

第一部分 汇编语言程序设计实验

实验一 汇编语言编程基础

汇编语言是一种面向机器的“低级”语言，是计算机能够提供给用户的最快而最有效的语言，也是能够利用计算机所有硬件特性并能直接控制硬件的唯一语言。要真正理解计算机的工作过程，理解计算机程序的执行过程，就必须学习汇编语言。也正是因为如此，汇编语言程序设计是计算机专业和电子，自动控制等相关专业的重要课程。

但是，对于刚开始学习汇编语言的学生而言，汇编语言的一些命令非常抽象，很难理解，往往学习了很长时间也编不出满意的程序，更别说自如的应用，以致我们认为汇编语言很难掌握，影响我们学习汇编语言的兴趣。实际上，为了掌握好汇编语言，我们可以从熟悉、使用 DEBUG 调试工具开始，先来分析 and 读懂一些与硬件相关的小程序，这也是我们实验一的目的。

1.1 汇编语言程序的上机步骤

以下列源程序为例，先学习汇编语言的上机步骤。

文件名为 1.asm:

```
DATA    SEGMENT
NUM1    DB  35, 35H
NUM2    DW  35, 35H
NUM3    DB  4 DUP (34, 3 DUP (34H))
NUM4    DB  ' 34AB'
NUM5    DW  ' 34', ' AB'
DATA    ENDS
CODE    SEGMENT
ASSUME CS:CODE, DS:DATA
START:  MOV  AX, DATA
        MOV  DS, AX
        MOV  BX, OFFSET NUM1
        MOV  AL, [BX]
        MOV  BX, OFFSET NUM5
        MOV  AX, [BX]
        MOV  AH, 4CH
        INT  21H
CODE    ENDS
END START
```

一. 上机步骤

汇编语言程序 MASM 软件由 EDIT.COM 编辑器, 汇编 MASM.EXE 程序, 连接 LINK.EXE 程序以及 DUEBUG.EXE 调试程序四个部分组成。汇编语言编制完成后, 在计算机上的操作过程就分为四个阶段。

1. 编辑 EDIT.COM

首先输入源程序, 有两种方法:

(1) 在记事本里录入, 特别注意的是: 在保存时文件格式必须选择所有文件, 文件后缀名为 .ASM, 即保存时文件名为 XXX.asm。

(2) 双击 MASM 软件中的编辑软件 EDIT.COM 文件框, 在 EDIT 下输入源程序。用 ALT+F 键打开 file 菜单, 用其中的 save 功能键将文件存盘。特别注意的是: 汇编语言源程序文件的后缀必须为 .asm, 即保存时文件名为 XXX.ASM。

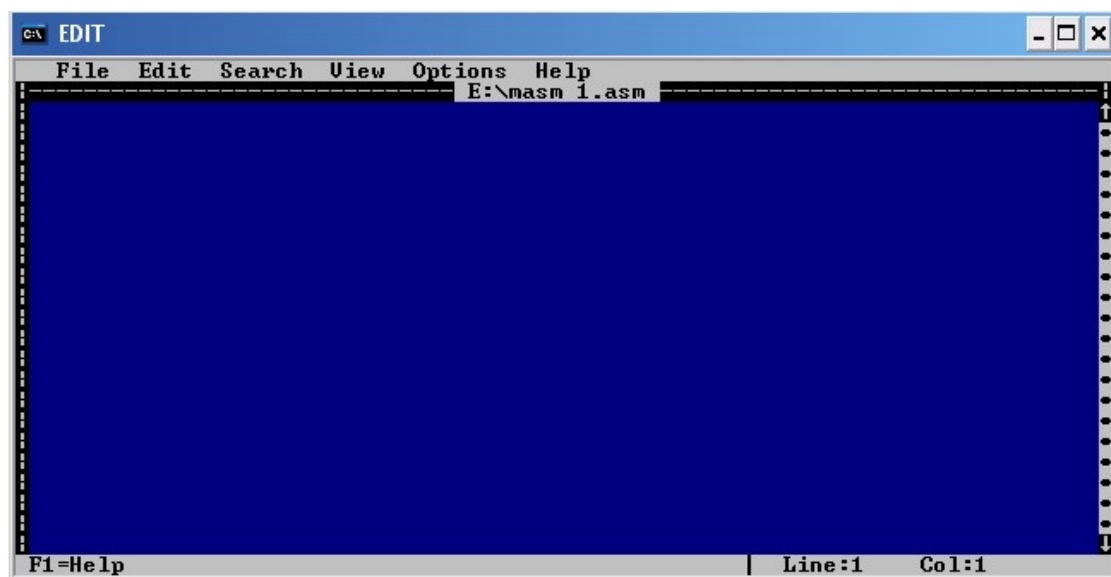
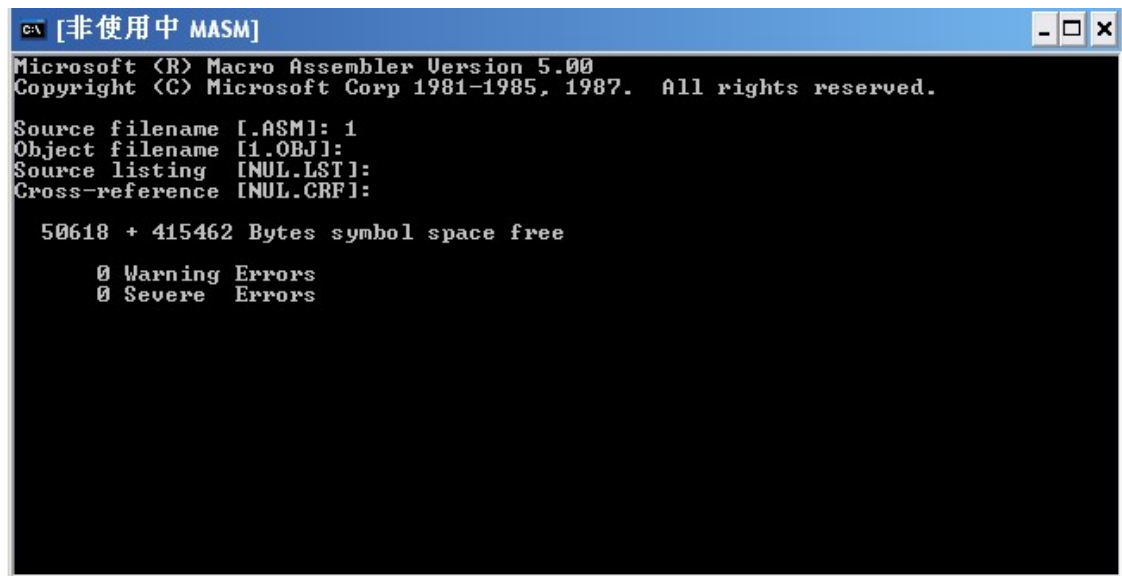


图 1-1 EDIT 编辑界面

2. 用汇编程序 MASM.Exe 对源程序 .ASM 文件汇编, 生成目标文件 .OBJ

汇编阶段的任务是把汇编语言源程序翻译成机器代码 (称为目标), 产生二进制格式的目标文件 XXX.OBJ (名字与源程序名相同, 只是后缀名不同), 如果源程序有语法错误, 则汇编过程结束后, MASM.EXE 汇编程序会指出源程序中错误的行号和错误的原因, 我们可以再用编辑程序 EDIT.com 来修改源程序中的错误, 汇编无错后, 方可得到正确的 .OBJ 目标文件, 才能进行下一部的连接 LINK。

双击 MASM 软件中的 MASM.Exe 文件框, 在命令行后键入源程序名 XXX.asm, (如果源程序与 MASM 软件在同一路径下, 可以只键入文件名, 而不要后缀), 如以下界面:



```
C:\ [非使用中 MASM]
Microsoft (R) Macro Assembler Version 5.00
Copyright (C) Microsoft Corp 1981-1985, 1987. All rights reserved.

Source filename [1.ASM]: 1
Object filename [1.OBJ]:
Source listing [NUL.LST]:
Cross-reference [NUL.CRF]:

50618 + 415462 Bytes symbol space free

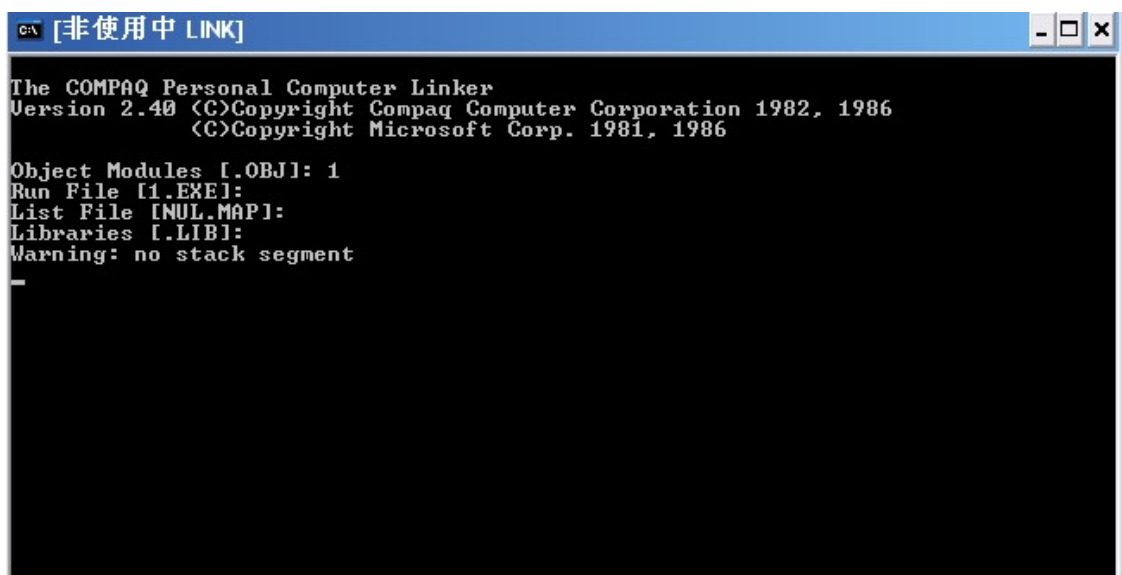
0 Warning Errors
0 Severe Errors
```

图 1-2 MASM 编译 1.asm 文件无错误时的界面

3. 连接 LINK.EXE

由于汇编所得到的目标代码的存放地址并不是可执行的绝对地址,而是浮动的相对地址,汇编产生的目标文件.OBJ 还不能在计算机上运行,需要用连接程序 LINK.EXE 把目标文件.OBJ 文件转换为可执行文件 XXX.EXE 文件.

双击 MASM 软件中的 LINK.Exe 文件框,在命令行后键入目标文件名 XXX.OBJ (如果源程序,目标文件与 MASM 软件在同一路径下,可以只键入文件名,而不要后缀),如以下界面:



```
C:\ [非使用中 LINK]
The COMPAQ Personal Computer Linker
Version 2.40 (C)Copyright Compaq Computer Corporation 1982, 1986
(C)Copyright Microsoft Corp. 1981, 1986

Object Modules [1.OBJ]: 1
Run File [1.EXE]:
List File [NUL.MAP]:
Libraries [1.LIB]:
Warning: no stack segment
-
```

图 1-3 LINK 连接 1.OBJ 文件界面

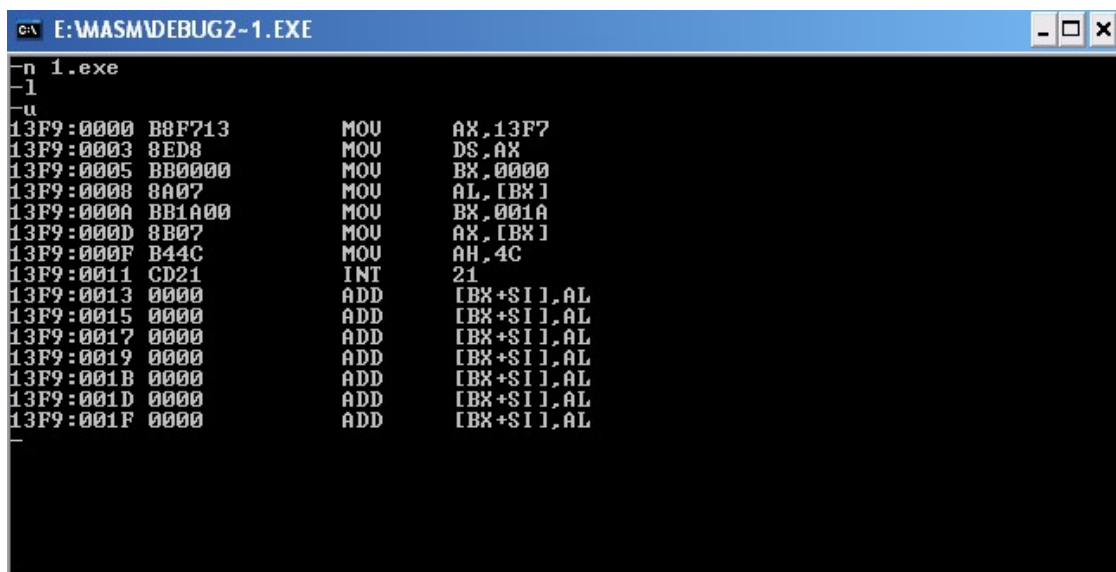
如果连接没有错误,就会产生一个 XXX.EXE 的可执行文件,如果.OBJ 文件有错误,连接时会指出错误的原因.对于无堆栈警告(warning: no stack segment)的提示,可以不予理睬,它是由于我们在源程序中没有定义堆栈段的原因,对于比较小的程源序和不需要再特别定义堆栈段的源程序,我们可以不定义堆栈段,它并不影响程序的正确执行.反而,如果连接时有其他的错误,则要检查并修改源程序 XXX.ASM,然后再重新汇编 MASM.EXE,连接 link.exe 的步骤,直到得到正确的 XXX.EXE 文件为止.

