8086/8088 HE ARM HE HUIBIAN YUYAN CHENGXU SHEJI SHIYAN JIAOCHENG

8086/8088和ARM核汇编语言程序设计实验教程

主　编　李敬兆

副主编　丁　刚　张　雷　谢晓东　詹　林

中国科学技术大学出版社

8086/8088 和 ARM 核汇编语言程序设计实验教程

主　编　　李敬兆

副主编　　张　雷　谢晓东

丁　刚　詹　林

中国科学技术大学出版社

2006 · 合肥

内 容 简 介

汇编语言是提供给用户直接访问计算机系统最快而又最有效的一种编程语言,使用汇编语言编写程序能够充分发挥计算机硬件系统的功能,那些需要对计算机硬件进行控制或对运行时间和效率有要求的系统软件或应用软件，通常都是用汇编语言编写而成的，因此熟练掌握汇编语言程序设计方法是非常重要的。

本书是《8086/8088 和 ARM 核汇编语言程序设计》教材的配套实训教程，共 11 章 20 个实验。目的是使学生通过实验加深对理论课程的理解，增强学生的实际动手能力和实践应用能力。

图书在版编目（CIP）数据

8086/8088 和 ARM 核汇编语言程序设计实验教程/李敬兆主编.—合肥：中国科学技术大学出版社，2006.9

ISBN7-312-01981-1

Ⅰ. 8… Ⅱ. 李 Ⅲ. ①微处理器，ARM-系统设计-高等学校-教学参考资料②汇编语言-程序设计-高等学校-教学参考资料 Ⅳ.①TP332 ②TP312

中国版本图书馆 CIP 数据核字（2006）第 086375 号

中国科学技术大学出版社出版发行

（安徽省合肥市金寨路 96 号，邮政编码：230036）

中国科学技术大学印刷厂印刷

全国新华书店经销

开本：787×1092/16　　印张：5.5　　字数：140 千

2006 年 9 月第 1 版　　2006 年 9 月第 1 次印刷

印数：1—4000 册

ISBN7-312-01981-1/TP・371　　定价：10.00 元

编 委 会 名 单

主　任：汪光阳

副主任：周鸣争　李敬兆　方潜生

委　员：（以姓氏笔划为序）

丁　刚　方潜生　纪　平　江　静

汪光阳　李敬兆　陈　辉　张　雷

周鸣争　宗欣欣　徐　辉　夏　巍

谢晓东　詹　林

前 言

汇编语言是提供给用户直接访问计算机系统最快而又最为有效的一种编程语言，使用汇编语言编写程序能够充分发挥计算机硬件系统的功能，具有占用存储空间少、运行速度快以及代码质量高等优点，那些需要对计算机硬件进行控制或对运行时间和效率有要求的系统软件或应用软件，通常都是用汇编语言编写的，因此熟练掌握汇编语言源程序的设计方法是非常重要的。

目前，基于80X86系列微处理器的PC机和基于ARM核微处理器的嵌入式系统应用最为广泛，本书是与《8086/8088和ARM核汇编语言程序设计》教材配套的实训教程，目的是使学生通过实验加深对理论课程的理解。本书在整个编写过程中，努力做到实验简单明了，具有一定的代表性，文字解释清晰，通俗易懂。

全书共11章20个实验，其中第1章至第7章是基于8086/8088汇编语言的实验，共有16个实验。第1章介绍了8086/8088汇编语言运行环境和方法，了解如何使用汇编语言编制程序，熟悉DEBUG有关命令的使用，掌握DEBUG有关指令的功能，利用DEBUG运行简单的程序段。第2章介绍了不同数码转换编程及程序调试。第3章介绍了汇编语言存储器操作的程序设计实验。第4章介绍了汇编语言程序设计基本结构实验，即顺序程序实验、分支程序实验。对每类实验都进行了详细的问题分析，实验验证。第5章介绍了数据运算类如双精度加法、BCD减法、乘法等程序设计实验。第6章介绍了I/O程序实验，对键盘扫描实验、绘图实验进行了详细论述。第7章介绍了汇编语言与C/C++语言的混合程序设计方式，重点叙述了C/C++嵌入汇编程序实验和C/C++调入汇编程序模块实验。第8章至第11章是基于ARM核微处理器汇编语言的实验，共有4个实验，分别对ARM、ADS集成开发环境、ADS1.2应用实例、ARM汇编语言程序设计和Thumb汇编语言程序设计进行了介绍。

