第 9 章 汇编语言程序实验指导

汇编语言源程序实验指导主要由两部分组成，第一部分介绍使用 DEBUG 调试程序输入汇编指令和程序段并调试运行的方法，以及用 DEBUG 调试程序反汇编可执行文件并进行修改、调试和运行的方法；第二部分介绍介绍用宏汇编软件 MASM 对完整的汇编语言源程序进行汇编和链接生成可执行文件的方法。同学们学习了部分指令后即可在 DEBUG 调试程序下测试这些指令的功能；学习了完整的汇编语言源程序的编写方法后，就可用汇编软件对源程序进行进行汇编和链接生成可执行文件，并在 DEBUG 等环境下调试运行。

9.1 DEBUG 调试程序的使用方法

DEBUG 程序是 DOS 操作系统下输入、调试和运行汇编语言程序的工具软件，该软件能使用户接触到计算机内部，允许用户直接观察和修改 CPU 的寄存器；能观察、修改内存单元；允许直接输入机器指令并单步执行；能反汇编程序。可以说，DEBUG 是观察和了解计算机内部运行情况的有利助手。在 Windows 操作系统下的 DOS 提示符下也可以运行 DEBUG 程序来编写、调试和运行汇编语言程序。所谓 DOS 命令提示符，实际上就是虚拟 8086 模式，因此在 DEBUG 下只能运行 16 位机的指令。另外，要注意的是，DEBUG 中的数的表示都是十六进制，且十六进制数后面不用 H 表示，以字母开头的十六进制数前面也可以不以 0 开头；DEBUG 中不允许用标识符；DEBUG 中不区分大小写，大小写可以混合使用。

9.1.1 DEBUG 程序的启动

在 DOS 的提示符下，输入如下命令可启动 DEBUG 调试程序：

```
C:\>DEBUG [文件名][参数1][参数2]
```

其中，DEBUG 是调试程序的文件名，后面用[ ]表示的内容都是可选项，可以给出，也可以不给。[文件名]是要调试的程序的文件名，若在命令中规定了文件名，则在 DOS 把 DEBUG 程序调入内存后，DEBUG 程序根据文件名把要调试的程序调入内存；若在命令中没有规定文件名，则 DEBUG 程序与正在内存中的内容打交道，可以输入程序进行调试、运行，也可以用 N 和 L 命令，从键盘上输入要调试的程序。[参数 1][参数 2]是被调试程序所需要的参数。

（1）不带文件名的启动

在 DOS 提示符下输入“DEBUG”，再按回车键可启动 DEBUG 程序，此时 DEBUG 命令后面没有指定要调试的文件名。按回车键后屏幕上出现 DEBUG 提示符“-”，这时就可以输入 DEBUG 的各种命令了，如图 9-1 所示。图中启动 DEBUG 后，输入了 DEBUG 的 R 命令，在屏幕上显示了此时所有寄存器的内容。

```
C:\Users\ZHOUJI~1\ZJY>debug
-r
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=17D7  ES=17D7  SS=17D7  CS=17D7  IP=0100   NV UP EI PL NZ NA PO NC
17D7:0100 5A            POP     DX
-r bx
BX 0000
:
-r dx
DX 0000
:5060
-r f
NV UP EI PL NZ NA PO NC  -ac PE OV
-R F
OV UP EI PL NZ AC PE NC  -
-
```

图 9-1　DEBUG 启动时寄存器的值

图 9-1 中以不带文件名的方式启动 DEBUG，各寄存器的值如下：所有段寄存器的值都相等，指向当前可用的主存段，即 DEBUG 程序末尾的第一个段的段边界值；除 SP 之外的通用寄存器都设置为 0，SP 指向栈顶 0FFEEH；IP=0100H；标志位中除 IF 标志位为 1 表示允许中断外，其他都为 0。

图 9-1 中，标志寄存器中标志位的值是用两个字母表示的，DEBUG 中，这些字母组合与标志位的对应关系如表 9-1 所示。图 9-1 中只给出了 8 个标志位的值，还有一个追踪标志位 T 不能直接用指令改变，在 DEBUG 中不显示。

表 9-1　　标志位的值与字母组合对应关系

标　志　位	置位（值为 1）	复位（值为 0）
溢出 OF（Overflow）	OV	NV（Not Overflow）
方向 DF（Direction）	DN（Down）	UP（增量修改地址）
中断 IF（Interrupt）	EI（Enable Interrupt）	DI（Disable Interrupt）
符号 SF（Sign）	NG（Negative）	PL（Plus）
零（Zero）	ZR（Zero）	NZ（Non Zero）
辅助进位（Auxiliary carry）	AC（Auxiliary carry）	NA（Non AC）
奇偶（Parity）	PE（Parity Even）	PO（Parity Odd）
进位（Carry）	CY（Carry）	NC（Non Carry）

不管以哪种方式启动 DEBUG，在出现提示符“-”后输入“？”，屏幕上会显示 DEBUG 的所有命令及格式，如图 9-2 所示。

DEBUG 命令后没有指定要调试文件的名字，按回车键进入 DEBUG 调试程序后，我们可以用 A 命令输入指令或程序段，然后调试、运行；还可以用 N 命令给输入的程序起名字，然后用 W 命令保存所输入的程序段到指定名字的文件中；或者用 N 命令后面加要调试运行的程序的名字，之后用 L 命令把该名字所指定的程序调入内存进行调试运行。

```
-?
assemble     A [address]
compare      C range address
dump         D [range]
enter        E address [list]
fill         F range list
go           G [=address] [addresses]
hex          H value1 value2
input        I port
load         L [address] [drive] [firstsector] [number]
move         M range address
name         N [pathname] [arglist]
output       O port byte
proceed      P [=address] [number]
quit         Q
register     R [register]
search       S range list
trace        T [=address] [value]
unassemble   U [range]
write        W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

图 9-2　DEBUG 的所有命令

（2）带文件名的启动

如果要调试某个已有的可执行程序，也可以在启动 DEBUG 时直接在 DEBUG 后面输入文件名启动，即在 DOS 提示符下输入 DEBUG 和文件名，再按回车键即可，如图 9-3 所示。

```
C:\Users\zhou jieying\ZJY>DEBUG F1.COM
-
```

图 9-3　DEBUG 带文件名的启动

9.1.2　DEBUG 命令的格式

图 9-2 中列出了 DEBUG 中的所有命令及格式。DEBUG 命令都是以一个字母开始，后面可以跟一个或多个参数，也可以不带参数。图 9-2 中，大写字母写的是命令，命令后面的小写字母是参数。实际输入时，命令和参数可以用大写、小写或混合方式输入，也即 DEBUG 中不区分大小写。图 9-2 中，有些参数放在[ ]中，则该参数是可选项，可以给出，也可以不给出。命令和参数间，可以用定界符（空格或逗号）分隔。但定界符只是在两个邻接的十六进制值数之间是必须的。因此，下列命令是等效的：

```
dds : 200 220;   d ds : 200 220 ;  d ,ds : 200,220
```

其中 d 是显示内存单元内容的命令。

DEBUG 命令输入完后，按回车键开始执行。若一个命令产生相当大数量的输出行，为了能在屏幕上读清楚，可以在显示过程中按【Ctrl+Num Lock】键，暂停屏幕滚动；按任何一个字符来继续输出。执行期间，可以按【Ctrl+Break】键来停止一个命令的执行，返回 DEBUG 提示符。

DEBUG 若检查到语法错误，会显示具有错误的行并指示错误所在，如图 9-4 所示。

在图 9-4 中，用 A 命令输入指令并汇编。输入完第 1 条指令后，DEBUG 指示有错误并指出了错误所在位置。错误的原因是，把数送给 8 位的累加器 AL，数不应该超过 2 位十六进制数，而这里加上 0 一起是 3 位。后面一条指令做了修改，去掉了最

```
C:\Users\zhou jieying>debug
-a
17D7:0100 mov al,0cf
                  ^ Error
17D7:0100 mov al,cf
17D7:0102 mov ax,0cf
17D7:0105 -
```

图 9-4　DEBUG 指示语法错误

前面的 0 就正确了。第 3 条指令是把与第 1 条指令相同的 3 位十六进制数送给 16 位的累加器，这个是可以的，因为 16 位的累加器可以存放 4 位十六进制数。

图 9-2 中给出的命令的参数格式中，出现得最多的是 address 和 range，下面先对这两个参数作些说明，其他的参数在介绍具体命令时再说明。

命令后面的参数 address 是指内存中的地址，用逻辑地址“段值:偏移量”表示，段值可以用段寄存器或数值表示，偏移量用十六进制数表示。可以只给偏移量而不给出段值，这个时候就用默认的段值。在图 9-2 中有些 address 放在[]中，即表示为[address]，这个时候可以不给出地址，虽然这些命令是要对内存单元操作，此时内存单元的段值和偏移量都用默认值。

图 9-2 中命令后面的参数 range，是指命令所操作的内存地址范围，有两种表示形式。第 1 种是“首地址:末地址”，规定末地址中的段值必须和首地址中的段值一样，因此，末地址中不写段值，如果写段值就是错误的，DEBUG 中会指出错误所在。首地址也是一种地址（Address），前面关于地址的用法在这里也都适合，如首地址可以只给出偏移量，段值就用默认的段值。第 2 种是“首地址 L 字节数”，L（Lenghth）是长度的意思，也可以只给出首地址，长度用默认值。图 9-2 中有些 range 放在[ ]中，即表示为[range]。这个时候可以不给出内存地址范围，虽然这些命令是要对内存中某一范围单元进行操作，此时内存单元的范围用默认范围。

9.1.3　DEBUG 常用命令介绍

1. 寄存器命令 R（Register）

R 命令可用来检查寄存器的内容，或修改寄存器的内容。R 命令的格式为：R [register]。

R 命令后面若不给出寄存器的名字，则显示所有寄存器的内容，如图 9-1 所示。图 9-1 中 R 命令后的前两行显示了 CPU 内部所有寄存器的内容，包括标志寄存器中 8 个标志位的内容，第 3 行显示了现行的 CS:IP 所指的指令机器码和反汇编符号，这就是下一条即将要执行的指令。

R 后面若给出寄存器的名字，则可以显示并修改该寄存器的内容，如图 9-1 所示。R 命令后面只能跟 16 位寄存器的名字，包括 16 位的通用寄存器和控制寄存器（如 IP 和 F）。如果不修改，直接按回车键即可。如果要修改，除了标志寄存器外，其他寄存器都可以输入 1～4 位十六进制数，再按回车键，实现修改。如果是标志寄存器，直接输入要修改的标志位的相反的字母组合即可，可以同时修改多个标志位，字母组合间可以没有间隔，而且字母组合的顺序也没有限制。

2. 显示命令 D（Dump）

D 命令用来显示内存单元的内容，D 命令的格式为：D [range]。下面结合例子来说明 D 命令和 range 的用法，如图 9-5 所示。第 1 个例子中指示有错误，并指出了错误的位置所在，错误原因在于指明范围的末地址用了段值。在第 2 个例子中去掉末地址的段值就正确了。第 3 个例子用了首地址加长度的方法指定显示范围，范围是 0200:0300(H)开始的 50(H)个字节，其末地址是 0200:034F(H)。第 4 个例子中没有指定范围，因此用默认范围。默认范围是，首地址接着上面显示完的继续，即从 0200:0350(H)开始，长度是 8 行，每行 16 个字节，共 128 个字节。

```
-D 200:300 400:350
             ^ Error
-D 200:300 350
0200:0300  3D 3E 3F 08 00 0F 00 00-00 00 00 00 10 0E 00 FF   =>?.............
0200:0310  7F 60 0F 00 00 00 00 07-02 08 FF 0E 00 00 3F 00   .`............?.
0200:0320  08 00 FC 02 74 70 00 00-00 00 DF DF DF 00 00 00   ....tp..........
0200:0330  EC EC EC 00 00 00 FF 00-00 00 00 00 00 00 00 00   ................
0200:0340  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0350  00                                                .
-D 200:300 L 50
0200:0300  3D 3E 3F 08 00 0F 00 00-00 00 00 00 10 0E 00 FF   =>?.............
0200:0310  7F 60 0F 00 00 00 00 07-02 08 FF 0E 00 00 3F 00   .`............?.
0200:0320  08 00 FC 02 74 70 00 00-00 00 DF DF DF 00 00 00   ....tp..........
0200:0330  EC EC EC 00 00 00 FF 00-00 00 00 00 00 00 00 00   ................
0200:0340  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-D
0200:0350  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0360  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0370  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0380  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0390  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:03A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:03B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:03C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-
```

图 9-5　D 命令的应用举例（一）

下面再看图 9-5 中的内容是怎么显示的。第 2 个例子正确显示后，其第 1 行显示的是首地址开始的 16 个单元中的内容。第 1 行最左边给出第 1 个单元的逻辑地址 0200:0300（H），第 1 行的第 1 个数 3D(H)就是第 1 个单元的内容，接着后面的数也就是首地址之后的那些单元的内容，总共 16 个单元。16 个单元的中间也就是第 8 个和第 9 个单元之间用一个横杠“-”隔开。一行 16 个单元，因此下一行的首地址从 0200:0310(H)开始。注意，DEBUG 中所有的地址和数都是用十六进制数表示的。依次往下显示到 350(H)单元结束。图 9-4 的右边是以该单元的数为 ASCII 码所对应的字符，也是一行显示 16 个字符。例如，0200:0300(H)单元的内容是 3D(H)，它是“=”的 ASCII 码；0200:0301(H)单元的内容是 3E(H)，它是“>”的 ASCII 码；0200:0302(H)单元的内容是 3F(H)，它是“?”的 ASCII 码。这些符号按顺序在那一行显示出来，对应的符号不能显示的，则用“.”显示。例如，0200:0303(H)单元的内容是 08(H)，它是控制符 BS（Back Space，退格）的 ASCII 码，在屏幕上显示不了，用“.”表示。

D 命令另外两种给出范围的方法如图 9-6 所示。两个例子都是只给了首地址，长度用了默认值，即 128 个字节。第 1 个例子的首地址只给出了偏移量，段值用默认值，即当前 DS 寄存器的内容。第 2 个例子的首地址给出了段值和偏移量。

```
-d 300
17D7:0300  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0310  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0320  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0330  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0340  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0350  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0360  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0370  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-D 0200:300
0200:0300  3D 3E 3F 08 00 0F 00 00-00 00 00 00 10 0E 00 FF   =>?.............
0200:0310  7F 60 0F 00 00 00 00 07-02 08 FF 0E 00 00 3F 00   .`............?.
0200:0320  08 00 FC 02 74 70 00 00-00 00 DF DF DF 00 00 00   ....tp..........
0200:0330  EC EC EC 00 00 00 FF 00-00 00 00 00 00 00 00 00   ................
0200:0340  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0350  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0360  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
0200:0370  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-_
```

图 9-6　D 命令的应用举例（二）

3. 汇编命令 A（Assemble）

A 命令用于小段程序的输入及汇编，或者用于修改目标程序。A 命令的格式为：A [address]。A 命令是一个输入汇编语句（包括指令语句和少数的伪指令语句）并对汇编语句进行汇编（将汇编语句翻译成机器代码）的命令。

A 命令的功能是从指定的地址开始，输入汇编语言的语句并将语句汇编成机器码存放在指定单元。每输入一条语句汇编一条语句并存放，相邻的语句连续存放。如果 A 命令中没有给出地址，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 单元开始连续存放。

A 命令可以用来输入新的程序，也可以用来对被调试程序进行修改或增补，并在 DEBUG 下汇编、运行和调试，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。

下面结合一个简单的程序，来看看 DEBUG 的 A 命令的用法。

【例 1.1】 假设要运行的程序如下。

```
MOV     AL, 8           ; AL←8
ADD     AL, 9           ;AL←(AL)+9
INT     3               ;设置断点
```

可以首先启动 DEBUG，然后用 A 命令输入语句并汇编，输入完程序后，可以用 W 命令来保存程序，如图 9-7 所示；保存好后在需要时可以用 L 命令再从外存调入内存，并用 U 命令反汇编检查是否与原来的内容相符，如图 9-8 所示。

```
C:\Users\ZHOUJI~1\ZJY>DEBUG
-A
17D7:0100 MOV AL,8
17D7:0102 ADD AL,9
17D7:0104 INT 3
17D7:0105
-N F2.COM
-R CX
CX 0000
:5
-W
Writing 00005 bytes
-
```

图 9-7　A 命令输入及 W 命令保存

```
-N F2.COM
-L 200:300
          ^ Error
-L CS:300
          ^ Error
-L CS:100
-U
17E8:0100 B008          MOV     AL,08
17E8:0102 0409          ADD     AL,09
17E8:0104 CC            INT     3
```

图 9-8　用 L 命令装入程序

在 A 命令下，除了可以输入指令语句外，还可以用两个最常用的伪指令 DB 和 DW，如图 9-9 所示。A 命令下所有输入的数字均默认为十六进制，并且不需要以 h 字符结尾。直接在 A 命令下输入跳转指令时，跳转指令后应跟跳转到的语句所在内存单元的地址，不能跟标号；使用跳转指令时，可以指定跳转指令的属性 near 和 far。near 前缀可以缩写为 ne。

