志位 ZF 的值进行转移。JE/JZ（即相等转移/等于 0 转移)是当ZF=1时进行转移，这是一条指令的两种助记符；JNE/INZ(即不相等转移/不等于 0 转移）是当 ZF=0 时进行转移。

③JS/JNS

这两条指令根据标志位 SF 的值进行转移。JS是当符号标志 SF=1(即负数）时转移，JNS 是当 SF=0（即为正数）时转移。

④JO/JNO

这两条指令根据溢出标志OF的值转移。JO是当OF=1(即溢出时)时转移，JNO 是当溢出标志OF=0时转移。

⑤JP/JPE或JNP/JPO

这两条指令根据奇偶标志PF的值转移。

JP/JPE(即偶转移)是当PF=1时转移，JNP/JPO(即奇转移)是当PF=0时转移。

```
例:    CMP CX，DX
       JE   LAB2          ;  当CX=DX(或CX-DX=0)时，转至LAB2
       INC CX
LAB2:  MOV AX，0
         ⋮
```

(2)用于无符号数的条件转移指令：

这类转移指令有四条。

①JA/JNBE

JA即高于转移，JNBE即不低于且不等于转移，所以两种写法是同义的。JA/JNBE 是当 CF=0 且 ZF=0时转移。它用于两个无符号数a、b比较时，若a>b，则满足条件实现转移。

②JAE/JNB

JAE/JNB(即高于或等于转移/不低于转移)是当CF=0或ZF=1时转移。它用于两个无符号数a、b比较时，若a≥b则满足条件实现转移。

③JB/JNAE

JB/JNAE(即低于/不高于且不等于转移)是当CF=1且ZF=0时转移。它用于两个无符号数a、b比较时，若a < b则满足条件实现转移。

④JBE/JNA

JBE/JNA（即低于或等于/不高于转移）是当CF=1或ZF=1时转移。它用于两个无符号数a、b比较时，若a≤b则满足条件实现转移。

(3)用于有符号数的条件转移指令：

这类指令共有四条。

①JG/JNLE

JG/JNLE（大于/不小于且不等于转移）是当标志 SF与 OF 同号（即（SF异或 OF)=0）且 ZF=0 时转移。它用于两个带符号数 a、b 比较时，若 a>b 则满足条件转移。

②JGE/JNL

JGE/JNL（大于或等于/不小于时转移）是当标志 SF 与 OF 同号（即(SF异或OF)=0）或 ZF=1 时转移。它用于两个带符号数a、b比较时，若a≥b则满足条件转移。

③JL/JNGE

JL/JNGE(小于/不大于也不等于转移)是当标志SF与OF异号（即（SF异或OF）=1)且ZF=0时转移。它用于两个带符号数a、b比较时，若a<b，则满足条件转移。

④JLE/JNG

JLE/JNG（小于或等于/不大于转移)是当标志SF与OF异号(即(SF异或OF)=1)或 ZF=1时转移。它用于两个带符号的数比较时，若a≤b则满足条件转移。

二、分支程序设计

分支的实现有多种方法，这里介绍两种基本类型。

1．利用比较转移指令实现分支

用于比较、判断的指令如：两数比较指令 CMP，串比较指令CMPS，串搜索指令SCAS，用于实现转移的指令如：无条件转移指令 JMP和各种类型的条件转移指令。它们都可以互相配合以实现不同情况的分支　　对于多路分支的情况，可以采用多次判断转移的方法实现。每次判断转移形成两路分支，n次判断转移可以形成n+1路分支。

例 1．符号函数 $Y=\begin{cases}1 & \text{当 } X>0 \\ 0 & X=0 \\ -1 & X<0\end{cases}$　　(-128 <= X >= 127)

抽象地看，符号函数仅是一个数学问题，但可以把它用于实际问题。例如：关于产品的分类、包装等问题就可以应用符号函数。假设这里所指的产品是机床加工的轴承。用自动检测系统测量轴承的直径，若直径误差在允许的范围内，打印标记 0；若直径大于标准

值，打印标记 1；其余的打印标记 -1 。接下来，自动分类系统可以根据标记进行分类。读到标记 0 的产品，可由传送带送到成品包装线去包装出厂；标记为 1的产品可以送回车间重新加工；标记为 -1 的产品是尺寸太小的，若修补费用太高的话，只好将它送到废品库去。

这里假设任意给定的 X 值存放在 XX 单元，函数 Y 的值存放在 YY 单元。那么根据 X 的不同取值给 Y 赋值的程序如下：

```
        MOV  AL，XX
        CMP AL，0
        JGE BIGR
        MOV AL，0FFH
        MOV YY，AL          ；X<0时，-1送入YY单元
        HLT
BIGR:   JE EQUL
        MOV  AL，1
        MOV YY，AL          ；X>0时1送入YY单元
        HLT
EQUL:   MOV YY，AL          ；X=0时，0送YY单元
        HLT
```

本例题中，CMP AL，0指令可以用SUB AL，0或OR AL，AL或 AND AL，AL代替。

例2． 数据块的传送

把内存中某一区域的原数据块传送到另一个区域。若源数据块与目的数据块之间地址没有重迭， 可直接用数据传送指令或串操作中的传送指令实现， 如果它们之间的地址重迭， 怎么办呢？ 可以先判断一下源地址加数据块长度是否小于目的地址。若是，可按增量方式传送； 若不， 则要把指针修改为指向数据块的底部，然后采用减量方式传送（这里假设从低地址区域向高地址区域传送)。程序如下所示：

```
DATA  SEGMENT
STRG  DB  1000 DUP(?)
STG1  EQU STRG+7
STG2  EQU STRG+25
STRSE EQU 50
DTAT  ENDS
STACK SEGMENT PARA STACK ' STACK '
STAPN DB 100 DUP(?)
STACK ENDS
COSEG SEGMENT
```

```
        ASSUME CS: COSEG , DS: DATA , ES: DATA , SS: STACK
GOO     PROC FAR
        PUSH DS
        MOV AX , 0
        PUSH AX
        MOV AX , DATA
        MOV DS , AX
        MOV ES , AX
        MOV CX , STRSE
        MOV SI , OFFSET ST61
        MOV DI , OFFSET ST62
        CLD                             ; 增量方式
        PUSH SI
        ADD SI , STRSE-1
        CMP SI , DI
        POP SI
        JL OK
        STD                             ; 减量方式传送
        ADD SI , STRSE-1                ; 指向数据块底部，
        ADD DI , STRSE-1
 OK:    REP MOVSB                       ; 重复传送50个数据
        RET
 GOO    ENDP
COSEG   ENDS
        END GOO
```

2．利用表实现分支

(1)跳转表的组成

跳转表可以放在内存的一片连续单元中，表中连续存放一系列跳转地址，跳转指令或关键字等。

例如：某工厂 8 种产品的加工程序 R0 到 R7 分别存放在以 SBR0，SBR1，SBR2，... SBR7为首地址的内存区域中。而这 8个首地址偏移量连续存放在以 BASE 为首址的跳转表内。如图4-5所示：

如图所示的表可称作跳转表。表开始的第一个单元的地址称作表基地址(或表首址)。要查找的元素在表中的地址叫表地址。

例如：加工程序R1的入口偏移地址SBR1在表中的地址(表地址)是 BASE+2。不同的加工

程序对应有不同的表地址。

表地址相对于基地址的偏差字节数称作偏移量。例如表地址BASE+2相对于基地址的偏移量是 2。

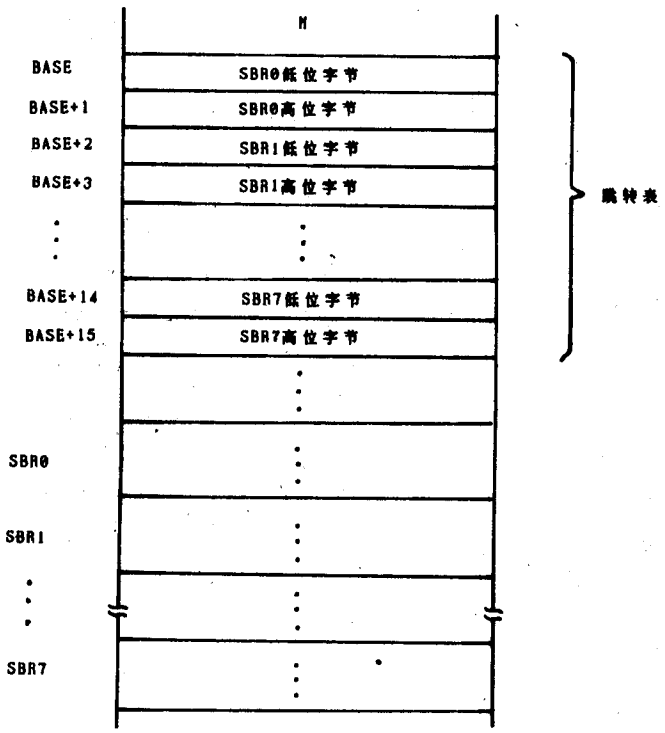


图 4-5 跳转表

(2)表的使用

①根据表内地址分支

假设上例中 8 种产品的编号分别为 0，1，... 7 。如果现在已经知道了目前要加工的产品编号，应如何编写一段程序，利用上述的跳转表自动转入该种产品的加工程序呢？问题的实质是，如何根据已知的编号从表中查出该种产品加工程序的入口地址，而首要的问题是先要求出该种产品对应的表地址。从上看出

表地址 = 表基地址 + 偏移量

这里表基地址是已知的。通过分析表的结构，可以看出偏移量由产品编号*2求得。因些表地址就可求了。

由此得到了编写程序的算法思想，画出流程图如图4-6。

具体程序如下：

```
DATA SEGMENT
BASE DW SBR0，SBR1，SBR2，SBR3
     DW SBR4，SBR5，SBR6，SBR7
BN   DB   ?        ；产品编号
```

```
 DATA ENDS
STACK SEGMENT PARA STACK 'STACK'
      DB 100 DUP(?)
STACK ENDS
COSEG SEGMENT
      ASSUME CS: COSEG , DS: DATA
START PROC FAR
      PUSH DS
      MOV  AX , 0
      PUSH AX
      MOV  AX , DATA
      MOV  DS , AX
      MOV  AL , BN
      MOV  AH , 0
      ADD  AL , AL
      MOV  BX , OFFSET BASE
      ADD  BX , AX

      MOV  AX , [BX]
      JMP  AX
      RET
START ENDP
COSEG ENDS
      END START
```

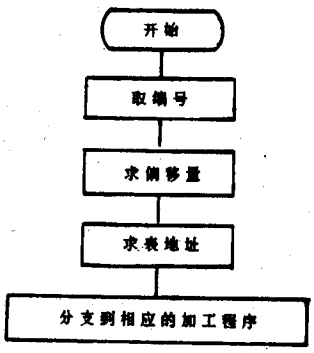


图 4-6 表分支流程图

②根据表内指令分支

许多监控程序、键盘管理程序或应用程序中常常要用到跳转表。在这些跳转表中存放的可以是跳转地址，也可以是转移指令。例如TP-801单板机的监控程序中的一个跳转表，其表内存放的是转移指令。

最原始的TP-801单板机上有12个命令键，如执行键(EXEC)，单步键(STEP)，存贮器检查键(MEM)，寄存器检查键(REG)，等等，按下任一键相当于发出TP-801的一个键盘命令。而这些命令的实现分别由监控程序中的十二段子程序完成。这些子程序的入口地址分别为：

执行键子程序入口地址：ADR0

单步键子程序入口地址：ADR1

存贮器检查子程序入口地址：ADR5

EPROM编程子程序入口地址：ADR 11

命令键跳转表存放在以BASE0为首地址的内存区域中，表内存放着转移指令，如图4-7所示：

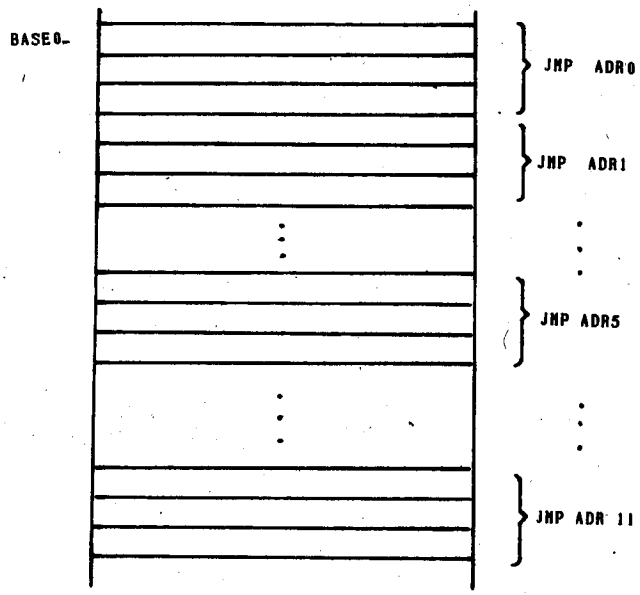


图 4-7 命令键跳转表

假设12个命令键的编号分别为 0～11。跳转表中每三个单元存放着一条转移指令。分别对应着各命令键的分支转移指令。如果我们已通过程序获得了所按下的命令键的编号X，并假设已送入寄存器 AL，那么就可编一段程序，用以实现转向相应的命令子程序去执行。程序如下：

```
MOV AH，0
MOV BL，AL
ADD AL，AL
ADD AL，BL                ；编号*3为偏移量
MOV BX，OFFSET BASE0      ；取表首址
ADD BX，AX                ；求表地址
JMP BX
```

从上述程序可以看出，仍然要根据已知量（ 编号 ）求表地址。这里是用编号乘 3得偏移量，道理很简单，每一指令在表中占了三个单元。因为这儿的跳转表中存放的是转移指令，所以不必象上例一样，先取出表地址内的地址，再实现转移，而是求出表地址后，直接转去执行表地址内的指令就可以实现正确的分支了。

再如 PC-DOS 中有50多个功能调用。也就是说有50多个功能模块可供用户直接调用，

这些模块都已编号，用户调用时只要说明编号及参数即可。由此我们可以想到，这里可以利用一个跳转表，表内存放着各功能模块的入口地址或跳转指令。同时也要有一段程序，它根据用户给定的编号，经过运算等操作，然后转到相应模块的入口去执行，以达到用户调用此模块的目的。当然方法不是唯一的。

③根据关键字分支

这类问题形式可以多种多样，关键字可以是给定的，可以是在表中的，或是表中给出了关键字的地址等等，然后根据关键字的内容分支。下面举例加以说明：

例：有一台主机为 8台外部设备服务，对于每台外设的服务程序已经编好并已分别存放在以首址为 SR0，SR1，．．．SR7的主机的内存中，每台外设有一条联络线与主机中的寄存器相连，如图4-8所示。

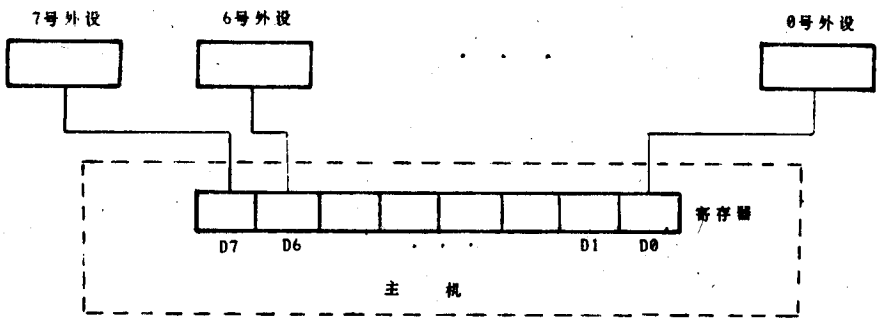


图 4-8 系统框图

平时所有联络线上均为“0”信号，当其中一台(且同一时刻只许一台）外设要求为之服务时，就在其联络线上发出“1”信号。那么由主机寄存器中的内容（称为关键字)就可以得知是哪台外设要求服务，关键字与外设的对应关系如下：

关键字：	D7	D6	D5	D4	D3	D2	D1	D0	外设
	0	0	0	0	0	0	0	1	0号外设要求服务
	0	0	0	0	0	0	1	0	1号外设要求服务
					⋮				⋮
	1	0	0	0	0	0	0	0	7号外设要求服务

现在我们可以根据关键字的值转到相应的服务程序去。同样我们首先造一张表，表内存放关键字值与其对应的外设服务程序入口地址。如图4-9所示，表的首地址为 BASE。

现设主机中寄存器的端口地址是 5FH、那么根据从寄存器中获得的关键字，并利用跳转表实现分支的程序流程图如图4-10。

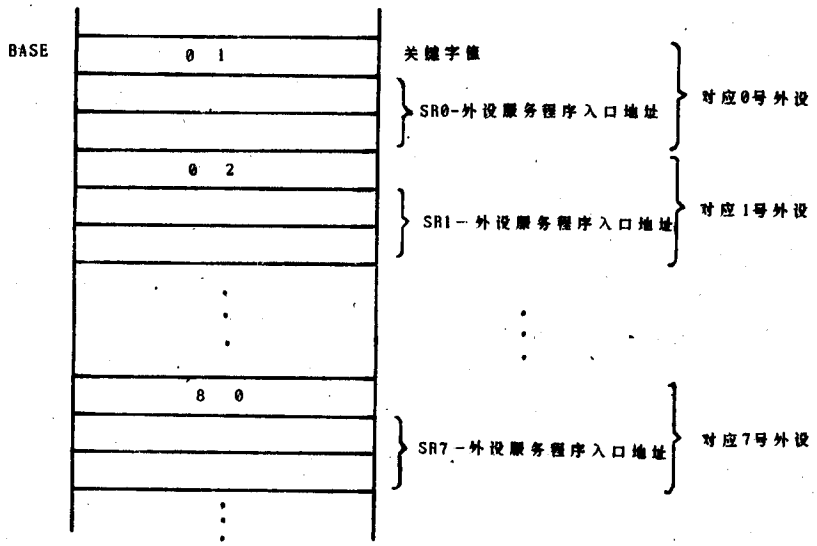


图 4-9 关键字表

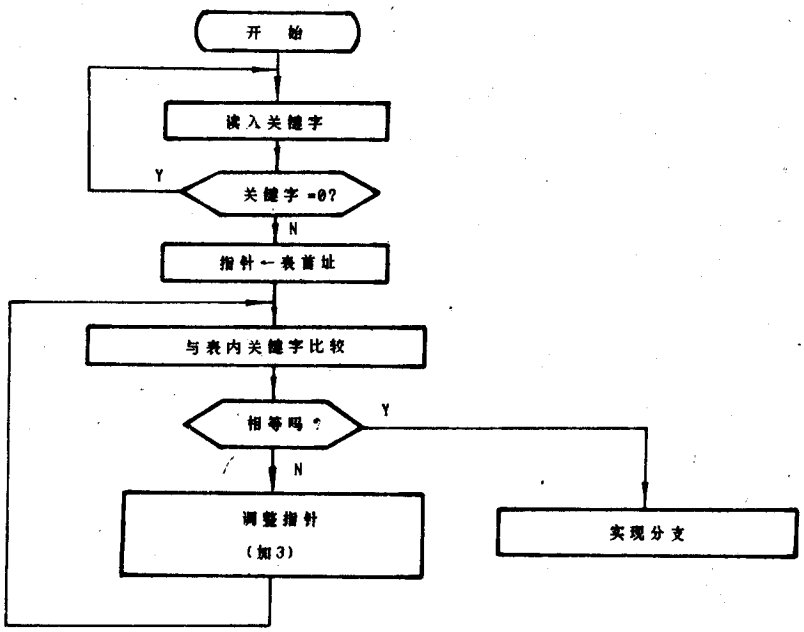


图 4-10 关键字分支流程图

具体程序如下：

```
DATA  SEGMENT
BADSE DB 1          ; 关键字
      DW SR0        ; 外设服务程序入口地址
      DB 2
```

```
        DW SR1
        DB 4
        DW SR2
          .
          .
          .
        DB 80H
        DW SR7
DATA ENDS
STACK SEGMENT PARA STACK 'STACK'
        DB 100DUP(?)
STACK ENDS
CODE SEGMENT
      ASSUME CS: CODE , DS: DATA , SS: STACK
START PROC FAR
        PUSH DS
        MOV  AX , 0
        PUSH AX
        MOV  AX , DATA
        MOV  DS , AX
LOP:    IN   AL , 5FH
        CMP  AL , 0
        JE   LOP
        MOV  BX , OFFSET BASE
RSH:    CMP AL , [BX]
        JE BRCH
        INC BX
        INC BX
        INC BX
        JMP RSH
BRCH:   MOV CX , [BX+1]
        JMP CX
              .
              .
              .
 SR0:         .
              .
              .
```

```
SR1:    .
        .
        .
SR7:    .
        .
        .
START   ENDP
CODE    ENDS
        END START
```

第四节　循环程序设计

前面说过，凡是要重复执行某些指令，最好用循环程序实现。

假如：计算　$Y = \sum_{i=0}^{100} ai$

a1，a2，a3，· · ·a100是一组已知数，求这 100 个数的和（设和不大于2字节），可由下列程序实现：

```
DATA SEGMENT
TABL DW  a1, a2, · · · , a10
     DW  a11, a12, · · · , a20
      .
      .
      .
     DW a91 , a92, · · · , a100
YY   DW ?
DATA ENDS
STACK SEGMENT PARA STACK 'STACK'
      DB 100DUP(?)
STACK ENDS
CODE  SEGMENT
      ASSUME CS: CODE , DS: DATA , SS: STACK
GO    PROC FAR
      PUSH DS
      MOV  AX , 0
      PUSH AX
      MOV  AX , DATA
```

```
        MOV  DS , AX
        MOV  AX , 0
        ADD  AX , TABL
        ADD  AX , TABL+2
        ADD  AX , TABL+4        ;   加100次
             :
             :
        ADD  AX , TABL+198
        MOV  YY , AX
        RET
GO      ENDP
CODE    ENDS
        END GO
```

上述程序中求 100项之和，共要加 100次，这样程序太长了。由于数据是有规律存放的，每加一项所用的指令是一样的，只是数据的地址不一样，所以我们可以用间接寻址的方法，将数据地址放在寄存器中，用寄存器加 1指令修改地址，则对任一加法，所用加法指令完全一样，这样可以循环使用这条指令。由此，实现上述求和运算的程序可如下：

```
        MOV AX , 0
        MOV BX , OFFSFT TABL    ; 初始化
        MOV CX , 100
LOP:    ADD AX , [BX]           ;  循环体
        INC BX
        INC BX                  ; 修改部分
        DEC CX                  ; 控制部分
        JNZ  LOP
        MOV YY , AX
        HLT
```

这个程序中，前三条指令是为循环创造条件的，称为初始化部分。第四条指令是循环的核心，包括要求重复执行的所有指令，称作循环体。第五、六条指令是修改部分，它修改操作数的地址以保证每次参加运算的是不同数据。第七、八条指令是控制部分，它用CX作计数器，每循环一次，计算器减1，直至减到0，作为循环结束条件，以达到控制循环次数的目的。第九条指令是结果处理部分。

从这个例子可以看到，循环程序缩短了程序的长度，所以凡能采用循环程序的地方都要尽量采用它，但由于需要循环准备，修改、结束控制等指令，速度稍慢些。

一．重复控制指令

重复控制指令在循环的首部或尾部确定是否进行循环。重复控制指令的目的地址必须在本指令的+127和-128字节范围之内，这些指令对标志位无影响，这些指令对于串操作及数据块操作是很有用的。8086/8088有四条此类指令。

1．LOOP 标号

LOOP指令使CX减1，且判断，若CX≠0，则循环至目标操作数，即IP+偏移量(符号扩展到16位)。

要使用LOOP指令，必须把重复次数置于寄存器CX中，一条LOOP指令相当于以下两条指令的组合：

```
        DEC CX
        JNZ AGAIN
```

所以，把数据区的数据按正、负数分开，并分别送至两个缓冲区的程序如下：

```
START: MOV SI，OFFSET BLOOK          ;  SI指向数据区
       MOV DI，OFFSET PLUS__DATA     ; DI指向正数缓冲区
       MOV BX，OFFSET MINUS__DATA    ; BX指向负数缓冲区
       MOV CX，COUNT                 ; 数据字节数送CX
GOON:  LODSB
       TEST AL，80H
       JNZ MIUS                      ; 送往DI指的缓冲区
       STOSB
       JMP AGAIN
MIUS:  XCHG BX，DI
       STOSB
       XCHG BX，DI
AGAIN: LOOP GOON
       HLT
```

2．LOOPZ（或LOOPE）标号

该指令有两种助记符 LOOPZ 及 LOOPE 。此指令使 CX-1 ，且判断只有在 CX≠0，而且标志 ZF=1 的条件下，才循环至目标操作数，即 IP + 偏移量（符号扩展到 16位 ）。

例： LOOPE AGAIN

3．LOOPNZ（或LOOPNE）标号

该指令有两种助记符 LOQPNZ和LOOPNE。此指令使 CX-1，且判断只有当 CX≠0，而且标志 ZF=0的条件下，才能循环至目标操作数。即 IP+偏移量（符号扩展到16位）。

4. JCXZ 标号 （Jump if CX=0）

若 CX=0，此指令控制转移到目标操作数，即 IP + 偏移量(符号扩展到 16 位)。

若在循环程序的开始处就跳过循环，只要使CX=0即可。

二、循环程序的基本结构

1．循环程序的组成部分

从上例可以看出循环程序一般由四部分组成。

（1）初始化部分：为循环作准备工作，包括建立指针，置计数器，设置其它变量的初值等。

（2）循环体：完成循环的基本操作，核心部分。

（3）修改部分：修改操作数地址等，为下次循环作准备。

（4）控制部分：修改计数器，查看循环控制条件，作循环控制。

各部分之间的关系如图4-11所示。有时这几部分可以简化，形成互相包含互相交叉的情况，不一定分成明显的这样四个部分。

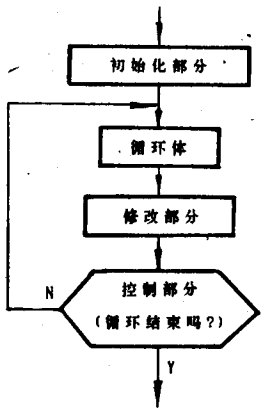


图 4-11循环结构图

2．循环程序的基本结构形式：

（1）“先执行，后判断”结构

这种结构中，进入循环后，先执行一次循环体后，再判断循环是否结束。所以这种结构至少执行一次循环体。上面所举例题属于这种结构。再如下例：

例：统计一个数据块中负元素的个数，数据块是这样定义的：

```
DATA SEGMENT
; 定义若干字节带符号数
D1   DB  -1, -3, 5, 6, 9, ...
; 供存放负元素个数使用
RS   DW  ?
DATA ENDS
```

一字节带符号数中，最高位(即符号位)为 1的数为负数。要统计负数个数即查看每一个数的符号位，并统计符号位为 1的个数。这是重复性工作，可用循环程序实现。

进而我们想到循环程序要包括四个基本部分组成，因此本题的流程图如图4-12。

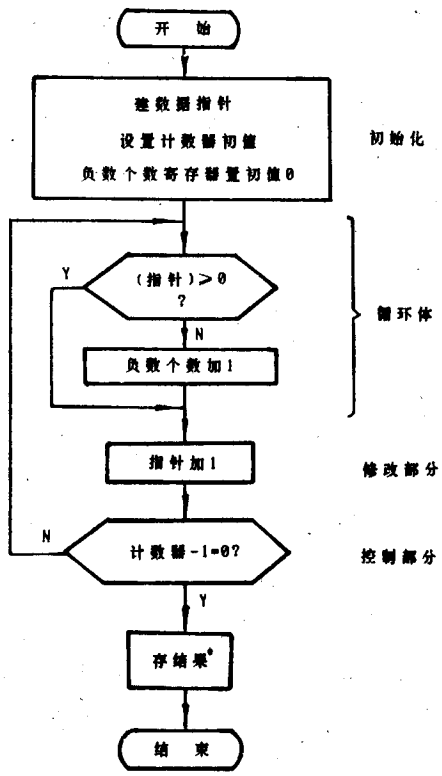


图 4-12 程序流程图

对应的代码段程序如下：

```
CODE SEGMENT
     ASSUME CS: CODE , DS: DATA
     ASSUME   SS: STACK
START PROC FAR
     PUSH DS
     MOV  AX , 0
     PUSH AX
     MOV  AX , DATA
     MOV  DS , AX
     MOV  BX, OFFSET D1     ; 建数据指针
     MOV  CX , LENGEH D1    ; 置计数器初值
     MOV  DX , 0            ; 置结果初值
LOP1: MOV  AX , [BX]
     CMP  AX , 0            ; 用 AND AX , AX 也可以
     JGE  JUS
     INC  DX
JUS:  INC  BX
     INC  BX
     DEC  CX
     JNZ  LOP1              ; 或 LOOP LOP1
     MOV  RS , DX
     RET
START ENDP
CODE  ENDS
     END  START
```

(2) “先判断，后执行”结构：

这种结构的特点是进入循环后，首先判断循环结束的条件，再视判断结果决定是否执行循环体。 这种情况下， 如果一进入循环就满足循环结束条件，就会一次也不执行循环体，即循环次数为零。因此又称为“可零迭代循环”。用流程图形式表示出来如图4-13。

当你确保一个循环程序在任何运行条件下都不会出现循环次数为零的情况下，采用以上任一种结构形式都可以，当不能确保时，用后一种结构形式为好。

例： AX寄存器中有一个 16位二进制数，编程序统计其中值为 1的位的个数。统计结果存在CX寄存器中。

这个程序最好采用“先判断，后执行”的结构，如果AX寄存器中的数为全 0，则不必再做统计工作了，程序流程图如图4-14。

从流程图看出，一进入循环，首先是控制部分，先进行判断。再者这个程序的循环体和控制部分之间没有明显的分界，具体程序段如下：

```
        MOV CX, 0
LOP:    AND AX, AX
        JZ  STP
        SAL AX, 1
        JNC NOD
        INC CX
MOD:    JMP LOP
STP:    HLT
```

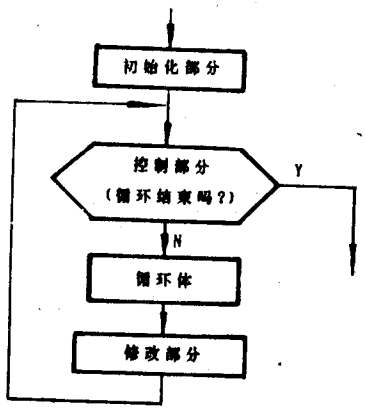


图 4-13 循环结构图

三．多重循环：

有些问题比较复杂，一重循环不够用，必须用多重循环，这些循环是一个套一个的。

例：　软件延时程序

```
SOFTDLY PROC
        MOV BL, 10
; 内循环延时10ms
DELAT:  MOV CX, 2801
WAIT:   LOOP WAIT
        DEC BL
        JNZ DELAT
        RET
SOFTDLY ENDP
```

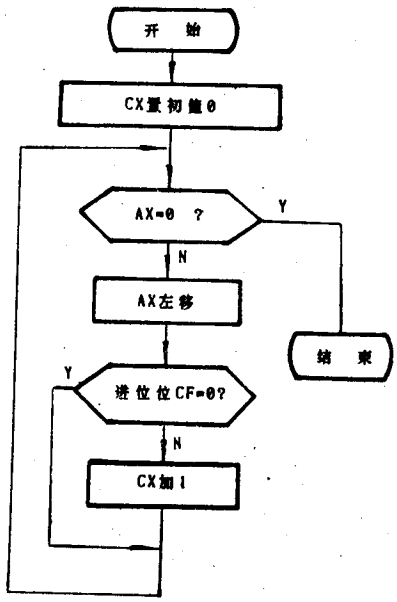


图 4-14 程序流程图

程序执行过程中，BL、CX的变化如下所示：

BL	CX
10	2801
10	2800
↓	⋮

9	0000

9	2801
↓	2800
↓	.
↓	.
↓	.
9	0000

8	2801
↓	.
↓	.
↓	.
1	0000
0	0000

在每个内循环中，CX由2801减至0，BL维持不变。大循环进行10遍。

这个子程序（过程)可以实现100ms的延时。程序本身很简单，但就其结构而言则是双重循环。程序框图如图 4-15。

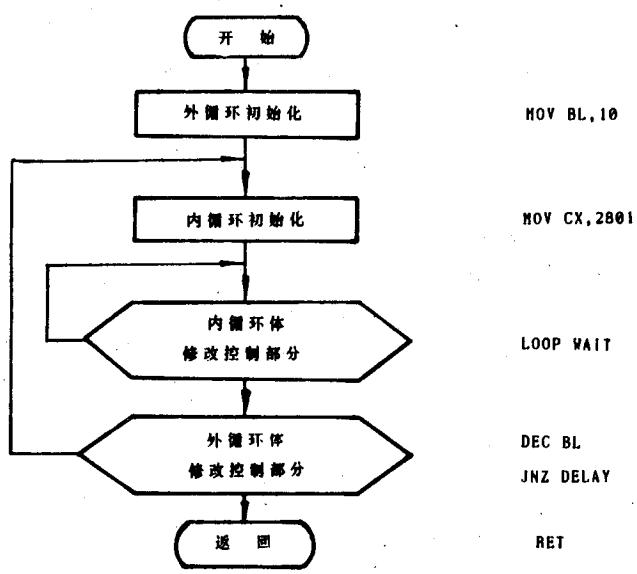


图 4-15 软件延时程序框图

程序中第二、三条指令组成内循环，CX寄存器内容从2801开始，每循环一次减 1，减至零，延时10ms。外循环中BL的初值是10，每循环一次减1，减至零时，共延时10*10ms。

两重循环程序的一般结构形式如图4-16。

从框图中可以看出内循环必须完整地包含在外循环中，循环可以嵌套、 并列， 但不可以交叉，可以从内循环中直接跳到外循环，不可从外循环中直接跳进内层循环。

注意：千万不要使循环返回到初始化部分，否则会出现死循环。例如不小心将标号DELAY或 WAIT上移一行，都会使程序陷入死循环。WAIT上移一行，内循环陷入死循环，DELAY上移一行则外循环陷入死循环。

四、循环控制方法

循环的控制方法很多，较常用的有如下几种：

1．用计数法控制循环

使用条件：计数次数已知。

使用形式：正或倒计数法。

正计数法的常用指令及格式为：

设计数次数为 n

```
        XOR AX,AX   ; 或 MOV AX,0等
LOP:      .
          .
          .
        INC AX
        CMP AX,n
        JNE LOP
          .
          .
          .
```

倒计数法的常用指令及格式为：

```
        MOV  AX, n
          .
          .
          .
LOP :     .
          .
          .
          .
          .
        DEC     AX
        JNZ     LOP
          .
          .
          .
```

只要计数次数事先能确定，这种方法直观，方便。

2． 按条件控制循环

有些情况下，计数次数无法事先确定，但循环次数与问题中的某些条件有关，这些条

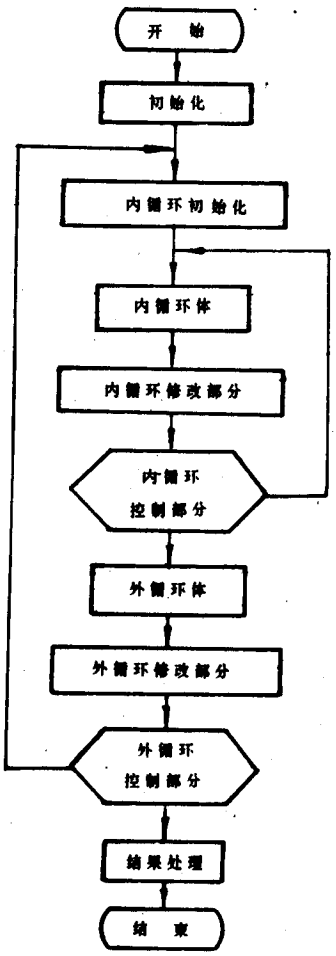


图 4-16 双重循环程序框图

件可以检测到，这时采用条件控制循环为好。

使用条件：循环次数与某些条件有关。条件可以检测。

使用形式：检测，比较，判断。

前面例题中，统计寄存器 AX 中值为 “1” 的位的个数就属这种类型的问题。因为事先并不知道寄存器的情况，若是全零，已不必再循环；若仅最高位为 1，则移位一次即可；若最低位为 1，则要移位 16 次，即循环 16 次。循环结束的条件就看寄存器中是否为全 0，当然对于这个问题来说，也可以采用计数方法。无论什么情况，都强迫循环进行 16 次，不过当 AX 为零时，空循环 16 次，统计结果，“1” 的个数仍为 0 罢了。这里存在一个优劣问题。这样的问题采用计数方法控制就显得太不合适了。

在生产进程中，用条件控制循环的情况是很多的。 某化工生产过程， 只要温度，压力，电流等不超过某一设定值或不超出某一给定范围，就一直重复生产下去，一个月，一年，三年・・・。一旦上述条件不满足，或一开始条件就不满足，则可能要马上停止循环(停止生产)或一次也不生产。再如某些安全检测，事故处理等也属于这种情况。而上述各种条件都可以利用各种传感元件、各种仪器、仪表加以测量，或许再经过模数转换成数字信号，再经过接口电路输送给计算机，那么计算机里的程序就可以通过检测这些信号的状态用以控制循环了。

3．用开关变量控制分支循环

有些情况下，循环程序内部需有分支，这时可用开关变量控制分支和循环。下面用例题加以说明。

使用条件：分支规律已知，计数次数或循环条件已知。

使用形式：条件判断，转移，正、倒计数法。

例：利用一台双头钻床给一批零件钻孔。如图4-17所示。

双头钻床

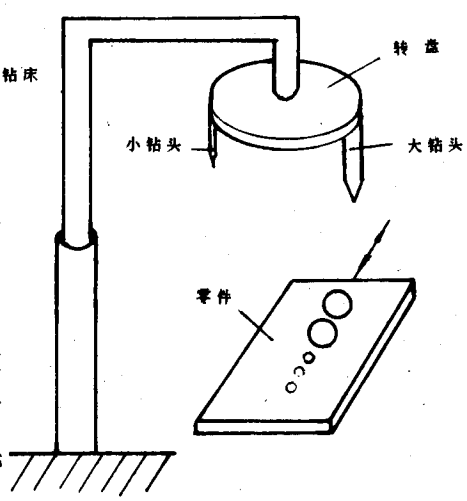


图4-17 钻床加工示意图

假设转盘的起始位置是大钻头在前，钻床下降，可在零件上钻大孔。转盘转180° 角，小钻头即可在前，可钻小孔。如果零件上要求钻3个大孔，2个小孔，那么我们可以设置一开关变量（如 AL寄存器的第0位），用它的“0”状态表示大孔 ，“1”表示小孔。整个钻孔过程是重复性工作，因此可用循环实现，然而循环内又有分支（分为钻大孔或小孔 ）。因为先要钻大孔， 我们可先将开关变量设为“0” ，待大孔钻完后， 再将开关变量设为“ 1” ，以便钻小孔。实现这一问题的程序流程图如图4-18。

从流程图中可以看出，因为将开关变量AL 预置为零， 所以循环必然走钻大孔的分

支，每钻一大孔，BL减 1，零件前进一步，直到 BL 减至零，大孔钻完，将开关变量设为“ 1”，同时转盘旋转 180°，小钻头调至前方。下次循环因AL等于1(≠0)了，所以钻小孔，直至两个小孔钻完结束。

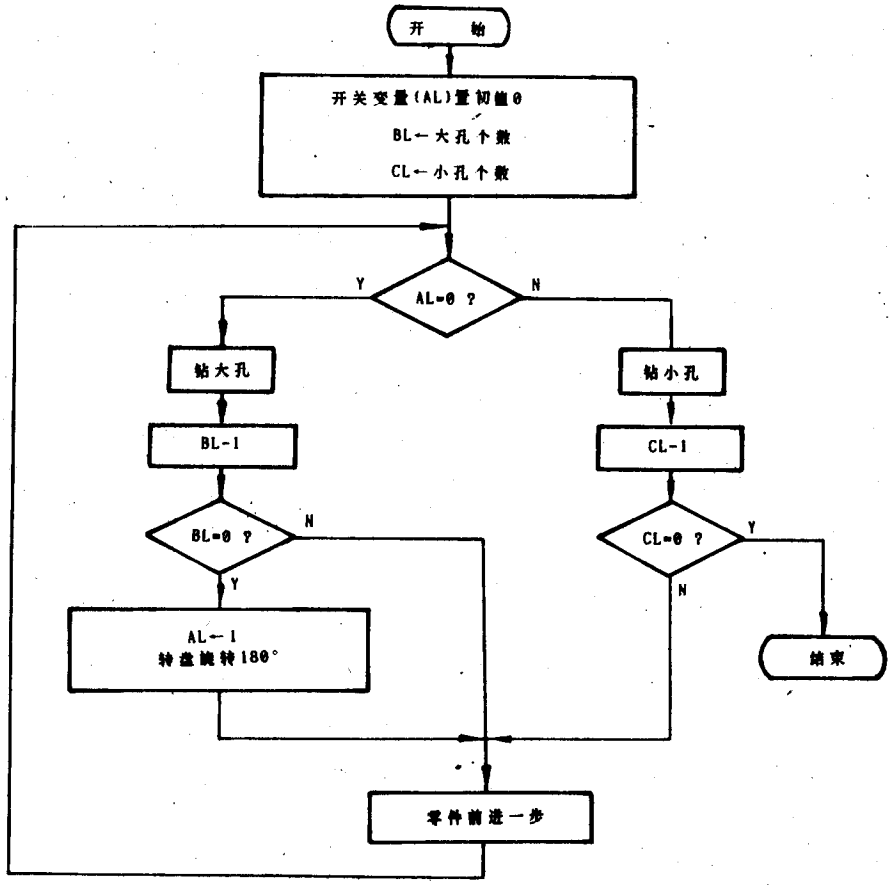


图4-18程序流程图

利用寄存器的 1位作开关变量，它的“0”和“1”状态可以实现两路分支，那么利用寄存器或存贮单元的多位的多个状态就可以实现多路分支。例如 AL 的 2位，可有四个状态，00、01、10、11可实现四路分支。

利用开关变量实现多路分支的示意图如图4-19。

从示意图我们可以看出，就总体而言这是一个循环程序，而循环内又有许多分支。我们用AX作开关变量以实现分支。因为分支规律事先已知，因此可为开关变量设初值，这里用 MOV AX，F1 设为第一路分支，JMP AX 实现分支转移，第一路分支的工作完成后，这里又用 MOV AX，F2 设为第二路分支，同是 JMP AX 则转到第二路分支去执行程序，以次类推。

4．用逻辑尺控制循环

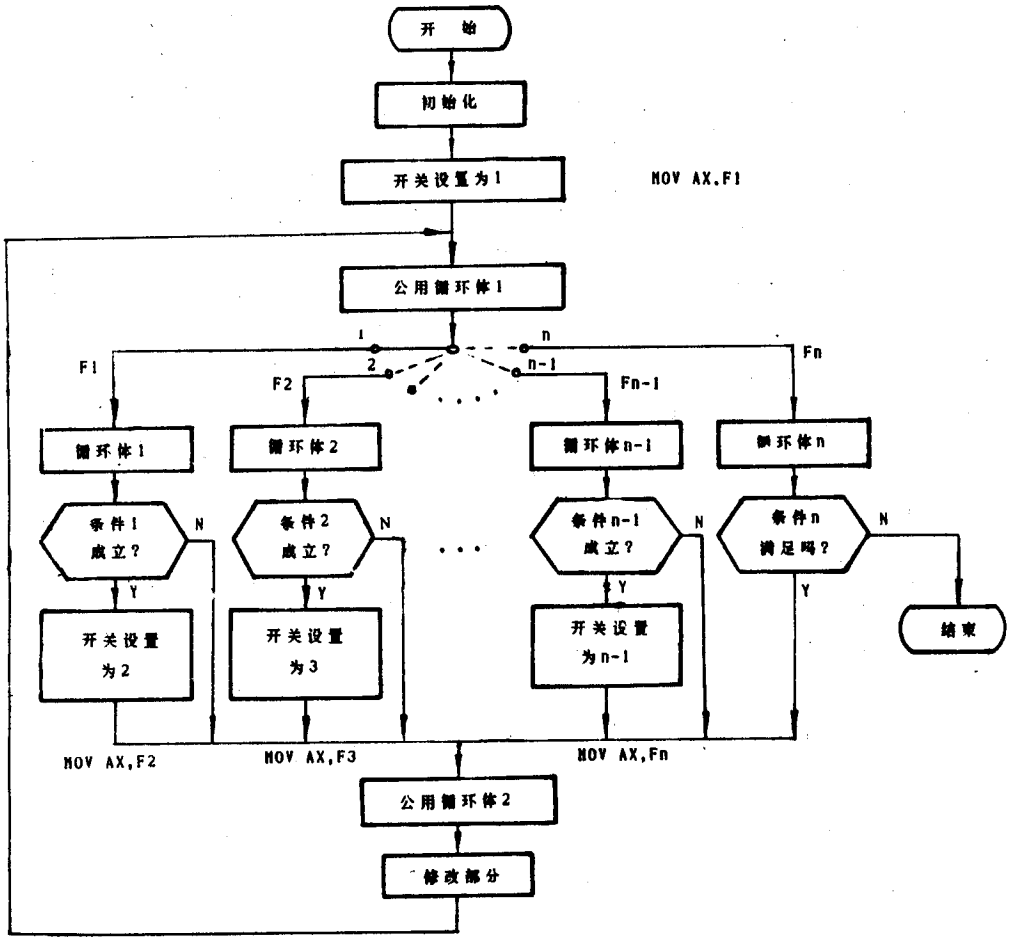


图4-19多路分支框图

如上所述的用开关变量控制循环的方法适合于在编程序以前，就已经知道了分支的规律，而且这种规律是固定的、有限的几种情况。如果分支规律在编程序时无法确定，只有到用户运行程序时才能临时确定，并且分支规律也不固定，那么这种方法就不合适了。例如双头钻床钻孔问题，各种零件的钻孔要求各不统一，先钻大孔还是先钻小孔不一定，钻多少大孔、小孔不一定。前面我们所编程序只适合同一种类型的零件成批加工的要求，换一种零件，那个程序就不行了。如何编写出更通用一些的程序呢 ?方法是有的，我们事先规定好用“0”代表大孔，用“1”代表小孔，用户可以用一串代码表达出目前要加工的零件的加工要求，例如：

01011001

表示先钻一个大孔，接着一个小孔，再一个大孔，两个小孔，再 2个大孔一个小孔。要求用户在加工前写出一串代码描述加工要求，这一点不难做到。

我们在程序中可以首先安排接受(输入)用户加工代码的指令，然后再根据用户的代码实现分支控制，这样的程序就通用了。实现这一要求的流程示意图如图4-20。

用户输入的一串代码，作为分支的依据，称为逻辑尺。

讨论：

(1) 逻辑尺既不是参入运算的常数，又不是指令，而是判断分支的依据，犹如尺子可以描述布的长短一样。

(2) 可以随意地、方便地改变逻辑尺以实现不同的加工要求。

(3) 逻辑尺可长可短，一个字节、一个字不够用，可用几个连续的存贮单元。

(4) 循环结束的控制可以用计数法，可以固定次数，也可以不固定，例如用户输入加工代码时，也同时输入要加工的孔的总个数，用这个数可以控制循环，也可以利用或设置某些特征条件控制循环。

(5) 如果分支不仅两路怎么办？可以选择几位代码的组合实现。如用2位代码可实现四路分支。例如钻孔的大小分为1号、2号、3号、4号，对应四路分支如图4-21所示。

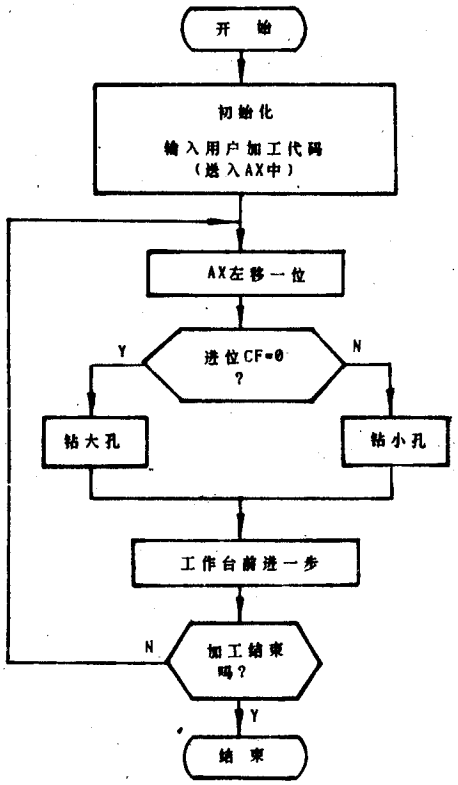


图 4-20 程序框图

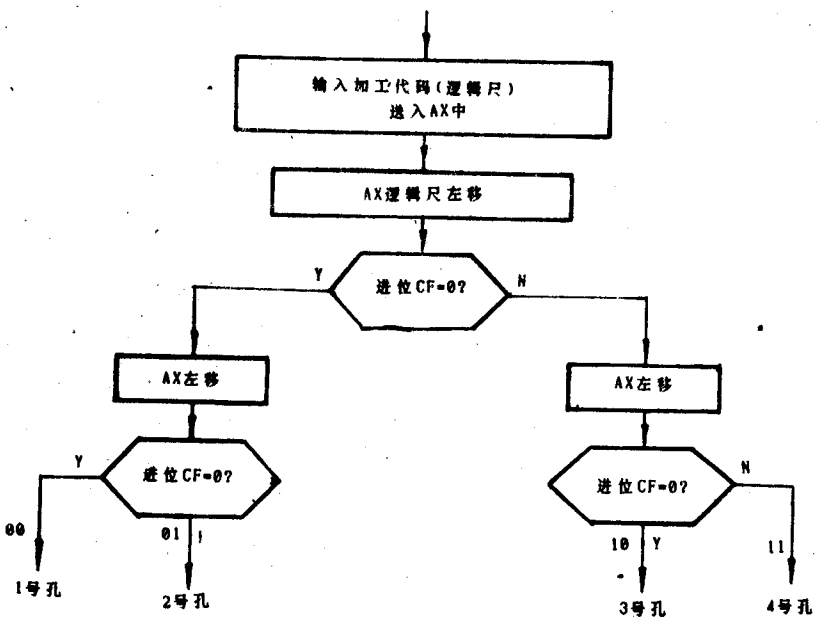


图 4-21 程序框图

这里用逻辑尺的最高2位，通过2次移位，分支四路，对应于代码00、01、10、11。同理，3位代码，3次移位可分支8路。

以上所述各种基本循环控制方法可以互相穿插配合使用，以实现较复杂的循环控制。

第五节　子程序设计

一、概念

如果在一个程序中的多个地方或多个程序中的多个地方用到了同一段程序，那么可以将这段程序抽取出来存放在某一存贮区域，每当需要执行这段程序时，就用调用指令转到这段程序去，执行完毕再返回原来的程序。把抽取出来的这段程序叫做子程序或过程，而调用它的程序称为主程序或调用程序。主程序向子程序的转移叫子程序调用或过程调用，也叫转子。

二、调用和返回指令

1．调用指令 CALL

CALL实现子程序（或过程）的调用，调用结束后，要由所调用的子程序（或过程）中的RET指令返回到CALL指令的下一条指令，为了能保证正确返回，就需要把断点(即CALL指令的下一条指令的地址)入栈保护。

8086/8088 允许要调用的子程序（或过程）在现行码段区域中（此时子程序名中包含NEAR），也可以不在现行代码段区域中(此时子程序名中包含FAR)，为了保证能正确返回，CALL指令的类型必须与RET指令的类型相匹配。

CALL指令有两种得到目标地址的方法：直接和间接寻址。

直接调用(Direct CALL)，目标地址(如NEAR_PRG)直接在CALL指令中。

例如：CALL NEAR_PRG

间接调用(Indirect CALL)，目标地址在由指令指定的寄存器或内存单元中。

例如：　CALL　DWORD　PTR　[BX]。

对于一个段内的直接调用CALL，堆栈指针SP减 2，使IP(指令指针)入栈，从指令中得到的目标过程的相对偏移量(最大为32K字节)加到指令指针上。

一个段内的间接调用指令。首先SP减 2，IP入栈。目标过程的地址偏移量由指令中指定的16位通用寄存器或存贮单元内取出，用它代替IP。

例如：　CALL AX　将使,IP入栈，且AX→IP。

段间直接CALL指令，SP减 2，把现行的码段寄存器CS的内容入栈，CS由指令中包含的段字代替。SP又一次减 2，IP入栈，且IP由在指令中的地址偏移字代替。

例如：　CALL FAR_PRG　；FAR_PRG在过程定义中被说明是FAR过程。

对一个段间间接CALL指令，SP减 2，把现行的CS入栈。CS由指令指定的双字存贮器指针的第二个字的内容所代替。SP再次减 2，IP入栈，然后IP由指令中指定的双字指针的第

一个字的内容所代替。而双字存贮器应在数据段加以定义。

CALL指令对标志位无影响。

2．返回指令RET

RET 通常作为一个子程序或过程的最后一条指令，它用以返回到调用这个子程序的断点处。

一个段内的返回指令，把SP所指的堆栈顶部的一个字的内容弹回到指令指针，且SP加2。

若是一个段间返回指令，除了上述操作以外，把新的SP所指的堆栈顶部的一个字的内容弹回 CS，SP再次加 2。

RET指令对标志位无影响。

三、什么样的程序适合编成子程序

1．程序是多次重复使用的。一个子程序只占一段存贮空间，但可以无数次地调用它，避免了子程序的重复、节省存贮空间。由于增加了调用、返回指令以及现场保护，因此程序执行时间会增长。

2．程序具有通用性、便于共享。例如键盘管理程序，磁盘读写程序，标准函数程序等等，许多程序中要用到，大家可共享的程序适合写成子程序。

四、子程序文件

为了使用的方便，子程序应以文件形成编写。子程序文件由子程序说明和子程序本身构成。

1、子程序说明部分要求语言简明、确切。

子程序说明一般由如下几部分组成：

(1)功能描述：包括子程序的名称、功能、性能指标(如执行时间)等。

(2)所用的寄存器和存贮单元。

(3)子程序的入口、出口参数。

(4)子程序中又调用的其它子程序。

(5)调用实例　(可有可无)。

例：　今有一子程序说明如下：

```
;　子程序　　DTOB
;　将两位十进制数(BCD码)转换成二进制数
;　入口参数：　AL寄存器中存放十进制数
;　出口参数：　CL寄存器中存放转换完的二进制数
;　所用寄存器：　　BX
;　执行时间：　　0.06 ms
```

看了这一子程序说明，尽管我们并不知道子程序本身的情况，但根据说明已可以调用这个子程序了。说明的第一、二行告诉我们DTOB子程序可以完成什么功能。入口参数告诉

我们要调用这一程序必须在调用前将要转换的十进制数送入 AL 寄存器。出口参数告诉我们，子程序执行完毕，转换结果就在CL寄存器中，所用寄存器告诉我们本子程序执行过程中要用到BX寄存器。因此在调用本子程序前，若你的这些寄存器里存放着有用数据的话，应该事先转存或保护起来，否则可能会被破坏，执行时间一项告诉你本程序执行需 0.06 mS，以便给你一个时间的概念。有的说明中给出一个调用实例，以实例的形式教给用户如何使用和调用子程序。