4. 运行和调试 DEBUG.exe

运行可执行文件，即双击 XXX.EXE 文件框即可，或在 DOS 下运行此程序，
E: \XXX.EXE。

1.2 熟悉、使用 DEBUG 调试工具

以下重点介绍一些调试命令：以 1.asm 源程序为例：



```
C:\ E:\WASM\DEBUG2-1.EXE
-n 1.exe
-l
-u
13F9:0000 B8F713      MOV     AX,13F7
13F9:0003 8ED8        MOV     DS,AX
13F9:0005 BB0000      MOV     BX,0000
13F9:0008 8A07        MOV     AL,[BX]
13F9:000A BB1A00      MOV     BX,001A
13F9:000D 8B07        MOV     AX,[BX]
13F9:000F B44C        MOV     AH,4C
13F9:0011 CD21      INT     21
13F9:0013 0000      ADD     [BX+SI],AL
13F9:0015 0000      ADD     [BX+SI],AL
13F9:0017 0000      ADD     [BX+SI],AL
13F9:0019 0000      ADD     [BX+SI],AL
13F9:001B 0000      ADD     [BX+SI],AL
13F9:001D 0000      ADD     [BX+SI],AL
13F9:001F 0000      ADD     [BX+SI],AL
```

图 1-4 运行和调试 DEBUG 界面

1) **N 命令**: 用于指定进行读写的磁盘上的文件。

命令格式: N[path][filename]

如: -n 1.exe 指定要装载 (Load) 或写入磁盘 (Write) 的文件名。

2) **L 命令**: 将指定的文件装入内存中

命令格式: L 回车键

-l 回车键

3) **反汇编命令 U**

就是将存放在制定范围内的目标代码，反汇编成 8086/8088 的汇编指令格式，并按目标代码首地址，目标代码和对应的源指令的格式，在屏幕上显示出来。把目标代码反汇编后，用户可以方便地知道，程序从什么地址开始执行，执行的是哪一条指令，执行到什么地址为止。

命令格式: U[地址范围]

```
-u
13FE:0000 B8FC13      MOV     AX,13FC
13FE:0003 8ED8        MOV     DS,AX
13FE:0005 BB0000      MOV     BX,0000
13FE:0008 8A07        MOV     AL,[BX]
```

13FE:000A BB1A00	MOV	BX, 001A
13FE:000D 8B07	MOV	AX, [BX]
13FE:000F B8004C	MOV	AX, 4C00
13FE:0012 CD21	INT	21
13FE:0014 0000	ADD	[BX+SI], AL
13FE:0016 0000	ADD	[BX+SI], AL
13FE:0018 0000	ADD	[BX+SI], AL
13FE:001A 0000	ADD	[BX+SI], AL
13FE:001C 0000	ADD	[BX+SI], AL
13FE:001E 0000	ADD	[BX+SI], AL

13FE 为 CS 即代码段的段基地址, 0000, 0003, 0005 是偏移地址, B8FC13 等是机器码(由代码段中的二进制机器指令反汇编得到)也就是程序在机器中的代码, MOV AX, 13FC 是助记符, 帮助记忆机器中的指令。(源指令)

4) 运行命令:

T 命令:单步运行命令

以 1. EXE 调试为例, 单步运行二次

13FE:0000 B8FC13	MOV	AX, 13FC
13FE:0003 8ED8	MOV	DS, AX
13FE:0005 BB0000	MOV	BX, 0000
13FE:0008 8A07	MOV	AL, [BX]
13FE:000A BB1A00	MOV	BX, 001A
13FE:000D 8B07	MOV	AX, [BX]
13FE:000F B8004C	MOV	AX, 4C00
13FE:0012 CD21	INT	21
13FE:0014 0000	ADD	[BX+SI], AL
13FE:0016 0000	ADD	[BX+SI], AL
13FE:0018 0000	ADD	[BX+SI], AL
13FE:001A 0000	ADD	[BX+SI], AL
13FE:001C 0000	ADD	[BX+SI], AL
13FE:001E 0000	ADD	[BX+SI], AL

-t

AX=13FC BX=0000 CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13EC ES=13EC SS=13FC CS=13FE IP=0003 NV UP EI PL NZ NA PO NC
13FE:0003 8ED8 MOV DS, AX

-t

AX=13FC BX=0000 CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=0005 NV UP EI PL NZ NA PO NC
13FE:0005 BB0000 MOV BX, 0000

可以看到机器随机分配给该程序的数据段的段基地址被 AX 赋予给 DX。

该程序的代码段的段基地址为 13FE, 数据段的段基地址为 13FC 。

单步执行命令 T 执行一条命令, 并显示 CPU 中寄存器中的内容和要执行的下一条命令, 大家可以看到 AX, BX, CX 等寄存器的内容, T 命令可以跟踪程序中的每一条指令的执行情况。

P 命令:也是单步运行命令。但是 P 命令对于每一条指令语句都是一次执行完成。比如 CALL、LOOP 和 DOS 功能调用的 INT n 等指令语句 T 命令能在程序的执行中, 跟随 IP 指示的地址, 跟踪指令的执行, 而 P 命令则是按指令语句, 连续执行的。在调试中, 用户可根据需要, 选择不同的运行程序命令, 以适应调试的要求。

5) G (Go)命令:连续执行内存中的程序, 还可以在程序中设置断点, 逐段地执行程序, 以便一段一段地对程序进行调试。

命令格式: G[=address[address[address...]]

其中第一个参数=address, 规定了执行的起始地址, 即以 CS 的内容为段地址, 以等号后面的地址为偏移地址, 在输入时, 等号是不可缺少的, 若不输入起始地址, 则以 CS: IP 为起始地址, 后面的地址参数是断点地址。如果在 G 命令中没有设置断点, 或设有断点但程序在执行中未能到达断点处, 这时程序将一直运行, 直至结束。在结束时显示提示信息“Program terminated normally”。

在 G 命令中可以设置断点, 格式为 G=0000 000D

```
13FE:000D 8B07      MOV     AX, [BX]
13FE:000F B8004C     MOV     AX, 4C00
13FE:0012 CD21      INT     21
```

```
AX=1323 BX=0000 CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=000A NV UP EI PL NZ NA PO NC
13FE:000A BB1A00      MOV     BX, 001A
```

- 可以看到 AL 中的数为 23

再设置一次断点 G=0000 000F

-g=0000 000f

```
AX=3334 BX=001A CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=000F NV UP EI PL NZ NA PO NC
13FE:000F B8004C     MOV     AX, 4C00
```

-

可以看到 AX=3334

6) D (Dump)命令:显示指定范围(range)内的内存单元的内容。

其中, 参数范围(range)有以下两种表示方式:

a) 第一种表示方式: Addr1 Addr2

这里 Addr1 和 Addr2 分别代表待显示内存单元的首地址和末地址。

2) 第二种表示方式: Addr1 L Value

这里表示显示从地址 Addr1 开始、长度为 Value 个字节的内存单元。例如下面两条命令是等效的。

— D 200 2FF

—D 200 L 100

显示内容分为三部分。最左边是本行内存单元首地址(XXXX: XXXX)，第二部分是十六进制形式显示的相继各字节单元的内容，前后八个单元间用符号“—”隔开。第三部分是本行显示的十六进制值所对应的 ASCII 字符。如果某十六进制值的 ASCII 字符是不可显示的，便以“.”代替。注意，每行只显示 16 个单元的内容，而且每行的首地址都是

16 的整倍数，或者说，每行的首地址都是以 16 为边界的。

对于 D 命令，又有两种简化格式，如：

—D 200

显示从 DS: 0200 单元开始的 80H 个单元的内容，即在只给定第一个地址的命令中，约定隐含 L 80 参数。

—D

除第一次从 DS: 100 开始，显示 80H 个单元的内容外，以后都继前次地址之后，依次显示 80H 个单元的内容。

—d

```
13FE:0000  B8 FC 13 8E D8 BB 00 00-8A 07 BB 1A 00 8B 07 B8  .....
13FE:0010  00 4C CD 21 00 00 00 00-00 00 00 00 00 00 00 00  .L.!.
13FE:0020  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FE:0030  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FE:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FE:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FE:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FE:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
```

我们从内存单元偏移地址[0000]单元开始，

—d0

```
13FC:0000  23 35 23 00 35 00 22 34-34 34 22 34 34 34 22 34  #5#.5."444"444"4
13FC:0010  34 34 22 34 34 34 33 34-41 42 34 33 42 41 00 00  44"44434AB43BA..
13FC:0020  B8 FC 13 8E D8 BB 00 00-8A 07 BB 1A 00 8B 07 B8  .....
13FC:0030  00 4C CD 21 00 00 00 00-00 00 00 00 00 00 00 00  .L.!.
13FC:0040  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FC:0050  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FC:0060  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
13FC:0070  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
```

—可以看到从偏移地址[0000]单元开始显示内存单元的内容。

7) R (Register)命令

格式: R [register_name]

功能: 显示 CPU 中的一个或所有 16 位寄存器(包括标志寄存器)的内容。对于标志寄存器来说，显示的是各标志位的状态(置位 / 复位)。

—r

```
AX=3334 BX=001A CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=000F NV UP EI PL NZ NA PO NC
13FE:000F B8004C      MOV     AX,4C00
```

—命令 Rregister_name 不但能显示出该寄存器的当前值，而且还显示提示符“:”。这时

只要输入新的值，便可更新该寄存器的内容。若只按回车键，则寄存器的值将保持不变。例如命令：

```
-r
AX=3334 BX=001A CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=000F NV UP EI PL NZ NA PO NC
13FE:000F B8004C      MOV     AX,4C00

-rax
AX 3334
:6677

-r
AX=6677 BX=001A CX=0034 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13FC ES=13EC SS=13FC CS=13FE IP=000F NV UP EI PL NZ NA PO NC
13FE:000F B8004C      MOV     AX,4C00
```

RDS, RCS 命令可以修改当前段的地址。同学们下来可以试以下。

8)-F 是标志寄存器的名字。命令 RF 将显示各状态位的当前状态，其后显示提示符“—”，这时只要输入合法的标志(对顺序无要求)，便可修改标志位状态，若只按回车键，则标志位的状态将保持不变。如命令：

```
— RF
NV UP DI NG NZ AC PE NC—PL EI CY
显示了 F 中的各状态位，其后又通过输入 PL EI CY，修改了对应的标志位。

-rf
NV UP EI PL NZ NA PO NC —pl ei cy

-rf
NV UP EI PL NZ NA PO CY —
```

9) E (Enter)命令

格式：E address [list]

功能：用键入的字节(Byte)值或替换值(字节)列表修改指定内存单元的内容参数，address 表示待修改的内存单元的首地址，list 是一个选择项，它表示替换值(字节)列表。

1) E address

在该命令格式中，没有给出替换值。DEBUG 自动显示起始地址和它的内容，并等待用户键入替换值。用户这时可选择按空格键(Space)、横杠键(—)或回车键(Enter)，以实现不同的操作：

按回车键(Enter)——结束 E 命令。

按空格键(Space)——自动显示下一个内存单元的内容，并等待键入替换值。按照这种方式，一直进行下去，直到按回车键(Enter)，才结束 E 命令。

按横杠键(—)——自动显示前一个内存单元的内容，并等待键入替换值。按照这种方式，一直进行下去，直到按回车键，才结束 E 命令。

在上述两种修改方式中，如果在未键入替换值的情况下就按回车键，这时该单元的内容保持不变并结束 E 命令；如果只按空格键或横杠键，不键入替换值，则只显示内存单元的内容。

```
-e
```



```

^ Error
-e0
13FC:0000 23.    35.    23.    00.    35.    00.    22.    34.
13FC:0008 34.    34.    22.    34.    34.    34.    22.    34.
13FC:0010 34.    34.    22.    34.    34.    34.    33.    34.
13FC:0018 41.    42.    34.    33.    42.    41.    00.    00.
13FC:0020 B8.    FC.    13.    8E.    D8.    BB.    00.    00.
13FC:0028 8A.

-e0
13FC:0000 23.    35.67  23.34  00.    35.    00.    22.    34.
13FC:0008 34.    34.    22.

-e0
13FC:0000 23.    67.    34.    00.    35.    00.    22.    34.
13FC:0008 34.    34.    22.    34.    34.    34.    22.    34.
13FC:0010 34.    34.    22.    34.    34.    34.    33.    34.
13FC:0018 41.    42.    34.    33.    42.    41.    00.    00.

. -
-e0003    显示从[0003]开始的内容
13FC:0003 00.    35.    00.    22.    34.
13FC:0008 34.    34.    22.    34.    34.    34.    22.    34.
13FC:0010 34.    34.    22.    34.    34.    34.

```

10) F (Fill) 命令

格式: F range list

功能: 用 list 所表示的字节值对指定范围(range)的内存单元进行填充在命令中, 若指定范围的内存单元数多于 list 中的填充字节数, 则 F 命令将反复使用填充字节, 直到填满指定范围中的每个单元; 若内存单元数少于填充字节数, 则 F 命令将只使用 list 中的前面若干个字节值, 填充各个内存单元。

```

-f ds:0003 L 10 6
-e0
13FC:0000 23.    67.    34.    06.    06.    06.    06.    06.
13FC:0008 06.    06.    06.    06.    06.    06.    06.    06.
13FC:0010 06.    06.    06.    34.    34.    34.    33.    34.
13FC:0018 41.    42.    34.    33.    42.    41.    00.

```

11) A(Assemble)命令

格式: A [address]

功能: 接收从键盘键入的 8086 / 8088 指令, 将其汇编成目标代码, 并存放在内存单元中。

参数 address 是指定存放目标代码的起始地址, 其后指令的起始地址将由系统自动依次

安排。

一 A CS: 100 ; 从 CS: 100H 开始, 编写程序

1409:001C 0000 ADD [BX+SI], AL

1409:001E 0000 ADD [BX+SI], AL

-a

1409:0000

-a cs:000a

1409:000A mov bx, 0009

1409:000D

这时我们再反汇编一下, 可以看到, 程序发生了变化。

-u0

1409:0000 B80714 MOV AX, 1407

1409:0003 8ED8 MOV DS, AX

1409:0005 BB0000 MOV BX, 0000

1409:0008 8A07 MOV AL, [BX]

1409:000A BB0900 MOV BX, 0009

1409:000D 8B07 MOV AX, [BX]

1409:000F B8004C MOV AX, 4C00

1409:0012 CD21 INT 21

A 命令只是在 DEBUG 下, 汇编程序, 不能存盘, 临时修改看结果。

12) Q 命令: 结束 DEBUG。

至此, 通过调试示例程序, 学习了 DEBUG 调试命令, 并了解了 8088 汇编语言的段结构、常用的指令与伪指令、存储空间的分配, 调试程序的方法, 但这还只是简单的程序, 调试复杂的程序比此例要复杂的多, 还需要我们在学习过程中, 勤动手, 多动脑, 加强练习, 在实践中进步。

1.3 汇编语言程序上机操作和调试训练

一。上机目的:

了解并逐步熟悉汇编语言的编辑方法及特点。

复习 8088 汇编语言的段结构、常用的指令与伪指令、存储空间的分配等。

掌握汇编语言的编辑、汇编及连接的过程。

了解并逐步掌握运用 DEBUG 进行调试汇编语言程序。

二。实验内容:

运用 8086 汇编语言, 编辑多字节非压缩型 BCD 数除法的简单程序, 文件名取为*.ASM。

运用 MASM . EXE 文件进行汇编, 修改程序中的各种语法错误, 直至正确, 形成*.OBJ 文件。

运用 LINK. EXE 文件进行连接, 形成*.EXE 文件。

仔细阅读和体会 DEBUG 调试方法, 掌握各种命令的使用方法。

运用 DEBUG. EXE 文件进行调试, 使用单步执行命令—T 两次, 观察寄存器中内容的变化,

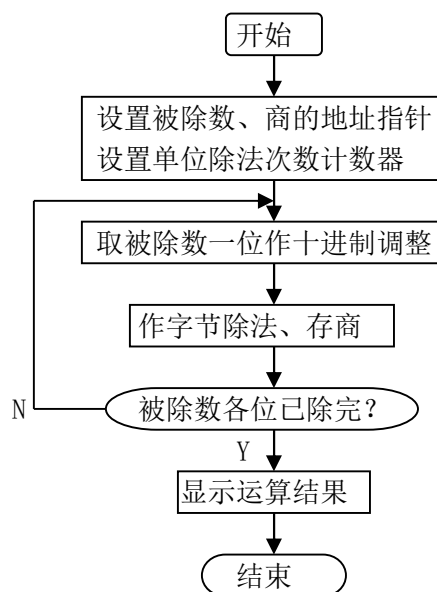
使用察看存储器数据段命令—D，观察存储器数据段内数值。

再使用连续执行命令—G，执行程序，检查结果是否正确，若不正确可使用 DEBUG 的设置断点，单步执行等功能发现错误所在并加以改正。

程序清单：

多字节非压缩型 BCD 数除法 (96875/5=19375)

```
DATA    SEGMENT
A        DB  9, 6, 8, 7, 5
B        DB  5
C        DB  5DUP (0)
N        EQU 5
DATA    ENDS
CODE    SEGMENT
        ASSUME CS: CODE; DS: DATA; ES: DATA,
START   MOV AX, DATA
        MOV DS, AX
        MOV ES, AX
        CLD
        LEA SI, A
        LEB DI, C
        MOV CX, N
        MOV AH, 0
LP1:    LODSB
        AAD
        DIV B
        STOSB
        LOOP LP1
        MOV CX, N
        LEA DI, C
LP2:    MOV DL, [DI]
        MOV AH, 2
        INT 21H
        DEC DI
        LOOP LP2
        MOV AH, 4CH
        INT 21H
CODE:   ENDS
        END START
```