```
-a
17D7:0105 db 1,2,3,4,5,6
17D7:010B dw 7,8,9,a,b,c,d,ef
17D7:011B
-
```

图 9-9　A 命令中 DB、DW 的应用

4. 反汇编命令 U（Unassemble）

U 命令也可以将某一段内存区域的内容反汇编，显示该区域二进制机器代码对应的汇编指令。U 命令的格式为：U[range]。

如果 U 命令后给出 range，则将指定地址范围的内存单元的机器代码翻译成汇编指令，便于阅读。Range 的指定方法在前面已介绍过，在 D 命令中也举例讨论过。

若在 U 命令中没有指定地址范围，则以上一个 U 命令的最后一条指令地址后的下一个单元作为起始地址；若没有输入过 U 命令，则起始地址由 DEBUG 初始化的段寄存器的值为段值、以 0100H 作为地址偏移量；两种情况下，反汇编的字节数都为 16 个字节（40 列显示方式）或 32 个字节（80 列显示方式）。

U 命令显示的数据中，第 1 列给出逻辑地址，第 2 列给出机器代码，第 3 列给出该机器代码对应的汇编语句。如图 9-8 所示，U 命令后的第 1 行中，首先给出内存单元的逻辑地址 17E8:0000(H)；

之后给出两个字节的数据：B008(H)，对应的是 17E8:0000(H)和 17E8:0001(H)单元的内容，也是随后给出的汇编语句的机器代码；最后给出一条汇编指令“MOV AL, 08”，即机器代码 B008(H)对应的汇编指令。其他各行以此类推。

5. 写命令 W（Write）

W 命令的功能是把内存中一段区域的数据写入到磁盘指定区域中或写入到某个文件中。W 命令的格式为：W　[address][drive][first sector][number]。

W 命令后面的每个参数都是可选项。其中[address]是要保存的一段数据在内存单元的起始地址；如果 W 命令后没有给出地址，则以 CS:100(H)为起始地址。这段数据所占内存单元的字节数要放在 BX 和 CX 寄存器中，BX 中放高 16 位，CX 中放低 16 位。这样，起始地址加长度就指明了一段内存区域。

如果指定的内存区域的数据要直接写到磁盘的指定区域中的话，则在 W 命令后要给出后面的 3 个参数，即[drive][first sector][number]，对应中文表示为[驱动器][第 1 个扇区][扇区的个数]，也就是说要给出是哪个磁盘、磁盘的第 1 个扇区、共几个扇区的详细信息，指明磁盘上的一段连续扇区来存放信息。这种操作要十分小心，因为若有差错，就会破坏盘上的原有内容。建议没有把握不要尝试这一操作，也就是不用给出这 3 个参数。

如果指定的内存区域的数据要写到文件中（这是保存数据的常用方法），则在 W 命令执行前，要先用 N 命令指定要保存数据的文件名，文件的扩展名可以为.COM，但不能为.EXE 或.HEX；文件也可以没有扩展名。

回过头来再来看看图 9-7 中将内存中的数据保存到文件的方法。首先用 A 命令输入并汇编了 3 条指令，这 3 条指令翻译成机器码后存放在内存 17D7:0100～17D7:0104 的共 5 个单元中。为了保存输入的程序段到文件中，需用 W 命令。在执行 W 命令前，用 N 命令指出后面要操作的文件名为 F2.COM，并用 R 命令将 CX 寄存器的内容改为文件的长度 5，也就是要将 5 个字节写到文件中。之后输入 W 命令，就将 3 条指令共 5 个字节写入到 F2.COM 文件中了。

在图 9-7 的例子中，W 命令后面没有跟任何参数，但是完成了写操作。W 命令后面没有给出地址，是因为要写入的数据正好在从 CS:100(H)单元开始的一段内存区域中（CS 的值为 17D7），否则 W 命令后需要指明地址。W 命令后面没有给出下列参数：[drive][first sector][number]，是因为这里不是将数据写到磁盘指定的扇区，而是写到文件。

在图 9-7 的例子中，只是用 R 命令将 CX 寄存器的内容改为了文件的长度 5，没有修改 BX 寄存器的内容，这是因为刚进入 DEBUG 时已将 BX 寄存器的内容置为 0，如果不确切知道 BX 的内容，需要检查并按照文件的实际长度修改。

6. 装入命令 L（Load）

L 命令的功能正好与 W 命令的功能相反，它是将磁盘指定区域或指定文件中的数据装到内存中的一段区域中。L 命令的格式为：L　[address][drive][first sector][number]。

可以看到，L 命令的参数与 W 命令的参数是一样的。其中[address]是指出要将数据装入到内存中的首地址，是可选项，若命令中没有规定地址，则数据装入从 S:100(H)开始的内存区域中。[drive][first sector][number]3 参数也是在将磁盘指定区域的数据装入内存时用的，如果是从文件装入数据则不需要给出这 3 参数。如果从文件装入数据，则需要在 L 命令执行前用 N 命令指定要装载的文件的名称。

需注意的是，如果 N 命令指定装载的文件的扩展名为.COM，则数据或程序装入从 CS:100(H)开始的内存区域中。此时，L 命令后可以不给出地址，如果要给的话，也只能是 CS:100 或 100。

如果已知 CS 的值，也可以写出值，例如，若 CS 的值为 17D7，也可以写为 L 17D7:100。如果在命令中给出其他就是错误的，DEBUG 会指出错误，修改后继续，如图 9-8 所示。

如果 N 命令指定装载的文件的扩展名为.EXE，则数据或程序装入从 CS:0000(H)开始的内存区域中。此时，L 命令后可以不给出地址，给出地址的话会被忽略或者指出有错。

文件装入内存后，在 BX 和 CX 中包含所装入的文件的字节数。但如果所装入的文件的扩展名为.EXE，则 BX 和 CX 中包含实际的程序长度，而不是文件的大小。

7. 运行命令 G（Go）

程序输入内存或以文件形式装入内存后，可以用 G 命令运行程序，检查程序的运行结果。G 命令的格式为：G [=address][addresses]。

其中，第 1 个参数=address 规定执行的起始地址，如不给出，则以当前的 CS:IP 为运行的起始地址；后面的参数 addresses 为程序运行的断点地址，最多可以有 10 个断点，也可以没有，断点间以空格或逗号隔开；地址中如果没有给出段值，则以 CS 为段值。

当程序执行到一个断点地址时，就停下来，显示 CPU 内部所有寄存器的内容和全部标志位的状态；被调试程序的所有断点处的指令被恢复；全部断点被取消；返回 DEBUG。这样的话，实际上设 10 个断点地址也就会在一个断点停下来。因此，如果一个断点设在指令的中间是不会停下来的，如图 9-10 所示。

```
C:\Users\ZHOUJI~1\ZJY>DEBUG F1.COM
-U 100 L 5
17E8:0100 B002          MOV     AL,02
17E8:0102 00D8          ADD     AL,BL
17E8:0104 CC            INT     3
-G 101 102 103 104

AX=00CC  BX=0000  CX=0005  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=17E8  ES=17E8  SS=17E8  CS=17E8  IP=0102   NV UP EI PL NZ NA PO NC
17E8:0102 00D8          ADD     AL,BL
-
```

图 9-10　断点的设置

图 9-10 中，断点 101 是在指令 MOV AL,02 的中间是不会停下来的。如果确切地知道是要在哪条指令前停下来的话，可以只设一个断点，如 G=100 102，从 100H 开始到 102H 停下来。断点实际上就是断点中断指令 INT 3，也可以在程序中以 INT 3 设置断点。调试程序时可以采用 INT 3 作为程序的最后一条指令，这样比较好检查结果。

DEBUG 中返回 DOS 操作系统的指令是 INT 20(H)，程序运行遇到这条指令会正常结束，显示信息“Program terminated normally”，返回 DOS 操作系统，回到 DEBUG 提示符下。但如果以 INT 20(H)作为程序的最后一条指令的话，程序正常结束并返回 DOS 操作系统后，有些结果就清除了，也就是说有时候想要看的结果已经没有了。因此，调试时常以 INT 3 作为最后一条指令。

G 命令中的起始地址参数所指的单元，必须包含有效的 8086/8088 的指令码，若指定的地址单元不包含有效指令的第 1 个字节，则会出现不可预料的结果。

8. 追踪命令 T（Trace）

如果要程序运行时每执行一条指令就停下来检查结果，可以用 T 命令。T 命令的格式是：

T[=address][value]，两个参数都是可选项。

T 命令中的参数=address 与 G 命令中的作用相同，指明执行的起始地址；参数 value 用于指明执行几条指令后停下来。若在 T 命令中没有指定起始地址，则从 CS : IP 的现行值执行；若在 T 命令中没有给出值，则默认为 1，如图 9-11 所示。

```
-t 2

AX=0008  BX=0000  CX=0005  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=17E8  ES=17E8  SS=17E8  CS=17E8  IP=0102   NV UP EI PL NZ NA PO NC
17E8:0102 0409           ADD     AL,09

AX=0011  BX=0000  CX=0005  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=17E8  ES=17E8  SS=17E8  CS=17E8  IP=0104   NV UP EI PL NZ AC PE NC
17E8:0104 CC             INT     3
-t

AX=0011  BX=0000  CX=0005  DX=0000  SP=FFF8  BP=0000  SI=0000  DI=0000
DS=17E8  ES=17E8  SS=17E8  CS=123B  IP=13B1   NV UP DI PL NZ AC PE NC
123B:13B1 55             PUSH    BP
-
```

图 9-11　T 命令的执行

图 9-11 中，第 1 条 T 命令以当前的 CS:IP 开始运行，运行两条指令后停下来，并显示这两条指令的运行结果，可以看到 IP 的值跟着修改；第 2 条 T 命令以当前的 CS:IP（即修改后的）开始运行，运行了一条指令就停下来，并显示这一条指令的运行结果。

T 命令实际上是利用了处理器的单步中断，也常称为单步执行或单步命令；而 G 命令则称为连续执行命令。T 命令遇到调子程序（CALL）、中断调用（INT N）以及重复指令时，会进入到子程序、中断服务子程序及重复指令内部执行。因此，碰到调子程序或中断调用等情况时，如果不是要看这些指令内部的具体执行情况，建议不要用 T 命令执行，而是改用下面要介绍的 P 命令。

9. 继续命令 P（Proceed）

P 命令的格式是：P[=address][number]，两个参数都是可选项。

P 命令的用法与 T 命令类似，只是不会进入到子程序或中断服务子程序中执行。其中：[=address]指定第 1 条要执行指令的位置，如果不指定地址，则默认地址是在 CS:IP 寄存器中指定的当前地址；[number]指定在将控制返回给 DEBUG 之前要执行的指令数，默认值为 1。

10. 退出命令 Q（Quit）

退出 DEBUG 时，只需在 DEBUG 提示符“-”后面输入“Q”即可。

11. 修改内存单元内容的命令 E（Enter）

E 命令用于修改内存单元的内容，其格式为：E　address [list]。

其中，address 为要修改内容的内存单元的地址，或内存区域的首地址；[list]为向这些内存单元输入的数据列表，每个单元一个字节，因此每个数据不能超过一个字节的表示范围，每个数之间用空格或逗号隔开。[list]是可选项，如果不给出则一个一个输入，每输入一个按空格键就可以接着输入下一个单元的数据，要退出输入则按回车键，如图 9-12 所示。

```
C:\Users\ZHOUJI~1\ZJY>DEBUG
-E 100 23,34,56,7,8,1 2
-D 100 L10
17D7:0100  23 34 56 07 08 01 02 31-46 31 00 00 01 B0 04 D7   #4V....1F1......
-E 200
17D7:0200  00.1    00.2    00.3    00.4    00.5    00.6    00.7    00.8
-D 200 L 10
17D7:0200  01 02 03 04 05 06 07 08-00 00 00 00 00 00 00 00   ................
-
```

图 9-12　E 命令的用法

12. 填充命令 F（Fill）

F 命令用于给一段内存区域填充数据，或是修改这段内存区域的内容。F 命令的格式为：F range List。

range 的用法与前面 D 命令中相同，List 为要填入的数据列表，用法与 E 命令中相同。如果 List 所含的字节数比 range 要求的少，则 List 被重复使用，如果 List 所含的字节数比 range 要求的

大，则多余的被略去，如图 9-13 所示。

```
C:\Users\ZHOUJI~1\ZJY>DEBUG
-D 500 520
17D7:0500  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0510  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
17D7:0520  00                                                .
-F 500 50F 1,2,3,4
-F 510 L 6 11 22 33 44 55 66 77 88 99
-D 500 520
17D7:0500  01 02 03 04 01 02 03 04-01 02 03 04 01 02 03 04   ................
17D7:0510  11 22 33 44 55 66 00 00-00 00 00 00 00 00 00 00   ."3DUf..........
17D7:0520  00                                                .
-
```

图 9-13　F 命令的用法

13. 传送命令 M（Move）

M 命令的功能是把一段内存区域的内容拷贝到另一段内存区域。M 命令的格式为：M　range address。其中，range 指定要被拷贝的内存区域的地址范围，address 指定要拷贝去的内存区域的首地址。图 9-14 给出两个 M 命令的用法举例。

```
C:\Users\ZHOUJI~1\ZJY>DEBUG F21.COM
-U 10F 120
17E8:010F B409          MOV     AH,09
17E8:0111 BA0001        MOV     DX,0100
17E8:0114 CD21          INT     21
17E8:0116 CD20          INT     20
17E8:0118 0000          ADD     [BX+SI],AL
17E8:011A 0000          ADD     [BX+SI],AL
17E8:011C 0000          ADD     [BX+SI],AL
17E8:011E 0000          ADD     [BX+SI],AL
17E8:0120 0000          ADD     [BX+SI],AL
-M 10F 118 118
-U 10F 120
17E8:010F B409          MOV     AH,09
17E8:0111 BA0001        MOV     DX,0100
17E8:0114 CD21          INT     21
17E8:0116 CD20          INT     20
17E8:0118 B409          MOV     AH,09
17E8:011A BA0001        MOV     DX,0100
17E8:011D CD21          INT     21
17E8:011F CD20          INT     20
-
```

```
C:\Users\zhou jieying\ZJY>DEBUG F21.COM
-U 10F 120
17E8:010F B409          MOV     AH,09
17E8:0111 BA0001        MOV     DX,0100
17E8:0114 CD21          INT     21
17E8:0116 CD20          INT     20
17E8:0118 0000          ADD     [BX+SI],AL
17E8:011A 0000          ADD     [BX+SI],AL
17E8:011C 0000          ADD     [BX+SI],AL
17E8:011E 0000          ADD     [BX+SI],AL
17E8:0120 0000          ADD     [BX+SI],AL
-M 10F L 9 118
-U 10F 120
17E8:010F B409          MOV     AH,09
17E8:0111 BA0001        MOV     DX,0100
17E8:0114 CD21          INT     21
17E8:0116 CD20          INT     20
17E8:0118 B409          MOV     AH,09
17E8:011A BA0001        MOV     DX,0100
17E8:011D CD21          INT     21
```

图 9-14　M 命令的用法举例

14. 比较命令 C（Compare）

C 命令的功能是比较两段内存区域中的内容是否相同，若不同则按字节显示其地址和内容，若相同则不显示任何内容。

C 命令的格式为：C range address。其中，range 指定要被比较的内存区域的地址范围，address 指定要比较的另一内存区域的首地址。

图 9-15 给出了 C 命令的用法举例，第 1 个 C 命令比较的两个区域完全相同，所以什么也没显示；第 2 个 C 命令比较的两个区域只有一个单元不相同，则把那两个对应单元的内容显示出来；第 3 个 C 命令比较的两个区域内容完全不相同，则两个区域中每个单元的内容都一一列出。

15. 十六进制运算命令 H（Hex）

H 命令的格式为：H Value1 Value2；其中 Value 为从 0～FFFFh 的任何十六进制数。H 命令的功能是，分别显示两个十六进制数相加的和以及第 1 个数减去第 2 个数的差，以补码形式显示，如图 9-16 所示。

```
17E8:012B 0000          ADD     [BX+SI],AL
17E8:012D 0000          ADD     [BX+SI],AL
-U 10F 121
17E8:010F B409          MOV     AH,09
17E8:0111 BA0001        MOV     DX,0100
17E8:0114 CD21          INT     21
17E8:0116 CD20          INT     20
17E8:0118 B409          MOV     AH,09
17E8:011A BA0001        MOV     DX,0100
17E8:011D CD21          INT     21
17E8:011F CD20          INT     20
17E8:0121 0000          ADD     [BX+SI],AL
-C 10F 117 118
-C 10F 118 118
17E8:0118  B4  00  17E8:0121
-C 118 120 127
17E8:0118  B4  00  17E8:0127
17E8:0119  09  00  17E8:0128
17E8:011A  BA  00  17E8:0129
17E8:011C  01  00  17E8:012B
17E8:011D  CD  00  17E8:012C
17E8:011E  21  00  17E8:012D
17E8:011F  CD  00  17E8:012E
17E8:0120  20  00  17E8:012F
-
```

图 9-15　C 命令的用法

```
-H 3 6
0009  FFFD
-H 6 3
0009  0003
-
```

图 9-16　H 命令的用法

16. 输入命令 I（Input）

I 命令的功能是从指定端口读入数据并按十六进制数显示。I 命令的格式为：I　port。其中，port 为要读取数据的端口地址。

17. 输出命令 O（Output）

O 命令的功能是将一字节数据送指定端口。O 命令的格式为：O port byte。其中，port 为要输出数据的端口地址，byte 为要输出的那一字节数据。

18. 搜索命令 S（Search）

S 命令的功能是在指定的地址范围内查找给定的一字节数据或一个字符串。S 命令的格式为：S range list。其中 range 指定要搜索的内存范围，list 中给出要搜索的字节值或字符串，字符串应包括在引号中。找到后会显示找到指定数的内存单元的地址。

9.1.4．用 DEBUG 程序调试.EXE 文件和.COM 文件

DEBUG 可调入处理的可执行文件有两种，一种是.EXE 文件，另一种是.COM 文件。调入的方法也有两种，一种是在 DEBUG 后加可执行文件的名字，在启动 DEBUG 时即调入该可执行文件文件；另一种是在启动 DEBUG 后，在 DEBUG 运行中，用 DEBUG 的 N 命令和 L 命令将这些文件调入。不管是哪种调入方法，在装入这些可执行文件后，DEBUG 对寄存器和标志位的初始化情况如下。

装入.COM 文件后，BX 和 CX 中包含被装入文件大小的字节数，高 16 位在 BX 中，低 16 位在 CX 中。其他寄存器的情况与不带文件名用 DEBUG 直接启动时的情况一样，如图 9-1 所示。即所有段寄存器的值都相等，指向当前可用的主存段；除 SP 之外的通用寄存器都设置为 0，SP 指向栈顶；IP=0100H；标志位中除 IF 标志位为 1 表示允许中断外，其他都为 0。

装入.EXE 文件后，BX 和 CX 中包含被装入程序大小的字节数（而不是文件的大小），高 16 位在 BX 中，低 16 位在 CX 中。另外，DEBUG 必须进行重新定位， 段寄存器 CS 和 SS 、SP 置成程序中规定的值，DS 和 ES 置为程序段前缀的段地址；除 SP 之外的通用寄存器都设置为 0，IP=0000H；标志位中除 IF 标志位为 1 表示允许中断外，其它都为 0。

下面结合一个例子来说明 DEBUG 下用.COM 文件和.EXE 文件调试运行程序的情况。

【例 1.2】 下面这段程序实现的功能是在屏幕上显示“HELLO WORLD!”。

```
DATA        SEGMENT
STRING      DB    'HELLO WORLD!', 0DH, 0AH, '$'
DATA        ENDS
CODE        SEGMENT
            ASSUME CS:CODE, DS:DATA
BEGIN:      MOV         AX, DATA
            MOV         DS, AX
            MOV         AH, 09H
            LEA         DX, STRING
            INT         21H
            MOV         AX, 4CH
            INT         21H
CODE        ENDS
            END         BEGIN
```

上面这段程序在清华大学的 TPC-USB 微机接口实验系统集成开发环境下编辑、汇编、链接后生成了一个可执行文件：F22.EXE。如图 9-17 所示，F22.EXE 文件的大小为 544 个字节，而在 DEBUG 下装入该文件后，BX 和 CX 中显示的值是 20H，即程序的实际长度 20H。用 U 命令反汇编后检查，发现程序代码段的长度为 10H 个字节。再用 D 命令检查 F22.EXE 应用程序的数据段的内容，其变量'HELLO WORLD!'，0DH，0AH，'$'所占的字节数也是 10H。这样，该程序合起来总共占了 20H 个字节。

```
C:\Users\ZHOUJI~1\ZJY>DIR
 Volume in drive C is S3A6642D006
 Volume Serial Number is 6CBB-67E8

 Directory of C:\Users\ZHOUJI~1\ZJY

2010/08/22  19:22    <DIR>          .
2010/08/22  19:22    <DIR>          ..
2010/08/18  10:58                 5 F1.COM
2010/08/20  10:54                 5 F2.COM
2010/08/22  19:21               544 F22.EXE
2010/08/20  10:54                 5 F3.EXE
               4 File(s)            559 bytes
               2 Dir(s)   3,658,616,832 bytes free

C:\Users\ZHOUJI~1\ZJY>DEBUG
-N F22.EXE
-L
-R
AX=0000  BX=0000  CX=0020  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=17E8  ES=17E8  SS=17F8  CS=17F9  IP=0000   NV UP EI PL NZ NA PO NC
17F9:0000 B8F817        MOV     AX,17F8
-U
17F9:0000 B8F817        MOV     AX,17F8
17F9:0003 8ED8          MOV     DS,AX
17F9:0005 B409          MOV     AH,09
17F9:0007 BA0000        MOV     DX,0000
17F9:000A CD21          INT     21
17F9:000C B44C          MOV     AH,4C
17F9:000E CD21          INT     21
17F9:0010 0000          ADD     [BX+SI],AL
-D 17F8:0
17F8:0000  48 45 4C 4C 4F 20 57 4F-52 4C 44 21 0D 0A 24 00   HELLO WORLD!..$.
17F8:0010  B8 F8 17 8E D8 B4 09 BA-00 00 CD 21 B4 4C CD 21   ...........!.L.!
-G
HELLO WORLD!