2．子程序本身在8086/8088中常以过程形式存放在代码段中，通常以一个标号开始，以返回指令结束。例如：

```
DTOB    PROC
        .
        .
        .
        RET
DTOB    ENDP
```

五、子程序使用中的问题

1．主程序与子程序的联接

例：　如下程序段：

```
CODE    SEGMENT
STRT:     .
          .
          .
          MOV   AL，XX        ；设要求转换的数在XX单元
          PUSH  BX
          CALL  DTOB
NEXT:     MOV   YY，CL        ；转换结果存放在YY单元
          POP   BX
            .
            .
            .
DTOB      PROC
            .
            .
            .
          RET
DTOB      ENDP
CODE      ENDS
          END   STRT
```

主程序向子程序的转移是由CALL指令完成的。这个主程序中的MOV　AL，XX 是传送入口参数，PUSH　BX保护BX寄存器的内容。CALL指令的执行分两步进行：

(1)先将断点(即CALL指令下面一条指令的地址)NEXT进栈，即保护断点。

(2)程序转 到DTOB去执行，实现了转子。

子程序执行到最后一条指令RET时，将堆栈的内容（断点NEXT)弹出到指令指针中，使程序从NEXT条指令继续执行，实现了从子程序返回。POP BX将恢复BX的内容。

2、寄存器及所用工作单元内容的保护

如果在子程序中要用到某些寄存器或存贮单元，为了不破坏原有信息，要将它们的内容推入堆栈加以保护，存入另外一些空闲的存贮单元或存入某些目前不用的寄存器中也可以。

保护可以在子程序中实现，也可以在主程序中实现。一般情况下，在子程序的开始安排一段保护程序，用以对它所用到的寄存器或存贮单元内容加以保护。在子程序结束前，再将有关的内容恢复。例如：

```
DTOB        PROC
            PUSH        BX
              .
              .
              .
            POP         BX
            RET
DTOB        ENDP
```

在子程序中实现保护比较好，遇到粗心的用户忘记在调用前安排保护的话，有关内容也不会破坏。如果在子程序中没有安排保护指令；则可以在主程序中，在调用前安排保护指令；调用后安排恢复指令。如前一例子所示。

用于为中断服务的子程序，则一定要在子程序中安排保护指令。因为中断是随机出现的，也就是说主程序中转入子程序的地点是不固定的，无法在主程序安排保护程序。

3、参数的传递

子程序中允许改变的数据叫参数。分为入口参数和出口参数。正因为可以接受参数，才使子程序具有灵活、方便、通用之优点。入口参数使子程序可对不同数据进行处理，出口参数使子程序可送出不同的结果。一般的子程序都可接收参数。

参数传递的方法一般有三种：

(1) 用寄存器传递：适合于参数较少的情况。

(2) 用参数表传递：适合于参数较多的情况，但要求事先建立一个参数表，参数表可建在内存中或外设端口中。

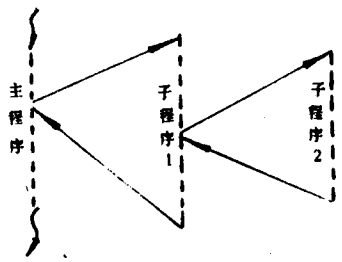
(3) 用堆栈传送：适合于参数多并且子程序有嵌套，递归调用的情况，主程序将参数推入堆栈，子程序将参数从堆栈中弹出。

无论哪种方法，主程序与子程序要配合默契。子程序要求到哪里取参数，主程序就应将参数送入哪儿，而且要注意参数的先后顺序。

六、名词解释

1．子程序嵌套：子程序中调用别的子程序称为嵌套，只要堆栈空间允许，嵌套层次不限。

示意图如右图：



2．子程序递归调用

子程序调用该子程序本身称为递归调用。

当上图中的子程序1与子程序2是同一个程序时，就是递归调用。

3．可重入子程序

子程序可被中断并能再次被中断程序调用者。

4．可重定位子程序

子程序可重定位在内存的全体区域，子程序内不采用绝对地址，全部采用相对地址。

七、子程序应用举例

例 1． 利用堆栈传递参数和多个代码段的使用。

本例题在数据段定义了两个数组，程序段实现数组分别求和。

主程序和过程分别安排在两个不同的段中，过程应是FAR类型的。

主程序向过程的参数传递是利用堆栈实现的，子程序中参数的读取及返回就应配合好。程序执行过程中，堆栈变化示意图4-22。

```
STACK    SEGMENT
SPAE     DW    20 DUP(?)
TOP      EQU   LENGTH  SPAE
STACK    ENDS
DATA     SEGMENT
ARY1     DB    10DUP(?)              ; 定义数组1
SUM1     DW    ?
ARY2     DB    100DUP(?)             ; 定义数组2
SUM2     DW     ?
DATA     ENDS
MAIN     SEGMENT                     ; 主程序段
         ASSUME     CS:  MAIN , DS: DATA , SS: STACK
STR:     PUSH  DS
         MOV   AX, 0
         PUSH  AX
         MOV   AX, DATA
```

```
        MOV  DS , AX
        MOV  AX , SIZE ARY1
        PUSH AX                          ; SUM过程的入口参数1进栈
        MOV  AX , OFFSET ARY1
        PUSH AX                          ; SUM过程的入口参数2进栈
        CALL SUM
             .
             .
             .
        MOV AX , SIZE ARY2
        PUSH AX
        MOV AX , OFFSET ARY2
        PUSH AX
        CALL SUM
        HLT
MAIN    ENDS
PROCE   SEGMENT                          ; 过程段
        ASSUME   CS: PROCE , DS: DATA , SS: STACK
SUM     PROC  FAR
        PUSH  AX                         ; 保护现场
        PUSH  BX
        PUSH  CX
        PUSH  BP
        MOV   BP, SP
        PUSHF
        MOV  CX , [BP+14]
        MOV  BX , [BP+12]
        MOV  AX , 0
ADN:    ADD  AL , [BX]
        INC  BX
        ADC  AH , 0
        LOOP  ADN
        MOV   [BX], AX                   ; 数组之和送到结果区

        POPF                             ; 恢复现场
        POP  BP
```

```
        POP   CX
        POP   BX
        POP   AX
        RET   4                    ; 返回并废除参数1和2。
SUM     ENDP
PROCE   ENDS
        END   STR
```

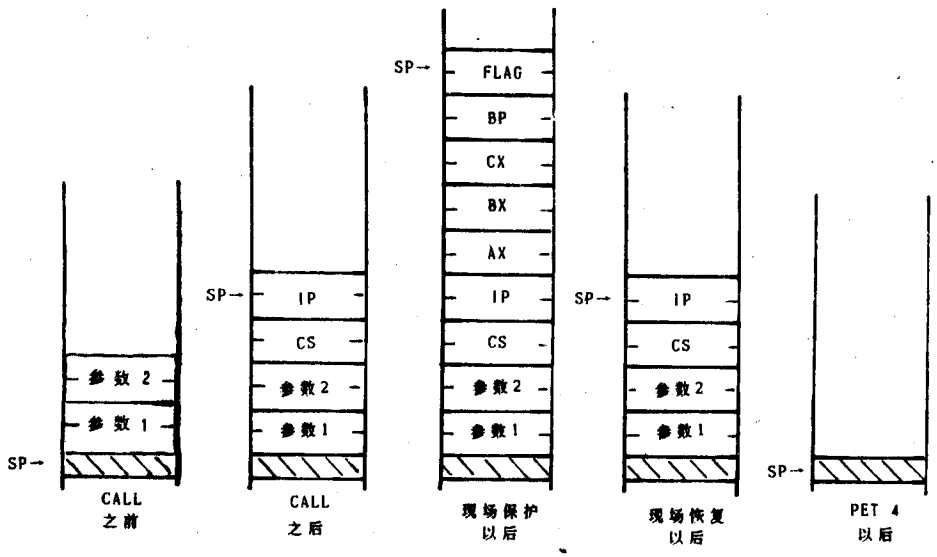


图 4-22 堆栈变化示意图

例题中出现了指令RET 4，它的格式是RET n，这种指令实现返回，并使SP再加n。

例 2: 求 n!

$$n! = \begin{cases} n*(n-1)! & \text{当}n>0\text{时} \\ 1 & \text{当}n=0\text{时} \end{cases}$$

下面所给出的程序即包括子程序嵌套，也包括递归调用。设数n放在AL中，n! 放在BX中。

```
; 主程序
MAIN:   MOV   AX, 3                ; 设n=3
        CALL  FACT
XI:     MOV   BX, DX
        HLT
; 阶乘子程序
; 入口参数  :  AL中存放n
```

```
;出口参数  :  DX中存放n!
;所用寄存器:  CX
FACT    PROC
        CMP   AL,  0
        JNE   IIA
        MOV   DL,1
        RET                    ;①
IIA:    PUSH  AX
        DEC   AL
        CALL  FACT
X2:     POP   CX
        CALL  MULT
X3:     MOV   DX,AX
        RET                    ;②
FACT    ENDP
;无符号字节数乘法子程序
;入口参数:   CL、DL中各为一乘数
;出口参数:   AX中为乘积
MULT    PROC
          .
          .
          .
        RET                    ;③
MULT    ENDP
```

本程序调用过程示意图如图 4-23。

程序调用过程中堆栈变化示意图如图4-24。

从程序的调用过程示意图和堆栈变化示意图中，我们看出，程序在递归调用过程中，用 PUSH 指令将参数 3、2、1 推进堆栈，而在返回过程中，调用乘法子程序，依次实现 1!，2!，3!。当然这不是实现阶乘的唯一方法。

当遇到比较复杂的程序时，借助调用过程示意图和堆栈变化示意图进行分析，是一种十分直观的方法。

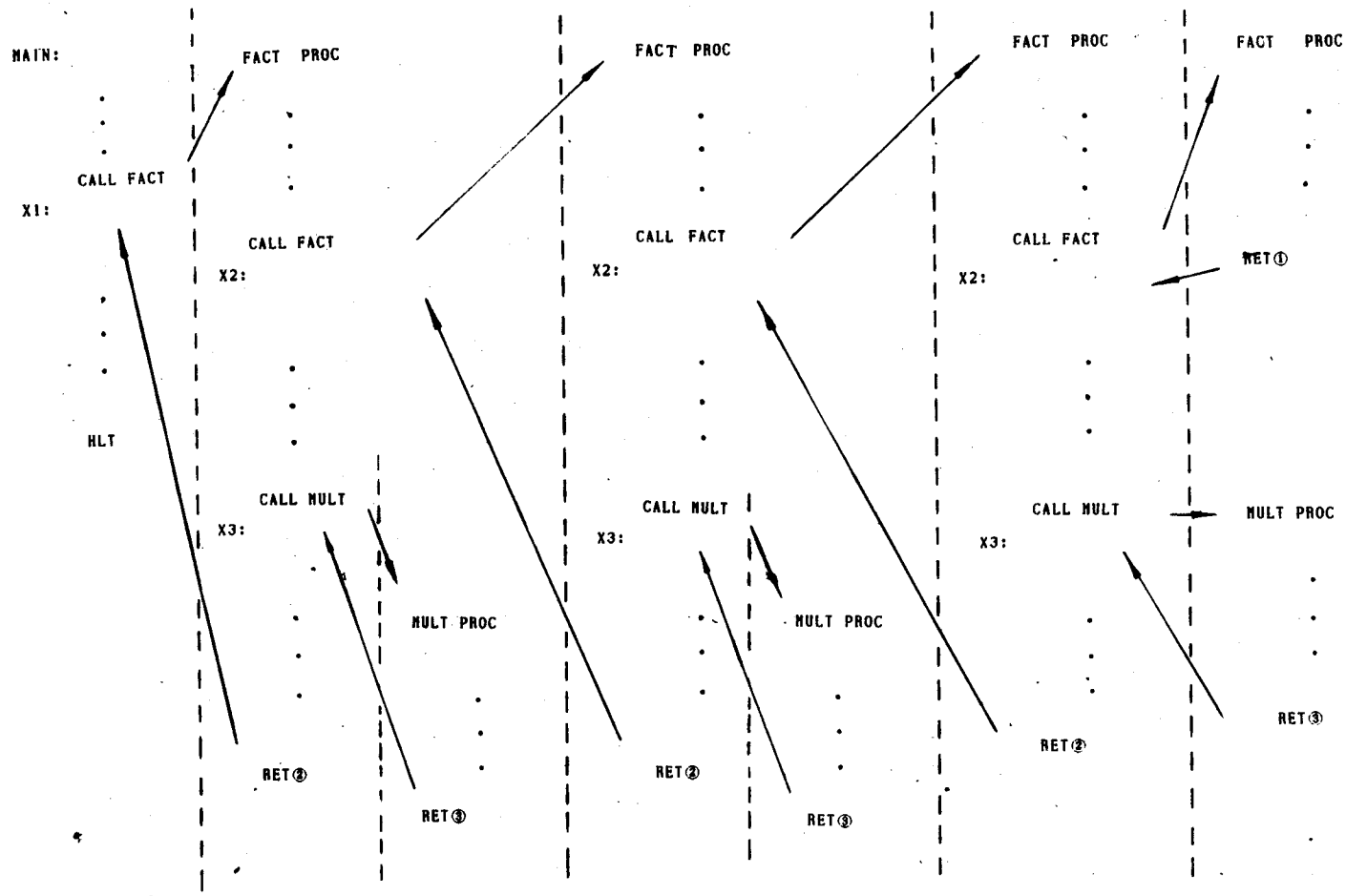


图 4-23 程序调用过程示意图

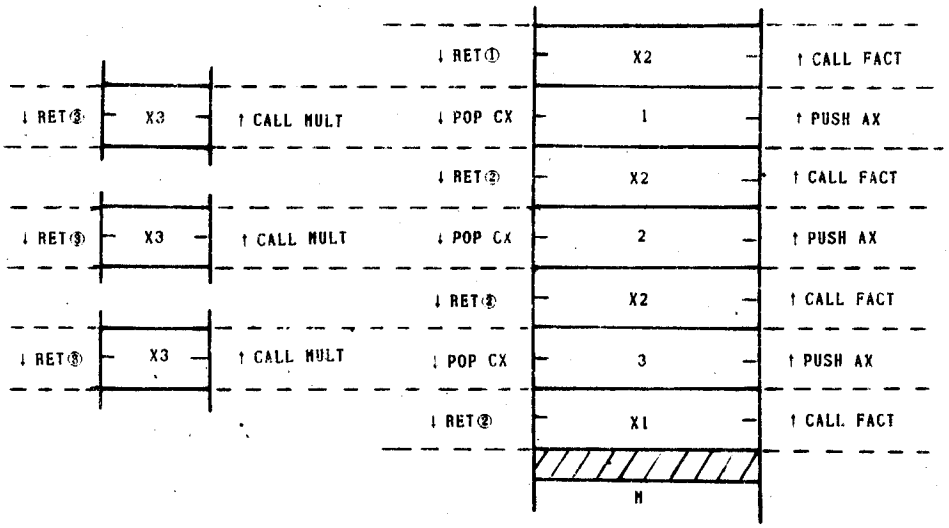


图 4-24 堆栈变化示意图

第六节　DOS 系统功能调用

一、概述

PC-DOS不仅为用户提供了许多命令可以直接使用，而且也提供了八十多个子程序，可供汇编程序员直接调用，八十多个子程序主要分为主要三个方面：

(1) 磁盘的读写、管理

(2) 内存管理

(3) 基本输入输出管理(如键盘、打印机、显示器、磁带等管理)，另外还有时间、日期等子程序。

这几十个子程序给汇编语言程序员提供了极大方便，程序员不必编写这些繁杂程序，也不必搞清有关的设备、电路、接口等，只需直接调用即可。

二、系统功能调用方法

为了使用的方便，已将所有子程序顺序编号。一共87个子程序，编号从0到57H。调用时应包括如下三个方面的内容：

(1) 入口参数。

(2) 子程序编号送入AH。

(3) INT　21H，　；子程序请求中断指令。

有的子程序不要入口参数，但大部分需要将参数送入指定地点。

程序员只须给出这三个方面的信息，不必关心具体程序如何，在内存中的存放地址如何，DOS 根据所给的信息，自动转入相应的子程序去执行。调用结束后有出口参数，一般

在寄存器中。有些子程序，如屏幕显示字符，调用结束会在屏幕上马上看到结果。

例如：
```
MOV    DL，“：”
MOV    AH，2
INT    21H
```

这是 2号功能调用(编号为 2)，实现将字符送入屏幕（或打印机）显示的功能。它要求将要显示的字符的ASCII码值送入DL，调用结果，屏幕上显示DL中的内容“：”。

三、系统功能调用分组说明

系统功能调用可以分组如下：

0～0CH：传统的字符I/O管理。包括键盘、显示器、打印机、异步通讯口的管理。

0C～24H：传统的文件管理。包括复位磁盘、选择磁盘，打开文件，关闭文件，查找目录项，删除文件，顺序读、写文件、建立文件，重新命名文件，查找驱动器分配表信息，随机读、写文件，查看文件长度等。

25～26H：传统的非设备系统调用。包括设置中断向量，建立新程序段。

27～29H：传统的文件管理。包括随机块读写，分析文件名。

2A～2EH：传 统的非设备系统调用，包括读取、设置日期、时间等。

2F～38H：扩充的系统调用组。包括读取 DOS版本号，中止进程，读取中断矢量，读取磁盘空闲空间等。

39～3BH：目录组。包括建立子目录，修改当前目录，删除目录项。

3C～46H：扩充的文件管理组。包括建立、打开、关闭文件，从文件或设备读写数据。在指定的目录里删除文件，移动文件，读、写文件，读取、修改文件属性，设备I/O控制，复制文件标记等。

47H：目录组。取当前目录。

48～4BH：扩充的内存管理组。包括分配内存，释放已分配的内存，分配内存块，装入或执行程序等。

4C～4FH：扩充的系统调用组。包括终止进程，查询子进程的返回代码，查找第一个相匹配的文件，查找下一个相匹配的文件。

50～53H：扩充的系统调用，DOS内部使用。

54～57H：扩充的系统调用。包括读取校验状态，重新命名文件，设置读取日期和时间。

从 39H以后的文件管理系统调用都是为了处理树形目录结构而提供的。下面我们选择其中一部分常用的系统功能调用分类加以介绍，并应用举例。其余部分可参考IBM PC DOS资料。

四、基本I/O功能调用

1．键盘输入（1号调用）

1号系统功能调用等待从标准输入设备输入一个字符并送入寄存器 AL，不需入口参

数。例如：

```
MOV    AH，1
INT    21H
```

执行上述指令，系统将扫描键盘，等待有键按下，一旦有键按下，就将键值（相应字符的ASCII码值)读入，先检查是否是Ctrl-Break，若是，则退出命令执行；否则将键值送入AL寄存器，同时将这个字符显示在屏幕上。

2．控制台输入但无显示（8号调用）

8号调用与1号调用类同，只是不在屏幕上显示 输入的字符。

3．打印输出（5号调用）

把DL中的字符输出到打印机上。例如：

```
MOV    DL，'A'
MOV    AH，5
INT    21H
```

4．直接控制台输入/输出（6号和7号调用）

6 号调用可以从标准输入设备输入字符， 也可以向屏幕上输出字符。 并且不检查Ctrl-Break。

若 DL=FFH时， 表示从键盘输入，

若标志ZF=0，表示AL中为键入的字符值。

若标志ZF=1，表示AL中不是键入的字符值，即尚无键按下。

若DL≠FFH时，表示向屏幕输出，DL中为输出字符的ASCII码值。

例如：

```
MOV  DL，OFFH
MOV  AH，6
INT  21H
```

即为从键盘输入字符。

```
MOV  DL，24H
MOV  AH，6
INT  21H
```

即将24H对应的字符“$”输出。

5．直接控制台输入但不显示（7号调用）

等待从标准输入设备输入字符，然后将其送入AL，如同6号调用，对字符不做检查。

6．输出字符串（9号调用）

调用时，要求 DS：DX必须指向内存中一个以“$”作为结束标志的字符串。字符串中每一个字符（不包括结尾标志$)都输出显示或打印。

例：

```
DATA    SEGMENT
BUF     DB       'HOW  DO  YOU  DO ?$'
```

```
        .
        .
        .
DATA    ENDS
CODE    SEGMENT
        .
        .
        .
        MOV  AX , DATA
        MOV  DS , AX
        .
        .
        .
        MOV  DX , OFFSET  BUF
        MOV  AH , 9
        INT  21H
        .
        .
        .
CODE    ENDS
```

执行本程序，屏幕将显示： HOW DO YOU DO ?

7．字符串输入(0AH号调用)

从键盘接收字符串到内存输入缓冲区。要求事先定义一个输入缓冲区，缓冲区内第一个字节指出缓冲区能容纳的字符个数，不能为零。第二个字节保留以用作填写输入的字符个数。 从第三个节开始存放从键盘上接收的字符。 若实际输入的字符数少于定义的字节数，缓冲区内其余字节填零，若多于定义的字节数，则后来输入的字符丢掉，且响铃。

调用时，要求DS：DX指向输入缓冲区

例如：

```
DATA    SEGMENT
BUF     DB    50            ; 缓冲区长度
        DB    ?             ; 保留为填入实际输入的字符个数
        DB    50 DUP(?)     ; 定义50个字节存贮空间
        .
        .
        .
DATA    ENDS
CODE    SEGMENT
        .
        .
        .
        MOV  AX , DATA
```

```
        MOV  DS,  AX
          .
          .
          .
        MOV  DX, OFFSET  BUF
        MOV  AH, 10
        INT  21H
          .
          .
          .
CODE    ENDS
```

8．异步通信口输入(03H)

从标准异步通信接口等待输入一个字符，然后送到寄存器AL中。启动时 DOS把一个异步通信端口初始化为2400波特，没有奇偶校验位，一个停止位，字长为8位。

9．异步通信口输出(04H)

在DL中的数据被输出到异步通信接口去。

关于异步通信口的输入/输出，推荐使用 ROM BIOS 中断调用 14H（详见第五章 第三节)。

10．日期设置(2BH调用)

调用时，CX：DX中必须有一个有效的日期，CX中存放年号(1980-2099)，DH中存放月号 (1-12)， DL中放日号。若日期有效，设置成功，AL = 0；否则 AL = 0FFH。

例如： 下列程序可把日期设置为1985年9月10日。

```
MOV  CX, 1985
MOV  DH, 9
MOV  DL, 10
MOV  AH, 2BH
INT  21H
```

11．取得日期 (2AH)

调用后返回日期在 CX：DX 中。CX中放年号，为二进制数，DH 中放月号，DL 中放日号。如果日时钟转到下一天，日期将自动调整，也考虑每月的天数和闰年。

不需要入口参数。

12．设置时间 (2DH)

时间的格式是4个8位二进制数，具体地说：CH表示小时(0～23)，CL表示分(0～59)，DH表示秒(0～59)，DL表示百分之一秒(0～99)，这个格式很容易转化为打印 /显示形式，也可以用来计算，比如从一个时间值中减去另一个时间值。

调用时，要求CX：DX中存放要求的时间。若此时间是有效的，设置成功，AL返回00，若时间的组成部分无效，设置操作取消，AL返回0FFH。

13．取得时间 （2CH）

时间的格式如同2DH功能调用。不需要入口参数。调用结束时，CX：DX中为返回的时间。

例如：
```
MOV   AH，2DH
MOV   CX，5FH
MOV   DX，900H
INT   21H
```
将把系统时间设置为5点15分9秒。
```
MOV   AH，  2CH
INT   21H
```
将在CX：DX中得到时间的二进制值。

例题：利用DOS系统功能调用实现人机对话。

下述程序可以在屏幕上显示一行提示信息，然后接收用户从键盘输入的信息并将其存入内存缓冲区。
```
DATA    SEGMENT
PARS    DB    100                          ; 定义输入缓冲区
        DB    ?                            ; 准备接收键盘输入信息
        DB    100  DUP(?)
 MESG   DB   'WHAT  IS   YOUR  NAME ?'     ; 要显示的提示信息
        DB    '$'                          ; 提示信息结束标志
DATA    ENDS
STACK   SEGMENT   PARA STACK ' STACK '
        DB    100  DUP(?)
STACK   ENDS
CODE    SEGMENT
        ASSUME   CS: CODE，  DS: DATA，SS: STACK
SART    PROC     FAR
        PUSH     DS
        MOV      AX，0
        PUSH     AX
        MOV      AX，DATA
        MOV      DS，AX
DISP:   MOV      DX，  OFFSET  MESG
        MOV      AH，  9              ; 利用9号功能调用
        INT      21H                  ; 显示提示信息
```

```
KEYBD:    MOV     DX , OFFSET  PARS
          MOV     AH , 10              ; 利用10号调用
          INT     21H                  ; 接收键盘输入
          RET
SART      ENDP
CODE      ENDS
          ENS     SART
```

关于基本I/O功能调用，以后的章节中还将进一步引用。

关于磁盘文件管理方面的系统功能调用详见下节。

第七节　磁盘文件管理

关于磁盘上信息存放的格式、文件管理的方式等内容，请参见第八章第一节。

磁盘操作系统为磁盘文件的存取提供必要的支持，它负责把物理磁盘空间分配给各文件，以目录形式管理磁盘上的所有文件，并把用户对文件中数据的访问请求转换成对特定磁盘单元的访问等等。

磁盘文件中数据的最小单位通常叫做记录，一个记录可以包括一个或更多个字节。一个记录内字节数的多少称为记录长度，而且对任一给定的文件而言，记录长度是一个常数，它由 FCB中的两个字指定(见下述)。文件的传输总是一次一个记录，记录以块的形式存在磁盘上，每个块最多包括128个记录。一个块内的记录编号为0～127。文件的最后一个块可能会少于128个记录。块的编号从0开始，而且对文件里的每个接续的块以1递增。为了定位文件中的一个指定记录，DOS必须知道块号和块内记录号，FCB中包括了这两项内容，图 4-25 说明了DOS 中文件的传统结构形式。

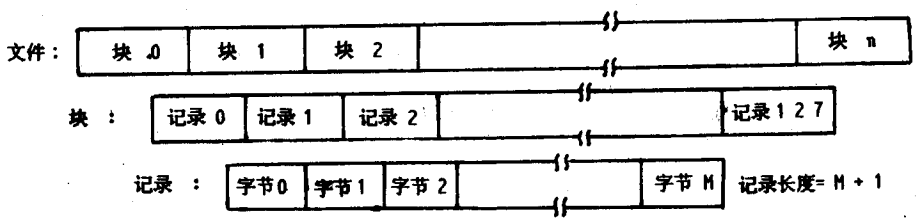


图4－25　传统文件结构形式

我们要处理一个文件，可以借助DOS系统功能调用实现。而且传统的文件管理功能调用要用到内存中的两个数据区。一个称作文件控制块(FCB)，一个称做磁盘传输区(DTA)。

FCB 中包括由 DOS 管理文件时用的信息，通常被指定在程序段前缀的偏移地址05CH处。处理文件的过程总是应先把文件名和文件扩展名放到FCB中相应的字段内。

DTA 是实现数据缓冲的存贮区，一切磁盘输入/输出操作都和它打交道。读磁盘文件时，读出的数据将放在DTA中，写磁盘文件时，待写的数据也指定是在DTA中。当DOS 把控

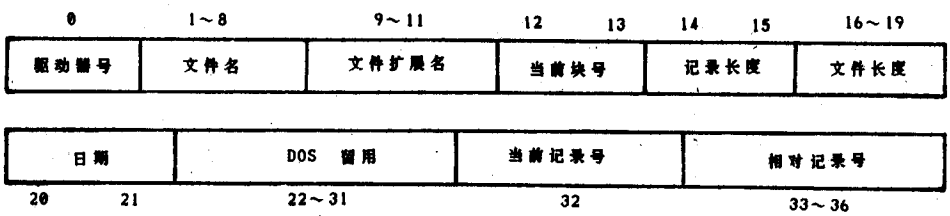
制权转给我们的程序时，DTA被设置在程序段前缀的偏移地址80H处开始。我们可以直接利用它，也可以用1AH号系统功能调用在自己的数据段内重新定义DTA。DTA 内应有足够的字节数，以满足读写文件时的数据量要求。

一、文件控制块(FCB)和文件标记

1．文件控制块分为标准文件控制块与扩充的文件控制块

(1)标准文件控制块

标准文件控制块占用37个字节，其格式规定如图4-26。



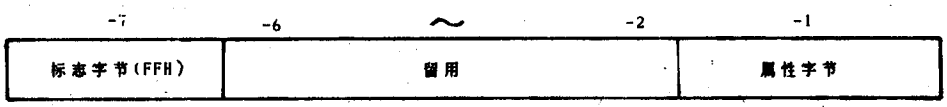
其中：字节序号是十进制数

图 4-26 FCB 的格式

字节号	功能
0	驱动器号，0为约定的驱动器，1为驱动器A，2为驱动器B
1-8	文件名，左对齐，后边补空格
9-11	文件扩展名，左对齐，后边补空格(也可全为空格)
12-13	相对于文件开始的当前块号，开始为零（由打开文件 OPEN ）功能调用设置为零。
14-15	以字节数表示的记录长度。由OPEN功能调用设置为80H，如果不采用这个值，可以自己设置。
16-19	用字节数表示的文件长度，占用两个字，第一个字是文件长度的低位部分。
20-21	文件建立或最后修改的日期。月、日、年中（mm/dd/yy），mm 为 1-12，dd 为 1-31，yy 为 0-199(1980-2099)。
22-31	留给DOS使用
32	当前块内的当前记录号(0-127)。OPEN系统功能调用不初始化这个字节，应在对软盘顺序读/写操作前，由用户设置这个字节。
33-36	相对于文件头的相对记录号，从零开始。OPEN功能调用也不初始化这一字节应在对软盘进行随机读/写操作前，由用户设置这一字节。

(2)扩充的文件控制块

扩充的文件控制块是用来建立或查找磁盘目录中具有特殊属性的文件。它是在FCB前增加7个字节的前缀而产生的。前缀的格式如下：



其中字节序号（-1～-7 ）是相对于标准 FCB 中字节 0 而言。字节-7中存放0FFH，用来标志是一个扩充的 FCB。字节-1用以存放文件属性。允许用户为自己的文件定义不同属性，如只读文件、隐藏文件等。

说明：

一个未打开的FCB由如下几部分组成：FCB前缀(扩充的FCB时)、驱动器号、文件名与扩展名，在一个打开的FCB中，剩余的部分是由建立(CREAT)或打开(OPEN)文件的系统功能调用填写的。

．字节0-15和32-36应该由用户程序设置，其中0-11也可以由DOS命令行解释程序设置(详见下述)。字节16-31是由DOS设置而且用户程序中不要去改动它。

．FCB中所有存放字的区域，都是字的低位字节在前。例如记录长度若为80H，则在字节14处存80H，字节15处存00H。

．在DOS系统功能调用中对FCB的引用，可以是标准的也可以是扩充的 FCB，如果使用扩充的FCB，则指向FCB的寄存器就应设置为FCB前缀的字节-7，而不是字节0。

当通过键入命令或文件名来调用某个程序时，DOS 的命令行解释程序会把命令的第一个文件名装入程序段前缀内偏移地址05CH处，如果有两个文件名，DOS会把第二个放到06CH处。

例如，键盘输入如下命令：

A>DELIN EXAM．ASM

DOS命令行解释程序读入文件名 EXAM.ASM，并把它放在程序段前缀偏移地址05CH处。

用户可以设置自己的FCB存放区。

2．文件标记

文件标记是为了适合于树形目录结构的特点而增加的新概念。DOS 2，0版本以后的扩充的文件管理系统功能调用中，可采用文件标记实现文件系统的管理。这里需要一个ASCII码字符串，这个字符串包含驱动器名、路径名和文件名，后跟一个字节的00H。

如：B: \PT1\FILE1

建立或打开文件时，要求入口指针指向这样一个 ASCII码字符串。文件一旦建立，就获得了一个文件标记，以后在打开、读、写、关闭文件时就使用这个文件标记。

在树形目录结构的文件管理中，文件的记录长度皆为一个字节。既然皆为一个字节，所以记录的概念实际上已不复存在，代之而起的是文件中的字节号和字节数。

文件的读/写系统功能调用不再分为顺序读 /写和随机读 /写，只有一组读/写功能调用（3FH和40H)。这一组读 /写功能调用都是从当前读 /写指针所指的字节开始读/写指定的字节数。在建立文件和打开文件时，读 /写指针指向文件开头的一个字节。要想实现随

机读/写，可以借助于42H号功能调用事先把读/写指针够到所要求的字节位置。

二、磁盘文件存取方式

磁盘文件的存取方式分为两种，一种是顺序存取方式，假定我们想按从头到尾的次序存取文件中的记录时，就可采取这种方式。另一种是随机存取方式。这种方式允许我们按所希望的任意次序访问任一记录。

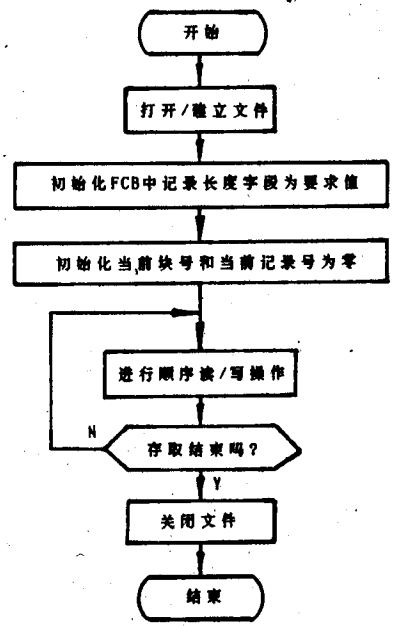


图 4-27 顺序存取文件过程

1．按顺序的存贮方式存取文件的过程如图4-27所示。

为了保证从文件的开头开始进行读/写，先要把FCB当前块号和当前记录号两个字段初始化为零。顺序读/写操作可以利用DOS的 14H 或 15H 写功能调用实现（详见下述），DOS 读/写当前块中当前记录以后，自动修改当前块号和当前记录号，以指向文件的下一个记录。

2．随机存取方式存取文件的过程又分为随机存取和随机块存取两种。随机存取过程如图4-28所示。

随机存取方式中采用指定一个相对记录号的办法实现随机文件存取。值得注意的是相对记录号不同于前面提到的当前块号中的当前记录号。相对记录号是相对于文件的开头，第一条记录的相对记录号为 0，对每一个相继的记录，相对记录号递增 1，直到文件结束。这种方式允许我们指定任一相对记录号实现对任一记录的存取，同时也要求我们在每次读写操作前，设置相对记录号字段的值。但在实际进行磁盘存取之前，由 DOS 把相对记录号转换成对应的当前块号和当前记录号。

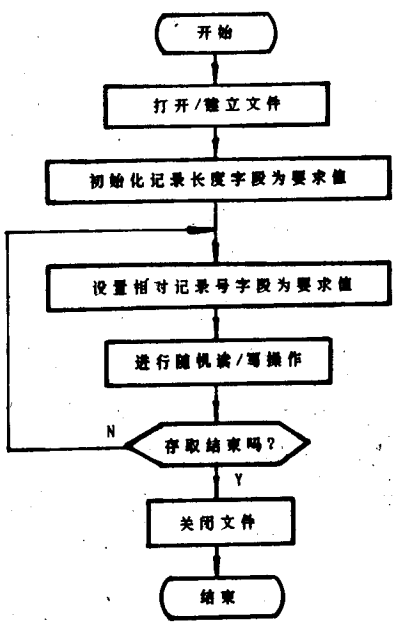


图 4-28 随机存取文件过程

随机读写操作由21H和22H号系统功能调用实现。

3．按随机块方式存取文件的过程如图4-29所示。

随机块方式同随机方式类似，不同的是它允许我们一次传输一条以上的记录。这种情

况下，我们应指定一个相对记录号和要存取的记录数，这个相对记录号用来定位要传输的第一条记录，其它记录的传输是从这个定位开始向文件尾方向顺序地进行，直到CX中的记录数传输完为止。

随机块读/写操作由 27H 和 28H 号系统功能调用实现。

4．利用文件标记存取文件的过程如图4-30所示。

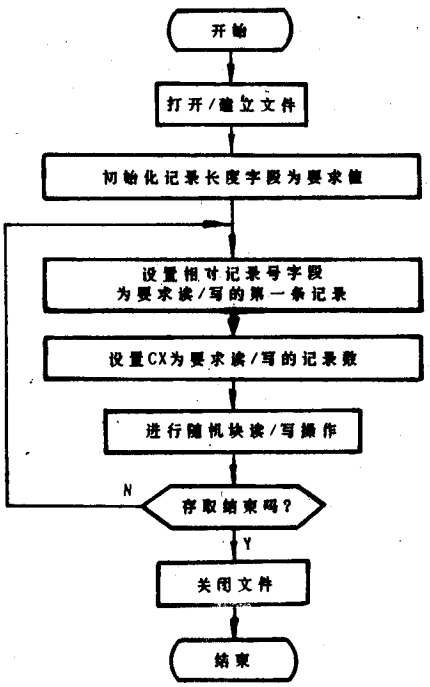


图 4-29 随机块存取文件过程

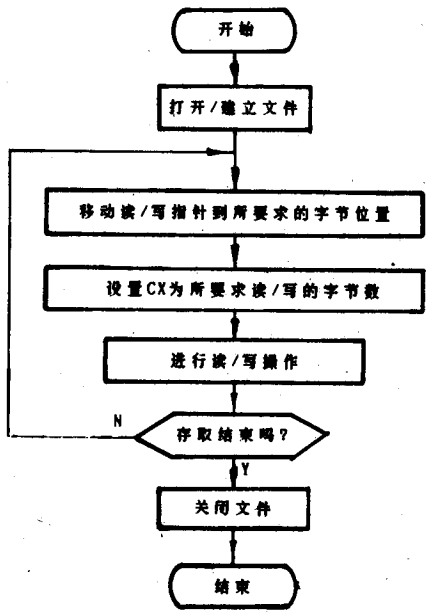


图 4-30 利用文件标记存取文件过程

利用文件标记存取文件的过程有些类似于随机块存取过程。可以移动读/写指针到所要求的位置，以实现随机存取，而在读/写CX中指定的字节数时是顺序存取。这种情况下，主要的区别是：建立和打开文件要用 3CH和 3DH系统功能调用，读和写操作要用3FH和40H系统功能调用实现。

三、磁盘文件管理系统功能调用

磁盘操作系统为汇编语言程序员提供了一系列磁盘文件管理的功能调用。包括传统的文件管理和扩充的文件管理两部分。功能号 39H以前的属于传统的文件管理组，其余的属于扩充的文件管理组。下面首先介绍常用的传统文件管理系统功能调用。

1．查找文件 (11H) SEARCH FIRST

入口参数：DS：DX指向一个未打开的FCB ，AH=11H。

功　　能：查找当前磁盘目录，寻找第一个符合的文件名。

返回参数：若没有一个文件名相符合，AL=0FFH，否则AL=0。

2．打开文件 (0FH) OPEN

入口参数：DS：DX指向一个未打开的FCB，AH=0FH。

功　　能：打开指定的文件。

返回参数：若打开成功，AL=0；若文件找不到，AL=0FFH。

3．建立文件（16H）CREAT

入口参数：DS：DX指向一个未打开的 FCB，AH=16H。

功　　能：除了文件找不到时，在磁盘上建立此文件外，其余同OPEN。

返回参数：若文件打开或建立成功，AL=0，若文件不存在或建立不成功，AL=0FFH。

4．设置磁盘传输地址（1AH）SET DTA

入口参数：DS：DX 指向 DTA 的首址，AH=1AH。

功　　能：设置DTA为DS：DX中地址。

6．关闭文件（10H）CLOSE

入口参数：DS：DX指向一个打开的 FCB，AH=10H。

功　　能：关闭指定文件。

返回参数：若关闭不成功，AL=0FFH，关闭成功 AL=0。

7．顺序读文件（14H）SEQUENTIAL READ

入口参数：DS：DX指向一个打开的FCB，AH=14H。

功　　能：将由当前块号和当前记录号指定的记录，从磁盘上传送到 DTA 中，然后修改当前记录号与当前块号使之指向下一条记录。

返回参数：若传输成功，AL=00；若文件结束，AL=01或03，01表示部分记录可用，不足部分填零；若DTA无足够空间容纳一条记录，AL=02。

8．顺序写文件（15H）SEQUENTIAL WRITE

入口参数：DS：DX指向一个打开的FCB，AH=15H。

功　　能：功能类似14H，仅传送方向相反。

返回参数：若写成功，AL=0；若磁盘已满，AL=01；若DTA中空间不足，AL=02。

9．随机读文件（21H）RANDOM READ

入口参数：DS：DX指向一个打开的 FCB，AH=21H。

功　　能：首先将相对记录号换算成当前块号和当前记录号，再将此记录读到DTA中。

返回参数：同顺序读文件（14H）。

10．随机写文件（22H）RANDOM WRITE

入口参数：DS：DX指向一个打开的 FCB，AH=22H。

功　　能：功能类似于随机读(21H)，仅传送方向相反。

返回参数：同顺序写(15H)。

11．随机块读文件（27H）RANDOM BLOCK READ

入口参数：DS：DX指向一个打开的FCB，CX=要读的记录个数，AH=27H。

功　　能：除了读取CX中指定的记录数外，类同随机读(21H)。

返回参数：若读成功，AL=0；若所有记录读完前遇到文件结束，AL=01或03，01表明文件结束且最后一个记录完成，03表明最后记录是部分记录；若DTA中地址在0FFFFH之上出现了折回，AL=02。任何情况下，CX中为实际读入的记录数。相对记录号、当前块号与当前记录号指向下一条记录。

12．随机块写文件（28H）RANDOM BLOCK WRITE

入口参数：AH=28，其余同随机块读(27H)。

功　　能：除了传送方向相反和写保护检查外，其余同随机块读(27H)。

出口参数：类同随机写(22H)。

从39H开始以后的系统功能调用，都是为处理树形目录结构而提供的。它与传统的文件管理系统的差别前面已作过介绍，这里再强调两点：

(1)建立或打开文件时，入口指针不是指向FCB，而是指向一个含 有驱动器名、路径和文件名的ASCII码字符串。文件一旦建立或打开之后，就得到了一个16位的文件标记。

(2)读、写、关闭文件时，不再使入口指针指向FCB，而是利用文件标记。

下面介绍常用的扩充文件管理系统功能调用。

1．建立文件（3CH）CREATE FILE　　（与16H对应）

入口参数：DS：DX指向含有驱动器名、路径和文件名的ASCII码字符串。CX 中为文件属性，CX=01时表示只读文件，=02表示隐藏文件，=04为系统文件，=10H 表示入口是子目录等。

返回参数：如果CF=CY，AX中为错误类型码(3、4或5，见下面的注释)；否则，AX中为文件标记。

2．打开文件（3DH）OPEN FILE　　（与0FH对应）

入口参数：DS：DX内容同3CH，AL=存取码，AL=0表示打开文件供读，AL=01表示打开文件供写，AL=2表示供读写。

返回参数：如果CF=CY，AX=错误码(3、4、5、或12)；否则，AX=文件标记。

3．关闭文件（/标记）（3EH）CLOSE HANDLE

（与10H对应）

入口参数：BX=由3CH或3DH返回的文件标记。

返回参数：若CF=CY，AX中为错误码(6)。

4．读文件（3FH）READ HANDLE

入口参数：BX=文件标记，CX=要读的字节数，DS：DX指向接收数据的缓冲区。

返回参数：如果CF=CY，AX=错误码(5或6)；否则，CX中为实际读入的字节数。

5．写文件（40H）WRITE HANDLE

入口参数：类同3FH功能调用。

返回参数：类同3FH功能调用，若AX≠CX，表示出错。

6．移动读写指针（42H）LSEEK

入口参数：AL=移动方法代码，当AL=0时，表示从文件开始移到偏移值处，AL=1表示按偏移值增加，AL=2表示移到文件尾加偏移值处。CX：DX=偏移值，BX为文件标记。

返回参数：如果CF=CY，AX中为错误类型码(1或6)，否则，CX:AX为读写指针的新值。

注释：当从功能调用返回时，若进位标志(CF)置位(CF=CY=1)，表示发生错误，错误码的意义如下：

错误码	意义
1 H	无效的功能号
2 H	文件未找到
3 H	路径未找到
4 H	打开文件太多（无文件标记可用）
5 H	拒绝存取
6 H	无效文件标记
12 H	无更多文件

四、文件管理应用举例

如何应用前面介绍的知识进行文件管理呢？下面的第一个例题目的在于说明怎样利用DOS系统功能调用和FCB建立、打开、读、写、关闭文件等。第二个例题说明怎样利用文件标记处理文件。

例 1．下述程序的名字叫 AAA．ASM，这个程序在磁盘上建立一个名叫EXAMPLE．FIL的文件，并且可以随机读这个文件。当编辑、汇编、连接完这个程序后，再用 PC DOS 中的 EXE2BIN 程序将 .EXE文件转换成．COM 文件，转换命令是：

A>EXE2BIN AAA．EXE AAA．COM

可键入如下命令执行这个程序：A>AAA EXAMPLE．FIL

DOS 命令行解释程序会读入文件名 EXAMPLE．FIL，并把它放在程序段前缀偏移地址05CH的FCB中。

具体程序如下：

```
CODE        SEGMENT
            ORG     05CH
 FCB        DB 37 DUP(0)                ; 定义FCB
            ORG     90H
DTA         DB   100DUP(0)              ; 定义DTA
            ORG  100H
            ASSUME CS: CODE，DS: CODE，ES: CODE
BGN :       JMP START
FILE_ERR    DB  0AH, 0DH, 'File already exists'，'$'
CREAT_ERR DB  0AH, 0DH, 'ERROR in creating file'，'$'
```

```
WRITE_ERR DB  0AH, 0DH, 'ERROR in writing file', '$'
INP_ERR   DB 0AH, 0DH, 'ERROR in input', '$'
IN_MSG    DB 0AH, 0DH, 'Please input relative record number', '$'
READ_ERR  DB 0AH, 0DH, 'ERROR in reading file', '$'
CLOSE_ERR DB 0AH, 0DH, 'ERROR in closing file', '$'
START:    MOV AH, 1AH
          MOV DX, OFFSET DTA
          INT 21H                          ; 设置DTA
          MOV AH, 11H
          MOV DX, OFFSET FCB
          INT 21H                          ; 查找文件
          OR AL, AL
          JNZ CREAT
          MOV DX, OFFSET FILE_ERR
ERR_EXIT: MOV AH, 9
          INT 21H
          INT 20H                          ; 返回系统
CREAT:    MOV AH, 16H
          MOV DX, OFFSET FCB
          INT 21H                          ; 建立文件(打开文件)
          OR AL, AL
          JZ CREAT_OK
          MOV DX, OFFSET CREAT_ERR
          JMP ERR_EXIT
CREAT_OK: MOV FCB+32, 0                    ; 设置当前记录号
          MOV WORD PTR FCB+12, 0           ; 设置当前块号
          MOV WORD PTR FCB+14,110          ; 设置记录长度
          MOV AL, '0'
WRITE:    MOV DI, OFFSET DTA
          MOV CX,110
          REP STOSB                        ;110个相同字符写入DTA
          PUSH AX
          MOV DX, OFFSET FCB
          MOV AH, 15H                      ; 顺序写入磁盘文件
          INT 21H
```

```
        OR AL , AL
        POP AX
        JZ WRITE_OK
        MOV DX , OFFSET WRITE_ERR
        JMP ERR_EXIT
WRITE_OK: INC AL
        CMP AL , '9'+1                  ; 共写十次，字符'0'～'9'
        JNE WRITE
KB_IN:  MOV DX , OFFSET IN_MSG
        MOV AH , 9
        INT 21H
        MOV AH , 1                      ; 接受键入相对记录号
        INT 21H
        CMP AL , '$'
        JZ EXIT
        CMP AL , '0'
        JL IN_ERR
        CMP AL , '9'
        JA IN_ERR
        JMP IN_OK
IN_ERR: MOV DX , OFFSET INP_ERR
        MOV AH , 9
        INT 21H
        JMP KB_IN
IN_OK:  SUB AH , AH
        SUB AL , '0'
        MOV WORD PTR FCB+33 , AX
        MOV DX , OFFSET FCB
        MOV AH , 21H
        INT 21H                         ; 随机读文件
        OR AL , AL
        JE READ_OK
        MOV DX , OFFSET READ_ERR
        JMP ERR_EXIT
READ_OK: MOV DTA+110 , 10
```

```
            MOV DTA+111, 13
            MOV DTA+112, '$'
            MOV DX, OFFSET DTA
            MOV AH, 9
            INT 21H                         ; 显示读出内容
            JMP KB_IN
EXIT:       MOV AH, 10H
            MOV DX, OFFSET FCB
            INT 21H                         ; 关闭文件
            OR AL, AL
            JZ CLOSE_OK
            MOV DX, OFFSET CLOSE_ERR
            JMP ERR_EXIT
CLOSE_OK:   INT 20H
CODE        ENDS
            END BGN
```

这个程序是按照 .COM 文件的格式编写的。

关于 .COM文件的规定详见第八章 第七节。

上述程序首先利用11H功能调用根据FCB中的文件名 . 扩展名查找目录项。若指定的文件已存在，则不破坏已有文件，退出程序，若指定的文件不存在，则利用 16H功能调用在磁盘上建立并打开这个文件。然后程序向文件中写入10个记录。每个记录110 个字节（字符），分别是110个'0'，110个'1'，. . . . . 110个'9'。接下来程序提示用户输入相对记录号，当用户输入 0-9之间的数字时，程序将相对应的记录从磁盘上随机读出，并显示在屏幕上。当用户输入 $字符时，程序运行结束，正常退出，返回操作系统。当打开、关闭、读、写文件不成功时，程序给出提示，然后退回操作系统。

程序中打开文件利用的是 16H功能调用，它建立并打开文件，若对已存在的文件，可用0FH功能调用打开。打开文件时，DOS搜索磁盘目录，找到那个文件后，设置 FCB与文件长度有关的字段。接下来把记录长度由系统隐含的128个字节改为程序要求的110个字节。文件一经打开就不必要每次读写文件时都去搜索磁盘目录， 而由系统自动维护 FCB中与读写文件有关的信息，直到关闭这个文件为止。

程序的WRITE部分向文件中写入10个记录。 先通过REP STOSB 指令将110 个相同的字符写入 DTA，然后通过 15H 调用将 DTA中的记录写入文件中。INC AL指令产生字符'1'～'9'，准备好了10条记录的数据。

程序的KB_IN部分利用1号功能调用接收键盘输入相对记录号，判断输入是否在允许范围内。然后程序用 21H功能调用采取随机读方式把用户指定的记录从文件中读出，并显示在屏幕上。

程序的最后部分是关闭文件。关闭文件的一个重要功能是保证 DOS会把所有已写入或修改过的记录都写到磁盘上。因为一条记录的字节数比较少(最多 128个字节)，而磁盘扇区比较大(512/1024字节等等)，所以在正常的程序执行中，为了加快速度，并不是每写入/修改一条记录，DOS都启动一次写盘操作，而是积累到一定程序才启动一次写盘操作。这样就可能会造成最后几个记录没写到盘上去。关闭文件会把缓冲区的内容全部写到磁盘上去。

程序中在 100H(标号 BGN)处安排一条JMP指令，而且数据在程序之前，这样做不易出错，数据在后易错。

例 2. 利用文件标记读写文件

利用文件标记读写文件需要一个 ASCII码字符串，这个问题可以通过提请用户键盘输入的方法解决。下述程序可以将一个指定的已存在的文件拷贝成另一个指定的文件。 DOS 2.0 以后的版本可以支持这个程序。

```
STACK SEGMENT PARA STACK'STACK'
      DB 100DUP(9)
STACK ENDS
DATA  SEGMENT
SFILE DB 64
      DB ?
      DB 64 DUP(' ')
DFILE DB 64
      DB ?
      DB 64 DUP(' ')
ASK1  DB 0AH,0DH, 'PLEASE INPUT SOURCE'
      DB 'FILE NAME ', '$'
ASK2  DB 0AH,0DH, 'PLEASE INPUT DESTNATION'
      DB 'FILE NAME ' '$'
NOTE  DB 0AH,0DH, 'PLEASE INSERT DISKETTES'
      DB 'AND STRIKE ANY KEY WHEN READY ',' '$'
ER1   DB 0AH, 0DH, 'CREATE ERROR ','$'
ER2   DB 0AH, 0DH, 'OPEN ERROR','$'
ER3   DB 0AH, 0DH, 'READ ERROR ','$'
ER4   DB 0AH, 0DH, 'WRITE ERROR','$'
```

```
ER5   DB 0AH, 0DH, 'CLOSE SOURCE FILE ERROR', '$'
ER6   DB 0AH, 0DH, 'CLOSE DEST FILE ERROR', '$'
BUFR  DW ?                    ; 存放文件标记
DATA  ENDS
CODE  SEGMENT
      ASSUME CS: CODE, DS: DATA, ES: DATA
START PROC FAR
      PUSH DS
      SUB AX, AX
      PUSH AX
      MOV AX, DATA
      MOV DS, AX
      MOV ES, AX
      LEA DX, ASK1
      CALL DISP               ; 提示输入源文件标识符
      LEA DX, SFILE
      CALL INPT               ; 接收源文件标识符
      MOV CL, SFILE+1
      XOR CH, CH
      MOV SI, CX
      MOV [SI+SFILE+2], 0     ; 在(文件标识符)字符串后加00H字节
      LEA DX, ASK2
      CALL DISP
      LEA DX, DFILE
      CALL INPT               ; 接收目标文件标识符
      MOV CL, DFILE+1
      XOR CH, CH
      MOV SI, CX
      MOV [SI+DFILE+2], 0
      LEA DX, NOTE
      CALL DISP               ; 提示插入磁盘
      MOV AH, 7
      INT 21H
      CALL COPY
      RET
```

```
START ENDP
DISP  PROC NEAR
      MOV AH , 9
      INT 21H
      RET
DISP  ENDP
INPT  PROC NEAR
      MOV AH , 0AH
      INT 21H
      RET
INPT  ENDP
COPY PROC NEAR
     MOV AH , 3CH
     LEA DX , DFILE+2
     MOV CX , 0020H
     INT 21H                    ; 建立目标文件
     LEA DX , ER1
     MOV BX , AX
     JC ERR
     MOV BUFR , AX
     MOV AH , 3DH
     MOV AL , 0
     LEA DX , SFILE+2
     INT 21H                    ; 打开源文件
     LEA DX , ER2
     MOV BX , AX
     JC ERR
R_W: MOV CX , 0010H
     MOV AH , 3FH
     LEA DX , SFILE+2           ; 读源文件
     INT 21H
     LEA DX , ER3
     JC ERR
     OR AX , AX
     JE EXIT
     MOV AH , 40H
     LEA DX , SFILE+2
```

```
        XCHG BUFR , BX
        INT 21H                         ; 写入目标文件
        LEA  DX , ER4
        JC ERR
        XCHG RUFR , BX
        JMP R_W
EXIT:   MOV AH , 3EH                    ; 正常结束，关闭文件
        INT 21H
        LEA  DX , ER5
        JC ERR
        XCHG BUFR , BX
        MOV AH , 3EH
        INT 21H
        LEA  DX , ER6
        JC ERR
        RET
ERR:    MOV AH , 3EH                    ; 出现错误，关闭文件
        INT 21H
        XCHG BUFR , BX
        MOV AH , 3EH
        INT 21H
        CALL DISP                       ; 错误提示
        RET
COPY    ENDP
CODE    ENDS
        END START
```

程序的数据段定义了两个输入缓冲区，是按照 0AH功能调用要求的格式定义的。一个是存放源文件的ASCII码字符串(包括盘号、文件名、扩展名)用的SFILE，另一个是存放目标文件的ASCII码字符串用的DFILE。紧接着，数据段中又定义了操作提示信息和错误提示信息。

代码段中，初始化之后，首先提请用户输入源文件标识符。SFILE+1 单元中存放着实际输入的字符个数。接下去程序自动在输入的字符串后加00H字节，这是文件使用的ASCII码字符串要求的。按照同样的方式，程序又提示用户输入目标文件标识符，也在实际输入的字符串后加了 00H字节。

然后，程序提请用户插入软盘，当用户插入软盘，并按任一键之后，程序就开始了复

制工作。

复制过程中，首先利用了3CH功能调用建立目标文件，将20H送入CX表示为隐藏文件。再利用3DH功能调用打开源文件，AL内送入0，表示打开文件只供读用。接下来，程序开始重复读写工作，直至源文件结束。每次利用3FH功能调用从源文件中读出 16个字节，再利用 40H功能调用将这16个字节写入目标文件。读写过程使用的数据缓冲区首址是SFILE+2。

程序中特别注意的是文件标记的使用。在建立目标文件之后，将所得到的文件标记存到了BUFR 变量中，打开源文件之后，把其文件标记存到了BX寄存器。自WRITE以后的程序段中，多次使 BUFR和BX中的内容互换，目的都是为了在文件的读/写过程中引用相应的文件标记。

遇到文件结束时，利用 3EH功能调用将两个文件都关闭，然后退出程序，返回系统。当在打开、建立、读、写、关闭文件的任一过程中出现错误时，程序给出错误提示，关闭文件而后退回系统。

本节所介绍的系统功能调用，以后的章节中还将应用。

本章所介绍的分支程序、循环程序、子程序等的设计方法是汇编语言程序设计的基本方法，应该熟练地掌握它。DOS 系统功能调用为汇编语言程序员提供了很大的方便，很好地利用它，可以充分利用机器的性能，提高程序设计的效率，提高程序的质量和功能。

习题四

1．为符号函数例题画程序流程图。

2．某软件共可接收10个键盘命令(分别为A，B，C，．．．．．J)，完成这 10个命令的程序分别为过程P0，P1．．．．．．P9。编程序从键盘接收命令，并转到相应的过程去执行。要求用两种方法：

(1)用比较、转移指令实现。

(2)用跳转表实现。

3．内存自BUF单元开始的缓冲区连续存放着1000个学生的英文分数，编程序统计其中 90-100、60-89、60以下者各有多少人，并把结果连续存放到自RESUT开始的单元(要求画出程序流程图，各段定义完整)。

4．编程序重复接收学生分数（用A、B、C、D表示），且自动汇总各种分数的人数，并输出显示汇总结果。若按下 $键，表示输入结束。按下A、B、C、D以外的任一键都无效，并提示ERROR。(要求同上题)。

5．子程序文件说明中应包括哪几方面的内容，为什么?