三。实验报告要求：

- 1) 写出本次试验报告的实验目的，内容，正确的程序清单，适当的中文注释以及程序框图。
- 2) 指出本实验给出的源程序错误，并加以改正。
- 3) 给出实验结果。
- 4) 调试的心得体会。

实验二 设计汇编语言程序

要设计汇编语言程序，我们先复习一些基本的指令。

1. 加法指令

(1) ADD——加法指令

格式：ADD 目的操作数，源操作数

功能：目的操作数 \leftarrow 目的操作数+源操作数

(2) ADC——带进位的加法指令

格式：ADC 目的操作数，源操作数

功能：目的操作数 \leftarrow 目的操作数+源操作数+CF (CF=1，有进位)

(3) INC——增量指令

格式：INC 目的操作数

功能：目的操作数 \leftarrow 目的操作数+1

2. 减法指令

(1) SUB——减法指令

格式：SUB 目的操作数，源操作数

功能：目的操作数 \leftarrow 目的操作数-源操作数

(2) SBB——带借位的减法指令

格式：SBB 目的操作数，源操作数

功能：目的操作数 \leftarrow 目的操作数-源操作数-CF

(3) DEC——减量指令

格式：DEC 目的操作数

功能：目的操作数 \leftarrow 目的操作数-1

(4) NEG 求补指令

格式：NEG 目的操作数

功能：目的操作数 \leftarrow 0-目的操作数

3. 乘法和除法指令

(1) MUL——无符号数乘法

格式：MUL 源操作数

功能：8 位源操作数时：AX \leftarrow (AL) \times 源操作数

16 位源操作数时：DX, AX \leftarrow (AX) \times 源操作数

32 位源操作数时：EDX, EAX \leftarrow (EAX) \times 源操作数

(2) IMUL 有符号数乘法

格式 1：IMUL 源操作数

功能：8 位源操作数时：AX \leftarrow (AL) \times 源操作数

16 位源操作数时: $DX, AX \leftarrow (AX) \times \text{源操作数}$

32 位源操作数时: $EDX, EAX \leftarrow (EAX) \times \text{源操作数}$

格式 2: IMUL 目的操作数, 源操作数

功能: 目的操作数 \leftarrow 目的操作数 \times 源操作数

格式 3: IMUL 目的操作数, 源操作数 1, 源操作数 2

功能: 目的操作数 \leftarrow 源操作数 1 \times 源操作数 2

(3) DIV —— 无符号除法

格式: DIV 源操作数

功能: 8 位源操作数时: $(AX) \div \text{源操作数}$, $AL \leftarrow$ 商, $AH \leftarrow$ 余数

16 位源操作数时: $(DX, AX) \div \text{源操作数}$, $AX \leftarrow$ 商, $DX \leftarrow$ 余数

32 位源操作数时: $(EDX, EAX) \div \text{源操作数}$, $EAX \leftarrow$ 商, $EDX \leftarrow$ 余数

(4) IDIV —— 有符号除法

格式: IDIV 源操作数

功能: 8 位源操作数时: $(AX) \div \text{源操作数}$, AL 商, AH 余数

16 位源操作数时: $(DX, AX) \div \text{源操作数}$, $AX \leftarrow$ 商, $DX \leftarrow$ 余数

32 位源操作数时: $(EDX, EAX) \div \text{源操作数}$, $EAX \leftarrow$ 商, $EDX \leftarrow$ 余数

实验 2.1 简单程序设计

实验目的

理解各种指令的功能。

进一步学习程序的调试。

使用以上指令, 我们可以进行一些表达式的计算。

练习 1. 实验题目: 编程计算下列表达式: $A=90$, $B=-70$, $C=5$, Y 均为有符号数,

计算 $Y=2 \times (A+B) + (A \times C) \div 5$

要编写一些稍微复杂的程序, 我们会遇到一段程序被反复执行, 这样, 我们会用到 LOOP 指令。使用该指令时, 需在 CX 中装入循环次数。

练习 2. 码转换程序设计

编制程序, 把十进制数 15786 转化成二进制数。

提示: $15786=1 \times 10 \times 10 \times 10 \times 10+5 \times 10^3+7 \times 10^2+8 \times 10+6$

循环 CX=5

实验要求: 1. 绘出练习 1 和 2 的程序流程图。

2 编写完整的程序, 上机调试。

3 使用 DEBUG 调试命令, 查看中间结果, 并查看最终结果。

大多数的程序, 都有人机对话的过程。也就是说, 我们从键盘上输入程序所需要的控制信息和数据, 并把程序的运行结果和运行状态显示出来。这就涉及到字符及字符串的输入输出。

1. 字符的输出

(1) 输出单个字符

DL \leftarrow 待输出字符的 ASCII 码

AH = 02

INT 21H

输入的字符放在 AL 中，并显示在屏幕上。

(2) 输出一个字符串

DS: DX ← 待输出字符串的首地址

AH=09H

Int 21H

2. 字符的输入

(1) 输入单个字符

AH=01

INT 21H

输入的字符放在 AL 中

(2) AH=07

INT 21H

输入的字符放在 AL 中，不会显示在屏幕上（无回显）。

3. 字符串的输入

DS: DX ← 输入缓冲区首地址

AH=0AH

INT 21H

一行字符以回车键作为结束的标志。假如：一行最多不超过 250 个字符（不含回车键），输入缓冲区格式如下：BUFFER DB 250,?,250 DUP(?)

缓冲区由 3 个部分组成：

第一字节：输入字符存放区的大小。

第二字节：初始状态为空。从服务程序返回后，由服务程序填入实际的字符个数，不包括回车。

第三字节之后：输入字符存放区，存放输入的字符和回车。

练习 3. 编制程序，从键盘输入最多 5 个数，求他们的和，存入 SUM。

实验要求：1. 绘出程序流程图。

2 编写完整的程序，上机调试。

3 要求编制的程序最终求和的结果要显示在屏幕上。

4. 写实验报告中程序要加中文注释。

实验 2.2 汇编语言程序设计

---- 循环结构程序

几乎所有的应用程序都离不开循环结构。

循环结构一般有以下 4 个部分组成。

1 初始化部分：为循环做准备，如累加器清零，设置地址指针和计数器的初始值。

2. 工作部分：实现循环的基本操作，也就是需要重复执行的一段程序。

3. 修改部分：修改指针，计数器的值，为下一个循环做准备。

4. 控制部分：判断循环条件，结束循环或继续循环。

练习 1. 用“冒泡”法对一组数 300, 250, 280, 240, 260, 按从小到大的顺序排列。

提示：用冒泡的方法对一组数据元素排序，它的基本方法是：将相邻的两个元素通过比较进行排序，通过多次，多遍的邻元素排序，实现整个一组数的排序。

对于 5 (N) 个元素，整个排序通过 4 遍 (=N-1) 邻元素排序完成。每一遍的排序由若干次邻元素的排序组成。

4 遍排序中，邻元素的排序依次为 4, 3, 2, 1 遍。完成第一遍排序后，最大数沉底，已经到达它应占据的位置，不需要参加下一遍的排序。

外循环的次数为 CX=4 次(N-1)，内循环的次数为 4, 3, 2, 1(DEC CX)

排序遍数	本遍排序前	第一次排序后	第二次排序后	第三次排序后	第四次排序后
1	300, 250, 280, 240, 260	250, 300, 280, 240, 260	250, 280, 300, 240, 260	250, 280, 240, 300, 260	250, 280, 240, 260, 300
2	250, 280, 240, 260, 300	250, 280, 240, 260, 300	250, 240, 280, 260, 300	250, 240, 260, 280, 300	
3	250, 240, 260, 280, 300	250, 240, 260, 280, 300	240, 250, 260, 280, 300		
4	240, 250, 260, 280, 300	240, 250, 260, 280, 300			

实验要求：

1. 编制程序，从键盘输入 300, 250, 280, 240, 260 这五个数，并思考如何输入任意五个数，五个数可以有一位数，二位数，三位数，四位数，五位数，混合输入比较大小；
2. 对这组数用冒泡法进行排序，并输出原始数据及排序后数据，两两数据之间用空格分隔；
3. 利用 DEBUG 调试工具，用 D0 命令，查看排序前后，内存数据的变化，以及会用调试命令查看程序运算中寄存器中的值；
4. 去掉最大和最小的两个值，求出其余值的平均值，并输出最大值，最小值和平均值；
5. 用压栈 PUSH 和出栈 POP 指令“先进后出”的特点，设计算法将平均值按位逐个输出(即输出 263)；
6. 用移位指令将平均值以二进制串的形式输出。
7. 设计程序要有模块化的思想，用子程序实现不同的功能；
8. 所有数据输出前要用字符串的输出指令，进行输出提示(如：zui da zhi shi : 300 等)，所有数据结果能清晰地显示在电脑屏幕上。

实验报告：

整理出运行正确的程序清单（加以注释）。并给出程序流程图。

第二部分 接口程序设计

第三次实验 8253 定时器/计数器与接口实验

-----乐曲程序

计算机是如何产生音乐和声响的呢？原来在计算机中有一个可编程时间间隔定时器 8253，它能根据程序提供的计数值和工作模式，产生各种形状和各种频率的计数/定时脉冲，提供给系统的各个部件使用。它还可以产生不同频率的脉冲作为扬声器的声源。

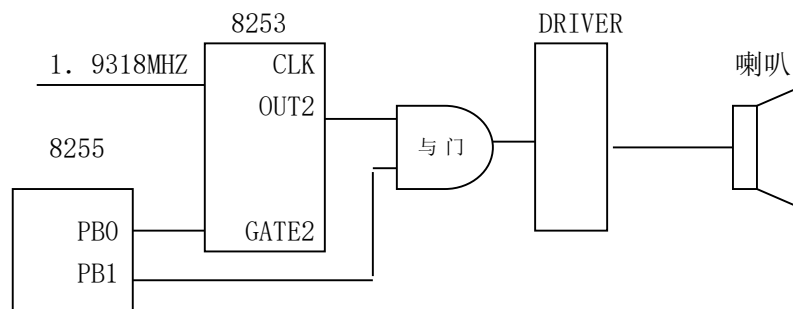
在 8253 定时器内部有 3 个独立工作的计数器：Counter0, Counter1, Counter2，每个计数器都分配有一个端口地址，分别是 40H，41H，42H。8253 内部还有一个公用的控制寄存器，端口地址为 43H，端口地址输入到 8253 的 CS，A1，A0 端，分别对三个计数器和控制器寻址。

对 8253 编程时，先要设定控制字，以选择计数器，确定工作模式和计数值的格式，每个计数器由三个引脚与外部联系，CLK 为时钟输入端，GATE 为门控制信号输入端，OUT 为计数/定时信号输出端。每个计数器是以倒计数的方式计数，也就是说，从计数初值开始逐次减 1，直到减为 0 为止。

端口地址与计数器的关系：

40H	选中计数器 0：作为定时器为系统时钟提供计时基准
41H	选中计数器 1：作为定时器使用
42H	选中计数器 2：用来控制扬声器发声
43H	公用的控制器：分别对三个寄存器和控制器寻址

那么，如何触动扬声器发出声音呢？原来 PC 机的主音箱上装有一只小喇叭，由定时器 8253 和并行接口芯片 8255（可编程外围接口芯片）控制其发声，8255 包括三个 8 位寄存器，两个用于输入功能，一个用于输出功能。输入寄存器分配的 I/O 端口地址为 60H 和 62H，输出寄存器分配的 I/O 端口地址为 61H。8253 定时器计数器 2 连接到扬声器，其电路如下：



装入计数器 2 的计数初值为 533H（ $1.9318\text{MHz}/896\text{Hz}=1331=533\text{H}$ ），这样，得到的控制

字为 10110110B=0B6H，即选中计数器 2，读/写，工作方式三（方波发声器），二进制。

计数器 2 的初始化程序为：

```
MOV    AL,0B6H
OUT    43H,AL
```

1) 计算计数值程序段：

```
MOV    DI,    给定频率
MOV    DX,    12H
MOV    AX,    34DCH
DIV    DI
AX 中即为计数值。
```

2) 打开扬声器发声，8255PB0，PBI 送出高电平：

```
IN     AL ,    61H
OR     AL ,    3
OUT    61H ,    AL
```

3) 关闭扬声器，停止发声：

```
IN     AL ,    61H
AND    AL ,    0FCH
OUT    61H,    AL
```

两只老虎的音频表定义在数据段中，如下：

```
STACK SEGMENT
        DW 100 DUP (?)
STACK ENDS
```

```
DATA SEGMENT
```

```
MUSIC  DW 2 DUP (262, 294, 330, 262)           ; 频率表, 1231;1231
        DW 2 DUP (330, 349, 392)               ; 345;345;
        DW 2 DUP (392, 440, 392, 349, 330, 262) ; 565431;565431
        dw 2 dup (294, 196, 262), 0             ; 251;251, 0 表示结束
```

```
TIME   DW 10 DUP (250*50), 500*50, 250*50, 250*50, 500*50 ; 节拍表
        DW 2  DUP (120*50, 120*50, 120*50, 120*50, 250*50, 250*50)
        DW 2  DUP (250*50, 250*50, 500*50)
```

```
N      EQU 32                                ; 32 个音符
DATA ENDS
```

二. 实验目的：

学习 8253 计数器 2 输出方波信号用以驱动扬声器发声的原理，通过程序设置计数器

2 的输出波形的频率和延续时间，控制扬声器的音调和发生长短。

三. 实验内容:

1. 设计程序让微机演奏一段两只老虎的乐曲。
2. 思考如何让 PC 机演奏一遍，两遍以及数遍这段两只老虎的乐曲，并修改程序实现。

四. 实验报告:

整理出运行正确的源程序清单（加以中文注释），并给出详细程序流程图。

第四次实验 8253 定时器/计数器接口实验

-----键盘控制发声程序

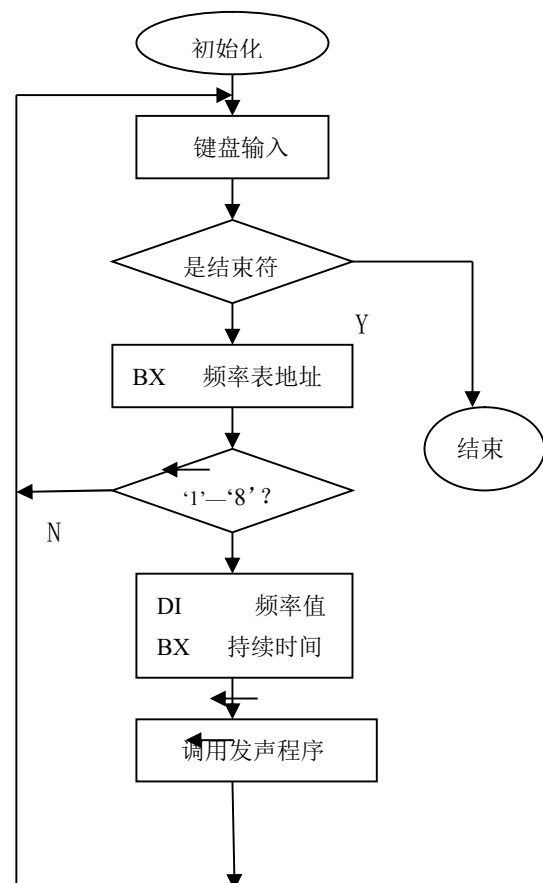
音符和频率之间有一定的对应关系，如果计算机键盘上的数字键和音符，频率也形成一种对应关系，则可以通过键盘控制扬声器发出各种音符声音，这是计算机键盘就变成了电子钢琴键盘，就可以用它弹奏出简单的音乐。我们让数字键 1-8 对应一个音阶的八个音符。

