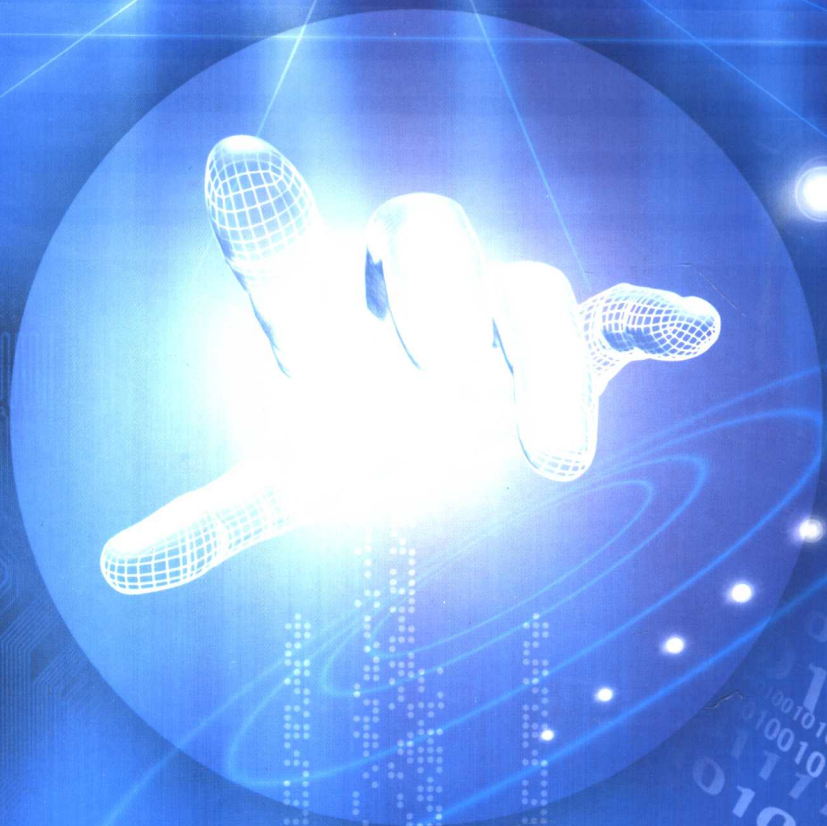
DEBUG与软件维修技术

主编　孙维连　刘涛



哈尔滨工程大学出版社

DEBUG 与软件维修技术

主　编　孙维连　刘　涛
副主编　息明东　郑佳昕
主　审　戚常林

哈尔滨工程大学出版社

图书在版编目(CIP)数据

DEBUG与软件维修技术/孙维连,刘涛主编.—哈尔滨:哈尔滨工程大学出版社,2005.1
ISBN 7-81073-653-1

Ⅰ.D… Ⅱ.①孙…②刘… Ⅲ.软件维护 Ⅳ.TP311.53

中国版本图书馆CIP数据核字(2004)第141323号

内 容 简 介

本书根据编者多年的教学与实践经验,深入讨论了Windows 98 DOS系统中鲜为人知的重要数据结构及参数,并以DEBUG下的汇编语言为工具,通过大量实例,给出如何应用DOS的功能调用、DOS中断、BIOS中断、端口技术、硬盘扇区读写技术、DOS内部数据结构去解决实践中所遇到软件维护维修等方面的技术难题。最后给出十余个C语言开发维修工具的实例。

本书有很强的技术性与实用性,可供软件编程人员、大中专院校师生及计算机爱好者参考使用,是一本比较完善的计算机软件维修参考书。

哈尔滨工程大学出版社出版发行
哈尔滨市南通大街145号 哈工程大学11号楼
发行部电话:(0451)82519328 邮编:150001
新 华 书 店 经 销
黑龙江省教育厅印刷厂印刷
*
开本 787mm×1 092mm 1/16 印张 20.5 字数 498千字
2005年1月第1版 2006年1月第2次印刷
印数:1 001—3 000册
定价:25.00元

前　言

在计算机应用过程中，用户难免会遇到许多技术问题。这些问题的解决，往往需要查阅大量资料及相关程序。但由于资料不足或短时间内找不到相关程序，使问题不能迅速得到解决，进而影响计算机的使用效果。在用户遇到的这些问题当中，大部分与软件维修技术相关。所以，为使用户能迅速地解决这些问题，编者根据多年的教学与实践研究及软件维修经验编成此书，奉献给广大读者。