6．系统功能调用的方法如何，举例说明 ?

7．编程序一边从键盘上接收字符，一编将其写入指定的文件，字符个数共 60。

8．编程序顺序读第 7 题中产生的文件内容，并将其显示在屏幕上。

第五章　输入输出和中断

第一节　输入和输出

通常计算机与外设之间的信息交换是通过读写外设端口实现的。8088与外设之间的数据交换是8位的，它可提供64KI/O端口空间(即外设端口地址)。而8086一次可传送 16位数据，它提供32KI/O端口空间，8086和8088用相同的输入输出指令在累加器和外设端口之间传送数据。

I/O 端口空间也可以安排在存贮器空间， CPU 对待 I/O 设备就象对待存贮器一样（只要 I/O 设备象存贮器一样响应 CPU ），因而可以利用丰富的存贮器访问指令访问 I/O 设备，从而提高了程序设计的灵活性。但这种方式减少存贮器的地址空间（ 因为 I/O 设备占用了部分地址空间)。

一、输入输出指令

1．输入指令 IN

输入指令允许把一个字节或一个字由一个输入端口(PORT)传送至AL(若是一个字节)或AX(若是一个字)。例如：

```
IN    AL，n
IN    AX，n
IN    AL，DX
IN    AX，DX
```

指令中 n与 DX指明端口地址。若为n，则直接寻址端口0～255，共256个 PORT，若为DX，则端口地址在DX中，可达64K个PORT，为间接寻址方式。

2．输出指令　　OUT

输出指令把在AL中的一个字节或在AX中的一个字传送至一个输出端口。端口寻址方式与IN指令相同。例如：

```
OUT    n，  AL
OUT    DX，  AL
OUT    n，  AX
OUT    DX，  AX
```

二、CPU与外设数据传送的方式

CPU与一个I/O设备之间的交换信息包括数据、状态和控制信息。

(1)数据：

数据通常为8位或16位，可分为三种基本类型：数字量、模拟量和开关量。

由键盘、光电输入机等提供的二进制形式的信息为数字量数据。

可用两个状态表示的量，如电机的启停、开关的开合等，只要用一位二进制数即可表示的量，称为开关量。

由传感器等提供的信号往往是模拟量，它需先经模 / 数（A/D）转换后再输入到计算机去。例如：温度、电压等信号。

数据有串行传送(一位一位传送)和并行传送（n位同时传送)方式，但都要经过I/O指令实现。

(2)状态信息

在输入时，有表示输入装置是否已准备好的信息（READY)；在输出时，有表示输出装置是否忙(BUSY)的信息等。

(3)控制信息

例如控制I/O装置的启停等信号。

状态信息和控制信息与数据是不同性质的，必须要分别传送。但它们都通过 IN和OUT指令在数据总线上进行传送，所以通常采用分配不同端口的方法将它们加以区别。

CPU与外设数据传送的方式一般有如下几种：

1．无条件传送方式：

在不需要查询外设的状态，即已知外设已准备好或不忙时，可以直接使用 IN 或 OUT 指令与外设传送数据。这种方式软硬件简单。只要指令中指明端口地址，就可选通指定外设进行输入输出。

例1：一个无条件传送的例子如图 5-1。图中 10H、11H、20H 是来自地址总线的端口地址信号。

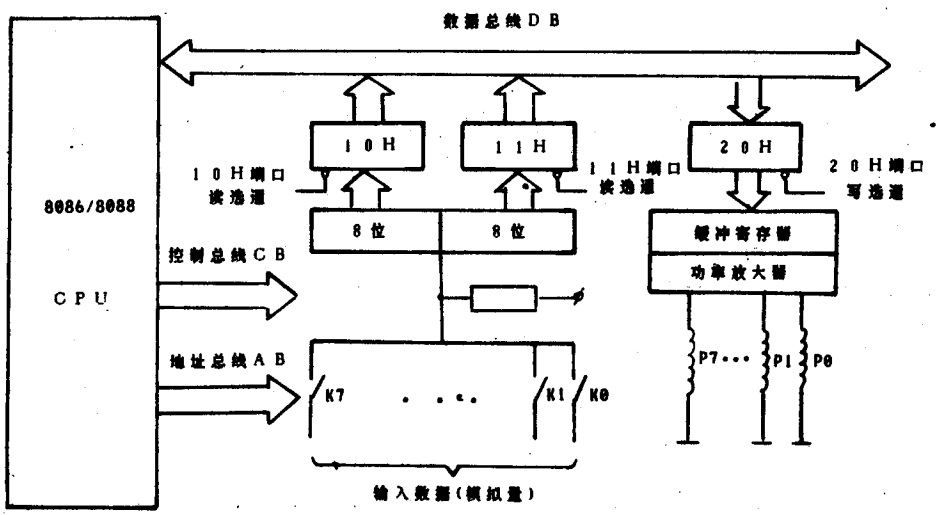


图 5 - 1 无条件传送举例

这是一个同步的数据采集系统。被采样的数据是8个模拟量，由继电器绕组P0、P1...P7 控制触点 K0、K1.....K7逐个接通，用一个四位(10进制数)数字电压表测量，把被采样的模拟量转换成16位 BCD代码，高8位和低8位通过两个不同的端口输入，他们的地址分别为 10H 和 11H。CPU 通过端口20H输出控制信号，以控制继电器约吸合，实现采集不同的模拟量。

数据采集过程，可用以下程序来实现：

```
START: MOV DX , 0100H          ; 01→DH，设置合第一个继电器代码
                               ;  00→DL，设置断开所有继电器代码
       LEA BX , DSIOK          ; 输入数据缓冲区的地址偏移量→BX
       XOR AL                  ; 清AL及进位标志
AGAIN: MOV AL , DL
       OUT 20H , AL            ; 断开所有继电器线圈
       CALL NEAR DELAY1        ; 模拟继电器触点的释放时间
       MOV AL , DH
       OUT 20H , AL            ; 使P0吸合
       CALL NEAR DELAY2        ; 模拟触点闭合及数字电压表的转换时间
       IN AX , 10H             ; 输入
       MOV [BX] , AX
       INC BX
       INC BX
       RCL DH , 1              ; DH左移一位，为下一个触点闭合作准备。
       JNC AGAIN               ; 8个模拟量未输入完，循环此段程序。
```

例 2. 编写产生任意波形、任意频率的脉冲信号程序。现给定三种波形A、B和C 如图5-2 所示。

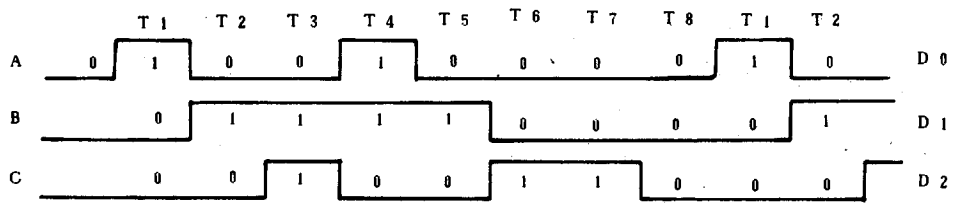


图 5 - 2 输出波形

其中时间间隔 T=10ms 每种波形由 T1～T8 八个状态组成，要求连续重复发送 T1～T8 信号。

上述波形信号若由硬件电路实现，比较麻烦，需要硬件开支，而且改变波形或频率也不方便。

利用数据总线的D0、D1、D2位产生波形A、B、C，并将此信号送至输出缓冲器去。如图5-3所示。

可以编写一个延时10ms的软件延时程序控制时间间隔。把每个 T周期内波形的变化数值化，得数据01H，02H，06H，03H，02H，04H，04H，00H。产生上述波形的程序如下所示。

```
DATA SEGMENT
 TAB DB 08，1，2，6，3              ；波形数据
     DB 2，4，4，0
PORT EQU 80H                       ；输出缓冲器端口地址为80H
DATA ENDS
STACK SEGMENT PARA STACK 'STACK '  ；堆栈段
      DB 100DUP(9)
STACK ENDS
CODE SEGMENT                       ；代码段
     ASSUME CS: CODE，DS: DATA，SS: STACK
BGN: PUSH DS
     MOV AX，0
     PUSH AX
     MOV AX，DATA
     MOV DS，AX
STR: LEA BX，TAB
     MOV CH，0
     MOV CL，[BX]
LOP: INC BX
     MOV AL，[BX]
     OUT PORT，AL                  ；输出波形
     PUSH CX
     CALL DELY                     ；调用延时过程。
     POP CX
     LOOP LOP
     JMP STR
DELY PROC  NEAR                    ；延时 10 MS 过程。
     MOV CX，2801
WAT: LOOP WAT
     RET
```

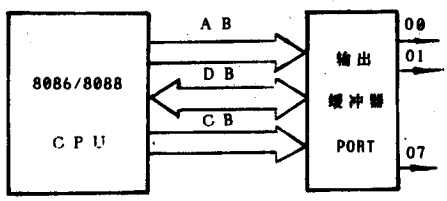


图 5 - 3 输出波形接口电路

```
DELY ENDP
CODE ENDS
    END BGN
```

讨论：(1)通过改变DELY过程的常数2801，可以改变脉冲信号的频率。

(2)通过改变数据段中TAB中的数据，可实现波形的变化。

(3)一个8个输出缓冲器端口可同时输出8路不同的脉冲信号，多个输出端口可同时输出多路脉冲信号。脉冲信号的幅值可由缓冲器进行放大或缩小。

2.查询传送方式

当 CPU 与外部过程是同步工作时， 无条件传送方式是方便的。若两者不同步，则很难确保在 CPU 执行 IN 操作时，外设一定是准备好的；而在执行OUT操作时，外设寄存器一定是空的。所以，通常在程序控制下的传送方式，在传送前，必须去查询一下外设的状态，当外设准备好了才传送；若未准备好，则CPU就等待。

所以，接口电路部分除了传送数据的端口以外，还必须有传送状态信号的端口。

(1)查询式输入：

查循式输入的接口电路如图5-4所示。

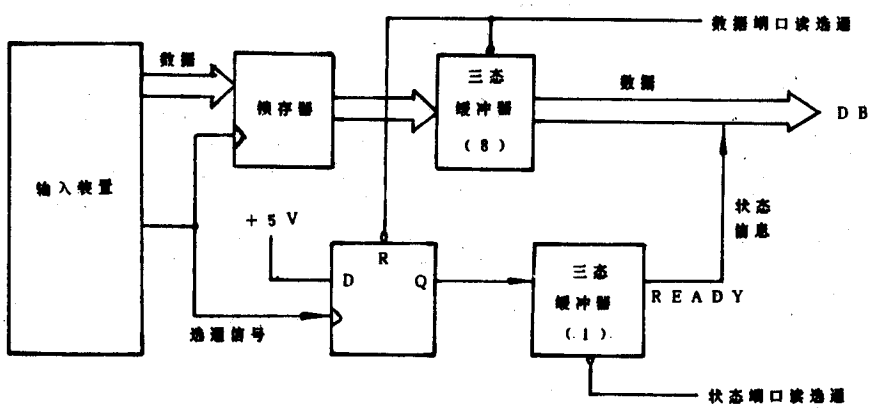


图 5 - 4 查询式输入的接口电路

当输入装置的数据已准备好并发出一个选通信号后，一边把数据送入锁存器，一边使D触发器置“1”，给出“准备好”(Ready)信号。数据与状态必需由不同的端口输入至CPU数据总线。当CPU要由外设输入信息时，CPU先输入状态信息，检查数据是否已准备好，当数据已经准备好后，才输入数据，读入数据的命令，使状态信息清“0”。

读入的数据是8位的；而读入的状态信息往往是一位的（比如用 D7位），如图 5-5 所示。所以，不同外设的状态信息，可以使用同一个端口的不同位。

这种查询输入方式的程序流程图。如图 5-6所示。

查询部分的程序如下：

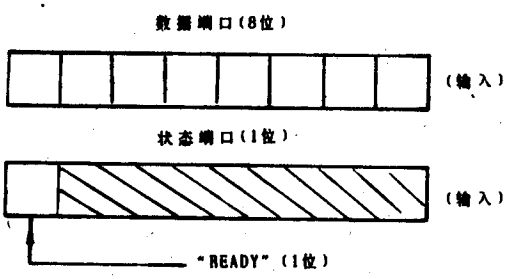


图 5 - 5 查询式输入的数据和状态信息

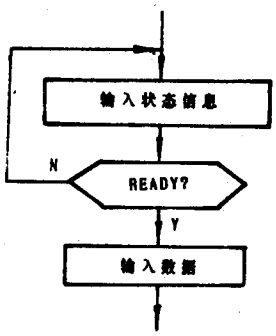


图 5 - 6 查询式输入程序流程图

```
POLL:   IN  AL ,  STATUS _ PORT  ; 从状态端口输入状态信息
        TEST    AL ,  80H        ; 检查READY是否是1
        JE      POLL             ; 未READY，循环
        IN      AL  DATA _ PORT  ;  READY，从数据端口输入数据。
```

这种 CPU 与外设之间状态信息的交换方式，称为应答式，状态信息称为“联络”（handshake）信息。

(2)查询式输出

在输出时 CPU 必须了解外设的状态，看外设是否为空(即外设若不处在输出状态，或外设的数据寄存器是空的就可以接收 CPU输出的信息)，若为空，则 CPU执行输出指令；否则就等待。因此，接口电路中也必须要有状态信息的端口，其方框图见图5-7。

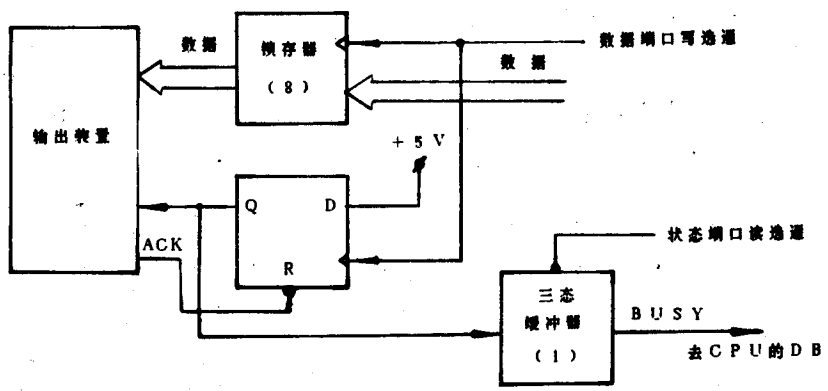


图 5 - 7 查询式输出接口电路

当输出装置把 CPU 输出的数据输出以后，发出一个 ACK (Acknowledge) 信号，使 D 触发器置 “0”，也即使“BUSY”线为 0，当 CPU 输入这个状态信息后，知道外设为“空”，于是就执行输出指令，输出指令执行后，发出选通信号，把在数据线上的输出数据送至锁存器。同时，令 D 触发器置“1”，它一方面通知外设输出数据已经准备好，可以执行输出操作；另一方面在数据由输出装置输出以前，一直为“1”，告知CPU(CPU通过读

状态端口而知道)外设“Busy”，阻止CPU输出新的数据。

输出接口电路的端口信息为：数据端口 8 位；状态信息一位。如图5-8所示。

查询式输出的程序流程图如图5-9所示。

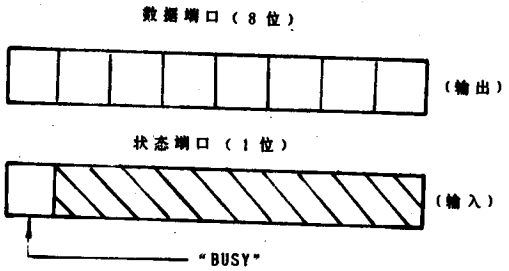


图 5 － 8 查询式输出的端口信息

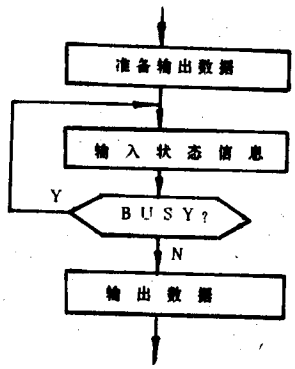


图 5 － 9 查询式输出程序流程图

查询部分的程序为：

```
POLL:  IN    AL , STATUS_PORT    ; 从状态端口输入状态信息。
       TEST  AL , 80H            ; 检查BUSY位。
       JNE   POLL                ; BUSY则循环等待。
       MOV   AL , STORE          ; 否则从缓冲区取数据。
       OUT   DATA_PORT , AL      ; 从数据端口输出。
```

其中，STATUS_PORT是状态端口的符号地址； DATD_PORT 是数据端口的符号地址；STORE是内存数据单元的地址。

(3)查询传送方式举例

例 1： 一个有8个模拟量输入的数据采集系统，用查询的方式与 CPU传送信息，电路如图5-10所示。

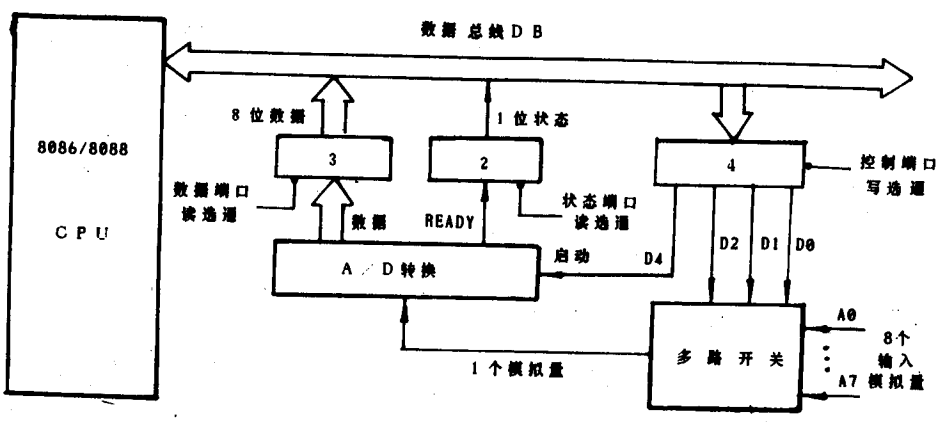


图 5 － 1 0 查询式数据采集系统

8个输入模拟量，经过多路开关选通后送入 A/D 转换器，多路开关由端口 4 输出的三位二进制码（对应于 D0、D1、D2位)控制。000 相应于选通 A0输入，···，111选通 A7 输入。每次只送出一个模拟量至 A/D 转换器；同时由端口4输出的 D4位控制A/D 转换器的启动与停止。A/D 转换器的READY信号由端口2的D0输至CPU数据总线，经A/D 转换后的数据由端口3输至数据总线。所以，这样的一个数据采集系统，需要用到三个端口，它们有各自的地址。

实现这样的数据采集过程的程序为：

```
START:  MOV DL , 0F8H          ; 设置启动A/D转换的信号。
        MOV DI , OFFSET DSTOR  ;  输入数据缓冲区的地址偏移量→DI
AGAIN:  MOV AL , DL
        AND AL , 0EFH          ; 使D4=0
        OUT 4 , A              ; 停止A/D转换。
        CALL DELAY             ; 等待停止A/D操作的完成。
        MOV AL , DL
        OUT 4 , AL             ; 启动A/D，且选择模拟量A0。
POLL:   IN AL , 2              ; 输入状态信息。
        SHR AL , 1
        JNC POLL               ; 若未READY，程序循环等待。
        IN AL , 3              ; 否则，输入数据。
        STOSB                  ; 存至内存。
        INC DL                 ; 修改多路开关控制信号，指向下一个模拟量。
        JNE AGALN              ; 8个模拟量未输入完，循环。
            .
            .
            .                  ; 已完，执行别的程序段。
```

例 2：一个利用并行打印机打印寄存器 AL 中的字符的程序如下：

```
; 本过程打印AL寄存器内的字符
; 打印机未打印AL中字符前，程序不返回。
PRINT PROC NEAR
      PUSH AX
      PUSH DX                  ; 保护所用寄存器内容
; 输出数据
      MOV DX , 378H            ; 数据端口地址378H
      OUT DX , AL              ; 输出要打印的字符
; 检查打印机状态
      MOV DX , 379H            ; 状态端口地址379H
```

```
WAT:  IN AL , DX            ; 读打印机状态
      TEST AL , 80H         ; 检查“忙”位
      JE WAT
; 选通打印机 打印
      MOV DX , 37AH         ; 控制端口地址37AH
      MOV AL , 0DH          ; 选通位=1(D0位)
      OUT DX , AL           ; 选通打印机
      MOV AL , 0CH          ; 选通位=0(D0位)
      OUT DX , AL           ; 关打印机选通
      POP DX                ; 恢复寄存器内容
      RET
PRINT ENDP
```

3.中断传送方式

在上述查询传送方式中，CPU要不断地询问外设，当外设没有准备好时，CPU要等待，不能做别的操作。这样就浪费了CPU 的时间。而且许多外设的速度是较低的，如键盘，打印机等等，它们输入或输出一个数据的速度是很慢的，在这个过程中，CPU 可以执行大量的指令。为了提高CPU的效率，可采用中断传送方式，即当CPU需要输入或输出数据时，执行一条指令，发出启动外设工作的命令，然后 CPU就继续执行主程序。输入时，若外设的输入数据已存入寄存器；在输出时，若外设已把上一个数据输出，输出寄存器已空，则由外设向CPU发出中断中请，CPU就暂停原执行的程序(即实现中断)，转去执行输入或输出操作（中断服务），待输入输出操作完成后即返回，CPU再继续执行原来的程序，这样就可以大大提高 CPU的效率，而且有了中断概念，就允许CPU与外设(甚至多个外设)同时工作。

关于 8086/8088 的中断系统详见下节。

有一些可编程的并行接口芯片可用来实现CPU与外设间的数据交换，如 8255等。它们的功能强，通用性好，既可以查询方式也可以中断方式实现输入输出。有关它们的使用可查看 IBM PC 的硬件资料。

4. 直接数据传送(DMA)方式:

中断传送方式可以大大提高CPU的利用率，但它仍然是由CPU通过程序进行传送，每次需要保护断点、保护现场等所用的时间，这对于高速的 I/O设备，以及成批数据交换的情况（例如磁盘与内存间的信息交换），就显得太慢。

所以希望用硬件控制在外设与内存间直接进行数据交换(DMA)而不通过CPU。但是，通常系统的地址和数据总线以及一些控制信号线是由CPU管理的。在DMA方式时，就希望 CPU把这些总线让出来（即 CPU 连到这些总线上的线处于高阻状态）而由 DMA 控制器接管。

DMA 控制器要能控制进行数据交换，它必须有以下功能:

(1)能向CPU发出总线请求信号HOLD。

(2)当CPU发出总线回答信号HLDA后，DMA接管对总线的控制，进入DMA方式。

(3)发出地址信息，能对存贮器寻址及能修改地址指针。

(4)能发出读或写等控制信号。

(5)能决定传送的字节数，及判断DMA传送是否结束。

(6)发出DMA结束信号，使 CPU恢复正常工作状态。

通常 DMA 的工作流程图如图 5-11所示。

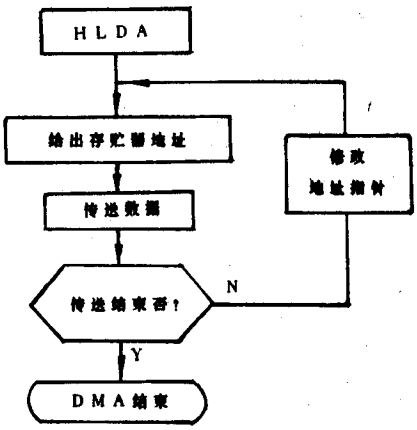


图5－11 DMA工作流程图

DMA 控制器接收到总线回答信号 HLDA 后，进入DMA方式，DMA控制器就接管总线，向地址总线发出地址信号，在数据总线上给出数据，并给出存贮器读或写的命令，就可把由存贮器读出到外设或输入的数据写入存贮器。然后修改地址指针，修改计数器，检查传送是否结束，若未结束则循环直至整个数据传送完。

IBM PC 中使用的DMA控制器是8237A。

在高性能的输入输出应用中，Intel 公司提供了 8089 I/O 处理器（IOP)。在原理上 8089象带有两个 DMA 通道的处理器，但它功能强，能独立工作，本身具有专为输入输出的指令系统，完成存贮器到存贮器或 I/O到I/O的数据传送。

第二节　中断

关于中断的概念前节中已讲过。引起中断的原因或发出中断申请的来源，称为中断源。可以根据重要性的不同为各中断源按排不同的优先级，CPU 首先响应优选权高的中断。8086/8088具备一个简单而灵活的中断系统，它能处理 256 种类型的中断，中断类型由类型号 0～255指定。中断可分为内部中断与外部中断。中断源的情况可见图5-12。

一、外部中断

8086/8088有两条外部中断请求线：

NMI 为非屏蔽中断，INTR 为可屏蔽中断。NMI用于重要的中断源，如电源掉电等，它的类型号是 2。CPU不禁止 NMI 线上的中断请求，但可由指令 STI(中断允许标志置“1”）和 CLI（中断允许标志置“0”）允许和禁止 INTR 线上的中断申请，即 CPU 可用 STI和 CLI 指令开、关中断。当关中断时，CPU 将不响应 INTR 线上的中断申请。

二、内部中断

1.在执行除法指令时，若发现除数为 0或商超过了寄存器所能表达的范围，则立即产生一个类型0的内部中断。

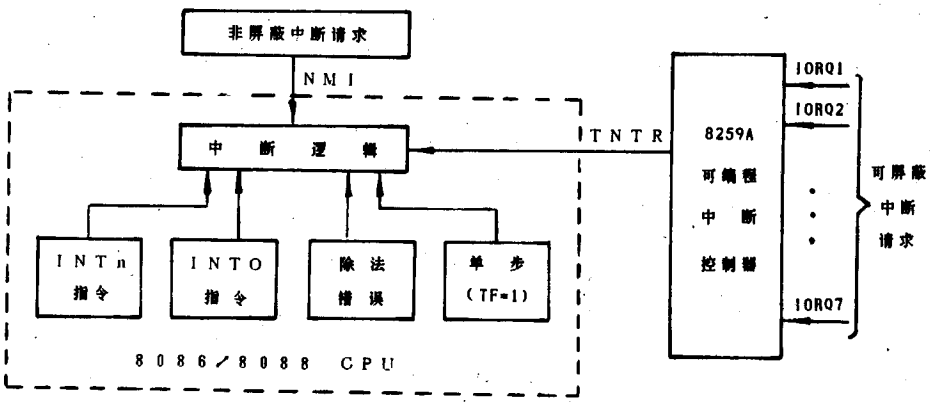


图 5 - 1 2 .8086/8088 的中断源

2.溢出中断指令INTO

若上一条指令使溢出标志OF置1，那么当执行溢出中断指令INTO时， 立即产生一个类型4的中断，若标志OF为0，则此指令不起作用。

3.INT n 指令

CPU执行完INT n指令时立即产生一个中断，所以又称它为“软件中断”。中断的类型由指令中的n指明。因为INT指令中可以指定任何的类型号，故此指令可方便地用来调试为外设编写好的中断服务程序。

4.单步中断

若单步标志 IF 为 1，则在每条指令执行后，CPU 自动产生一个类型 1的中断（单步中断），使程序单步执行，它提供给用户强有力的调试手段。

8086/8088 规定这些中断的优先权从高到底的顺序为：①除法错误、INTO、INT n，②NMI，③INTR，④单步中断。

三、中断矢量表

中断矢量表占用内存中00000H到 003FFH 的 1K 字节空间。 表中内容分为 256 项，对应于类型号 0～255，每一项占用 4 个字节，用来存放相应类型的中断服务程序的入口地址，高两字节存放入口地址的段地址部分，低两字节存放段内偏移地址部分。如图5-13。

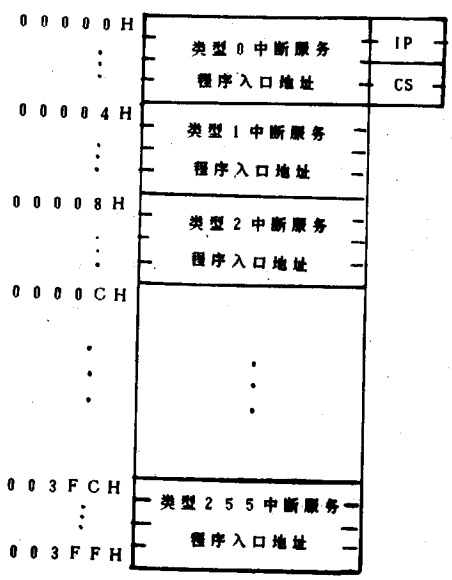


图 5 - 1 3 中断矢量表

四．中断服务程序的调用

对于任一指定类型的中断， CPU 只要将其类型号乘 4 就可以得到其中断矢量（ 即此类中断在中断矢量表中占用的 4 个字节的最低字节的地址 ）， 然后取出它所占有的

4 个字节的内容分别送到 IP 和 CS ，就实现了中断服务程序的调用，所以中断矢量表是中断类型号与其对应的中断服务程序之间的连接链。

中断类型号如何得到呢?

1．除法错误、单步中断、非屏蔽中断NMI、断点中断和溢出中断分别自动提供类型号0～5。

2．对于外部中断 INTR，可以有两种方法提供中断类型号。

第一种方法是自己设计接口电路，利用寄存器/缓冲器或利用组件（如 8212芯片 ）存放中断类型号。CPU 响应中断后，接口电路将此类型号送入数据总线，CPU 读数据总线从而获得中断类型号。

中断方式时接口电路的方框图见图5-14。

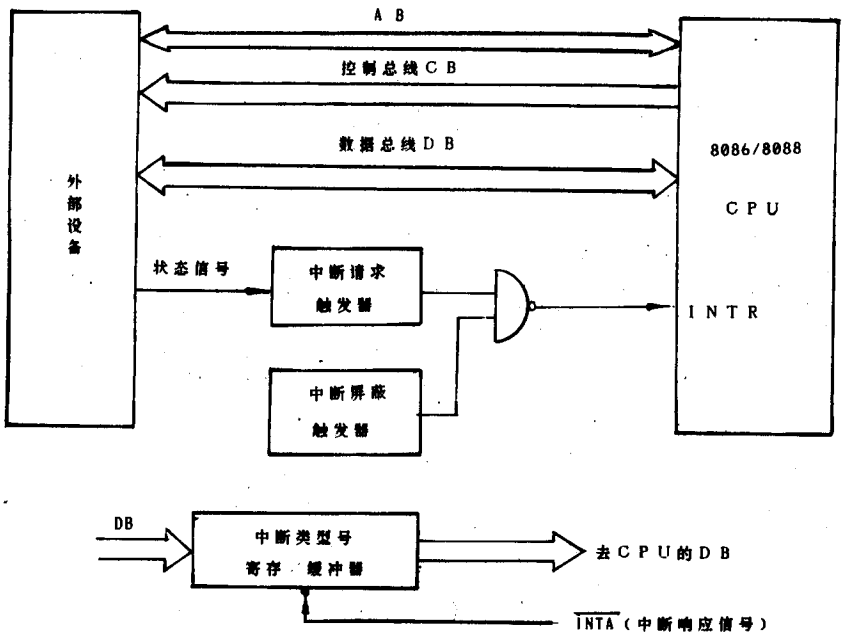


图 5 － 1 4 中断方式接口电路方框图

某一外设的中断类型号可事先由输出指令送入它的中断类型号寄存 /缓冲器或预先将组件(如8212芯片)的引线接好。

当外部设备已准备好数据可以向CPU输送， 或外设已准备好可以接收来自 CPU的信号时，可以在状态信号线上发一脉冲信号，经中断申请触发器向 CPU 的 INTR线发出中断申请，CPU响应中断后，进入中断响应周期，发 INTA信号，此信号将已预先装入的或由硬件芯片提供的中断矢量号送入数据总线，CPU 即可读得。在中断服务程序中可以按排与此外设的数据交换。

图中的中断屏蔽触发器可以通过设置它为 1 或为 0 来控制是否让外设发中断申请。

例如：已为某一外设指定中断类型号为 6，为此外设编写了中断服务程序在代码段程

序中，如下述。

```
CODE   SEGMENT
MAIN:                          ; 主程序
        .
        .
        .
       HLT
INTR6  PROC   NEAR             ; 中断类型号为6的中断服务程序(过程)
        .
        .
        .
       STI
       IRET
INTR6  ENDP
        .
        .
        .
CODE   NED
```

做了以上安排后，首先不要忘记应在主程序的初始化部分将此中断服务程序的入口地址送入中断矢量表内对应中断类型6的4个单元中。如下指令可以完成这个任务：

```
MOV AX，0
MOV ES，AX
MOV DI，0*H*4
MOV AX，OFFSET INTR6
CLD
STOSW
MOV AX，CS
STOSW
```

另外在所有中断服务程序的结尾处不要忘记安排开中断指令STI和从中断返回的指令IRET。

关于中断系统的应用，本章第四节中有综合举例。

获得中断类型号的第二个方法是利用Inte8259(或8259A)芯片。Intel 8259A 是可编程的中断控制器，如图5-12所示，它可以接收来自外设的8个各自独立的中断申请信号，分别为 IRQ0～IRQ7。8259A 将它们按优先权的高低进行排队。IRQ0优先权最高，依次降低，IRQ7 最低。当同一时刻出现两个或两个以上的中断申请信号时，8259A 首先响应优先权高者。将中断信号送到 8086/8088 的 INTR 线上，进而又将对应于该中断源 的唯一的中断类型号送给 8086/8088。CPU获得此中断类型号，就自动转入对应的中断服务程序。

IBM PC内装有一片8259A芯片，适当的接线和配置已安排好，表 5-1 列出了为每一个

中断源 分配的标准设备。中断源0～7队应于中断类型号08H～0FH。其中 8259A 的输入端 IRQ3 和 IRQ5未用，用户可以使用。

表 5-1 8259A的中断源

8259A输入	中断类型号	设 备
IRQ0	08H	定时器(通道0)
IRQ1	09H	键盘
IRQ2	0AH	彩色图象接口
IRQ3	0BH	未用
IRQ4	0CH	串行(RS-232)接口
IRQ5	0DH	未用
IRQ6	0EH	软盘
IRQ7	0FH	打印机

8259A 中有一个中断屏蔽寄存器(IMR)，它的 I/O 端口地址是 21H。它的位0～位7对应于IRQ0～IRQ7，可以通过设置这个寄存器的任一位为 0 或 1 去控制任一中断源的中断允许或禁止。某位为0表示允许该中断源发出的中断申请信号经8259A产生一个要发给8086/8088的中断，为1则禁止该中断源。例如：只允许键盘中断，则可设置如下中断屏蔽字：

```
MOV  AL，0FDH
OUT  21H，  AL
```

使用8259A时，应在主程序开始处按如上原则初始化中断屏蔽寄存器。在中断服务程序的结束处应发出“中断结束”(EOI)命令(20H)给8259A的中断命令寄存器（I/O 端口地址 20H），具体程序如下：

```
MOV  AL，  20H
OUT  20H，  AL
```

关于8259A的使用，本章第四节中有举例。

五、中断响应过程

当一个可屏蔽中断被响应后，CPU 进入中断响应周期。此时被响应的外设应将本身的中断类型号送往数据总线(NMI和内部中断不做此工作)。CPU读取这个类型号，将其乘 4，就得到中断矢量表中相应的中断矢量入口。

在调用相应的中断服务程序前，CPU 先将机器状态用标志位入栈的方法保存起来。接着 CPU清除标志位I和T，屏蔽新的中断请求和单步中断。然后，CPU 把当前的码段寄存器的内容入栈保护，从向量表(高两个字节)中取出新的码段寄存器值送至 CS；接着CPU又把当前的指令指针值入栈，再从向量表中取出新的IP值，送至IP中。于是程序就转到了中断服务程序。

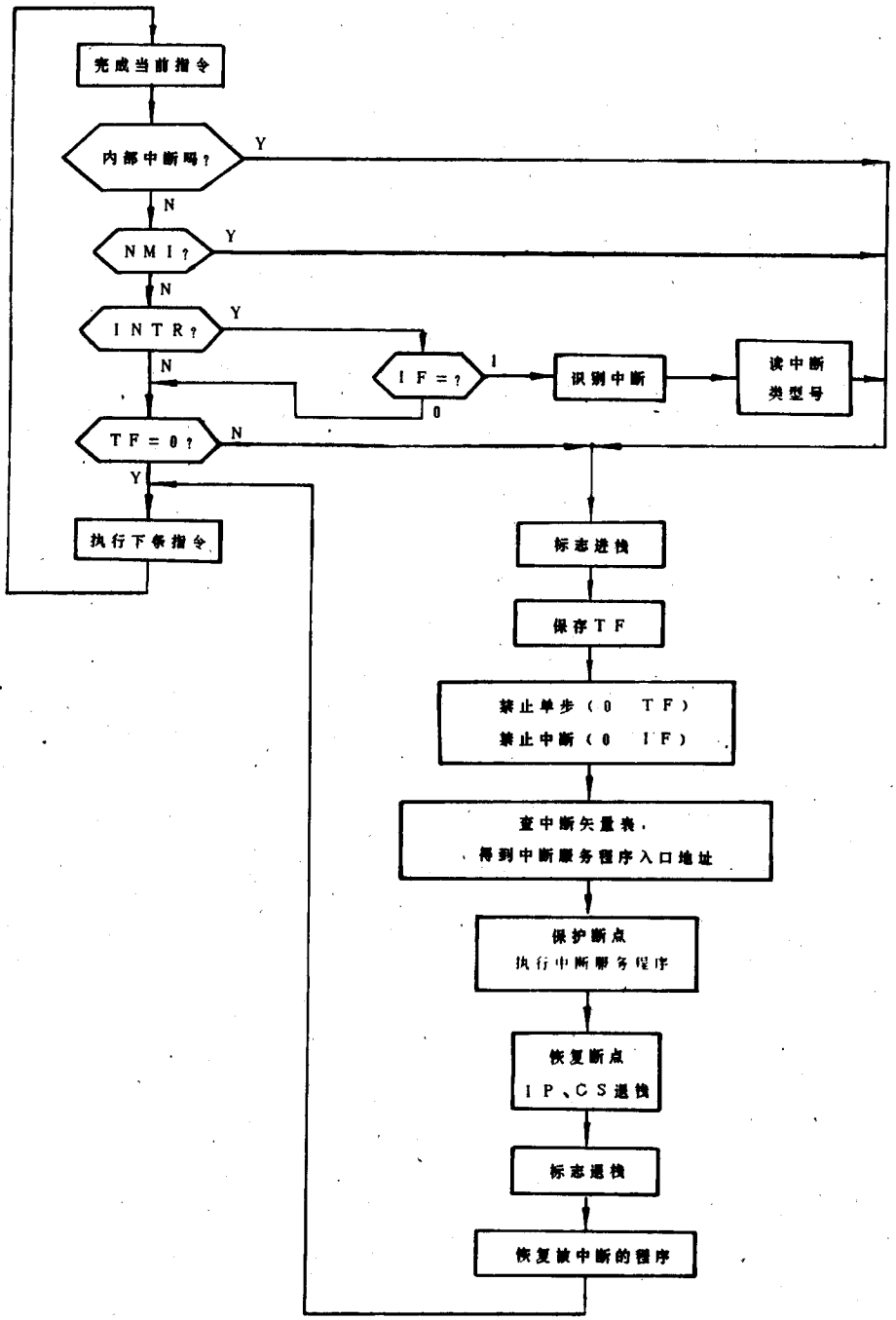


图5－15中断响应过程

执行中断服务程序

中断服务程序可按各个设备的要求来加以编制，但通常有保护现场(入栈)指令，在返

回前要恢复现场（退栈指令），最后要用中断返回指令 IRET恢复断点处的 CS值和 IP值，以及保存的CPU的状态。

8086/8088中断响应过程的基本流程图如图5-15所示。

第三节 ROM BIOS 中断调用

一、概述

IBM PC 的系统板中装有40K ROM，地址从0FE000H开始。其中 8K为 ROM BIOS，32K为BASIC解释程序。

驻留在 ROM 中的基本输入输出程序 BIOS 提供了系统加电自检，引导装入以及对主要 I/O 接口控制等功能。其中对 I/O 接口的控制，主要是指对键盘、磁带、磁盘、显示器、打印机、异步串行通讯接口等的控制，此外 BIOS 还提供了最基本的系统硬件与软件间的接口。

ROM BIOS为程序员提供了很大的方便。汇编语言程序员可以不必了解硬件 I/O接口的特性，而靠直接调用 ROM BIOS中的程序完成对主要I/O设备的控制管理。

系统软件程序员也需要调用ROM BIOS中的程序，例如 PC-DOS、CP/M 86等操作系统也都调用了 ROM BIOS中的程序，从而使操件系统程序精练、简洁、易于移植等。

二、ROM BIOS 的使用

ROM BIOS 由许多功能模块组成。每个功能模块的入口地址都在中断矢量表中，通过软件中断指令可以直接调用这些功能模块，称为中断调用。

ROM BIOS 使用了中断类型号8～1FH。

各类中断的功能如下所示：

中断类型号(H)	功能	中断类型号(H)	功能
0	除法错误	1	单步中断
2	非屏蔽中断NMI	3	断点中断
4	溢出中断	5	屏幕打印中断
6	保留	7	保留
8	定时中断	9	键盘中断
A	保留	B	异步通讯口2中断
C	异步通讯口1中断	D	硬盘中断
E	软盘中断	F	打印机中断
10	显示器I/O调用	11	设备检验调用
12	存贮器检验调用	13	软盘I/O调用
14	异步通讯I/O调用	15	磁带I/O调用

16	键盘I/O调用	17	打印机I/O调用
18	磁带BASIC入口点	19	自举程序入口
1A	时间调用	1B	Ctrl-Break控制
1C	定时处理	1D	显示参数表
1E	软盘参数表	1F	字符点阵结构参数表
20	程序结束运行，返回DOS	21	系统功能调用
22	结束地址	23	CrtL--Break退出地址
24	标准错误处理	25	绝对磁盘读
26	绝对磁盘写	27	程序运行结束，但驻留内存
28～3F	为DOS保留	40～7F	未用，用户可用
80～85	为BASIC保留	86～F0	BASIC使用
F1～FF	未用		

其中：中断类型号 8～1FH为 ROM BIOS 使用，20H～2FH 为 DOS 使用，0～4 为专用中断。

中断调用的方法是：首先给出入口参数，然后写明软件中断指令。例如：

```
INT 12H
```

这将调用存贮器容量测试程序。这个程序不需要入口参数，它以 1KB为测试单位，并将测试结果送入 AX寄存器。

通常 BIOS 程序保护除 AX 和状态标志位以外的所有寄存器，BIOS 程序清单中每一个功能模块的说明中都确切地注明了寄存器的使用情况。

二、中断调用举例

下面就部分中断调用加以介绍。

1.键盘 I/O 中断调用（16H）

16号中断调用有三个功能，功能号在AH中。

(1)AH=0

入口参数：AH=0

功　　能：从键盘读入字符送AL寄存器。

出口参数：AL中为键盘输入的字符的ASCII码值

(2)AH=1

入口参数：AH=1

功　　能：从键盘读入字符送 AL，并设置 ZF标志，若按过任一键（即键盘缓冲区不空），置ZF=0，否则ZF=1。

出口参数：若ZF=0，则AL中为输入的字符的ASCII码。

(3)AH=2

入口参数：AH=2

功　　能：读取特殊功能键的状态

出口参数：AL 为各特殊功能键的状态，位 7 是插入键（INS），位 6 是大小号字母键（CAPS），位 5 是数字键（NUM），位 4 是滚动键（SCROU），位 3 是交替键（ALT），位 2 是控制键（CTL），位 1 是左边的 SHIFT键，位 0 是右边的 SHIFT键。

例　MOV AH，0

　　INT 16H

调用结果，将键盘输入字符的 ASCII 码值送 AL 中。

2．打印机 I/O 中断调用（17H）

17H中断调用有三个功能，功能号在AH中，打印机号在DX中(BIOS支持三个打印机，编号分别为0，1，2)。

(1)AH=0

入口参数：AH=0，DX=打印机号，AL=要打印的字符的ASCII码。

功　　能：把AL中指定的字符在打印机上打印出来。

出口参数：无。

(2)AH=1

入口参数：AH=1，DX=打印机号，AL=初始化命令。

功　　能：按 AL中的命令对打印机初始化。

出口参数：无。

(3)AH=2

入口参数：AH=2，DX=打印机号。

功　　能：读取打印机的状态信息。

出口参数：打印机的状态在 AL 中，AL 的位 0 表示是否超时（1为超时），位 3表示是否错误（1为错误），位 4 是选择信号，(1 表示打印机连机，0 表示打印机未连机)，位 5 表示是否有纸（1为无纸），位 6 是肯定信号，位 7 表示是否忙（1 为忙）。

例：MOV DX，0

　　MOV AL，'A'

　　MOV AH，0

　　INT 17H

调用结果，打印机将字符A打印出来。

3.时间中断调用（1AH）

1AH中断调用有两个功能，功能号在AH中。

(1)AH=0

入口参数：AH=0。

功　　能：读取时间计数器的当前值。

出口参数：CX=计数值的高位字，DX=计数值的低位字。若上次读它后，计数未超过24小时，AL=0；否则AL≠0。

(2)AH=1

入口参数：AH=1，CX=时间值的高位字，DX=时间值的低位字。

功　　能：设置时间计数器的当前值。

出口参数：时间计数器设为CX与DX中的值。时间计数器是每55个ms自动加 1的。

例 1：
```
MOV AH，1
MOV CX，0
MOV DX，0
INT 1AH
```

调用结果，将时间计数器的当前值设置为零。

例 2：
```
MOV AH，0
INT 1AH
```

调用结果，在CX：DX得到时间计数器的当前值。因为时间计数器每55ms计数加 1，所以 CX：DX中的数除以65520得小时数，余数再除以1092得分数，所得余数再除以18.2得秒数。

我们可用1AH中断调用作定时控制，例如要求每隔 5.5 s 从键盘读一个字符并将这个字符的 ASCII码送入内存输入缓冲区中，共读入 100个字符，实现这个要求的程序如下：

```
XI    DB 100 DUP(? )          ；定义100字节的输入缓冲区
STR:  MOV CX，100
      LEA BX，XI
      STI
LOP1: MOV AH，1
      PUSH  CX
      MOV CX，0
      MOV DX，0
      INT 1AH                 ；设置时间计数器为0
LOP2: MOV AH，0
      INT 1AH                 ；读时间计数值
      CMP DL，100
      JNZ LOP2
      MOV AH，0
      INT 16H                 ；从键盘接受字符
     MOV [BX]，AL
     INC BX
     POP CX
     LOOP LOP1
```

```
        HLT
```

例 3： 计算程序执行时间。

如果你已为某一外设编写好一段中断服务程序，可以利用 INT n中断指令，模拟外设发中断请求，以便调试或执行你的中断服务程序，如果想计算某段程序（过程）的执行时间，可以借助1AH中断调用，由程序自动计算，具体程序如下：

```
        STI
        MOV CX,  0
        MOV DX,  0
        MOV AH,  1
        INT 1AH                 ;  设置时间计数器初始为0
        CALL   PROCES           ;  执行过程
        MOV  AH,0
        INT  1AH                ;  读时间计数器的值
```

由时间计数器的值(返回在CX：DX中)乘以0.055或除以18.2，即可得PROCES过程执行的时间，这种计算方法所产生的误差约为54.9ms。

例4．随机数的产生。

利用1AH中断调用也可以产生随机数，如下过程可以产生 0～50 之间的随机整数在BL中。

```
RAND    PROC    NEAR
        PUSH    CX
        PUSH    DX
        PUSH    AX
        STI
        MOV     AH,0
        INT     1AH
        MOV     AX,DX
        AND     AH,3
        MOV     DL,51
        DIV     DL
        MOV     BL, AH
        POP     AX
        POP     DX
        POP     CX
        RET
RAND    ENDP
```

为避免除法结果溢出，程序中将AH寄存器的高6位清零。

关于1AH中断的调用在第六章中还会用到。

4．显示器I/O中断调用 （10H）

关于10H中断调用的介绍与使用详见第六章第六节图形部分。

5．异步通讯 I/O 中断调用(14H)，详见本章下节。

其余的中断调用不再一一详述，可参考IBM PC技术资料中ROM BIOS部分。

第四节 PC之间、PC与Z80之间的通信

一、串行通信的实现

IBM PC内装有通信适配器板，它使 PC 有能力与其它具有标准 RS 232 串行通信接口的计算机或设备进行通信。我们知道串行通信可以节省庞大的电缆开支和接口设备。有了串行通信协议和适当的接口设备，利用电话线就可以实现两设备间的对话。串行通信有同步、异步之分，这里主要介绍异步通信。

1．异步串行通信协议

为发送和接受一个信息字符所需的一切数据和控制信息都应该在单根数据线上移动，而且每次一位，这就需要有一个格式协议。图 5-16 给出了实现它的异步串行数据格式。

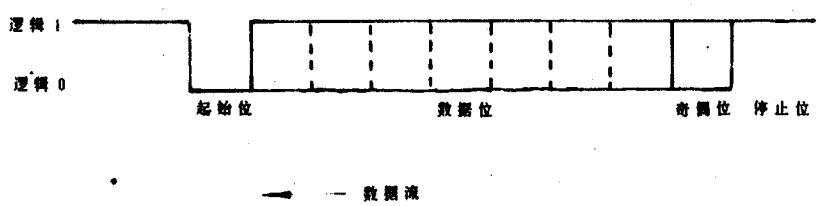


图 5 － 1 6 异步串行数据格式

图5-16中，每一位的宽度由数据传输速度确定，而传输速度是以每秒多少个二进制位来度量的。这个速度叫做波特率。也就是说，如果数据以每秒 300 位的速度向通信线发送，那么这个传输速度就是300波特。

当数据线上没有数据发送时，该线处于逻辑1状态（识别标记状态）。当要发送一个字符数据时，发送的第一位是起始位（逻辑0）。起始位后紧跟着数据位，这些数据位构成要发送的字符信息。数据位个数，可以规定为 5、6、7 或 8（ 但同一传输中，每个字符应包括相同的个数 ）。奇偶检验位用来识别某些类型的传输错。奇偶位之后可以跟着1、1.5 或 2个停止位。在异步协议中，，接收方以每个字符的起始位与发送方保持同步。

2.通用异步接收发送器8250简介

可以根据协议的要求编写程序完成异步串行通信中数据字符的接收和发送，但这比较复杂。有专用的器件叫作通用异步接收发送器(UART)，它可以完成大多数串行协议中的要

求。PC 中用的UART叫做8250异步通信组件。有了8250， 异步通信的实现就方便多了。

图5-17中给出了UART的功能框图。为清晰起见，图中省略了调制解调器和中断控制部分，从图中可以看出通信中的许多功能是由组件自动完成的。使用它的时候，开始要根据用户的要求和协议规定的波特率、数据位数、奇偶类型和停止位数用输出指令对它进行设置，一旦设置(初始化)完了，就可以用来发送和接受数据了。