Program terminated normally
-
```

图 9-17　用 U 命令检查程序、用 D 命令检查数据段、用 G 命令运行

由上面分析可知，装入.EXE 文件后，BX 和 CX 中包含被装入的.EXE 程序大小的字节数（而

不是文件的大小），高 16 位在 BX 中，低 16 位在 CX 中。

DEBUG 将程序装入内存后必须进行重新定位，段寄存器 CS 置成当前程序代码段在内存中的段值，IP=0000H，这样 CS:IP 指向所装入程序的第一条可执行指令。SS：SP 指向所装入程序的堆栈段的栈顶单元。而 DS 和 ES 置为程序段前缀的段地址，也就是说 DS 没有指向所装入程序的数据段的段值，应用程序的数据段的段值需在程序中设定，对应汇编语言源程序中的语句如下：

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

由图 9-17 中可以看到，装入 DEBUG 后，与上面两条指令对应的指令为：

```
MOV   AX,17F8
MOV   DS,AX
```

对比可知，F22.EXE 应用程序的数据段（DATA）的段值为 17F8（H）。因此，显示应用程序数据段的内容的命令为：-D 17F8:0。

图 9-17 中，用 G 命令执行程序后，在屏幕上显示'HELLO WORLD!'，并正常退出。

例 1.2 的程序在 DEBUG 下直接输入、汇编并运行的情况如图 9-18 所示。在 DEBUG 的 A 命令下，可以用 DB、DW 伪指令，但不能用标识符。因此例 1.2 中的指令 LEA DX, STRING，改为 MOV DX,100(H)。另外，在 DEBUG 下输入的程序实际上是.COM 文件格式，程序的最后只能以 INT 20（H）返回 DOS 操作系统。

该程序的第 1 条可执行指令在 17D7:010F(H)单元，命令-G=10F 是从偏移量为 10F(H)开始执行的意思。G 命令中没有指明段值，即采用当前段寄存器的值。DEBUG 后面不带文件名的启动以及 DEBUG 后面带.COM 文件名的启动，段寄存器的值就是当前程序的段值。如果 G 命令后面不带地址会从 100H 开始执行，而 100H 存放的是字符串，并不是指令。如果从 100H 执行，对应的指令如图 9-19 所示，并不是我们所要的操作。图 9-19 是图 9-18 中程序 F21.COM 的反汇编，逻辑地址后面那一列是机器代码。比较两个图可以看到，在图 9-18 中用 DB 伪指令定义的那些 ASCII 码存在了 100H～10EH 单元中，最右边的是这些数当作指令的机器码时所对应的汇编指令。

```
C:\Users\zhou jieying\ZJY>DEBUG
-A
17D7:0100 DB 'HELLO WORLD!',0D,0A,'$'
17D7:010F MOV AH,9
17D7:0111 MOV DX,100
17D7:0114 INT 21
17D7:0116 INT 20
17D7:0118
-N F21.COM
-R CX
CX 0000
:18
-W
Writing 00018 bytes
-G=10F
HELLO WORLD!

Program terminated normally
```

图 9-18　DEBUG 下直接输入、汇编并运行程序

```
C:\Users\ZHOUJI~1\ZJY>DEBUG F21.COM
-U
17E8:0100 48              DEC     AX
17E8:0101 45              INC     BP
17E8:0102 4C              DEC     SP
17E8:0103 4C              DEC     SP
17E8:0104 4F              DEC     DI
17E8:0105 20574F          AND     [BX+4F],DL
17E8:0108 52              PUSH    DX
17E8:0109 4C              DEC     SP
17E8:010A 44              INC     SP
17E8:010B 210D            AND     [DI],CX
17E8:010D 0A24            OR      AH,[SI]
17E8:010F B409            MOV     AH,09
17E8:0111 BA0001          MOV     DX,0100
17E8:0114 CD21            INT     21
17E8:0116 CD20            INT     20
17E8:0118 0000            ADD     [BX+SI],AL
```

图 9-19　F21.COM 文件的反汇编

9.2 汇编语言源程序的实验方法

9.2.1 概述

DEBUG 下不能识别标识符，完整的汇编语言源程序不能直接在 DEBUG 下运行。从输入到运行汇编语言源程序的主要过程包括：通过编辑软件编辑扩展名为.ASM 的汇编语言源程序文件；通过汇编软件将汇编语言的源程序翻译成机器代码，生成.OBJ 的目标文件；使用连接程序将.OBJ 文件转换成可执行文件（扩展名为 EXE），然后运行可执行文件即可检验程序的效果；如果不正确可以在 DEBUG 软件下或其它调试环境下对程序进行调试、修改，直到程序正确为止。

实现汇编语言源程序的输入、汇编、调试运行的软件有两大类，一类是 Windows 环境下运行的高级“汇编器”，另一类是基于 DOS 命令行的汇编软件。

Windows 环境下运行的高级“汇编器”是集编辑、编译（汇编）、调试为一体的集成开发环境，如 Borland 公司的 TASM 系列编译器、Microsoft 公司推出的汇编开发环境以及一些小公司推出的或者免费的汇编软件包等。

如清华大学科教仪器厂开发的 TPC-USB 实验系统提供了接口实验箱及配套的 TPC-USB 集成软件开发环境，在 TPC-USB 实验系统上做接口实验时必须要用 TPC-USB 集成软件开发环境调试汇编程序并控制接口工作。PC 机没有与实验箱连接时也可以安装、运行该实验系统的 TPC-USB 软件开发环境，并用于调试运行汇编语言源程序，运行时出现提示说“没有连接设备”等，选择“continue and don’t ask again”就可以运行了，汇编实验程序的编辑、编译、连接、调试、运行和修改的全过程都可在这个 Windows 下的集成环境中完成。

Windows 环境下运行的高级汇编软件拥有可视化的开发界面，使开发人员不必再使用 DOS 命令进行汇编的开发，编译速度快，支持 80x86 汇编，是 Windows 下开发汇编程序的利器，不过安装这一软件需要对 PC 机做些设置，在上课的教室、图书馆等公共电脑上安装时还需要有安装权限才能进行安装，占用计算机的资源较多。这一类软件安装后很容易使用，在此不多介绍。

基于 DOS 命令行的汇编软件，使用者需要使用简单的 DOS 命令，好处是不需要安装可以直接使用，这样在公共教室、图书馆等临时使用的计算机上都可以使用，占用计算机资源不多，比较方便。如 Microsoft 公司推出的 DOS 环境下的 MASM 宏汇编软件 MASM5.0/6.0/6.15 等，在 Windows 的 DOS 命令提示符下也可以直接使用，不需要安装，使用者只需要掌握简单的 DOS 命令即可使用，是目前广泛使用的汇编语言程序的输入、汇编、链接软件，该软件经过了很多次的升级，有很多个版本。相对而言，MASM 6.15 比 MASM5.0 容易使用。下面以广泛使用的 MASM 5.x/6.15 汇编软件为例来介绍汇编语言源程序的输入、汇编、链接及调试运行方法。

9.2.2　MASM5.x/6.15 汇编软件介绍

大家在网上可以下载免费的 MASM5.0 或 MASM6.15(官方解压版)软件，将其放在某个文件夹中解压后即可使用。最好将其解压在某个磁盘的根目录下，如图 9-20 和图 9-21 所示，这样要调用所编写的汇编语言源程序时，可以不用指定那么长的路径。

```
H:\masm5.0>dir
 Volume in drive H is ???
 Volume Serial Number is 6CE8-E9D1

 Directory of H:\masm5.0

2012/01/03  09:07    <DIR>          .
2012/01/03  09:07    <DIR>          ..
1987/07/31  00:00            15,830 CREF.EXE
1987/07/31  00:00             9,499 ERROUT.EXE
1987/07/31  00:00            12,149 EXEMOD.EXE
1987/10/15  05:00            14,803 EXEPACK.EXE
1987/07/31  00:00            32,150 LIB.EXE
1987/07/31  00:00            39,100 LINK.EXE
1987/07/31  00:00            24,199 MAKE.EXE
2002/10/12  11:27               967 MAKE.PIF
1987/07/31  00:00            65,557 MASM.EXE
1987/07/31  00:00            10,601 SETENV.EXE
              10 File(s)        224,855 bytes
               2 Dir(s)  32,769,331,200 bytes free
```

图 9-20　MASM5.0 解压后的文件夹的内容

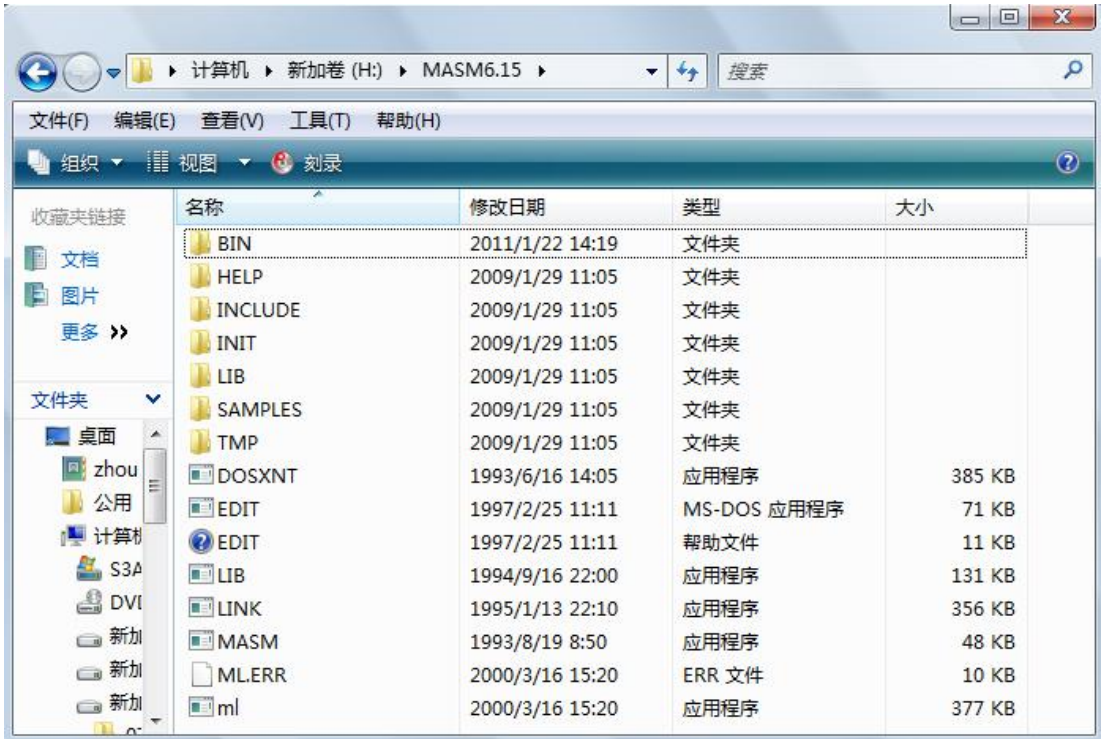


图 9-21　解压后的 MASM6.15(官方解压版)软件

如图 9-20 所示，在 MASM5.0 文件夹中有 MASM、LINK 等几个应用程序。如图 9-21 所示，在 MASM6.15 文件夹中有 EDIT、MASM、LINK 以及 ML 等几个应用程序。MASM6.15 中 EDIT 编辑程序可用于编辑汇编语言源程序。DOS 本身也自带 EDIT 编辑软件，在 MASM5.0 中也可以直接调用。MASM5.0 和 MASM6.15 中都有 MASM 应用程序，可用于对扩展名为.ASM 的汇编语言源程序进行汇编，也即翻译成二进制代码，生成扩展名为.OBJ 的目标文件，两者的用法稍有区别。MASM5.0 和 MASM6.15 中都有 LINK 应用程序，可用于将一个或多个目标文件和库文件中的相关模块链接合成一个扩展名为.EXE 的可执行文件。MASM6.15 中比 MASM5.0 多了一个 ML 应用程序，该程序是 MASM 和 LINK 两者功能相的结合的应用程序，即对.ASM 的源文件先汇编再链接，用一个 ML 命令即可形成可执行文件。

9.2.3 汇编语言源程序的编辑

MASM6.15 中带有一个 EDIT 全屏幕文本编辑器应用程序，双击该应用程序即可出现全屏幕文本编辑界面，可以输入汇编语言源程序。注意，一般要求汇编语言源程序文件的扩展名必须为.ASM。除了双击 EDIT 应用程序进入编辑界面外，也可以在 DOS 命令提示符下输入 EDIT 命令进入编辑界面，如图 9-22。

```
Microsoft Windows [版本 6.0.6002]
版权所有 (C) 2006 Microsoft Corporation。保留所有权利。

C:\Users\zhou jieying>H:

H:\>CD MASM6.15

H:\MASM6.15>EDIT_
```

图 9-22 用 DOS 命令行的方法进入 EDIT 文本编辑环境

图 9-22 中，DOS 提示符下输入“H:”是由当前盘（C 盘）换到 H 盘的意思，“CD MASM6.15”是转到当前盘的 MASM6.15 文件夹的意思，“CD”是“CHANGE DIRECTORY”的缩写，是改变目录的 DOS 命令。WINDOWS 中的“文件夹”，在 DOS 中称为“目录”。常用的 DOS 命令还有“DIR”、“REN”、“COPY” 等。 “DIR”是“DIRECTORY” 的缩写，用于当前目录下所包含的文件、目录；“REN” ,是“RENAME” 的缩写，用于更改文件名；“COPY”命令用于将某个文件 COPY 为另一个名字的文件。建议大家查阅一下相关的常用 DOS 命令。

EDIT 命令后面也可以带文件名，例如打入命令：H:\>EDIT　F1.ASM ,则进入编辑 F1.ASM 文件的界面，如图 9-23 所示。 其中 F1 为文件名，.ASM 为扩展名。

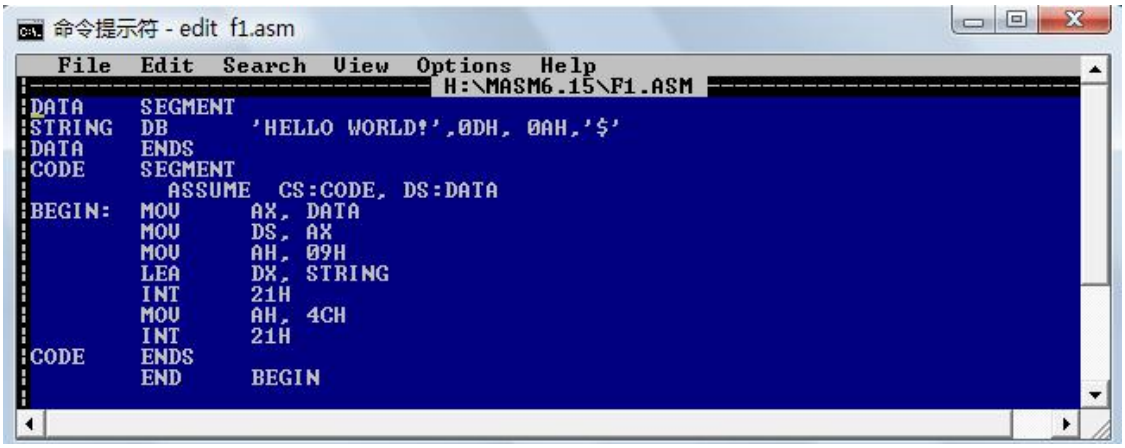


图 9-23　EDIT　F1.ASM 的显示界面

DOS 本身也自带 EDIT 全屏幕文本编辑器，在 DOS 提示符下直接输入 EDIT 回车即可进入图 9-23 所示的 EDIT 编辑界面。在使用 MASM5.0 汇编软件时就可调用 DOS 下的 EDIT 应用程序编辑输入汇编语言源程序。

编辑汇编语言源程序除了用 EDIT 编辑软件外，还可以用其它的文本编辑器，如记事本、写字板以及 word 等。需要注意的是，汇编语言源程序文件的扩展名一般要求为.ASM。在记事本中可以将程序保存为扩展名为.ASM 的文件，在其它编辑软件中若不能存为扩展名为.ASM 的文件的话，保存好文件后再改文件的扩展名也行。

在 DOS 提示符下输入 REN 命令可以更改文件名，或者用 COPY 命令也行，这两种方法

的用法如下所示：

C:\>REN F1.TXT F1.ASM　；　　C:\>COPY F1.TXT F1.ASM　　。

还有一种编辑输入汇编语言源程序方法是在 word 等编辑软件下编辑好后，再粘贴到 EDIT 编辑器中修改并保存为.ASM 文件。

不管用哪种方法输入汇编语言源程序，需要注意事项如下：

（1）汇编语言源程序的扩展名一般要求为.ASM;

（2）汇编语言源程序中除了注释外不能出现中文的符号，标点符号也要用英文的符号，如逗号、引号、分号等；

（3）汇编程序不区分大小写，多个空格和 1 个空格是一样的；

（4）取文件名以及目录名（文件夹名）最好用英文字母，这样输入简单，而且 DOS 下不一定输得了中文。

9.2.4　MASM5.0 中对源程序的汇编

汇编语言源程序采用助记符表达指令操作码，采用标识符表示指令操作数，输入计算机后不能直接被计算机识别和执行，必须翻译为二进制机器代码才能在计算机上运行使用。这个翻译的过程被称为汇编。

汇编过程实际上就是把扩展名为.ASM 的源程序转换成用二进制代码表示的扩展名为.OBJ 的目标程序的过程。在汇编过程中，汇编程序对源文件进行二次扫描。第一次扫描检查源程序中是否有语法错误并建立该源程序使用的符号名字表。如果有语法错误，则汇编过程结束并指出源程序中有错误的行号及错误信息，用户修改源程序中的错误后重新汇编。如果源程序没有错误，汇编程序再对源程序进行第一次扫描，建立该源程序使用的符号名字表，之后进行第二次扫描。在第二次扫描中将汇编指令翻译成对应的机器码，并得到对应于符号地址的实际地址值，第二次扫描还产生相应的列表文件。

MASM5.0 中实现汇编的应用程序是 MASM.EXE，在 MASM5.0 文件夹的 DOS 提示符下，输入“MASM/HELP”可以得到 MASM 的使用格式，如图 9-23 所示。汇编程序的输入文件就是用户编写的汇编语言源程序，它必须以 ASM 为文件扩展名。汇编程序的输出文件有三个，第一个是目标文件，它以 .OBJ 为扩展名，这是汇编操作所要形成的文件；第二个是列表文件，它以 .LST 为扩展名，列表文件同时给出源程序和机器语言程序，便于调试。第三个是交叉符号表文件，它以 .CRF 为扩展名，该表给出了标识符（段名、过程名、变量名、标号）在源程序中定义的行号和被引用的行号，并在定义行号上加上"#"号。

图 9-24 给出了 MASM 程序的几种用法，可以只输入 MASM 直接运行，屏幕上提示输入相关文件名时再输入；也可以在 MASM 后输入要汇编的文件名及要输出产生的文件名；或者在 MASM 后输入部分文件名，其它在屏幕上提示时再输入。屏幕上出现提示信息时若采用默认用法，直接回车即可；否则输入文件名再回车。屏幕提示信息中“NUL”表示空，即不需要的意思。例如，“.lst”文件和“.crf”文件是为了调试程序方便而产生的，不是必须的，如不需要，在屏幕上出现提示信息[NUL.CRF]或[NUL.LST]：时，打入回车即可；如果需要，则打入文件名和回车。“/option”选项，不用时可以不写。

扩展名为.lst 的列表文件是纯文本文件，用文本编辑器可以打开，也可以在 DOS 命令行中用 TYPE 命令打印出来，如图 9-25 所示。扩展名为 “.crf”的交叉符号表文件是二进制文件，可以使用实用程序 CREF.EXE 将其转换为文本文件，该文本文件的扩展名默认为 REF，并可用文本编辑器打开或在 DOS 中用 TYPE 命令打印出来，如图 9-26 所示。