弹奏时，对应乐谱上的 1（C 音符），按下数字键“1”（ASCII 码为 31H），程序的工作就是要接受这个键，并将频率表中与它对应的频率值 262Hz 送给发声程序，以发出 1（C 音符）的音调。按下数字键“2”（ASCII 码为 32H），程序就将 292Hz 的频率值送给发声程序，从而发出 2（D 的音调）。按下数字键“8”，将发出比第一个 C 高八度的音调，音符的频率表作为数据定义在数据段中。流程图如下：

编程提示:

1. 数据段可以定义如下:

```
DATA    DW    262    ;1 (C)
          DW    294    ;2 (D)
          DW    330    ;3 (E)
          DW    349    ;4 (F)
          DW    392    ;5 (G)
          DW    440    ;6 (A)
          DW    494    ;7 (B)
          DW    523    ;8 (C)
DATA    ENDS
```



2. 从键盘输入一个键值，可以用 BIOS 16H 中断，AL 中回送的是键的 ASCII 码，AH 中回送的是键的扫描码。也可以用 DOS 的键盘输入功能。

```
MOV AH, 07    ; 功能号 AH=07
INT 21H       ; DOS 21 号中断
```

一. 实验内容：利用 DOS 的键盘管理功能，将 PC 机变为一个具有简单功能的电子琴。

二. 实验报告：整理出运行正确的源程序清单（加以中文注释）。

附录 A 标准 ASCII 码字符表

ASCII 码	字符	ASCII 码	字符	ASCII 码	字符	ASCII 码	字符
00H	NUL	20H	SP	40H	@	60H	`
01H	SOH	21H	!	41H	A	61H	a
02H	STX	22H	"	42H	B	62H	b
03H	ETX	23H	#	43H	C	63H	c
04H	EOT	24H	\$	44H	D	64H	d
05H	ENQ	25H	%	45H	E	65H	e
06H	ACK	26H	&	46H	F	66H	f
07H	BEL	27H	'	47H	G	67H	g
08H	BS	28H	(48H	H	68H	h
09H	HT	29H)	49H	I	69H	i
0AH	LF	2AH	*	4AH	J	6AH	j
0BH	VT	2BH	+	4BH	K	6BH	k
0CH	FF	2CH	,	4CH	L	6CH	l
0DH	CR	2DH	-	4DH	M	6DH	m
0EH	SO	2EH	.	4EH	N	6EH	n
0FH	SI	2FH	/	4FH	O	6FH	o
10H	DLE	30H	0	50H	P	70H	p
11H	DC1	31H	1	51H	Q	71H	q
12H	DC2	32H	2	52H	R	72H	r
13H	DC3	33H	3	53H	S	73H	s
14H	DC4	34H	4	54H	T	74H	t
15H	NAK	35H	5	55H	U	75H	u
16H	SYN	36H	6	56H	V	76H	v
17H	ETB	37H	7	57H	W	77H	w
18H	CAN	38H	8	58H	X	78H	x
19H	EM	39H	9	59H	Y	79H	y
1AH	SUB	3AH	:	5AH	Z	7AH	z
1BH	ESC	3BH	;	5BH	[7BH	{
1CH	FS	3CH	<	5CH	\	7CH	
1DH	GS	3DH	=	5DH]	7DH	}
1EH	RS	3EH	>	5EH	^	7EH	~
1FH	US	3FH	?	5FH	_	7FH	DEL

附录 B 键盘扫描码表

键	接通 扫描码	断开 扫描码	键	接通 扫描码	断开 扫描码	键	接通 扫描码	断开 扫描码
ESC	01	81	U	16	96	右 Alt	E0 38	E0 B8
F1	3B	BB	I	17	97	右 Windows	E0 5C	E0 DC
F2	3C	BC	O	18	98	Application	E0 5D	E0 DD
F3	3D	BD	P	19	99	右 Ctrl	E0 1D	E0 9D
F4	3E	BE	[{	1A	9A	Print Screen Sys Rq	E0 37	E0 B7
F5	3F	BF	}]	1B	9B	Scroll Lock	46	C6
F6	40	C0	Enter	1C	9C	Pause Break	E1 1D 45 E1 9D C5	-
F7	41	C1	Caps Lock	3A	BA	Insert	E0 52	E0 D2
F8	42	C2	A	1E	9E	Home	E0 47	E0 C7
F9	43	C3	S	1F	9F	Page Up	E0 49	E0 C9
F10	44	C4	D	20	A0	Delete	E0 53	E0 D3
F11	57	D7	F	21	A1	End	E0 4F	E0 CF
F12	58	D8	G	22	A2	Page Down	E0 51	E0 D1
`~	29	A9	H	23	A3	↑	E0 48	E0 C8
1!	02	82	J	24	A4	←	E0 4B	E0 CB
2@	03	83	K	25	A5	↓	E0 50	E0 D0
3#	04	84	L	26	A6	→	E0 4D	E0 CD
4\$	05	85	;;	27	A7	Num Lock	45	C5
5%	06	86	"'	28	A8	÷(/)	E0 35	E0 B5
6^	07	87	左 Shift	2A	AA	×(*)	37	B7
7&	08	88	Z	2C	AC	-	4A	CA
8*	09	89	X	2D	AD	Home 7	47	C7
9(0A	8A	C	2E	AE	↑ 8	48	C8
0)	0B	8B	V	2F	AF	PaUp 9	49	C9
-_	0C	8C	B	30	B0	← 4	4B	CB
=+	0D	8D	N	31	B1	5	4C	CC
\	2B	AB	M	32	B2	→ 6	4D	CD
Back Space	0E	8E	,<	33	B3	+	4E	CE
Tab	0F	8F	.>	34	B4	End 1	4F	CF
Q	10	90	/?	35	B5	↓ 2	50	D0
W	11	91	右 Shift	36	B6	PgDn 3	51	D1
E	12	92	左 Ctrl	1D	9D	Ins 0	52	D2
R	13	93	左 Windows	E0 5B	E0 DB	Del.	53	D3
T	14	94	左 Alt	38	B8	Enter (数字键盘)	E0 1C	E0 9C
Y	15	95	Space	39	B9			

附录 C 80×86 指令系统

1. 指令符号说明

符 号	说 明
R8/r16/r32	一个通用 8/16/32 位寄存器
reg	通用寄存器
seg	段寄存器
mm	整数 MMX 寄存器:MMX0~MMX7
xmm	128 位的浮点 SIMD 寄存器:XMM0~XMM7
ac	AL/AX/EAX 累加寄存器
m8/m16/m32/m64/m128	一个 8/16/32/64/128 位存储器操作数
mem	一个 m8 或 m16 或 m32 存储器操作数
I8/I16/I32	一个 8/16/32 位立即操作数
imm	一个 I8 或 I16 或 I32 位立即操作数
dst	目的操作数
src	源操作数
label	标号
m16&32	16 位段限和 32 位段基地址
d8/d16/d32	8/16/32 位偏移地址
EA	指令内产生的有效地址

2. 16/32 位 80×86 基本指令

助 记 符	功 能	备 注
AAA	把 AL 中的和调整到非压缩的 BCD 格式	
AAD	把 AX 中的非压缩 BCD 码扩展成二进制数	
AAM	把 AX 中的积调整为非压缩的 BCD 码	
AAS	把 AL 中的差调整为非压缩的 BCD 码	
ADC reg, mem/imm/reg mem, reg/imm ac, imm	$(dst) \leftarrow (src) + (dst) + C_f$	
ADD reg, mem/imm/reg mem, reg/imm ac, imm	$(dst) \leftarrow (src) + (dst)$	
AND reg, mem/imm/reg mem, reg/imm ac, imm	$(dst) \leftarrow (src) \wedge (dst)$	
ARPL dst, src	调整选择器的 RPL 字段	286 起, 系统指令
BOUND reg, mem	测数组下标(reg)是否在指定的上下界(mem)之内。在界内, 则往下执行; 不在界内, 产生 INT5	286 起
BSF r16, r16/m16 r32, r32/m32	自右向左扫描 (src), 遇第一个为 1 的位, 则 $ZF \leftarrow 0$, 该位位置装入 reg; 如(src)=0, 则 $ZF \leftarrow 1$	386 起

助 记 符	功 能	备 注
BSR r16, r16/m16 r32, r32/m32	自左向右扫描 (src), 遇第一个为 1 的位, 则 $ZF \leftarrow 0$, 该位位置装入 reg; 如(src)=0, 则 $ZF \leftarrow 1$	386 起
BSWAP r32	(r32)字节次序变反	486 起
BT reg, reg/i8 mem, reg/i8	把由(src)指定的(dst)中的位内容选 CF	386 起
BTC reg, reg/i8 mem, reg/i8	把由(src)指定的(dst)中的位内容选 CF, 并把该位取反	386 起
BTR reg, reg/i8 mem, reg/i8	把由(src)指定的(dst)中的位内容选 CF, 并把该位置 0	386 起
BTS reg, reg/i8 mem, reg/i8	把由(src)指定的(dst)中的位内容选 CF, 并把该位置 1	386 起
CALL reg/mem	段内直接:PUSH (IP 或 EIP), $(IP) \leftarrow (IP) + d16$ 或 $(EIP) \leftarrow (EIP) + d32$ 段内间接:PUSH (IP 或 EIP), $(IP \text{ 或 } EIP) \leftarrow (EA)/reg$ 段内直接:PUSH CS, PUSH(IP 或 EIP), $(CS) \leftarrow dst$ 指定的段地址, $(IP \text{ 或 } EIP) \leftarrow dst$ 指定的偏移地址 段内间接:PUSH CS, PUSH(IP 或 EIP), $(IP \text{ 或 } EIP) \leftarrow (EA)$, $(CS) \leftarrow (EA+2)$ 或 $(EA+4)$	
CBW	(AL)符号扩展到(AH)	
CDQ	(EAX)符号扩展到(EDX)	386 起
CLC	$CF \leftarrow 0$	
CLD	$DF \leftarrow 0$	
CLI	$IF \leftarrow 0$	
CLTS	清除 CR0 中的任务切换标志	386 起, 系统指令
CMC	进位位变反	
CMP reg, reg/mem/imm mem, reg/imm	(dst)-(src), 结果影响标志位	
CMP SB	[SI 或 ESI]-[DI 或 EDI], SI 或 ESI, DI 或 EDI 加 1 或减 1	
CMP SW	[SI 或 ESI]-[DI 或 EDI], SI 或 ESI, DI 或 EDI 加 2 或减 2	
CMP SD	[SI 或 ESI]-[DI 或 EDI], SI 或 ESI, DI 或 EDI 加 4 或减 4	
CMPXCHG reg, reg/mem, reg	(ac)-(dst), 相等: $ZF \leftarrow 1$, $(dst) \leftarrow (src)$ 不相等: $ZF \leftarrow 0$, $(ac) \leftarrow (src)$	486 起
CMPXCHG8B dst	(EDX, EAX)-(dst), 相等: $ZF \leftarrow 1$, $(dst) \leftarrow (EDX, EAX)$ 不相等: $ZF \leftarrow 0$, $(EDX, EAX) \leftarrow (dst)$	586 起
CPUID	(EAX) \leftarrow CPU 识别信息	586 起
CWD	(AX)符号扩展到(DX, AX)	
CWDE	(AX)符号扩展到(EAX)	386 起
DAA	把 AL 中的和调整为压缩的 BCD 格式	
DAS	把 AL 中的差调整为压缩的 BCD 格式	
DEC reg/mem	$(dst) \leftarrow (dst-1)$	
DIV r8/m8 r16/m16	$(AL) \leftarrow (AX)/(src)$ 的商, $(AH) \leftarrow (AX)/(src)$ 的余数 $(AX) \leftarrow (DX, AX)/(src)$ 的商, $(DX) \leftarrow (DX, AX)/(src)$ 的余数	

助 记 符	功 能	备 注
r32/m32	$(EAX) \leftarrow (EDX, EAX) / (src)$ 的商, $(EDX) \leftarrow (EDX, EAX) / (src)$ 的余数	386 起
ENTER I16, I8	建立堆栈帧, I16 为堆栈帧字节数, I8 为堆栈帧层数	386 起
HLT	停机	
IDIV r8/m8 r16/m16 r32/m32	$(AL) \leftarrow (AX) / (src)$ 的商, $(AH) \leftarrow (AX) / (src)$ 的余数 $(AX) \leftarrow (DX, AX) / (src)$ 的商, $(DX) \leftarrow (DX, AX) / (src)$ 的余数 $(EAX) \leftarrow (EDX, EAX) / (src)$ 的商, $(EDX) \leftarrow (EDX, EAX) / (src)$ 的余数	386 起
IMUL r8/m8 r16/m16 r32/m32	$(AL) \leftarrow (AX) * (src)$ $(EAX) \leftarrow (AX) * (src)$ $(EDX, EAX) \leftarrow (EDX, EAX) * (src)$	386 起
IMUL r16/r32, reg/mem	$(r16) \leftarrow (r16) * (scr)$ 或 $(r32) \leftarrow (r32) * (scr)$	286 起
IMUL reg, reg/mem, imm	$(r16) \leftarrow (reg/mem) * imm$ 或 $(r32) \leftarrow (reg/mem) * imm$	286 起
IN ac, I8/DX	$(ac) \leftarrow ((I8))$ 或 (DX)	
INC reg/mem	$(dst) \leftarrow (dst) + 1$	
INSB INSW INSD	$((DI \text{ 或 } EDI) \leftarrow ((DX)), (DI \text{ 或 } EDI) \leftarrow (DI \text{ 或 } EDI) \pm 1$ $((DI \text{ 或 } EDI) \leftarrow ((DX)), (DI \text{ 或 } EDI) \leftarrow (DI \text{ 或 } EDI) \pm 2$ $((DI \text{ 或 } EDI) \leftarrow ((DX)), (DI \text{ 或 } EDI) \leftarrow (DI \text{ 或 } EDI) \pm 3$	286 起
INT I8	PUSH (FLAGS), PUSH (CS), PUSH (IP), $(IP) \leftarrow (i8 * 4)$ $(CS) \leftarrow (I8 * 4 + 2)$ 若 OF=1, 则 PUSH (FLAGS), PUSH (CS), PUSH (IP), $(IP) \leftarrow (10H)$, $(CS) \leftarrow (12H)$	
INVD	使高速缓存无效	486 起, 系统指令
IRET	$(IP) \leftarrow POP()$, $(CS) \leftarrow POP()$, $(FLAGS) \leftarrow POP()$	
IRETD	$(EIP) \leftarrow POP()$, $(CS) \leftarrow POP()$, $(EFLAGS) \leftarrow POP()$	386 起
JZ/JE JNZ/JNE JS JNS JO JNO JP/JPE JNP/JPO JC/JB/JNAE JNC/JNB/JA E JBE/JNA JNBE/JA JL/JNGE JNL/JGE JLE/JNG JNLE/JG	如果 ZF=1, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 ZF=0, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 SF=1, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 SF=0, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 OF=1, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 OF=0, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 PF=1, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 PF=0, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 CF=1, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 CF=0, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $ZF \vee CF=1$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $ZF \vee CF=0$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $SF \oplus OF=1$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $SF \oplus OF=0$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $(SF \oplus OF) \vee ZF=1$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$ 如果 $(SF \oplus OF) \vee ZF=0$, 则 $(IP) \leftarrow (IP) + d8$ 或 $(EIP) \leftarrow (EIP) + d16/d32$	D16/d32 从 386 起