本书以 DEBUG 下的汇编语言为工具，通过大量实例，深入剖析了 Windows 98 内部数据结构、参数及各种 DOS 功能调用、DOS 中断、BIOS 中断、端口技术，并给出应用技巧。目前计算机真正的硬件故障不多，但由于软件原因造成的计算机不能正常工作，或对软件故障处理不当，造成重大损失的情况则随处可见。为此，本书又列举了大量有价值的实例，对计算机软件故障的排除予以详尽分析，并给出具体的解决方法及步骤，如：硬盘、软盘软件故障的排除。书中又对计算机的安全技术问题，予以详尽讨论，并给出与计算机安全相关的各种方法策略及具体的计算机软件安全防护措施。

本书针对计算机常用的存储设备、软盘、硬盘及文件系统进行深入剖析，并给出十余个 C 语言开发维修工具的实例，应用这些程序，可实现存储软盘、硬盘、文件等中的数据，就像看书一样，任我们随意去翻阅每一页（扇区）的单词、语句，并根据需要进行浏览、修改或重写。

根据应用技术要求深入剖析了大容量硬盘的多种读写技术，并提供维护软盘、硬盘多种技术手段，对软盘、硬盘数据丢失，给出多种恢复技术的实践操作。通过对软盘、硬盘结构的分析，给出对硬盘引导扇区、分区表、分区引导扇区丢失或损坏的修复技术，对不能启动的硬盘上恢复文件的方法，对硬盘的保护，对 FAT16、FAT32 文件系统，都进行了详尽剖析，为解决许多实际问题列举了大量实用性极强的例题。

编者竭力想奉献给读者一本既有价值，又有特色的软件维修工具书，但时间仓促，水平所限，书中错误在所难免，敬请广大读者批评指正，愿与广大读者相互交流，提高软件维修的理论与实践技术水平。

全书由孙维连、刘涛任主编，息明东、郑佳昕任副主编，贺雷参编。第一章、附录由息明东编写，第二章由刘涛编写，第三章由郑佳昕编写，第四章、第五章由孙维连编写，全书由孙维连统稿。戚常林教授对此书进行了审阅，编者对参与支持本书的人员及主审表示衷心感谢。

编　者

2004 年 8 月

目　录

第 1 章　调试程序 DEBUG …… 1
1.1　前言 …… 1
1.2　怎样启动 DEBUG 程序 …… 1
1.3　汇编与反汇编命令 …… 3
1.4　显示与修改内存单元内容的命令 …… 7
1.5　显示与修改寄存器内容命令 …… 9
1.6　运行和跟踪命令 …… 12
1.7　磁盘文件与扇区的读写命令 …… 17
1.8　有关内存单元的几个命令 …… 21
1.9　DEBUG 的其它命令 …… 24
1.10　简单的汇编语言程序怎样在 DEBUG 下直接输入 …… 25
1.11　根据 DEBUG 下的程序清单及字符串,如何建立执行程序 …… 27
第 2 章　DOS 中断与 BIOS 中断应用技术 …… 31
2.1　DOS 功能调用及应用 …… 31
2.2　修改执行文件,扩展其功能应用实例 …… 58
2.3　BIOS 中断与 DOS 中断应用技术 …… 67
2.4　与端口直接相关的应用技术 …… 85
第 3 章　DOS 的内部结构浅析与内存管理 …… 89
3.1　DOS 的组成、功能与启动 …… 89
3.2　DOS 的文件管理 …… 93
3.3　DOS 控制块和工作区域 …… 114
3.4　系统综合应用举例 …… 119
3.5　内存管理应用实例 …… 150
3.6　软盘维护技术与读写故障排除 …… 158
3.7　DOS 系统内部多重表的应用 …… 168
第 4 章　硬盘管理及维护 …… 176
4.1　DOS 在硬盘上的存放及硬盘的体系结构 …… 176
4.2　硬盘主引导扇区剖析 …… 179
4.3　硬盘分区引导扇区剖析 …… 189
4.4　硬盘维护技术 …… 198
4.5　硬盘维护技术综合实例 …… 242
4.6　计算机病毒的检测、清除及预防 …… 275
第 5 章　TC2.0 实用工具开发技术 …… 286
5.1　应用基本 INT 13H 读写 8GB 以下硬盘的开发技术 …… 286
5.2　应用扩展 INT 13H 读写大硬盘的开发技术 …… 290