```
H:\masm5.0>MASM/HELP
Usage: masm /options source(.asm),[out(.obj)],[list(.lst)],[cref(.crf)][;]

/a                Alphabetize segments
/b<number>        Set I/O buffer size, 1-63 (in 1K blocks)
/c                Generate cross-reference
/d                Generate pass 1 listing
/D<sym>[=<val>] Define symbol
/e                Emulate floating point instructions and IEEE format
/I<path>          Search directory for include files
/l[a]             Generate listing, a-list all
/M[lxu]           Preserve case of labels: l-All, x-Globals, u-Uppercase Globals
/n                Suppress symbol tables in listing
/p                Check for pure code
/s                Order segments sequentially
/t                Suppress messages for successful assembly
/v                Display extra source statistics
/w<012>           Set warning level: 0-None, 1-Serious, 2-Advisory
/X                List false conditionals
/z                Display source line for each error message
/Zi               Generate symbolic information for CodeView
/Zd               Generate line-number information
```

图 9-23　MASM5.0 中 MASM 的使用格式

```
H:\masm5.0>masm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Source filename [.ASM]: f1
Object filename [f1.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  50148 + 399132 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

```
H:\masm5.0>masm f1.asm
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Object filename [f1.OBJ]:
Source listing  [NUL.LST]:
Cross-reference [NUL.CRF]:

  50148 + 399132 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

```
H:\masm5.0>masm f1.asm f1.obj f1.lst f1.crf
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.


  49072 + 382192 Bytes symbol space free

      0 Warning Errors
      0 Severe  Errors
```

图 9-24　MASM5.0 中对汇编语言源程序的汇编

```
H:\masm5.0>type f1.lst
Microsoft (R) Macro Assembler Version 5.00                  1/18/12 09:44:58
                                                             Page     1-1


       1 0000                                DATA    SEGMENT
       2 0000  48 45 4C 4C 4F 20 57          STRING  DB      'HELLO WORLD!',0DH, 0AH
                                             ,'$'
       3       4F 52 4C 44 21 0D 0A
       4       24
       5 000F                                DATA    ENDS
       6 0000                                CODE    SEGMENT
       7                                       ASSUME  CS:CODE, DS:DATA

       8 0000  B8 ---- R                     BEGIN:  MOV     AX, DATA
       9 0003  8E D8                                 MOV     DS, AX
      10 0005  B4 09                                 MOV     AH, 09H
      11 0007  8D 16 0000 R                          LEA     DX, STRING
      12 000B  CD 21                                 INT     21H
      13 000D  B4 4C                                 MOV     AH, 4CH
      14 000F  CD 21                                 INT     21H
      15 0011                               CODE    ENDS
      16                                            END     BEGIN
Microsoft (R) Macro Assembler Version 5.00                  1/18/12 09:44:58
                                                            Symbols-1


Segments and Groups:

                N a m e                 Length   Align   Combine Class

CODE . . . . . . . . . . . . . .        0011     PARA    NONE
DATA . . . . . . . . . . . . . .        000F     PARA    NONE

Symbols:

                N a m e                 Type     Value   Attr

BEGIN  . . . . . . . . . . . . .        L NEAR   0000    CODE

STRING . . . . . . . . . . . . .        L BYTE   0000    DATA

@FILENAME  . . . . . . . . . . .        TEXT  f1


     14 Source  Lines
     14 Total   Lines
      6 Symbols

  49074 + 382190 Bytes symbol space free

      0 Warning Errors
```

图 9-25　对 F1.ASM 文件汇编后所形成的 F1.LST 列表文件

```
H:\masm5.0>CREF F1.CRF
Microsoft (R) Cross-Reference Utility  Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987.  All rights reserved.

Listing [F1.REF]:

4 Symbols

H:\masm5.0>TYPE F1.REF
♀
Microsoft Cross-Reference  Version 5.00       Wed Jan 18 09:51:49 2012

  Symbol Cross-Reference            (# definition, + modification)      Cref-1

BEGIN. . . . . . . . . . . . . .    8#    16

CODE . . . . . . . . . . . . . .    6#     7     15

DATA . . . . . . . . . . . . . .    1#     5      7      8

STRING . . . . . . . . . . . . .    2#    11

 4 Symbols
```

图 9-26 对 F1.ASM 文件汇编后所形成的 F1.REF 交叉符号表文件

汇编过程结束时，会给出源程序中的警告性错误[Warning Errors]和严重错误[Severs Errors]，前者指一般性错误，后者指语法性错误，当存在这两类错误时，屏幕上除给出错误个数以及错误所在的行号外，还给出错误信息代号，程序员可以通过查找手册弄清错误的性质。要指出的是汇编过程只能指出源程序中语法错误，并不能指出算法错误和其它错误。

9.2.5　MAM5.0 中对目标文件的链接

汇编过程根据源程序产生出二进制的目标文件（OBJ 文件），其中存放的是与源程序一一对应的机器码。不过，.OBJ 文件用的是浮动地址，文件中各段是独立的，它们的起始地址均被设为 0，各段没有形成一个整体，不能直接上机执行。必须使用链接程序（LINK）将一个或多个目标文件链接起来，形成一个整体，生成扩展名为.EXE 的可执行文件，并可选择生成扩展名为. MAP 的映像文件。这些目标文件可以是由用户编写的汇编语言源程序经过汇编后生成的.OBJ 文件，也可以是某个程序库中存在的扩展名为.LIB 的库文件。链接过程无错误时生成 EXE 文件，否则，不能生成 EXE 文件。

```
H:\masm5.0>LINK F1.OBJ

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Run File [F1.EXE]:
List File [NUL.MAP]:
Libraries [.LIB]:
LINK : warning L4021: no stack segment
```

```
H:\masm5.0>link

Microsoft (R) Overlay Linker  Version 3.60
Copyright (C) Microsoft Corp 1983-1987.  All rights reserved.

Object Modules [.OBJ]: F1
Run File [F1.EXE]:
List File [NUL.MAP]: F1
Libraries [.LIB]:
LINK : warning L4021: no stack segment

H:\masm5.0>F1
HELLO WORLD!
```

图 9-27 LINK 程序的用法

图 9-27 给出了 LINK 程序链接 F1.OBJ 文件生成 F1.EXE 的两种常用方法，并给出了运行 F1.EXE 文件时在屏幕上显示的内容，即“HELLO WORLD!”。图 9-27 LINK 过程的提示信息中给出了“无堆栈段”的警告性错误，这并不影响程序的执行。当然，若源程序中设置了堆栈段，则无此提示信息。

链接时若需要用到某个库文件（.LIB），在屏幕上出现提示信息[.LIB]：时，打入文件名和回车;若不需要，打入回车即可。生成映像文件（.MAP）文件不是必须的，如果不需要的话，在屏幕上出现提示信息[NUL.MAP]：时，打入回车即可；如果需要，则打入文件名和回车。

图 9-28 给出了对应于 F1.ASM 的映像文件 F1.MAP 的内容，其中显示了程序中用到的各个段的起止地址和长度。

```
H:\masm5.0>TYPE F1.MAP
LINK : warning L4021: no stack segment

 Start  Stop   Length Name                   Class
 00000H 0000EH 0000FH DATA
 00010H 00020H 00011H CODE
```

图 9-28　F1.MAP 文件的内容

连接程序 LINK 把 OBJ 文件（一个或多个 OBJ 文件）中的各段连接成一个整体形成.EXE 文件，为程序中所有段分配存储空间，确定各段相对于第一个段的起始地址的地址偏移量。汇编程序 MASM 把各段的起始地址均设为 0，LINK 程序把这些段连接为一个整体 EXE 文件时需确定各段相对于第一个段的起始地址的地址偏移量，如确定"MOV AX，DATA"指令中 DATA 代表的值。链接程序还要确定汇编程序 MASM 不能确定的偏移地址，包括浮动地址（如 count 变量的地址）以及外部符号的地址。

在模块化程序设计中，可以把一个大型程序的源程序存放为几个源程序文件，每个文件称为一个源程序"模块"。每个源程序模块被汇编程序 MASM 翻译为独立的 OBJ 文件（OBJ 模块），连接程序 LINK 把这些 OBJ 模块连接为一个 EXE 文件。在某个源程序模块中， 可以引用另一个源程序模块中定义的符号（标号、过程名、变量） 即外部符号"。汇编程序 MASM 在翻译某源程序模块时，不能确定外部符号的地址，外部符号的地址由连接程序 LINK 来确定。

库文件是一种特殊的 OBJ 文件。在程序设计中，可以把一些常用的程序段设计为子程序（过程），用汇编程序 MASM 把它们翻译为 OBJ 文件，然后由库管理程序 LIB.EXE 生成库文件 LIB。以后需使用某子程序时，直接调用，不必写出这些子程序的代码。由连接程序 LINK 把这些代码加入可执行文件 EXE 中。

有了 .EXE 文件后，就可以执行程序了，此时，只要打入文件名即可。图 9-27 中显示了 F1.EXE 文件的运行结果。实际上，大部分程序必须经过调试阶段才能纠正程序设计中的错误，从而得到正确的结果，调试时可用调试程序（DEBUG 程序）发现错误 ，再经过编辑、汇编、链接纠正错误。关于 DEBUG 程序中的各种命令，可参阅上一节内容。

9.2.6　MASM6.15 中的汇编与链接

```
H:\MASM6.15>ml/help
Microsoft (R) Macro Assembler Version 6.15.8803
Copyright (C) Microsoft Corp 1981-2000.  All rights reserved.


        ML [ /options ] filelist [ /link linkoptions ]

/AT Enable tiny model (.COM file)          /omf generate OMF format object file
/Bl<linker> Use alternate linker           /Sa Maximize source listing
/c Assemble without linking                /Sc Generate timings in listing
/Cp Preserve case of user identifiers      /Sf Generate first pass listing
/Cu Map all identifiers to upper case      /Sl<width> Set line width
/Cx Preserve case in publics, externs      /Sn Suppress symbol-table listing
/coff generate COFF format object file     /Sp<length> Set page length
/D<name>[=text] Define text macro          /Ss<string> Set subtitle
/EP Output preprocessed listing to stdout /St<string> Set title
/F <hex> Set stack size (bytes)            /Sx List false conditionals
/Fe<file> Name executable                  /Ta<file> Assemble non-.ASM file
/Fl[file] Generate listing                 /w Same as /W0 /WX
/Fm[file] Generate map                     /WX Treat warnings as errors
/Fo<file> Name object file                 /W<number> Set warning level
/FPi Generate 80x87 emulator encoding      /X Ignore INCLUDE environment path
/Fr[file] Generate limited browser info    /Zd Add line number debug info
/FR[file] Generate full browser info       /Zf Make all symbols public
/G<c|d|z> Use Pascal, C, or Stdcall calls /Zi Add symbolic debug info
/H<number> Set max external name length    /Zm Enable MASM 5.10 compatibility
/I<name> Add include path                  /Zp[n] Set structure alignment
/link <linker options and libraries>       /Zs Perform syntax check only
/nologo Suppress copyright message
```

图 9-29　MASM6.15 中 ML 程序的使用格式

MASM6.15 中有一个宏汇编（MACRO）与链接(LINK)合二为一的应用程序 ML.EXE，我们可用这个程序对汇编源程序进行汇编生成.OBJ 文件，之后链接生成.EXE 可执行文件。在 DOS 提示符下输入“ML/HELP”,则屏幕上会显示 ML 程序的用法，如图 9-29 所示。ML.EXE 使用格式如下：

ML [/options] filelist [/link linkoptions]

Filelist 是汇编语言源程序的文件名，必须是扩展名为.ASM 的文件，这一项是必须写的。其它方括号内的都是可选项（也称为参数）。Filelist 前面的是汇编可选项，Filelist 后面的是链接可选项，各参数选项之间用空格隔开。参数选项有很多，在此列出常用的参数选项如下，注意参数是大小写敏感的：

- /c---只对源程序进行汇编（编译），不进行自动连接。注意是小写字母 c（compile），高级语言里面将这个翻译的过程称为编译，在汇编语言中称为汇编。
- /Fe filename：指定生成的可执行文件名（.EXE 文件），而不是采用缺省名。
- /Fo filename：指定生成的目标文件名（.OBJ 文件），而不是采用缺省名。
- /Fl [filename]：生成.lst 列表文件，若没给出文件名，则与源文件同名。
- /Fm [filename]：生成连接映像文件（扩展名为 MAP），若没给出文件名，则与源文件同名。
- /I pathname: 指定 Include 文件的路径。
- /Zm ：与 MASM5.10 兼容

ML.EXE 程序的使用是可以带参数的，不带参数则采用默认用法（也称为缺省用法），生成与原文件同名的.OBJ 文件和.EXE 文件。如图 9-30 所示，我们不带参数运行 ML 程序对 F1.ASM 文

件进行了汇编生成了目标文件 F1.OBJ，并对 F1.OBJ 文件进行了连接，生成了可执行文件 F1.EXE（可以用 DOS 命令 DIR 检查）。之后，在 DOS 提示符下输入 F1(可执行文件 F1.EXE 的文件名) 即执行该程序，在屏幕上显示了该程序的运行结果。

```
H:\MASM6.15>ML F1.ASM
Microsoft (R) Macro Assembler Version 6.15.8803
Copyright (C) Microsoft Corp 1981-2000.  All rights reserved.

 Assembling: F1.ASM

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

Object Modules [.obj]: F1.obj
Run File [F1.exe]: "F1.exe"
List File [nul.map]: NUL
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment

H:\MASM6.15>F1
HELLO WORLD!
```

图 9-30　用 ML 程序对 F1.ASM 文件进行汇编及链接生成 F1.OBJ 和 F1.EXE

在 MASM6.15 中，我们也可以先用 MASM 程序对.ASM 的源文件进行汇编生成.OBJ 的目标文件，之后再用 LINK 程序对.OBJ 的目标文件进行链接生成可执行文件。在图 9-31 中，我们先用 COPY 命令将 F1.ASM 文件拷贝为另一个文件 F2.ASM 来进行实验。然后用 MASM 程序对 F2.ASM 的源文件进行汇编生成 F2.OBJ 的目标文件。在图 9-32 中，我们再用 LINK 程序对 F2.OBJ 文件进行了链接，生成了可执行文件。运行 F2 可执行文件得到了程序的运行结果。MASM6.15 中先执行 MASM 程序进行汇编再运行 LINK 程序进行链接的方法与 MASM5.0 中类似，只是 MASM6.15 中汇编时是激活了带参数的 ML 程序来实现汇编的，其它类似，在此不多介绍。

```
H:\MASM6.15>COPY F1.ASM F2.ASM
        1 file(s) copied.

H:\MASM6.15>MASM F2.ASM
Microsoft (R) MASM Compatibility Driver
Copyright (C) Microsoft Corp 1993.  All rights reserved.

 Invoking: ML.EXE /I. /Zm /c /Ta F2.ASM

Microsoft (R) Macro Assembler Version 6.15.8803
Copyright (C) Microsoft Corp 1981-2000.  All rights reserved.

 Assembling: F2.ASM
```

图 9-31　用 MASM 程序对 F2.ASM 文件进行汇编生成目标文件 F2.OBJ

```
H:\MASM6.15>LINK F2.OBJ

Microsoft (R) Segmented Executable Linker  Version 5.60.339 Dec  5 1994
Copyright (C) Microsoft Corp 1984-1993.  All rights reserved.