本书的第1章、第6章的实验二、第7章至第11章由安徽理工大学徐辉老师、江静老师、李敬兆老师编写。其余各章实验均由承担相应章节教材的安徽工业大学张雷老师、纪平老师，安徽工程科技学院谢晓东老师、安徽建筑工程学院丁刚老师、夏巍老师编写。安徽理工大学李敬兆教授任主编。

由于编者水平所限，书中难免存在错误和不妥之处，敬请广大读者批评指正。

编 者

2006.6

目 录

第 1 章　汇编语言程序调试方法

实验一　汇编语言运行环境及方法、简单程序设计

一、实验目的

1. 熟悉汇编语言运行环境和方法。
2. 了解如何使用汇编语言编制程序。
3. 熟悉 DEBUG 有关命令的使用方法。
4. 利用 DEBUG 掌握有关指令的功能。
5. 利用 DEBUG 运行简单的程序段。

二、实验内容

1. 学会输入、编辑汇编语言程序。
2. 学会对汇编语言程序进行汇编、连接和运行。
3. 学会进入和退出 DEBUG 程序。
4. 学会 DEBUG 中的 D 命令、E 命令、R 命令、T 命令、A 命令、G 命令等的使用。对于 U 命令、N 命令、W 命令等，也应试一下。

三、实验准备

1. 仔细阅读有关汇编语言环境的内容，事先准备好使用的例子。
2. 准备好源程序清单、设计好调试步骤、测试方法、对运行结果的分析。
3. 一个程序：比较 2 个字符串所含的字符是否相同。若相同则显示‘Match.’，否则显示‘No match!’；
4. 细阅读有关 DEBUG 命令的内容，对有关命令，都要事先准备好使用的例子。

四、实验步骤

1. 在 DOS 提示符下，进入 MASM 目录。
2. 在 MASM 目录下启动 EDIT 编辑程序，输入源程序，并对其进行汇编、连接和运行。
   1) 调用 edit 输入、编辑源程序并保存在指定的目录中；例：

   ```
   edit abc.asm
   ```

   2) 用汇编程序 masm 对源程序汇编产生目标文件 obj。例：

   ```
   masm abc
   ```

不断修改错误，直至汇编通过为止。

3) 用连接程序 link 产生执行文件 exe.例：link abc

4) 执行程序

可直接从 DOS 执行程序，即在 DOS 环境中，输入文件名即可。

3. 记录每一步所用的命令，以及查看结果的方法和具体结果。

五、实验方法

有关汇编语言程序的上机过程请读者参阅清华大学出版社 1991 年出版的《IBMPC 汇编语言程序设计》的 4.4 节。在这里，我们举例简要说明该过程以及程序的调试方法。

例 1.1 比较字符串 sample

试编写一程序：比较两个字符串 string1 和 string2 所含的字符是否相同。若相同则显示'Match'，否则，显示'Nomatch'。

我们可以用串比较指令来完成程序所要求的功能。上机过程如下：

1. 调用字处理程序 wordstar 建立 asm 文件

当然也可以用其它编辑程序如 pced 或行编辑程序 edlin 来建立源文件。

C>WS

使用非文本文件方式(n 命令)建立以 sample. asm 为文件名的源文件如图 1-1 所示。然后用 CTRL K X 命令将文件存入磁盘，并使系统返回 DOS。

```
;PROGRAM TITLE GOES HERE--Compare string
;*****************************************************************
datarea segment      ;define data segment
string1     db       'Move the cursor backward.'
string2     db       'Move the cursor backward.'
;
mess1       db       'Match.',13,10,'$'
mess2       db       'No match!',13,10,'$'
datarea ends
;*****************************************************************************
prognam segment      ;define code segment
;-------------------------------------------------------------------------------------------
main        proc     far
assume cs:prognam,ds:datarea,es:datarea
start:      ;starting execution address
;set up   stack for return
push        ds       ;save old data segment
sub         ax,ax    ;put zero in AX
push        ax       ;save it on stack
;set DS register to current data segment
```

```
mov     ax,datarea      ;datarea segment addr
mov     ds,ax           ;into DS register
mov     es,ax           ;into ES register
;MAIN PART OF PROGRAM GOES HERE
lea     si,stringl
lea     di,string2
cld
mov     cx,25
repz    cmpsb
jz      match
lea     dx,mess2
jmp     short disp
match:
lea     dx,mess1
disp:
        mov ah,09
        int     21h
ret     ;return to DOS
main    endp    ;end of main part of program
;-------------------------------------------------------------------------------
prognam ends    ;end of code segment
;*****************************************************************
end     start   ;end assembly
```

图 1-1　例 1.1 的原文件 sample.asm

2. 用汇编程序 masm(或 asm)对源文件汇编产生目标文件 obj

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


  51520 + 423584 Bytes symbol space free


      0 Warning Errors
      0 Severe  Errors
```

3. 用连接程序 link 产生执行文件 exe

```
C:\>link sample;