附录 1 ASCII 码字符表 …… 305
附录 2 CMOS 数据格式 …… 306
附录 3 部分 BIOS 数据区 …… 308
附录 4 部分 BIOS 中断功能调用表 …… 310
附录 5 部分中断向量表 …… 317
参考文献 …… 319

第 1 章　调试程序 DEBUG

1.1　前　　言

DEBUG 是一个 DOS 实用程序，是软件工作人员在调试程序时经常使用的工具。DEBUG 的功能很强，它包含 19 个子命令。这些子命令不仅能跟踪运行程序的踪迹，以便了解程序中每条指令的执行结果，从而分析程序流程的正确性，而且能直接与磁盘文件或它的指定扇区进行对话，以便读写磁盘文件或它的某个扇区的具体内容，为显示、分析和修改磁盘文件提供了方便。除此之外，还可以显示和修改指定范围的内存内容，以及 CPU 内部寄存器和标志位的内容等。

因此 DEBUG 程序可用于：

1.提供一个可控制的测试环境，让用户能监视和控制被调试程序的执行情况，还可在程序中直接指定问题，然后立即执行它，并判定所指问题是否已经解决，这使得不必重新汇编程序就可以看所作的修改是否有效；

2.装入、修改或显示任何文件(EXE、HEX 文件需改名)；

3.执行目标文件，目标文件是用机器语言格式表示的可执行程序；

4.检查、搜索内存，及内存映像存盘；

5.检查、修改磁盘扇区中的内容；

6.修改文件分配表 FAT、目录项表 FDT；

7.用 DEBUG 汇编命令或通过 DOS 重定向输入文本文件去建立短小 COM 文件；

8.恢复内存中的文本文件或二进制文件；

9.恢复被删除的文件；

10.对文件名、子目录名进行修改及加密或解密；

11.清除计算机病毒；

12.修复磁盘被破坏的文件。

本章先介绍如何启动 DEBUG 程序，以及有关 DEBUG 命令的通用信息，然后按功能分类逐一详细介绍它的 19 个子命令的用途、格式及有关的示例，最后综合举例说明这些命令的使用。

1.2　怎样启动 DEBUG 程序

1.2.1　DEBUG 程序的启动

在 DOS 提示符 A > 下，键入如下的命令：

A > DEBUG [d:][path][filename[.exe]][parm1][parm2]

其中DEBUG是调试程序的文件名,后面跟的是被调程序的文件说明。如果输入了filename,DEBUG程序就把该指定文件装入内存。此后,就可以打入命令修改、显示或执行所指定文件的内容。

如果没有输入filename,则只能把当前内存中的内容进行修改、显示或执行。或者可用命名命令Name或装入命令Load将所需要的文件装入内存,然后就可以使用命令对内存中的内容进行修改、显示和执行。

可选参数parm1和parm2,表示命名文件说明时可选择的参数。例如:

DEBUG DISKCOMP.COM A: B:

在这个命令中,A:和B:是DEBUG为DISKCOMP程序准备的参数。

当DEBUG启动成功后,发出提示符"-"(短横线)。这说明现在系统在DEBUG程序的管理之下。所有的DEBUG命令,只有跟在提示符"-"后键入才有效。

1.2.2 DEBUG程序初始化

当启动DEBUG时,对要调试的程序设置寄存器和标志位为下面的数值。

1.四个段寄存器(CS,DS,ES和SS)置于内存空闲块的底部,即DEBUG程序结束以后的第一个段地址。

2.指令指针IP置为100H。

3.堆栈指针(SP)置到该段的结尾处,或者是装入程序的临时底部,取决于哪一个较低。

4.其余的寄存器(AX,BX,CX,DX,BP,SI和DI)置为0。然而当启动DEBUG时使用一个要调试程序的文件标识符,则CX中包含以字节表示的该文件的长度;若文件大于64K,则该文件长度包含在BX和CX中(高位在BX中)。