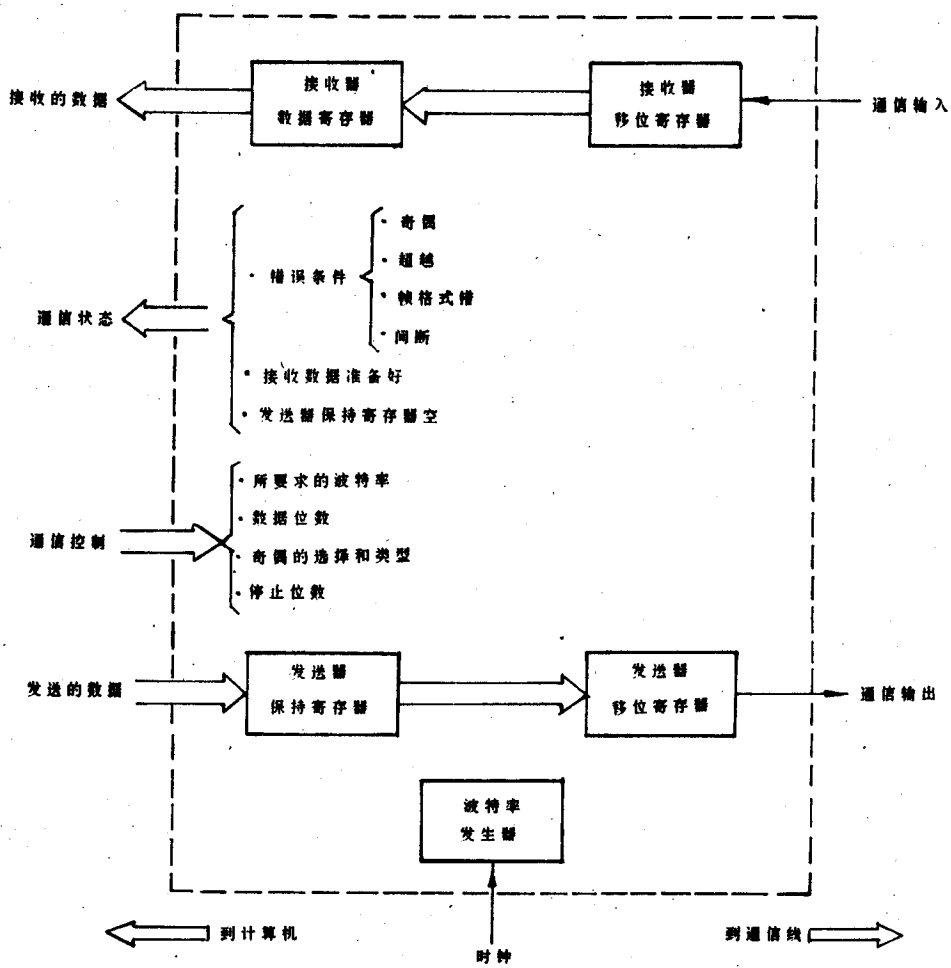


图5－17UART功能框图

当我们要发送一个数据字符时，可以利用 8086/8088 的输入指令检查一下发送器保持寄存器是否为空，若空，则用输出指令将一个数据字节 (并行)输出给 UART即可，UART会自动地依据初始化设置的要求，把相应的起始位、奇偶位和停止位加到发送器移位寄存器里的字符上，然后把这个二进制位串一位一位地发送到串行通信线上。

UART 从通信线上接收串行数据， 把它放在接收器移位寄存器中。在完成接收检测之后，UART自动摘出有效的数据位，把这个串行数据转换成并行数据字节送到接收器数据寄

存器中，并发出接收数据准备好信号。计算机接收这个信号后就可以用输入指令从这里读入一个数据字节了。UART在接受数据时，还可以进行错误检查。如果数据在传输过程中，由于噪声等原因引起某些数据位改变，可能会产生奇偶校检错。如果计算机尚未取走UART中的前一个数据字符，UART又将一个新的字符送入接收器数据寄存器中，那么就产生超越错误，如果UART没有接收到它所期望的仃止位，就产生一个帧格式错。有些异步通信协议考虑到传输线上的特殊条件，叫做间断条件。 在发送间断条件时， 通信线保持在逻计零状态。

我们可以利用查询的方式查看UART的状态或错误信息， 但 UART可以提供如下的中断信息，接收器数据寄存器中已有字符准备好等待取走；发送器保持寄存器空闲可以再送来字符；有接收错误。

这里我们给出了UART 8250的基本介绍，后面我们将讲述它的使用。

3．串行通信的实现

有了异步通信协议，也有了UART 8250 之后，如何将通信两地连接起来呢？如果两地距离较近，可以直接用标准的 RS 232 接口直接连接，如果距离较远，还应该利用附加设备，叫做调制解调器 (MODEM)。如图5-18所示。

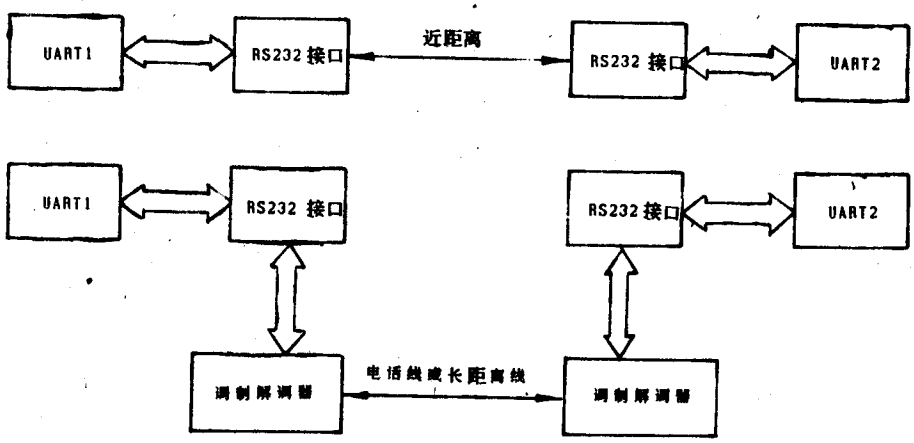


图 5 - 1 8 通讯设备连接示意图

RS 232 接口把 UART 的电信号转换成 EIA 的标准电平。逻辑 0（是空格条件）以+3 到 +15V 表示，逻辑 1（是识别标记条件）以 -3 到 -15V 表示。该接口也将输入电平返转成 UART 用的正确的二进制电信号。IBM PC 用的通信适配器板向外界接供了标准的 RS 232 接口，也就是说，UART 8250 和 RS 232 的接口已在 PC 内通信适配器板上装好，从适配器出来的插头座是标准的母 25 芯“D”型插座。 两台 PC 近距离通信时，可直接将它们相连。

连接方法如图 5-19所示。

图中引线端标的数字是“D”型插座的引脚号。

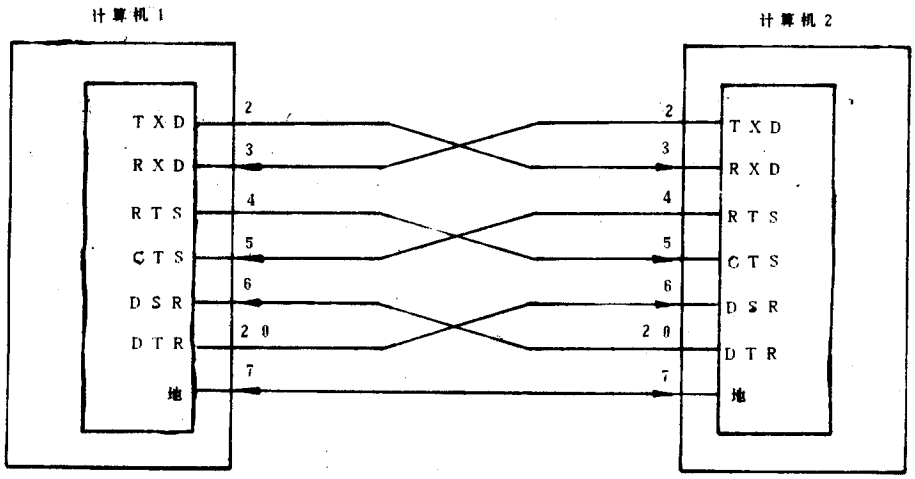


图 5 - 19 两个 RS 232 之间的连接

各引脚号的意义如表5-2所示。

表 5-2　　标准 RS 232 引脚

引脚号	方　向	功　能
2	输出	发送的数据
3	输入	接收的数据
4	输出	请求发送
5	输入	为发送清零
6	输入	数据设备准备好
7		信号地
8	输入	数据载波检测
20	输出	数据终端准备好
22	输入	振铃指示器

当两台计算机或设备远距离通信时，要借助调制解调器。 RS 232 输出的是电压，不能直接上电话线，调制解调器把代表逻辑 1 和 0 的电压信息转换成为能在电话线上传输的不同频率信号。电话线另一端的调制解调器把这不同频率的信号返转成为 RS 232 接收器能识别的电压1和0。用这种方法，逻辑1和0信号就可以被远距离传输。

一般的调制解调器能在普通电话线上以 300或1200波特的速率传输二进制数据。允许在两个计算机间同时完成发送和接收数据的调制解调器叫做全双工调制解调器。具有拨号远程计算机调制解调器， 并可在程序下建立连接能力者， 称为自动拨号调制解调器。能够检测出正在振铃的是哪台机器，并能以应答对方建立计算机连接的叫做自动应答调制解调器。

二、 UART 8250 的使用

1．8250的初始化

8250中有10个可访问的寄存器。如表 5-3 所示。但其中的五个寄存器，只需在主程序的开始处用输出指令将它们初始化即可。初始化以后，在使用 8250 时可以再不去管它们，这些寄存器是波特率因子寄存器（2个）、线控制寄存器、调制解调器控制寄存器和中断控制寄存器。

表 5-3　　8250　中可访问的寄存器

I/O端口地址	输入或输出	寄存器名字
3F8H	输出	发送器保持寄存器
3F8H	输入	接收器数据寄存器
3F8H	输出	波特率因子(低位字节)
3F9H	输出	波特率因子(高位字节)
3F9H	输出	中断控制寄存器
3FAH	输入	中断识别寄存器
3FBH	输出	线控制寄存器
3FCH	输出	调制解调器寄存器
3FDH	输入	线状态寄存器
3FEH	输入	调制解调器状态寄存器

(1) 波特率因子寄存器

初始化的第一个参数是波特率因子。由它决定传输的速率。表5-4给出波特率因子值与波特率的对应关系。

表 5-4　波特率因子值

波特率	波特率因子寄存器值	
	高位字节	低位字节
50	09H	00H
75	06H	00
110	04H	17
134.5	03H	59
150	03H	00
300	01H	80
600	00	C0
1200	00	60

表 5-4 （续）

1800	00	40
2000	00	3A
2400	00	30
3600	00	20
4800	00	18
7200	00	10
9600	00	0C

为了波特率因子的初始化，首先必须把线控制寄存器的高位置 1，即用 OUT指令给I/O地址3FBH 的高位置 1，然后，再把波特率因子值的高、低位字节分别送给 I/O 地址3F8H和3F9H(详见后面程序实例)。

(2)线控制寄存器

初始化的第二个参数是线控制寄存器，线控制寄存器各位的意义如表 5-5所示。

表 5-5 线控制寄存器各位功能

位 号	状 态	功 能
位1位0	00	字符长度为5个数据位
	01	字符长度为6个数据位
	10	字符长度为7个数据位
	11	字符长度为8个数据位
位2	0	1位停止位
	1	1.5位停止位(字符长度为5时)/2位停止位(字符长度为6.7.8时)
位3	0	不生成奇偶位
	1	生成奇偶位
位4	0	奇校验
	1	偶校验
位5	0	禁止
	1	若位3=1位4=0则奇偶位总为1，若位3=1，位4=1时，奇偶位总为0，若位3=0则无奇偶位。
位6	0	禁止
	1	串行输出数据强迫成空格条件(逻辑0)
位7	0	正常值
	1	寻址波特率因子寄存器

线控制寄存器确定在串行传输中要用的字符长度、停止位个数和奇偶校验类型。通常高3位置成0。位7置0表示不访问波特率因子寄存器。只有希望引起间断条件并往线路上输出时才把位 6 置 1。只有奇偶位是常数时才把位 5 置 1，其它位可根据要求选择设置。例如，若要传送具有7个数据位、一个停止位和奇偶校验位的数据，则应将1AH输出给 I/O 地址3FBH。

```
即:     MOV     DX，3FBH
        MOV     AL，1AH
        OUT     DX，AL
```

(3)调制解调器控制寄存器

初始化的第三个参数是调制解调器控制寄存器。它的位0等于1时，表示实际的数据终端准备好调制解调器信号；位1 等于1 表示实际的请求发送调制解调器信号；位 2(OUT1)是辅助的用户设计的输出，不用；位 3(OUT2) 是辅助的用户设计的输出，为将8250 产生的中断信号经系统总线到达 8259A 中断控制器的输入端 IRQ4 ，此位必须置 1；位4通常置为 0，若要置为 1，则 8250的串行输出被回送到 8250自己的串行输入中，利用这个特点，可以编写程序直接测试 8250 的工作是否正常，而不须任何附加设备；位 5.6.7强迫置 0。

(4)中断控制寄存器

初始化的最后一个参数是中断控制寄存器。如果不用中断，则把这个寄存器置 0；若要允许接收数据准备好中断，则把位 0置 1； 若要允许发送器保持寄存器空闲中断， 则将位 1置 1； 若允许接收字符错或间断条件中断，则将位 2 置 1；若允许改变调制解调器状态中断，则置位3为 1。位4.5.6.7强迫置0。

2.用 8250 通信

8250的初始化结束以后，就可以用它来实现串行通信。每当我们要发送一个数据字符时，若发送器保持寄存器是空的，我们可以把这个字符输出给发送器保持寄存器；如果接受器数据寄存器已接收好一个字符， 我们就可以从中读出一个字符， 将它送进计算机的 CPU ，然而关键问题在于：我们如何知道何时发送器保持寄存器为空？何时接受器数据寄存器已接收好(准备好)一个字符？解决这个问题的方式有两种，一种是查询方式，一种是中断方式。

(1)查询方式：

线状态寄存器能够告诉我们何时可以输入或输出一个字符数据。线状态寄存器的各位含义如下：

位0等于1 表示接收数据准备好；位1=1表示超越错误；位2=1表示奇偶错误；位3=1表示帧格式错，位4=1 表示间断检测；位5=1表示发送器保持寄存器空闲；位6=1表示发送器移位寄存器空闲；位7=1表示超时(对发送和接收中断调用)。

如果我们要发送一个字符， 那么必须先读线状态寄存器并检查位5，若为 0，则查询等待，直到变为 1，就可将发送字符输出到发送器保持寄存器。字符输出之后，线状态寄

存器的位5位将变成0直到UART将这个字符发送完毕且可以接收另一个字符为止。

当线状态寄存器的位0变成1时，表示8250已经接收了一个数据字符并将它放在了接收器数据寄存器中。程序发现这个条件时，就应该将这个字符取走(用IN指令)。该字符被取走以后，线状态寄存器的位0变成0直到下一个字符被8250接收为止，如果在8250 接收完下一个字符前，该字符未被取走，将产生一个超越错误。

线状态寄存器也可以用来检测任一接收数据错误或接收间断条件。如果相应位之一变成 1，则接收器数据寄存器中的内容就不是有效的数据字符。线状态寄存器内容一旦被读入，8250 中所有错误位便自动地复位成零，即使已产生的错误未被程序处理也是这样。这一点程序员应该注意。

调制解调器的状态也可以通过读入调制解调器状态寄存器的内容进行检测。这一点对涉及有关调制解调器控制信号的程序将是很有利的，调制解调器状态寄存器的各位含义如下：

位 0=1 表示 DELTA 为发送清零；位 1=1 表示 DELTA 数据设备准备好；位2=1 表示 DELTA 振铃指示器；位 3=1 表示 DELTA 数据载波检测；位4=1为发送清零；位 5=1 为数据设备准备好；位6=1表示振铃指示器；位7=1为数据载波检测。低四位允许我们检查从上一次读这个寄存器以来调制解调器状态输入有无任何变化。

(2)中断方式

前面已讲过，利用查询方式进行数据的输入或输出太浪费 CPU 的时间，8250 允许我们采用中断方式进行通信。

将调制解调器控制寄存器的 OUT2 位置1，则8250的中断输出可以发送给8259A中断控制器的输入端 IRQ4 上。为了使 8086/8088 CPU 能接受这个中断，必须初始化8259A中断屏蔽寄存器，必须编写中断服务程序，必须设置中断矢量表（单元 0CH*4），CPU也必须开中断。这些问题的实际处理见后面的举例。

在8250的初始化中，对中断控制寄存器的设置可以有选择地允许 4类中断中的某些或全部产生中断。当产生中断时，可以读入中断识别寄存器以判明是谁引起的中断。中断识别寄存器各位的含义如下：

位 0为 0表示有中断请求尚未处理，为1表示没有中断请求未处理；位2 位1 等于 00 表示调制解调器状态变化中断，等于01表示发送器保持寄存器空闲中断，等于10表示接收数据准备好中断，等于11表示接收字符错或接收间断条件中断。

如果在同一时刻有一个以上的中断发生，中断识别寄存器按优先级将它们排队。优先级按排如表 5-6所示。

如果只允许接收器数据准备好中断，那么中断发生时就不必要查看中断识别寄存器，也不必要判别是谁中断。中断服务程序应该从接收器数据寄存器中输入字符并检查线状态寄存器，看一看接收的字符是否有错（具体应用请看下面举例 ）。

表 5-6 8250中断优先级

中 断 类 型	中断识别寄存器位2位1	优先级	中断复位动作
接收字符错或间断条件	11	第一	读线状态寄存器
接收数据准备好	10	第二	读接收器数据寄存器
发送器准备好	01	第三	输出一字符到发送器保持寄存器
调制解调器状态变化	00	第四	读调制解调器状态寄存器

如果只允许接收器数据准备好中断，那么中断发生时就不必要查看中断识别寄存器，也不必要判别是谁中断。中断服务程序应该从接收器数据寄存器中输入字符并检查线状态寄存器，看一看接收的字符是否有错（具体应用请看下面举例）。

三、串行输入输出中断调用

前面，我们从8250的硬件角度讨论了实现通信的问题。ROM BIOS 提供了我们实现串行I/O的软件方便，而无需并弄清硬件细节。

如上所述，14H中断调用是异步通信I/O调用。它包括四个功能，功能号 0～3，如下所述：

1.初始化通信口（0号功能）

调用参数：AH=0 BX=0(若安装了一个以上通信适配器，BX 可为 0或 1)，AL=初始化参数。

AL 位1 和位0 等于 00、01、10、11 时分别表示字符长度为 5、6、7、8 位；AL 的位2 等于0 表示一位停止位，等于1 表示 1.5 位停止位（字符长度=5 位时）或 2位停止位（字符长度等于 6、7、8 位时）；AL 的位 4 位 3 等于 00、10 表示无奇偶校验，等于 01 为奇校验，等于 11 为偶校验；AL 的位 7、6、5 的 8 种状态 000、001、...110、111分别表示波特率 110、150、300、600、1200、2400、4800、9600。

返回参数：AX中为线状态寄存器和调制解调器状态寄存器中的内容。

注：这种功能调用不允许我们的通信带有中断。

2.发送数据字符(1号功能)

调用参数：AH=1，BX=0，AL=要发送的字符

返回参数：AH中为线状态寄存器内容。若字符没有发送，AH的位7置1。

3.等待接收字符(2号功能)

调用参数：AH=2，BX=0

返回参数：AL=接收的字符；AH=线状态寄存器内容，但位5、6为零。

4.读通信口状态(3号功能)

调用参数：AH=3，BX=0

返回参数：AH=线状态寄存器内容，AL=调制解调器状态寄存器的内容。

利用 ROM BIOS的14H中断调用编写串行通信程序是很简单方便的。通过上述介绍，我们不难看出，当对UART的控制达到必要的深度时，8250的一些硬件特点将没法利用。

四、PC 之间文件的发送和接收

要实现两台 IBM PC 之间的近距离通信，首先应将两台 IBM PC 的 RS 232 接口按图5-19所示连接起来，接下来就是编写相应的程序。作为应用实例，下面给出用查询方式进行半双工传送文件和用中断方式进行半双工接收并存贮文件的程序，这两个程序是综合性的，既包括了 DOS 系统功能调用，又包括了ROM BIOS 中断调用，既包括了文件的管理，又包括了8250、8259A 的使用，既采用了查询方式，又应用了中断方式。本例意在使读者从中得到有益的启发。

1.以查询方式发送文件。

下面给出以查询方式发送文件的程序。程序流程图如图5-20所示。

```
;            TRANSLATE FILES
STACK       SEGMENT  PARA STACK 'STACK'
            DB       256 DUP (0)
STACK       ENDS
DATA        SEGMENT  PARA PUBLIC 'DATA'
FCB         DB       37 DUP (0)
DTA         DB       0
COUNT       DB       0
INMSG       DB       'INPUT FILENAME: '
            DB       10,13,'$'
ERRMSG      DB       'FILE ACCESS ERROR !'
            DB       10,13,'$'
ERR         DB       'INPUT ERROR !$'
DATA        ENDS
;
CODE        SEGMENT  PARA PUBLIC 'CODE'
START       PROC     FAR
            ASSUME   CS: CODE,DS: DATA
            ASSUME   ES: DATA,SS: STACK
            PUSH     DS
            MOV      AX,0
            PUSH     AX
            MOV      AX,DATA
            MOV      ES,AX
```

```
        MOV     DS , AX
;  8250 初始化
        MOV     DX , 3FBH
        MOV     AL , 80H
        OUT     DX , AL
        MOV     DX , 3F8H
        MOV     AL , 0C0H
        OUT     DX , AL
        MOV     DX , 3F9H
        MOV     AL , 0
        OUT     DX , AL
        MOV     DX , 3FBH
        MOV     AL , 0AH
        OUT     DX , AL
        MOV     DX , 3FCH
        MOV     AL , 03H
        OUT     DX , AL
        MOV     DX , 3F9H
        MOV     AL , 0
        OUT     DX , AL
;  建立 DTA
        MOV     DX , OFFSET DTA
        MOV     AH , 1AH
        INT     21H
;  接收键盘输入文件名，并填写 FCB
PLE:    CALL    SETFCB
;  打开指定文件
        MOV     DX , OFFSET FCB
        MOV     AH , 0FH
        INT     21H
        CMP     AL , 0
        JNZ     PLE
        MOV     WORD PTR FCB+0CH , 0   ;  置当前块号和记录号为0 ,
        MOV     WORD PTR FCB+0EH , 1   ;  记录长度为 1。
        MOV     FCB+20H , 0
```

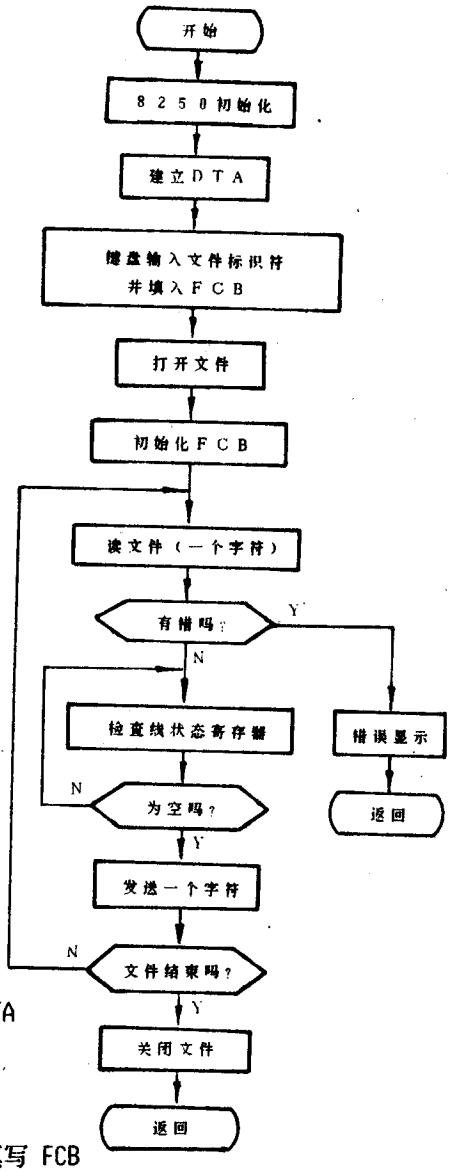


图 5 - 2 0 查询方式发送文件

```
AGAIN:  MOV     DX,OFFSET FCB       ;  顺序读文件（一个字符 ）
        MOV     AH,14H
        INT     21H
        CMP     AL,0
        JNZ     ERROR
SEND:   MOV     DX,3FDH    ; 查询线状态寄存器的位 5 ，若发送保持寄存器空，
        IN      AL,DX      ;  发送一个字符
        TEST    AL,20H
        JZ      SEND
        MOV     AL,DTA
        MOV     DX,3F8H
        OUT     DX,AL
        CMP     AL,AH          ;  若是文件结束字符（CTRL-Z） ，转 EOF
        JZ      EOF
        CALL    DISPCHAR            ;  显示该字符
        JMP     AGAIN
ERROR:  MOV     DX,OFFSET ERRMSG    ;  错误提示
        MOV     AH,9
        INT     21H
        RET
EOF:    MOV     DX,OFFSET FCB       ;  关闭文件
        MOV     AH,10H
        INT     21H
        RET
START   ENDP
SETFCB  PROC    NEAR                ;  SETFCB 过程
STA:    MOV     DX,OFFSET INMSG     ;  显示提示信息
        MOV     AH,9
        INT     21H
        MOV     AH,1                ;  接收键盘输入磁盘驱动器号
        INT     21H
        CMP     AL,'A'
        JZ      X1
        CMP     AL,'B'
        JZ      X1
```

```
        JMP     TEST1
X1:     SUB     AL , 40H
        MOV     FCB+00H , AL
        JMP     IN
TEST1:  CMP     AL , 'a'
        JZ      X2
        CMP     AL , 'b'
        JZ      X2
ERROR1: MOV     DX , OFFSET ERR
        MOV     AH , 9
        INT     21H
        JMP     STA
X2:     SUB     AL , 60H
        MOV     FCB+00H , AL
IN:     MOV     AH , 1
        INT     21H
        CMP     AL , ': '
        JNZ     ERROR1
        MOV     COUNT , 1
        MOV     DI , OFFSET FCB+1
LOP:    MOV     AH , 1                  ;  接收键入文件名
        INT     21H
        CMP     AL , '.'
        JZ      SET1
        CMP     COUNT , 9
        JZ      ERROR1
        STOSB
        INC     COUNT
        JMP     LOP
SET1:   MOV     AH , 1                  ;  接收键入文件扩展名
        INT     21H
        MOV     FCB+9 , AL
        MOV     AH , 1
        INT     21H
        MOV     FCB+10 , AL
```

```
            MOV     AH , 1
            INT     21H
            MOV     FCB+11 , AL
            MOV     AL , 0DH
            CALL    DISPCHAR
            MOV     AL , 0AH
            CALL    DISPCHAR
            RET
SETFCB      ENDP
;  显示一个字符子程序
DISPCHAR    PROC    NEAR
            PUSH    BX
            MOV     BX , 0
            MOV     AH , 14
            INT     10H
            POP     BX
            RET
DISPCHAR    ENDP
CODE        ENDS
            END START
```

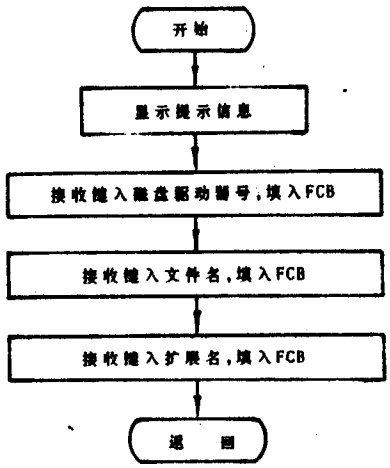


图5－21 SETFCB过程流程图

程序的第一部分是8250初始化。其中设置波特率因子高、低位寄存器为 600波特；设置线控制寄存器为 7位数据位、1位停止位、1位起始位和奇偶校验位；设置调制解调寄存器为非循回式，即发送出的信息不回送；设置中断控制寄存器为不允许中断。程序的第二部分是建立一个字节的磁盘传输区（DTA)。第三部分是调用一个名叫SETFCB的过程。SETFCB过程的流程图如图 5-21 所示。这个过程接收用户从键盘上输入的文件标识符，包括磁盘驱动器号、文件名和扩展名三项。用户输入的这些信息就指定了要从通信线上发送出去的文件。SETFCB 过程一边接收用户键入信息一边将这些信息送入 FCB 中(即设置 FCB)。

接下来，程序的第四部分就是打开用户指定的文件，顺序读一个记录（这里一个记录为一个字节)，若读文件有错，就显示出错误信息，然后返回。若读文件无错，就进入程序的第五部分，即按查询方式发送字符。首先检查线寄存器，若为空，即可将刚从文件中读入的定符发送出去，若不空，就查询等待。这样重复发送一个个字符，直到迁到文件结束标志 CTRL-Z（即 ASCII 码 1AH)，就关闭文件，返回。

SETFCB过程中允许键入的磁盘驱动器号可为大写字母A、B或小字母 a、b ，若想用 C 驱动器，可自己修改一下程序，文件名最多允许8个字符。护展名为3个字符。

2.以中断方式接收和存贮文件

下面给出的第二个程序是以中断方式接收和存贮文件。程序流程图如图5-22所示。

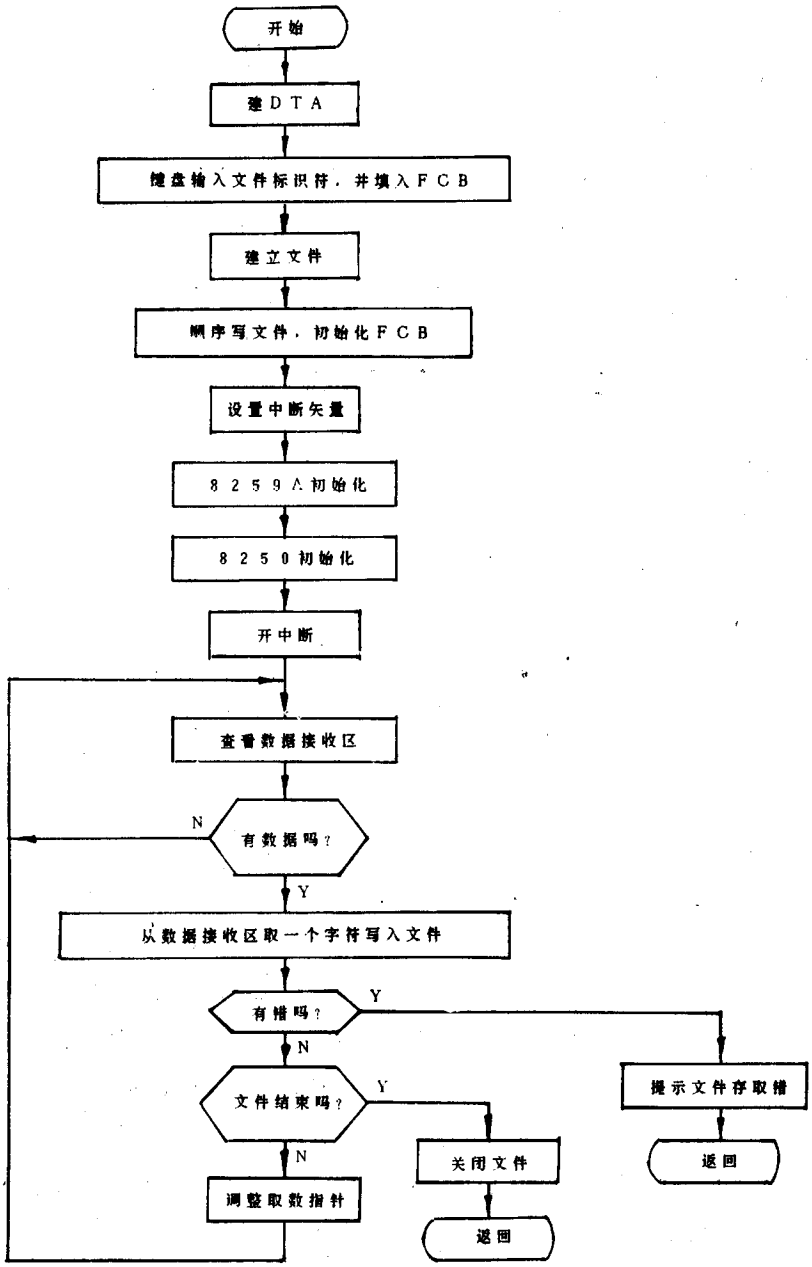


图 5 - 2 2 中断方式接收和存储文件

以中断方式接收和存贮文件的程序如下：

```
;              RECEIVE FILES
STACK      SEGMENT  PARA STACK 'STACK'
           DB       256 DUP (0)
```

```
STACK     ENDS
DATA      SEGMENT  PARA PUBLIC 'DATA'
FCB       DB       37 DUP (0)
DTA       DB       0
COUNT     DB       0
INMSG     DB       'PLEASE INPUT FILENAME:  ',10,13,'$'
ERRMSG    DB       'FILE ACCESS ERROR !',10,13,'$'
BUFFER    DB       100 DUP (0)
BUFFER1   DW       0
BUFFER2   DW       0
DATA      ENDS
CODE      SEGMENT   PARA  PUBLIC  'CODE '
START     PROC     FAR
          ASSUME   CS: CODE,DS: DATA,ES: DATA
          PUSH     DS
          MOV      AX,0
          PUSH     AX
          MOV      AX,DATA
          MOV      ES,AX
          MOV      DS,AX
          MOV      DX,OFFSET DTA       ;  建立 DTA
          MOV      AH,1AH
          INT      21H
PLE:      CALL     SETFCB              ;  键入文件标示符并填入 FCB
          MOV      DX,OFFSET FCB       ;  建立文件
          MOV      AH,16H
          INT      21H
          CMP      AL,0
          JNZ      PLE
          MOV      WORD PTR FCB+0CH,0        ;  顺序写文件初始化
          MOV      WORD PTR FCB+0EH,1
          MOV      FCB+20H,0
          CLI
          MOV      AX,0                      ;  在中断矢量表中设中断矢量
          MOV      ES,AX
```

```
        MOV     DI , 0CH*4
        MOV     AX , OFFSET RECEIVE
        CLD
        STOSW
        MOV     AX , CS
        STOSW
        MOV     AL , 2CH            ; 8259A 初始化
        OUT     21H , AL
        MOV     DX , 3FBH           ; 8250 初始化
        MOV     AL , 80H
        OUT     DX , AL
        MOV     DX , 3F8H
        MOV     AL , 0C0H
        OUT     DX , AL
        MOV     DX , 3F9H
        MOV     AL , 0
        OUT     DX , AL
        MOV     DX , 3FBH
        MOV     AL , 0AH
        OUT     DX , AL
        MOV     DX , 3FCH
        MOV     AL , 0BH
        OUT     DX , AL
        MOV     DX , 3F9H
        MOV     AL , 01H
        OUT     DX , AL
        STI
WRITE:  MOV     BX , BUFFER1        ; 查看数据接收区 ( 圆形队列 ),
        CMP     BX , BUFFER2        ; 若有字符 , 取入 DTA
        JZ      WRITE
        MOV     AL , [BUFFER+BX]
        MOV     DTA , AL
        PUSH    AX
        MOV     DX , OFFSET FCB     ; 顺序写文件
        MOV     AH , 15H
```

```
        INT     21H
        CMP     AL,0
        JNZ     ERROR
        POP     AX
        CMP     AL,1AH          ;  若为文件结束符 ，转 EOF
        JZ      EOF
        CALL    DISPCHAR
        MOV     BX,BUFFER1      ;  调整取数指针
        CMP     BX,99
        JNZ     ADN
        MOV     BUFFER1,0
        JMP     WRITE
ADN:    INC     BUFFER1         ;  继续写文件
        JMP     WRITE
ERROR:  MOV     DX,OFFSET ERRMSG
        MOV     AH,9
        INT     21H
        RET
EOF:    MOV     DX,OFFSET FCB   ;  关闭文件 ，返回
        MOV     AH,10H
        INT     21H
        RET
RECEIVE PROC    NEAR            ;  接收数据中断服务程序
        PUSH    AX
        PUSH    BX
        PUSH    DX
        PUSH    DS
        MOV     DX,3FDH ;  检查接收的数据是否有错 ，若有错 ，转 ERR
        IN      AL,DX
        TEST    AL,1EH
        JNZ     ERR
        MOV     DX,3F8H         ;  读入数据 ，存入数据接收区
        IN      AL,DX
        AND     AL,7FH
        MOV     BX,BUFFER2
```

```
                MOV     [BUFFER+BX], AL
                CMP     BX, 99              ;  调整存数指针
                JZ      SET0
                INC     BUFFER2
                JMP     RTN1
SET0:           MOV     BUFFER2, 0
RTN1:           MOV     AL, 20H             ;  发中断结束命令给 8259A
                OUT     20H, AL
                POP     DS
                POP     DX
                POP     BX
                POP     AX
                STI
                IRET
ERR:            MOV     AL, '?'
                MOV     BX, 0
                MOV     AH, 14
                INT     10H
                JMP     RTN1
RECEIVE         ENDP
SETFCB          PROC    NEAR                ;  接收键入文件标识符（同前）。
                .
                .
SETFCB          ENDP
DISPCHAR        PROC    NEAR                ;  显示字符（同前）。
                .
                .
DISPCHAR        ENDP
START           ENDP
CODE            ENDS
                END START
```

程序中需要首先说明的是：在数据区自 BUFFER单元开始设置了100个字节的缓冲区，用它作为数据接收区，以便存放从通信线上接收来的数据。设立指针 BUFFER2 作为存数指针，即从 8250 读入的（也就是从数据线上接收的）数据存入数据接收区的指针，又设 BUFFER1 作为取数指针，即从数据接收区取数据写入文件时用的指针。设立了这两个指针后，取数和存数可非同步进行。当两指针不相等时，说明数据接收区内有数据，可直接取

出写入文件，当两指针相等时，表明数据区中已无数据。这个数据接收区又称圆形队列。

本程序中建立 DTA 和接收键盘输入文件标识符并填入 FCB 两部分与上述程序相同，不同的是这里键入的文件标识符是用来指定要在磁盘上建立的文件，此文件中存放从通信线上接收来的数据。

因为要用中断方式，所以 8250的初始化部分，是把01H送入它的中断控制器，即设置为允许接收数据准备好中断。程序中也包括了对8259A中断控制器的初始化设置(参见本章第二节)。

这里采用中断方式接收数据，因而编写了中断服务程序 RECEIVE（详见下述 ）。同时应设置中断失量表中类型号为 OCH 的中断失量， 即把中断服务程序的入口地址的段内偏移地址和段地址存入中断失量表中自 OCH*4 单元开始的四个连续单元内。 然后不要忘记安排开中断指令。

机器执行完初始化程序，并开中断以后，只要8250从通信线上接收了数据字符并将它存入接收器数据寄存器以后， 就会发出接收数据准备好中断申请信号， 这个信号由8259A转达给 CPU，CPU响应中断后，自动转入 RECEIVE中断服务程序去执行，RECEIVE程序的流程图如图5-23所示。

RECEIVE 程序的开始处，首先检查8250的线状态寄存器，看接收数据是否有错。若无错，则将其存入数据接收区，并调整存数指针，然后据 8259A的要求，发中断结束命令给8259A，再返回。直到 8250从通信线上接收到下一个字符数据，再发中断申请信号，重复进行。如果接收的数据有错，本程序的安排是显示“？”，不把接收的有错的数据写入数据接收区， 直接发中断结束命令给 8259A 后，从中断返回。这种安排会丢失有错的数据，用户可根据不同要求安排不同的处理方法。

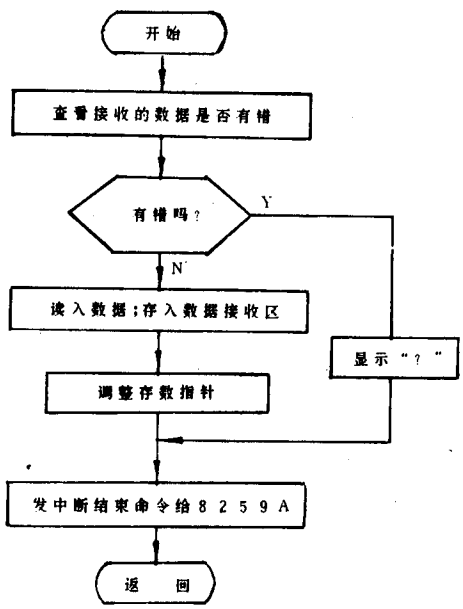


图 5 - 2 3 接收数据中断服务程序流程图

作为主程序来说，初始化工作完了并开中断以后，主程序的任务就是看看数据接收区中是否已存入数据，若有就把它写入文件中，若没有，就循环等待。直至接收到通信线上送来的文件结束标志（ 1AH）时，关闭文件并返回。如果在顺序写文件中出现错误，则提示文件存取错，并返回。关于文件的存取管理请翻阅本书第四章第七节。

五． PC 与 Z80 之间的文件发送和接收

8 位机，特别是 Z80 系列机， 如 TP-801 单板机及其兼容机在我国被广泛应用，因

此这里讨论 IBM PC 与 TP-801 单板机之间的通信问题。　因为 TP-801 本身并不带有串行通信设备，因此我们选择了Z80 SIO 串行接口芯片以便完成 UART 的功能，又选择了电平转换芯片(1488和1489)将SIO输出的TTL电平转换或 RS 232 的标准电平。最后用 RS 232 标准接口就可以与 IBM PC 的串行通信接口直接相连了。图5-24给出了实现这些连接的线路图。

关于 Z80 SIO 的详细情况这里不作介绍，读者可参考有关 Z80 的书籍。 IBM PC 与 Z80 系列机之间的通信可以采用查询方式，也可以采用中断方式。上面刚刚讲述的在 IBM PC 上运行的发送和接收文件的程序在这里完全有效，只要与下述的 TP-801中的程序配合使用即可。不再重述。下面给出在TP-801单板机上以各种方式发送和接受文件的程序。其它 Z80 系列机与此类同。程序中各指令的含义可参见本节后的注释。

1. TP-801单板机以查询方式发送文件

下述程序将TP-801单板机的内存中自2200H单元开始连续存放的以 1AH 为结束标志的文件串行发送到通信线上。采用的是查询方式。其中7位数据位，1位停止位、奇校位、600波特。

```
        源程序                          指令代码
        ORG   2000h
;  CTC0 初始化
START:  LD    A,      05H       ;   3E  05
        OUT   (84H),  A         ;   D3  84
        LD    A,      07H       ;   3E  0D
        OUT   (84H),  A         ;   D3  84
;  ZIO 初始化
        LD    A,      18H       ;   3E  18
        OUT   (97H),  A         ;   D3  97
        LD    A,      01H       ;   3E  01
        OUT   (97H),  A         ;   D3  97
        LD    A,      00H       ;   3E  00
        OUT   (97H),  A         ;   D3  97
        LD    A,      04H       ;   3E  04
        OUT   (97H),  A         ;   D3  97
        LD    A,      45H       ;   3E  45
        OUT   (97H),  A         ;   D3  97
        LD    A,      03H       ;   3E  03
        OUT   (97H),  A         ;   D3  97
        LD    A,      41H       ;   3E  41 (61)
```

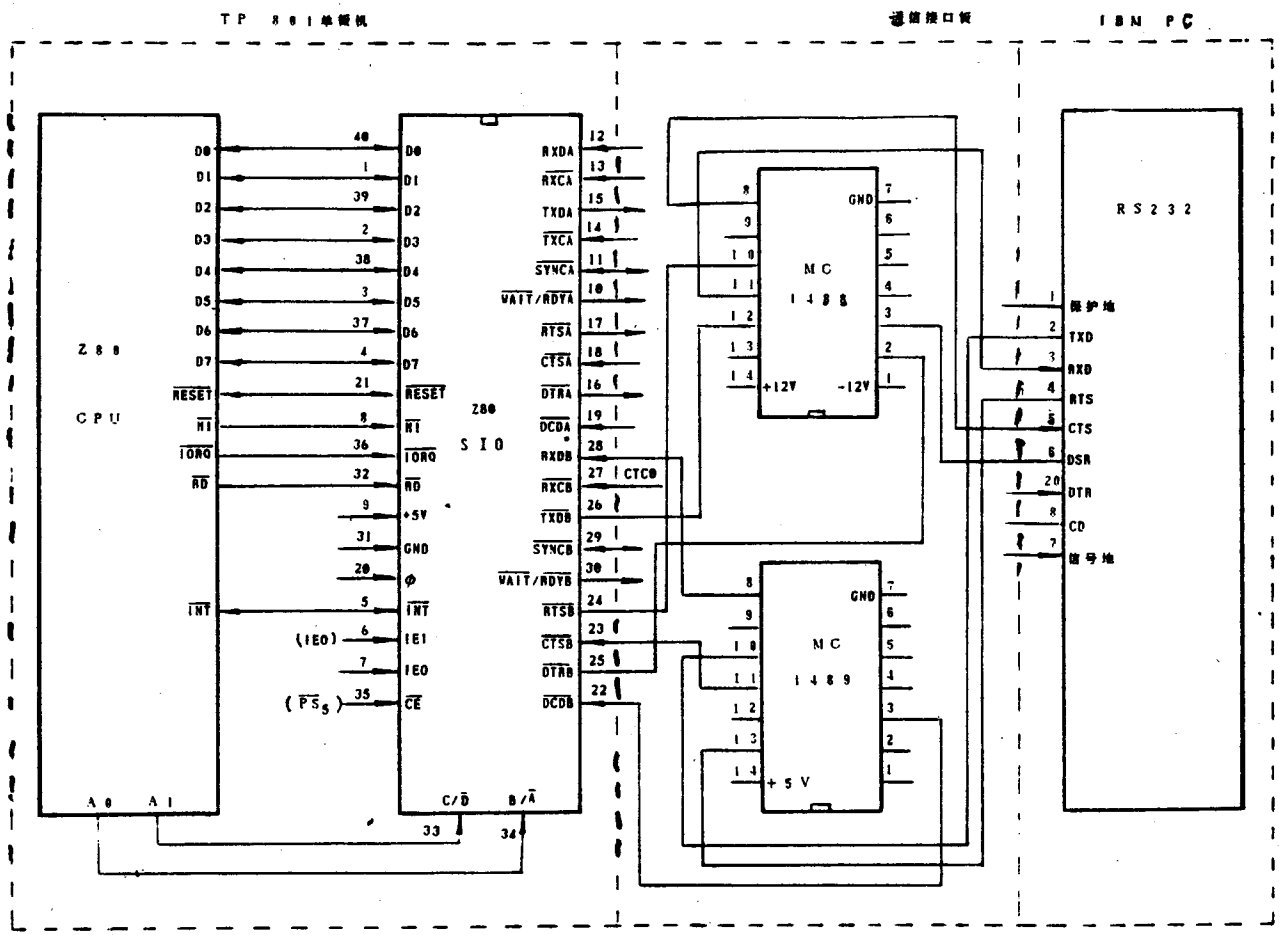


图 5-24 PC与单板机之间通信接线图

```
        OUT   (97H),  A          ;  D3  97
        LD    A,      05H        ;  3E  05
        OUT   (97H),  A          ;  D3  97
        LD    A,      28H        ;  3E  28 (AA)
        OUT   (97H),  A          ;  D3  97
    ;  设置数据区指针 HL
        LD    HL,     2200H      ;  21  00  22
        DI                       ;  FB
    ;  查询发送寄存器是否为空
TEST:   IN    A,      (97H)      ;  DB  97
        AND   04H                ;  E6  04
        JP    Z,      TEST       ;  CA  30  20
        LD    A,      (HL)       ;  7E
    ;  发送数据
        OUT   (95H),  A          ;  D3  95
        INC   HL                 ;  23
        CP    A,      1AH        ;  FE  1A
        JP    NZ,     TEST       ;  C2  30  20
        HALT                     ;  76
```

2. TP-801单板机以查询方式接收文件

下述程序使TP-801单板机接收通信线上送来的字符数据，存入从内存 2200H 单元开始的数据区，直到接收到1AH字符为止。

```
            源程序                    指令代码
        ORG   2000h
    ;  CTC0 初始化
START:  LD    A,      05H        ;  3E  05
        OUT   (84H),  A          ;  D3  84
        LD    A,      07H        ;  3E  0D
        OUT   (84H),  A          ;  D3  84
    ;  SIO 初始化
        LD    A,      18H        ;  3E  18
        OUT   (97H),  A          ;  D3  97
        LD    A,      01H        ;  3E  01
        OUT   (97H),  A          ;  D3  97
        LD    A,      00H        ;  3E  00
```

```
        OUT   (97H),  A          ;  D3  97
        LD    A,      04H        ;  3E  04
        OUT   (97H),  A          ;  D3  97
        LD    A,      45H        ;  3E  45
        OUT   (97H),  A          ;  D3  97
        LD    A,      03H        ;  3E  03
        OUT   (97H),  A          ;  D3  97
        LD    A,      41H        ;  3E  41 (61)
        OUT   (97H),  A          ;  D3  97
        LD    A,      05H        ;  3E  05
        OUT   (97H),  A          ;  D3  97
        LD    A,      28H        ;  3E  28 (AA)
        OUT   (97H),  A          ;  D3  97
        LD    HL,     2200H      ;  21  00  22
        DI                       ;  FB
;  查询接收数据是否准备好
TEST:   IN    A,      (97H)      ;  DB  97
        AND   01H                ;  E6  01
        JP    Z,      TEST       ;  CA  30  20
;  输入数据
        IN    A,      (95H)      ;  DB  95
        AND   7FH                ;  E6  7F
        LD    (HL),   A          ;  77
        INC   HL                 ;  23
        CP    A,      1AH        ;  FE  1A
        JP    NZ,     TEST       ;  C2  30  20
        HALT                     ;  76
```

3. TP-801 单板机以中断方式接收文件

下述程序是TP-801单板机以中断方式接收 PC 送来的数据，存入内存自 2200H开始的单元，直至接收到 1AH 结束符为止。

```
        源程序                          指令代码
        ORG   2000H
START : LD    SP,     2F00H      ;  31  00  2F
;  CTC0 初始化
        LD    A,      05H        ;  3E  05
```

```
        OUT   (84H),  A          ;  D3  84
        LD    A,      0DH        ;  3E  0D
        OUT   (84H),  A          ;  D3  84
;  SIO 初始化
        LD    A,      18H        ;  3E  18
        OUT   (97H),  A          ;  D3  97
        LD    A,      01H        ;  3E  01
        OUT   (97H),  A          ;  D3  97
        LD    A,      18H        ;  3E  18
        OUT   (97H),  A          ;  D3  97
        LD    A,      04H        ;  3E  04
        OUT   (97H),  A          ;  D3  97
        LD    A,      45H        ;  3E  45
        OUT   (97H),  A          ;  D3  97
        LD    A,      03H        ;  3E  03
        OUT   (97H),  A          ;  D3  97
        LD    A,      61H        ;  3E  61 (41)
        OUT   (97H),  A          ;  D3  97
        LD    A,      05H        ;  3E  05
        OUT   (97H),  A          ;  D3  97
        LD    A,      0AAH       ;  3E  AA (28)
        OUT   (97H),  A          ;  D3  97
        LD    A,      02H        ;  3E  02
        OUT   (97H),  A          ;  D3  97
        LD    A,      00H        ;  3E  00
        OUT   (97H),  A          ;  D3  97
        LD    HL,     2200H      ;  21  00  22
        LD    A,      21H        ;  3E  21
        LD    I,      A          ;  ED  47
        IM    2                  ;  ED  5E
;  开中断
        EI                       ;  FB
LOOP:   HALT                     ;  76
        JP    LOOP               ;  C3  41  20
END :   HALT                     ;  76
```

```
;  中断矢量表
2100:    10
2101:    21

;  接收数据中断服务程序
        ORG   2110H
        IN    A,      (95H)        ;   DB  95
        AND   7FH                  ;   E6  7F
        LD    (HL),   A            ;   77
        INC   HL                   ;   23
        CP    A,      1AH          ;   FE  1A
        JP    Z,      END          ;   CA  45  20
        EI                         ;   FB
        RETI                       ;   ED  4D
```

4. TP-801 单板机以双中断全双工方式传送字符。

SIO的控制字为：

WR0=18H，　WR1=1EH，　WR2=00H，

WR3=41H(61H)，　WR4=45H，

WR5=28H(0AAH)

主程序流程图如图 5-25 所示，以中断方式接收字符的服务程序的流程图如图5-26所示，以中断方式发送字符的服务程序的流程图如图 5-27 所示。

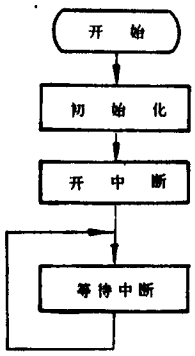


图 5 － 2 5 主程序流程图

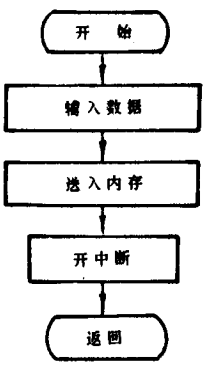


图 5 － 2 6 中断接受服务程序流程图

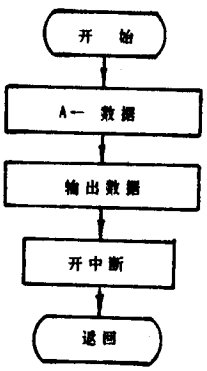


图 5 － 2 7 中断发送服务程序流程图

TP-801 单板机以双中断全双工方式传送字符的程序如下：

源程序				指令代码		
	ORG	2000H				
START:	LD	SP ,	2F00H ;	31	00	2F
; CTC0	初始化					
	LD	A ,	05H ;	3E	05	
	OUT	(84H) ,	A ;	D3	84	
	LD	A ,	0DH ;	3E	0D	
	OUT	(84H) ,	A ;	D3	84	
; SIO	初始化					
	LD	A ,	18H ;	3E	18	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	01H ;	3E	01	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	1EH ;	3E	1E	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	04H ;	3E	04	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	45H ;	3E	45	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	03H ;	3E	03	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	61H ;	3E	61	(41)
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	05H ;	3E	05	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	0AAH ;	3E	AA	(28)
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	02H ;	3E	02	
	OUT	(97H) ,	A ;	D3	97	
	LD	A ,	00H ;	3E	00	
	OUT	(97H) ,	A ;	D3	97	
	LD	B ,	64H ;	06	64	(0A)
	LD	HL ,	2400H ;	21	00	24
	LD	A ,	21H ;	3E	21	
	LD	I ,	A ;	ED	47	

```
;  Z80  中断方式
        IM      2               ;   ED   5E
        OR      A               ;   B7
        LD      A,      01H     ;   3E   01
        LD      D,      A       ;   57
        OUT     (95H),  A       ;   D3   95
        EI                      ;   FB
LOOP :  HALT                    ;   76
        JP      LOOP            ;   C3   49   20
;   中断矢量表
2100:   20
2101:   22
2104:   30
2105:   22
;   中断发送服务程序
        ORG     2220H
        INC     D               ;   14
        LD      A,      D       ;   7A
        OUT     (95H),  A       ;   D3   95
        EI                      ;   FB
        RETI                    ;   ED   4D
;   中断接收服务程序
        ORG     2230H
        IN      A,      (95H)   ;   DB   95
        AND     7FH             ;   E6   7F
        LD      (HL),   A       ;   77
        INC     HL              ;   23
        DEC     B               ;   05
        EI                      ;   FB
        RETI                    ;   ED   4D
```