Run File [F2.exe]:
List File [nul.map]:
Libraries [.lib]:
Definitions File [nul.def]:
LINK : warning L4021: no stack segment

H:\MASM6.15>F2
HELLO WORLD!
```

图 9-32　用 LINK 程序对 F2.OBJ 文件进行链接生成可执行文件 F2.EXE

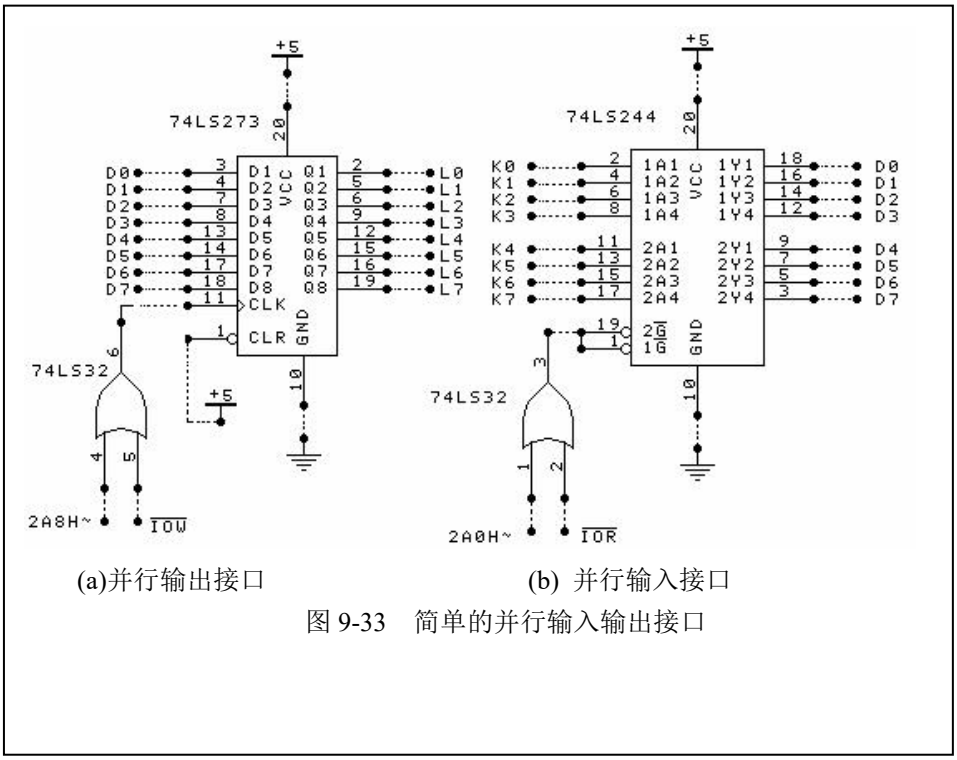
9.3 微机接口提高性实验

TPC-USB 通用微机接口实验系统是目前国内高校上微机原理、汇编语言、微机接口技术课实验教学用得比较多的实验系统。TPC-USB 实验系统由一块 USB 总线接口模块、一个扩展实验台及软件集成实验环境组成。USB 总线接口模块通过 USB 总线电缆与 PC 机相连，模块与实验台之间由一条 50 芯扁平电缆连接。实验台接口集成电路包括：可编程定时器/计数器（8253）、可编程并行接口（8255）、数/模转换器(DAC0832)、模/数转换器(ADC0809)等。外围电路包括：逻辑电平开关、LED 显示、七段数码管显示、8×8 双色发光二极管点阵及驱动电路、直流电机步进电机及驱动电路、电机测速用光藕电路、数字测温传感器及接口电路、继电器及驱动电路、喇叭及驱动电路。8279 键盘显示控制电路。与实验箱配套的实验指导书上已经设计了一些实验给学生们练习，并可以在 TPC-USB 集成软件开发环境下安装实验指导和演示程序。同学们通过这些实验可以得到基本的锻炼，在此，我们再结合该实验系统，设计一些提高性实验和综合性实验给同学们练习。需要提醒大家的是,在本节给出的图中,大家只需要连接虚线所示的线路即可,实线是实验箱内部已经连接好的。

实验一、简单的并行输入输出接口实验

1．ASCII 码转换为 BCD 码并输出显示

将从键盘读入的一位十进制数的 ASCII 码，转换为未组合 BCD 码后，从图 9-33（a）所示的 74LS273 接口输出，控制 8 个发光二极管发光，根据发光管的亮灭情况验证其转换的正确性；若输入的不是十进制数的 ASCII 码，则对应 8 个发光二极管全亮，显示内容为“FFH”。



(a)并行输出接口　　(b) 并行输入接口

图 9-33　简单的并行输入输出接口

设计思路：十进制数对应的 ASCII 码为 30H~39H，根据实验要求需要将其转换为 BCD 码，可以看出，只需要屏蔽掉高四位（置 0），即可正确显示。如果 ASCII 码不在 30H~39H 范围之内，则直接输出 FFH。74LS273 并行输出接口的端口地址为 2A8H。

2．检测开关闭合实验

如图 9-33（b）所示，假设逻辑电平开关中每次只有一个开关置为 1，请编写程序检测是 K0~K7 中哪个开关置 1，并将该开关的序号送至屏幕显示。74LS32 并行输入接口的端口地址为 2A0H。

实验二、　可编程定时器 / 计数器（8253）

实验内容：8253 定时器 0 的输出作为定时器 1 的输入，定时器 1 的输出作为定时器 2 的输入，定时器 2 的输出接在一个 LED 上。编程将 8253 定时器 0 设为方式 3，定时器 1 设为方式 2，定时器 2 设为方式 2，并设置好各定时器的计数初值，要求程序运行后可观察到该 LED 在不停闪烁。该 8253 芯片的起始端口地址为 280H，CLK 引脚信号从实验箱上的 1MHz 的频率信号输入。

编程提示：

方式 3 为方波频率发生器，即产生方波信号；方式 2 为分频器，与方式 3 类似。本实验的工作在于对 8253 的初始化，只要初始化正确，尤其是计数初值设置得合适就可达到预期目标。

实验三、可编程并行接口（8255方式0）--LED走马灯花样实验

实验内容：如图9-34连线，利用K7—K0的8位开关，控制LED产生8种走马灯花样。例如，将拨动开关的 1号开关合上时，8个LED彩灯从两端向中间依次点亮；2号开关合上时，彩灯从中间向两端依次点亮等等。

编程提示：

需要定义八个变量，每个变量依次定义八组数据对应LED显示状态，先从C 口读入逻辑开关的状态，然后根据这些状态选择不同的数据从A口输出，从A口输出每个数据后需要延时，这样才能观察到结果。8255控制寄存器端口地址为28BH，A口的地址为288H，C口的地址为28AH。

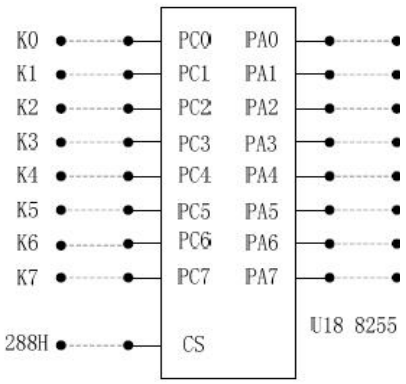


图 9-34 8255 连接图

实验四、8253和8255联合实验--电子音乐

请利用实验箱上的 8253 定时器及 8255 并行口，控制实验箱的扬声器发声，并唱出“Mary Had a Little Lamb”的歌曲。该实验在实验箱上的电路如图 9-35 所示，8255 的起始端口地址为 288H，8253 的起始端口地址为 280H。歌中各音调的频率值及长度参见表 9-2。

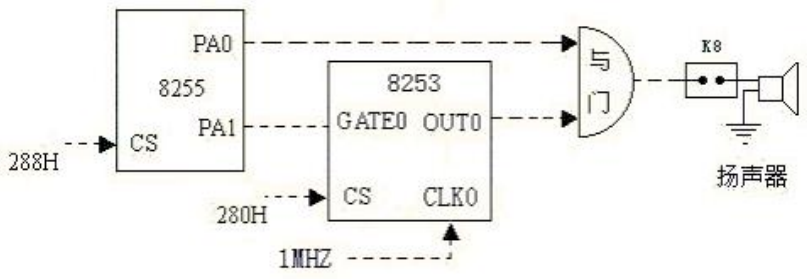


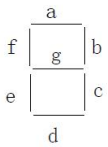
图 9-35 电子音乐实验原理图

表 9-2　　各音调的频率值及长度

歌词	音调	频率（Hz）	长度	歌词	音调	频率（Hz）	长度
Mar	E4	330	1	Mar	E4	330	1
y	D4	294	1	y	D4	294	1
had	C4	262	1	had	C4	262	1
a	D4	294	1	a	D4	294	1
lit	E4	330	1	lit	E4	330	1
tle	E4	330	1	tle	E4	330	1
lamb	E4	330	2	lamb	E4	330	1
lit	D4	294	1	whose	E4	330	1
tle	D4	294	1	fleece	D4	294	1
lamb	D4	294	2	was	D4	294	1
lit	E4	330	1	white	E4	330	1
tle	G4	392	1	as	D4	294	1
lamb	G4	392	2	snow.	C4	262	4

实验五、七段数码管

实验箱上有两个共阴极七段数码管及相关驱动电路，当段码a、b、c、d、e、f、g输入端加高电平时选中的数码管亮，数字0-9对应的七段码如表9-3所示。两个数码管的连接图如图9-36所示，两个七段数码管对外共用a、b、c、d、e、f、g几个控制引脚以及小数点dp（小数点dp为1亮，为0不亮），两个数码管受S1、S0两个控制位控制,Si输入端(i=0或1）为高电平选中，每次只能有一个数码管工作。S1控制左边的管（位1管），S0控制右边的管。



实验时，我们可以选择只让其中一个管显示，称为静态显示，静态显示时显示的内容可以改变，但数码管的控制位Si不变；如果让两个数码管轮流显示，只要交替变换的频率足够高，会感觉到两个数码管同时稳定地显示，这种显示方式称为动态显示。下面我们将分别设计静态显示和动态显示的提高性实验。

表9-3七段数码管的字型代码表

十进制数	0	1	2	3	4	5	6	7	8	9
七段码（H）	3F	06	5B	4F	66	6D	7D	07	7F	6F

1.静态显示实验

实验内容：通过实验箱上简单的并行输入接口74LS244读入逻辑电平开关的值，检测是哪个开关置1了（假设每次只有一个开关置为1），如果检测到有开关置1，则将该开关的序号送七段数码管显示，否则，七段数码管不亮。请设计该实验的硬件连接，并编写程序实现上述功能。

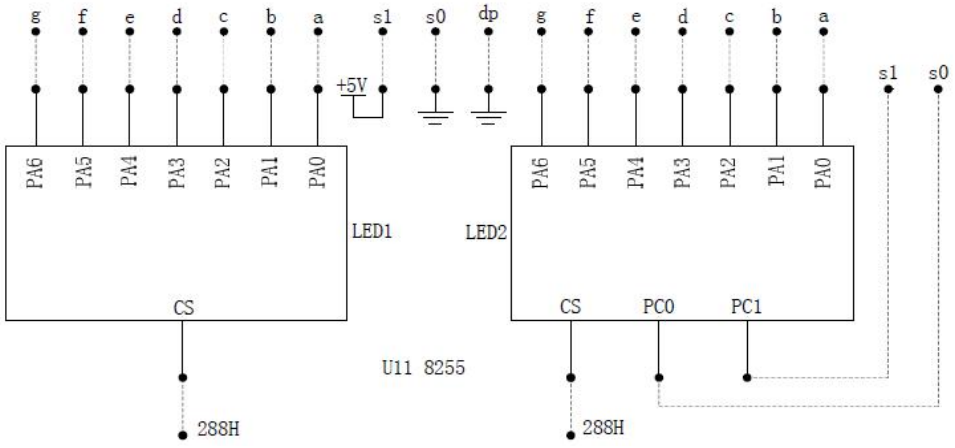
硬件连接提示：

（1） 逻辑电平开关与简单并行输入接口74LS244的连接可以参考图9-33（b），该图中74LS244并行输入接口的端口地址为2A0H。

（2） 七段数码管的连接可参考图9-36（a），将8255的A口PA0～PA6分别与七段数码管的段码驱动输入端a～ g 相连,图中8255的起始端口地址为288H。也可以考虑用图9-33（a）所示的简

单的并行输出接口74LS273，其输出Q1～Q7分别与七段数码管的段码驱动输入端a～ g 相连，图中74LS273的端口地址为2A8H。如果选择左边的七段管亮，则位码驱动输入端S1接+5V(选中)，S0、dp接地，即第1个七段数码管亮，第0个七段数码管及小数点都不亮；如果选择右边的七段管亮的话，则位码驱动输入端S0接+5V(选中)，S1、dp接地。

编程提示：只需将置1开关的序号转换为七段码，将该七段码值送8255的A口输出即可。十进制数对应的七段码表如图9-2所示，可以通过查表转换获得对应十进制数的七段码。



(a)　静态显示　　（b）动态显示

图 9-36　七段数码管的连接

2．动态显示实验

实验内容：连线如图9-36（b）所示，七段数码管段码连接不变，位码驱动输入端S1，S0分别接8255 C口的PC1，PC0。编程从键盘读入两位数字，在两个数码管上显示出来，并在数码管上倒计数显示。当键盘有新的输入时，按新的数字进行倒计数显示。

读键盘并显示编程提示：从键盘读入的数据放入存放要显示的十位和个位数据定义区，判断键盘是否有输入，如有则去读取键盘输入数据取代要显示的数据。

判断键盘是否有键按下参考程序：