5.设置标志寄存器为被清除的状态。

6.约定的磁盘缓冲区置于代码段80H处。

注意,若由DEBUG调入的程序,具有扩展名EXE,则DEBUG必须进行再分配,把段寄存器、堆栈指针置为文件中规定的值。

经过上述初始化后,在屏幕上显示DEBUG的提示符"-",等待键入命令。

1.2.3 有关DEBUG命令的一些通用信息

在详细介绍DEBUG命令以前,现将有关它的通用信息介绍如下:

1.DEBUG的每个命令是一个字母,通常后面有一个或多个参数;

2.命令和参数可以用大写、小写或大小写混合方式输入;

3.命令和参数可以用定界符(空格或逗点)分隔开,然而,只有在两个连续的十六进制数之间必须使用定界符,因此,下面的命令是等价的:

DCS:100 110

DCS:100 L10

D,CS:100,110

4.可以用Ctrl-Break键来停止一个命令的执行,返回DEBUG提示符;

5.每个命令只有在按了Enter键之后,输入命令才起作用,并开始执行;

6.对于产生大量输出的命令,可用Ctrl-Numlock键去暂停显示,以便在这些输出从屏幕

上滚动消失之前看清楚,打入任何键可重新继续下面的显示;

7.在使用 DEBUG 程序期间,能使用控制键和 DOS 编辑键;

8.如果遇到一个语法错误,则显示出该行并指出错误之所在,如

DCS:100 CS:110

^Error

在这个例子里,显示命令期望第二个地址里仅包括一个十六进制位移值,但当它发现 c 时,识别出它不是一个十六进制数,故出错。

1.2.4 DEBUG 命令中的地址和地址范围参数

DEBUG 命令中的参数多,现将常用的地址和地址范围参数介绍如下:

DEBUG 中的地址格式为:[< 段地址 > :] < 位移量 > 。据此,它有如下的三种形式:

1.用字符表示的段寄存器名称,加上一个位移值,其间用冒号隔开,例如 CS:100

2.一个段地址,加上一个位移量,其间仍用冒号隔开,如 4BA:0100

3.只有一个位移值(段地址用约定值),如 100,此时对子命令 A、G、L、T、U 和 W,约定段为 CS,对所有其它子命令的约定段为 DS。

为确定一个地址范围的低地址和高地址,可输入下面两种格式之一:

(1) < 段地址 > : < 始位移量 > < 终位移量 > 如 CS:100 110

其中第二地址仅用位移量,且与前面地址需用空格符分隔开。

(2) < 段地址 > : < 始位移量 > < 长度 > 如 CS:100 L11

在这里,要注意,所有的数字值都可为 1 至 4 个字符的十六进制值,而且地址和地址范围指定的内存位置必须是内存实际存在的。如果企图去存取一个不存在的存贮单元,将会产生预想不到的后果。

以下各节介绍 DEBUG 的子命令及其应用。

1.3 汇编与反汇编命令

1.3.1 汇编命令 A(Assemble Command)

用途:把 8086/8087/8088 助记符直接汇编到存贮器内。

格式:A[< 地址 >]

说明:汇编命令 A 将用户输入的汇编语句从[< 地址 >]参数所指定的地址开始汇编到存贮器中连续的单元里以得到可执行的机器码。例如:

```
A > DEBUG
 - A200
08B4:0200 XOR AX,AX
08B4:0202 MOV [BX],AX
08B4:0204 RET
08B4:0205 < 按 ENTER 键 >
```

若在命令中没有指定地址,但前面用过汇编命令,则接上一个汇编命令的最后一个单元

开始存放;若前面没有用过汇编命令,则从CS:100单元开始连续存放。

当所有需要的语句都输入完后,这时提示输入下一个语句,在此提示下打入ENTER键作为响应,于是就返回到DEBUG提示符。

注:对于DEBUG在装入EXE文件后,首次使用A命令,且后面没有指定地址,则从CS:IP开始,连续存放程序。若指定地址,则从指定地址开始存放。

```
A > DEBUG ASM.EXE
-A <按 ENTER 键>
XXXX:XXXX(CS:IP)MOV AX,0
如:-A5000:0
   5000:0 MOV AX,1
又如:-A1000
      CS:1000 开始
```

若输入语句有错,DEBUG就显示:^Error并重新显示当前汇编行的地址,等待新的键入。DEBUG支持标准的8086/8088汇编语言的语法(和8087指令系统)并具有以下的一些规则:

1.所有输入的数字值全为16进制数,可输入1到4个16进制数字字符;