助 记 符	功 能	备 注
JCXZ d8 JECXZ d8/d16/d32	如果(CX)=0, 则(IP) \leftarrow (IP)+d8 如果(ECX)=0, 则(IP) \leftarrow (EIP)+d8/d16/d32	386 起
JMP label JMP mem/reg JMP label JMP mem/reg	段内直接转移, (IP) \leftarrow (IP)+d8/d16, 或(EIP) \leftarrow (EIP)+d8/d32 段内间接转移, (EIP/IP) \leftarrow (EA) 段内直接转移, (EIP/IP) \leftarrow EA, CS \leftarrow label 决定的段基址 段内间接转移, (EIP/IP) \leftarrow (EA), CS \leftarrow (EA+2/4)	
LAHF	(AH) \leftarrow (FLAGS 的低字节)	
LAR reg, mem/reg	取访问权字节	286 起, 系统指令
LDS reg, mem	(reg) \leftarrow (mem), (DS) \leftarrow (mem+2 或 4)	
LEA reg, mem	(reg) \leftarrow EA	
LEAVE	释放堆栈帧	286 起
LES reg, mem	(reg) \leftarrow (mem), (ES) \leftarrow (mem+2 或 4)	
LFS reg, mem	(reg) \leftarrow (mem), (FS) \leftarrow (mem+2 或 4)	386 起
LGDT mem	装入全局描述符表寄存器:(GDTR) \leftarrow (mem)	286 起, 系统指令
LGS reg, mem	(reg) \leftarrow (mem), (GS) \leftarrow (mem+2 或 4)	386 起
LIDT mem	装入中断描述符表寄存器:(IDTR) \leftarrow (mem)	286 起, 系统指令
LLDT reg/mem	装入局部描述符表寄存器:(LDTR) \leftarrow (reg/mem)	286 起, 系统指令
LMSW reg/mem	装入机器状态字 (在 CR0 寄存器中):(MSW) \leftarrow (reg/mem)	286 起, 系统指令
LOCK	插入 LOCK#信号前缀	
LODSB LODSW LODSD	(AL) \leftarrow (SI 或 ESI), (SI 或 ESI) \leftarrow (SI 或 ESI) \pm 1 (AX) \leftarrow (SI 或 ESI), (SI 或 ESI) \leftarrow (SI 或 ESI) \pm 2 (EAX) \leftarrow (SI 或 ESI), (SI 或 ESI) \leftarrow (SI 或 ESI) \pm 4	ESI 自 386 起 自 386 起
LOOP label LOOPZ/LOOPE label LOOPNZ/LOOPNE label	(ECX/CX) \leftarrow (ECX/CX)-1, (ECX/CX) \neq 0 则循环 (ECX/CX) \leftarrow (ECX/CX)-1, (ECX/CX) \neq 0 且 ZF=1 则循环 (ECX/CX) \leftarrow (ECX/CX)-1, (ECX/CX) \neq 0 且 ZF=0 则循环	ECX 自 386 起
LSL reg, reg/mem	取段界限	286 起, 系统指令
LSS reg, mem	(reg) \leftarrow (mem), (SS) \leftarrow (mem+2 或 4)	386 起
LTR reg/mem	装入任务寄存器	286 起, 系统指令
MOV reg, reg/mem/imm mem, reg/imm reg, CR0-CR3 CR0-CR3, reg reg, DR DR, reg reg, SR SR, reg	(reg) \leftarrow (reg/mem/imm) (mem) \leftarrow (reg/imm) (reg) \leftarrow (CR0-CR3) (CR0-CR3) \leftarrow (reg) (reg) \leftarrow (调试寄存器 DR) (DR) \leftarrow (reg) (reg) \leftarrow (段寄存器 SR) (SR) \leftarrow (reg)	386 起, 系统指令 386 起, 系统指令 386 起, 系统指令 386 起, 系统指令
MOVS	((DI 或 EDI) \leftarrow ((SI 或 ESI)), (SI 或 ESI) \leftarrow (SI 或 ESI) \pm 1, (DI 或 EDI) \leftarrow (DI 或 EDI) \pm 1	

助 记 符	功 能	备 注
MOVSW MOVSD	$((DI \text{ 或 } EDI) \leftarrow ((SI \text{ 或 } ESI)),$ $(SI \text{ 或 } ESI) \leftarrow (SI \text{ 或 } ESI) \pm 2, (DI \text{ 或 } EDI) \leftarrow (DI \text{ 或 } EDI) \pm 2$ $((DI \text{ 或 } EDI) \leftarrow ((SI \text{ 或 } ESI)),$ $(SI \text{ 或 } ESI) \leftarrow (SI \text{ 或 } ESI) \pm 4, (DI \text{ 或 } EDI) \leftarrow (DI \text{ 或 } EDI) \pm 4$	386 起
MOVSX reg, reg/mem	$(reg) \leftarrow (reg/mem \text{ 符号扩展})$	386 起
MOVZX reg, reg/mem	$(reg) \leftarrow (reg/mem \text{ 零扩展})$	386 起
MUL reg/mem	$(AX) \leftarrow (AL) * (r8/m8)$ $(DX, AX) \leftarrow (AX) * (r16/m16)$ $(EDX, EAX) \leftarrow (EAX) * (r32/m32)$	386 起
NEG reg/mem	$(reg/mem) \leftarrow (reg/mem)$	
NOP	无操作	
NOT reg/mem	$(reg/mem) \leftarrow (reg/mem \text{ 按位取反})$	
OR reg, reg/mem/imm Mem, reg/imm	$(reg) \leftarrow (reg) \vee (reg/mem/imm)$ $(mem) \leftarrow (mem) \vee (reg/imm)$	
OUT I8, ac DX, ac	$(I8 \text{ 端口}) \leftarrow (ac)$ $((DX)) \leftarrow (ac)$	
OUTSB OUTSW OUTSD	$((DX)) \leftarrow ((SI \text{ 或 } ESI)), (SI \text{ 或 } ESI) \leftarrow (SI \text{ 或 } ESI) \pm 1$ $((DX)) \leftarrow ((SI \text{ 或 } ESI)), (SI \text{ 或 } ESI) \leftarrow (SI \text{ 或 } ESI) \pm 2$ $((DX)) \leftarrow ((SI \text{ 或 } ESI)), (SI \text{ 或 } ESI) \leftarrow (SI \text{ 或 } ESI) \pm 4$	386 起
POP reg/mem/SR POPA POPAD POPF POPFD	$(reg/mem/SR) \leftarrow ((SP \text{ 或 } ESP)), (SP \text{ 或 } ESP) \leftarrow (SP \text{ 或 } ESP) + 2 \text{ 或 } 4$ 出栈送 16 位通用寄存器 出栈送 32 位通用寄存器 出栈送 FLAGS 出栈送 EFLAGS	286 起 386 起 386 起
PUSH reg/mem/SR/imm POPA POPAD POPF POPFD	$((SP \text{ 或 } ESP)) \leftarrow (SP \text{ 或 } ESP) - 2 \text{ 或 } 4$ $((SP \text{ 或 } ESP)) \leftarrow (reg/mem/SR/imm)$ 16 位通用寄存器进栈 32 位通用寄存器进栈 FLAGS 进栈 EFLAGS 进栈	imm 自 386 起 286 起 386 起 386 起
RCL reg/mem, 1/CL/I8	带进位循环左移	18 自 386 起
RCR reg/mem, 1/CL/I8	带进位循环右移	18 自 386 起
RDMSR	读模型专用寄存器: $(EDX, EAX) \leftarrow (MSR[ECX])$	586 起
REP REPE/REPZ REPNE/REPNZ	$(CX \text{ 或 } ECX) \leftarrow (CX \text{ 或 } ECX) - 1$, 当 $(CX \text{ 或 } ECX) \neq 0$, 重复执行后面的指令 $(CX \text{ 或 } ECX) \leftarrow (CX \text{ 或 } ECX) - 1$, $(CX \text{ 或 } ECX) \neq 0$ 且 $ZF=1$, 重复执行后面的指令 $(CX \text{ 或 } ECX) \leftarrow (CX \text{ 或 } ECX) - 1$, $(CX \text{ 或 } ECX) \neq 0$ 且 $ZF=0$, 重复执行后面的指令	
RET RET dl6	段内: $(IP) \leftarrow POP()$, 段间: $(IP) \leftarrow POP()$, $(CS) \leftarrow POP()$ 段内: $(IP) \leftarrow POP()$, $(SP \text{ 或 } ESP) \leftarrow (SP \text{ 或 } ESP) + d16$ 段间: $(IP) \leftarrow POP()$, $(CS) \leftarrow POP()$, $(SP \text{ 或 } ESP) \leftarrow (SP \text{ 或 } ESP)$	

助 记 符	功 能	备 注
	+d16	
ROL reg/mem, 1/CL/I8	循环左移	18 自 386 起
ROR reg/mem, 1/CL/I8	循环右移	18 自 386 起
RSM	从系统管理方式恢复	586 起, 系统指令
SAHF	(FLAGS 的低字节) \leftarrow (AH)	
SAL reg/mem, 1/CL/I8	算术左移	18 自 386 起
SAR reg/mem, 1/CL/I8	算术右移	18 自 386 起
SBB reg, reg/mem/imm Mem, reg/imm	(dst) \leftarrow (dst)-(src)-CF	
SCASB SCASW SCASD	(AL)-((DI 或 EDI)), (DI 或 EDI)-(DI 或 EDI) \pm 1 (AX)-((DI 或 EDI)), (DI 或 EDI)-(DI 或 EDI) \pm 2 (EAX)-((DI 或 EDI)), (DI 或 EDI)-(DI 或 EDI) \pm 4	386 起
SETcc r8/m8	条件设置:指定条件 cc 满足则(r8/m8)送 1, 否则送 0	386 起
SGDT mem	保存全局描述符表寄存器:(mem) \leftarrow (GDTR)	386 起, 系统指令
SHL reg/mem, 1/c1/i8	逻辑左移	18 自 386 起
SHLD reg/mem, reg, i8/CL	双精度左移	386 起
SHR reg/mem, 1/c1/i8	逻辑右移	18 自 386 起
SHRD reg/mem, reg, i8/CL	双精度右移	386 起
SIDT mem	保存中断描述符表:(mem) \leftarrow (IDTR)	286 起, 系统指令
SLDT reg/mem	保存局部描述符表:(reg/mem) \leftarrow (LDTR)	286 起, 系统指令
SMSW reg/mem	保存机器状态字:(reg/mem) \leftarrow (MSW)	286 起, 系统指令
STC STD STI	进位位置 1 方向标志置 1 中断标志置 1	
STOSB STOSW STOSD	((DI 或 EDI) \leftarrow (ac), (DI 或 EDI) \leftarrow (DI 或 EDI) \pm 1 ((DI 或 EDI) \leftarrow (ac), (DI 或 EDI) \leftarrow (DI 或 EDI) \pm 2 ((DI 或 EDI) \leftarrow (ac), (DI 或 EDI) \leftarrow (DI 或 EDI) \pm 4	386 起
STR reg/mem	保存任务寄存器:(reg/mem) \leftarrow (TR)	286 起, 系统指令
SUB reg, mem/imm/reg mem, reg/imm ac, imm	(dst) \leftarrow (dst) \leftarrow (src)	
TEST reg, mem/imm/reg mem, reg/imm ac, imm	(dst) \wedge (src), 结果影响标志位	
VERR reg/mem	检验 reg/mem 中的选择器所表示的段是否可读	286 起, 系统指令
VERW reg/mem	检验 reg/mem 中的选择器所表示的段是否可写	286 起, 系统指令
WAIT	等待	
WBINVD	写回并使高速缓存无效	486 起, 系统指令

助 记 符	功 能	备 注
WRMSR	写入模型专用寄存器:MSR(ECX) \leftarrow (EDX, EAX)	586 起, 系统指令
XADD reg/mem, reg	TEMP \leftarrow (src)+(dst), (src) \leftarrow (dst), (dst) \leftarrow TEMP	486 起
XCHG reg/ac/mem, reg	(dst) \leftarrow \rightarrow (src)	
XLAT	(AL) \leftarrow ((BX 或 EBX)+(AL))	
XOR reg, mem/imm/reg mem, reg/imm ac, imm	(dst) \leftarrow (dst) \oplus (src)	

3. MMX 指令

指令类型	助 记 符	功 能
算术运算	PADD[B, W, D]	mm, mm/m64
	PADDS[B, W]	mm, mm/m64
	PADDUS[B, W]	mm, mm/m64
	PSUB[B, W, D]	mm, mm/m64
	PSUBS[B, W]	mm, mm/m64
	PSUBUS[B, W]	mm, mm/m64
	PMULHW	mm, mm/m64
	PMULLW	mm, mm/m64
	PMADDWD	mm, mm/m64
比较	PCMPEQ[B, W, D]	mm, mm/m64
	PCMPGT[B, W, D]	mm, mm/m64
类型转换	PACKUSWB	mm, mm/m64
	PACKSS[WB, DW]	mm, mm/m64
	PUNPCKH[BW,WD,DQ]	mm, mm/m64
	PUNPCKL[BW,WD,DQ]	mm, mm/m64
逻辑运算	PAND	mm, mm/m64
	PANDN	mm, mm/m64
	POR	mm, mm/m64
	PXOR	mm, mm/m64
移位	PSLL[W, D, Q]	mm, m64/mm/18
	PSRL[W, D, Q]	mm, m64/mm/18
	PSRA[W, D]	mm, m64/mm/18
数据传送	MOVD	mm, r32/m32
	MOVD	r32/m32, mm

	MOVQ	m64/mm, mm	(m64/mm)←(mm)
	MOVQ	mm, m64/mm	(mm)←(m64/mm)
状态清除	EMMS		清除 MMX 状态（浮点数据寄存器清空）