注释 ：

Z80 CPU 中有 22 个寄存器，本节中用到的有 8 位寄存器 A、B、D、I，寄存器对 HL（16位）。

本节中用到的指令有 ：

指令助记符		含　义
LD	A,n	; 将 8 位立即数 n 送入 A 寄存器
LD	HL,nn	; 将 16 位立即数送入 HL 寄存器对
LD	SP,nn	; 将 16 位立即数送入堆栈指针 SP
LD	I,A	; 寄存器 A 的内容送入中断矢量寄存器 I
LD	D,A	; 寄存器 A 的内容送入 D 寄存器
LD	A,D	; 寄存器 D 的内容送入 A 寄存器
LD	(HL),A	; 寄存器 A 的内容送入 HL 所指的内存单元
IN	A,(n)	; I/O 端口 n 的内容读入 A 寄存器
OUT	(n),A	; A 寄存器的内容输出到 I/O 端口 n
INC	D	; D 寄存器的内容加 1
INC	HL	; HL 寄存器对的内容加 1
DEC	B	; 寄存器 B 的内容减 1
AND	A,n	; 寄存器 A 的内容 "与" 立即数 n 的结果送 A
OR	A	; 寄存器 A 的内容自身 "或" 的结果送 A
CP	A,n	; 寄存器 A 的内容减立即数 n,结果不送回
JP	标号	; 无条件转移到标号
JP	Z,标号	; 若结果为 0，转移到标号
JP	NZ,标号	; 若结果不为 0，转移到标号
EI		; 开中断
DI		; 关中断
IM	2	; 中断方式 2
RETI		; 从中断返回
HALT		; 暂停

习题五

1.编程序从8位数据端口1FH的第0位(D0)连续输出矩形波。波形周期为20ms。

2.如果你在系统中新增加了一个中断源，在软件方面，应该做哪几项工作才能使此中断源投入运行，(可举例说明)

3.编程序，利用1AH中断调用，产生5秒延时。

4.编程序，利用21H和1AH中断调用，在屏幕上显示 1～9之间的随机数。

5.编程序实现两台 PC 间的对话，即在一台 PC 的键盘上输入的任何信息，在另一台 PC 的 CRT 上显示出来。

第六章　程序设计的一些技法

在第四章中我们已经介绍了程序设计的基本方法。此后还要掌握程序设计的一些技法，现将其中主要的几方面加以介绍。

第一节　字符处理

前面已讲过最常用的字符编码是ASCII码，它在微型计算机中的应用尤为普遍。ASCII码数据的处理包括内容很多，例如：计算字符串的长度，查找某一个字符，改换字符，判断字符串是否一致等等。基本上分为字符的检查、搜索、删除、插入、改换、转换等几类。下面我们将结合串操作指令的介绍及应用分别加以叙述。

8086/8088 中有一些一字节指令，它们能完成各种基本的字节串或字串（即字节或字的序列）的操作。任一个这样的基本操作，能在指令的前面用一个重复操作前缀使它们重复地操作。

所有的基本的串操作指令，用寄存器 SI 寻址源操作数，且假定是在现行的数据段区域中（段地址在段寄存器 DS 中）；用寄存器 DI 寻址目的操作数，且假定是在现行的附加段区域中（段地址在段寄存器 ES 中）。这两个地址指针在每一个操作以后要自动修改，但按增量还是按减量修改，取决于标志位DF。若标志 DF＝0，则在每次操作后 SI 和 DI 增量（字节操作则加 1，而字操作则加 2）；若标志 DF＝1，则每次操作后，SI 和 DI 减量。

任何一个串操作指令，可以在前面加上一个重复操作前缀，于是指令就重复执行，直至在寄存器 CX 中的操作次数满足要求为止。

重复操作是否完成的检测，是在操作以前进行的。所以若初始化使操作次数为 0，它不会引起重复操作。

若基本操作是一个影响ZF标志的操作，在重复操作前缀字节中也可以规定与标志ZF相比较的值。在基本操作执行以后，ZF 标志与指定的值不等，则重复终结。

在重复的基本操作执行期间，操作数指针（SI 和 DI）和操作数寄存器在每一次重复后修改。然而指令指针将保留重复前缀字节的偏移地址。因此，若一个重复操作指令，被外部源中断，则在中断返回以后，可以恢复重复操作指令。

应该避免串操作指令的重复前缀与别的两种前缀同时使用。

8086/8088 有五种基本的串操作指令。

1. MOVS OPRD1, OPRD2　(Move String)

把由SI 作为指针的源串中的一个字节或字，传送至由 DI作为指针的目的串，且相应地修改指针，以指向串中的下一个元素。OPRD1和OPRD2分别为源串和目的串的符号地址。

在前面介绍数据传送指令 MOV时，我们说过 MOV指令不能实现内存单元之间的数据传送。而这种传送要求又是经常会遇到的，这时就要以某一通用寄存器作为桥梁，要实现重复传送，还必须修改地址。

而 MOVS 指令就是为了实现这样的传送而设置的，一条指令，除了直接完成数据从源地址传送至目的地址以外，还自动完成修改地址指针。但 MOVS 指令中规定源操作数必须用 SI 寻址，目的操作数必须用 DI 寻址。

前面的传送 100个操作数的例子，可以改为：

```
        MOV     SI，OFFSET SOURCE
        MOV     DI，OFFSET DEST
        MOV     CX，100
        CLD
AGAIN:  MOVS    DEST，SOURCE
        DEC     CX
        JNZ     AGAIN
```

若采用重复前缀，则可以用一条指令完成整个数据块的传送，但要用重复前缀，数据长度必须放在寄存器 CX 中。上述程序可简化为：

```
        MOV     SI，OFFSET SOURCE
        MOV     DI，OFFSET DEST
        MOV     CX，100
        CLD
   REP  MOVS    DEST，SOURCE
```

MOVSB 和 MOVSW 指令分别指明为字节和字串传送指令，不带操作数，其余同 MOVS。

此指令对标志位无影响。

2. CMPS OPRD1，OPRD2 (Compare String)

从由 SI 作为指针的源串中减去由 DI 作为指针的目的串（字或字节），减的结果反映到标志位上，而不送至任何一操作数。同时相应地修改源和目的串指针，使指向串中的下一个元素。OPRD1 和 OPRD2 分别为源串和目的串的符号地址。标志位 A、 C、 O、P、S 和 Z反映了目的串元素和源串元素之间的关系。

这个指令可以用来检查两个串是否相同。通常在此指令之后，应有一条件转移指令。

CMPSB 和 CMPSW 分别指明是字节和字串比较指令，不带操作数，其余同 CMPS。

下面是一个利用 CPMS 指令进行两个字符串比较的程序例子：

```
MY__DATA    SEGMENT
STRING1     DB      'THIS IS A STRING1'
STRING2     DB      'THIS IS A STRING2'
```

```
COUNT       EQU     $-STRING2
RESULT      DB      ?
MY__DATA    ENDS
STACK       SEGMENT PARA STACK 'STACK'
STAPN       DB      100  DUP(?)
STACK       ENDS
COSEG       SEGMENT
            ASSUME  CS:  COSEG , DS:  MY__DATA , ES:  MY__DATA , SS:  STACK
FH          PROC    FAR
START:      PUSH    DS
            MOV     AX , 0
            PUSH    AX
            MOV     AX , MY_DATA
            MOV     DS , AX
            MOV     ES , AX
            MOV     SI , OFFSET  STRING1
            MOV     DI , OFFSET  STRING2
            MOV     CX , COUNT
            CLD                         ; 地址增量
            REPZ    CMPSB
            JNZ     UNMAT
            MOV     AL , 0
            JMP     OUTPT
UNMAT:      MOV     AL , OFFH
OUTPT:      MOV     RESULT , AL         ; 若两串相同，0→RESULT单元
            RET                         ; 若两串不相同，OFFH→RESULT单元
FH          ENDP
COSEG       ENDS
            END     START
```

因为，若两个字符串的长度不同，则它们必然不相等，因此可在程序开始处先比较两字符串的长度，若相等，再逐个字符进行比较。

若 CMPS 指令加上前缀 REPE 或 REPZ ，则操作可解释为：＂当串未结尾（CX≠0 ）且串是相等（ZF 标志为 1 ）时继续比较＂。

若 CMPS 指令加以前缀 REPNE或 REPNZ，操作解释为：＂当串未结尾（CX≠0 ）且串不相等（ZF 标志为0 ）继续比较＂。

3. SCAS OPRD (Scan String)

从 AL (字节操作) 或 AX (字操作) 的内容中减去由 DI 作为指针的目的串元素，结果反映在标志位上，但并不改变目的串元素以及累加器中的值。SCAS 也修改 DI，使指向下个元素，在标志位 A、C、O、P、S和 Z中反映了在 AL / AX中的搜索值与元素之间的关系。OPRD 是目的串的符号地指。

SCASB 和 SCASW 分别指明是字节串和字串搜索指令，不带操作数，其余同 SCAS。

利用 SCAS 指令可以进行搜索，下面我们举一个例子。把要搜索的关键字放在 AL (字节) 或AX (字) 中，程序用以搜索内存的某一数据块或字符串中有无此关键字？若有，把搜索次数记下来 (若次数为 0，表示无搜索的关键字)。且记录下存放关键字的地址。

程序一开始，当然要设置数据块的地址指针 (SCAS 指令要求设在 DI 中)，要设立数据块的长度 (要求设在 CX 中)，把关键字送入 AL中或 AX中。搜索可以用循环程序，或利用 8088/8086 中的重复前缀。利用 ZF 标志以判断是否搜索到，以便分别处理。

```
        MOV     DI , OFFSET BLOCK
        MOV     CX , COUNT
        MOV     AL , CHAR
        CLD
        REPNE   SCASB
        JZ      FOUND
        MOV     DI , 0              ; 串结束且不相等，即找不到，DI←0
        JMP     DONE
FOUND:  DEC     DI
        MOV     POINTR , DI         ; 关键字地址→POINTR单元
        MOV     BX , OFFSET BLOCK
        SUB     DI , BX             ; 找到，DI中为搜索次数
        INC     DI
DONE:   HLT
```

以下是一个能对字符串进行搜索的程序实例:

```
BUF_DAT   SEGMENT
CHAR      EQU     '$'
PTRN      DB      'THIS IS A EXAMPLE $'
COUNT     EQU     $-PTRN
BUF_DAT   ENDS
STACK     SEGMENT PARA STACK 'STACK'
STAPN     DB      100 DUP(?)
STACK     ENDS
```

```
COSEG       SEGMENT
            ASSUME    CS: COSEG , DS: BUF_DAT ,  ES: BUF_DAT , SS: STACK
BEG         PROC      FAR
BEGIN:      PUSH      DS
            MOV       AX , 0
            PUSH      AX
            MOV       AX , BUF_DAT
            MOV       DS , AX
            MOV       ES , AX
            MOV       DI , OFFSET PTRN
            MOV       AL , CHAR
            MOV       CX , COUNT
            CLD
            REPNE     SCASB           ; 串未结束且不等于搜索值，继续搜索
            JZ        PASTPR
            MOV       DI , 0          ; 未找到  DI←0
PASTPR:     DEC       DI              ; 搜索值所在地址
            RET
BEG         ENDP
COSEG       ENDS
            END       BEGIN
```

若 SCAS 指令前加上前缀 REPE 或 REPZ ，则操作解释为："当串未结束（CX≠0）且串元素＝搜索值（ ZF 标志＝1）时继续搜索"。这种格式可用来搜索与给定值不同的内容；若 SCAS 指令前加上前缀 REPNE 和 REPNZ，则操作解释为："当串未结束（CX≠0 ）且串元素不等于搜索值（ ZF 标志＝0）时继续搜索"。这个方式可以用来在一个串中查出一个值。

4. LODS OPRD (Load String)

本指令把由 SI 作为指针的串元素，传送至 AL （字节操作）或 AX （字操作），同时修改SI使指向串中的下一个元素。这个指令正常地是不重复执行的，因为每重复一次，累加器中的内容就要改写，只保留最后一个元素。但是LODS指令在一个软件循环程序中，在用基本的串操作指令，构成复杂的串操作时，LODS指令作为其中一部分是十分有用的。OPRD 是字符串的符号地址。LOADSB 和 LOADSW 分别指明是字节和字串，不带操作数，其余同 LOADS。此指令对标志位无影响。

5. STOS OPRD (Store String)

从累加器 AL （字节操作）或 AX （字操作）传送一个字节或字，到由 DI 作为指针

的目的串中，同时修改 DI 以指向串中的下一个单元。利用重复操作，可以在串中建立一串相同的值。OPRD 是目的串的符号地址。STOSB 和 STOSW 分别指明是字节和字串，不带操作数，其余同STOS。

此指令对标志位无影响。

例 1.内存中以 BUFR 单元开始的区域连续存放着一个 ASCII 字符串，编程序统计其中包括多少个字符串 'AM '，将统计的个数以十进制形式输出到显示器上。

实现上述功能的程序如下：

```
DATA    SEGMENT
BUF     DB 'KFAMANAAMAMAMAMAJFKAM'     ; 定义字符串
COUNT   EQU $-BUF
BUF1    DB 3 DUP(' ')                  ; 定义输出缓冲区
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
DB      100 DUP(?)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS: CODE , DS: DATA , SS: STACK , ES: DATA
START   PROC FAR
        PUSH    DS
        MOV     AX , 0
        PUSH    AX
        MOV     AX , DATA
        MOV     DS , AX
        MOV     SI , OFFSET BUF
        MOV     CX , COUNT
        MOV     BX , 0
        MOV     AH , 0
LOP1 :  MOV AL , [SI]
        CMP AL , 'A '                  ; 查找字符 'A '
        JNZ SRO
        INC SI
        DEC CX
        JZ  OK
        MOV AL , [SI]
        CMP AL , 'M'                   ; 查找字符 'M '
        JNZ LOP1                       ; 不是 'M '，再看是否为 'A '
```

```
        MOV AL , BL
        ADD AL , 1                      ; 是 'AM ', 十进制个数加 1
        DAA
        MOV BL , AL
SR0 :   INC SI
        LOOP LOP1
  OK:   MOV DI , OFFSET BUF1
        MOV DL , BL                     ; 十进制数转换成 ASCII 码
        SHR BL , 1
        SHR BL , 1
        SHR BL , 1
        SHR BL , 1
        MOV AL, 30H
        ADD AL , BL
        MOV [DI] , AL
        INC DI
        AND DL , 0FH
        ADD DL , 30H
        MOV [DI] , DL
        INC DI
        MOV AL , '$'
        MOV [DI] , AL
        MOV DX , OFFSET BUF1            ; 结果输出显示
        MOV AH , 9
        INT 21H
        RET
START   ENDP
CODE    ENDS
END     START
```

编写这个程序的过程需要注意：

当找到字符 'A '后，但下一个字符不是 'M '时，字符串指针不要加 1，应在字符 'A '后再找 'A '，以免漏掉字符串 'AAM '的情况。

例2：利用串操作指令对带符号的字节数进行比较，且把最大值在 CRT上显示出来。

源程序如下：

```
DATA      SEGMENT
BUFFER    DB        0，12H，35H，64H，52H，41H，7FH，80H
          DB        0ABH，  0DFH，  0FFH
COUNT     EQU       $-BUFFER
MAX       DB        3 DUP(?)
DATA      ENDS
STACK     SEGMENT   PARA STACK 'STACK'
STAPN     DB        100 DUP (?)
STACK     ENDS
COSEG     SEGMENT
          ASSUME    CS: COSEG，DS: DATA，  ES: DATA，SS: STACK
START     PUSH      DS
          MOVE      AX，0
          PUSH      AX
          MOV       AX，DATA
          MOV       DS，AX
          MOV       ES，AX
          MOV       SI，OFFSET BUFFER
          MOV       BX，OFFSET MAX
          MOV       CX，COUNT
          CLD
          LODSB
          DEC       CX
COMPA:    CMP       AL，[SI]
          JG        NEXT
          MOV       AL，[SI]
NEXT:     INC       SI
          LOOP      COMPA
          MOV       CL，AL
          SHR       AL，1
          SHR       AL，1
          SHR       AL，1
          SHR       AL，1
```

```
        CMP     AL , 0AH
        JS      OK1
        ADD     AL , 7
OK1:    ADD     AL , 30H
        MOV     [BX] , AL
        MOV     AL , CL
        INC     BX
        AND     AL , 0FH
        CMP     AL , 0AH
        JS      OK
        ADD     AL , 7
OK:     ADD     AL , 30H
        MOV     [BX] , AL
        INC     BX
        MOV     AL , '$'
        MOV     [BX] , AL
        MOV     DX , OFFSET MAX
        MOV     AH , 9
        INT     21H
        HLT
COSEG   ENDS
        END     START
```

第二节　代码转换

一．概述：

计算机在通信、管理、控制等各种应用中经常会遇到代码转换问题。比如，我们从键盘上输入一个数字，机器中接收到的是它的 ASCII 码值。 要这个数字参加二进制运算，则首先要将其转换成二进制数字，若要参加十进制运算，应先转换成十进制数字。再如内存中的二进制数字要在屏幕上以十进制形式显示出来，则首先要将其转换成十进制数，再转换成 ASCII 码才能够输出。 另外对于许多应用实例来说，代码转换是必需的。

常用的代码有二进制、八进制、十六进制、十进制、BCD 码、ASCII 码、七段显示代码等等。这里将结合实例介绍几种主要的代码间的转换。

二．代码转换

1. 代码转换形式

常用的代码转换形式有两种，一种是硬件实现，一种是软件实现。

(1) 硬件方法实现代码转换是利用一些电子线路、门电路、集成电路等制成专门的代码转换器。例如有一种集成电路芯片，它能实现十进制代码向二进制的转换。如图 6-1。

在芯片的下端8个脚上输入十进制（BCD码）数据，马上会在上端8个脚上得到其对应的二进制代码输出。

这种方法需要硬件设备投资，电路设计，产品检验等，但其转换速度快。在一些专门设备和固定应用场合可以使用。

(2) 软件方法实现是利用现有机器的指令系统编写代码转换程序，用程序的执行实现代码转换。程序设计时应找出各种代码间的关系特点，总结算法或由查表实现。这种方法经济、方便、但速度比上者慢。

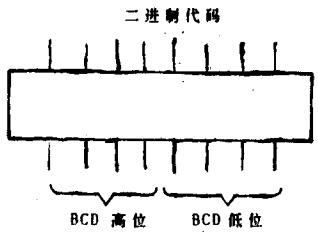


图 6-1 代码转换器

2. 代码转换程序

例 1. ASCII 码转换成 BCD 码

通常在微型计算机中，从键盘上输入的十进制数的每一位数码（即0～9中任一个），是以它的 ASCII码表示的；要向 CRT输出的十进制数的每一位代码也是用 ASCII码表示的。而在机器中的一个十进制数，或者把它转换为相应的二进制数存放，或者是以BCD 码形式存放。

若在内存的输入缓冲区中，已有若干个用 ASCII码表示的十进制数据，则每一个存贮单元只存放一位十进制数码。要求把它转换为相应的 BCD码，且把两个相邻单元的十进制数码的 BCD码合并在一个存贮单元中，且地址高的放在前四位，这样就可以节省一半存贮单元。

要把十进制数码的 ASCII码转换为 BCD码，只要把高四位变为0就可以了；要把两位并在一个存贮单元中，则只要把地址高的字节左移四位，再与地址低的字节组合在一起就可以了。

输入缓冲区中，已存放的 ASCII 码的个数有可能是偶数，但也可能是奇数。若是奇数，则把地址最低的一个转换为 BCD码（高四位为0）；然后把剩下的偶数个按统一的方法处理。下面是一个能满足上述要求的汇编语言的程序实例：

```
DATA    SEGMENT
ASCBUF  DB      31H，32H，33H，34H，35H，
        DB      36H，37H，38H，39H，30H，
COUNT   EQU     $-ASCBUF        ；DB定义的字节个数
BCDBUF  DB      5 DUP(?)
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
```

```
STAPN   DB      100 DUP(?)
STACK   ENDS
COSEG   SEGMENT
        ASSUME  CS: COSEG , DS: DATA , ES: DATA , SS: STACK
STR     PROC    FAR
GO:     PUSH    DS
        MOV     AX , 0
        PUSH    AX
        MOV     AX , DATA
        MOV     DS , AX
        MOV     ES , AX
        MOV     SI , OFFSET ASCBUF
        MOV     DI , OFFSET BCDBUF
        MOV     CX , COUNT
        ROR     CX , 1      ; 右移最低位进CF , 若数据个数为偶数转移
        JNC     NEXT
        ROL     CX , 1      ; 恢复CX内容
        LODSB               ; 取一数进 AL , 指针 SI + 1
        AND     AL , 0FH    ; 转换成 BCD码
        STOSB               ; 送入内存单元 , DI+1
        DEC     CX          ; 数据个数减 1
        ROR     CX , 1      ; 数据个数除 2
NEXT:   LODSB
        AND     AL , 0FH
        MOV     BL , AL     ; 转换一个 ASCII码到 BCD码
        LODSB
        PUSH    CX          ; 保护 CL 内容
        MOV     CL , 4
        SAL     AL , CL     ; 再转换一个 ASCII码到 BCD存入高四位
        POP     CX
        ADD     AL , BL     ; 两个 BCD码合为一个字节
        STOSB
        LOOP    NEXT        ; CX-1 , CX<>0 转NEXT重复直到CX=0结束
        RET
STR     ENDP
```

```
COSE        ENDS
            END     GO
```

例2　BCD 码转换成 ASCII码

BCD码的十进制数字在 CRT上显示时，要求先将其转换为 ASCII码。

本例题的流程图如图6－2。

下述程序自动在 CRT上循环输出 0～98 之间的十进制数。

```
;    NAME      OUTPUT CHARACTER 0～98
STACK     SEGMENT   PARA STACK 'STACK'
STP         DB      100 DUP(?)
STACK       ENDS
DATA        SEGMENT
; 定义缓冲区
BUFR        DB      3 DUP(?)
DATA        ENDS
COSEG       SEGMENT
            ASSUME  CS: COSEG
            ASSUME  DS: DATA
            ASSUME  ES: DATA
START:      PUSH    DS
            MOV     AX , 0
            PUSH    AX
            MOV     AX , DATA
            MOV     DS , AX
            MOV     ES , AX
            MOV     BL , -1
            PUSH    BX   ; 保护 BL
GOON:       MOV     SI , OFFSET BUFR
  ; 输出回车
            MOV     DL , 0DH
            MOV     AH , 2
            INT     21H
            MOV     DL , 0AH
            MOV     AH, 2              ; 输出换行
            INT     21H
            POP     BX
```

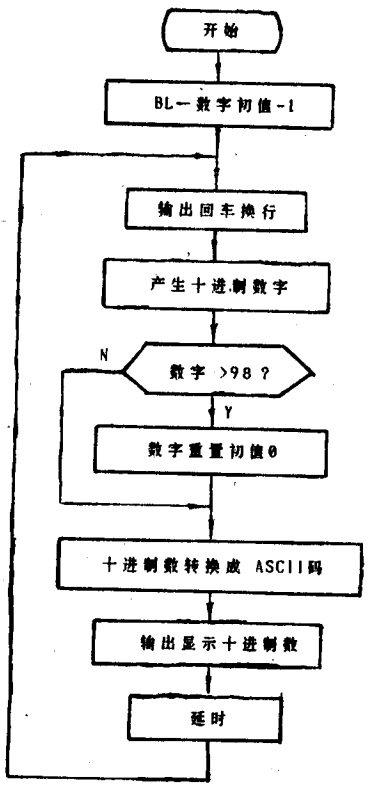


图 6-2　程序流程图

```
        MOV     AL，BL              ；产生0～98数据
        INC     AL
        DAA
        CMP     AL，99H
        JC      NEXT        ；若数字大于98，再从0 开始
        MOV     AL，0
NEXT:   MOV     BL，AL
        PUSH    BX          ；保护BL，以便下个循环在此数上加 1
        MOV     DL，AL      ；暂存入DL，以备转成 ASCII码用
        MOV     CL，4
        SHR     AL，CL      ；右移四次
        OR      AL，30H     ；高位十进制数转成 ASCII码
        MOV     [SI]，AL    ；存到缓冲区
        INC     SI
        MOV     AL，DL      ；恢复原十进制数
        AND     AL，0FH
        OR      AL，30H     ；低位十进制数转成 ASCII码
        MOV     [SI]，AL
        INC     SI
        MOV     AL，'$'
        MOV     [SI]，AL    ；'$'存入缓冲区，9号调用要求的
        MOV     DX，OFFSET BUFR
        MOV     AH，9
        INT     21H         ；缓冲区数据输出显示
        MOV     CX，0FFFFH
AGN:    DEC     CX          ；延时后再显下一个数
        JNE     AGN
        JMP     GOON
COSEG   ENDS
        END     START
```

程序本身是一个死循环。可以安排一段读键盘输入程序，当按下某一预定的停止标志键时，使循环结束。

例3　二进制数转换成 ASCII码。

假设 CX 寄存器中有一个带符号的二进制数，现将其以十进制形式输出到 CRT。

CX中的数是 16 位二进制数，所表示的数的范围在 +32767 到 -32768 之间。程序应

先检查CX中数的符号位，以决定输出"＋"或"－"，若是负数，应先求补，得到原码，然后再将其转换成十进制数。转换方法是首先将其减 10000 ，看其中包含几个 10000，那么它的十进制数的万位数字就是包含的一万的个数，不够减时再用余下的数减一千，统计其中含一千的个数，得出千位上的数字，以此类推，求出百位、十位数的数字，剩下的就是个位数字了。

转换后的十进制数在缓冲区的存放格式如图6－3。

有了十进制数字以后，要在 CRT上显示出来，还要再将其转换成 ASCII码才行。

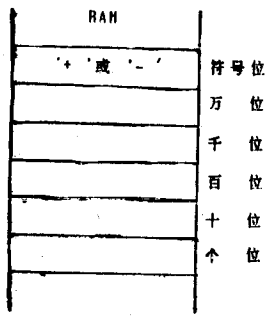


图 6-3 内存缓冲区存放格式

下述程序将实现这个功能。

```
DATA        SEGMENT
CONT        DW          ?
BUF         DB          9 DUP(?)
DATA        ENDS
STACK       SEGMENT     PARA STACK 'STACK'
            DB          100 DUP(?)
STACK       ENDS
CODE        SEGMENT
            ASSUME      CS: CODE, DS: DATA, ES: DATA , SS , STACK
START       PROC        FAR
            PUSH        DS
            SUB         AX , AX
            PUSH        AX
            MOV         AX , DATA
            MOV         DS , AX
            MOV         ES , AX
            MOV         CX , CONT    ;  CX 中为待输出的二进制数
            CALL        PTDN
            RET
START       ENDP
; 输出 16 位二进制带符号数子程序
; 入口参数:  CX 中为待输出的数据
; 所用子程序:  CHANG
PTDN        PROC
            PUSH        AX
            PUSH        BX
```

```
        PUSH    DX                  ; 保护现场
        PUSH    SI
        MOV     BX,OFFSET BUF       ; BX指向缓冲区
        MOV     AL,0DH
        MOV     [BX],AL             ; 回车符存入缓冲区
        INC     BX                  ; 以便显示时回车
        MOV     AL,0AH
        MOV     [BX],AL             ; 换行符存入缓冲区
        INC     BX
        MOV     AL,CH
        OR      AL,AL               ; 查看符号位
        JNS     PLUS                ; 正数转 PLUS
        NEG     CX                  ; 负数求补
        MOV     AL,'-'
        MOV     [BX],AL             ; 缓冲区存入负号
        JMP     GOON
PLUS:   MOV     AL,'+'
        MOV     [BX],AL         ; 缓冲区存入正号
GOON:   INC     BX
        MOV     SI,10000
        CALL    CHANG           ; 求万位数，并转成 ASCII码存入缓冲区
        MOV     SI,1000
        CALL    CHANG           ; 求千位数
        MOV     SI,100
        CALL    CHANG           ; 求百位数
        MOV     SI,10
        CALL    CHANG           ; 求十位数
        MOV     AL,30H
        ADD     AL,CL
        MOV     [BX],AL         ; 求个位数
        INC     BX
        MOV     AL,'$'          ;  $送入缓冲区尾，9号调用要求的
        MOV     [BX],AL
        MOV     DX,OFFSET BUF
        MOV     AH,9
```

```
        INT     21H             ; 9号调用输出显示
        POP     SI
        POP     DX
        POP     BX              ; 恢复现场
        PO      AX
        RET
PTDN    ENDP
; 本子程序统计 CX 寄存器所包含权（在SI中）的个数，
; 并把这个个数转换成 ASCII码。
; 入口参数：SI中为权码，CX中为给定的数，
;         BX指向 ASCII码结果缓冲区。
; 出口参数：结果缓冲区的当前单元存放转换完的 ASCII码，
;         并且调整 BX 指向缓冲区的下一个单元。
CHANG   PROC
        MOV     DL，0       ; DL存放权的个数，初值0
AGIN:   SUB     CX，SI
        JC      DOWN        ; 不够减转 DOWN
        INC     DL          ; 够减 DL 加1
        JMP     AGAIN       ; 再减权
DOWN:   ADD     CX，SI      ; 恢复 CX
        MOV     AL，30H
        ADD     AL，DL
        MOV     [BX]，AL    ; 转 ASCII码并存入缓冲区
        INC     BX          ; 指针调整
        RET
CHANG   ENDP
CODE    ENDS
        END     START
```

例4　ASCII码转换成二进制数

要求编制接收从键盘上输入的十进制整数的程序（设数的范围为+32767到-32768）。

接收键盘输入可用10号功能调用，10号调用要求预先定义一个缓冲区，缓冲区的第一单元存放缓冲区长度，第二单元留作存放实际输入的字符个数，第三个单元开始存放数据，这可在数据段加以定义。

从键盘接收的十进制数字的 ASCII字符，应将它转换成二进制数。转换方法是，先将其转成十进制数字，再用累加和乘 10 加 X 的方法变成二进制数。例如，将 369 转二进

制数，可先将累加和赋为0，再计算 (((0*10+3)*10+6)*10+9)，结果就是二进制数。然后再由符号位决定是否需要求补。

下述程序能够从键盘上接收两个带符号十进制数，将它们转换成二进制数，并且求出这两个数的乘积，然后再将乘积的高、低位字分别转换成十进制数在屏幕上显示出来。

```
DATA    SEGMENT
BUF     DB      10
        DB      ?
        DB      10 DUP(?) ; 定义输入缓冲区
SIGN    DB      ?         ; 存放输入数据的正负标志
BLK     DB      9 DUP(?)  ; 输出缓冲区
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
        DB      100 DUP (?)
STACK   ENDS
COSEG   SEGMENT
        ASSUME  CS: COSEG , DS: DATA , ES: DATA , SS: STACK
START   PROC    FAR
        PUSH    DS
        MOV     AX , 0
        PUSH    AX
        MOV     AX , DATA
        MOV     DS , AX
        MOV     ES , AX
        CALL    GETD    ; 接收键入的第一个带符号十进数并转成二进制数
        PUSH    CX
        MOV     DL , 0DH
        MOV     AH , 2
        INT     21H
        MOV     DL , 0AH
        MOV     AH , 2
        INT     21H       ; 输出回车换行
        CALL    GETD      ; 接收第二个十进制数，并转成二进制数
        POP     AX
        IMUL    CX        ; 两数相乘结果在 DX , AX
        CALL    PUTDN     ; 准备将二进制积转成十进制，先处理符号
```

```
        MOV     CX,DX
        CALL    GOON        ;积的高位字转成十进数输出
        MOV     CX,AX
        CALL    GOON        ;积的低位字转成十进数输出
        RET
START   ENDP
;输入十进制数,并将其转换成二进制数。
;结果在 CX 寄存器中。
;所用寄存器 AX,BX,DX,SI
GETD    PROC
        PUSH    AX
        PUSH    BX
        PUSH    DX
        PUSH    SI
        MOV     DL,':'
        MOV     AH,2
        INT     21H             ;输出提示符":"
        MOV     DX,OFFSET BUF
        MOV     AH,10
        INT     21H             ;接收键盘输入
        MOV     SI,OFFSET BUF+1
        MOV     BL,[SI]
        DEC     BL              ;实际输入的字符个数送 BL
;记录符号位 .
        INC     SI
        MOV     AL,0
        MOV     SIGN,AL         ;符号单元送0
        MOV     AL,[SI]         ;取实际符号位
        CMP     AL,'+'
        JZ      NEXT1           ;若实际符号为负,
        MOV     AL,1            ;则符号单元送1
        MOV     SIGN,AL
;  ASCII码转换成 BCD码
NEXT1:  PUSH    BX              ;保存字符个数
NEXT2:  INC     SI
```

```
            MOV     AL , [SI]
            AND     AL , 0FH
            MOV     [SI] , AL          ;  ASCII码转成 BCD码送回
            DEC     BL                 ; 字符个数减 1
            JNZ     NEXT2              ; 若不等于 0 , 继续转换
            POP     BX
;  BCD码转换成二进制码
            MOV     DI , OFFSET BUF+3
            MOV     CX , 0             ; 累加和初值
AG1:        PUSH    BX
            ADD     CX , CX
            MOV     BX , CX
            ADD     CX , CX
            ADD     CX , CX
            ADD     CX , BX            ; 累加和 *10
            MOV     BL , [DI]
            MOV     BH , 0
            INC     DI
            ADD     CX , BX            ; 累加和 *10后加下一个 BCD数字
            POP     BX
            DEC     BL
            JNE     AG1                ; 未转换完时继续
            MOV     AL , SIGN
            OR      AL , AL            ; 查看符号位
            JZ      DONE
            NEG     CX                 ; 若为负数求补
DONE:       POP     SI
            POP     DX
            POP     BX
            POP     AX
            RET
GETD        ENDP
; 本过程根据 DX 和 AX 中的一个带符号数的正负号 ,
; 在缓冲区建立符号标志 , 并输出。
; 所用寄存器:  BX , CX
```

```
; 入口参数: DX 和 AX 中为一带符号二进制数
; 出口参数: DX 和 AX 中内容为原数的原码
PUTDN   PROC
        PUSH    CX
        PUSH    BX
        MOV     BX,OFFSET BLK
        MOV     CL,0DH
        MOV     [BX],CL
        INC     BX
        MOV     CL,0AH
        MOV     [BX],CL         ; 回车、换行送输出缓冲区
        INC     BX
        OR      DH,DH
        JNS     PLUS
        NOT     AX
        NOT     DX
        ADD     AX,1
        JNC     AD1
        INC     DX              ; 负数求补
AD1:    MOV     CL,'-'
        MOV     [BX],CL
        JMP     RON
PLUS:   MOV     CL,'+'
        MOV     [BX],CL
RON:    INC     BX
        MOV     CL,'$'
        MOV     [BX],CL
        PUSH    AX
        MOV     DX,OFFSET BLK
        MOV     AH,9
        INT     21H             ; 输出回车、换行、数的负号
        POP     AX
        POP     BX
        POP     CX
        RET
```

```
PUTDN       ENDP
; 下述过程将 CX 中的无符号数转换成十进制数并输出
; 所用寄存器:  BX , AX , DX , SI
; 入口参数:  CX 中为二进制无符号数
; 出口参数: 缓冲区 BLK中为转换成的十进制数字的ASCII代码
; 所用子程序:  CHANG

GOON        PROC
            PUSH    AX
            PUSH    BX
            PUSH    DX
            PUSH    SI
            MOV     BX , OFFSET BLK
            MOV     SI , 10000
            CALL    CHANG
            MOV     SI , 1000
            CALL    CHANG
            MOV     SI , 100
            CALL    CHANG
            MOV     SI , 10
            CALL    CHANG
            MOV     AL , 30H
            ADD     AL , CL
            MOV     [BX] , AL
            MOV     AL , '$'
            INC     BX
            MOV     [BX] , AL
            MOV     DX , OFFSET BLK
            MOV     AH , 9
            INT     21H
            POP     SI
            POP     DX
            POP     BX
            POP     AX
            RET
```

```
GOON        ENDP
;  CHANG过程与上例中完全相同，从略。
CHANG       PROC
            .
            .
            RET
CHANG       ENDP
COSEG       ENDS
            END       START
```

第三节　表的处理和应用

表的应用是很广泛的。前面我们已经接触到多种用途的表，如求平方值的表，实现程序分支的跳转表等等。表中还可以存放一系列供机器执行的任务，一组组的结果，一系列有关联的数据……，供各种运算、查询等用。

一．表的处理

对表的处理是多样的，主要的有以下几个方面：

1. 查看：　检查一下表中某个或某些单元的内容是什么，以便对不同内容做不同处理等。

2. 插入：　将一新的内容插入到表中某个单元以前或以后。这里就需要先将插入位置以后的数据后移，然后再将数据插入，同时表元素的个数也应做相应的修改（增加）。

3. 删除：　将表中某些内容删除。为保持表的完整，应将被删内容以后的数据前移，并修改表元素个数。

4. 排序：　按某种规律（升序或降序）将表中内容重新排列组织。

5. 搜索：　给定某元素，到表中查找，看是否存在此元素，存在何处，并做些其它处理等。

其中某些方面在前几章的例题中已讲述过。现举几例将其它方面的问题加以讲述。

首先我们来介绍一条换码指令 XLAT。XLAT 指令用表中的一个字节（称为换码字节）来置换累加器 AL 中的内容。用此指令以前，要求先把表的起始地址取入 BX 寄存器，且在 AL 中置好所需的初值，AL 中的初值为所需的换码字节在表中的相对位置（用字节数给出），又可称为查找所需换码字节的索引值。然后 XLAT 指令将 BX 内容加上 AL 内容所形成的地址单元中的内容（即所需的换码字节）取到 AL 中去。现结合实例看其应用。

例 1．利用 XLAT 指令把十六进制数转换成 ASCII码。其程序如下：

```
TABLE       SEGMENT
TAB_DA      DB        30H，31H，32H，33H，34H，35H，36H，37H，38H，39H
```

```
        DB      41H，42H，43H，44H，45H，46H
CONT    EQU     $-TAB_DA
HEXTAB  DB      0，1，2，3，4，5，6，7，8，9，0AH，0BH，0CH，0DH
        DB      0EH，0FH
ASIBUF  DB      16 DUP(?)
TABLE   ENDS
STACK   SEGMENT PARA STACK 'STACK'
STAPN   DB      100 DUP(?)
STACK   ENDS
COSEG   SEGMENT
        ASSUME  CS: COSEG，DS: DATA，ES: TABLE，SS: STACK
STR     PROC    FAR
START   PUSH    DS
        MOV     AX，0
        PUSH    AX
        MOV     ES，AX
        MOV     DS，AX
        MOV     BX，OFFSET TAB_DA
        MOV     SI，OFFSET HEXTAB
        MOV     DI，OFFSET ASIBUF
        MOV     CX，CONT
NEST:   LODSB
        XLAT    TAB_DA
        STOSB
        LOOP    NEST
        RET
STR     ENDP
COSEG   ENDS
        END     START
```

例2　数据或程序的加密和解密

为了使数据能够保密，可以建立一个密码表，利用 XLAT 指令将数据加密。例如，从键盘上输入0-9 间的数字，经加密后存到内存中，密码表可选择为：

原数字：　　0，1，2，3，4，5，6，7，8，9

密码数字：　7，5，9，1，3，6，8，0，2，4

该加密程序如下所示，程序接受键入的一个数字，加密后存入 MIMA 单元。

```
DATA    SEGMENT
MITAB   DB      '7591368024'    ; 加密密码表
CONT    EQU     $-MITAB
JMITAB  DB      '7384915062'    ; 解密密码表
MIMA    DB      ?
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS: CODE , DS: DATA
STAR    PROC    FAR
        PUSH    DS
        MOV     AX , 0
        PUSH    AX
        MOV     AX , DATA
        MOV     DS , AX
        MOV     AH , 1
        INT     21H
        AND     AL , 0FH
        LEA     BX , MITAB
        XLAT    MITAB
        MOV     MIMA , AL
        RET
START   ENDP
CODE    ENDS
        END     STAR
```

可以改造上述程序为循环程序，使之能连续地接收键盘输入数字，遇到某一规定的标志字符时结束输入，并将输入的数字加密后存到内存缓冲区。

用类似的方法，可将内存中的程序代码加密，也可将编译或汇编后的高级语言程序或汇编语言程序加密。

在数据通信中也可以用类似的方法，先将要发送的代码加密以后再发送。

为将加密后的数据或程序复原，可编写解密程序。下述程序可将 MIMA 单元中的数据解密，结果送屏幕显示。

```
        MOV     AL , MIMA
        AND     AL , 0FH
        LEA     BX , JMITAB
        MOV     AH , 0
```

```
        ADD     BX，AX
        MOV     DL，[BX]
        MOV     AH，6
        INT     21H
        HLT
```

还可以利用 XLAT 指令将键盘输入的密码数字解密。下述程序接收键盘输入一个密码数字，解密后的数字在 AL 中。

```
        MOV     AH，1
        INT     21H
        AND     0FH
        LEA     BX，JMITAB
        XLAT    JMITAB
        HLT
```

可以用同样的方法对加密的程序或通信中的数据解密。

例3：有一 100个字节的数据表，表内元素已按从小到大的顺序排列好。现给定一元素，试编程序在表内查找，若表内已有此元素，则结束；否则，按顺序将此元素插入表中适当的位置，并修改表长。

这个问题的主要环节是：当发现表中无此元素时，应将其插在表中适当位置，也就是大于（或等于）前一个元素，并且小于（或等于）后一个元素的位置。同时其后的元素应依次后移。详见下列程序。

```
DATA    SEGMENT
LTH     DB      100             ；数据表长
TAB     DB      5FH，……         ；有序数据表
TEM     DB      'X'             ；给定元素
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
        DB      100 DUP(?)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS: CODE，DS: DATA，ES: DATA
STR     PROC    FAR
START:  PUSH    DS
        MOV     AX，0
        PUSH    AX
        MOV     AX，DATA
```

```
        MOV     DS , AX
        MOV     ES , AX
        MOV     BX , OFFSET TAB   ; BX指向数据表
        MOV     AL , TEM          ; 取给定元素
        MOV     CX , LTH          ; 取表长
LOP:    CMP     AL , [BX]         ; 在表中查找
        JE      SOP               ; 找到转 SOP
        JL      INST              ; 若给定元素小于表内元素，转INST插入
        INC     BX
        DEC     CX
        JNZ     LOP
        MOV     [BX] , AL         ; 给定元素一直大于（或等于）表内元素，
        JMP     JUST              ; 应将给定元素插在表末。
INST:   MOV     AH , [BX]         ; 取出表中元素暂存入 AH
        MOV     [BX] , AL         ; 插入给定元素
        INC     BX
LOPI:   MOV     AL , [BX]
        MOV     [BX] , AH         ; 表中插入位置后的元素后移
        INC     BX
        MOV     AH , [BX]
        MOV     [BX] , AL
        INC     BX
        DEC     CX
        DEC     CX
        JNZ     LOPI
JUST:   INC     LTH               ; 修改表长
SOP:    RET
STR     ENDP
CODE    ENDS
        END     START
```

至于删除操作，则有许多地方与插入类似，它应将被删元素之后的数据一一前移，并将表长减一。

例 4　气泡排序法

在前面我们介绍串操作指令 SCAS 时，曾介绍了利用 SCAS 对数据块（或字符串）进行搜索的程序例子，但是，这种搜索是线性搜索，即从数据块的第一个单元开始，一个一

个单元的进行搜索，这种搜索方法最简单，但它是最慢的一种方法，它的平均搜索次数D=N/2 。当数据个数 N很大时，则搜索次数就很多，搜索时间就很长。较常用的方法是我们在下例中要介绍的对分搜索法，它可以大大减少搜索次数。但是，对分搜索的一个前提是要求表格中的数据（或字符）的排列是有次序的。例如对于数字，要求它按数字的大小排列，字符则按其 ASCII码值的大小排列。于是对一个无次序的表格，首先要设法对其排序，排序的方法也很多，下例中我们以 10 个数为例介绍一种气泡排序法 (Bubble Sort)。

如有一数组它有 N个数，在本例中为如下的 10 个数： 22，-12，80，-6，-70，-9，127，-10，00，40。希望把它们按从小到大排列。参见图 6 − 4 。

I=2									I=3								
J=10	9	8	7	6	5	4	3	2	J=10	9	8	7	6	5	4	3	2
22	22	22	22	22	22	22	22	22 -70	-70	-70	-70	-70	-70	-70	-70	-70	-70
-12	-12	-12	-12	-12	-12	-12	-12 -70	22	22	22	22	22	22	22	22	22 -12
80	80	80	80	80	80	80 -70	-12	-12	-12	-12	-12	-12	-12	-12	-12	-12 22
-6	-6	-6	-6	-6	-6 -70	80	80	80	80	80	80	80	80	80 -10	-10	-10
-70	-70	-70	-70	-70	-70 -6	-6	-6	-6	-6	-6	-6	-6	-6 -10	80	80	80
-9	-9	-9	-9 -10	-10	-10	-10	-10	-10	-10	-10	-10	-10 -10	-6	-6	-6	-6
127	127	127 -10	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9	-9
-10	-10 -10	127	127	127	127	127	127	127	127	127 00	00	00	00	00	00	00
00→	00→	00	00	00	00	00	00	00	00	00	00 127	127	127	127	127	127	127
40→	40→	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40

图 6-4 气泡排序法

我们采用两两比较的方法：先拿 e_N 与 e_{N-1} 比，若 $e_N > e_{N-1}$ ，则不交换，反之则交换；然后拿 e_{N-1} 与 e_{N-2} 相比，按同样原则决定是否交换，这样一直比下去，最后拿 e_2 与 e_1 相比，也按同样原则决定是否交换，当第一次大循环结束时， 数组中的最小值冒到了顶部。但是数组尚未按大小排列好，还要进行第二次大循环，这样数组中的第二个最小值，也上升到顶部。……这样不断地循环下去，若数组的长度为 N，则最多经过 N-1次上述的大循环，就可以使数组按大小的次序排列整齐。在每一个大循环中，数两两比较的次数，在第一次大循环时为 N-1；在第二次大循环时为 N-2……。但是，有的数组不需要经过 N-1次大循环就已经排列整齐了。为了在程序中去除不必要的循环，就可以设置一个标志，在每次大循环开始时，置此标志为 0；若在整个大循环中，两两比较后，发生过交换，则置此标志为 -1 。然后在下一次大循环开始时，检查此标志，若不为 0 ，表示数组未排列好，继续进行循环；若为 0 ，则表示数组已按大小次序排列好（每次两两比

较时，都是大的数在下，小的数在上，故不用交换），就停止循环。

能够满足上述要求的程序如下：

```
            NAME    BUBBLE SORT
SORT_DA     SEGMENT
ARRAY       DW      1234，5673，7FFFH，8000H，0FFFFH，0AB55H
            DW      0369H，005FH，5634，9069H
COUNT       EQU     $-ARRAY
SORT_DA     ENDS
STACK       SEGMENT PARA STACK 'STACK'
            DB      100 DUP(?)
STACK       ENDS
CODE        SEGMENT
            ASSUME  CS: CODE，DS: SORT_DA，ES: SORT_DA
STR         PROC    FAR
START:      PUSH    DS
            MOV     AX，0
            PUSH    AX
            MOV     AX，SORT_DA
            MOV     DS，AX
            MOV     ES，AX
            MOV     BL，0FFH        ；标志减1送 BL
A1:         CMP     BL，0
            JE      A4              ；若 BL=0 ，表示已排序好
            XOR     BL，BL          ；清 BL
            MOV     CX，COUNT
            SHR     CX，1
            DEC     CX              ；CX 中为循环次数
            XOR     SI，SI
A2:         MOV     AX，ARRAY[SI]
            CMP     AX，ARRAY[SI+2] ；两数比较
            JLE     A3              ；若 e(N-1)<eN 则不交换
            XCHG    ARRAY[SI+2]，AX
            MOV     ARRAY[SI]，AX   ；否则，两数交换
            MOV     BL，0FFH        ；发生交换后，置标志为-1
A3:         INC     SI
```

```
            INC     SI
            LOOP    A2
            JMP     A1
A4:         RET
STR         ENDP
CODE        ENDS
            END     START
```

二．查表方法

常用的查表方法有以下几种。

1.顺序查表法

有时要查找的内容与表之间没有什么规律可循，则只好从表首开始逐个比较、查找。如果要查找的内容恰在表首，则一次找到，查找时间最短， 若恰在表尾， 则查找时间最长。若将比较、判断、查找一次所用的时间记为 t单次，并假设表中有 N个数据元素，则顺序查表法的平均查找时间为：

t平均＝ t单次 * N/2

这种查找方法速度是慢的。

例如，许多单板机上的显示器采用七段代码方式的数码管。如图 6-5 所示。

数码管的各段由数据总线的 D0～D6 位控制，若某一位为 0 ，则相应段点亮 ；若某位为 1，则相应段不亮。要让数码管显示数字 0 ，应送给它代码 40H；显示 1 ，应送 79H；显示 2 ，应送 24H；…… 显示 F ，应送 0EH。这些毫无规律的代码可以组成一个表，存放在内存以 SSEG 为首址的连续单元中。可用 DB 定义：

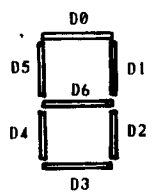


图 6-5 七段显示数码管

```
SSEG        DB      40H，79H，24H，30H，・・・0EH
```

它们对应的显示数字分别是十六进制数字 0～F 。如果现在任给一七段显示代码，要求利用此表求出它所对应的十六进制数字来。 因为无规律可循， 只好用顺序查表方法。实现它的程序如下：

```
            MOV     AL，XX              ；设任给的代码在 XX 单元
            MOV     CX，10H             ；表长
            MOV     BX，OFFSET SSEG     ；BX指向表首指
            MOV     AH，0               ；十六进制数字初值
LOP1:       CMP     AL，[BX]
            JE      AT
            INC     AH
```

```
        INC     BX
        LOOP    LOP1
        MOV     AL，0FFH        ；若查不到，0FFH 送 AL
        HLT
AT:     MOV     AL，AH          ；查到时 AH 中为对应的十六进制数字
        HLT
```

本程序的执行结果在 AL 中，若表中无所给代码，则 AL 中为 0FFH，若有，则 AL 中即为对应的十六进制数字，因为表中代码存放的顺序对应于 0～F 的顺序。

2. 计算查表法

如果要查找的内容和它在表的位置之间存在一定的规律，即可以用某一公式（或表达式）表示出来，这时用计算查表法又快又好。例如我们前面讲过的跳转表的查找就属于这种方法。它是利用"表地址＝表首址＋偏移量"这个公式进行计算查找的。有时侯这种规律可以人为地去建立（例如跳转表的建立），而后再利用这一规律进行查找。有时侯全局性的规律没有，可以将问题分段处理，也许可以有局部性规律可循。这儿不再细述。

3. 对分查表法

若表中内容已按大小次序排列好，则可采用对分查找法，这种方法可以大大减少查找次数。对分查找的思想是：先取表中间的值 $e_{N/2}$（N/2 处的数值）与要查找的值 X 比较，看是否相等，若相等则搜索到；若不等则比较两数的大小若 $X > e_{N/2}$，则下次取N/2～N 之间的中间值 $e_{3N/4}$ 与 X比较；若 $X < e_{N/2}$，则下一次取 0～N/2之间的中间值 $e_{N/4}$ 与 X比较，这样每查找一次使区间缩小 1/2，如此一直进行下去，直至或者是被搜索的字找到；或者是搜索的区间变为 0，则表示搜索不到所要找的数。其过程如下所示：

若有一组数：00，11，15，21，34，57，60，78，90，27。数组的个数N=10，而数的排列序号为 0～N-1=9。若要搜索的数为 X=78。

取区间上限 L=0，区间下限 R=10，则序号

$$J = (L+R) / 2 = (0 + 10) / 2 = 5$$

取出的数 e_J =57 与 X=78 相比，两者不等；而且 $X > e_J$。于是下一次要到下半区间去寻找，则取 L=J=5，R=10 不变，求出新的序号

$$J = (L+R) / 2 = (5+10) / 2 = 7.5$$

往下取整，取 J=7，则 e_J =78与 X相等，找到。上述的搜索过程可用表 1 表示。

表 1

搜索次数 K	1	2
区间上限 L	0	5
区间下限 R	10	10
序　号　J	5	7

数值	e_J	57	78
比 较		$X>e_J$	$X=e_J$

若要搜索的数字为 X=20 ，则搜索过程可用表 2 表示。

表 2

搜索次数 K	1	2	3	4
区间上限 L	0	0	2	2
区间下限 R	10	5	5	3
序 号 J	5	2	3	2
数 e_J	57	15	21	15
比 较	$X<e_J$	$X>e_J$	$X<e_J$	$X<>e_J$

当搜索过程中，求出的序号 J已与区间的上限相等时，表示搜索的区间已压至最小，若此时的 e_J 仍不等于 X，则表示被搜索的数组中没有要搜索的关键字。按上述方法进行搜索的程序如下：

```
NAME        BINARY_SEARCHING
BUF_DAT     SEGMENT
BUFFER      DB      '$ AGHI NRSTUVWXYZ '
COUNT       EQU     $-BUFFER
PTRN        DW      ?           ; 存放查找次数或未找到标记
CHAR        EQU     'A'         ; 关键字 'A'
BUF_DAT     ENDS
STACK       SEGMENT PARA STACK 'STACK'
STAPN       DB      100 DUP(?)
STACK       ENDS
CODE        SEGMENT
            ASSUME  CS: CODE , DS: BUF_DAT , ES: BUF_DAT
STR         PROC    FAR
START:      PUSH    DS
            MOV     AX , 0
            PUSH    AX
            MOV     AX , BUF_DAT
            MOV     DS , AX
            MOV     ES , AX
```

```
        MOV     SI , OFFSET BUFFER  ; 区间上限送 SI
        MOV     CX , COUNT
        MOV     DX , 1              ; 查找次数
        MOV     AX , SI
        ADD     AX , CX             ; 最后数的地址加 1
        MOV     DI , AX             ; 下限送 DI
        MOV     AL , CHAR
CONT1:  MOV     BX , SI
        ADD     BX , DI
        SHR     BX , 1              ;  BX 中为 J
        CMP     AX , [BX]           ; 关键字与 eJ 比较
        JZ      FOUND               ; 找到转 FOUND
        PUSHF                       ; 保护标志
        CMP     BX , SI             ;  J = 上限吗?
        JZ      NOFID               ; 相等未找到
        POPF
        JL      LESS                ; 关键字< eJ 转,
        MOV     SI , BX             ;  J作上限
        JMP     NEXT
LESS:   MOV     DI , BX             ;  J作下限
NEXT:   INC     DX                  ; 查找次数加 1
        JMP     CONT1
NOFID:  MOV     DX , OFFFFH         ; 未找到标记 OFFFFH
FOUND:  MOV     AX , DX
        MOV     PTRN , AX           ; 结果送 PRTN 单元
        RET
STA     ENDP
CODE    ENDS
        END     START
```