```
mov ah,06 ;是否有键按下
mov dl,0ffh
int 21h
jne dushuju ;若有则转dushuju
```

实现倒计时编程提示：

在上面实验基础上，实现这一点比较容易。只需要添加两个变量作为缓存，一个记录键盘输入的值，一个循环减 1，并将减 1 计数后的值通过数码管显示出来，减至 0 时重装键盘输入的初值，这样就能够实现倒计数显示了。

实验六 继电器控制

实验内容：实验电路如图 9-37 所示，继电器开关量输入端 Ik 输入数字“1”时，继电器常开

触点闭合，电路接通，指示灯发亮；Ik 输入“0”时断开，指示灯熄灭。编程使用 8253 定时，让继电器周而复始的闭合 n 秒钟(指示灯灯亮)，断开 n 秒钟(指示灯灯灭)。继电器开合时间 n 秒（5<n<9）可由键盘输入，由 8253 准确定时，并将定时时间显示在七段数码管上（以秒为单位）。要求数码管上能够倒计时显示继电器的开合时间。

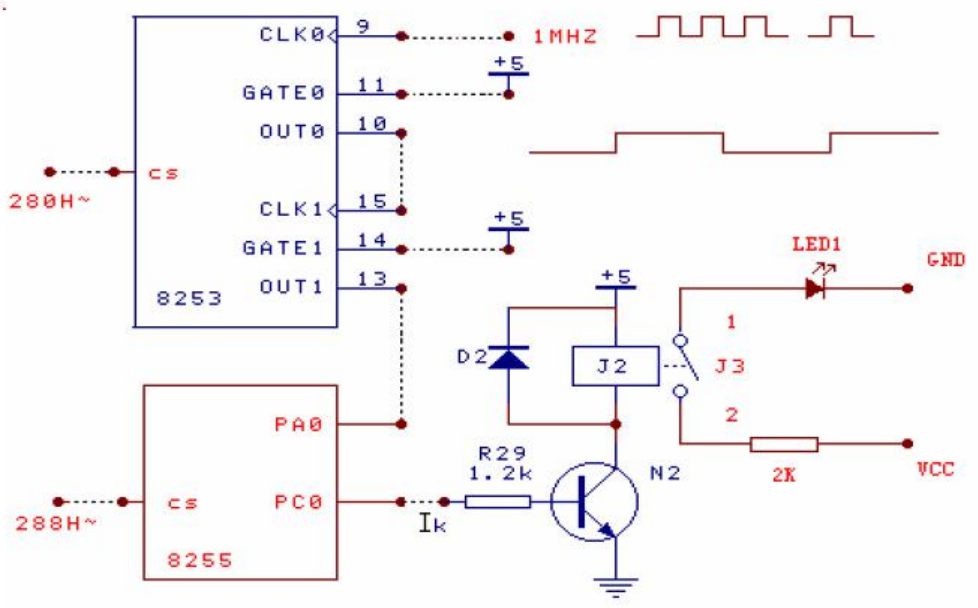


图 9-37 继电器控制实验原理图

编程提示:继电器是一种比较简单而又广泛应用的控制装置。8253定时器的初始时钟为1MHz，经过2级级联后可以满足继电器n秒交替开闭的控制目标。8253后一级定时器采用方式0，则可以在计数值减为0时，OUT端输出由低电平变为高电平，因此只需检测后一级定时器OUT端的电平就可判断是否到定时时间，进而改变继电器的开闭状态。也可以考虑计数器8253 三级级联，第一级对CLK时钟进行分频（如方式3）；第二级计数器对第一级的输出信号（OUT）进行分频，输出以1秒为周期的方波信号（方式3）；最后一级以第二级的输出作为时钟信号，用方式0进行秒级计数，其初值即为需要延时的秒数。同学们也可以考虑其他方法。

这是一个较为综合的实验。程序可以采用模块化的设计思想。首先将实验任务分解，主要包括 5-9s 延时，从键盘读数并送数码管显示，读写 8255、8253。因此可以封装 3 个子程序：延时子程序、从键盘读数并送数码管显示子程序、读写 8255、8253 子程序，然后将这 3 个子程序有机结合就可完成程序的设计。难点在于要不断地查询键盘的信息，进而能够实时改变继电器的开合时间，可以将查询放在延时子程序之中，如果有键按下，就重新更新显示和定时器的值，以便实时调整继电器的开关时间。

倒计时显示部分编程提示：记录键盘输入的值，计数器 8253 以秒级计数，1 秒钟定时到时数码管显示的数减 1；当减至 0 时，重新载入记录到的初值。

可采用 8255 C 口的 7 位控制七段数码管的显示，剩余的 1 位控制继电器工作，或采用简单并行接口控制七段码显示。

实验七 竞赛抢答器

实验要求：由裁判按下按钮产生单脉冲作为中断请求信号，CPU接到中断请求后，显示提示信息进入抢答。在中断服务程序中检测竞赛抢答按钮0～7号，当某个逻辑电平开关置“ 1 ”时，相当于某组抢答按钮按下。在七段数码管上将其组号(0～7)显示出来，并使微机或实验箱播放音乐，同时开始计时，计时50秒后关闭音乐。请设计硬件连接并编程实现上述功能。

硬件连接提示：

（1）中断请求信号的产生：TPC-USB实验板上，有一个中断请求引脚IRQ（插孔），该引脚固定地接到了PC机中8259中断控制器的3号中断IRQ3上，即其中断类型号为0BH。PC机系统已对8259芯片进行初始化，设定中断请求信号为“边沿触发”，普通中断结束方式。实验箱上有一个单脉冲信号发生器，按一下按钮就可输出一个正脉冲，直接将此单脉冲信号发生器的输出连接到实验板上的IRQ引脚即可，当裁判按下按钮即可发出中断请求启动新一轮抢答。

（2）关于检测开关K0-K7哪个置1，并在七段数码管上显示对应序号的电路已在前面介绍了几种,大家可以尝试进行修改：如改为用实验箱上8255的C口输入开关状态，A口输出控制七段码管显示，如图9-38所示；或者将简单的并行输入输出接口与可编程的输入输出接口8255结合起来用也行。

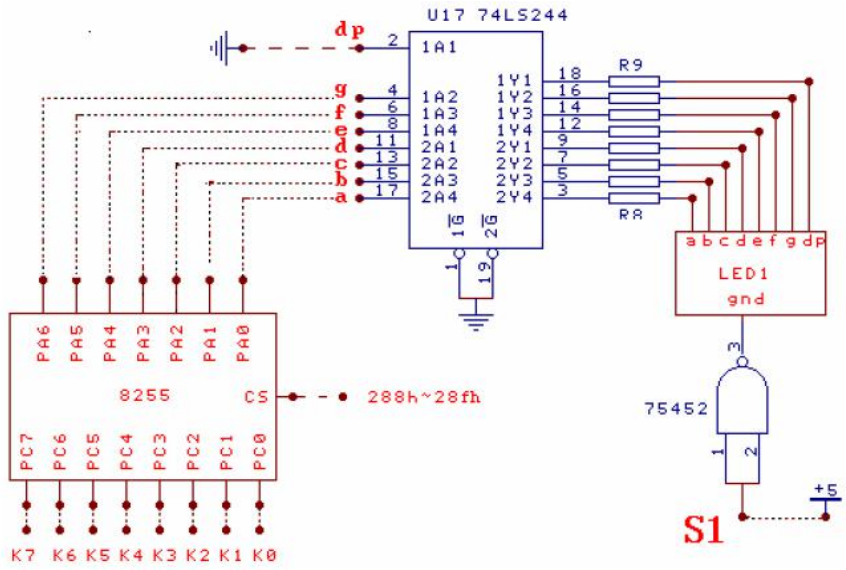


图9-38 竞赛抢答器(模拟)的原理图

（3）播放音乐和定时电路部分可以参考前面的实验，也可以将简单的并行输入输出接口与可编程的输入输出接口8255结合起来用。

编程提示：

这个实验是一个综合性实验，许多内容都是前面做过的，在这个实验中第一次遇到的主要是中断程序。关于中断程序的设计请参考第 7 章 7.3.4 节的 8259 的两个应用实例中的第一个例子。

实验八 交通灯控制

一、实验要求

基本要求：通过实验箱上的并行接口 8255 对两组红、黄、绿 LED 指示灯进行控制，模拟十

字路口南北与东西方向交通灯的控制。每个方向由实验箱上的红黄绿三个 LED 灯控制，共 6 个 LED 灯用于交通信号指示。绿灯的剩余时间（以秒为单位）在七段数码管上同步显示。

完成上述要求后，有条件的同学可以尝试以下工作：

（1）设计一个紧急控制开关信号，当紧急开关信号为“1”时，两个方向的灯全为红灯。紧急开关撤消后，按照开关按下之前的状态继续运行（其参数要保存）。

（2）设计一个夜间行车开关，当开关按下后，两个方向都只有黄灯闪烁，其它灯熄灭。

二、编程提示

根据交通控制要求，不同的亮灯对应有不同的时间。一般以 s 为单位对交通灯进行控制，计时可通过定时器 8253 来实现，根据当前时刻决定各个灯的亮灭。40s 为一个周期循环，一个周期内各灯的变化规律如表 9-4 所示（注：表中‘1’表示灯亮的状态，空白表示灯灭的状态）。紧急情况和夜行情况作为两种特殊的情况处理即可。

另外，由于并行接口 8255 接口数目的限制，七段数码管可采用简单并行接口芯片 74LS273 来进行控制。

表 9-4 交通指示灯的变化规律

时间（s）	东西绿	东西黄	东西红	南北绿	南北黄	南北红
0-15	1					1
15-20		1				1
20-35			1	1		
35-40			1		1	

实验九 串行通信

实验内容：按图9-39连接电路，8251插在通用插座上，8253计数器用于产生8251的发送和接收时钟，TXD和RXD连在一起。请实现从键盘输入一个字符，将其ASCII码加1后发送出去，并接收回来在屏幕上显示，实现自发自收。要求分别编写实现下列两种收发方式的程序：

1．系统以查询方式发送数据，以中断方式接收数据；

2．系统以中断方式发送数据，以查询方式接收数据。

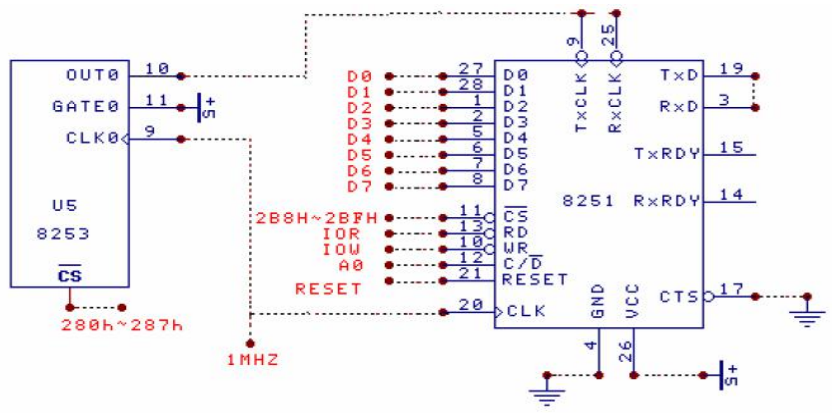


图 9-39 串行通讯电路图

实验提示：

8251没有内置的波特率发生器，必须由外部产生建立波特率的时钟信号，TXC、RXC可与8253连接。

采用查询方式，在数据交换前应读取状态寄存器。状态寄存器D0=1，CPU可以向8251数据端口写入数据，完成串行数据的发送。状态寄存器D1=1，CPU可以从8251数据端口读出数据，完成一帧数据的接收。

编程提示：

(1) 8251 的控制口地址为2B9H，数据口地址为2B8H。

(2) 8253 计数器的计数初值=时钟频率 /（波特率×波特率因子），这里的时钟频率接脉冲源1MHz，波特率若选1200，波特率因子若选16，则计数器初值为52。

(3)8251使用规则：方式选择字和命令控制字用同一个地址寄存器，先写方式字，后写控制字；状态寄存器是只读的。由于串行异步通信速率一般比较慢，在开始初始化时，要注意每送一个命令字，都必须等到8251接收之后才能再进行下一步。

实验十 可编程并行接口（二）（8255方式1）

实验内容：（1）在图9-40电路中，加入简单并行接口74LS244接拨动开关K0-K7，每按一次单脉冲按钮产生一个正脉冲使8255产生一次中断请求，让CPU进行一次中断服务：从244并行接口读入拨动开关状态，并从8255A口输出，在LED显示出来。

（2）在图9-41电路中，每按一次单脉冲按钮产生一个正脉冲使8255产生一次中断请求，让CPU进行一次中断服务：读取拨动开关K0-K7的状态，设定4种状态，每种状态屏幕上显示不同信息，例如“1”状态屏幕显示信息‘We are doing the experiment’等。

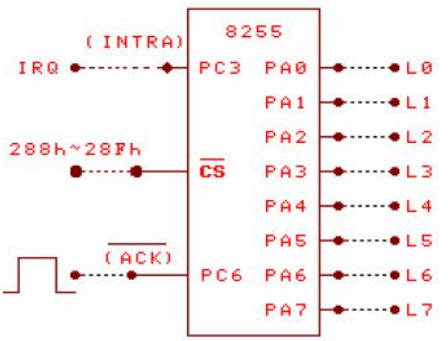


图 9-40 输出电路图

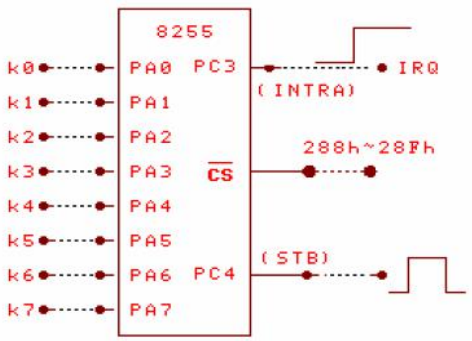


图 9-41 输入电路图

实验十一　模/数转换—定时采样并画采样波形

1．原理：实验电路原理图如图9-42所示。通过实验台左下角电位器RW1输出0～5V直流电压送入ADC0809通道0(IN0)，启动A/D转换器（即向ADC0809的端口地址输出），输入命令读取转换结果。对于图9-22，一次A/D转换的程序可以为