2.前缀助记符,必须在相关指令之前输入,也可另起一行;

3.段超越助记符为CS:,DS:,ES:和SS:;

4.字符串操作助记符必须明确地说明字符串长度,例如,MOVSW必须用于去传送字串,而MOVSB必须用于去传送字节串;

5.汇编程序将自动地汇编短转移、近转移或远转移和调用,这取决于对目标地址的字节位移,也能用NEAR和FAR前缀取代它们,例如:

```
0100:0500 JMP 520          ;二字节的短转移指令
0100:0502 JMP NEAR 505     ;三字节的近转移指令
0100:0505 JMP FAR 50A      ;五字节的远转移指令
```

6.交叉段远返回的助记符是RETF;

7.DEBUG无法说明一些操作数是属于字存贮单元还是字节存贮单元,必须用前缀"WORD PTR"(可缩写为"WO")或"BYTE PTR"(可缩写为"BY")来明确说明数据的类型,例如:

```
NEG BYTE PTR[128]
DEC WO[SI]
```

8.DEBUG也无法说明操作数是属于存贮单元还是属于立即数,因此DEBUG中把存贮单元的地址放在括号中,例如:

```
MOV AX,21          ;把21H送入AX中
MOV AX,[21]        ;把地址为21H以及21H+1的存贮单元的内容送至AX
```

9.也能包含两个常用的伪指令:DB操作码将把字节数值直接汇编到存贮器内;DW操作码将把字的数值直接汇编到存贮器内。例如:

```
DB 1,2,3,4 "THIS IS AN EXAMPLE"
DW 1000,2000,3000,"BACH"
```

10.DEBUG支持所有形式的寄存器间接寻址命令,例如:

```
ADD BX,34[BP+2][SI-1]
```

POP [BP + DI]

11.DEBUG 支持所有操作码的同义词,例如:

LOOPZ 100

JA　200

A 命令主要用于小段程序的汇编或者调试程序中发现其中一部分需要改写,或要增补一段等,就可直接在 DEBUG 下编程序,变成机器码并调试和运行,亦可存入磁盘。这样就省去了调编辑程序、汇编程序、连接程序的复杂手续,给用户提供了方便。下面是说明 A 命令使用的例子。

例 1-1　数据块传送(将 200 开始的 16 个字节单元的数据传送到 220 开始的单元中去)。

```
A > DEBUG
- A100
08F8:0100 MOV SI,0200
08F8:0103 MOV DI,0220
08F8:0106 MOV CX,10
08F8:0109 REPZ MOVSB
08F8:010B INT 3
08F8:010C
- A200
08F8:0200 DB 60,61,62,63,64,65,66,67,68,69,70,71,72,73,74,75
- G = 100
- D220 L10
08F8:0220 60 61 62 63 64 65 66 67 - 68 69 70 71 72 73 74 75
```

例 1-2　读软盘 A:DOS 引导区。

```
- A100                          ;从 100H 输入程序
29EC:0100 MOV AX,0201          ;读一个扇区
29EC:0103 MOV BX,0300          ;将扇区内容装入内存偏移地址 300H 处
29EC:0106 MOV CX,0001          ;从 0 道 1 扇区开始读
29EC:0109 MOV DX,0000          ;从 0 头 A:盘开始读
29EC:010C INT 13               ;磁盘 I/O 中断
29EC:010E INT 3                ;断点中断
29EC:010F
- G = 100                      ;执行程序
AX = 0001  BX = 0300  CX = 0001  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
DS = 29EC  ES = 29EC  SS = 29EC  CS = 29EC  IP = 010E  NV UP EI PL ZR NA PE NC
29EC:010E CC              INT 3
- D300 L200                    ;显示读入内存中的内容
29EC:0300  EB 3C 90 4D 53 44 4F 53 - 35 2E 30 00 02 02 01 00  .<.MSDOS5.0.....
29EC:0310  02 70 00 D0 02 FD 02 00 - 09 00 02 00 00 00 00 00  .p..............
```

```
29EC:0320  00 00 00 00 00 00 29 E9-0F 67 41 4E 4F 20 4E 41   ......)..gANO NA
......
29EC:04D0  20 61 6E 79 20 6B 65 79-20 77 68 65 6E 20 72 65   any key when re
29EC:04E0  61 64 79 0D 0A 00 49 4F-20 20 20 20 20 20 53 59   ady...IO      SY
29EC:04F0  53 4D 53 44 4F 53 20 20-20 53 59 53 00 00 55 AA   SMSDOS    SYS..U.
-U100 LF                          ;将输入程序反汇编
29EC:0100 B80102          MOV AX,0201
29EC:0103 BB0003          MOV BX,0300
29EC:0106 B90100          MOV CX,0001
29EC:0109 BA0000          MOV DX,0000
29EC:010C CD13            INT 13
29EC:010E CC              INT 3
                  ——————————→反汇编即从机器码翻译出助记符
                  ←——————————汇编即从助记符翻译出机器码
-Q
```