当数据较多时，对分查找比顺序查找速度快得多，但它要求表内容有序。设表长为 N = 65536。顺序查找平均次数 n1 为：

$$n1=N/2=65536/2=32768$$

而对分查找次数 n2 有公式（公式证明从略）可计算。

$$n2=LOG_2(N-1)=LOG_2\ 65535-1=15$$

第四节　算术运算

一．十进制算术运算

前面已讲过加法、减法等运算指令，这些指令都是对二进制数进行操作。8086/8088还提供了各种校正操作指令，故可以进行十进制算术运算。这里所说的十进制数是以 BCD码表示的，分为未组合的（或非压缩的）十进制数和组合的（或压缩的）十进制数。组合的十进制数是一字节表示二位 BCD码十进制数，未组合的则一个字节表示一位十进制数，字节的高四位为零。

1. 未组合的 BCD码加法调整指令 AAA:

这条指令对在 AL 中的由两个未组合的十进制数相加后的结果进行校正，产生一个未组合的十进制和在 AX 中。

AAA指令要紧跟在加法指令之后使用。例如寄存器 AL 、 BL 中各存有两个未组合的十进制数 08 和 07 。要求其和， 则可直接用 ADD 指令相加，而后紧跟着一条校正指令AAA，在 AX 中就得到一未组合的十进制和。实际上 ADD AL，BL是按二进制规则进行的，即:

```
        0 0 0 0 1 0 0 0
+       0 0 0 0 0 1 1 1
-------------------------
        0 0 0 0 1 1 1 1
```

相加的结果也是二进制的 0FH。 AAA指令的调整操作为:

若 (AL&0FH)>9 或标志 AF=1 ，则

AL←AL+6 ， AH←AH+1 ， AF←1 ， CF←AF ，

AL←AL & 0FH

上题调整结果使 AH=01，AL=05 ，即为未组合的十进制和。其实这其中的道理也很简单。因为本来 BCD码就是取二进制 8421 码的前 10 个数字 0000～1001 表示十进制0～9的，按二进制规则 9 (1001)加 1 应是 A(1010)，而按 BCD码的十进制规则 9 加 1 应是 10 (0001 0000) ，而这中间相差 6。所以当 AL 的低四位大于 9 或辅助进位为 1 时，用加 6 方法强制跳过二进数字 1010～1111 (即 A－F) 而成为 0001 0000 即可。

2. 十进制加法调整指令 DAA:

这条指令对在 AL 中的两个组合的十进制数相加的结果进行校正，得到正确的组合的十进制结果在 AL 中。

DAA指令要紧跟在加法指令之后使用，以实现组合的十进制数加法运算。它的校正操作为:

若 (AL&0FH)>9 或标志 AF=1 ，则

AL←AL+6 ， AF←1 ；

若 (AL&0F0H)>90H 或标志 CF=1 则

AL←AL+60H , CF←1 。

3. 未组合的 BCD码减法调整指令 AAS

本指令与 AAA指令类似，能把在 AL 中的由两个未组合的十进制数相减的结果校正成一个正确的未组合的十进制数差在 AL 中。

AAS指令应紧跟在减法指令之后使用，以实现未组合的十进制数的减法运算。其校正操作为：

若 (AL&0FH)>9 或标志 AF=1 ，则

AL←AL-6 , AH←AH-1 , AF←1 , CF←AF ,

AL←AL&0FH

4. 十进制减法调整指令 DAS：

本指令与 DAA指令相似，能对在 AL 中的由两个组合的十进制数相减的结果校正成正确的组合的十进制差在 AL 中。

8086/8088 允许两个组合的十进制数直接相减，但要得到正确的结果，必须在减法指令后紧跟一条 DAS指令加以校正才行。校正操作：

若 (AL&0FH)>9 或标志 AF=1 ，则

AL←AL-6 , AF←1

若 (AL&0F0H)>90H 或 CF=1 则

AL←AL-60H , CF←1

例：内存中以 FIRST和 SECOND 开始的单元中分别存放着两个 16 位组合的十进制（BCD码）数，低位在前。编程序求这两个数的组合的十进制和，并存到以 THIRD开始的单元。

十六位组合的十进制数占用8个字节，可用循环程序重复加8次，但要注意字节间进位，同时要对加的结果进行调整。实现上述要求的程序段可如下：

```
        MOV     BX,OFFSET FIRST
        MOV     SI,OFFSET SECOND
        MOV     DI,OFFSET THIRD
        MOV     CX,8
        CLC
AGN:    MOV     AL,[BX]
        ADC     AL,[SI]
        INC     BX
        ADC     BYTE PTR[BX],0
        DAA
        MOV     [DI],AL
        INC     SI
```

```
        INC     DI
        DEC     CX
        JNZ     AGN
        HLT
```

如果用串操作指令会更好，见下述程序：

```
DATA    SEGMENT
FIRST   DB      11H，52H，...       ；定义十个 BCD码数
SECOND  DB      66H，69H，...       ；定义十个 BCD码数
THIRD   DB      11 DUP(?)
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
        DB      100 DUP(?)
STACK   ENDS
CODE    SEGMENT
        ASSUME  CS: CODE，DS: DATA，ES: DATA
STR     PROC    FAR
START   PUSH    DS
        MOV     AX，0
        PUSH    AX
        MOV     AX，DATA
        MOV     DS，AX
        MOV     ES，AX
        MOV     SI，OFFSET FIRST
        MOV     DI，OFFSET THIRD
        MOV     BX，OFFSET SECOND
        MOV     CX，LENGTH THIRD
        DEC     CX
        CLD
        CLC
        MOV     AH，0
ADIT:   LODS    FIRST           ；取第一个数
        ADC     AL，[BX]        ；与第二个数相加
        INC     BX
        ADC     BYTE PTR[BX]，0 ；存结果
        DAA
        STOS    THIRD
```

```
            LOOP    ADIT
            ADC     AH , 0
            MOV     AL , AH
          , STOSB
            RET
STR         ENDP
CODE        ENDS
            END     START
```

二．乘法运算

8086/8088 中有三条乘法操作指令

1. MUL （ MULtiplication ）

本指令完成在 AL （字节）或 AX （字）中的无符号数与另一个无符号数的乘法。双倍长度的乘积，送回到 AL 和 AH （在两个 8 位数相乘时），或送回到 AX 和它扩展部分 DX （在两个字操作数相乘时）。

若结果的高半部分（在字节相乘时为 AH ；在字相乘时为 DX ）不为零，则标志 CF =1，OF=1 ；否则 CF=0，OF=0。（标志 CF=1，OF=1 表示在 AH 或 DX 中包括有结果的有效数）。

本指令影响标志C和O。

相乘时的另一操作数可以是寄存器操作数或内存操作数。

若要把内存单元 FIRST 和 SECOND 这两个字节的内容相乘，乘积放在 THIRD 和 FOURTH 单元中，可以用以下程序段：

```
            MOV     AL , FIRST
            MUL     SECOND
            MOV     THIRD , AX
```

下面是一个实现两个字（ 16 位）相乘的程序例子。

```
MY_DATA     SEGMENT
M1          DW      00FFH           ; 被乘数
M2          DW      00FFH           ; 乘数
P1          DW      ?               ; 存积
P2          DW      ?
MY_DATA     ENDS
STACK       SEGMENT PARA STACK 'STACK'
STAPN       DB      100 DUP(?)
STACK       ENDS
COSEG       SEGMENT
```

```
            ASSUME    CS: COSEG , DS: MY_DATA , ES: MY_DATA
STR         PROC      FAR
MULT:       PUSH      DS
            MOV       AX , 0
            PUSH      AX
            MOV       AX , MY_DATA
            MOV       DS , AX
            MOV       ES , AX
            MOV       AX , M1
            MUL       M2
            MOV       P1 , AX              ; 存结果
            MOV       P2 , DX
            RET
STR         ENDP
COSEG       ENDS
            END       MULT
```

2. IMUL (Integer Multiply)

整数乘法指令。这条指令除了是完成两个（一个是 AL ，若是字节相乘；或 AX ，若是字相乘）带符号数相乘以外，其它与 MUL完全类似。若结果的高半部分（对于字节相乘则为 AH ，对于字相乘则为 DX ）不是低半部分的符号扩展的话，则标志CF=1，OF=1；否则 CF=0，OF=0。（若结果的 CF=1，OF=1，则表示高半部分包含有结果的有效数（ 不光是符号部分））。

```
例:         IMUL      CL     结果 AX=AL*CL       (有符号数)
            IMUL      BX     结果 DX: AX=AX*BX   (有符号数)
```

3. AAM (Unpached BCD Adjust for Multiply)

这条指令能把在 AX 中的两个未组合的十进制数相乘的结果， 进行校正， 最后在AX中能得到正确的未组合的十进制数的乘积（即高位在 AH 中，低位在 AL ）。

8086/8088 允许两个未组合的十进制数相乘，但要得到正确的结果，必须在 MUL指令之后，紧跟着一条 AAM指令进行校正，最后可在 AX 中得到正确的两个未组合的十进制数的乘积。

校正的操作为：

```
AH←AL/0AH       ( AL 被 0A 除的商→ AH )
AL←AL%0AH       ( AL 被 0A 除的余数→ AL )
```

如前所述，一个未组合的十进制数是一位十进制数。所以当两个未组合的十进制数按二进制的规则相乘时，乘积的有效数在 AL 中，其值为两位十进制数。要在 AX 中得到用

未组合的十进制表示的乘积，则乘积的十进数值应在 AH 中， AL 中为个位数值，所以用上述的校正操作能得到正确的结果。

此指令影响标志位P、S、Z。

例1．BCD 码相乘

8086/8088 中有两个未组合的十进数相乘后的调整指令。所以可以实现一位未组合的十进制数 X与一个十进制串相乘。把串中的未组合的十进制数，从低位开始取入AL，与 X 相乘，对乘积进行调整，然后把乘积的低位与上一次的高位（即进位部分）相加。这样一位位乘下去，最后可得乘积。能实现上述过程的程序如下：

```
DATA    SEGMENT
A       DB      3,7,5,4,9          ; 十进制数 94573
COUNT   EQU     $-A
B       DB      6
C       DB      COUNT+1 DUP(?)
DATA    ENDS
STACK   SEGMENT PARA STACK 'STACK'
STAPN   DB      100 DUP(?)
STACK   ENDS
COSEG   SEGMENT
        ASSUME  CS: COSEG,DS: DATA,ES: DATA
STR     PROC    FAR
GO:     PUSH    DS
        MOV     AX,0
        PUSH    AX
        MOV     AX,DATA
        MOV     ES,AX
        MOV     DS,AX
        CLD
        MOV     SI,OFFSET A
        MOV     DI,OFFSET C
        MOV     CX,COUNT
        MOV     BYTE PTR [DI],0
CYCLE:  LODSB
        AND     AL,0FH
        MUL     B
        AAM
```

```
            ADD     AL,[DI]
            AAA
            STOSB
            MOV     [DI],AH
            LOOP    CYCLE
            RET
STR         ENDP
COSEG       ENDS
            END     GO
```

例2　两个两位的 BCD码相乘

8086/8088 中的乘法指令可实现 8位或 16 位二进制数相乘；或两个未组合的十进制数组乘（要经过 AAM调整）。但若是两个两位的用 BCD码表示的十进制数，就不能直接相乘（即没有相应的调整指令）。但可以用累加的方法，编一个程序来实现两位组合的十进制数乘法。算法是对被乘数累加乘数所规定的次数。被乘数的每次累加和都要经过 DAA调整；乘数每次减 1 以后也要经过调整。程序如下所示：

```
            NAME    MULTIPLY BCD
DATA        SEGMENT
FIRST       DB      25H
SECOND      DB      25H
THIRD       DB      2 DUP(?)
DATA        ENDS
STACK       SEGMENT PARA STACK 'STACK'
STAPN       DB      100 DUP(?)
STACK       ENDS
COSEG       SEGMENT
            ASSUME  CS: COSEG,DS: DATA,ES: DATA
STR         PROC    FAR
START:      PUSH    DS
            MOV     AX,0
            PUSH    AX
            MOV     AX,DATA
            MOV     DS,AX
            MOV     ES,AX
            MOV     BL,FIRST        ; 取乘数
            MOV     CL,SECOND       ; 取被乘数
```

```
        MOV     DX,0            ;  DX 存积
        MOV     AL,BL
AGAIN:  OR      AL,AL
        JZ      DONE            ; 若乘数为0，则转 DONE
        MOV     AL,DL
        ADD     AL,CL
        DAA                     ; 结果在 AL
        MOV     DL,AL
        MOV     AL,DH
        ADC     AL,0            ; 处理进位（加进位）
        DAA
        MOV     DH,AL
        MOV     AL,BL
        SUB     AL,1            ; 乘数减1
        DAS
        MOV     BL,AL
        JMP     AGAIN
DONE:   MOV     BX,OFFSET THIRD
        MOV     [BX],DX
        RET
STR     ENDP
COSEG   ENDS
        END     START
```

三．除法运算

8086/8088 有三条除法指令，另外有两条符号扩展操作指令，以支持带符号数的除法运算。

1. DIV (DIVision)

这条无符号数的除法指令，能把在 AX 和它的扩展部分（若是字节相除，则在 AH 和 AL 中，若是字相除，则在 DX 和 AX 中）中的无符号被除数被源操作数除，且把相除后的商送至累加器（8位时送至 AL，16 位时送至 AX），余数送至累加器的扩展部分（8位时送至 AH，16 位时送至 DX）。

若除数为0，或商的结果超出相应寄存器的范围，则在内部产生一个类型0的中断。

例如：　　DIV CL

将实现 AX/CL，所得商在 AL 中，余数在 AH 中。

2. IDIV (Integer Division)

这条指令除了是完成带符号数相除以外，与 DIV完全相似。在字节相除时，最大的商是+127(7FH)，而最小的负数商为-127(81H)；在字相除时，最大的商为 .+32767(7FFFH)，最小的负数商为 -32767 (8001H)。若相除以后， 商是正的且超过了上述的最大值，或商是负的且小于上述的最小值，则与被零除一样，在内部产生一个类型0的中断。

3. AAD (Unpacked BCD Adjust For Division)

这条指令能把在 AX 中的两个未组合的十进制数在两个数相除以前进行校正，这样在两个未组合的十进制数相除以后可以得到正确的未组合的十进制结果，商在 AL 中，余数在 AH 中。8086/8088 允许两个未组合的十进制数直接相除，但要得到正确的未组合的十进制商和余数，则在相除之前，先用一条 AAD指令，然后再用一条 DIV指令，则相除以后的商送至 AL 中，而余数送至 AH 中， AH 和 AL 中的高半字节全为0。

这条指令影响标志位P、S、Z。

校正的操作为：

AL←AH*0AH+AL

AH←0

4. CBW (Convert Byte to Word)

这条指令能扩展在寄存器 AL 中的字节的符号，把它送至 AH 中。

若 AL < 80H ，则扩展以后 AH ←0

若 AL >=80H ，则扩展以后 AH←0FFH

这条指令能在两个字节相除以前，产生一个双倍长度的被除数。此指令不影响标志位。

5. CWD (Convert Word to Donable Word)

这条指令能把在 AX 中的字的符号扩展送至寄存器 DX 中。

若 AX < 8000，则 DX ←0 ；否则 DX←0FFFFH。这条指令不影响标志位。

这条指令能在两个字相除以前，把在 AX 中的符号扩展至 DX 中，形成双倍长度的被除数，从而能完成相应的除法。

若在内存的数据段中，有一个缓冲区 BUFFER ，前两个字节是一个 16 位带符号的被除数，第三、四字节是一个 16 位带符号的除数。接着两个字节放商，再下两个字节放余数。能实现除法运算的程序为：

```
LEA     BX，BUFFER
MOV     AX，[BX]
CWD
IDIV    2[BX]              ；带符号数除法
MOV     4[BX]，AX
MOV     6[BX]，DX
```

四．复杂运算

8086/8088 为我们提供了实现加、减、乘、除运算的基本指令，但对于复杂的运算（例如：多字节数的乘、除运算等），需要我们利用这些基本指令并结合一些算法编程序实现。

现举几例加以说明：

1. 32 位有符号数乘法

设有两个 32 位数，一个是被乘数存放在字单元 MCD1_HI 和 MCD1_LO 中，一个是乘数存放在字单元 MCD2_HI 和 MCD2_LO 中，求它们的积。有符号数变成无符号数运算，符号位单独处理。

首先我们来看一下多位十进制数相乘的情况，这里采取每次乘的结果（部分积）分别存放的方法，如下：

```
        36
    *   62
-------------
        12          第一个部分积（2*6）
       06           第二个部分积（2*3）
       36           第三个部分积（6*6）
  +    18           第四个部分积（6*3）
-------------
      2232          积
```

由此我们可以得到启发，DX:AX 和 CX:BX 中的两个 32 位数相乘可采用如下方法：

```
                    DX      AX
                *   CX      BX
          ---------------------
                    pt1_hi  pt1_lo      BX*AX 产生的部分积 1
              pt2_hi  pt2_lo            BX*DX 产生的部分积 2
              pt3_hi  pt3_lo            CX*AX 产生的部分积 3
  +  pt4_hi   pt4_lo                    CX*DX 产生的部分积 4
-------------------------------------------------------------
                                        乘积
```

可在程序中定义 8 个字的缓冲区分别存放各个部分积的高位和低位字，然后对应字相加，得最终乘积；也可以一边乘一边把部分积累加，后一种方法比较好。用后一种方法运算的具体程序如下：

```
STACK       SEGMENT   PARA STCK 'STACK'
            DW        100 DUP(?)
```

```
STACK     ENDS
DATA      SEGMENT
RLT_S     DB      ?          ; 存结果符号位
MCD1_HI   DW      ?          ; 被乘数
MCD1_LO   DW      ?
MCD2_HI   DW      ?          ; 乘数
MCD2_LO   DW      ?
RESULT    DB    8 DUP(0)     ; 结果区（低位字节在前）
DATA      ENDS
CODE      SEGMENT
          ASSUME  CS: CODE , DS: DATA, ES: DATA
MUL32B_S  PROC    FAR
          PUSH    DS
          MOV     AX , 0
          PUSH    AX
          MOV     AX , DATA
          MOV     DS , AX
          MOV     ES, AX
          MOV     DX, MCD1_HI
          MOV     AX, MCD1_LO
          MOV     CX, MCD2_HI
          MOV     BX, MCD2_LO
          MOV     RLR_S , 0       ; 结果符号位置初值 0
          CMP     DX , 0          ; 被乘数是负数吗?
          JNS     SIGN            ; 不是负数转走
          NOT     AX
          NOT     DX
          ADD     AX , 1          ; 是负数，求补
          ADC     DX , 0
          NOT     RLT_S           ; 结果符号位置 1
SIGN:     CMP     CX , 0          ; 乘数为负数吗?
          JNS     CALMUL          ; 非负，转走
          NOT     CX
          NOT     BX              ; 负乘数求补
          ADD     BX , 1
```

```
            ADC     CX , 0
            NOT     RLT_S           ; 结果符号取反
CALMUL:     CALL    MUL32B          ; 调无符号数乘法过程
            CMP     RLT_S , 0       ; 查看结果符号位
            JZ      DONE            ; 为正，即结束
            NOT     WORD PTR [DI]   ; 为负，求补
            NOT     WORD PTR [DI+2]
            NOT     WORD PTR [DI+4]
            NOT     WORD PTR [DI+6]
            ADD     WORD PTR [DI] , 1
            ADC     WORD PTR [DI+2] , 0
            ADC     WORD PTR [DI+4] , 0
            ADC     WORD PTR [DI+6] , 0
DONE:       RET
MUL32B_S    ENDP
MUL32B      PROC    NEAR
            MOV     DI, OFFSET RESULT
            MOV     MCD1_HI , DX
            MOV     MCD1_LO , AX
            MUL     BX                  ; 产生部分积 1
            ADD     [DI], AX
            ADD     [DI+2], DX
            MOV     AX , MCD1_HI
            MUL     BX                  ; 产生部分积 2
            ADD     [DI+2],AX
            ADC     [DI+4], DX
            MOV     AX , MCD1_LO
            MUL     CX                  ; 产生部分积 3
            ADD     [DI+2], AX
            ADC     [DI+4], DX
            ADC     WORD PTR [DI+6] , 0
            MOV     AX , MCD1_HI
            MUL     CX                  ; 产生部分积 4
            ADD     [DI+4], AX
            ADC     [DI+6], DX
```

```
        RET
MUL32B  ENDP
CODE    ENDS
        END     MUL32B_S
```

2，32位数除法

32位数除16位数的运算，可以直接利用 8086/8088提供的除法指令，不过当除数为零或商的结果超出相应寄存器的范围时，会产生类型为零的中断。

假设 32位的被除数在 DX：AX中，16位的除数在 BX 中(无符号数)。为了在结果超出相应寄存器范围时，仍能得到正确的结果，可以分两次做除法运算。第一次用被除数的高位字除除数，得商的高位字部分，第二次用余数和被除数的低位字部分除除数，得商的低位字部分和余数。实现这一运算的程序如下所示。

```
    ;  DX：AX中为被除数，BX 中为除数
    ;  商在BX：AX中，余数在DX中
    CMP  BX，0
    JZ   DONE                    ；若除数为0 则停止运算
    PUSH AX
    MOV  AX，DX
    MOV  DX，0
    DIV  BX                      ；求商的高位字，余数在DX中
    POP  CX
    PUSH AX
    MOV  AX，CX
    DIV  BX                      ；求商的低位字
    POP  BX                      ；商在BX：AX中
DONE:   HLT
```

我们还可以编写一个通用程序，使原有的被除数与除数直接相除，若相除后没有产生零号中断，则表明得到了正确的结果；若产生了零号中断怎么办呢？我们可以按照上述分两次做除法的思想编写一个中断服务程序，当发生零号中断时，让程序自动转入我们的中断服务程序来执行，那么也可以得到正确的结果。

为了达到这一目的，就要修改中断矢量表，用我们的中断服务程序的入口地址去替换中断矢量表中原零号中断所占用的四个字节内容。详见下述程序。

```
  DATA      SEGMENT
  OPRD1_HI  DW  ?          ；被除数
  OPRD1_LO  DW  ?
  OPRD2     DW  ?          ；除数
```

```
RESULT_HI  DW  ?            ; 结果
RESULT_LO  DW  ?
ERR        DB 'DIVISOR=0', '$'
DATA       ENDS
STACK      SEGMENT PARA STACK 'STACK'
           DB 100 DUP(9)
STACK      END
CODE       SEGMENT
           ASSUME CS: CODE, DS: DATA, ES: DATA
  STA      PROC FAR
           PUSH DS
           MOV AX, 0
           PUSH AX
           MOV AX, DATA
           MOV DS, AX
           MOV ES, AX
           PUSH ES                    ; 保护原0号中断矢量
           MOV CX, 0
           MOV ES, CX
           PUSH ES: [0]
           PUSH ES: [2]
           LEA CX, MODIFY_INT         ; 使零号中断矢量指向中断
           MOV ES: [0], CX            ; 服务程序MODIFY_INT
           MOV CX, CS
           MOV ES: [2], CX
           STI
           MOV BX, OPRD2
           CMP BX, 0
           JE  DONE1
           MOV DX, OPRD1_HI
           MOV AX, OPRD1_LO
           DIV BX                     ; 若结果溢出，将产生0号中断
           MOV RESULT_LO, AX          ; 结果不溢出
           MOV RESULT_HI, 0
DONE:      POP ES: [2]                ; 恢复原零号中断矢量
```

```
            POP ES: [0]
            POP ES
            RET
DONE1:      MOV DX , OFFSET ERR          ; 显示错误信息
            MOV AH , 9
            INT 21H
            JMP DONE
MODIFY_INT: POP CX
            LEA CX , DONE
            PUSH CX
            MOV AX, OPRD1_HI
            MOV DX , 0
            DIV BX                       ; 商的高位字在 AX, 余数在 DX
            MOV RESULT_HI, AX
            MOV AX , OPRD1_LO
            DIV BX
            MOV RESULT_LO, AX
            STI
            IRET
STA         ENDP
CODE        ENDS
            END    STA
```

上述程序首先保护中断矢量表中原零号中断矢量，然后修改中断矢量表，使零号中断矢量指向我们的中断服务程序 MODIFY_INT。 程序检查除数是否为零，若为零则显示错误信息后，返回。若除数不为零，则两数相除，若商未超出允许范围，即商在AX中，余数在 DX 中，正常返回；若商超出 AX 寄存器的范围，则产生中断，执行中断服务程序MODIFY_INT，产生正确结果， RESULT_HI和RESULT_LO中为商，DX中为余数，从中断返回时，返到DONE语句，再从那儿退出程序。

不论哪一种情况，退出程序前都恢复中断矢量表中原有的零号中断矢量。

第五节　浮点数运算

本节只对浮点数运算做大概介绍，不讲述具体程序。目的是帮助读者建立一点关于浮点数运算的初步概念。感兴趣的同志可参考有关 INTEL 8087协处理器（浮点运算器)的资料。

一．浮点数

8086/8088 是定点机，它处理的数据的小数点是固定的。一般指定小数点在数的最末位之后，这时认为参入运算的都是整数。也可以指定小数点在符号位之后，这时认为参入运算的数是纯小数。如果参入运算的数不是整数或纯小数，在编程序时要采用比例因子的方法对参加运算的数进行处理，使它满足要求。

小数点不固定的数称为浮点数。当机器字长一样时，采用浮点数表示的数比定点数范围大。定点机可进行浮点运算，INTEL 8087 运算处理器可进行快速浮点运算，8086/8088 汇编语言中引进浮点数主要供它使用。

浮点数类似于科学记数法。在计算机中，浮点数通常有如下三种格式： 32 位（短实数）、64位(长实数)和80位(Intel暂用实数格式)。前两种是IEEE浮点数标准推荐的，后一种是8087内部使用。浮点数由符号、指数、尾数几部分组成。以64位为例，如图6-6所示。

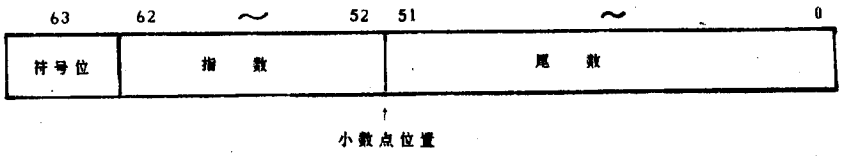


图 6-6 浮点数表示法

计算机中采用隐含的方法表示小数及其位置，即认为尾数中的整数位只有一位，小数点紧跟在这位整数之后。浮点数要求以规格化形式存放。即尾数的整数部分必须为 1，不符合规格化的数，就应移位（ 指数也相应加减变化 ）使之规格化。既然整数部分一定是 1，因此为节省，计算机中并不存贮此位，所以实际尾数总在1～2之间。

符号位表示数的正 (0)、负 (1)。数的实际小数点位置是由指数决定的。指数有正有负，但不采取补码表示，而采用偏码表示法。所谓偏码即用实际的指数再加上一个常数构成，从而使得偏码指数永为正。在64位浮点数中此常数为 1023 (3FFH)。例如实际指数为 -2，则偏码指数将是1021H。例如：

十进制数	浮点数(64位格式)		
10	40240000	0000	0000H
-127	C05FC000	0000	0000H

32 位格式中，一位符号位、8 位指数、23 位尾数。所能表示的正数范围约等于 1.175×10^{-38} ～ 3.40×10^{38}，64 位格式中所能表示的正数范围约等于 3.36×10^{-4932} ～1.19×10^{4932}。

二、浮点数运算

1. 浮点数乘除运算

浮点数进行乘除法运算，可分几步进行。首先，若参加运算的数不符合规格化要求，应先进行规格化处理。即将运算数右移或左移同时每移一位指数减1或加1，直到整数位为 1，并舍弃它，得尾数部分。

结果的符号位，众所周知，同符号数相乘除结果为正，异号为负，那么结果的符号可由两数的符号位进行“异或”运算而得到(也可用“或”运算，或者“按位加”)。

结果的指数，应等于两运算数的指数之和(乘法)或指数之差(除法)。

尾数处理，结果的尾数等于两运算数之尾数的积(乘法)或商(除法)，这可直接调用前述的多字节定点数乘除法运算程序实现。

运算结果应进行规格化处理以后再存贮，乘除法运算的流程图如图 6-7 所示。

程序清单从略

2.浮点数加减运算

浮点数进行加法运算，也要分几步进行。两数相加，应首先使其指数对齐(相等)，指数对齐的方法可先比较两指数大小，将小的指数换成大的指数，同时将对应的尾数右移n位（n应等于两指数之差，即保证移位后，运算数的实际值不变)。指数对齐后，尾数才可进行加法运算。

关于符号处理，可先检查两数的符号是否相同，若相同，可将两个尾数相加，若不同，则先检查两尾数绝对值是否相等，若相等，则结果的尾数为 0 ，若两尾数的绝对值不相等，则以绝对值大者减绝对值小者，作为和的尾数，而以绝对值大者的符号作为和的符号。

最后，结果应规格化处理，当结果为 0 时，直接将尾数和符号位清零，指数变成偏码常数即可。

浮点数减法运算，只要改变减数的符号，即可将减法变为加法，再调用浮点数加法程序即可。

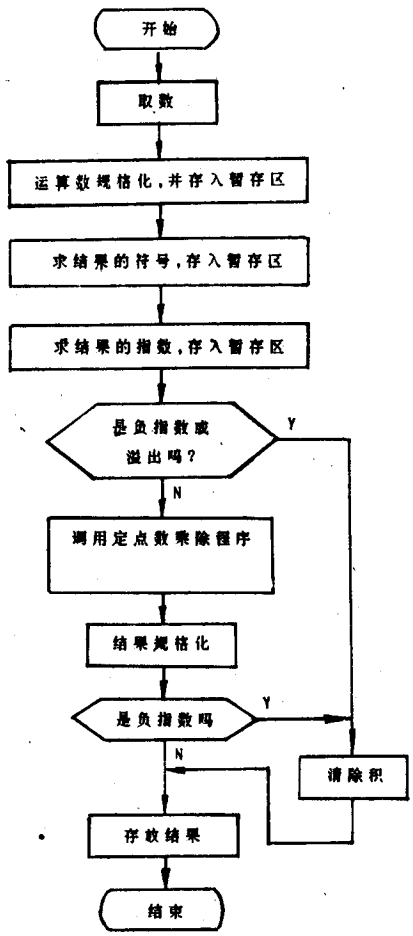


图 6-7 浮点数乘除运算流程图

第六节 图形显示

一、概述：

IBM PC 的标准显示器接口板有两种，一是单色显示和并行打印接口板，它能显示黑白字母、数字及方块图形字符，即以文本方式工作。显示方式为 80列 25行，即80×25 。二是彩色 /图形监视器接口板。它可以两种方式工作，即文本方式或图形方式，显示可以

黑白或彩色的。文本方式时，接口板可以产生 80×25的高分辨率或 40×25的低分辨率显示。 图形方式时，接口板把屏幕分割成图象的点素。而不再是方块显示。屏幕可由320×200 或 640×200个点素组成。汉字就是按图形方式显示的。

IBM PC 的接口电路提供了 CRT 以文本方式和图形方式显示的硬件基础，而且在ROM BIOS 中还提供了有关的显示驱动程序。这些显示驱动程序的调用可以由10H号中断调用来实现。下面我们首先介绍 10H中断调用的功能，再举例说明如何利用这些功能实现图形显示。

二、 10H中断调用

10H中断调用包括16个功能，功能编号为 00～0FH。调用之前要求将功能号送入AH寄存器中。调用期间，寄存器 CS、SS、DS、ES、CX、DX的值被保护，其它寄存器被破坏。

现分别介绍各项功能如下：

1．设置显示方式 （功能号 0H）

调用参数： AH=0，为功能号 0

AL中为方式字， AL=0， 为40×25黑白文本方式

AL中为方式字， AL=1， 为40×25彩色文本方式

AL中为方式字， AL=2， 为80×25黑白文本方式

AL中为方式字， AL=3， 为80×25彩色文本方式

AL中为方式字， AL=4， 为320×200彩色图象方式

AL中为方式字， AL=5， 为320×200黑白图象方式

AL中为方式字， AL=6， 为640×200黑白图象方式

AL中为方式字， AL=7， 为80×25黑白文本方式(单色显示板)

返回参数： 无

例：
```
MOV  AH，0
MOV  AL，2
INT  10H                ；为80×25黑白文本方式
```

2．设置光标大小 （功能号01H）

调用参数 ：AH=1　CH=光标开始行号，CL=光标结束行号。

返回参数：无

3.设置光标位置 （功能号2H）

调用参数：AH=2，BH=页号，DH=行号，DL=列号

返回参数：无

例：
```
MOV AH，2
MOV BX，0
MOV DX，1020H           ；行号16，列号32
INT 10H
```

4．读当前光标位置 （3H）

调用参数：AH=3，BH=页号

返回参数：DH=行号，DL=列号，CX=当前光标大小。

5．读光笔位置 （4H）

调用参数：无

返回参数：若AH=0，表示光笔开关未按下或没触发。

若AH=1，表示下列寄存器中为合法的光笔值。

DH=行号，DL=列号(文本方式下)。

DH=行号(0～199)，BX=列号(0～314/639)(图象方式下)。

6．选择显示页 （05H）

调用参数：AH=5，AL=新页号（方式为 0、1 时，新页号为 0～7，方式 2、3 时，为 0～3 ）。

返回参数：无

7．屏幕上滚（ 06H ）

调用参数： AH=6，AL=上滚行数，窗口底部为空白输入行，若AL=0；表示整屏为空白。

CH、CL=滚动的左上角行、列号，DH、DL=滚动的右下角行、列号，BH =空白输入行的属性。

返回参数：无

```
例：MOV AX，0601H      ；上滚一行
    MOV BH，07         ；通常显示(属性)
    MOV CX，0
    MOV DX，184FH      ；从0行0列到24行79列
    INT 10H
```

8.屏幕下滚 （07H）

调用参数：AH=07 其余类同6号功能，只是窗口顶部为输入行。

返回参数：无

9.读当前光标位置的字符与属性 （08H）

调用参数：AH=08，BH=页号

返回参数：AL为读出的字符，AH为读出的字符属性。

10.在当前光标位置写字符（包括属性和ASCII码） （09H）

调用参数： AH=9，BH=页号，AL=要写的字符的ASSII码，

BL=字符属性(文本方式下)/字符颜色(彩色方式下)，

CX=要写的字符个数。

返回参数：无

11.在当前光标位置写字符（属性不改变）（0AH）

调用参数：除AH=0AH且无属性参数外，其余类同9号功能。

返回参数：无

12．设置彩色组或边缘颜色 （0BH）

调用参数：AH=0BH

当BH=0时，为设置边缘颜色(文本方式下)或设置背景颜色(图象方式下)；BL=彩色值(0～15)，当BH=1时，为设置彩色组，BL=0或1。

其中BL=0则选彩色组为绿/红/黄， BL=1则选 彩色组为青/品红/白。

返回参数：无。

13．写点 （0CH）

调用参数： AH=0CH，DX=行号，CX=列号，AL=彩色值(若AL的位7为1，则彩色值与当前点内容做“异或”运算。

返回参数：无

14．读点 （0DH）

调用参数： AH=0DH，DX=行号，CX=列号

返回参数： AL=被读点的返回值

15．写字符并移光标位置 （0EH）

调用参数 AH=0EH，AL=要写的字符，BH=页号，BL=前景颜色(图象方式下)。

返回参数： 无

16．读当前显示状态 （0FH）

调用参数： AH=0FH

返回参数： AL=当前显示方式（见0号动能 ），

AH=屏幕上字符列数(Width)， BH=当前页号。

三、字符图形显示

在文本方式下，CRT可以显示方块方符图形和由方块字符图形组装成的复杂图形，并且可以使图形移动。

1.简单字符图形的显示

为在显示器上显示方块字符图形，首先要给出它的 ASCII码值，这一点可以通过查阅本书的附录 B 得到（附录 B 的 IBM PC ASCII 码字符表中，给出了所有字母数字，方块字符图形的 ASCII码值），同时还要给出一个8位的属性字节。这个属性字节指定字符或背景的颜色、显示亮度的强弱以及是连续显示还是闪动。表 6-3给出了黑白方式显示字符的属性字节。

要在屏幕上显示字符或字符图形，可利用 10H中断调用的09H或0AH功能。这两个功能的区别仅在于属性是否改变，0AH是按在此以前设置的属性进行显示，09H在显示字符的同时，也设置属性。下段程序将在行号为 9，列号为 9 的位置显示“红桃”。

表 6-3　黑白方式的显示属性字节

显示方式	位号 7 6 5 4 3 2 1 0	字符颜色	背景颜色
	BL R G B I Z G B		
通常显示	BL 0 0 0 I 1 1 1	白(绿)	黑
反相显示	BL 1 1 1 I 0 0 0	黑	白(绿)
不显示(无物)	BL 0 0 0 I 0 0 0	黑 色	黑 色
不显示(白框)	BL 1 1 1 I 1 1 1	白(绿)	白(绿)

注：BL=0为前景字符不闪动，BL=1为前景字符闪动；I=0前景字符为一般强度，I=1前景字符为高强度。

```
MOV   AH，0
MOV   AL，2
INT   10H                ；设置为80×25黑白文本方式
MOV   AH，15
INT   10H                ；获当前页号
MOV   DX，  0909H
MOV   AH，  2
INT   10H                ；置光标位置
MOV   AL，  03           ；取“红桃”字符图形
MOV   BL，  70H          ；反相显示属性
MOV   CX，  1
MOV   AH，  9
INT   10H                ；显示“红桃”
```

2．字符图形移动

利用上述的 10H 号中断，我们可以编写使图形字符移动的程序。便如，让"太阳 "字符(编码 0FH)首先在屏幕位置(0，0)显示，然后沿斜线向下，每次移动一行一列。

图形的移动可以分几步进行：

(1)．先在屏幕上显示某个图形。

(2)．延时适当时间。

(3)．清除这个图形。

(4)．改变图形显示的行、列坐标。

(5)．返回第(1)步，重复上述过程。

实现“太阳”移动的程序如下：

```
MOVE        PROC   FAR
```

```
        MOV     AH,   15
        INT     10H
        MOV     AH,   0
        MOV     AL,   2
        INT     10H
        MOV     CX,   1          ;  要显示的字符个数为1
        MOV     DX,   0
REPT:   MOV     AH,   2
        INT     10H              ;  置光标位置(0,0)
        MOV     AL,   0FH
        MOV     AH,   10
        INT     10H              ;  显示“太阳”
        CALL    DELAY            ;  延时
        SUB   AL,AL
        MOV   AH,10              ;  清除原图形
        INT   10H
        INC   DH
        INC   DL
        CMP   DH,25
        JNE   REPT
        RET
MOVE    ENDP
EDLAY   PROC
        PUSH    CX
        PUSH    DX
        MOV     DX, 50
DL500:  MOV     CX, 2801
DL10MS: LOOP    DL10MS
        DEC     DX
        JNZ     DL500
        POP     DX
        POP     CX
        RET
DELAY   ENDP
```

时间的延时是调用过程 DELAY 实现的。这个过程用指令的执行时间延时 0.5 秒，也

可以用前面讲过的 1AH 中断计算时间，时间计数器加到 9(55ms*9≈0.5s)也表示半秒钟。写一个空字符到原来位置的操作可以清除图形。如果没有延时和清图程序，将看不出图形的移动，而是一条由“太阳”字符组成的斜线。

3．复杂图形显示：

我们可以用多个字符图形组成复杂图形，也可以使这个图形移动起来。多字符图形要多次显示，一次显示一个字符。字符很多时，可以组成图形字符表。并且选定某个字符位置作参考点，计算出其它字符的行、列相对偏移量。

6-8 多字符图形

例如，由三个字符组成的“小人”图形如图 6-8 所示。

若指定最上一个字符图形位置为参考点，则中间一个字符和最下方字符的行列号相对偏移量分别为（1，0），(1，0)， 将字符的代码及行、列号相对偏移量组成一个表，如下所示：

```
CHRTAB   DW   3
         DB   01H，0，0；  最上方字符的代码，行、列号相对偏移量。
         DB   04H，1，0；  中间字符的代码，行、列号相对偏移量。
         DB   13H，1，0；  下方字符的代码，行、列号相对偏移量。
```

下述程序可以完成这个多字符图形的显示。

```
DATA     SEGMENT
CHRTAB   DW   3
         DB   01H，0，0，04H,1,0，13H,1,0
DATA     ENDS
STACK    SEGMENT PARA STACK 'STACK '
         DB 50 DUP( ? )
STACK    ENDS
CODE     SEGMENT
         ASSUME CS: CODE，DS: DATA
PICTURE PROC  FAR
         PUSH  DS
         MOV AX,0
         PUSH AX
         MOV AX,DATA
         MOV DS，AX
         STI
         MOV  AH，15
```

```
        INT  10H                    ; 读当前页号
        MOV  DI, OFFSET CHRTAB
        MOV  CX, [DI]
        MOV  DX,  0                 ; 参考点定位在 0行 0页
        ADD  DI, 2
NEXT:   ADD  DH,[DI+1]
        ADD  DL, [DI+2]
        MOV  AH,2                   ; 显示字符图形
        INT  10H
        MOV  AL, [DI]
        PUSH CX
        MOV  CX, 1
        MOV  AH, 10
        INT  10H
        POP  CX
        ADD  DI,3
        LOOP  NEXT
        RET
PICTURE ENDP
CODE    ENDS
        END PICTURE
```

再配合一个外层循环程序，修改参考点DX中的内容，即可使图形移动起来。仍然不要忘记显示延时和清除图形(一个一个字符清除)，才能产生好的移动效果。

如果让参考点 DX 中的值取自一个随机数值，则上述的组合图形将随机地出现在屏幕上，有跳跃感。也可以让某一字符以闪烁方式显示。

CRT 可以在彩色文本方式下工作，16种可能的颜色组合如表 6-4。

表 6-4 TRGB组合颜色

I	R	G	B	颜 色
0	0	0	0	黑
0	0	0	1	兰
0	0	1	0	绿
0	0	1	1	青(深兰)

表 6-4 （续）

0	1	0	0	红
0	1	0	1	品红
0	1	1	0	棕
0	1	1	1	亮灰
1	0	0	0	暗灰
1	0	0	1	亮兰
1	0	1	0	亮绿
1	0	1	1	亮青
1	1	0	0	亮红
1	1	0	1	亮品红
1	1	1	0	黄
1	1	1	1	白

彩色文本方式的显示属性字节如表 6-5 所示。

表 6-5 中，位 0～3 表明前景颜色（见表 6-4 ），位 4～6 表明背景颜色。位 7 (BL) 为 0 表示字符不闪烁，位 7 为 1 表示字符闪烁。

彩色方式下，字符图形显示程序的编写与黑白方式类同。

表 6-5

位号	7	6	5	4	3	2	1	0
属性选择	BL	R	G	B	I	R	G	B

四、图象显示

在图形方式下，每一图象由一组点形成，这个方式允许我们在屏幕上画出专用点。

这个方式可提供两种分辨率。在高分辨率方式中可生成640个水平点乘200个垂直点的图象，每个点可以是黑的或白的；在中分辨率方式中，可生成320个水平点乘200个垂直点的图象，每个点可以是黑白的或彩色的。两种分辨率下，垂真点数是相同的，但是只有在中分辨率下才可以生成彩色点。

在中分辨率方式下，每一点可以予选为四个颜色之一。第一个颜色可以从表 6-4中选择，这叫做背景颜色，显示点的颜色可以在彩色组 0（（1）绿、（2）红、（3）黄 ）或彩色组 1（（1）青、（2）品红、（3）白 ）中选择。同一时刻屏幕上只能出现四种颜色。

若要在屏幕上显示彩色图象，可以利用10H中断调用。利用10H中断调用的 0号功能，可以将显示方式，设置为320×200彩色图形方式，利用 0BH号功能，可以设置背景颜色或

选择彩色组；利用 0CH号功能可以在屏幕上写彩色点，这个彩色点的颜色只能在指定的彩色组中选择。下面给出一段简单程序。这段程序可以在屏幕上自0行0列开始画一条下斜的彩色直线。

```
CODE SEGMNET
STA  PROC FAR
     ASSUME CS: CODE
     PUSH DS
     MOV AX , 0
     PUSH AX
     MOV AH , 0
     MOV AL , 4
     INT 10H          ; 设置为320×200彩色图象方式
     MOV AH , 0BH
     MOV BH , 0
     MOV BL , 0FH
     INT 10H          ; 设置背景颜色
     MOV AH , 0BH
     MOV BH , 1
     MOV BL , 0
     INT 10H          ; 选择彩色组0
     MOV DX , 0
     MOV CX , 0
LOP: MOV AH , 0CH
     MOV AL , 02
     INT 10H          ; 写彩色点(AL=1为缘，2为红，3为黄)
     INC DX
     INC CX
     CMP CX , 200
     JNE LOP
     RET
 STA ENDP
CODE NEDS
     END STA
```

上述程序比较简单，但是从这里出发，利用各种算法和编程技巧，可以在屏幕上绘制出各种彩色图形，例如显示熊猫、大象、山水画等。

第七节　声音与乐曲

一、声音的产生

IBM PC 的主机箱上装的一只小喇叭，由定时器8253和并行接口芯片 8255(IBM PC XT 是 8255A) 控制其发音。主机板上有喇叭的控制驱动电路。如图 6-9。

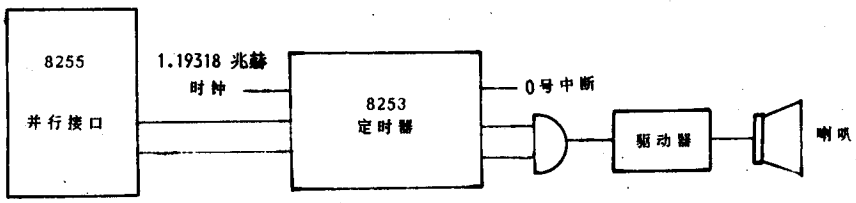


图 6-9 喇叭控制驱动电路

在 ROM BIOS 中有一个称为 BEEP 的过程（它的程序清单在 IBM PC 或 IBM PC XT硬件技术手册的附录 A中)。BEEP 过程根据 BL 中给出的时间计数值控制 8253 定时器产生一个或几个 500ms、约 1000Hz(实为 896Hz) 的脉冲信号，此信号经 8255 芯片（端口地址为 61H)控制喇叭的接通与断开，使其发出长或短声音。关于 8253 和 8255 的详细介绍请见参考资料(6)、(8)、(12)。

二、演奏乐曲

BEEP 过程只能产生896HZ的声音，且声音的持续时间只能是 500ms的倍数。若演奏乐曲，应能产生任一音频的声音，并且持续时间容易调整（例如可以是 10ms的倍数)。我们可以利用并改造BEEP过程。BEEP 是将计数值 533H送给定时器 8253的通道2而产生 896Hz 声音的，那么产生其它频率声音的计数值可用比例方法计算：

533H×896÷给定频率 = 1234DCH÷给定频率。

也可以直接用定时器时钟1193180HZ计算计数值：

1193180÷给定频率 = 1234DCH÷给定频率

假设给定频率在DI中，可用下述程序产生对应的常数。

```
MOV DX，12H
MOV AX，34DCH
DIV DI
```

为了不使除法产生溢出，限制 DI 中频率不小于 19Hz，一般音符的频率不会如此低。

10 ms 延时可由程序 DL10ms实现，这个软件延时程序前边已用过多次。如：

```
        MOV  CX，2801
DL10ms: LOOP DL10ms
```

现在我们编写一段程序，让它产生任何音频、持续时间是10mS的倍数的声音，详见下面例题中的SOUND过程。SOUND 的频率范围是19Hz～65535Hz(由DI决定)，上限实际是多余

的，因为人的耳的最高辨听频率是 20000Hz。这个过程利用BX内容控制声音的持续时间，BX 从 1 变到 65535，对应时间是 0.01 秒 ～ 655.35 秒，BX=0 时对应 655.36 秒。

利用SOUND过程，就可以编写演奏乐曲的程序。下面给出音律表的一部分，见表 6-6。

演奏乐曲的程序中需要定义两组数据：一组是频率数据，一组是节拍时间数据。频率可由表 6-6 查得。节拍时间取决于速度和每个音符持续的节拍。在 4/4 中，每小节包括 4 拍，全音符持续 4 拍，二分音符持续 2 拍，4 分音符持续一拍，8分音符持续半拍等。

表 6-6

音名	c	d	e	f	g	a	b	c'	d'	e'	f'	g'	a'	b'	c''
唱名	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1
频率	131	147	165	175	196	220	247	262	294	330	349	392	440	494	523

例题：图 6-10 给出了乐曲“两只老虎”的简谱和五线谱，其中大多数是四分音符，其次是二分音符和八分音符，现为全音符分配一秒时间，那么可以得到此乐曲对应的节拍时间数据。详见下述程序。

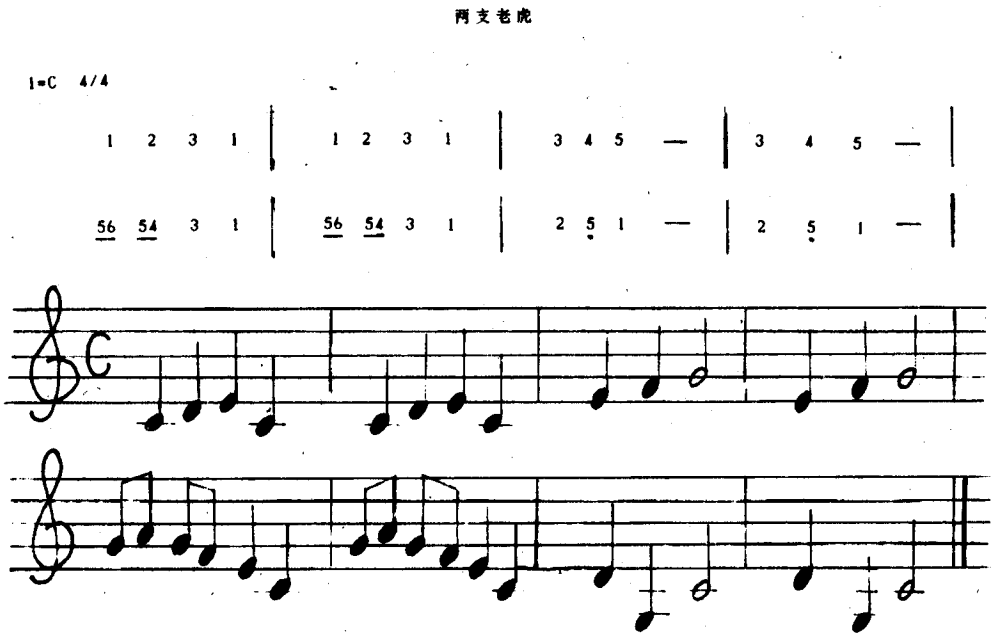


图 6-10 乐曲实例

下面给出可以演奏上述乐曲的程序。数据段中定义了频率数据（FREQ）和节拍时间数据（TIME），0000H 作为频率数据结束标志。代码段中，演奏过程（SING)要求把频率数据

的首地址送到 SI ，节拍时间数据的首地址送到 BP。演奏过程中多次调用 SING 过程。

```
STACK   SEGMENT
        DW   100 DUP(?)
STACK   ENDS
DATA    SEGMENT
BG      DB 0AH,0DH,"TWO TIGER : $"
FREQ    DW 2 DUP(262,294,330,262)          ; 频率数据
        DW 2 DUP(330,349,392)
        DW 2 DUP(392,440,392,349,330,262)
        DW 2 DUP(294,196,262),0
TIME    DW 10 DUP(25),50,25,25,50          ; 时间数据
        DW 2 DUP(12,12,12,12,25,25)
        DW 2 DUP(25,25,50)
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA
STAT    PROC FAR
        PUSH DS
        MOV  AX,0
        PUSH AX
        MOV  AX,DATA
        MOV  DS,AX
        MOV  DX, OFFSET BG                 ; 显示歌名
        MOV  AH,09
        INT  21H
        MOV  SI, OFFSET FREQ
        MOV  BP, OFFSET TIME
        CALL SING                          ; 调用 SING 过程
        RET
STAT    ENDP
SING    PROC  NEAR
        PUSH DI
        PUSH SI
        PUSH BP
        PUSH BX
```

```
      REPT:  MOV DI,[SI]
             CMP DI,0
             JE END_SING
             MOV BX, DS:[BP]
             CALL  SOUND
             ADD SI,2
             ADD BP,2
             JMP REPT
END_SING:    POP BX
             POP BP
             POP SI
             POP DI
             RET
  SING       ENDP                                              15
  SOUND      PROC  NEAR              ; 声音过程
             PUSH AX
             PUSH BX
             PUSH CX
             PUSH DX
             PUSH DI
             MOV AL,0B6H             ; 8253  初始化(选通道2，产生方波信号)
             OUT 43H,AL              ; 43端口是8253的命令寄存器
             MOV DX, 12H             ; 计算时间常数
             MOV AX, 34DCH
             DIV DI
             OUT 42H,AL              ; 设置时间常数
             MOV AL,AH
             OUT 42H,AL
             IN  AL,61H
             MOV AH,AL
             OR  AL,3
             OUT 61H,AL              ; 开喇叭(8255  I/O 端口61H的低两位置1)
  DELAY:     MOV CX,2801             ; 延时
  DL10MS:    LOOP DL10MS
             DEC  BX
```

```
        JNZ   DELAY
        MOV   AL, AH
        OUT   61H, AL          ; 关喇叭
        POP   DI
        POP   DX
        POP   CX
        POP   BX
        POP   AX
        RET
SOUND   ENDP
CODE    ENDS
        END     STAT
```

习题六

1. 使用指令 REPNZ SCASB 时，请问：

(1) 要求什么初始条件 ？

(2) 指令完成什么功能 ？

(3) 指令结束的条件是什么 ？

2. 使用指令 REPZ CMPB 时，回答上述问题。

3. 利用 REP STOSB 指令编一段程序，将自 BUF 开始的 100 个连续单元初始化为 0 ，要求程序结构完整。

4. 编程序将自FIRST单元开始的数据块传送至自SECOND单元开始的区域中去，但若发现被传送数据为0，则结束传送，分两种情况编程序：

(1) 两数据区域不重迭

(2) 两数据区域有重迭 ，SECOND单元可能在FIRST之前，也可能在后。

5. 内存中自 AREA1 单元开始连续存放了 100 个已排好序的无符号字节数，编程序将其传送到自 AREA2 开始的单元中。要求传送后的数据不重复出现。

6. 为本章第二节中例3画程序流程图。

7. 为本章第二节中例4画程序流程图。

8. 在 AH 存放着组合的 BCD 码十进制数，要求：

(1) 将 AH 中的数转换成二进制数。

(2) 求 AH 与 AL 中数的和，并将结果转换成 ASCII 码，然后在 CRT 上显示出来。

9．编写一个程序，接收键盘输入100个十进制数字，按下字符 T，则停止输入；并利用本章第三节中例2中的加密密码表，对输入的数字加密后存到内存缓冲区BUFR。

10．内存缓冲区 BCDBUFR 中存放着 10 个字节组合的十进制数，编程序求这 10 个数的和，结果送 SUM 缓冲区（占两个字节）。

11．本章第四节的 32 位数除法例题中，若被除数和除数都为有符号数，应如何修改该程序？把程序写完整。

12．选择几个图形字符组成一个你所喜欢的图形，编程序让这个图形在屏幕的某一行上循环移动。

13．编程序调用本章第六节中的过程 SING 和 SOUND ，演奏一曲你所喜欢的乐曲。

14．编写在图形方式下显示一幅彩色图画的程序。

第七章　软件开发

前几章主要介绍了用流程图方法进行程序设计的问题。然而从一个具体任务的提出，直到把任务变成程序，形成一个工作系统，这样一个软件开发过程包括很多阶段。一般要经过系统分析、系统设计，而后才进入程序设计阶段。这里不准备作全面介绍，着重谈谈程序设计问题，顺便涉计到系统分析和系统设计的内容。主要谈及的有以下几个方面：