```
MOV     DX,298H
```

```
OUT     DX,AL        ；启动转换
……                   ；延时
IN      AL,DX        ；读取转换结果放在AL中
```

ADC0809有8路输入，采集哪一路由A2 A1 A0决定，A2 A1 A0=000，采集IN0，对应图9-22中的地址为298H；若要采集IN1，则需设置A2 A1 A0=001，对应图9-22中的地址为299H。

2．要求：用8253定时每5秒启动ADC转换，同时利用ADC转换结束信号(EOC)，以查询方式采集一批数据（采集过程中扭动变阻器RW1），并以采样到的样值为纵坐标、时间间隔为横坐标在屏幕上画图形，采集到的数据还送往内存保存。按ESC键，停止采集，程序退出。通过改变各定时器的计数初值控制采样频率和采样持续时间。

请设计硬件连接并编写程序实现上述功能。

3．提示：可选择8253的某个计数器工作在方式0，OUT引脚的输出信号通过8255的PC1读入，变为高电平时，启动A/D转换。A/D转换的结束信号EOC的状态通过8255的PC0读入，查询PC0确定是否转换结束，然后读入转换结果。原理示意图如图9-43所示。

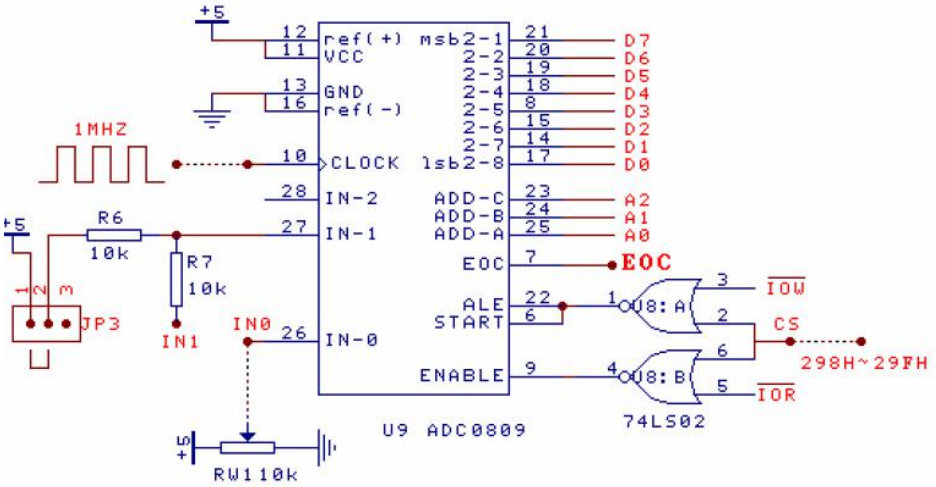


图 9-42　模数转换电路原理图

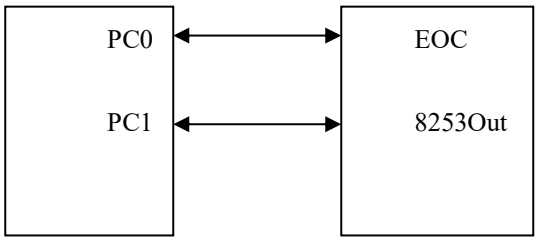
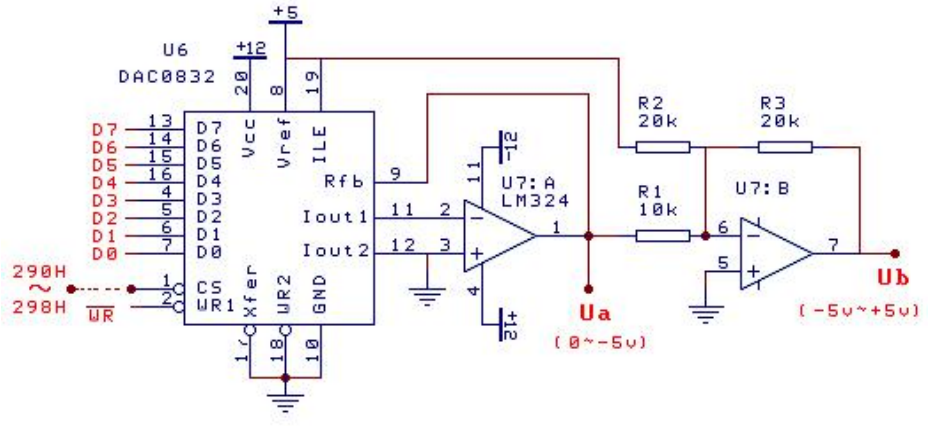


图9-43 定时采集、查询输入示意图

实验十二　数/模、模/数转换联合实验



$$Ua = -\frac{U_{REF}}{256} \times N \qquad Ub = -\frac{U_{REF}}{256} \times N - 5$$

(UREF 表示参考电压,N 表示数数据)，这里的参考电压为P C 机的+ 5 V 电源。

图9-44　数/模转换原理图

实验内容：PC机用OUT指令将数据送给D/A转换器进行转换，再将转换后得到的电压值送A/D转换器进行转换，并将A/D转换后的数据送屏幕画波形。要求画出的波形为锯形波，请设计硬件连接，并编程序实现上述功能。如果要得到正弦波，又该怎么连接？请编程实现这一功能。

参考方案：

（1）画锯形波时,从D/A转换器的单极性输出端Ua将D/A转换后的电压值送A/D转换器的IN0进行转换，电路如图9-44所示；

（2）画正弦波时,从D/A转换器的双极性输出端Ub将D/A转换后的电压值送A/D转换器的IN1进行转换，将JP3的1、2短接，使IN1处于双极性工作方式。

实验十三　步进电机控制实验

一、实验原理

步进电机驱动原理是通过对每相线圈中电流的顺序切换来使电机作步进式旋转。驱动电路由脉冲信号来控制，所以调节脉冲信号的频率便可改变步进电机的转速。

如图9-45所示：电机线圈由四相组成，即:φ1(BA)；φ2(BB)；φ3(BC)；φ4(BD)

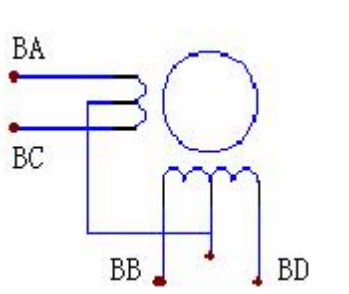


图9-45　步进电机原理图

驱动方式为二相激磁方式，各线圈通电顺序如表9-5所示。

表9-5　　各线圈通电顺序

顺序 \ 相	Φ 1	Φ 2	Φ 3	Φ 4
0	1	1	0	0
1	0	1	1	0
2	0	0	1	1
3	1	0	0	1

反时针方向回转

正时针方向回转

表中首先向φ1线圈－φ2线圈输入驱动电流，接着φ2－φ3，φ3－φ4，φ4－φ1，又返回到φ1－φ2，按这种顺序切换，电机轴按顺时针方向旋转。反之则反方向旋转。

实验可通过不同长度延时来得到不同频率的步进电机输入脉冲，从而得到多种步进速度。

二、提高性实验内容

1、按图9-46连接线路，设置8255A口为方式1输出，用简单接口74LS244读取实验箱上8个逻辑电平开关状态。每按一次单脉冲产生一个正脉冲使8255产生一次中断 ，让CPU进行一次中断服务，在中断服务程序中通过简单接口74LS244读取拨动开关K0-K7状态，如果K0=1，则转速最高；K6=1,转速最低；转速调整通过改变延时时间来实现。如果K0-K6都为0，则不转，由K7决定是正转还是反转。实验中判断键盘有无键按下，若有键按下，则结束中断并返回，继续等待下一个中断脉冲（单脉冲）的到来。

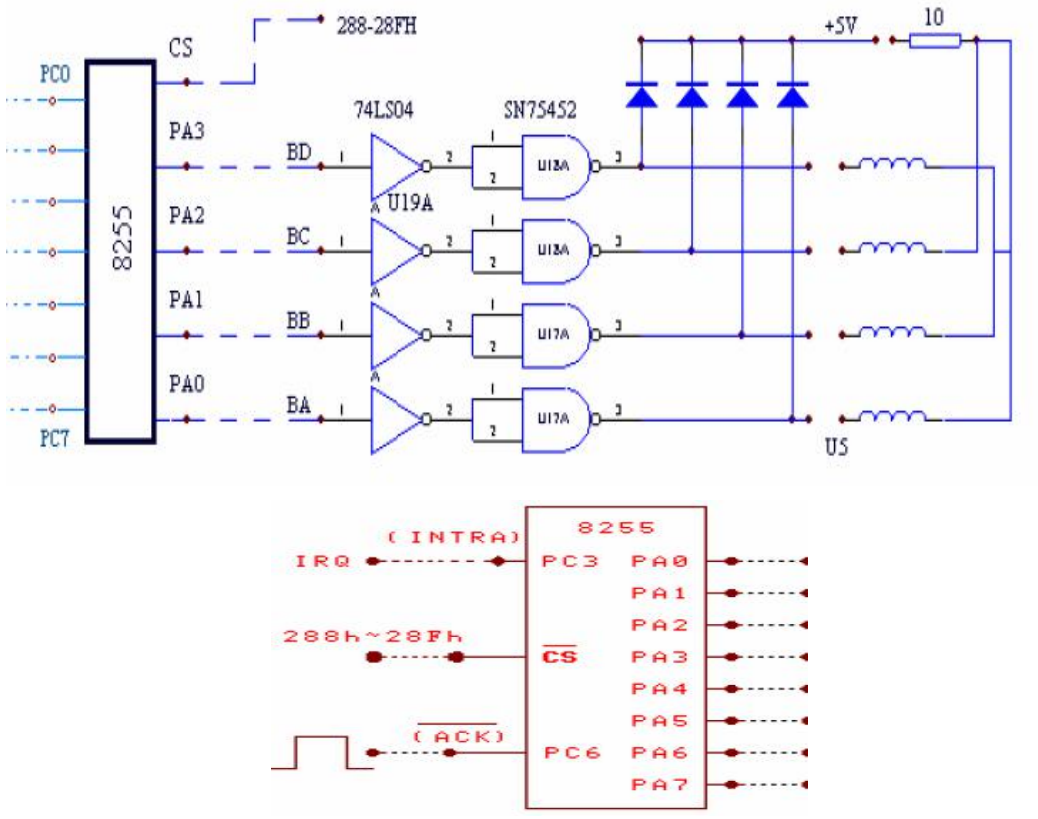


图9-46　通过8255（方式1）控制步进电机

三、综合性实验内容

基于TPC通用微机接口实验系统，在微处理器的控制下，利用模数转换芯片ADC0809，并行接口芯片8255，可编程计数器芯片8253，通过实验箱外围的电位器旋钮实时地对转动中的步进电机进行转速和方向的调控。

1. 步进电机的控制

实验中所用步进电机的驱动原理为让步进电机的四个端口的通电顺序依次为1100→0110→0011→001→1100循环进行，即可实现转动。同理，如果让四个端口的通电顺序颠倒，即按照1100→1001→0011→0110→1100 循环进行，即可实现步进电机按相反方向转动，可由8255输出控制。步进是靠8255 每进行一次I/O 输出后的固定延时来实现。延时时间越长，转动越慢；延时时间越短，转动越快。延时可用8253实现。

2. 实时转速控制的实现

在步进电机工作时，转动电位器旋钮，改变电位器的电压值，经ADC0809转换后，实时读取电位器旋钮的电压输入值，通过输入值的不同来改变延时长度，从而控制电机转动。

3. 紧急停止功能

如果要让步进电机停止转动，可设计为先按停止键“S”，然后再判断是退出“Q”还是继续工作“R”。也可以采用中断处理方式，通过实验板上的单脉冲按钮，每按一次单脉冲发出一次中断请求，在中断服务程序中进行判断停止运行等。

4. 转速设置提示

变阻器中间位置转速为零。逆时针旋转一点，步进电机开始逆时针旋转，随着逆时针旋转的增加，旋转逐渐加快（可设计5个转速档）。如果在中间位置顺时针转动，步进电机顺时针转动，随着旋转的增加，旋转逐渐加快（也可以是5个转速档）。即，如果旋钮在最左边，向右旋转，步进电机的转速依次为：

逆5档（最快）—>逆4档—>逆3档（中）—>逆2档—>逆1档（最慢）—>停止—>顺1档（最慢）—>顺2档—>顺3档（中）—>顺4档—>顺5档（最快）

实验十四　期末综合实验---设计实现一个音乐闹钟

这里给出一个实验例子，该例子是微机原理及接口实验的一个期末考试答卷，该程序已经在TPC-USB 系统上调试运行，达到了设计的要求，效果也不错，供大家参考。

【设计目的】

- 进一步掌握一个学期以来所做实验用到的各种元件的使用方法与编程；
- 使用的芯片包括：8255、8253、74LS244、74LS273、AD0809 ；
- 提高综合解决问题的能力

【设计基本要求】

a) 实现时钟功能，可以在两个七段数码管上显示秒钟时间或者分钟时间，用一个开关控制两者的切换。

b) 实现闹钟功能，时间到播放一段音乐，并在发光二极管上播放走马灯图案，在双色点阵发光二极管上滚动显示自己的学号。能控制滚动显示的速度以及音乐播放的速度，且用一个置位开关控制闹钟的开和关。

【设计已达到效果】

a) 实现时钟功能，可以在两个七段数码管上显示秒钟时间或者分钟时间，可以用一个开关切换；

b) 可以设定闹钟时间，用开关控制闹钟的开启与关闭；

c) 实现并行地播放音乐、跑马灯、滚动显示学号和名字，同时还可以显示时间，进行分秒切换，进行速度控制；

d) 在闹钟响起时，可以通过开关关闭闹钟，关闭闹钟后能重新设定闹钟与显示时间；

e) 通过 ADC0809 控制滚动显示的速度以及音乐播放的速度；

f) 跑马灯速度节奏随音乐节奏而变；

g) 双色点阵 LED 可以红、黄灯显示切换；

h) 可以播放不同的音乐。

【设计原理】

1. 时钟功能

8253具有3个独立的16位计数器通道，每个通道都可以通过编程设定为6种工作方式之一，可

设定为按二进制计数或二—十进制计数。将计数器0、计数器1分别设置为方式3，计数初值设为1000，使 OUT1输出频率为1Hz电平的变化。8253的连接如图9-47所示。

将OUT1接至IRQ引脚，在中断服务子程序里对秒数进行计数。

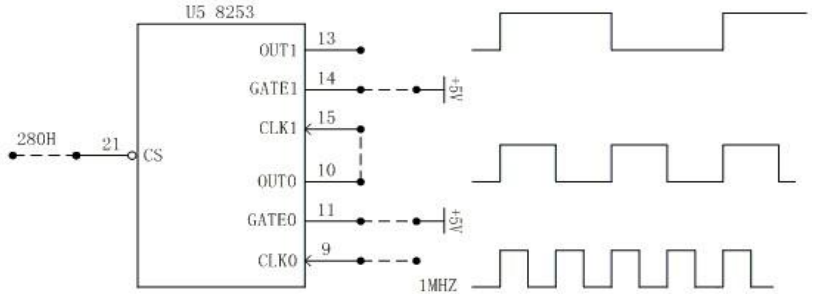


图9-47　8253连接图

2. 数码管动态显示

a) 8255 是一个可编程并行输入/输出的多功能的 I/O 器件，具有 PA、PB、PC 三个 I/O 端口，可配置 3 种工作方式。

b) 实验台上的七段数码管为共阴型，段码采用同相驱动，输入端加高电平,选中的数码管亮，位码加反相驱动器，位码输入端高电平选中。

c) 按图 9-48 电路图连线，将 8255 的 A 口 PA0～PA6 分别与七段数码管的段码驱动输入端 a～g 相连，8255 的 C 口 PC1、PC0 与位码驱动输入端 S1、S0 相连。

d) 将 8255 的 PA、PC 口均设为输出工作方式 0，从 A 口输出要显示的数字的段码，PC1、PC0 控制选通位码，通过软件控制实现动态显示。

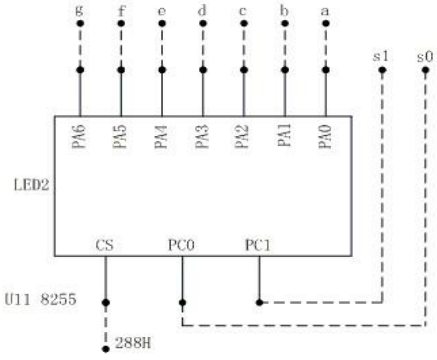


图 9-48　8255 连接图

3. **音乐播放**

a) 电路如图 9-49，8253 的 CLK2 接 1MHz 时钟，GATE2 接 8255 的 PC1，OUT2 和 8255 的 PC0 接到与门的两个输入端，K8 跳线连接喇叭。

b) 利用 8255 的 PC0 口作为与门输入，用来控制扬声器的开关状态。再利用设置不同的计数值，使 8253 产生不同频率的波形，使扬声器产生不同频率的音调，达到类似与音阶的高低音变

换。对于音乐，每个音阶都有确定的频率。只要给定一首曲子的每个音调的频率和节拍，便可编程控制输出美妙的音乐。

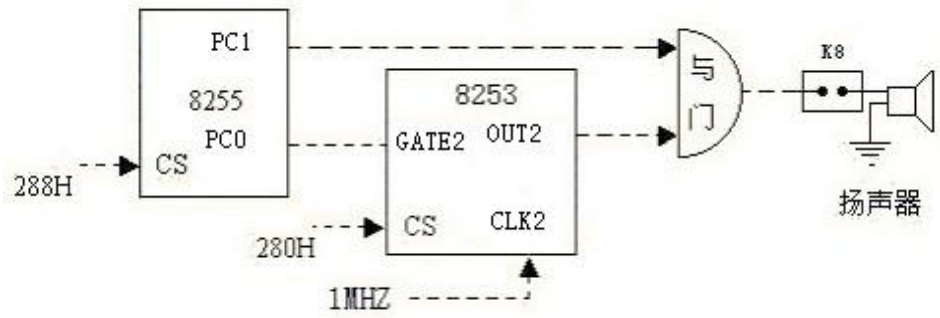


图 9-49 音乐播放连接图

4. **走马灯图案**

a) 按下面图 9-50 简单并行输出接口电路图连接线路（74LS273 插通用插座，74LS32 用实验台上的“或门”）。74LS273 为 8 个 D 触发器，8 个 D 输入端分别接数据总线 D0～D7，8 个 Q 输出端接 LED 显示电路 L0～L7。

b) 通过编程，控制在不同时间，向 74LS273 输出不同的数据，从而使 L0～L7 有不同组合的亮灭情况，从而形成走马灯花样。

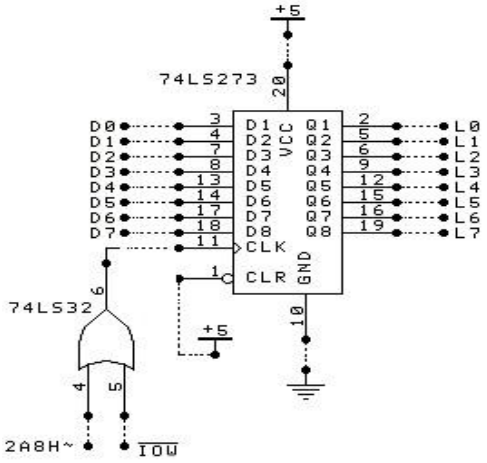


图 9-50 走马灯连接图

5. **双色点阵 LED 上滚动显示自己的学号**

a) 点阵 LED 显示器是将许多 LED 类似矩阵一样排列在一起组成的显示器件，双色点阵 LED 是在每一个点阵的位置上有红绿或红黄或红白两种不同颜色的发光二极管。当微机输出的控制信号使得点阵中有些 LED 发光，有些不发光，即可显示出特定的信息，包括汉字、图形等。

b) 实验仪上设有一个共阳极 8×8 点阵的红黄两色 LED 显示器。该点阵对外引出 24 条线，其中 8 条行线，8 条红色列线，8 条黄色列线。若使某一种颜色、某一个 LED 发光，只要将与其

相连的行线加高电平，列线加低电平即可

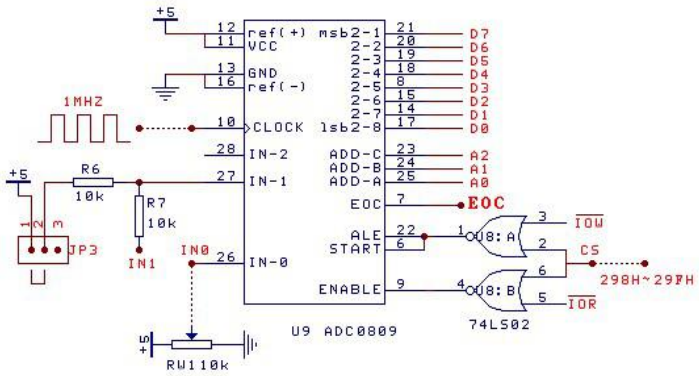
c) 先定义自己的学号，如“07302172”所对应的点阵数据，然后编程控制逐次取出 8 个点阵数据送往双色点阵 LED 显示，直至遇到结束符“－1”。

6. 速度控制

a) 在播放音乐、跑马灯和滚动显示学号时，要求可以控制它们的速度，可以通过在程序中插入延时达到目的。而且为了控制速度，需要可以随时调整延时的大小。

b) 可以利用 AD0809 转换结果作为延时变量。在改变模拟量输入时，AD0809 转换结果也随之改变，从而延时变量的大小发生改变，即延时长短得以改变，达到控制速度的目的。

c) 模数转换器 AD0809 接口如图 9-51 所示：



$$N = \frac{U_i}{U_{REF}/256}$$

其中 U_i 为输入电压，UREF 为参考电压，这里的参考电压为 PC 机的+５Ｖ电源。

图 9-51　速度控制连接图

7. 分秒切换与闹钟的开关

a) 74LS244 为 8 缓冲器，8 个数据输入端分别接逻辑电平开关输出 K0～K7，8 个数据输出端分别接数据总线 D0～D7，如图 9-52 所示。

b) 读取 8 个开关的逻辑状态，根据开关 K0 是否为“1”来确定是否开启闹钟；根据开关 K1 是否为“1”来决定是显示分还是秒。

c) 由于实验箱上只有一个或门，因此需要在 40 引脚通用插座上扩展一个 74LS32 来作为 74LS244 的地址选通和读端口。

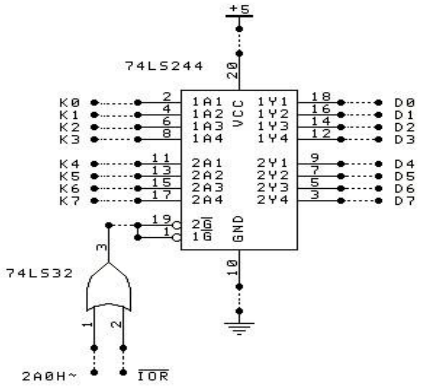


图 9-52　分秒切换与闹钟的开关连接图

【软件流程图】

1.　主程序流程图

主程序流程图如图 9-53 所示。

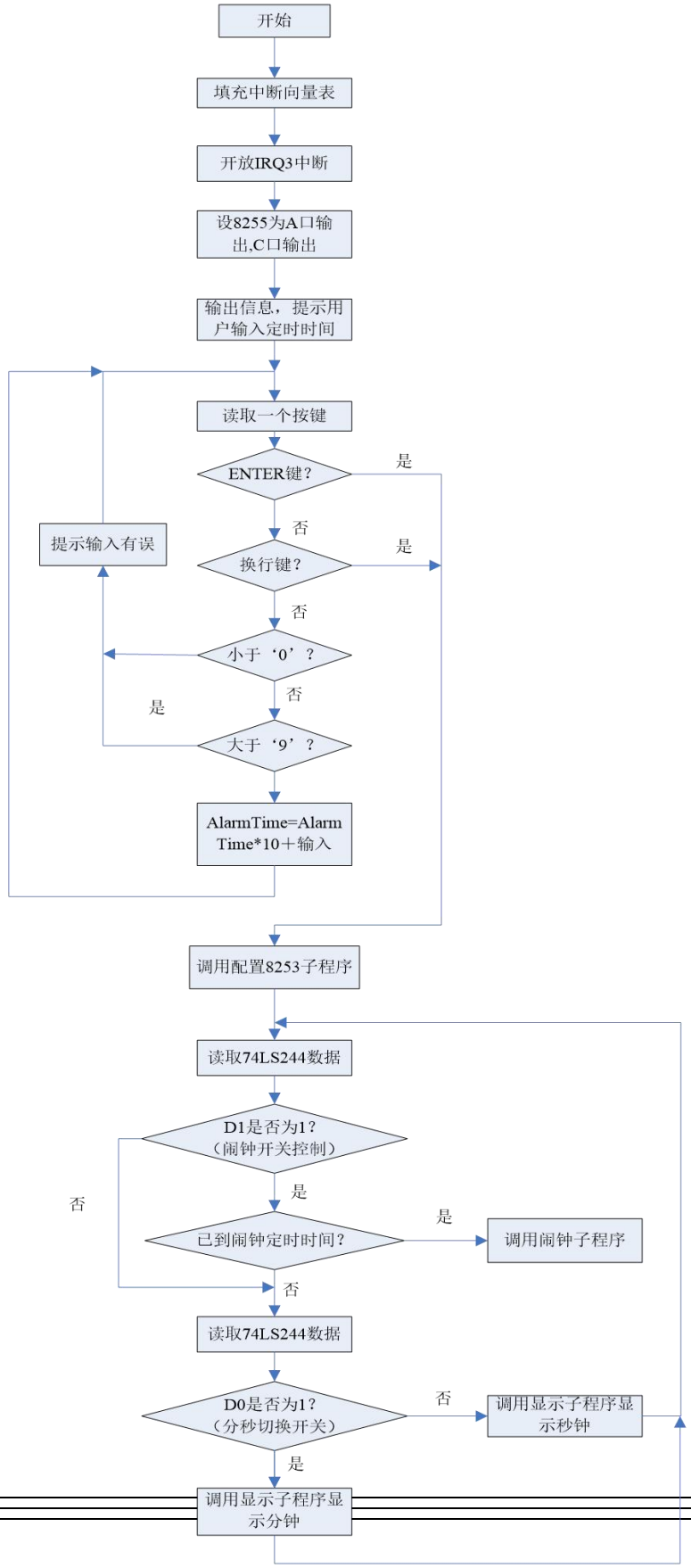

开始
填充中断向量表
开放IRQ3中断
设8255为A口输出,C口输出
输出信息，提示用户输入定时时间
读取一个按键
ENTER键?
是
否
换行键?
是
否
小于‘0’?
否
是
大于‘9’?
提示输入有误
AlarmTime=AlarmTime*10＋输入
调用配置8253子程序
读取74LS244数据
D1是否为1?（闹钟开关控制）
否
是
已到闹钟定时时间?
是
调用闹钟子程序
否
读取74LS244数据
D0是否为1?（分秒切换开关）
否
调用显示子程序显示秒钟
是
调用显示子程序显示分钟

图 9-53 程序主流程图

2. 闹钟音乐子程序

本设计的难点在于音乐、跑马灯，滚动显示和时间显示同时运行。

设计亮点：考虑到音乐播放的时候，由于音调节拍之间的延时较长，因此可以在音乐的　延时程序里实现其它功能，利用执行跑马灯，滚动显示和时间显示等程序的时间来作为音乐的延时，可以有效地解决并行同时实现各种功能的问题。

闹钟子程序流程图如图 9-54 所示。

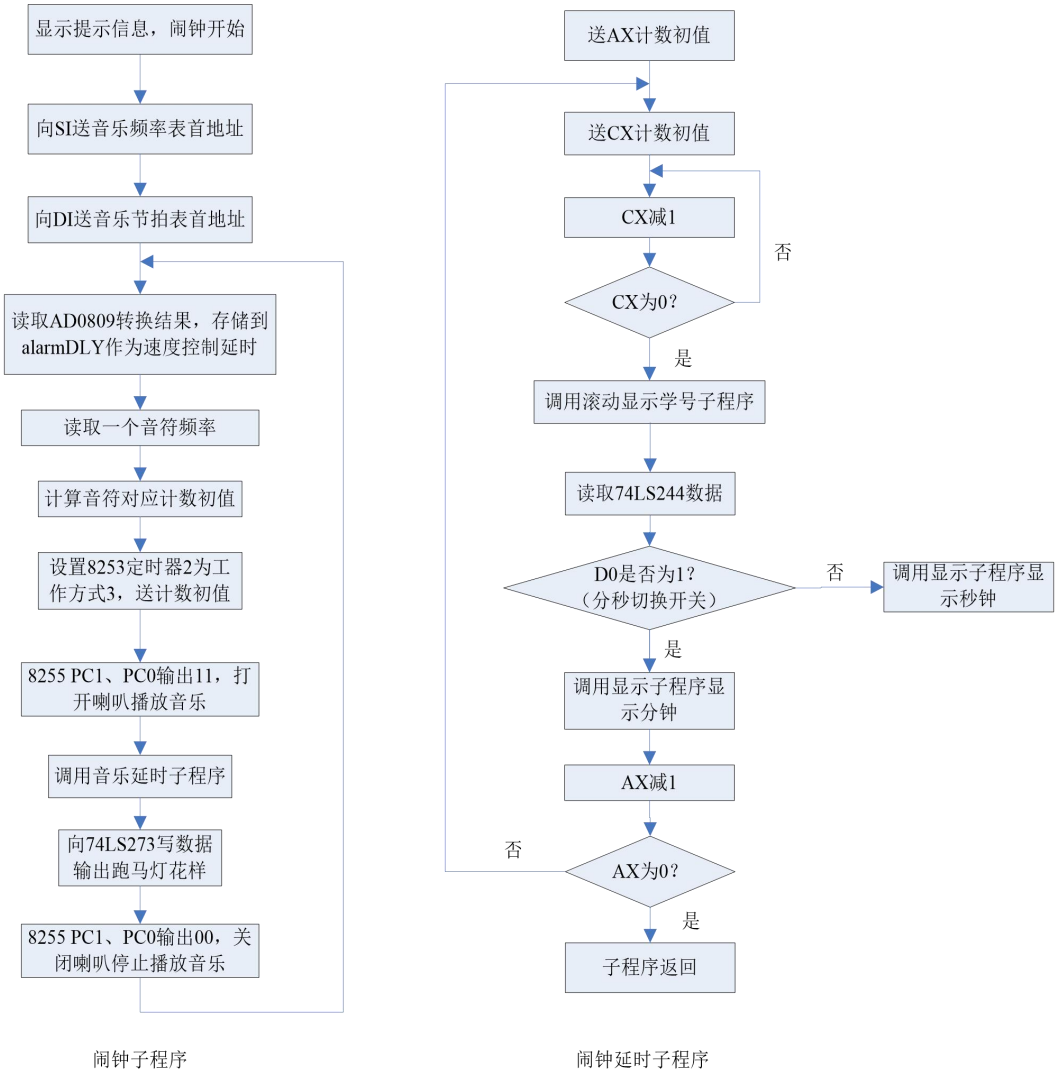


图 9-54 闹钟子程序流程图

3. 滚动显示学号子程序

要滚动显示学号和名字，只需要将事先定义好的学号和名字数据顺序逐次读取显示即可，

其程序流程图如图 9-55 所示。

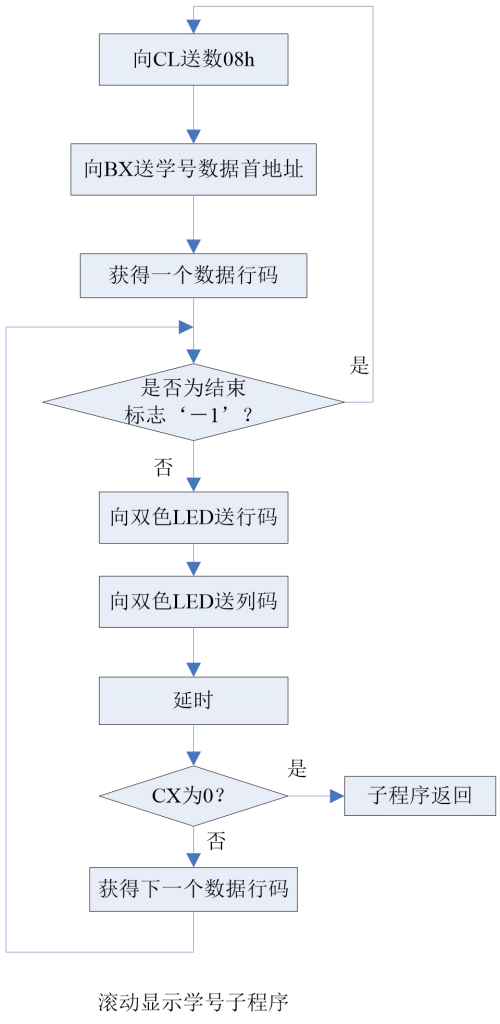


图 9-55 滚动显示学号和名字流程图

4. 数码管显示子程序

在调用显示子程序之前，先要将待显示的数字放入 AX，然后利用 AAM 指令进行 BCD 码校正，以得到十进制的时间数据。数码管显示子程序流程图如图 9-56 所示。

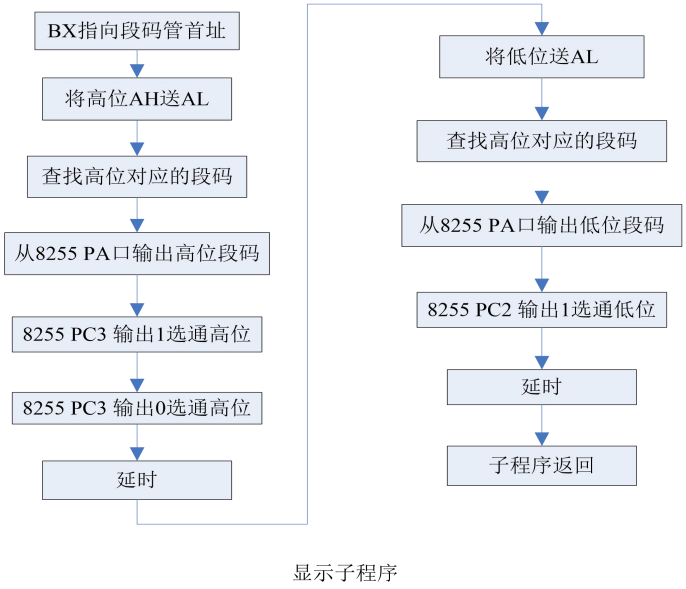


图 9-56 数码管显示子程序流程图

【汇编程序代码】