4. SSE 指令

指令类型	指令助记符		功 能
转换指令	CVTPI2PS	xmm, mm/m64	紧缩整数转换为浮点数:将 mm/m64 中两个 32 位有符号整数转换为浮点数存入 xmm 低 64 位, 高 64 位不变
	CVTPS2PI	mm, xmm/m64	紧缩浮点数转换为整数:将 xmm 低 64 位或 m64 中两个 32 位浮点数转换为两个 32 位有符号整数存入 mmx 寄存器
	CVTPI2SS	xmm, r32/m32	标量整数转换为浮点数:将 r32/m32 中 32 位有符号整数转换为浮点数存入 xmm 低 32 位, 高 96 位不变
	CVTSS2PI	r32, xmm/m32	标量浮点数转换为整数:将 xmm 低 32 位或 m32 中 32 位浮点数转换为 32 位有符号整数存入 r32
浮点 数据传送	MOVAPS	xmm, xmm/m128	对齐数据传送:(xmm) ←(xmm/m128)
	MOVAPS	xmm/m128, xmm	对齐数据传送:(xmm/m128) ←(xmm)
	MOVUPS	xmm, xmm/m128	不对齐数据传送:(xmm) ←(xmm/m128)
	MOVUPS	xmm/m128, xmm	不对齐数据传送:(xmm/m128) ←(xmm)
	MOVHPS	xmm, m64	高 64 位传送:xmm 高 64 位←(m64)
	MOVHPS	m64, xmm	高 64 位传送:m64←(xmm 高 64 位)
	MOVLPS	xmm, m64	低 64 位传送:xmm 低 64 位←(m64)
	MOVLPS	m64, xmm	低 64 位传送:m64←(xmm 低 64 位)
	MOVHLPs	xmm, xmm	64 位高送低:(xmm 低 64 位) ←(xmm 高 64 位)
	MOVLHPS	xmm, xmm	64 位低送高:(xmm 高 64 位) ←(xmm 低 64 位)
	MOVMSKPS	r32, xmm	屏蔽位传送:将 xmm 中 4 个单精度浮点数字符号位送 r32 低 4 位, 其余位清零
	MOVSS	xmm, m32	标量数据传送:m32 送 xmm 低 32 位, 其余 96 位清零
组合指令	SHUFPS	xmm,xmm/m128,I8	紧缩浮点数组合:将目的 xmm4 个单精度浮点数组组合到目的 xmm 低两个浮点数位置, 源操作数 4 个单精度浮点数进行组合送目的 xmm 高两个浮点数位置, 组合方法由 I8 决定
	UNPCKHPS	xmm,xmm/m128	高交叉组合:将 xmm/m128 和 xmm 的高两个浮点数交叉组合到 xmm
	UNPCKHPS	xmm,xmm/m128	低交叉组合:将 xmm/m128 和 xmm 的低两个浮点数交叉组合到 xmm
浮点 比较运算	CMPPS	xmm,xmm/m128,I8	紧缩浮点数比较:xmm 和 xmm/m128 中 4 对浮点数比较, 如果满足 I8 或 dd 要求, xmm 置全“1”, 否则置全“0” dd 为:EQ, LT, LE, UNORD, NEQ, NLT, NLE, ORD 对应 I8 为:0~7
	CMPddPS	xmm,xmm/m128	
	CMPSS	xmm,xmm/m128,I8	标量浮点数比较:xmm 和 xmm/m128 中最低一对浮点数比较, 如果满足 I8 要求, xmm 置全“1”, 否则置全“0”

指令类型	指令助记符		功 能
浮点 比较运算	COMISS	xmm, xmm/m32	设置整数标志有序标量浮点数比较:比较 xmm 和 m32 或 xmm 中最低一对浮点数, 设置 EFLAGS 寄存器。源操作数是 SNaN 或 QNaN 时产生无效数值异常
	UCOMISS	xmm, xmm/m32	设置整数标志无序标量浮点数比较:比较 xmm 和 m32 或 xmm 中最低一对浮点数, 设置 EFLAGS 寄存器。源操作数是 SNaN 时产生无效数值异常
转换指令	CVTPI2PS	xmm, mm/m64	紧缩整数转换为浮点数:将 mm/m64 中两个 32 位有符号整数转换为浮点数存入 xmm 低 64 位, 高 64 位不变
	CVTPS2P	mm, xmm/m64	紧缩浮点数转换为整数:将 xmm 低 64 位或 m64 中两个 32 位浮点数转换为两个 32 位有符号整数存入 mmx 寄存器
	CVTPI2SS	xmm, r32/m32	标量整数转换为浮点数:将 r32/m32 中 32 位有符号整数转换为浮点数存入 xmm 低 32 位, 高 96 位不变
	CVTSS2PI	r32, xmm/m32	标量浮点数转换为整数:将 xm 低 32 位或 m32 中 32 位浮点数转换为 32 位有符号整数存入 r32
浮点 算术运算	ADDPS	xmm, xmm/m128	紧缩浮点数加:xmm 和 xmm/m128 中四对浮点数加
	ADDSS	xmm, xmm/m32	标量浮点数加:xmm 和 xmm/m32 中最低一对浮点数加
	SUBPS	xmm, xmm/m128	紧缩浮点数减:xmm 和 xmm/m128 中四对浮点数减
	SUBSS	xmm, xmm/m32	标量浮点数减:xmm 和 xmm/m32 中最低一对浮点数减
	MULPS	xmm, xmm/m128	紧缩浮点数乘:xmm 和 xmm/m128 中四对浮点数乘
	MULSS	xmm, xmm/m32	标量浮点数乘:xmm 和 xmm/m32 中最低一对浮点数乘
	DIVPS	xmm, xmm/m128	紧缩浮点数除:xmm 和 xmm/m128 中四对浮点数除
	DIVSS	xmm, xmm/m32	标量浮点数除:xmm 和 xmm/m32 中最低一对浮点数除
	SQRTPS	xmm, xmm/m128	紧缩浮点数平方根:求 xmm/m128 中四对浮点数平方根, 存入 xmm
	SQRTPS	xmm, xmm/m32	标量浮点数平方根:求 xmm/m32 中最低一对浮点数平方根, 存入 xmm 低 32 位, 高 96 位不变
	MAXPS	xmm, xmm/m128	紧缩浮点数最大值:求 xmm/m128 和 xmm 中四对浮点数最大值, 存入 xmm
	MAXSS	xmm, xmm/m32	标量浮点数最大值:求 xmm/m32 和 xmm 中最低一对浮点数最大值, 存入 xmm 低 32 位, 高 96 位不变
	MINPS	xmm, xmm/m128	紧缩浮点数最小值:求 xmm/m128 和 xmm 中四对浮点数最小值, 存入 xmm
	MINSS	xmm, xmm/m32	标量浮点数最小值:求 xmm/m32 和 xmm 中最低一对浮点数最小值, 存入 xmm 低 32 位, 高 96 位不变
逻辑运算 指令	ANDPS	xmm, xmm/m128	逻辑与:实现两个 128 位操作数按位逻辑与
	ANDNPS	xmm, xmm/m128	逻辑非与:对 128 位目的操作数求反, 然后与源操作数按位逻辑与
	ORPS	xmm, xmm/m128	逻辑或:实现两个 128 位操作数按位逻辑或
	XORPS	xmm, xmm/m128	逻辑异或:实现两个 128 位操作数按位逻辑异或
整数指令	PAVG[B,W]	mm, mm/m64	紧缩整数求平均值:按字节/字求对应两个无符号数平均值
	PEXTRW	r32, mm, I8	取出字:将 mm 中 18 低两位指定的 16 位字送 r32 低 16 位, 高位清零

指令类型	指令助记符		功 能
	PINSRW	mm, r32/m16, I8	插入字:将 m16 或 r32 中低 16 位字送 mm 中由 18 指定的 16 位字
	PMAXUB	mm, mm/m64	紧缩整数求最大值:按字节求对应 8 对无符号数最大值存入 mm
整数指令	PMAXSW	mm, mm/m64	紧缩整数求最大值:按字求对应 4 对有符号数最大值存入 mm
	PMINUB	mm, mm/m64	紧缩整数求最小值:按字节求对应 8 对无符号数最小值存入 mm
	PMINSW	mm, mm/m64	紧缩整数求最小值:按字求对应 4 对有符号数最小值存入 mm
	PMOVMASKB	r32, mm	屏蔽位传送:将 mm 寄存器每个字节最高位送 r32 的最低 8 位, 高位清零
	PMULHUW	mm, mm/m64	无符号高乘:进行 4 对无符号整数乘法, 积的高 16 位送 mm 寄存器
	PSADBWB	mm, mm/m64	绝对差求和:求 8 对有符号数差的绝对值, 它们的和送 mm 的低 16 位
	PSHUFBW	mm, mm/m64, I8	紧缩整数组合:按照 18 给出的方式, 把两个紧缩字整数组合到 mm 中
状态管理指令	STMXCSR	m32	保存 SIMD 控制/状态寄存器:将 SIMD 控制/状态寄存器内容装入 m32
	LDMXCSR	m32	恢复 SIMD 控制/状态寄存器:将 m32 内容装入 SIMD 控制/状态寄存器
	FXSAVE	m512	保存所有状态:将 FPU, MMX 和 SIMD 所有状态装入 m512
	FXRSTOR	m512	恢复所有状态:从 m512 恢复 FPU, MMX 和 SIMD 所有状态
高速缓存优化处理指令	MASKMOVQ	mm, mm	字节屏蔽写入:将目的 mm 寄存器内容按源 mm 寄存器 8 字节最高位屏蔽后(等于 1, 数据不变; 等于 0, 数据清零)送 DS:DI/EDI 指定的存储单元。数据不经过 Cache
	MOVNTQ	m64, mm	64 位传送:将 mm 寄存器内容不经过 Cache 写入 m64 存储器
	MOVNTPS	m128, xmm	紧缩浮点数传送:将 xmm 寄存器内容不经过 Cache 写入 m128 存储器
	PREFETCH	m8 [T0,T1,T2,NTA]	预取:将 m8 指定的 Cache 行组内容预取进入由 T0, T1, T2, NTA 规定的各级 Cache T0——预取数据进入各级 Cache T1——预取数据进入除第一级以外的各级 Cache T2——预取数据进入除第一级、第二级以外的各级 Cache TA——仅将预取数据进入第一级 Cache
	SFENCE		存储隔离:保证在执行本指令之前的写入指令对本指令后的写入指令可见, 避免预取指令的副作用

附录 D 伪指令和操作符

1. 伪指令

伪指令类型	伪 指 令
变量定义	DB/BYTE/SBYTE, DW/WORD/SWORD, DD/DWORD/SDWORD/REAL4, FWORD/DF, QWORD/DQ/REAL8, TBYTE/DT/REAL10
定位	EVEN, ALIGN, ORG
符号定义	RADIX, =, EQU, TEXTEQU, LABEL
简化段定义	.MODEL, .STARTUP, .EXIT, .STACK, .DATA?, .CONST, .FARDATA, .FARDATA?
完整段定义	SEGMENT, ENDS, GROUP, ASSUME, END, .DOSSEG/.ALPHA/.SEQ
复杂数据类型	STRUCT/STRUC, UNION, RECORD, TYPEDER, ENDS
流程控制	.IF, .ELSE, .ELSEIF, .ENDIF, .WHILE, .ENDW, .REPEAT, .UNTIL, .BREAK, .CONTINUE
过程定义	PROC, ENDP, PROTO, INVOKE
宏汇编	MACRO, ENDM, PURGE, LOCAL, PUSHCONTEXT, POPCONTEXT, EXITM, GOTO
重复汇编	REPEAT/REPT, WHILE, FOR/IRP, FORC/IRPC
条件汇编	IF, IFE, IFB, IFNB, IFDEF, IFNDEF, IFDIF, IFIDN, ELSE, ELSEIF, ENDIF
模块化	PUBLIC, EXETEN/EXTERN, COMM, INCLUDE, INCLUDELIB
条件错误	.ERR, .ERRE, .ERRB, .ERRNB, .ERRDEF, .ERRNDEF, .ERRDIF, .ERRIDN
列表控制	TITLE, SUBTITLE, PAGE, .LIST, .LISTALL, .LISTMACRO, .LISTMACROALL, .LISTIF, .NOLIST, .TFCOND, .CREF, .NOCREF, COMMENT, ECHO
处理器选择	.8086, .186, .286, .286P, .386, .386P, 486, .486P, .8087, .287, .387, .NO87
字符串处理	CATSTR, INSTR, SIZESTR, SUBSTR

2. 操作符

操作符类型	操 作 符
算术运算符	+, -, *, /, MOD
逻辑运算符	AND, OR, XOR, NOT
移位运算符	SHL, SHR
关系运算符	EQ, NE, GT, LT, GE, LE
高低分离符	HIGH, LOW, HIGHWORD, LOWWORD
地址操作符	[], \$, :, OFFSET, SEG
类型操作符	PTR, THIS, SHORT, TYPE, SIZEOF/SIZE, LENGTHOF/LENGTH
复杂数据操作符	(), < >, ., MASK, WIDTH, ?, DUP, “ ”
宏操作符	&, < >, !, %, ;
流程条件操作符	=, !=, >, >=, <, <=, &&, , !, &, CARRY?, OVERFLOW?, PARITY?, SIGN, ZERO?

附录 E DOS 功能调用

AH	功 能	调用参数	返回参数
00	程序终止(同 INT21H)	CS=程序段前缀 PSP	
01	键盘输入并回显		AL=输入字符
02	显示输出	DE=输出字符	
03	辅助设备(COM1)输入		AL=输入数据
04	辅助设备(COM1)输出	DL=输出字符	
05	打印机输出	DE=输出字符	
06	直接控制台 I/O	DL=FF(输入) DE=字符(输出)	AL=输入字符
07	键盘输入(无回显)		AL=输入字符
08	键盘输入(无回显) 检测 Ctrl+Break 或 Ctrl+C		AL=输入字符
09	显示字符串	DS:DX=串地址(字符串以‘\$’ 结尾)	
0A	键盘输入到缓冲区	DS:DX=缓冲区首址 (DS:DX)=缓冲区最大字符数	(DS:DX+1)=实际输入的字符数
0B	检验键盘状态		AL=00 有输入 AL=FF 无输入
0C	清除缓冲区并请求 指定的输入功能	AL=输入功能号(1,6,7,8)	AL=输入字符
0D	磁盘复位		清除文件缓冲区
0E	指定当前默认的磁盘驱动器	DL=驱动器号(0=A,1=b, ...)	AL=系统中驱动器数
0F	打开文件(FCB)	DS:DX=FCB 首地址	AL=00 文件找到 AL=FF 文件未找到
10	关闭文件(FCB)	DS:DX=FCB 首地址	AL=00 目录修改成功 AL=FF 目录中未找到文件
11	查找第一个目录项(FCB)	DS:DX=FCB 首地址	AL=00 找到匹配的目录项 AL=FF 未找到匹配的目录项
12	查找下一个目录项(FCB)	DS:DX=FCB 首地址 使用通配符进行目录项查找	AL=00 找到匹配的目录项 AL=FF 未找到匹配的目录项
13	删除文件(FCB)	DS:DX=FCB 首地址	AL=00 删除成功 AL=FF 文件未删除
14	顺序读文件(FCB)	DS:DX=FCB 首地址	AL=00 读成功 =01 文件结束, 未读到数据 =02DTA 边界错误 =03 文件结束, 记录不完整

AH	功 能	调用参数	返回参数
15	顺序写文件(FCB)	DS:DX=FCB 首地址	AL=00 写成功 =01 磁盘满或是只读文件 =02DTA 边界错误
16	建文件(FCB)	DS:DX=FCB 首地址	AL=00 建文件成功 =FF 磁盘操作有错
17	文件改名(FCB)	DS:DX=FCB 首地址	AE=00 文件被改名 =FF 文件未改名
19	取当前默认磁盘驱动器		AL=默认的驱动器号 0=A, 1=B, 2=C, ...
1A	设置 DTA 地址	DS:DX=DTA 地址	
1B	取默认驱动器 FAT 信息		AL=每簇的扇区数 DS:BX=指向介质说明的指针 CX=物理扇区的字节数 DX=每磁盘簇数
1C	取指定驱动器 FAT 信息	DL=驱动器号	同上
1F	取默认磁盘参数块		AL=00 无错 =FF 出错 DS:BX=磁盘参数块地址
21	随机读文件(FCB)	DS:DX=FCB 首地址	AL=00 读成功 =01 文件结束 =02 DTA 边界错误 =03 读部分记录
22	随机写文件(FCB)	DS:DX=FCB 首地址	AL=00 写成功 =01 磁盘满或是只读文件 =02 DTA 边界错误
23	测定文件大小(FCB)	DS:DX=FCB 首地址	AL=00 成功, 记录数填入 FCB =FF 未找到匹配的文件
24	设置随机记录号	DS:DX=FCB 首地址	
25	设置中断向量	DS:DX=中断向量 AL=中断类型号	
26	建立程序段前缀 PSP	DX=新 PSP 段地址	
27	随机分块读(FCB)	DS:DX=FCB 首地址 CX=记录数	AL=00 读成功 =01 文件结束 =02 DTA 边界错误 =03 读部分记录 CX=读取的记录数
28	随机分块写(FCB)	DS:DX=FCB 首地址 CX=记录数	AL=00 写成功 =01 磁盘满或是只读文件 =02 DTA 边界错误