. 问题定义
. 程序设计
. 编写程序
. 查　　错
. 测　　试
. 文件编制
. 维护和再设计

这几方面对于构造一个工作系统都是重要的。下面逐一介绍。

第一节　问题的定义

问题的定义指的是根据任务对计算机提出的要求来描述该任务。例如，让一台计算机去控制一台机床，或控制数台智能化仪器。问题定义要求你决定输入和输出的形式及速率，所需处理的量和速度，以及可能的错误类型及其处理方法。问题的定义为构成一个计算机控制系统建立了笼统的概念，并明确了任务和它对计算机的要求。

一．定义输入

着手定义先从输入开始。首先应列出在此项应用中计算机可能接收的所有输入，包括反映外设状况的状态信息和来自外设的数据。然后，对每一个输入考虑以下问题：

1.输入形式：输入到计算机的信号可能是开关量、数字量或模拟量，它们都应通过适当的接口电路与计算机相接，而模拟信号是要先经过模／数（A/D）转换后接入计算机。

2.输入何时准备好：计算机如何知道它已准备好，处理器是否用一选通信号请求输入等。

3.输入可供使用多长时间，要否锁存，变化周期如何，处理机如何配合其变化。

4.输入数据的多少，次序是否重要。

5.数据中若有错误如何处理？可能是数据错、传输错、顺序错等。

6.本输入与其它输入输出间的关系。

二．定义输出

首先应列出计算机必须产生的所有输出，包括需要输出的数据和控制信号。要输出模拟信号应先经数／模(D/A)转换。然后，对每个输出考虑如下问题：

1.输出形式：从计算机输出的开关量或数字量要经过适当的接口电路或D/A 转换再送到外设。

2.输出应何时准备好，外设如何得知它已准备好。

3.输出信号维持多长时间，要否锁存／缓冲，多长时间变化一次，外设如何得知其变化。

4.输出数据的多少，次序是否重要。

5.应采取什么措施避免错误，检测和排除外设故障。

6.本输出与其它输入输出间的关系。

三．处理阶段

输入和输出之间的阶段是处理阶段。这儿涉及的问题有：

1.由输入数据而得到输出结果的基本过程是什么？需要什么算法，应使用什么标准程序或表等。

2.对处理时间（或运算速度）有什么限制，内存有什么限制。

3.运算精度的要求。

4.程序应安排对处理错误、结果溢出及特殊情况的处理。

四．错误处理

错误处理在很多应用项目中是一个重要内容。设计者必须为排除一般错误或故障诊断而做好准备。在定义阶段，设计者就应考虑如下问题：

1.系统可能会发生哪些错误。

2.将错误分类，分别处理。对系统有威胁的错误，重点处理；由操作人员产生的人为差错最为常见；其次是通信传输错误，而机械、电气、处理机错误较为少见。

3.系统如何以最快时间和最低损失来发现和处理这些错误。

4.为现场工作人员提供查找故障或排除故障的方便。对此，内部测试程序、专用诊断程序或特征代码分析会有帮助。

五．人机关系

在整个研制过程中必须考虑人的因素，人与机器之间的关系。设计者应考虑到：

1.操作者使用机器是否容易，系统响应是否直观、清晰；系统操作是否简单方便。

2.操作人员如何得知操作正确与否，如何方便地查找与排除操作错误、过程差错及设备故障等。例如：给出提示、说明、帮助信息，人机对话等是有效的方法。

3.考虑不同水平操作员的需求，尽量提高系统效用和人的效能。例如：提供多种合理选择方案。

问题的定义与任何特定的计算机、计算机语言或开发系统无关，但它却能对特定应用

项目需要什么类型或速度的计算机，以及设计者能就硬、软件做什么类型的折衷，提供指导。

第二节　模块化结构化程序设计

这里所谈的程序设计是指制定计算机程序的纲要，这个程序将完成前面已定义的任务。在这个阶段，利用能容易地转变成程序的方式对任务作出描述。诸如常用的流程图法、结构化程序设计、模块程序设计和自顶向下程序设计等，都属于此阶段中的一些有用方法。

程序设计是把问题定义转化为程序的准备阶段。如果程序较小且简单，此阶段可能仅仅是绘制一张流程图。如果程序较大或较复杂，设计者就要考虑用较为完善的方法。

下面我们将讨论几种常用的程序设计方法，指出它们的优缺点。但并不推荐任何特定方法，因为没有证据说明某一种方法总比所有其它方法优越。可以根据不同的环境和要求选择或配合使用某一或某几种方法。如：

(1) 把大的作业分割成小的、逻辑上独立的若干任务。尽可能使它们相互独立，以便互不影响、分别测试。

(2) 把清晰性、简单性、直观性放在首位，以方便对系统的调试、维护等。

(3) 要尽量使用核对清单和标准过程。

(4) 在程序设计阶段未结束前，不要急于动手开始编程。那将欲速则不达。

一．流程图法．

前面几章中我们已着重介绍了流程图法，它也是人们最为熟悉的一种方法。流程图的基本优点在于它是一种图解表示方法。这种表示法比文字描述更醒目、直观。设计者可借以直接观察整个系统，了解各部分之间的关系，逻辑错误和不相容性往往跃然纸上而不是隐藏在其中。

然而，这种方法对较小任务或简单程序较为合适。对于复杂问题，遍及图上的线和箭头以及环路将呈现出杂乱，它同良好的结构设计原则相对立，是一种非结构设计。再者：

1.流程图除简单情况外，难于设计、作图或更改。

2.没有查错和测试流程图的简易方法。

3.流程图只表示出程序的组织结构，表示不出数据的组织结构或输入／输出模块的结构。

4.流程图对硬件或定时问题无帮助，也不能就这些问题可能发生在什么地方提供线索。

二．模块程序设计

一旦程序变得大而复杂，流程图就难以应付。把整个程序分成若干子任务或模块，称为＂模块程序设计＂。显然前几章中介绍的大部分程序，都是大系统程序中的一些模块，设计者在模块程序设计中面临的问题是：如何把程序分成模块，以及如何把模块装配在一起。

1.模块程序设计的优点是：

(1) 单个模块易于编写、查错、调试和修改。

(2) 一个模块可以在多个地方或多个程序中使用，有的可以写成标准模块。

(3) 程序员可直接利用已有的模块。

(4) 整个程序功能的修改或错误查除，可以局限到某一模块进行。

2.模块程序设计的缺点是：

(1) 有些程序很难模块化，即难以分割成几个独立的模块，而且突出的问题是没有经证实的、系统化的程序模块化方法，只有一些原则可参考。

(2) 各模块的装配也是难题，特别是这些模块由不同人编写时。

(3) 模块程序往往需要额外时间和存贮器。因不同模块间可能会有部分功能是重复的。

(4) 单独调试一个模块，需额外地为其准备调试数据、专用程序等。

三．结构程序设计

如何使各模块截然分开并防止它们相互作用，如何编写一个操作顺序分明的程序，使你能检查和纠正错误呢？回答是利用称之为 "结构程序设计" 的方法。七十年代初，Boehm 和Jacobi 提出并证明了结构定理：任何程序都可以由三种基本结构程序构成结构化程序。这三种结构是：顺序结构、条件结构和循环结构，每个结构只有一个入口和一个出口。三种结构可以任意组合和嵌套。

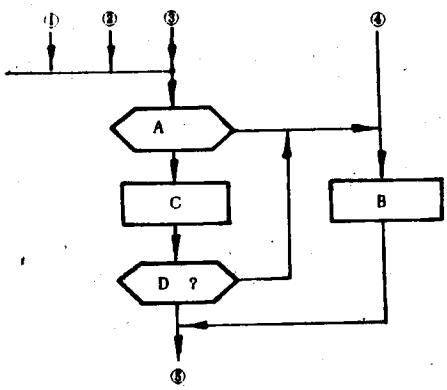


图 7-1 非结构化程序流程图

图 7-1 所示的是一个非结构化程序的流程图。如果在模块 B 中发生一个错误，该错误的产生有5个可能的源，这就使得查错及纠正变得复杂而困难。解决的方法是采用结构程序设计。

1.顺序结构，如图 7-2 所示：

在顺序结构中的语句或结构被连续执行。图 7-2 中 S1、S2 可以是单一的指令、语句或整个程序。

2.条件结构

这种结构的常见形式是："IF C THEN S1 ELSE S2"，其中 C 是条件，S1 和 S2 是语句或语句序列。如果 C 是真，计算机执行 S1，如果 C 是假，则执行 S2。如图 7-3 (1) 所示；若 S1 或 S2 中有一项为空语句，则是这种结构的简单形式，如图 7-3 (2)。

此结构只有一个入口和一个出口。

3.循环结构

常见的循环结构是："WHILE C DO S"，其中 C 是条件，S 是语句或语句序列。计

算机检查 C ，若 C 为真则执行 S ，循环进行 ，若 C 的初始值为假 ，则计算机一次也不执行 S，即为零次循环。这种结构形式称作 "先判断，后执行" ，如图 7-4(1) 所示。

如图 7-4 (2) 中的结构形式则为 " 先执行，后判断 " ，循环次数至少为一。

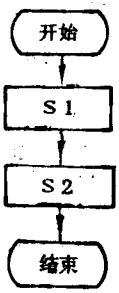
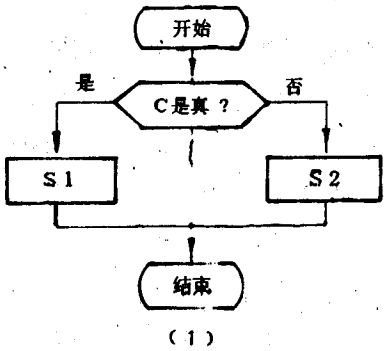


图 7 - 2 顺序结构流程图



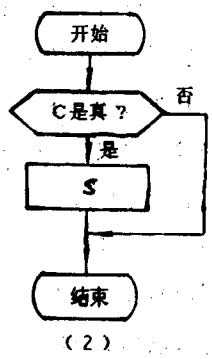
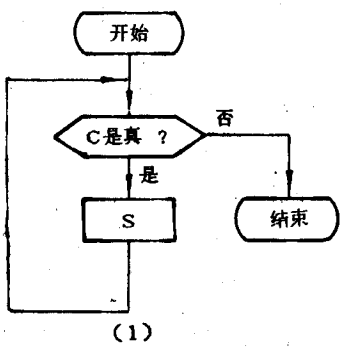


图 7 - 3 条件结构流程图



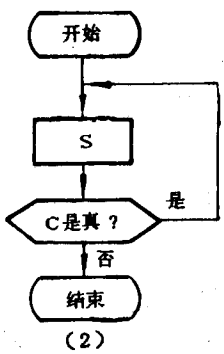


图 7 - 4 循环结构流程图

1 . 结构程序的优点

(1) 操作顺序易于跟踪，便于查错和测试。

(2) 三种结构是完备的，经理论证明的。术语是标准化的，易于阅读，易做成模块。

(3) 结构化程序易于用程序框图描述，有助于文件编制。

(4) 实践证明提高了程序员的工作效率。

2 . 结构程序的缺点

(1) 结构化程序比起非结构化，往往执行速度慢，占用存贮单元多。

(2) 结构程序只考虑程序操作的次序而不考虑数据流，因而，不适于数据处理。

(3) 很多程序员认为标准结构不灵活且受约束，不习惯于结构程序设计。

一般地高级结构程序设计在如下情况下是最有用的:

(1) 超过１０００条语句的大型程序。

(2) 存贮器容量并不紧张，但软件开发费用（特别是测试和检错费用）成为重要因素。

⑶ 涉及字符串处理、过程控制或其它算法而不是简单位处理的应用场合。

硬件价格在下降，程序规模在增大，软件开发费用将会增加，因而结构程序设计会越来越有价值。

四．自顶向下程序设计

余下的问题是如何检查和组装模块或结构。我们把一个大任务分成若干个子任务，但如何分别地检查这些子任务，并把它们结合在一起，标准过程称之为"自底向上设计"，这种过程要求额外的测试和查错工作，并把整个组装任务留待末尾进行。我们需要的是能在实际程序环境下进行查错和测试，以及使系统组装也变成模块化的方法。此方法就是"自顶向下设计"。

这里我们从编写总管理程序开始。用一些程序"根"来代替未实际完成的子任务，这种程序根是暂时性的。然后我们测试该管理程序，检查它的逻辑是否正确。

例如：检查一个开关的状态，若是处于闭合状态，则让指示灯点亮一段时间。程序可如下安排：

```
SWITCH=OFF                          ; 循环初始条件
DO WHILE SWITCH=OFF
READ SWITCH                         ; 读开关状态
END
LIGHT=ON                            ; 开指示灯
DELAY                               ; 延时
LIGHT=OFF                           ; 关指示灯
```

这里 READ SWITCH 和 DEALY 都可以看成是暂时性的程序根。我们从扩展各程序根着手，将程序自顶向下一层层展开，逐步求精。每个程序根常常包含若干子任务，也可再用一些程序根暂时代替。例如 READ SWITCH 可以根据问题的定义，从某个输入端口读入信息，然后屏蔽无用信息，保留开关状态，这样就可以用具体程序代替。此扩展、查错和测试过程一直进行到所有程序根被一些可工作程序取代为止。在每一级上都有测试和组装，而不是全部集中到末尾进行。并不需要特殊的驱动程序或数据生成程序。这样我们可清楚地知道我们在设计中到底处于何处。自顶向下的设计方法采取模块程序设计，但它同结构程序设计兼容。

自顶向下设计的缺点是：可能难以得到通用模块，难以充分利用现有软件，某些程序根难以编写，顶层出现的错误可能会有不良影响等等。自顶向下与自底向上的方法往往配合使用。

在程序设计阶段，上述几种方法可帮助你系统地规定程序的逻辑，并将它编成文件。模块程序设计要求你把总程序划分成小的、个别的模块。结构程序设计提供了规定这些模块逻辑的系统化方法。而自顶向下的设计法提供组装和测试它们的系统方法。当然并不强迫遵循这些方法，事实上，它们主要是指导性的。

五．编写程序

到目前为止，我们还没有涉及任何特定的微处理机或汇编语言，也没有编写一行实际代码。虽然我们常把用计算机指令编写程序看成软件开发的一个关键部分，实际上它是最容易的阶段之一。

一旦问题已经定义，程序设计阶段已经结束，编写程序就变得容易了。首先就是熟悉所用机器的指令系统，然后我们在前几章中讲述的内容，在这里可以充分利用。尽量多用一些现有的、公用程序完成自己的具体程序。

第三节　查错和测试

查错和测试是软件研制中最花时间的阶段，而且也是很重要的阶段，不能忽视。

一．查错

查错的能力主要结合上机去提高，查错的基本方法在第八章结合程序的调试和运行做了介绍。调试程序DEBUG 为查错提供了极大的方便。主要的是单步操作和断点设置，并配合寄存器与存贮器内容的显示、打印与检查。

单步操作允许你每次执行一步程序。这时可以检查机器执行的结果是否正确，包括检查机器和输出线的状态、有关寄存器或存贮单元的内容是否正确等。单步操作很慢，而且不能检查定时错误、中断或 DMA 系统的错误。

所谓断点就是程序执行到断点处便自动停顿或进行等待，使用户可以检查系统的现实状态。断点设置使得人们可以检查或通过一整段程序。它不影响定时，可以检查输入或输出中断。

单步操作与断点设置互相配合可有效地检查错误所在，用断点设置确定错误范围，单步确定具体错误点。

另外还有一些更先进的查错工具，例如各种软件模拟程序可以自动跟踪程序的执行，代替人工对程序检查。各种用于检验信号和时序的逻辑分析仪（或称微处理机分析器）是解决查错问题的硬件工具，主要用于复杂时序的系统。

二．测试

测试主要是对程序性能的测定和检验。它与查错是紧密相关的。

用于查错的工具也适于测试，另外诸如I/O 模拟、联机仿真器、测试程序、实时操作系统等也有助于测试。

测试中一个重要的问题是测试数据的选择。用于测试的数据应有代表性，尽可能考虑到实际遇到的各种类型数据，选择一组数据样本，才能对程序的性能进行尽可能客观的测试。

通过查错和测试可以证明程序中有错误，不能证明程序的正确性。

第四节　文件编制与维护

一．文件编制

工作程序不是软件开发的唯一要求，适当的文件编制是软件产品的重要部分。文件编制不仅有助于设计者进行测试和排错，而且对以后的使用和扩充也是必不可少的。文件编得不好，程序就难于维护、使用和扩充。

文件编制包括以下几方面：

1.在编写程序时就尽量使之自成文件。这需要遵循以下规则：

．程序结构清晰，转移尽可能少。

．使用有含义的名字和标号。

．I/O 设备、参数、数值因子等用名字代替。

．强调简单而不一味强调节省内存和执行时间。

．简明清晰有效的注释。在一段段程序的开头或中间加好功能注释和说明是非常重要的，好的注释使得你不看程序只看注释就能知道程序的大概。而未加注释的程序，即使程序员自己也阅读困难。

2.划出存贮器分配图。它使你对存贮分配情况一目了然。它帮你决定存贮量、数据区、程序区、缓冲区和未分配区。方便地寻找存贮单元和程序入口点，容易地扩充和维护所需要的数据和子程序等。

一个典型的存贮器分配图如表 7-1 所示 。

表 7-1

程序存贮区		
地　址	过　程	含　意
0000--0002	RESET	控制转到主程序的40H 单元
0040--0265	MAIN	主程序
0270--027F	DELAY	延迟程序
0280--0290	DSPLY	显示控制程序
数据存贮区		
1000	NKEYS	键数

1001--1002	KPTR	键盘缓冲区指针
1042--1051	DPFR	显示缓冲区
1052--105F	TEMP	暂存区
10E0--10FF	STACK	RAM 堆栈

3.在每个程序和子程序的开头列出参量、标号和名字的清单，对理解和修改程序很有用。一个典型例子如下：

```
;   存贮系统常数
RESET EQU 0              ;   复位地址
INTRP EQU 38H            ;   中断入口
START EQU 40H            ;   主程序起始地址
;   I/O 设备
DSPLY EQU 0E0H           ;   显示器的输出PIO
KBDIN EQU 0E1H           ;   键盘的输入PIO
;   RAM 单元
          ORG RAMST
NKEYS     DS   1         ;   键数
KBDPTR    DS   2         ;   键盘缓冲区指针
DSPBFR    DS   10H       ;   显示数据缓冲区
;   参量
BOUNCE    EQU  2         ;   消除弹跳时间（单位 ms）
GOKEY     EQU  10        ;   'GO'键识别
;   定义
ALL1      EQU  0FFH      ;   全1的信息图样
STON      EQU  80H       ;   启动转换脉冲
```

4.程序库。一个个标准的子程序文件可以组成有用的程序库。标准库形式也应该包括一个总流程图和程序说明。

总之完整的文件可以涉及：

- 总流程图
- 程序的书面说明
- 所有参量的定义清单
- 存贮器分配图
- 成文的程序清单
- 测试计划和测试结果的说明

文件还可以包含：

. 程序员的流程图

. 数据流程图

. 结构程序

对于生产软件甚至需要更多的文件编制工作。包括：

. 程序逻辑手册

. 用户指南

. 维护手册

程序逻辑手册应阐明：系统设计目标是什么，所用算法和所做的折衷权衡。采用什么数据结构。所用表格和图，还应包括代码转换图、状态图、翻译矩阵和流程图等。

用户指南应给所有用户提供系统介绍、系统特性及其使用的详细说明。可使用大量例子逐步引导用户尽快掌握使用此软件。任何好的软件不易使用或未能有效使用都是无意义的。

维护手册是为维修、修改系统的人员而写的。它应概述重新配置系统的逐步规程，以及代码中为以后扩充所作的安排等。

文件编制应改在软件研制的每一个阶段连贯而完善。

二．系统维护和再设计

硬件需要维护，软件也需要维护、改进和扩充。程序运行一段时期后，如果发现其中有个别错误就需要修改；要提高程序功能以适应新任务就需要改进或扩充；为提高程序运行速度或减少存贮空间也需要改进等，以推出新版本。

如果要作的改动很大，或某些方面不合理，就可能需要再设计。这往往要付出较大代价。这时侯，最好从软件开发的各个阶段作全面考虑，以取得较大提高。

以上就软件开发的几方面作了简单介绍，详细内容请看有关资料。

习题七

一．问题的定义包括哪几方面的内容？就每一方面加以概述。

二．为什么要采用模块化程序设计，其优点是什么？

三．试述程序的结构原理，并结合流程图分别描述程序的三种基本结构形式。

四．查错与测试有何异同？各采用什么工具进行？

五．程序已经编写完毕，而且调试成功以后，为什么还要进行文件编写？

第八章　汇编语言程序的上机过程

第一节　基本概念

当我们上机操作，在机器上建立和运行汇编语言程序时，必然要与磁盘操作系统(DOS)打交道。关于 DOS 的操作与结构、其主要命令等可参阅《IBM PC 磁盘操作系统》等书。这儿只就建立和进行汇编语言程序中遇到的一些基本概念和DOS 的最基本命令作一简要介绍。

这里假定使用的机器具有5 英寸软盘驱动器或硬盘，使用 PC　DOS 2.0或 2.1 版本。DOS本身是作为文件存放在磁盘上的。

一．文件、文件名、文件目录

1.　文件

文件是具有名字的一维连续信息的集合。这些信息可以是各种程序——用各种语言写的源程序、目的程序、公用子程序等等；　也可以是数据，　可以是程序执行中所需要的数据，或程序建立起来的数据；也可以是文本，例如一封信、一个通知单、一篇文章等等。

目前，文件通常存放在磁盘上（ 软磁盘或硬磁盘 ），一个　5 英寸软盘的外形如图 8－1所示。

以塑料为基片两面复盖着磁性介质的圆形软磁盘，始终放在方形的黑色保护套内。当盘片插入驱动器后，驱动器的电机通过离合器，使盘片在黑色封套内旋转。黑色封套上开有一个读写槽，磁头就通过这个槽对磁头表面进行读写。磁头可以沿着槽的方向移动，这样就在磁盘表面形成一个个同心圆，如图（8－2）所示。

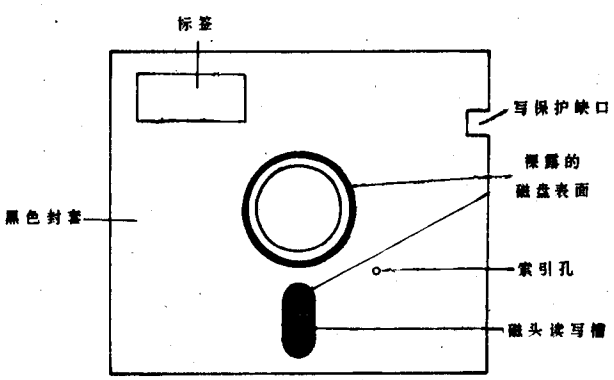


图 8-1 软磁盘外形图

这一个个同心圆就称为磁道（TRACK），信息就存放在磁道上。为了便于信息的存放，把磁盘分成一个个扇区或叫区段(SECTER)。

PC　DOS 2.00 版本，把 5 英寸软盘的一面分成0－39共40个磁道，每个磁道分成9个扇区，每个扇区能存放 512 字节 。因此，一个单面软盘能存放 184320 个字节；

双面软盘能存放 368640 个字节（即360KB）。

磁盘上信息的读写，是以一个扇区作为基本单位的。从物理上来讲，只要知道了信息是存放在哪个面（对于双面；盘面号分别为0或1）、第几磁道、哪个扇区，就可以进行读写。这就是扇区的三维地址；若是单面盘，则只要有磁道号和扇区号就可以确定某一扇区，这就是两维地址。

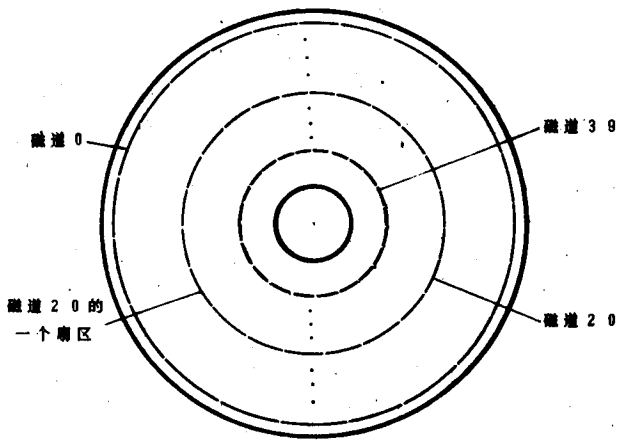


图 8-2 磁道与扇区

为了便于访问，在 DOS 中采用扇区的逻辑地址——相对扇区号，把盘上的所有扇区统一编号。0面0磁道1扇区（磁盘上的第一个扇区）的相对扇区号为0，接着是0面0磁道2扇区直到0面0磁道9扇区，相对区号为0～8；然后是1面0磁道1至9扇区，相对区号为 9～11 (H)；接着是0面1磁道1至9扇区，相对区号为 12～1A (H)・・・。

只要知道了某一扇区的相对扇区号，经过换算（由DOS 实现）就可以确定此扇区的三维物理地址，就可以进行读写。

由于盘上的存贮容量较大，用户要用物理地址或逻辑地址直接进行读写操作是很不方便的，也是有危险的。因为这样用户必须了解盘上文件的分布情况，哪些区已用，哪些区未用，否则有可能把别人的信息给冲掉了。这样也不利于信息的保密和共享。同时直接对扇区进行读写的程序也比较复杂。

为了方便用户，在 DOS中有一个文件管理系统。对文件进行管理。这样用户就可以直接用文件名和通过简单的命令对文件进行读写，而根本不用了解文件在盘上的物理位置，也不用去编写较复杂的磁盘读写驱动程序。

而文件与文件的物理地址之间的联系，是通过文件目录来实现的。

2. 文件名

DOS 2.00 对文件名的规定如下：

(1) 单义文件名

一个文件的名字（file name）由两部分组成：文件名（file name）和扩展名（extention）。

① 文件名由1至8个字符组成。这些字符可以是：

. 英文字母

. 数字0～9

. 以下这些特殊字符：

$ # & @ % (!) _ { } ' 等。

② 扩展名

一个扩展名由句号（.）开始，可由1至3个字符组成，它们跟随在文件名之后，扩展名常用来说明一些文件的特征，例如：

.ASM 表示是汇编语言的源程序

.OBJ 表示是汇编后的目标程序

.LST 表示是汇编后的可打印文件

.EXE 是可执行的机器码文件

.BAS 表示是BASIC 语言的源程序

.FOR 表示是FORTRAN 语言的源程序等等。

一个文件的名字中可以没有扩展名，但若用了扩展名的话，则在调用时必须包含扩展名。

⑵ 文件标识符(File Specification)

为了读写文件，除了文件名和扩展名外，还必须告诉 DOS 文件在哪一个驱动器的盘上。也即要指定驱动器标识符。驱动器标识符是一个字母和冒号，如A:，B:。

驱动器标识符加上文件名和扩展名，这三部分构成了一个文件标识符。文件名和扩展名必须紧跟在驱动器标识符之后。如：

A:ADDRLIST.BAS

若在文件的标识符中，不指定驱动器标识符，即是指文件在当前（现行）驱动器上，或称为隐含的驱动器(Default Drive)。

当系统启动以后，把A驱动器作为当前驱动器(Default drive)。但可以用以下的命令来改变当前驱动器。

A>B:

B>

当 DOS 启动以后，输入了适当的日期和时间以后，屏幕上显示 A>，这是 DOS 提示符，且表示驱动器A为当前驱动器。

但若打入B:，则出现提示符B:，表示已改为驱动器B作为当前驱动器。可以用以下命令改回来。

B>A:

A>

⑶ 多义文件名

在文件名中可以引入?和*号以形成多义文件名。所以多义文件名可以代表一组文件。

① ？号

可以代表它所在位置上的任意字符。

② * 号

若在文件名部分有一个* 号，则它可代表它所在的位置及文件名余下的部分的任意个

任意的字符；若在扩展名中有一个* 号，则它可代表它所在的位置及扩展名余下的部分的任意个任意的字符。即* 号相当于打入若干个? 号。

若在某一磁盘的目录中有以下文件：

PROGRAM1.FOR

PROGRAM2.FOR

PROGRAM1.OBJ

则 PROGRAM?.FOR代表 PROGRAM1.FOR

PROGRAM2.FOR

而 PROGRAM1.* 代表 PROGRAM1.FOR

PROGRAM1.OBJ

. 代表全部文件。

3. 文件目录

文件目录本身形成一个文件，存放在磁盘上。此文件中包含了该盘片上所有文件的文件名及扩展名，文件的一些特征，以及此文件在盘片上的分配信息。一个文件的上述信息构成一个目录项。

一个目录由 32 个字节组成，其分配如图 8-3 所示：

第0个字节：

若为00H 表示此目录项从未使用过。

若为E5H 表示此目录项已被删除。

若为2EH 表示此项为一个目录。

若为其它字符，则为文件名的第一个字符。

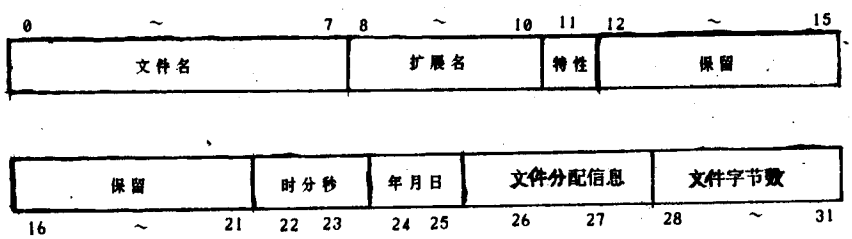


图 8-3 目录项结构

第11字节是文件的特性字节：

若为01H 表示文件是只读的。

若为02H 隐藏文件。此文件排除在正常的目录搜索之外。

若为04H 系统文件。此文件排除在正常的目录搜索之外。

若为08H 这一目录项包含卷号的前11个字节，没有其它的有用信息。

若为10H 这一目录项定义一个子目录，且排除在正常的目录搜索之外。

若为20H　归档位。当文件已经写入和关闭时，此位置位。

当一个文件建立(Create)时，就在目录文件中建立一个目录项。

每当我们要建立一个文件，打开文件，读写文件时，必须先在内存中，通常在DS:5CH和 DS:6CH 处，建立一个格式化的文件控制块－－有8个字节放文件名（若所用的文件名不是8个字符，则后面补空格），有三个字节放扩展名。

当我们在建立文件时，就用此文件控制块，在目录文件中开辟一个目录项。

当打开文件和读写文件时，就用文件控制块到目录文件中去查找文件。

4．目录结构

DOS 2.0 以前的版本是采用简单的目录结构，即每个盘只有一个目录，这个目录文件中最多可以有 64 个（单面盘)或 112 个(双面盘)目录项。 DOS 2.0 是支持硬盘的，采用一种树形目录结构，如图 8-4 所示。

最上边的一层为根目录，其中可以包含文件，也可以包含别的目录（称为子目录），子目录中又可以包含文件和别的子目录，这样一层层展开。

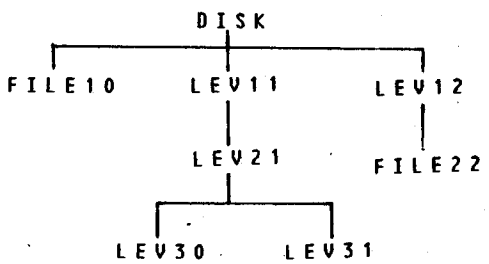


图 8-4 树形目录结构

磁盘格式化时建立了一个目录文件，这称为根目录（ Root　Dirctery ）或系统目录（ System Directory)。 DOS 启动时，自动把根目录当作当前目录。树形目录中要寻找一个文件，通常应该指出文件标识符和文件所在的目录名。若文件在当前目录中，可以省略目录名；若不在当前目录中，则必须指明文件的路径。路径由用 \分隔的一系列目录名组成，所要寻找的文件与所在目录名之间也用\分隔。如图 8-4 中，你要在子目录 LEV31中找文件 FILE31.LST ，该文件的路径名为：DISK\LEV11\LEV21\LEV31。

二． DOS 命令

1. DOS 命令类型

DOS 命令有两种类型：内部命令和外部命令。

内部命令是 DOS 内的命令处理程序完成的，当 DOS 启动后已调入内存，可以立即执行。例如：DIR,ERASE,COPY，RENAME，TYPE等等。

外部命令是以可执行的程序文件形式存于磁盘上 ，因此执行前要先从磁盘上读入内存，这就意味着存有该命令的软盘必须已在驱动器上，否则DOS 是找不到该命令的。

假若一个文件带有扩展名 .COM 或 .EXE ，这个文件表示是外部文件。例如 ：FORMAT.COM、COMP.COM 等都是外部命令。这就允许用户自己编写一些命令追加到系统中

去。当打入一个外部命令时，不必包含文件名的扩展名部分。

2. 几个常用的DOS 内部命令

(1) 查看目录命令DIR

通常在开始使用系统时，希望了解一下系统盘上有哪些文件、哪些程序等，就要用到DIR 命令。例如：

A>DIR B:

系统将列出B 盘上每一个文件的文件名，扩展名，文件长度，建立日期和时间，最后列出此盘上文件总数，以及剩余的磁盘自由空间字节数。

(2) 显示（/打印）命令 TYPE

在DOS 下，可用 TYPE 命令将磁盘上的某一个文件的内容显示在屏幕上或在打印机上打印出来(若打印机已联机)。注意，可用此命令打印的文件必须是由可打印字符组成的，即可是各种高级语言源程序或文本文件等。例如：

A>TYPE EXAMPE.ASM

将把A盘（当前盘）上文件(EXAMPE.ASM)的全部内容显示或打日印出来。

(3) 拷贝命令 COPY

把一个或多个文件拷贝成副本。副本可以改名，可以在另一个磁盘上，也可以在同一个磁盘上。例如：

A>COPY EX1.EXE B:

将把A盘的文件EX1.EXE 拷到B盘上，文件名不变。

A>COPY *.* B:

将把A盘上的所有文件拷到B盘上。

A>COPY A.XYZ+B.COM+B:C.TXT BIGFILE.TXT

将把 A盘上的文件 A.XYZ、B.COM 和 B盘上的文件 C.TXT连接在一起送入 A盘上的文件 BIGFILE.TXT中。

(4) 改名命令： RENAME

当需要把一个文件名改为另一个文件名时，可用改名命令。命令格式为：

RENAME [d:] 文件名 [.扩展名] 文件名 [.扩展名]

例如： A>RENAME EXA01.EXE EXA02.EXE

将把A盘上的文件 EXA01.EXE 改名为 EXA02.EXE 。此处改名后的文件仍保留在同一个盘上。

(5) 删除命令: ERASE （或DEL）

当需要从指定的或约定的驱动器上删除一个或多个文件时，可用此命令。

格式：

ERASE [d:]文件名. 扩展名

例如： A>ERASE A:EXAM1E.ASM

将删掉A盘上的汇编语言源程序文件EXAM1.ASM 。

又如： A>DEL A:*.*

将删除A盘上的所有文件，此时DOS 发出如下信息 ：

Are you sure (Y/N):

以便用户确认是否要真删除，若真，键入Y，否则键入N，则命令不被执行。

第二节 汇编语言程序的上机过程

当用户编制好汇编语言源程序之后，要在机器上运行，必须经过以下几个步骤：见图8-5。

1. 调用行编辑程序EDLIN 或全屏幕编辑程序 WORDSTAR 建立和修改源程序。

在任一种编辑程序下，通常用键盘打入源程序，此时源程序以 ASCII 码字符形式存放在内存缓冲区。若输入过程有错，可用有关命令修正。修正后，可以用命令使源程序存盘，于是在盘上就建立了一个源程序文件。

若源程序在以后的几个步骤中发现有错，还要在编辑程序下加以修改。这样，在盘上就有了源文件和它的备份（.BAK）文件（修改前的文件）。

2.源程序必须经过汇编，变成机器码的目标文件，机器才可运行。

汇编是通过调用在 PC DOS下的宏汇编程序 MASM 或小汇编程序ASM 实现的。

为了适应编制多模块组成的大程序和调用 PC DOS 支持下的公共子程序的需要，汇编以后的目标程序中的地址部分仍不是可执行的绝对地址，而是可浮动的相对地址。

3. 必须经过连接，把程序的各个模块连接在一起，或把要调用的子程序与主程序连接在一起，把相对地址变为绝对地址，形成可执行的文件。

连接是由调用 PC DOS下的 LINK 程序来实现的。

4 调试程序

经过了以上过程，在盘上有了可执行文件，则可在DOS 的提示下，直接打入文件名（不要扩展名），就可把执行文件从盘上装入内存，且立即执行此程序。

然而通常一个较复杂、较长的汇编语言源程序，希望一点错误也没有，一次通过的可能性是很小的，这样就需要调用PC DOS支

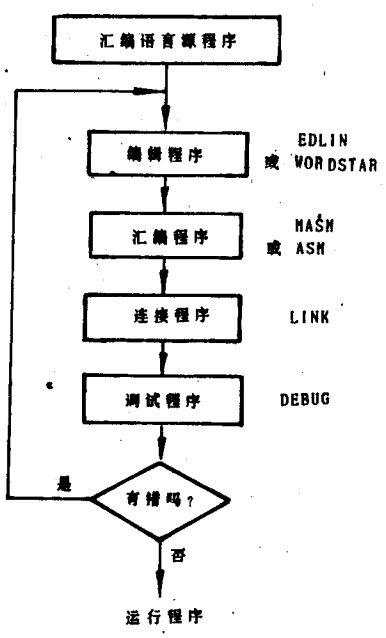


图 8-4 汇编语言程序的上机过程

持下的 DEBUG 程序调试我们的目标程序。在调试程序 DEBUG 的管理下，可以单步执行程序，也可以设置断点．可以显示和修改 CPU 内部寄存器和标志位的内容等，这样就便于寻找程序中的错误。

在发现了错误以后，通常还要重复上述的编辑、汇编、连接和调试程序的全过程，直至程序运行正确为止。

已调试正确的目标程序，任何时侯都可在 PC DOS 下，通过打入文件名而运行它。

为了便于上机操作，下面将把这个全过程中用到的主要内容分别加以介绍。详细内容可参阅 PC DOS 手册和宏汇编手册。

第三节 行编辑与全屏幕编辑程序

调用行编辑程序 EDLIN 或全屏幕编辑程序 WORDSTAR 都可完成源程序的建立和修改等工作。

行编辑程序与全屏幕编辑程序的根本区别是它们按不同的方式对源程序进行输入、显示、修改等工作。一个是按行方式，一个是按全屏幕方式。按行方式不够直观，操作麻烦。例如：当发现前面输入的行中有错误时，需先给出行号，将此行取出再供修改。若忘记了行号，则只好用显示命令去查找。全屏幕方式直观、方便。它将源程序一幕幕地显示出来，利用上下左右箭头键及翻页键可任意的将光标移到任一行任一字符位置，任意删、插、修改。不需考虑行号，很直观。以前人们习惯地使用行编辑程序建立ASCII 码文件（特别是汇编语言源程序的建立），而现在全屏幕编辑方式已在许多方面采用了，我们应该大力提倡应用它。况且 WORDSTAR 的功能很丰富，这里使用它，只不过利用了它的功能的一角。下面分别对两种编辑程序的功能和使用加以介绍。

一．行编辑程序 EDLIN (Line Editer)

1. 如何调用行编辑程序：

在 PC DOS 下（即在出现了DOS 提示符 A> 下）打入如下格式的命令：

EDLIN [d:] [PATH] FILENAME [.EXT]

其中 EDLIN 是行编辑程序的文件名，后面是要编辑的文件的文件标识符。PATH 是路径，.EXT是扩展名。[]括起来的是任选项。

在上述命令打入以后，机器可以有两种响应：

(1) 若命令中指定的文件存在，则DOS 在把行编辑程序调入后，在内存中建立一个编辑缓冲区，且把指定的文件从盘上调入编辑缓冲区后，显示：

END OF INPUT FILE

*-

* 号是EDLIN 的提示符，表明系统现在是在EDLIN 的管理之下，可以输入EDLIN 的各个命令（只有在出现* 后才能输入命令）。

(2) 若命令中指定的文件不存在，则DOS 在把行编辑程序调入后，在内存中开辟一个编辑缓冲区后显示：

```
NEW FILE
*-
```

此时可以输入各种命令，建立新文件了。

2. 主要命令

(1) 插入命令 (INSERT)

行编辑程序EDLIN 有两种基本工作方式：插入方式和命令方式。插入方式用于从键盘输入字符；而命令方式用于显示、删除、搜索、替换等，以进行修改。

当工作于插入方式时，从键盘输入的字符，除一些控制字符以外，都送至编辑缓冲区。所以，在插入方式下，打入的EDLIN 的其它命令都不起命令的作用，而作为相应的字符送入编辑缓冲区。

因此，若在插入方式下，发现了有打入错误，在未按回车键以前，可以用DEL 键删除一个字符，以进行改正；若按了回车键以后，可以用ESC 键删除正在显示的这一行。此外就不能用 EDLIN 的其它命令加以修改。只有在退出了插入方式，出现提式符 * 以后，才能用各种命令。

插入命令的格式为：

```
[行号]    I
```

对于新文件，在打入了指定的行号和命令字母 I 后，机器响应为：

```
A>EDLIN IABC
  NEW FILE
  * I
  1: *
```

这里行号1:后的* 不是EDLIN 的提式符而是行指针。这时就可以从键盘上打入要插入的内容了。每一行以回车键结束。然后出新的行号，就可以连续不断地插入。

每当插入一行，按了回车键以后，后续的各行会自动重新编号。若在插入命令中没有指定行号，则插入的内容，放在当前行（指针所指的行）之前。

若要输入的内容已插入完毕，则按 CTRL-BREAK 键，以退出插入方式。此时又出现EDLIN 提式符* 。

在退出插入方式以后，所插入的内容的后一行变为当前行。

(2) 显式行命令 (List Line)

在插入过程中，完全可能发生键入错误，或者源文件经过汇编后发现有语法错误，或者程序经运行后发现有错，都要用EDLIN 把源文件输入编辑缓冲区。要检查编辑缓冲区的内容，就要用到显示行命令。此命令的一般格式为：

```
[行号]    [,行号]  L
```

其中第一个行号是指定的起始的行号，在逗号后面的第二个行号是指定的终止行的行号。两者都可任选，L 是显示命令的命令字母。

此命令把指定范围内的所有行全部显示出来，但当前行保持不变。例如：

```
* 1,32L
```

将显示出 1～32 行的内容。

若命令中略去第一个行号，如：

```
, 行号 L
```

命令中的逗号是必须的。此命令由当前行（* 号所在行）的前１１行开始，一直显示到指定的结束行。

例如：假设* 在第一行。

```
*,25L
```

将 1～25 行的内容显示出来。

若命令中略去第二个行号，如：

```
行号 L  或  行号,L
```

则从指定的行开始，一共显示２３行。例如：

```
* 7, L
```

将从第七行开始显示到第２９行。

若命令中把两行号都略去，如：

```
* L
```

则从现行行的前１１行开始共显示２３行。

(3) DOS 的编辑键

为了在插入方式下输入和便于在输入过程中改错，在 IBM PC 中设置了几个具有一定编辑功能的键，称为DOS 的编辑键。

DOS 编辑键用于在一行内进行编辑。

从键盘输入的任何行，在按Enter 键后，你就可以把它作为一个样板以用于编辑。

这些键和它们的功能如下：

① Del 键

按下此键跳过在样板中的一个字符，但光标并不移动。

② ESC 键

按下此键删除正显示的行，但样板保留不变。

③ F1 或 → 键

按下此键，从样板中复制一个字符且显示。

④ F2 键

按下此键，同时输入一个字符，则从样板中复制字符直到遇到指定的字符为止。

⑤ F3 键

按下此键，把样板中所有余下的字符全部复制到屏幕上。

⑥ F4 键

按下此键，且输入一个字符。则跳过样板中所有字符，直到遇到所指定的字符为止。F4的作用与F2是相反的。

⑦ F5 键

按下此键，就把正在编辑的正在显示的行，作为样板，以便进一步进行编辑。但并没有把它送到所编辑的程序，只有按下Enter 键，才把它送至所编辑的程序。

⑧ Ins

按下此键，允许在一行内插入字符。

⑨ ENTER 结束一行的编辑。

(4) 编辑一行的命令(Edit Line Command)

此命令允许对编辑缓冲区中指定的行进行编辑。命令的格式为：

行号

当输入行号，按下Enter 键，则在屏幕上首先显示指定的行号和此行的内容，然后在下一行显示指定的行号，等待输入编辑命令。

若先打入一个句号（．）然后按下 Enter 键，则在屏幕上显示当前的行号和内容，在下一行再次显示当前行的行号，等待输入编辑命令。

在编辑时，可以打入一行新的内容；也可以利用上述的DOS 编辑键，对这一行进行修改、替换、以改正错误。

例如：在编辑缓冲区中，已有一行内容：

3. That is not a unsample Line.

就可以利用编辑命令来加以改正。先打入

```
* 3 <Enter>
3: That is not a unsample Line.
3:
```

下面要保持That is ，删去not ，可以用F2键。按下F2键，且打入 n，则显示变为：

```
3: That is not a unsample Line.
3: That is _
```

然后按四次Del 键，把not 和一个空格删去。要保留a 和一个空格，可以按两次F1键。显示变为：

```
3: That is not a unsample line.
3: That is a_
```

下面要把 un 两个字符删去，可以连续按两次 Del 键； 余下的全要保留， 则可以按 F3 键。于是显示变为：

```
3: That is a unsample line.
```

3: That is a sample line.

一行的编辑工作已经完成。下面可以有几种处理方法。

① 若认为修改工作正确，要用已修改的行代替原来的行，则可按 Enter 键 。此时编辑行命令结束，又出现EDLIN 提示符* ；已修改的行送入编辑缓冲区，代替原来的行。

② 若认为修改后的行有错误，要删除它，则可打入Esc 键，此时编辑行号命令没有结束，编辑缓冲区中的原来行也没有任何改变。

③ 若认为修改后的行有错误，又要结束编辑行命令，则可按 Ctrl Break 键。命令结束，又出现EDLIN 提示符，而原编辑缓冲区的内容保持不变。

⑸ 删除行命令(Delete lines Command)

此命令能把指定范围的行的内容删除。命令的格式:

[行号] [,行号] D

其中，第一个行号是删除范围的起始行行号；第二个行号是结束行行号；D是删除的命令字母。

在上述命令执行以后，指定范围的这些行被删除，范围中结束行的下一行变为当前行。

若命令中略去了一个行号，如:

,行号 D

则删除的范围为从当前行起至指定的结束行。

若命令中略去了结束行，如:

行号 D

或

行号,D

则只有指定的这一行被删除。

若命令中把两个行号都删去，则只删去当前行；且它的下一行变为当前行。

⑥ 搜索命令(Search Text Command)

此命令可以在指定的范围内（若干行内），搜索一个规定的字符串。命令的格式为:

[行号] [,行号] [?] S [字符串]

其中的两个行号，分别规定了搜索的范围的初始值和结尾值；? 号是一个任选值；S是搜索的命令字母；（字符串）就是要搜索的对象。

此命令执行以后，在指定范围内，第一个包含有规定字符串的行被显示出来，且此行变为现行行。若命令中没有包含任选项? 号，则命令结束。

若命令中包含有任选项? 号，则在显示了第一个包含有规定字符串的行以后，在下一行显示:

OK?

若打入字符Y ，或直接按Enter 键则命令结束；若打入N 或任何其它字符，就继续往下搜索，直至下一个包含规定字符串的行找到，且把它显示出来；接着下一行仍显示

OK?

询问是否要继续往下搜索。

若要继续往下搜索，有可能显示下一个包含规定字符串的行；或者搜索一直进行下去，直至指定范围内的所有行都已被搜索过。一旦指定范围内的所有行都搜索过了，则显示信息：

Not found

若在命令行中，没有规定的字符串，则S 命令就使用上一个替换命令(Replace) 或搜索命令(Search)中所规定的字符串。

若在指定的范围内找不到规定的字符串，则显示信息：

Not found

且命令结束。

若在命令中，略去初始行行号，则S 命令认为是从当前行的下一行开始搜索。

若在命令中，略去结束行行号，则S 命令认为搜索从指定的开始行起直至编辑缓冲区的最后一行。

命令的注意事项：

① 命令中的规定字符串是从命令字母 S 后立即开始（没有空格），直至由按 Enter 键结尾。

② 若一行中包含有多个命令，则 S 命令应以 Ctrl-Z（F6）作为结尾，从下一个字符位置开始下一个命令。

(7) 替换命令(Replace Text Command)

此命令的一般格式为：

[行号] [,行号] [?] R [字符串1] [<F6>字符串2]

其中的两个行号规定了替换命令的范围；? 号是一个任选项；R是替换命令字母；替换命令是要用后一个字符串替换前一个字符串；在这两个字符串之间用 F6 键分隔（若在操作中没有改变 F6 键的意义，否则要用 Ctrl-Z 键）。

此命令执行时有两种情况：

① 若在命令行中，没有规定任选项? 号。则R命令把规定范围内的由字符串1规定的所有字符串，用命令中规定的的字符串2代替，且显示每一个被替换的行。被替换的最后一行变为现行行。

② 若在命令中，规定了任选项? 号。则R命令显示第一个被替换的行，然后在下一行显示：

OK?

若回答Y 或ENTER 键，则命令结束。

若回答N 或任何其它字符，则继续进行替换，显示下一个被替换的行（或同一行中下一个被替换的字符串）。接着在下一行显示：

OK?