```
;*******************************************************
;a)      实现时钟功能，可以在两个七段数码管上显示秒钟时间或者分钟时间，
;       用一个开关控制两者的切换。
;b)      实现闹钟功能，时间到播放一段音乐，并在发光二极管上播放走马灯
;       图案，在双色点阵发光二极管上滚动显示自己的学号。能控制滚动显
;       示的速度以及音乐播放的速度，且用一个开关控制闹钟的开关。
;*******************************************************
;       定义各元件地址端口
CR_8253         equ 283h
T0_8253         equ 280h
T1_8253         equ 281h
T2_8253         equ 282h
PA_8255         equ 288h
CR_8255         equ 28bh
PC_8255         equ 28ah
LS244           equ 2a0h
io0809          equ 298h
LS273           equ 2a8h
proth           equ   2b0h
protlr          equ   2B8h
protly          equ   290h
data            segment         ;数据段声明
```

```
    led             db          3fh,06h,5bh,4fh,66h,6dh,7dh,07h,7fh,6fh; led 数码管段码查找
表
    ; 设定闹钟时间提示信息
    msg             db  0dh,0ah,       'please  input  the  alarm  time:  (end  with  Enter)',
0dh,0ah,'$'
    alarmsg         db 'begin alarm',0dh,0ah,'$'
    AlarmTime  dw ?       ; 闹钟时间
    second     dw 0         ; 秒钟
    minute     dw 0         ; 分钟
    alltime    db 0         ; 总时间
    alarmDLY   db 0         ; 速度控制延时变量
    light_status    db 10000000b ; 跑马灯状态变量
    roll_i          db 0          ; 控制双色 LED 滚动显示变量
    music_freq      dw 330,294,262,294,3 dup(330),3 dup(294),330,392,392     ;音乐频率表
                    dw 330,294,262,294,4 dup(330),294,294,330,294,262,-1
    music_time      dw 6 dup  (1),2 ,2 dup (1),1 ,2                  ; 音乐节拍表
                    dw 12 dup  (1),3 dup(1)
    buff            db 00h,00h,00h,00h,00h,00h,00h,00h    ; 学号与名字对应码表
                    db 00h,3Ch,42h,81h,81h,42h,3Ch,00h    ; '0'
                    db 00h,40h,40h,47h,48h,50h,60h,00h    ; '7'
                    db 00h,00h,40h,49h,59h,6Ah,44h,00h    ; '3'
                    db 00h,3Ch,42h,81h,81h,42h,3Ch,00h    ; '0'
                    db 00h,26h,4Ah,4Ah,52h,22h,00h,00h    ; '2'
                    db 00h,00h,00h,42h,0FEh,02h,00h,00h   ; '1'
                    db 00h,40h,40h,47h,48h,50h,60h,00h    ; '7'
                    db 00h,26h,4Ah,4Ah,52h,22h,00h,00h    ; '2'
                    db 00h,00h,00h,00h,00h
                    db 001h,07Eh,080h,0AFh,0FDh,0AFh,080,07Fh  ;'周'
                    db 002,014h,058h,07Fh,0F5h,07Fh,054h,012h  ;'春'
                    db 012h,07Ah,0C2h,027h,0F2h,02Ah,02Ah,008h ;'华'
                    db 00h,00h,00h,00h,00h,00h,00h,-1
    light_sample    db 01h,02h,04h,08h,10h,20h,40h,80h    ; 跑马灯花样表
    data            ends
    code            segment
              assume    cs:code, ds:data
    start:    mov       ax,cs
                  mov       ds,ax
                  mov       dx,offset int3    ; 填充中断向量表
                  mov       ax,250bh
                  int       21h
```

```
                in      al,21h              ; 开放中断 IRQ3
                and     al,0f7h
                out     21h,al
                sti                     ; 开放总中断
                mov     ax,data
                mov     ds,ax
                mov     dx, CR_8255         ;设置 8255 为 A 口输出,C 口输出
                mov     ax,80h
                out     dx,al
                mov     dx, offset msg      ;提示输入闹钟定时
                mov     ah, 09h
                int     21h
ReadAlarmTime :mov      AlarmTime,0             ;读取闹钟定时时间
rdagain:        mov     ah,01            ;read a char
                int     21h
                cmp     al,13           ;如果是 enter 键则结束读数
                je      fdone1
                jmp     tdone1
fdone1:         jmp     rdAlTimeDone
tdone1:         cmp     al,10           ;如果是换行键则结束读数
                je      fdone1
                cmp     al,'0'          ;判断输入范围是否为 0～9
                jb      ReadAlarmTime
                cmp     al,'9'
                ja      ReadAlarmTime
                push    ax
                mov     ax,10
                mul     AlarmTime
                mov     AlarmTime,ax        ;AlarmTime=AlarmTime*10
                pop     ax
                sub     bx,bx
                mov     bl,al
                sub     bl,30h
                add     AlarmTime, bx       ;AlarmTime=AlarmTime+input
                jmp     rdagain
rdAlTimeDone:   call    config_8253             ;读取闹钟定时时间完成
rd244:          mov     dx, LS244            ;读取 74LS244 端口数据
                in      al, dx
                and     al, 02h              ;D1 口是否为 1?  D1 为 1 时闹钟开，0 时闹钟
关
```

```
                jz      next2
                mov     ax,  AlarmTime      ;闹钟时间已到？？
                cmp     al, alltime
                je      alarm
next2:          mov     dx, LS244           ;读取 74LS244 端口数据
                in      al, dx
                and     al, 01h             ;D0 口是否为 1，D0 接分、秒选择开关。
                jnz     disp_min
                jmp     disp_second
disp_min:  mov      ax, minute            ;显示分钟
                AAM
                call    led_disp              ;调用数码管显示子程序
                jmp     rd244
disp_second:    mov     ax,second             ;显示秒钟
                AAM
                call    led_disp              ;调用数码管显示子程序
                jmp     rd244
alarm:          mov     ax,data               ;  闹钟子程序
                mov     ds,ax
                mov     ah,09h
                mov     dx, offset alarmsg    ;提示闹钟开始
                int     21h
                mov     si, offset music_freq ;送 SI 音乐频率表首地址
                mov     di,  offset music_time ;送 DI 音乐节拍表首地址
rd0809:         mov     dx, io0809            ;读 AD0809
                out     dx,al
                mov     cx,0fh                ;延时
ADdly:          loop    ADdly
                in      al,dx             ;从 A/D 转换器输入数据
                cmp     al,1
                ja      rd0809next
                mov     al,1
rd0809next:     mov  alarmDLY,   al     ;将读取结果存储到延时变量 alarmDLY
play_music:                               ;播放音乐
sing:           mov     ax,4240H        ;计数初值 = 1000000 / 频率, 保存到 AX
                mov     dx,0FH
                push    cx
                mov     cx, [si]        ;获得第 SI 个音乐频率
                cmp     cx, -1          ;判断是否为结束标志符－1
                je      alarm           ;是则重新开始播放
```

```
                div     cx
                pop     cx
                mov     bx,ax
                mov     dx,CR_8253 ;设置 8253 计时器 2 为方式 3, 先读写低字节, 再读写高
字节
                mov     al,10110110B
                out     dx,al
                mov     dx,T2_8253
                mov     ax,bx
                out     dx,al           ;写计数器 2 初值低字节
                mov     al,ah
                out     dx,al           ;写计数器 2 初值高字节
                mov     dx,CR_8255       ;设置 8255 C 口输出
                mov     al,10000000B
                out     dx,al
                mov     dx,PC_8255
                mov     al,03h
                out     dx,al           ;置 PC1PC0 = 11(开扬声器)
                push    cx
                mov     cx, [di]
music_pad: call     music_delay     ;调用音乐延时子程序
                loop    music_pad
                push    dx
                push    ax
light:    mov     dx,LS273          ;LS273 用于控制 LED 跑马灯
                mov     al, light_status
                out     dx,al           ;输出跑马灯花样
                shr     al,1
                jnz     l_next
                mov     al, 10000000b
l_next:         mov     light_status, al
                pop     ax
                pop     dx
                pop     cx
                mov     al,0h
                out     dx,al          ;置 PA1PA0 = 00(关扬声器)
                inc     si
                inc     si
                inc     di
                inc     di
```

```
                jmp     rd0809
music_delay     proc    near                ;音乐延时子程序
                push    cx
                push    ax
                mov     ax,5
x1:             mov     cx,10
x2:             dec     cx
                jnz     x2
                call    L_and_R             ;调用滚动显示子程序
                push    ax
                push    dx
rd2441:         mov     dx, LS244           ;读取 LS244 端口数据
                in      al, dx
                and     al, 02h             ;D1 口是否为 1?  D1 接闹钟开关
                jnz     next21
                mov     dx, PC_8255
                in      al,dx
                and     al,0fch
                out     dx,al
                jmp     rd244
next21:         mov  dx, LS244
                in      al, dx
                and     al, 01h               ;D0 口是否为 1，D0 接分、秒选择开关。
                jnz     disp_min1
                jmp     disp_second1
disp_min1:      mov     ax, minute              ;显示分钟
                AAM
                call    led_disp
                jmp     disp_end
disp_second1:   mov     ax,second               ;显示秒钟
                AAM
                call    led_disp
disp_end:       pop dx
                pop     ax
                dec     ax
                jnz     x1
                pop     ax
                pop     cx
                ret                         ;子程序返回
music_delay     endp
```

```
L_and_R         proc      near              ;滚动显示子程序
                push      ax
                push      cx
roll_name:      mov       cl,08h
                mov       ah,80h
r_next:         mov       al,roll_i
                mov       bx,offset buff
                xlat                        ;得到一个行码
                cmp       al,-1
                jnz       r_next1
                mov       roll_i ,0
                jmp       roll_name
r_next1:        mov       dx,proth
                out       dx,al
                mov       al,ah
                mov       dx,protlr
                out       dx,al             ;显示一个行码
                mov       al,0
                out       dx,al
                shr       ah,01
                inc       roll_i
                push        cx
                mov       cx,500
r_delay2:       push      cx
                push      ax
                mov al,   alarmDLY          ;控制总体速度延时
                mov ah,0
                mov cx,ax
                pop ax
r_delay3:       loop    r_delay3
                pop cx
                loop  r_delay2
                pop       cx
                loop        r_next
                mov       al,roll_i
                sub       al,7
                mov       roll_i,al
                pop cx
                pop ax
                ret
```

```
 L_and_R        endp
led_disp   proc near                ;数码管显示子程序
                push dx
                push bx
                push ax                 ;显示高四位
                mov  dx, PA_8255
                mov  al,ah
                mov  bx,offset led      ;使 BX 指向段码管首址
                xlat
                out  dx, al
                mov  dx, PC_8255;
                in   al, dx
                mov  bl, al
                or  al, 08h             ;数码管高位由 PC3 控制
                out  dx, al
                mov  bl, al
                and  al, 0f7h           ;数码管高位由 PC3 控制
                out  dx, al
                push cx
                mov  cx,100
led_delay:     loop led_delay           ;延时
                pop cx
                pop  ax                 ;显示低四位
                mov  dx, PA_8255
                mov  bx,offset led      ;使 BX 指向段码管首址
                xlat
                out  dx, al
                mov  dx, PC_8255;
                in   al, dx
                mov  bl ,al
                or   al, 04h            ;数码管高位由 PC2 控制
                out  dx, al
                mov  al, bl
                and  al, 0fbh           ;数码管高位由 PC2 控制
                out  dx, al
                push cx
                mov  cx,100
led_delay1:    loop led_delay1     ;延时
                pop cx
                pop bx
```

```
                pop dx
                ret
led_disp        endp
config_8253     proc near               ;8253 配置子程序
                push dx
                mov dx,CR_8253          ;设 8253 计数器 0 为方式 3
                mov al,36h
                out dx,al
                mov dx,T0_8253
                mov ax,10000            ;写入计数器初值 10000
                out dx,al
                mov al,ah
                out dx,al
                mov dx,CR_8253
                mov al,76h              ;设计数器 1 为工作方式 3
                out dx,al
                mov dx,T1_8253
                mov ax,100              ;写入计数器初值 100
                out dx,al
                mov al,ah
                out dx,al
                pop dx
                ret                     ;定时时间到，子程序返回
config_8253     endp
;中断服务子程序
int3:       inc  alltime            ;总计时时间加 1
                inc  second             ;秒钟加 1
                cmp  second, 60
                jb   int_next1
                mov  second, 0
                inc  minute             ;分钟加 1
int_next1:      cmp  minute, 60
                jb   int_next2
                mov  minute, 0
int_next2:      mov al,20h
                out 20h,al
                iret
CODE        ENDS
              END       START
```