AH	功 能	调用参数	返回参数
29	分析文件名字符串(FCB)	ES:DI=FCB 首址 DS:SI=ASCIZ 串 AL=分析控制标志	AL=00 标准文件 =01 多义文件 =FF 驱动器说明无效
2A	取系统日期		CX=年(1980~2099) DH=月(1~12) DL=日(1~31) AL=星期(0~6)
2B	置系统日期	CX=年(1980~2099) DH=月(1~12) DL=日(1~31)	AL=00 成功 =FF 无效
2C	取系统时间		CH:CL=时:分 DH:DL=秒:1/100 秒
2D	置系统时间	CH:CL=时:分 DH:DL=秒:1/100 秒	AL=00 成功 =FF 无效
2E	设置磁盘检验标志	AL=00 关闭检验 =FF 打开检验	
2F	取 DTA 地址		ES:BX=DTA 首地址
30	取 DOS 版本号		AL=版本号 AH=发行号 BH=DOS 版本标志 BL:CX=序号(24 位)
31	结束并驻留	AL=返回码 DX=驻留区大小	
32	取驱动器参数块	DL=驱动器号	AL=FF 驱动器无效 DS:BX=驱动器参数块地址
33	Ctrl+Break 检测	AL=00 取标志状态	DL=00 关闭 Ctrt-Break 检测 =01 打开 Ctrl-Break 检测
35	取中断向量	AL=中断类型	ES:BX=中断向量
36	取空闲磁盘空间	DL=驱动器号 0=默认, 1=A, 2=B, ...	成功: AX=每簇扇区数 BX=可用簇数 CX=每扇区字节数 DX=磁盘总簇数
38	置/取国别信息	AL=00 取当前国别信息 =FF 国别代码放在 BX 中 DS:DX=信息区首地址 DX=FFFF 设置国别代码	BX=国别代码 (国际电话前缀码) DS:DX=返回的信息区首址 AX=错误代码
39	建立子目录	DS:DX=ASCIZ 串地址	AX=错误码
3A	删除子目录	DS:DX=ASCIZ 串地址	AX=错误码

AH	功 能	调用参数	返回参数
3B	设置当前目录	DS:DX=ASCIZ 串地址	AX=错误码
3C	建立文件(handle)	DS:DX=ASCIZ 串地址 CX=文件属性	成功: AX=文件代号(CF=0) 失败: AX=错误码 (CF=1)
3D	打开文件(handle)	DS:DX=ASCIZ 串地址 AL=访问和文件共享方式 0=读, 1=写, 2=读/写	成功: AX=文件代号(CF=0) 失败: AX=错误码(CF=1)
3E	关闭文件(handle)	BX=文件代号	失败: AX=错误码 (CF=1)
3F	读文件或设备(handle)	DS:DX=ASCIZ 串地址 BX=文件代号 CX=读取的字节数	成功: AX=实际读入的字节数(CF=0) AX=0 已到文件尾 失败: AX=错误码(CF=1)
40	写文件或设备(handle)	DS:DX=ASCIZ 串地址 BX=文件代号 CX=写入的字节数	成功: AX=实际写入的字节数 失败: AX=错误码(CF=1)
41	删除文件	DS:DX=ASCIZ 串地址	成功: AX=00 失败: AX=错误码(CF=1)
42	移动文件指针	BX=文件代号 CX:DX=位移量 AL=移动方式	成功: DX:AX=新指针位置 失败: AX=错误码(CF=)
43	置/取文件属性	DS:DX=ASCIZ 串地址 AL=00 取文件属性 AL=01 置文件属性 CX=文件属性	成功: CX=文件属性 失败: AX=错误码(CF=1)
44	设备驱动程序控制	BX=文件代号 AL=设备子功能代码 (0~11H) 0=取设备信息 1=置设备信息	成功: DX=设备信息 AX=传送的字节数 失败: AX=错误码 (CF=1)
44	设备驱动程序控制	2=读字符设备 3=写字符设备 4=读块设备 5=写块设备 6=取输入状态 7=取输出状态 BL=驱动器代码 CX=读/写的字节数	
45	复制文件代号	BX=文件代号 1	成功: AX=文件代号 2 失败: AX=错误码(CF=1)
46	强行复制文件代号	BX=文件代号 1 CX=文件代号 2	失败: AX=错误码(CF=1)
47	取当前目录路径名	DL=驱动器号 DS:SI=ASCIZ 串地址 (从根目录开始的路径名)	成功: DS:SI=当前 ASCIZ 串地址 失败: AX=错误码(CF=1)

AH	功 能	调用参数	返回参数
48	分配内存空间	BX=申请内存数	成功: AX=分配内存的初始段地址 失败: AX=错误码(CF=1) BX=最大可用空间
49	释放已分配内存	ES=内存起始段地址	失败: AX=错误码(CF=1)
4A	修改内存分配	ES=原内存起始段地址 BX=新申请内存字节数	失败: AX=错误码(CF=1) BX=最大可用空间
4B	装入/执行程序	DS:DX=ASCIZ 串地址 ES:BX=参数区首地址 AL=00 装入并执行程序 =03 装入程序, 但不执行	失败: AX=错误码
4C	带返回码终止	AL=返回码	
4D	取返回代码		AL=子出口代码 AH=返回代码 00=正常终止 01=用 Ctrl+C 终止 02=严重设备错误终止 03=用功能调用 31H 终止
4E	查找第一个匹配文件	DS:DX=ASCIZ 串地址 CX=属性	失败: AX=错误码 (CF=1)
4F	查找下一个匹配文件	DTA 保留 4EH 的原始信息	失败: AX=错误码(CF=1)
50	置 PSP 段地址	BX=新 PSP 段地址	
51	取 PSP 段地址		BX=当前运行进程的 PSP
52	取磁盘参数块		ES:BX=参数块链表指针
53	把 BIOS 参数块(BPB)转换为 DOS 的驱动器参数块(DPB)	DS:SI=BPB 的指针 ES:BP=DPB 的指针	
54	取写盘后读盘的检验标志		AL=00 检验关闭 =01 检验打开
55	建立 PSP	DX=建立 PSP 的段地址	
56	文件改名	DS:DX=当前 ASCIZ 串地址 ES:DI=新 ASCn 串地址	失败: AX=错误码 (CF=1)
57	置/取文件日期和时间	BX=文件代号 AL=00 读取日期和时间 AL=01 设置日期和时间 (DX:CX)=日期, 时间	失败: AX=错误码 (CF=1)
58	取/置内存分配策略	AL=00 取策略代码 AL=01 置策略代码 BX=策略代码	成功: AX=策略代码 失败: AX=错误码 (CFM)

AH	功 能	调用参数	返回参数
59	取扩充错误码	BX=00	AX=扩充错误码 BH=错误类型 BL=建议的操作 CH=出错设备代码
5A	建立临时文件	CX=文件属性 DS:DX=ASCIZ 串(以\结束) 地址	成功: AX=文件代号 DS:DX=ASCIZ 串地址 失败: AX=错误代码 (CF=1)
5B	建立新文件	CX=文件属性 DS:DX=ASCIZ 串地址	成功: AX=文件代号 失败: AX=错误代码 (CF=1)
5C	锁定文件存取	AL=00 锁定文件指定的区域 =01 开锁 BX=文件代号 CX:DX=文件区域偏移值 SI:DI=文件区域的长度	失败: AX=错误代码 (CF=1)
5D	取/置严重错误标志的地址	AL=06 取严重错误标志地址 AL=0A 置 ERROR 结构指针	DS:SI=严重错误标志的地址
60	扩展为全路径名	DS:SI=ASCIZ 串的地址 ES:DI=工作缓冲区地址	失败: AX=错误代码(CF=1)
62	取程序段前缀地址		BX=PSP 地址
68	刷新缓冲区数据到磁盘	AL=文件代号	失败: AX=错误代码(CF=1)
6C	扩充的文件打开/建立	AL=访问权限 BX=打开方式 CX=文件属性 DS:SI=ASCIZ 串地址	成功: AX=文件代号 CX=采取的动作 失败: AX=错误代码(CF=1)

附录 F BIOS 功能调用

INT	AH	功 能	调用参数	返回参数
10	0	设置显示方式	AL=00 40x25 黑白文本, 16 级灰度 =01 40x25 16 色文本 =02 80x25 黑白文本, 16 级灰度 =03 80x25 16 色文本 =04 320x200 4 色图形 =05 320x200 黑白图形, 4 级灰度	
10	0	设置显示方式	=06 640x200 黑白图形 =07 80x25 黑白文本 =08 160x200 16 色图形(MCGA) =09 320x200 16 色图形(MCGA) =0A 640x200 4 色图形(MCGA) =0D 320x200 16 色图形(EGA/VGA) =0E 640x200 16 色图形(EGA/VGA) =0F 640x350 单色图形(EGA/VGA) =10 640x350 16 色图形(EGA/VGA) =11 640x480 黑白图形(VGA) =12 640x480 16 色图形(VGA) =13 320x200 256 色图形(VGA)	
10	1	置光标类型	(CH) ₀₋₃ =光标起始行 (CL) ₀₋₃ =光标结束行	
10	2	置光标位置	BH=页号 DH/DL=行/列	
10	3	读光标位置	BH=页号	CH=光标起始行 CL=光标结束行 DH/DL=行/列
10	4	读光笔位置		AX=0 光笔未触发 =1 光笔触发 CH/BX=像素行/列 DH/DL=字符行/列
10	5	置当前显示页	AL=页号	
10	6	屏幕初始化或上卷	AL=0 初始化窗口 AL=上卷行数 BH=卷入行属性 CH/CL=左上角行/列号 DH/DL=右下角行/列号	
10	7	屏幕初始化或下卷	AL=0 初始化窗口 AL=下卷行数 BH=卷入行属性 CH/CL=左上角行/列号 DH/DL=右下角行/列号	

INT	AH	功 能	调用参数	返回参数
10	8	读光标位置的字符和属性	BH=显示页	AH/AL=属性/字符
10	9	在光标位置显示字符和属性	BH=显示页 AL/BL=字符/属性 CX=字符重复次数	
10	A	在光标位置显示字符	BH=显示页 AL=字符 CX=字符重复次数	
10	B	置彩色调色板	BH=彩色调色板 ID BL=和 ID 配套使用的颜色	
10	C	写像素	AL=颜色值 BH=页号 DX/CX=像素行/列	
10	D	读像素	BH=页号 DX/CX=像素行/列	AL=像素的颜色值
10	E	显示字符 (光标前移)	AL=字符 BH=页号 BL=前景色	
10	F	取当前显示方式		BH=页号 AH=字符列数 AL=显示方式
10	10	置调色板寄存器 (EGA/VGA)	AL=0,BL=调色板号,BH=颜色值	
10	11	装入字符发生器 (EGA/VGA)	AL=0~4 全部或部分装入字符点阵集 AL=20~24 置图形方式显示字符集 AL=30 读当前字符集信息	ES:BP=字符集位置
10	12	返回当前适配器设置的信息(EGA/VGA)	BL=10H(子功能)	BH=0 单色方式 =1 彩色方式 BL=VRAM 容量 (0=64KB, 1=128KB, ...) CH=特征位设置 CL=EGA 的开关设置
10	13	显示字符串	ES:BP=字符串地址 AL=写方式(0~3) CX=字符串长度 DH/DL=起始行/列 BH/BL=页号/属性	
11		取系统设备信息		AX=返回值(位映像) 0=对应设备未安装 1=对应设备已安装
12		取内存容量		AX=内存容量(单位 KB)
13	0	磁盘复位	DL=驱动器号	失败: AH=错误码

INT	AH	功 能	调用参数	返回参数
			(00,01 为软盘,80H,81H,...为硬盘)	
13	1	读磁盘驱动器状态		AH=状态字节
13	2	读磁盘扇区	AL=扇区数 (CL) _{6,7} (CH) ₀₋₇ =磁道号 (CL) ₀₋₅ =扇区号 DH/DL=磁头号/驱动器号 ES:BX=数据缓冲区地址	读成功: AH=0 AL=读取的扇区数 读失败: AH=错误码
13	3	写磁盘扇区	同上	写成功: AH=0 AL=写入的扇区数 写失败: AH=错误码
13	4	检验磁盘扇区	AL=扇区数 (CL) _{6,7} (CH) ₀₋₇ =磁道号 (CL) ₀₋₅ =扇区号 DH/DL=磁头号/驱动器号	成功: AH=0 AL=检验的扇区数 失败: AH=错误码
13	5	格式化磁盘磁道	AL=扇区数 (CL) _{6,7} (CH) ₀₋₇ =磁道号 (CL) ₀₋₅ =扇区号 DH/DL=磁头号/驱动器号 ES:BX=格式化参数表指针	成功: AH=0 失败: AH=错误码
14	0	初始化串行口	AL=初始化参数 DX=串行口号	AH=通信口状态 AL=调制解调器状态
14	1	向通信口写字符	AL=字符 DX=通信口号	写成功: (AH) ₇ =0 写失败: (AH) ₇ =1 (AH) ₀₋₆ =通信口状态
14	2	从通信口读字符	DX=通信口号	读成功: (AH) ₇₋₀ , (AH) ₆₋₀ =字符 读失败: (AH) ₇ =1
14	3	取通信口状态	DX=通信口号	AH=通信口状态 AL=调制解调器状态
14	4	初始化扩展 COM		
14	5	扩展 COM 控制		
15	0	启动盒式磁带机		
15	1	停止盒式磁带机		
15	2	磁带分块读	ES:BX=数据传输区地址 CX=字节数	AH=状态字节 =00 读成功 =01 冗余检验错 =02 无数据传输 =04 无引导 =80 非法命令
15	3	磁带分块写	DS:BX=数据传输区地址 CX=字节数	AH=状态字节 (格式同上)
16	0	从键盘读字符		AL=字符码