1.3.2 反汇编命令 U(Unassemble Command)

什么叫反汇编？把内存中的机器码翻译成汇编指令符号形式，列在显示终端或打印机上(当打印机接通时)，以帮助用户了解内存中存的是什么程序，这就叫做反汇编。

因此，U 命令可以对二进制代码程序作反汇编，以帮助分析和调试目标程序。反汇编有两种常用的格式选择。

选择 1：它可以不指定地址去对指令进行反汇编，也可以指定一个起始地址去对指令进行反汇编，其形式是：

U < 地址 > 或 U

它们都能(在 40 列的显示屏上)反汇编 16 个字节或(80 列的显示屏上)反汇编 32 个字节。若指定了地址，则从指定地址开始，对指令进行反汇编；若没有指定地址，则 U 命令假定起始地址是上次 U 命令反汇编过的最后一条指令后面的地址。因此，输入没有参数的连续的 U 命令能反汇编连续的地址，并产生连续的反汇编显示。

若前面没有输入过 U 命令，则从 DEBUG 初始化的段寄存器的值作段地址，以 100H 为地址偏移值。

选择 2：是对指定地址范围内的指令进行反汇编，其格式为：

U < 地址范围 >

例如，为检验 A 命令中输入的程序是否正确，可以用 U 命令反汇编出：

```
-U CS:0110 011B
08F8:0110  BE0001 MOV  SI,0100
08F8:0113  BF0002 MOV  DI,0200
08F8:0116  B91000 MOV  CX,0010
08F8:0119  F3     REPZ
08F8:011A  A4     MOVSB
```

08F8:011B F4 HLT

也可以由始址和长度来表示地址范围,如下的命令:

-U CS:0110 LC

得到与上一个命令相同的效果。因此,若知道文件长度(字节数),使用此命令是比较方便的,它能一次把整个程序都反汇编出来,但由第一节知道,文件长度的16进制数的高位在BX中,低位在CX中,通过R命令即可找到这些寄存器值。

1.4 显示与修改内存单元内容的命令

1.4.1 显示内存单元内容的命令D(Dump Command)

使用D命令可以显示指定内存单元的内容,对于了解程序执行结果以及检查内存单元所装的内容是什么都是十分重要的。显示命令的格式为:

D[<地址>]

或者

D[<地址范围>]

这样,显示命令有两种常用的格式选择。

选择1:显示指定地址单元的内容,其格式为:

D<地址>或者D

则从指定地址开始,显示40H个字节(相应于系统的40列显示格式)或80H个字节(相应于80列显示格式)

如果没有指定地址,则D命令使用约定值,即前面的D命令显示最后一个单元后面的地址。如果以前没有输入D命令,那么约定地址是DEBUG初始化的寄存器的内容,加上偏移量100H,因此,通过不带有参数的连续的D命令,可能转贮内存连续的40H个字节或80H个字节区域的内容。

注意:如果始址只键入偏移量值,则D命令认为段地址包含在DS中。例如:

```
A>DEBUG DEBUG.COM          内存相应单元的机器码                    机器码所对应的字符
-D 100
118F:0100  E9 84 3D 43 6F 6E 76 65-72 74 65 64 00 00 00 00   ..=Converted....
118F:0110  4D 5A 6F 01 1F 00 03 00-20 00 00 00 FF FF 62 03   MZo.........b.
118F:0120  6A 01 81 5E 00 01 28 00-1E 00 00 00 01 00 DE 03   j..^..(.........
118F:0130  28 00 E5 03 28 00 EE 03-28 00 00 00 00 00 00 00   (...(...(.......
段地址:偏移地址                                    00--1F,7F--FF均用小数点表示
```