等待你的回答。

若在命令中，没有输入第二个字符串。则指定范围内的由第一个字符串所规定的字符串全部被删去。

若在命令中，两个字符串都没有输入。则R命令用上一个S或R命令的搜索字符串作为第一个字符串；用上一个R命令的替换字符串，作为字符串2。

若在命令中，略去了第一个行号，搜索从当前行的下一行开始。若在命令中略去了第二个行号，则搜索继续到编辑缓冲区中的最后一行，若两个行号都略去，则从当前行的下一行开始，搜索至编辑缓冲区的最后一行。

注意：

第一个字符串从字母R后立即开始（没有空格），一直继续到 F6 或 Ctrl Z （若没有第二个字符串，则为ENTER 键）。

第二个字符串从 F6 或Ctrl-Z 后开始，一直继续到Enter 键。

⑻ 读行命令

格式：　[n] A

把指定数目的行从盘上读到内存中被编辑的文件上，加到当前行的后面。若省略 n，则读若干行，直到填满缓冲区的７５％为止。

⑼ 复制命令

格式：　[行号１]，[行号２]，行号３ [，重复次数]，C

把行号１至行号２范围内的行复制到 由行号３开始的行号上，并重复指定的复制次数。行号１和行号２的隐含值为当前行，重复次数为１。复制后的第一行成为当前行。

⑽ 移动命令

格式：　[行号１] [，行号２]，行号３　M

把行号１至行号２范围内的行移到行号３的前面。行号１和行号２的隐含值为当前行。

⑾ 页显示命令

格式：　[行号１] [，行号２] P

把行号１至行号２范围内的行以页的形式显示出来，修改当前行为显示的最后一行。如果省略行号１，则为当前行加１。如果省略行号２，则显示２３行。

⑿ 读文件命令

格式：　[行号] T [驱动器名：] 文件名

把指定的文件的内容读入内存，插入到正在编辑的文件中指定行号的前面。

⒀ 写入命令

格式：　[n]W

把正在编辑的文件从行号１开始连续写ｎ行到软盘上去，剩下部分的行号重新编号。当正编辑的文件超过内存容量时，此命令有用。

14. 退出命令

有两种命令可以从EDLIN 退出而返回DOS 。

(1) E 命令

若通过编辑建立了一个新文件；或通过修改，已改正了文件。则要把编辑缓冲区中的内容存盘（若是新文件，则此退出命令在盘上建立所指定的文件；若是老文件，则此命令把编辑缓冲区中的内容，送至文件名所指定的文件中，而原存在盘上的文件，改为后备文件——扩展名为.BAK）。且退出EDLIN ，返回DOS 。例如：

```
* E
A>
```

(2) Q 命令

若由于某种原因，不准备保留编辑缓冲区中的内容，又希望退出EDLIN ，返回DOS ，则可用Q 命令。由于Q 命令要破坏编辑缓冲区的内容，而又未把它存盘，为了慎重，屏幕上提示，是否真要退出，若是打入Y ，则返回 DOS ；否则打入 N ，这样可以避免在编辑过程中，由于误打了Q 而破坏编辑缓冲区的内容。

例如：

```
* q
Abort edit (Y/N)? n
* q
Abort edit (Y/N) y
A>
```

二． 全屏幕编辑程序 WORSTAR

WORDSTAR 实际上是一个文字处理软件，它具有丰富的功能。例如在全屏幕编辑方式下，可以方便地拟稿、编辑、写文件等。 修改也很方便， 例如重新分段、合并段、移动段、插入、删除某些内容，查找、更换某些内容，利用排版功能对版面进行加工、整理。再者，它还可以进行拷贝、删除等文件操作，对打印格式、打印字体进行控制，运行.COM 或 .EXE 类型的文件等等。这里利用它建立、修改汇编语言程序文件，只是利用了它的全屏幕编辑非文书文件的功能。

1. 启动 WORDSATR

开动机器，调入操作系统以后，将装有 WORDSTAR 软件的盘片插入驱动器，键入WS再按回车键，一会儿屏幕上会出现〝起始命令表〞，起始命令如下：

D	进入编辑	E	更换文件名
P	打印文件／中断	O	拷贝文件
R	运行程序	Y	删除文件
N	编辑非文书文件	X	退出

这时，键入N，就是选择编辑非文书文件（即编辑各种高级语言、汇编语言程序文件）

功能。然后屏幕提示：文件名？，再输入要编辑的文件名即可，如 B:EXAM.ASM 。若这是一个新文件名，则意味着建立一个新文件，若是已有的文件名，系统将调出此文件，以供修改。如果采取输入： WS 〈文件名〉的方式，可以直接进入对文件的编辑修改。

输入文件名后，系统进入编辑状态，屏幕显示如图 8-6 所示。

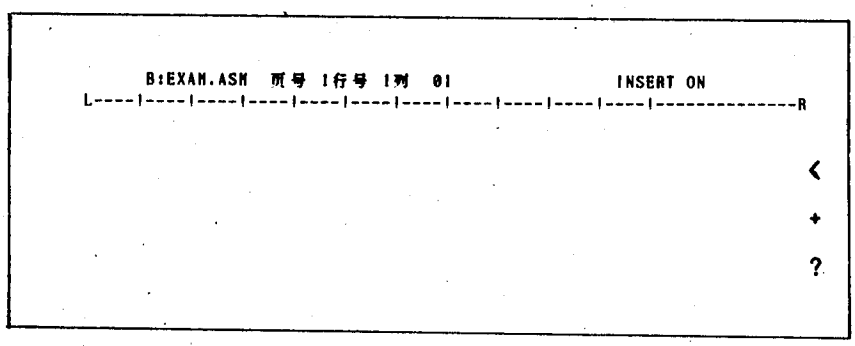


图 8-6 WORDSTAR 屏幕显示

显示的第一行是状态行，它告诉你：

(1) 当前正编辑着的文件名字。

(2) 光标所在页的页号，所在行的行号，所在列的列号。

(3) 当前是否处于插入状态（ INSERT ON）。

第二行是标尺行， 其中的 L、R分别表示左、右边界， 它控制了每行可输入的字符数。可通过 ^OL（或［F 3］键）控制左边界， ^OR（或［F 4］键）控制右边界（其中 ^ 即 [CTRL] 键，左手按 CTRL 键，右手按 O与 R 键即可，下同。屏幕的最右列是标志列。常用的标志为：

< : 用户输入硬回车的行。

+ : 屏幕右边还有字符。

? : 有疑问的行。

空格：程序加的自动回车。

屏幕的其余部分如同一张白纸，你可以一行行输入程序。一屏写满以后，所有行自动上滚，再出现新的空白行。输入过程中，可以随意移动光标到任一页，任一行，任一列，任意删、插、修改。输入或修改完毕，可随时存盘。

2. 编辑命令

WORDSTAR 提供的编辑命令很丰富，现选择主要命令介绍如下。

(1) 帮助命令 ^J

如果在操作过程中，忘记了有关的命令或命令的功能等，可以随时按下 ^J ，以查看系统提供的帮助信息。提示帮助分为文件操作（F）、字符串和字块操作（U）、打印字体控制（P）、编辑（D）、排版（B）、命令索引（I）几方面。这里主要查看编辑和命令索引两部分。

(2) 退出编辑状态的命令

退出编辑状态有如下几种选择：

① ^KD（或［F 1］键）命令：将当前正编辑的文件存到磁盘上，然后退出编辑状态，返回到"起始命令表"。

② ^KX命令：将正编辑的文件存盘后，退出编辑程序，返回操作系统。

③ ^KQ（或［F 2］键）命令：放弃当前的编辑文件（即不存盘），退出编辑状态返回到"起始命令表"。退出前，屏幕提问是否要放弃当前编辑着的文件（Y/N），按 Y 表示肯定，以防按错键，也为慎重。

⑶ 光标移动命令

IBM PC系列机中光标的控制方法有两种，一种是使用键盘右侧的小键盘，它具有双重功能：即用来控制光标移动或输入数字，按下［NUM LOCK］键可以选择光标状态或数字状态。在光标状态下：

↑： 光标向上移一行

↓： 光标向下移一行

←： 光标向左移一个 ASCII码位置

→： 光标向右移一个 ASCII码位置

PgUp：屏幕翻页，换成当前显示页的前一页。

PgDn：屏幕翻页，换成当前显示页的后一页。

另一种是利用控制键：

^E： 光标向上移一行

^X： 光标向下移一行

^S： 光标向左移一个 ASCII码位置

^D： 光标向右移一个 ASCII码位置

^R： 光标移到当前显示页的前一页。

^C： 光标移到当前显示页的后一页。

^W： 光标不动而整个屏幕向下移一行。

^Z： 光标不动而整个屏幕向上移一行。

^A： 光标左移一句（或一个英文单词）。

^F： 光标右移一句（或一个英文单词）。

⑷ 删除与插入命令：

^G： 删除光标处的字符

[DEL] 键：删除光标左边的字符

^T： 删除光标右边的一句

^Y： 删除光标所在行

^V或[INS] 键：变换插入／非插入状态

^N： 在光标处加一行

⑸ 移动、复制命令

WORDSTAR 提供的字块（/字符串）操作命令可以方便地移动、复制、考贝或读入某几个、某几行、某几段字符内容。

① 在进行移动、复制等操作以前，先把光标移到字块前、后，加注标记。

- ^KB ：在字块前加块首标记 〈B〉
- ^KK ：在字块后加块尾标记 〈K〉
- ^KH ：删除字块的首尾标记

再将光标移到预想位置，然后发如下命令 ：

- ^KV ：将加有首尾标记的字块移到当前光标位置
- ^KC ：将加有首尾标记的字块考贝到当前光标位置，原字块仍存在
- ^KY ：删除加有首尾标记的字块

②文件之间的内容交换可用如下命令 ：

- ^KW ：把当前文件中加有首尾标记的字块写入指定文件，首先屏幕提示 ：

  ？文件名

  用户回答后，考贝工作自动完成。

- ^KR ：可将指定文件的内容考贝到当前文件中光标所在位置，同样首先提示文件名。

⑹ 打印命令：

在＂起始命令表＂下，按下 P，屏幕上提示输入要打印的文件的文件名。输入完文件名后，屏幕上连续出现多个提问，如：输出到磁盘（Y/N ）吗？ ，按下 N或直接按回车表示不输出到磁盘。然后又提问：从头开始打印吗？打印到最后一页吗？等等。回答完提问后，查看打印机已准备好，再按下回车键，即开始打印。打印期间 ，随时按下 ^KP可以暂停或继续打印工作。

如果要打印整个文件，可在输入打印的文件的文件名后，按下［ ESC］键，则略去上述一连串提问，直接进入打印。

第四节　汇编与宏汇编程序

1. 功能：

经过编辑命令建立和修改后存盘的汇编语言源程序（扩展名为.ASM），要在机器上运行，必须先由汇编程序把它汇编为机器码的目标程序。汇编程序与宏汇编程序的区别在于宏汇编程序有宏处理功能，而汇编程序没有，但它比较简单。经IBM 汇编程序汇编后的程序在盘上建立三个文件：一个是扩展名为.OBJ的目标程序。在此程序中，操作码部分已变为机器码，但地址操作数只是一个可浮动的相对地址，而不是内存中的绝对地址。第二个是扩展名为.LST的可打印文件，它把源程序（包括注释）和汇编后的目标程序都制表，可

打印出来以供检查用。第三个是扩展名为.CRF的可以对符号进行前后对照的文件。在汇编时，汇编程序对要不要建立这些文件，要建立时的文件名进行提问。如下所示：

在DOS 状态下，打入MASM（或ASM ）调用宏汇编程序（或汇编程序），屏幕显示与操作如下：

```
A>MASM
  The IBM Personal Computer MACRO Assemble Version 1.00(c) Copyright
     IBM  Corp 1981
       Source filename [.ASM]: exam
       Object filename [EXAM.OBJ] :
       Source Listing [NUL.LST]: exam
       Cross Reference [NUL.CRT]: exam
       Warning Severe
       Errors Errors
```

在汇编程序调入后，先显示版本号，然后出现第一行提示，询问要汇编的源文件名。用户输入文件名，则出现第二个提示，询问目标程序的文件名，括号内的为机器规定的默认的（ default ）文件名，通常直接按回车，表示采用默认文件名。 接着出现第三个提示，问要否建立可打印文件，若要，打入文件名；若不要，可直接回车。最后出现第四个提示，询问是否要建立交叉索引文件，若需要则打入文件名，若不要则直接回车。在回答了第四个询问后， 汇编程序就对源程序进行汇编。 若汇编过程中发现源程序中有语法错误，则列出有错误的语句和错误的代码。在MASM中，还直接指出是什么性质的错误，即错误的类型。最后列出错误的总数。此时，就可分析错误，然后再调用编辑程序加以修正，修正后重新汇编，直至汇编后无错误。

汇编后建立的.LST文件可用于打印，以了解汇编后的情况。

汇编后建立的.CRT文件是用来了解源程序中的符号（包括变量）定义时和引用时的情况。要打印出符号（变量）表时，要用到CREF文件，其过程如下：

```
A>CREF
Cref filename [.CRT]:exam
List filename [EXAM.REF]:
A>TYPE EXAM1.REF
```

在 DOS 下打入 CREF 命令，在CREF文件调入后，提问文件名。在打入文件名后，出现第二个提示，问 LIST文件名，可用回车承认默认的 LIST 文件名。于是就建立了一个扩展名为.REF的文件，然后返回DOS 。

在DOS 下，用TYPE命令打印此文件，得类似如下的显示：

```
Symbol Cross Reference (# is definition) Crof--1
AGAIN ... ... ... ...32#    33
```

COSEG ..... 9# 10 35

其中，有# 号者是此符号定义时的语句号，后面是引用此符号时的语句号。

第五节 连接程序

由汇编程序建立的目标码文件必须经过连接以后，才能成为可执行文件。连接程序并不是专为汇编语言设计的。利用它可以把若干个模块连接在一起，这些模块可以是汇编程序产生的目标码文件，也可以是其它高级语言编译程序产生的目标文件。

汇编后的目标文件（.OBJ）中的地址操作数是可浮动的相对地址，经过连接后的文件成为可执行的（.EXE）文件。同时，在连接过程中，又建立了几个文件。连接过程如下：

在 DOS下，打入 LINK ，就会调入连接程序。调入以后，先显示版本号，然后依次提问四个问题，如下所示：

```
Object Modules [.OBJ]:exam
Run File   [exam.EXE]:
List File   [NUL.MAP]: exam
Libraries      [.LIB]:
```

首先询问要连接的目标文件名，操作员打入文件名（扩展名应为.OBJ）作为回答。如果有多个要连接的目标文件，应一次打入，各目标文件名间用＂＋＂号相间隔。第二个提示询问要产生的可执行文件的文件名，一般直接回车就采用了括号内规定的隐含文件名。第三个询问要否建立内存分配图文件。打入文件名再回车表示要建立，直接回车表示不要建立。最后提示是否用到库文件，汇编语言没有库文件，所以直接打入回车即可。

回答完以上问题后， 连接程序开始进行连接， 若连接过程有错，则显示错误信息。如：

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

有错误就要重新调用编辑程序修改，然后重新汇编，再经过连接，直至无错。

连接以后建立的可执行（.EXE）文件，可以在 DOS下，直接打入文件名（不必要扩展名.EXE）运行此文件。例如：

```
A>EXAM
```

连接以后产生的.MAP文件，提供了文件中的内存地址分配的一些信息，如下所示：

```
A>TYPE exam.MAP
  START     STOP      LENGTH    NAME      CLASS
 00000H    0012H     0013H     BUF_DAT
 00020H    00041H    0022H     COSOG
 00050H    000B3H    0064H     STACK STACK
```

Program Entry point at 0002:0000

由上看出，变量 BUF_DAT 的起始地址是 0000H，结束地址是 0012H，字节数是 13H，以此类推。

第六节　调试程序

DEBUG是专为汇编语言设计的一种调试工具，是汇编语言程序员必须掌握的调试手段。

一．如何调用DEBUG 程序

在 PC DOS 的提示符A>下，可以打入如下命令：

A>DEBUG [d:] [Path] [filename [ext]] [Parm1] [Parm2]

其中DEBUG 是调试程序的文件名；后面是要调试的程序的文件标识符。若在命令中规定了文件标识符，则在DOS 把DEBUG 程序调入内存后，DEBUG 程序把要调试的程序（根据文件标识符）调入内存。若在命令中没有规定文件标识符，则DEBUG 程序或者与正在内存中的内容打交道；或可以用Name和Load命令，从盘上输入要调试的程序。

命令行中的 [Parm1 （参数1）] [Parm2 （参数2）]。我们在后面结合具体的命令加以介绍。

在DEBUG 程序调入后，出现提示符"－"。说明现在系统在DEBUG 程序的管理之下，所有的DEBUG 命令，也只有在出现此提示符后才有效。

二．DEBUG 程序对寄存器和标志位的初始化

在DEBUG 程序启动后，它把各个寄存器和标志位，置成以下状态：

1. 段寄存器（CS DS ES 和 SS）置于自由存贮空间的底部，也就是DEBUG 程序结束以后的第一个段。

2. 指令指针（ IP ）置为 0100H。

3. 堆栈指针置到段的结尾处，或者是装入程序的临时底部，取决于哪一个更低。

4. 余下的寄存器（AX BX CX DX BP SI 和 DI）置为0。但是，若调用 DEBUG 时包含一个要调试的程序文件标识符，则 CX 中包含以字节表示的文字长度，若文件大于64K，则文件长度包含在 BX 和 CX 中（高位部分在 BX 中）。

5. 标志位都置为清除状态。

6. default 的磁盘缓冲区置于代码段的 80H 。

注：若由DEBUG 调入的程序，具有扩展名．EXE ，则DEBUG 必须进行再分配。把段寄存器，堆栈指针置为程序中所规定的值。

三．有关DEBUG 命令的一些共同信息：

1. DEBUG 命令都是一个字母，后面跟有一个或多个参数。

2. 命令和参数可以用大写或小写或混合方式输入。

3. 命令和参数间，可以用定界符分隔。然后定界符只是在两个邻接的 16 进制之间

是必需的。因此，下列命令是等效的：

```
d cs: 100 110
d,cs: 100,110
```

4. 可以用按 Ctrl Break 键来停止一个命令的执行，返回DEBUG 提示符。

5. 每一个命令，只有按了ENTER 键以后才有效，才开始执行。

6. 若一个命令产生相当多的输出行，为了能在屏幕上当一行卷走以前读清楚它，可以在显示过程中，按 Ctrl Num Lock 键，以暂停上卷；可以按任何一个字符来重新启动。

7. 若DEBUG 检查出一个语法错误，则显示具有错误的行和指示错误所在。

例如：

```
d  cs:100  cs:  110

                error
```

四．DEBUG 的主要命令

1. 显示内存单元内容的命令（Dump Command）

为了了解程序执行的结果，检查内存单元的内容是十分重要的。此命令能检查指定范围的存贮单元的内容。命令的格式为：

D [地址]

或 D [范围]

其中，D是命令字母，[地址] 或 [范围] 都是为了指定要显示的存贮单元的范围。

存贮单元的内容用两种方式显示：一种是每一个存贮单元的内容（每一字节）用两位16 进制数显示；另一种是用相应的 ASCII 字符显示 ，句号（.）表示不可显示的字符。

显示有两种格式，若是 40 列系统显示格式，每一行显示８个字节，若是 80 列显示格式，每一行显示16字节。第８、９字节间有一连字符（-）。

显示命令有两种常用的格式：

(1) 格式为：

D 地址

或 D

若命令中有指定地址，则从指定地址开始，显示40H 个字节（相应于系统的40列显示格示），或80H 个字节（相应于 80 列显示格示）。

若命令中没有指定起始地址，则从上一个D命令所显示的最后一个单元的下一个单元开始。若以前没有使用过D命令，则从由DEBUG 初始化的段寄存器的内容，加上地址偏移量0100H 作为起始地址。

若在命令中所打入的地址中，只包含起始地址的偏移量，则D命令认为段地址包含在DS中。

例如：对于 80 列的显示格式

```
- D 100
```

显示的起始地址由DS内容（作为段地址）与 100H（作为偏移量）组成。共显示80H个单元的内容。

⑵ 显示指定范围的内容：

D 范围

在范围中包含起始地址和结束地址。若输入的起始地址中，未包含段地址部分，则D命令认为段地址在DS中；而输入的结尾地址中，只允许有地址偏移量。

例如：

```
-D DS:100 0200
```

2. 修改存贮单元内容的命令

此命令用于修改存贮单元的内容，它有两种基本格式：

⑴ 用命令给定的内容表去代替指定范围的内存单元的内容：

E 地址 内容表

例如：

```
E DS:100 F3 "XYZ" 8D
```

内存单元 DS:100 到 DS:104 这 5 个单元的内容由表中给定的 5 个字节的内容（其中 2 个字节用 16 进制数表示，即 F3、8D；另三个用字符表示 ，就是 " XYZ " ，用它们的 ASCII 码值代入）所代替。

⑵ 一个单元一个单元地连续修改方式

E 地址

在输入了上述命令，屏幕上显示指定单元的地址和原有的内容之后，可以采用以下几种操作中的一种：

① 再输入一个字节的 16 进制数，以代替原单元中的内容，然后可以采取下面三种操作之一。

② 按空格键，则上一个替换要求完成，且显示下一个单元的地址和原有的内容，要修改的话，则输入两位 16 进制数，再按空格键.....这样就可以连续地进行修改。

若某一单元的内容不需要修改，而操作要进行下去，则可直接按空格键。

③ 输入一个连接号（－），则显示前一单元的地址和内容，若要修改的话，就输入一个字节的内容，然后再按 " － " ，则又显示前一个单元的地址和内容.....这样，就可以连续地进行反向修改。

若所示的前一单元的内容不需要修改，可直接按 " － " 键。

④ 按回车键结束此命令。

3. 检查和修改寄存器内容的命令（Register Command）

为了了解程序运行是否正确，检查寄存器内容的操作是十分重要的。

R命令有三种功能：

(1) 能显示CPU 内部的所有寄存器的内容和全部标志位的状态。

(2) 能显示和修改一个指定的寄存器的内容。

(3) 能显示和修改所有标志位的状态。

下面分别予以介绍：

① 命令格式为：

R

则系统的响应为：

```
A>debug example4.exe
-r
AS=0000  BX=0000  CX=004A  DX=0000  SP=0064  BP=0000  SI=0000  DI=0000
DS=07B5  ES=07B5  SS=07C6  CS=07C6  IP=0000   NV UP DI PL NZ NA PO NC
07C6:0000 B8C507        NOV     AX,07C5
```

前两行 80 列显示格式显示了所有CPU 内部寄存器的内容和全部标志的状态（其含意在下面介绍），第三行显示了现行CS:IP 所指的指令的机器码以及汇编符号，这就是下条即将要执行的指令。

② 命令的格式为：

R 寄存器名

例如为了检查和修改寄存器AX的内容，可打入以下命令：

```
-R AX
```

则系统可能出现如下响应：

```
AX F1F4
```

此时，可采取以下两种操作之一：

若不需要改变其内容，直接按Enter 键。

若需要改变其内容的话，可输入 1-4 个 16 进制数值，再按 Enter 键，以实现修改。例如：

```
-r  BX
BX  0369
:059F
```

则 BX 中的内容由0369H 改变成059FH 。

③ 显示和修改标志位状态

在8086／8088中，共有九位标志位，其中追踪标志T不能直接用指令改变，其它8个标志位可以显示和修改。

在显示时，8个标志的显示次序和符号如下：

标志名	置位	复位
溢出overflow （是 /否 ）	OY	NY
方向Direction （减量/增量）	DN	UP
中断Interrupt （允许/屏蔽）	EI	DI
符号 Sign （负 /正 ）	NG	PL
零 Zero （是 /否 ）	ZR	NZ
辅助进位Anxigiary Carry （是 /否 ）	AC	NA
奇偶 Parity （偶 /奇 ）	PE	PO
进位 Carry （是 /否 ）	CY	NC

命令的格式为：

RF

系统可能给出如下响应：

OV DN EI NG ZG ZR AC PE CY-

于是可以采取以下两种之一的操作：

若不需要修改任一个已设置的标志状态，可直接按Enter 。

若有一个或多个标志需要修改，则可以输入此标志的相反的值。输入的标志的次序是无关的，输入的各个标志之间，可以没有空格。然后按Enter 键，以实现修改。

例如：

OV DB EI NG ZR AC PE CY - PONZDINV

4. 运行命令 （Go Command）

为了检查程序运行是否正确，希望在运行中能设置断点，以便一段一段对程序进行调试。

G命令的格式为：

G [=address][address[address...]]

其中，第一个参数 =address 规定了执行的起始地址；以 CS 中的内容作为段地址，以等号后面的地址值作为地址偏移量。在输入时＝号是不可缺少的（以便与后面输入的断点地址相区分）。

若不输入起始地址，则以CS:IP 作为起始地址。

后面的地址参数是断点地址。 若在命令行中，除了起始地址以外， 没有任何地址参数，则程序执行时没有断点。

在开始调试程序时，往往要设置断点。DEBUG 程序中允许最多可设置10个断点，这些断点地址的次序是任意的。设置多个断点的好处是：若程序有多个模块，多个通路，不管是哪一个通路执行，都有可能在断点处停下来。

DEBUG 程序用一个中断类型3指令（操作码为CCH ）来代替被调试的程序在断点地址处的指令操作码。当程序执行到一个断点地址时就停了下来，显示CPU 内部所有寄存器的内容，和全部标志位的状态（相当于一条R命令）；被调试程序的所有断点处的指令被恢复（若程序执行未遇到断点，则不恢复）；全部断点被取消；返回DEBUG 。

于是，就可以利用各种DEBUG 命令来检查程序运行的结果和进行必要的修改。

注释：

① 命令中的地址参数所指的单元，必须包含有有效的8086／8088的指令码。若指定的地址单元，不包含有效指令的第一个字节，则会出现不可予料的结果。

② 对于 GO 命令，堆栈必须至少包含有6个可用的字节；否则会出现不可予料的结果。

③ 若输入的断点地址，只包含地址偏移量，则G命令认为其段地址在段寄存器CS中。

5. 追踪命令 （Traco Command ）

追踪命令有两种格式：

(1) 一条指令一条指令地追踪

若打入命令

T [=address]

则执行一条指定地址处的指令， 就停下来，显示 CPU 所有寄存器和全部标志位的状态，以及下一条指令的地址和内容，返回DEBUG 。此时可以用其它DEBUG 命令进一步检查此条指令执行的结果，和作必要的修改。

若在指令中没有指定地址，则从CS:IP 的现行值执行。

(2) 多条追踪命令

命令的格式为：

T [=address] [值]

此命令从指定的地址开始（若命令中用［＝地址］给定了起始地址，则从这起始地址开始，若未给定，则从当前的 CS:IP开始），执行由命令中的［值］所决定的几条指令，执行就停下来了。T命令显示每条指令执行后的寄存器的内容和全部标志位的状态。执行停下来后，返回 DEBUG ，可以用 DEBUG 的其它命令进一步检查指令执行后的结果，和作必要的修改。

6. 汇编命令 （Assemble Command）

若在调试中发现程序中的某一部分要改写；或要增补一段等等，就可以直接在DEBUG下输入、汇编、运行和调试这一段程序。这比每一次修改都要经过编辑、汇编、连接...这样的过程简便多了。

汇编命令的格式为：

A ［地址］

于是从指定的地址开始，可以输入汇编语言的语句，A命令把它们汇编成机器码从指

定的地址单元开始连续存放。

若在命令中没有指定地址，但前面用过汇编命令的话，则接着上一个汇编命令的最后一个单元开始存放。若前面未用过汇编命令，则从CS:100单元开始存放。

若输入的语句中有错，DEBUG 就显示

Error

且重新显示现行的汇编地址，等待新的输入。

DEBUG 支持标准的 8086／8088 汇编语言语法（和8087指令系统），具有以下一些规则：

(1) 所有输入的数字值，全为 16 进制数，可输入1-4 个16进制数字字符。

(2) 前缀助记符，必须在相关的指令之前输入，也可以分别放在不同的行。

(3) 段超越助记符为CS:、DS:、ES: 和SS: 。

(4) 串操作助记符中，必须注明串操作的长度：即说明是字操作还是字节操作。

(5) 交叉段返回（远返回）的助记符为RETF。

(6) 汇编程序能自动汇编短、近和远的转移和调用，取决于到目的地址的字节偏移量。这些也能够由NEAR和FAR 前缀来超越。

例如：

```
0100:0500  JMP      502          ; 两字节短转移指令
0100:0502  JMP  NEAR 505         ; 三字节短转移指令
0100:0505  JMP  FAR  50A         ; 五字节远转移指令
```

(7) DEBUG 不能确定某些操作数涉及的是字存贮单元还是字节存贮单元。在这种情况下，必须用前缀"WORD PTR"（可缩写为"WO"）OR"BYTE PTR"（可缩写为"BY"）来明确说明数据类型。例如：

```
NEG  BYTE PTR[126]
DEC  WORD PTR[SI]
```

(8) DEBUG 也不能确定一个操作数是立即数还是存贮单元的地址。所以DEBUG 中存贮单元的地址，放在方括号中。例如：

```
MOV  AX,21H      ; 把21H 送至AX
MOV  AX,[21]     ; 把地址为21H 以及22H 存贮单元的内容送至AX
```

(9) 两个最常用的指令 DB 和 DW 也能被使用，用来直接把字节和字的值送入相应的存贮单元。

例如：

```
DB   1,2,3,4,"THIS IS AN EXAMPLE"
DW   1000,2000,3000,"CH"
```

(10) DEBUG 支持所有形式的寄存器间接寻址命令。

例如：

```
ADD  BX,34[BP+2] [SI-1]
POP  [BP+DI]
```

(11) DEBUG 支持所有的操作码的同义词。

例如：

```
LOOP   100
JA     200
```

下面我们举一个例子，来说明A命令的使用。

```
-A CS:0100
08F8:0100  db    30,31,32,33,34,35,36,37,38,39,41,42,43,44,45,46
08F8:0110  mov   si,0100
08F8:0113  mov   di,0200
08F8:0116  mov   cx,10
08F8:0119  rep   movsb
08F8:011B  hlt
08F8:011C
```

先用伪指令DB，输入要传送的源操作数；然后输入数据块传送程序。在程序输入完毕后，最后一行不输入内容，直接按回车，使返回DEBUG 。

7. 反汇编命令 （Un assemble Command ）

若在内存某一区域中，已经有了某一个程序的目标码。为了能清楚了解此程序的内容，就希望能把目标程序反汇编为源程序。这就要用到U命令。

U命令有两种常用的格式。

(1) 一种格式为：

```
U
```

或

```
U  地址
```

则从指定的地址开始，反汇编 16 个字节（在系统的 40 列显示方式）或反汇编 32 个字节（在 80 列显示方式）。

若在命令中没有指定地址，则以上一个U命令的最后一条指令的地址的下一单元作为起始地址；若没有输入过U命令 ，则以由 DEBUG 初始化的段寄存器的值，作为段地址；以 0100 作为地址偏移量。

例如：

```
A>debug exam11.exe
-u
 0091:0000  B81F00        MOV    AX,091F
 0091:0003  8ED8          MOV    DS,AX
 0091:0005  8EC0          MOV    ES,AX
```

```
0091:0007  B82209        MOV     AX,0922
0091:000A  8ED0          MOV     SS,AX
```

这样的U命令一次显示 32 个字节（ 80 列显示格式），但程序未完，可再用U命令接着显示。

(2) 第二种格式为：

U 范围

对指定范围的内存单元进行反汇编。

范围可以由起始地址、结束地址来规定，例如：

```
U 04BA:0100  0108
```

也可以由起始地址及长度来规定。例如：

```
U 04BA:0100  L9
```

这与上一个命令的结果是相同的。

若命令中规定结束地址，则结束地址中只包含地址偏移量。

此命令一次把指定范围的内容反汇编出来。故若已知要反汇编的程序的长度（字节数），则用此命令，一次可把整个程序都反汇编出来。是很方便的。

8. 输入命令（Input Command ）

此命令能从指定的端口输入一个字节且显示出来。命令的格式为：

I 端口地址

例：

```
I  34
5F
```

从端口地址34H 输入一字节的内容5FH 。

9. 输出命令（Output Command）

此命令能向指定的端口输出一个字节。命令的格式为：

O 端口地址 字节值

10. 命名命令（Name Command）

此命令有两个功能：

(1) 用命令中给定的两个文件标识符（filespec）格式化在CS:5CH和 CS:6CH 的两个文件控制块（若在调用DEBUG 程序时具有一个文件标识符，则它已格式化在CS:5CH的文件控制块）。

文件控制块是后面要介绍的Load命令和Write 命令所需要的；且能提供所要求的文件名。

(2) 能把命令中所打入的文件标识符和别的参数，精确地按照打入的情况（包括定界符），放至自CS:81H 开始的参数保存区中，在CS:80H 中保存输入的字符个数。在寄存器AX中保存前两个文件标识符中的驱动器标志。

N命令的格式为：

N　文件标识符［文件标识符］

若在调用DEBUG 程序时，没有规定文件标识符，则必须先用N命令把要调用的文件标识符，格式化到 CS:5CH 的文件控制块中，才能用 Load 命令把它调入内存以进行调试。例如：

```
A>DEBUG
- N myprog
- L
-
```

为了定义正在调试的程序所需要的文件标识符及其他参数，可输入以下命令。

```
A>DEBUG MYPROG
- N file1  file2
-
```

在这种情况下，在调用 DEBUG 时，就已经把文件 MYPROG 调入且放在自 CS:100H 开始的存贮区中；而且此文件的文件控制块已格式化在CS:5CH 开始的区域中。

而N命令要格式化 file1 和 file2 的文件控制块至 CS:5CH 和 CS:6CH 处，这样 file1 的文件控制块就把原来的 MYPROG 的文件控制块冲掉了。

N命令还把命令中除命令字母N以外的所有字符（包括定界符）放至CS:81 起的缓冲区中，CS:80 中包含了字符个数。

11. 装入命令（Load Command）

L命令有两种基本的功能：

(1) 把磁盘上指定区域（指定驱动器和指定扇区范围）的内容，装入到内存的指定区域中，命令的格式为：

L　地址　驱动器　扇区号　扇区数

其中地址是装入内存的起始地址，若输入时没有给定段地址，则L命令认为段地址包含在 CS 中，第一个扇区号是指定的起始的相对扇区号；后一个是指定要装入的扇区数。L命令一次能读入的最大区段数为80H 。例如：

```
L  400:100 1 10 32
```

是指从B驱动器（由驱动器号 1 指定）相对扇区号10H 开始装 32H(50)个扇区的内容至内存从400:100H 开始的区域中。

(2) 装入指定的文件

命令格式为：

L　地址

或

L

此命令装入已在 CS:5CH 中格式化的文件控制块所指定的文件。所以在使用这种格式的L命令之前，在 CS:5CH 中必须有已格式化的文件标识符，这通常可以选用一条N指令来实现。

若命令中没有规定地址，则文件装入到CS:100开始的内存区域中；若命令中规定了地址，则装入至指定的区域中。但若是具有扩展名 .COM 或 .EXE 的文件，则始终是装入至CS:100的区域中，即使在命令中指定了地址，此地址也即忽略。

在 BX 和 CX 中包含所读的文件的字节数；但若所读的文件具有扩展名 .EXE，则 BX 和 CX 中包含实际的程序长度。

12. 写命令（Write Command ）

此命令把正在调试的数据，写入至磁盘中。W命令一次能写入的最大扇区数为 80H，W命令也有两种基本格式：

(1) 把数据写至指定的扇区

命令的格式为：

W 地址 驱动器 扇区号 扇区数

此命令把由地址所指定的内存区域中的数据，写入至指定的驱动器，起始扇区由命令中相对扇区号指定，要写入的扇区数也由命令中给定。

注释：

① 写数据至指定扇区的操作要十分小心，因若有差错，就会破坏盘上的原有内容。

② 若命令中给定的地址，只包含地址偏移量的话，则W命令认为段地址在 CS 中。

③ 记住起始扇区号和扇区数，都是用 16 进制数表示的。

例如：

```
W 1FD 1 100 A
```

此命令把内存起始地址为CS:01FD 的缓冲区的数据，写入到驱动器B，起始扇区号为相对扇区号100H（256 ），共写入0AH(10) 个扇区。

(2) 写入至指定的文件中

命令的格式为：

```
W
```

或

```
W 地址
```

此命令把指定内存区域中的数据，写入到由 CS:5CH 处的文件控制块所规定的文件中。

若命令没有指定地址，则内存区域从 CS:100H 开始；若命令中给定地址 ，则从指定地址开始。

在用W命令以前 ，CS:5CH 中必须要有要写入的文件的文件控制块，这可以先用一条N命令来实现。

在用W命令以前，在 BX 和 CX 中应包含有要写入文件的字节数。

若企图写入具有扩展名.EXE或.HEX的文件，则DEBUG 显示一个错误信息。因为这些文件的写入要用一种特殊格式，而这种格式，DEBUG 程序不支持。

13. 退出命令（Quit Command）

当把程序调试完，就要退出DEBUG 程序，返回DOS时，可采用Q命令。命令格式为：

Q

Q命令并不把在内存中的文件存盘。若要存盘的话，必须先用W命令。

第七节 运行程序

前面已说过，汇编语言源程序经过汇编、连接以后生成的 .EXE 文件，可在 DOS下直接键入文件名运行此文件（不必键入扩展名）。例如命令：

A>EXAMPLE

将把带有扩展名 .EXE 的文件 EXAMPLE装入内存，并从程序中指定的地址开始运行。装入文件并设置启动地址是由操作系统的 COMMAND.COM文件完成的。 COMMAND.COM文件在装入 .EXE文件以前，首先确定最低可用地址作为被装入程序的可用内存起点。（内存中这个区段称作程序段 ），再在程序段内偏移地址 0 处构造一个 100H 字节的程序段前缀 (PSP)。

程序段前缀的结构如下：

段内偏移地址	分配情况
00-01H	INT20H指令
02-03H	内存的总容量（以 16 字节为单位）
04-08H	FAR JUMP DOS子程序的调度程序
09-0CH	程序结束地址
0D-10H	CTRL-BREAK退出地址
11-14H	标准错误出口地址
5C-6BH	FCB1
6C-7BH	FCB2（若FCB1被打开，则FCB2被FCB1覆盖）
80-FFH	隐含的磁盘传输区（DTA）

COMMAND.COM 把 .EXE 文件装入后，自动设置 DS 和 ES 寄存器指向程序段前缀，CS、IP、SS和 SP 寄存器设置为由连接程序传过来的值。

程序段前缀的开始处是一条中断指令 INT 20H。执行这条指令以后，退出当前程序，返回操作系统。这是实现返回的常用方法。但这并不是说 INT 20H可以按排在程序的任一地方都可实现这一功能 ，而是要求在执行 INT 20H 指令以前，代码段寄存器必须指向这个程序段前缀所在的段地址。如何实现这一要求呢？在前面的许多例题中，我们已看到，

代码段中程序开始处按排了指令：

```
PUSH  DS
MOV   AX,0
PUSH  AX
```

程序的结束处按排了指令 RET 。 这种按排就是为了程序运行结束后， 正确的执行 INT 20H 以便返回操作系统。因为 .EXE 文件被装入后， DS和 ES是指向这个段前缀的，开始将 DS 进栈，再将段内偏移地址 00H进栈，执行 RET 时，又将它们退栈到 CS和IP，则正好执行 INT 20H 指令 。这也是代码段程序中经常看到上述三条指令的原因。

扩展名为 .COM 的文件也可以通过在 DOS下直接键入文件名的方法装入并执行，而且也产生一个程序段前缀。但是与 .EXE 文件不同的是：DOS 把段前缀看作是被装入文件的一部分，复盖着被装入文件的前面 100H 个字节。因此程序员必须保证程序中第一条可执行的指令一定在 CS:100H处。

.COM文件被装入后，CS、DS、ES、SS都设置为指向程序段前缀的段地址， IP 固定为 100H，整个程序只占一个物理段（最大 64K） ， SP 指向这一物理段的末尾，并在栈顶存放了两个字节 00H。

.COM文件不象 .EXE 文件那样带有定位信息，因此只能把整个文件看作一段，整段可重定位，但不能分段重定位。正因为不带定位信息，因此程序中不能出现如 MOVAX，DATA 这样的指令，因为装入程序无法确定 DATA 的定位信息。.COM文件中若要对段寄存器进行操作，程序员应确保这些操作是参照当前 CS 的。例如：应该用 PUSH CS与 POP DS 指令代替 .EXE 文件中的 MOV AX,CODE与 MOV DS,AX指令。 .COM 文件中任何地方都可以按排 INT 20H 指令，以实现退出程序，返回 DOS。.COM文件占用内存少。再者，因为不需要重定位信息，所以在用 DEBUG 调试完 .COM 文件后，可用 DEBUG 的写入命令将文件写回盘上。

例如，先用 DEBUG 的命令将 .COM 文件装入内存 ：

```
A>DEBUG
-N EXAMPLE.COM      ; 用 N 命令铭式化 CS: 5CH 的文件控制快
-L                  ; 装入上述文件
```

现在可以用 DEBUG 的各种命令调试这个程序。调试结束，再用 DEBUG 的 W 命令将程序写回盘上。但在写回以前应预置 CX 为大于等于文件的字节数，如下所示 ：

```
-R CX
  CX 0369           ; 假设 CX 的原值为 0369H
  : 100             ; 假设 CX 改为 100H
-W                  ; 写回 EXAMPLE 文件
 Writing 0100 bytes
-Q                  ; 退出
```

因 DEBUG 不能生成重定位信息，所以 .EXE 文件经 DEBUG 调试完后，不能直接写回盘上。

如果你的程序在编写时就符合 .COM 文件的规定要求，那么，经汇编连接后生成的 .EXE 文件可以直接转换为 .COM 文件。 PC DOS 中的 EXEZBIN 程序可以将 .EXE文件转换成. COM 文件，但被转换的 .EXE 文件必须符合 .COM 文件的规定。第四章第七节文件管理应用举例中，例 1 的程序就是一个符合 .COM 文件要求的例子。

习题八

1. 编写如下程序，并在机器上调试成功。

程序可以接收键盘输入的五个命令（分别为A、B、C、D、E），各个命令的功能如下所述。

(1) 按下A键（即输入A命令），程序自动唱一支歌；

(2) 按下B键，自动画一幅画；

(3) 按下C键，程序自动出题，让用户做十进制乘法运算题（<100的数），并指明用户答案正确与否，若不正确，应请用户重算。

(4) 按下D键，接收用户输入小写字母，马上转换成大写字母输出。

(5) 按下E键，接收用户输入多个（<80个）可显示字符，输入完，马上将其中ASCII码值最大的显示出来。

2. 编程序接收用户键盘输入的年、月、日数字，并自动计算和显示这个日期对应于星期几。要求有输入提示、输入错误检查和错误提示功能。

提示：计算星期几的公式为：

$$S = (X-1)*365+\lfloor (X-1)/4 \rfloor - \lfloor (X-1)/100 \rfloor + \lfloor (X-1)/400 \rfloor + C$$

其中：X 为年号，C 为从元旦到给定日总天数，S/7 的余数为星期几。

附录 A　指令系统综述与查阅表

按功能分类，8088/8086的指令系统可分成六组：

- 数据传送指令组
- 算术运算指令组
- 逻辑运算指令组
- 串操作指令组
- 程序控制指令组
- 处理器控制指令组

一、数据传送指令组

数据传送指令用来实现存贮器和寄存器之间的 8位(字节)或 16位(字)数据的传送，还实现累加器 AL或AX与 I/O端口之间的字节或字传送，堆栈操作，标志传送及取段寄存器内容指令也属于这组。

本组指令的查阅表如表 1。

表 1

类别	指令格式	功能	允许的操作数	讲述所在章节
通用数据传送指令	MOV OPRD1,OPRD2	OPRD1←OPRD2	OPRD1,OPRD2 存贮器,寄存器 寄存器,存贮器 段寄存器,通用寄存器 通用寄存器,段寄存器 存贮器,段寄存器 段寄存器,存贮器 通用寄存器,通用寄存器 通用寄存器,立即数 存贮器,立即数	第二章第三节
	PUSH OPRD	(SP)←OPRD	OPRD 寄存器(CS合法) 存贮器	第一章第四节

表 1　（续）

类别	指令铭式	功能	允许的操作数	讲述所在章节
通用数据传送指令	POP OPRD	OPRD←(SP)	OPRD 寄存器(CS非法) 存贮器	第一章第四节
	XCHG OPRD1,OPRD2	OPRD1←→OPRD2	OPRD1,OPRD2 通用寄存器,通用寄存器 通用寄存器,存贮器 存贮器,通用寄存器	第二章第三节
	XLAT OPRD	AL←[BX+AL]	OPRD 存贮器中表首地址	第六章第三节
I/O端口输入输出指令	IN OPRD1,IPRD2	OPRD1←OPRD2	OPRD1,OPRD2 AL或AX，端口地址n AL或AX，DX	第五章第一节
	OUT OPRD1,OPRD2	OPRD1←OPRD2	端口地址n，AL或AX DX，　AL 或 AX	第五章第一节
地址操作指令	LEA OPRD1,OPRD2	OPRD1←OPRD2的地址偏移量	16位通用寄存器,存贮器	第二章第三节
	LDS OPRD1,OPRD2	DS←OPRD2的段地址 OPRD1←OPRD2的地址偏移量	16通用位寄存器,存贮器 (双字)	第二章第三节
	LES OPRD1,OPRD2	除DS改为ES外,与上同	同上	同上
标志传送指令	LAHF	AH←标志寄存器低位字节	无操作数(隐含)	第二章第二节
	SAHF	标志寄存器低位字节←AH	同上	同上
	PUSHF	(SP)←标志寄存器	同上	同上
	POPF	标志寄存器←(SP)	同上	同上

注：对于存贮器操作数允许各种存贮器寻址方式。

二、算术运算指令组

8086/8088 的算术运算指令可以对四种类型的数进行运算。四种数是：无符号二进制数、有符号二进制数(整数)、无符号压缩型十进制数以及无符号非压缩十进制数。二进制数可以是 8位或16位长的，十进制数存放在字节中。

本组指令的查阅表如表 2。

表 2

类别	指令格式	功能	允许的操作数	讲述所在章节
加法指令	ADD OPRD1,OPRD2	OPRD1←OPRD1+OPRD2	OPRD1,OPRD2 通用寄存器,通用寄存器 通用寄存器,存贮器 存贮器,通用寄存器 存贮器,立即数 通用寄存器,立即数	第二章第三节
	ADC OPRD1,OPRD2	OPRD1←OPRD1+OPRD2	同上	第二章第三节
	INC OPRD	OPRD←OPRD+1	OPRD 通用寄存器 存贮器	第二章第三节
	AAA	AX←对AL中未组合的十进制数(和)调整	隐含	第六章第四节
	DAA	AL←对AL中组合的十进制数(和)调整	隐含	第六章第四节
减法指令	SUB OPRD1,OPRD2	OPRD1←OPRD1-OPRD2	同ADD	第二章第三节
	SBB OPRD1,OPRD2	OPRD1←OPRD1-OPRD2-CF	同上	同上

表 2　（续）

类别	指令格式	功能	允许的操作数	讲述所在章节
减法指令	DEC OPRD	OPRD←OPRD-1	同INC	同上
	NEG OPRD	OPRD←OPRD求补	同上	同上
	CMP OPRD1,OPRD2	OPRD1-OPRD2(比较)	同ADD	同上
	AAS	AL←对AL中未组合的十进数(差)调整	隐含	第六章第四节
	DAS	AL←对AL中组合的十进数(差)调整	隐含	第六章第四节
乘法指令	MUL OPRD	AX←AL*OPRD(字节) 或 DX:AX←AX*OPRD(字)	OPRD（无符号数） 通用寄存器 存贮器	第六章第四节
	IMUL OPRD	除为有符号数乘外,同上	除OPRD为带符号数外,同上	第六章第四节
	AAM	AX←对AX中未组合的十进制积调整	隐含	第六章第四节
除法指令	DIV OPRD	AL←AX ÷OPRD(字节)的商 AH←余数　　或 AX← DX:AX÷OPRD(字)的商 DX←余数	同MUL	第六章第四节
	IDIV	同上	同IMUL (有符号整数除法)	第六章第四节
	AAD	AX←对AX中未组合的十进制被除数调整	隐含(做除法前先调整)	第六章第四节

表 2 （续）

类别	指令铭式	功能	允许的操作数	讲述所在章节
除法指令	CBW	AX←扩展AL中的字节数为字	隐含(做除法前先扩展)	第六章第四节
	CWD	DX:AX←扩展AX中的字为双字	同上	第六章第四节

三、逻辑运算指令组

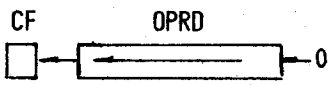
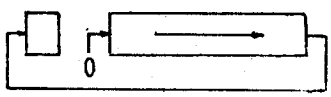
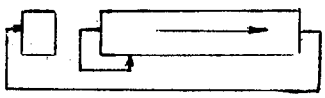
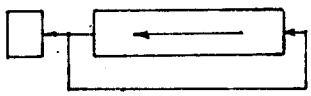
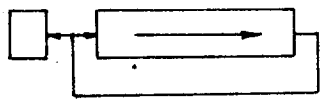
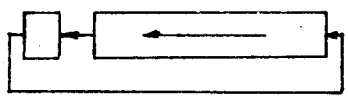
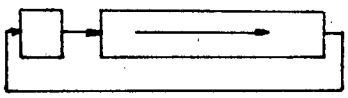
该组包括逻辑与、或、异或、非运算，以及逻辑测试指令。也包括一位或多位的移位指令或循环移位指令。

本组指令的查阅表如表 3 所示。

表 3

类别	指令格式	功能	允许的操作数	讲述所在章节
逻辑运算指令	NOT OPRD	OPRD←OPRD取反	OPRD 通用寄存器 存贮器	第二章第三节
	AND OPRD1,OPRD2	OPRD1←OPRD1 “与” OPRD2	OPRD1,OPRD2 通用寄存器,通用寄存器 通用寄存器,存贮器 存贮器,通用寄存器 存贮器,立即数 通用寄存器,立即数	第二章第三节
	OR OPRD1,OPRD2	OPRD1←OPRD1 “或” OPRD2	同上	第二章第三节
	XOR OPRD1,OPRD2	OPRD1←OPRD1 “异或” OPRD2	同上	同上
	TEST OPRD1,im	OPRD1 “与” im	OPRD1同上， im 为立即数	同上

表 3　（续）

类别	指令格式	功能	允许的操作数	讲述所在章节
移位指令	SHL OPRD，m 或 SAL OPRD，m	CF　OPRD  算术(或逻辑)左移次数= m	OPRD ，m 通用寄存器，1或CL 存贮器 ， 1或CL	第二章第三节
	SHR OPRD，m	CF　OPRD  逻辑右移次数=m	同上	第二章第三节
	SAR OPRD，m	CF　OPRD  算术右移次数=m	同上	第二章第三节
循环移位指令	ROL OPRD，m	CF　OPRD 	同上	第二章第三节
	ROR OPRD，m	CF　OPRD 	同上	第二章第三节
	RCL OPRD，m	CF　OPRD 	同上	第二章第三节
	RCR OPRD，m	CF　OPRD 	同上	第二章第三节

四、串操作指令组

该组指令能对字符串进行传送、比较、搜索、存和取。

该组指令的查阅表如表 4 。

表 4

类别	指令格式	功能	允许的操作数	讲述所在章节
重复前缀	REP	重复,直至CX=0	无	第六章第一节
	REPE/REPZ	当“相等/为零”时，重复,直至CX=0或ZF=0	无	同上
	REPNE/REPNZ	当“不相等/不为零”时，重复,直至CX=0或ZF=1	无	同上
串传送	MOVS OPRD1,OPRD2 可加重复前缀	[DI]←[SI] SI←SI±1(±取决于DF) DI←DI±1(±取决于DF)	OPRD1,OPRD2 目的串地址,源串地址	同上
	MOVSB/MOVSW 可加重复前缀	其中B为字节串,W为字串，其余同上	无	第六章第一节
串比较	CMPS OPRD1,OPRD2 可加重复前缀	[SI]-[DI] SI←SI±1 DI←DI±1	同MOVS	同上
	CMPSB/CMPSW 可加重复前缀	同上，仅B/W指明字节/字	无	第六章第一节
串搜索	SCAS OPRD 可加重复前缀	AL/AX-[DI] DI←DI±1	OPRD 目的串地址	第六章第一节
	SCASB/SCASW 可加重复前缀	同上,仅B/W指明字节/字	无	第六章第一节

表 4　（续）

类别	指令铭式	功能	允许的操作数	讲述所在章节
取串	LODS OPRD 一般不加重复前缀	AL/AX←[SI] SI←SI±1	OPRD 源串地址	第六章第一节
	LODSB/LODSW 一般不加重复前缀	同上，仅B/W指明字节/字	无	第六章第一节
存串	STOS OPRD 可加重复前缀	[DI]←AL/AX DI←DI±1	OPRD 目的串地址	第六章第一节
	STOSB/STOSW 可加重复前缀	同上，仅B/W指明字节/字	无	第六章第一节

五、程序控制指令组

程序控制指令用来操作指针指令 IP和代码段寄存器 CS，改变它们的内容以引起程序执行顺序的变化。包括无条件转移指令，条件转移指令、迭代控 制指令和与中断有关的指令。

这组指令的查阅表如表 5。

表 5

类别	指令格式	功能	允许的操作数	讲述所在章节
无条件转移	CALL OPRD	调用过程或子程序	OPRD 远/近过程名，标号， 通用寄存器，存贮器	第四章第五节
	RET OPRD	从过程或子程序返回。若OPRD存在，则SP←SP+OPRD	OPRD可没有或等于偶数n。	第四章第五节
	JMP OPRD	无条件转移	OPRD可为标号，存贮器， 寄存器	第四章第三节

表 5　（续）

类别	指令铭式	功能	允许的操作数	讲述所在章节
条件转移	JA/JNBE OPRD	高于/不低于也不等于时，转移	OPRD为近标号	同上
	JAE/JNB OPRD	高于或等于/不低于时转移	同上	同上
	JB/JNAE OPRD	低于/不高于也不等 时转移	同上	同上
	JBE/JNA OPRD	低于或等于/不高于时转移	同上	同上
	JC OPRD	有进(借)位时(CF=1)转移	同上	同上
	JE/JZ OPRD	相等/等于0时转移	同上	同上
	JG/JNLE OPRD	大于/不小于也不等于时转移	同上	同上
	JGE/JNL OPRD	大于或等于/不小于时转移	同上	同上
	JL/JNGE OPRD	小于/不大于也不等于时转移	同上	同上
	JLE/JNG OPRD	小于或等于/不大于时转移	同上	同上
	JNC OPRD	无进(借)位时转移	同上	同上
	JNZ/JNE OPRD	不等于0/不相等时转移	同上	
	JNO OPRD	不溢出时转移	同上	同上
	JNP/JPO OPRD	非奇偶/奇校验时转移	同上	同上
	JNS OPRD	正数时转移	同上	同上

表 5　（续）

类别	指令铭式	功能	允许的操作数	讲述所在章节
条件转移	JO OPRD	溢出时转移	同上	同上
	JP/JPE OPRD	奇偶/偶校验时转移	同上	同上
	JS OPRD	负数时转移	同上	同上
	JCXZ OPRD	若CX=0转移	同上	第四章第四节
迭代循环	LOOP OPRD	循环至 CX=0	同上	同上
	LOOPZ/LOOPE OPRD	等于零/相等时循环	同上	同上
	LOOPNE/LOOPNZ OPRD	不相等/不等于零时循环	同上	同上
中断	INT n	软中断	n为 8位的立即数	第五章第二节
	INTO	溢出中断	无	第五章第二节
	IRET	从中断返回	无	第五章第二节

六、处理器控制指令组

8086/8088 所提供的这组指令允许程序去控制各种 CPU 的功能。包括修改标志，使 8086/8088 与外部事务同步，以及空操作指令。

本组指令查阅表如表 6 。

表 6

类　别	指令格式	功　能	讲述所在章节
标志操作	CLC	进位标志清0	第二章第二节

表 6 （续）

标志操作	CMC	进位位取反	第二章第二节
	STC	进位位置1	第二章第二节
	CLD	DF标志清0	第二章第二节
	STD	DF标志置1	第二章第二节
	CLI	IF标志清0	第二章第二节
	STI	IF标志置 1	第二章第二节
其 它	HLT	处理器暂停	
	WAIT	处理器等待	
	ESC	交权指令	
	LOCK	总线封锁	
	NOP	空操作	

附录 B IBM PC ASCII 码字符表

低四位 B H \ 高四位 B H	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0	BLANK (NULL)	►	BLANK (SPACE)	0	@	P	‘	p	Ç	É	á	⋮	└	┘	∝	≡
0001	1	☺	◄	!	1	A	Q	a	q	ü	æ	í	⋮	┴	┬	β	±
0010	2	☻	↕	"	2	B	R	b	r	é	Æ	ó	⋮	├	┼	Γ	≥
0011	3	♥	‼	#	3	C	S	c	s	â	ô	ú	┌	┤	┐	π	≤
0100	4	♦	¶	$	4	D	T	d	t	ä	ö	ñ	─	═	╒	Σ	⌠
0101	5	♣	§	%	5	E	U	e	u	à	ò	Ñ	═	╞	╓	σ	⌡
0110	6	♠	▬	&	6	F	V	f	v	å	û	ª	╟	╠	╔	μ	÷
0111	7	•	↨	’	7	G	W	g	w	ç	ù	º	╫	╬	╩	τ	≈
1000	8	■	↑	(	8	H	X	h	x	ê	ÿ	¿	╪	╧	┴	Φ	°
1001	9	○	↓	)	9	I	Y	i	y	ë	Ö	⌐	┘	┌	┘	Θ	•
1010	A	◙	→	*	:	J	Z	j	z	è	Ü	¬	┌	┐	┌	Ω	•
1011	B	♂	←	+	;	K	[	k	{	ï	¢	½	┐	┘	▌	δ	√
1100	C	♀	∟	,	<	L	\	l	¦	î	£	¼	┘	┌	▐	∞	n
1101	D	♪	↔	—	=	M	]	m	}	ì	¥	¡	┌	┐	▀	φ	2
1110	E	♫	▲	.	>	N	^	n	~	Ä	₧	«	┐	┘	▐	∈	■
1111	F	☼	▼	/	?	O	_	o	⌂	Å	ƒ	»	┘	┌	▀	∩	BLANK FF

参 考 资 料

(1) The IBM Personal Computer Technical Reference Manual, IBM Corp., 1983.

(2) Disk Operating System, Microsoft Inc., 1983.

(3) MCS-86 Macro Assembly language Reference Manual, Intel Corp.

(4) Intel 8086 Family User's Manual, Intel Corp.

(5) 微型计算机 IBM PC (0520) 系统原理与应用，周明德主编，清华大学出版社出版，1985。

(6) IBM 汇编语言程序设计，IBM PC 译丛，刘明烈译，辽宁省电子计算机学会。

(7) MCS-86TM 宏汇编语言参考手册，北京工业大学无线电电子学系。

(8) IBM PC/XT 硬件技术手册，IBM PC 译丛，康保祥等译，辽宁省电子计算机学会。

(9) 微型计算机 IBM PC 的原理与应用，张福炎等编，南京大学出版社出版 1984。

(10) 8086 微计算机系列用户手册，(上、中、下册),<<舰船导航>>编辑部，1982.3。

(11) 微型计算机丛书 (软件) ，<<电子计算机动态>> 编辑部（中国科学院计算所)，1981。

(12) 微型计算机硬件、软件及其应用，周明德，清华大学出版社，1982。

(13) 微型计算机 - Z80 ，科学技术出版社重庆分社，1979。

(14) CCBIOS 分析，钱培德著，中国科学院声学所，1985。

(15) PC DOS 分析报告，电子工业部六所，1984。

(16) Z80 汇编语言程序设计，科学文献出版社重庆分社，1981。

(17) WORDSTAR 文书处理，蔡源斌、朝基合译。

[General Information]
书名=宏汇编语言程序设计
作者=
页数=291
SS号=10011277
出版日期=