INT	AH	功 能	调用参数	返回参数
				AH=扫描码
16	1	取键盘缓冲区状态		ZF=0 AL=字符码 AH=扫描码 ZF=1 缓冲区无按键，等待
16	2	取键盘标志字节		AL=键盘标志字节
17	0	打印字符，回送状态字节	AL=字符 DX=打印机号	AH=打印机状态字节
17	1	初始化打印机，回送状态字节	DX=打印机号	AH=打印机状态字节
17	2	取打印机状态	DX=打印机号	AH=打印机状态字节
18		ROM BASIC 语言		
19		引导装入程序		
1A	0	读时钟		CH:CL=时:分 DH:DL=秒:1/100 秒
1A	1	置时钟	CH:CL=时:分 DH:DL=秒:1/100 秒	
1A	6	置报警时间	CH:CL=时:分(BCD) DH:DL=秒:1/100 秒(BCD)	
1A	7	清除报警		
33	00	鼠标复位	AL=00	BX=鼠标的键数
33	00	显示鼠标光标	AL=01	显示鼠标光标
33	00	隐藏鼠标光标	AL=02	隐藏鼠标光标
33	00	读鼠标状态	AL=03	BX=键状态 CX/DX=鼠标水平/垂直位置
33	00	设置鼠标位置	AL=04 CX/DX=鼠标水平/垂直位置	
33	00	设置图形光标	AL=09 BX/CX=鼠标水平/垂直中心 ES:DX=16x16 光标映像地址	安装了新的图形光标
33	00	设置文本光标	AL=0A BX=光标类型 CX=像素位掩码或起始的扫描线 DX=光标掩码或结束的扫描线	设置的文本光标
33	00	读移动计数器	AL=0B	CX/DX=鼠标水平/垂直距离
33	00	设置中断子程序	AL=0C CX=中断掩码 ES:DX=中断服务程序的地址	

附录 H 中断向量地址一览

附表 3-1 80×86 中断向量

I/O 地址	中断类型	功 能
0~3	0	除法溢出中断
4~7	1	单步（用于 DEBUG）
8~B	2	非屏蔽中断（NMI）
C~F	3	断点中断（用于 DEBUG）
10~13	4	溢出中断
14~17	5	打印屏幕
18~1F	6、7	保留

附表 3-2 8259 中断向量

I/O 地址	中断类型	功 能
20~23	8	定时器（IRQ0）
24~27	9	键盘（IRQ1）
28~2B	A	彩色/图形（IRQ2）
2C~2F	B	串行通信 COM2（IRQ3）
30~33	C	串行通信 COM1（IRQ4）
34~37	D	LPT2 控制器中断（IRQ5）
38~3B	E	磁盘控制器中断（IRQ6）
3C~3C	F	LPT1 控制器中断（IRQ7）

附表 3-3 BIOS 中断

I/O 地址	中断类型	功 能
40~43	10	视频显示 I/O
44~47	11	设备检验
48~4B	12	测定存储器容量
4C~4F	13	磁盘 I/O
50~53	14	RS-232 串行口 I/O
54~57	15	系统描述表指针
58~5B	16	键盘 I/O
5C~5F	17	打印机 I/O
60~63	18	ROM BASIC 入口代码
64~67	19	引导装入程序
68~6B	1A	日时钟

附表 3-4 提供给用户的中断

I/O 地址	中断类型	功 能
6C~6F	1B	Ctrl+Break 控制的软中断
70~73	1C	定时器控制的软中断

附表 3-3 BIOS 中断

I/O 地址	中断类型	功 能
80~83	20	DOS 中断返回
84~87	21	DOS 系统功能调用
88~8B	22	程序终止时 DOS 返回地址（用户不能直接调用）
8C~8F	23	Ctrl-Break 处理地址（用户不能直接调用）
90~93	24	严重错误处理（用户不能直接调用）
94~97	25	绝对磁盘读功能
98~9B	26	绝对磁盘写功能
9C~9F	27	终止并驻留程序
A0~A3	28	DOS 安全使用
A4~A7	29	快速写字符
A8~AB	2A	Microsoft 网络接口
B8~BB	2E	基本 SHELL 程序装入
BC~BF	2F	多路服务中断
CC~CF	33	鼠标中断
104~107	41	硬盘参数块
118~11B	46	第二硬盘参数表
11C~3FF	47~FF	BASIC 中断

附录 F 汇编出错提示信息

汇编程序对源程序进行汇编过程中遇到错误或可疑的语法时，计算机将会给出提示信息，以帮助调试程序。与特定代码相关的错误是编号的，与整个源程序相关；而不是某一特定代码相关的错误是不编号的。

附录 6.1 有编号的错误信息

有编号的错误信息的显示格式：

源文件(行) 代码 信息

1) 源文件，是出现错误的源程序文件名。若错误出现在内含文件的宏指令中，则源文件是引用该宏指令的文件，而不是定义该宏指令的文件。

2) 行，是出错位置在源程序中的行号。

3) 代码，是所有微软语言程序使用的标识代码，它由“error”或“warning”开头，后跟一个五字符代码。首字符指出程序是哪一种语言编写的，汇编源程序用 A 开头：后四个是数字字符，第一个数字说明警告级别，致命错误是 2，严重警告是 4，劝告性警告是 5；接着的三个数字是错误号。

4) 信息，是错误的解释与说明。

MASM 5.0 中有编号的错误信息提示如下：

0 Block nesting error, 块嵌套错误。

1 Extra characters on line, 在一行中除存在所定义语句的全部信息外，还有多余的字符，即用户提供了较多的操作数。

2 Internal error-Register already define symbol, 寄存器被定义为符号的内部错误。

3 Unknown type specifier, 未知的类型说明符。可能是拼错了类型说明符。

4 Redefinition of symbol, 符号在两处按不同类型定义。

5 Symbol is multidefined, 符号被多次定义。

6 Phase error between passes, 二义性的语句引起偏移地址在遍 1 和遍 2 之间改变。

7 Already had ELSE clause, 条件汇编块中用了多个 ELSE 子句。

8 Must be in conditional block, 无与 ENDIF 或 ELSE 匹配的 IF。

9 Symbol not defined, 符号无定义。

10 Syntax error, 句法错。相应语句不正确。

11 Type illegal in context, 类型说明符非法。

12 Group name must be unique, 组名已被定义为其他类型的符号。

13 Must be declared during pass 1: symbol, 符号应预先定义，然后再引用。

14 Illegal public declaration, 在 PUBLIC 语句中，使用了非法操作数。如使用寄存器名。

15 Symbol already different kind: symbol, 符号已经被定义为另一种符号。

16 Reserved word used as symbol: name, 保留字作符号，这是警告错误。

17 Forward reference illegal, 符号提前引用错误。

如顺序的两语句 DB COUNT DUP(?)与 COUNT EQU 10 将产生该错误。两语句调换先后

顺序，将不会产生这种错误。

- 18 Operand must be register: operand, 操作数应当是寄存器, 但却是一个符号或常量。
- 20 Operand must be segment or group, 操作数应当是段名或组名。提供的却是其他名字或常数。
- 22 Operand must be type specifier, 操作数应当是类型说明符。如 NEAR 或 WORD, 但给出的是其他操作数。
- 23 Symbol already defined locally, 在当前模块中已定义的符号又说明为 EXTERN 的操作数。
- 24 Segment parameters are changed, 同名段操作数说明有矛盾。语句要求应相同。
- 25 Improper align/combine type, SEGMENT 语句操作数错, 检查定位和组合类型。
- 26 Reference to multidefined symbol, 引用多重定义的符号。
- 27 Operand expected, 需要一个操作数, 但接收到的是操作符。
- 28 Operator expected, 需要一个操作符, 但给的是操作数。
- 29 Division by 0 or overflow, 表达式结果被零除或结果溢出。
- 30 Negative shift count, 作为 SHL/SHR 的操作数的表达式计算成一个负的移位次数。
- 31 Operand types must match, 操作数类型应当匹配。如 DATA 是字节变量, 则语句“MOVAX, DATA”会产生这个错误。为消除该错误, 上述语句应改为“MOV AX, WORD PTR DATA”。
- 32 Illegal use of external, 外部变量使用不正确。
- 34 Operand must be record or field name, 要求操作数是记录或域名。
- 35 Operand must have size, 要求操作数有指定的尺寸。如语句 INC[BX]将产生这个错误。
- 38 Left operand must have segment, 段取代前缀必须是段寄存器、组或段名。
- 39 One operand must be constant, 应当有一个常数操作数。
- 40 Operands must be in the same segment, or one must be constant, 操作数应在同一段或应有一个常数。
- 42 Constant expected, 希望一个常数。
- 43 Operand must have segment, 操作数应当有段。
- 44 Must be associated with data, 要求数据相关项的地方用了与代码相关的项。
- 45 Must be associated with code, 要求代码相关项的地方用了数据相关的项。
- 46 Multiple base registers, 操作数中使用了多个基址寄存器。如语句“MOV AX, [BX+BP]”非法。
- 47 Multiple index registers, 操作数中使用了多个变址寄存器。如语句“MOV AX, [SI+DI]”非法。
- 48 Must be index or base registers, 存储器操作数要求是基址或变址寄存器, 但给的是其他寄存器。
- 49 Illegal use of registers, 寄存器的使用非法。
- 50 Value out of range, 值超出范围。如语句“MOV AL, 5000”是非法的, 因字节寄存

器应使用字节值。

51 Operand not in current CS ASSUME segment, 操作数超出 ASSUME 语句分配的代码范围。通常是对标号的调用或转移超出当前代码段。

52 Improper operand type: symbol, 非法的操作数类型。如语句“MOV mem1, mem2”非法, 原因是两个均为存储器操作数。

53 Jump out of range by number bytes, 条件转移指令不在要求的范围之内。

55 Illegal register value, 非法使用寄存器。如语句“MOV AX, BP+4”非法, 正确语句为“MOVAX, [BP+4]”。

56 Immediate mode illegal, 立即操作数非法。如语句“MOVDS, 1000H”的立即操作数非法。

57 Illegal size for operand, 特定语句的操作数非法。如 INC MEM32, MEM32 为双字变量。

58 Byte register illegal, 8 位寄存器用在要求 16 位寄存器的语句中。如 PUSHAL 非法。

59 Illegal use of CS register, 非法使用 CS。如语句“POPCS”非法。

60 Must be accumulator register, 在应使用 AL、AX 的语句中使用了其他寄存器。

61 Improper use of segment register, 段寄存器使用非法。如语句 INCCS。

62 Missing or unreachable code segment, 企图转移到 MASM 不能识别为代码段的标号。通常是因为无 ASSUME 语句使 CS 与代码段相连造成的。

63 Operand combination illegal, 操作数组合非法。

64 Near JAMP/CALL to different code segment, 近程调用和跳转语句企图转移到非当前代码段。

65 Label cannot have segment override, 标号不允许有段前缀, 即段取代前缀使用有误。

66 Must have instruction after prefix, 前缀后应当有指令。

67 Cannot override ES for destination, 段取代前缀 ES 不能用在串操作指令的目的操作数上。

68 Cannot address with segment register, 企图访问存储操作数, 但没有用 ASSUME 说明该操作数所在的段。请使用段取代或 ASSUME 标示出该操作数的属性。

69 Must be in segment block, 应在段内使用的伪指令用在了段外。

70 Cannot use EVEN or ALIGN with type, EVEN 和 ALIGN 伪指令用在字节对准的段内。

71 Forward reference needs override or FAR, 调用或转移语句提前引用其他段的标号需使用 PTR 操作符予以说明。如 CALL FAR PTR TASK, 引用标号 TASK 时还未定义 TASK 标号。

72 Illegal value for DUP count, 重复子句的数值表达式不为正整数值。

73 Symbol is already external, 已定义为外部名字的符号又被定义为内部名字。

74 DUP nesting too deep, 重复子句嵌套超过 17 层。

75 Illegal use of undefined operand(?), “?”使用不正确。如“MOV AH, ?”和“5 DUP(?+5)”均非法。

76 Too many values for structure or record initialization, 初始化结构或记录变量时给出过

多初始值。

77 Angle brackets required around initialized list, 初始化结构变量时, 没有使用尖括号括起初始值。

78 Directive illegal in structure, 结构定义中的语句非法。

79 Override with DUP illegal, 初始化结构变量时, 使用了重复子句 DUP。

80 Field cannot be overridden, 初始化结构变量时, 企图预置不能被预置的域。

83 Circular chain of EQU aliases, 用 EQU 伪指令定义的符号名指向自身。如语句“A EQU B”和“B EQU A”非法, 将产生该错误。

84 Cannot emulate coprocessor or opcode, 协处理器指令或操作数与产生协处理器仿真程序不支持的操作码一起使用。

85 End of file, no END directive. 源程序无 END 语句。该错误也可能在段嵌套出错时产生。

86 Data emitted with no segment, 应在段内使用的语句在段外使用。产生目标代码的语句必须在段内, 不产生目标代码的语句可在段内, 也可在段外。

87 Forced error-pass1, 用.ERR1 伪指令强制形成的错误。

88 Forced error-pass2, 用.ERR2 伪指令强制形成的错误。

89 Forced error, 用.ERR 伪指令强制形成的错误。

90 Forced error-expression true(0), 用.ERRZ 伪指令强制形成的错误。

91 Forced error-expression false(not0), 用.ERRZ 伪指令强制形成的错误。

92 Forced error-symbol not defined, 用.ERRNDEF 伪指令强制形成的错误。

93 Forced error-symbol defined, 用.ERRDEF 伪指令强制形成的错误。

94 Forced error-string blank, 用.ERRB 伪指令强制形成的错误。

95 Forced error-string not blank, 用.ERRNB 伪指令强制形成的错误。

96 Forced error-strings identical, 用.ERRIDN 伪指令强制形成的错误。

97 Forced error-strings different, 用.ERRDIF 伪指令强制形成的错误。

98 Wrong length for override value, 结构域的重设值太大以至不适宜于这个域。

99 Line too long expanding symbol: symbol, 使用 EQU 伪指令定义的等式太长。

101 Missing data; zero assumed, 缺少操作数, 假定是零。

103 Align must be power of 2, ALIGN 伪指令用了不是 2 的幂的参数。

104 Jump within short distance, JMP 指令的转移范围在短标号内, 故可在标号前加 SHORT 操作符, 从而使指令代码减少一个字节。

105 Expected element, 少了一个元素, 如标点符号或操作符。

106 Line too long, 源行超过 MASM 允许的最大长度。MASM5.0 规定行长为 128 个字符。

107 Illegal digit in number, 常数内包含当前的基不允许的数字, 如 108Q。

108 Empty string not allowed, 空串不允许出现。如“NULLDB”””语句非法。

109 Missing Operand, 语句缺少一个必需的操作数。

110 Open parenthesis or bracket, 语句中缺少一个圆括号或方括号。

111 Directive must be in macro, 只在宏定义里面使用的伪指令用在宏定义之外。

112 Unexpected end of line, 语句行不完整。

附录 6.2 无编号的错误信息

无编号错误信息提示在命令行、存储器分配或文件访问中出现错误。

(1) 命令行错误

MASM 时, 若用户给出无效命令行, 将产生如下提示信息之一:

Buffer size expected after B option

Error defining symbol "name" from command line

Extra file name ignored

Line invalid, start again

Path expected after I option

Unknown case option: option

Unknown option: option

(2) 文件访问错误

当 MASM 处理文件时, 若出现磁盘空间不够、错误文件名或其他文件错误, 则将产生如下提示信息之一:

End of file encountered on input file

Include file filename not found

Read error on standard input

Unable to access input file: filename

Unable to open cref file: filename

Unable to open input file: filename

Unable to open listing file: filename

Unable to open object file: filename

Write error on cross-reference file

Write error on listing file

Write error on object file

(3) 其他错误

表示存储分配出错或其他一些与特定源程序的行不相关的汇编出错等提示信息。

Internal error, 内部出错, 注意错误发生时的情况。

Internal error-problem with expression analyzer 分析表达式出现内部错误, 说明表达式构成有误。

Internal unknown error, MASM 不能识别的内部表有错。

Number of open conditionals: (number)有 IF 而没有 ENDIF 语句。

Open procedures, 无与 PROC 匹配的 ENDP 语句。

Open segments, 无与 SEGMENT 匹配的 ENDS 语句。

Out of memory, 存储器有效空间用完, 可能是源文件太长或符号表中定义了太多符号。