以上所示内存部分区域的内容由两部分组成,第一部分在显示行的中部,用两位16进制字符数显示内存区每个字节的内容,字节间由空格分开,最左边是所显示行首字节的地址,以段:位移量表示之。另一部分在显示行的右边,是该行显示的相应字节的ASCII码,字符若没有可见的相应ASCII字符,则用句点“.”表示。

显示还有两种格式,若是40列系统显示格式,每一行在8个字节界限上开始,并显示8

个字节。若是80列系统显示格式,每一行在16个字节界限上开始并显示16个字节,在第8和第9个字节之间,有一个连字符。

如果显示地址不在一个界限上,那么第一行可能少于8个或16个字节,在这种情况下,显示的第二行在下一个界限上开始。

选择2:显示地址范围内容,其格式为:

D<地址范围>例如命令:

```
-D CS:100 10C
118F:0100  E9 84 3D 43 6F 6E 76 65-72 74 65 64 00              ..=Converted.
```

注意:在地址范围中包含起始和结束地址,如果对开始地址只输入一个偏移值,那么D命令认为段地址在DS中,而输入的结束地址中只能允许有地址偏移值。

下面是80列系统显示格式的一个例子,其说明类似于选择1所述。

```
A>DEBUG DEBUG.COM
-D100 L20
118F:0100  E9 84 3D 43 6F 6E 76 65-72 74 65 64 00 00 00 00  ..=Converted....
118F:0110  4D 5A 6F 01 1F 00 03 00-20 00 00 00 FF FF 62 03  MZo.........b.
-Q
```

1.4.2 修改内存单元内容的命令E(Enter Command)

此命令用于修改内存单元的内容,它有两种基本格式。

选择1:用命令中给定的内容表去代替指定范围的内存单元的内容:

E<地址>[<内容表>]

其中内容表为一系列用空格隔开的16进制字节,或者是用引号括起的字符串,例如:

EDS:100 F3 "XYZ" 8D

则内存单元DS:100到DS:104共5个存贮单元的内容,由表中指定的5个字节所代替,其中两个用两位16进制数表示;如F3,8D;另外三个用字符表示,即用它们的ASCII值代入。以上替代,实则是修改内存单元的内容,因此不管原内存是什么,即用新的值加以取代。

例如,先用E命令修改内存:

```
-E 212 00 "END OF PROGRAM"
```

再用D命令显示内存:

```
-D 212 220
08F8:0212  00 45 4E 44 20 4F-46 20 50 52 4F 47 52 41        .END OF PROGRA
08F8:0220  4D                                               M
```

选择2:用来显示其地址和一个单元的字节,然后等待用户键入。例如:

E<地址>

在输入了上述命令,屏幕上显示指定单元的地址和原有的内容之后,用户可以采取以下几种操作中的一种:

1.输入一个或两个字符的16进制数,去代替这个字节的内容,然后执行下面的任一个操作。

2.按下空格键,显示下一个地址及其内容。如果想修改这个内容,那么再执行上述第一

步骤的操作;如果不想修改,再按一次空格键,这样就可以连续不断地进行修改操作。

3.按下一个连字符"-",则返回到前面的地址,在新的一行上显示前面的地址和它的内容。如果想修改,则输入一个字节的内容,然后再按连字符,则又在新行上显示前一个单元的地址和内容……,这样,就可以连续不断地进行反向修改。若所显示前一单元的内容不需要修改,就直接按连字符。

4.按 Enter 键,结束 E 命令。例如:

E CS:100

可能产生这样的显示:

04BA:0100 EB. -

为了把 04BA:0100 的内容由 16 进制的 EB 改为 16 进制的 41,输入 41

04BA:0100 EB.41 -

为了显示其后三个单元的内容,按空格键三次,这时屏幕上可能是这样:

04BA:0100 EB.41 10 00 BC -

为了把当前单元(04BA:0103)的内容由 16 进制的 BC 改为 16 进制的 42,则输入 42

04BA:0100 EB.41 10 00 BC 42 -

现在假设想返回将 10H 改为 6FH,在输入完两个连字符和替换字节之后,这时屏幕上看起来是这样:

04BA:0100 EB.41 10 00 BC 42 -

04BA:0102 00 -

04BA:0101 10.6F -

按下 Enter 键结束 E 命令,用户将看到提示符"-"出现。

例 1-3 建立数据文件。

```
A > DEBUG
- E100 'ABCDEFGH' D A 'IJKLMNOP' D A
- RCX
CX 0000
:14
- N STR
- W
- Q
A > TYPE STR
ABCDEFGH
IJKLMNOP
A >
```

1.5 显示与修改寄存器内容命令

为了了解程序运行的正确性,常常需要检查寄存器的内容,R 命令就是用来显示和修改寄存器内容的,它有以下三种功能:

1.能显示和修改一个指定寄存器的内容,其格式为:R < 寄存器名 > ;

2.能显示 CPU 内部的所有寄存器的内容和全部标志位的状态,其格式为 R;

3.能显示和修改所有标志位的状态,其格式为 RF。

下面分别予以说明。

1.5.1 显示和修改指定寄存器的内容

其命令格式为:

R < 寄存器名 >

其中寄存器名是以下的寄存器才是有效的:

AX BX CX DX SP BP SI DI

DS ES SS CS IP PC F

其中 IP 和 PC 都属于指令指针。例如,如果要显示单个寄存器的内容,可以输入:

R AX

系统可能应答为:

AX F1E4

:-

现在,可以采取以下两种操作中的一种:

按下 Enter 键,保留其内容不变,或者是输入 1 至 4 个字符的 16 进制数值来改变 AX 寄存器的内容,例如输入 16 进制的 FFF。

AX F1E4

:FFF-

现在按下 Enter 键,便把 AX 寄存器的内容修改为 16 进制的 0FFF。对其它寄存器内容的显示和修改,方法同上。

例 1-4 将例 1-3 中的文件 STR 读到内存。

```
A > DEBUG STR
- RCX
CX 0014              ;CX 表示文件长度的低 16 位
:
- Q
```

例 1-5 修改寄存器的值,为指令运行提供原始值。

```
A > DEBUG
- A100
XXXX:0100 MOV BX,AX
XXXX:0102 INT 3
XXXX:0103
- RAX
AX 0000
:4567                ;为寄存器 AX 赋值 4567
- G = 100
AX = 4567  BX = 4567  CX = 0000  DX = 0000  SP = FFEE  BP = 0000  SI = 0000  DI = 0000
```

DS=29EC ES=29EC SS=29EC CS=29EC IP=0102 NV UP EI PL ZR NA PE NC
29EC:0102 CC INT 3

1.5.2 显示所有寄存器和标志位

其命令格式为:

-R

则系统的响应为:

AX=0E00 BX=000F CX=0007 DX=01FF SP=039D BP=0000 SI=005C DI=0000
DS=04BA ES=04BA SS=04BA CS=04BA IP=011A NV UP DI NG NZ AC PE NC
04BA:011A CD21 INT 21

这是一个80列显示的系统表示,其中第一行显示8个寄存器的内容,第二行显示5个寄存器内容和置成的8个标志位的状态,第三行显示了现行的CS:IP所指的指令的机器码和反汇编符号,这就是下一条将要执行的指令。

例1-6 在DEBUG下测试文件长度超过64K时CX、BX的值。

A>COPY MEM386.EXE MEM
A>DEBUG MEM
-R
AX=0000 BX=0001 CX=D85E DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=29EC ES=29EC SS=29EC CS=29EC IP=0102 NV UP EI PL ZR NA PE NC

从以上可知该文件的长度为BX(文件长度高16位),CX(文件长度低16位)=0001D85E=1D85E。

1.5.3 显示和修改标志位状态

在8088中,共有九个标志位,其中追踪标志T不能直接用指令改变,其它8个标志可以显示和修改。显示时每个标志位由两个字母组成,它说明是"置位"状态(SET)还是"复位"状态(CLEAR),并且显示的次序和符号如下表所示:

标志位名字	置位(SET)	复位(CLEAR)
溢 出 Overflow(Yes/no)	OV	NV
方 向 Direction(增量/减量)	DN	UP
中 断 Interrupt(允许/禁止)	EI	DI
符 号 Sign(负的/正的)	NG	PL
零 标 志 Zero (Yes/No)	ZR	NZ
辅助进位 Auxiliary Carry(Yes/No)	AC	NA
奇偶校验 Parity(偶/奇)	PE	PO
进 位 Carry(Yes/No)	CY	NC