Microsoft (R) Overlay Linker  Version 3.60
```

LINK : warning L4021: no stack segment

执行程序

4. 可直接从 DOS 执行程序如下：

C:\>sample

Match.

终端上已显示出程序的运行结果。为了调试程序的另一部分，可重新进编辑程序修改两个字符串的内容，使它们互不相同。如修改后的数据区为:

```
;*********************************************************************
    datarea segment     ;define data segment
    stringl     db      'Move the cursor backward.'
    string2     db      'Move the cursor forword.'
    ;
    mess1       db      'Match.',13,10,'$'
    mess2       db      'No match!',13,10,'$'
    datarea ends
;*********************************************************************
```

然后，重新汇编、连接、执行，结果为：

C>sample

No mateh!

至此，程序已调试完毕，运行结果正确。

另一种调试程序的方法是使用 debug 程序。可调用如下：

C>debug sample.exe

-

此时，debug 已将执行程序装入内存，可直接用 g 命令运行程序。

-g

Match.

Program terminated normally

为调试程序的另一部分，可在 debug 中修改字符串内容。可先用 u 命令显示程序以便了解指令地址。显示结果如图 1-2 所示。

```
-u
0D36:0000 1E            PUSH    DS
0D36:0001 2BC0          SUB     AX,AX
0D36:0003 50            PUSH    AX
0D36:0004 B8310D        MOV     AX,0D31
0D36:0007 8ED8          MOV     DS,AX
0D36:0009 8EC0          MOV     ES,AX
```

```
0D36:000B 8D360000      LEA     SI,[0000]
0D36:000F 8D3E1900      LEA     DI,[0019]
0D36:0013 FC            CLD
0D36:0014 B91900        MOV     CX,0019
0D36:0017 F3            REPZ
0D36:0018 A6            CMPSB
0D36:0019 7406          JZ      0021
0D36:001B 8D163B00      LEA     DX,[003B]
0D36:001F EB04          JMP     0025
-u
0D36:0021 8D163200      LEA     DX,[0032]
0D36:0025 B409          MOV     AH,09
0D36:0027 CD21          INT     21
0D36:0029 CB            RETF
0D36:002A 8A4608        MOV     AL,[BP+08]
0D36:002D 98            CBW
0D36:002E 50            PUSH    AX
0D36:002F 8B4604        MOV     AX,[BP+04]
0D36:0032 03C6          ADD     AX,SI
0D36:0034 50            PUSH    AX
0D36:0035 E858FF        CALL    FF90
0D36:0038 83C406        ADD     SP,+06
0D36:003B 8BF8          MOV     DI,AX
0D36:003D 83FFFF        CMP     DI,-01
0D36:0040 750C          JNZ     004E
```

图 1-2　例 1.1 用 debug 调试时，u 命令的显示情况

将断点设置在程序的主要部分以前。

```
-g0b

AX=0D31  BX=0000  CX=007A  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=000B   NV UP EI PL ZR NA PE NC
0D36:000B 8D360000          LEA     SI,[0000]                    DS:0000=6F4D
```

根据其中指示的 ds 寄存器内容查看数据段的情况如下：

```
-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20   Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68   backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72   e cursor backwar
0D31:0030  64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61   d.Match...$No ma
```

```
0D31:0040   74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00     tch!..$.........
0D31:0050   1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D     .+.P.1.....6...
0D31:0060   3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB     >........t...;..
0D31:0070   04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B     ...2....!..F..P.
```

可用 e 命令修改数据区的字符串，操作如下：

```
-e29
0D31:0029  62.66    61.6f    63.72    6B.77    77.61    61.72    72.64
0D31:0030  64.2e    2E.20
```

再次用 d 命令查看修改结果。

```
-d0
0D31:0000   4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20     Move the cursor
0D31:0010   62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68     backward.Move th
0D31:0020   65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64     e cursor forward
0D31:0030   2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61     . Match...$No ma
0D31:0040   74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00     tch!..$.........
0D31:0050   1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D     .+.P.1.....6...
0D31:0060   3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB     >........t...;..
0D31:0070   04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B     ...2....!..F..P.
```

用 g 命令运行程序，结果为：

```
-g
No match!

Program terminated normally
```

用 q 命令退出 debug。

```
-q
```

至此，程序已调试完毕。为了进一步说明 debug 命令的使用方法，我们再次重复上述程序的调试过程，只是使用 e、a 和 f 命令来修改数据区的内容而已。必须注意，由于在用 debug 调试程序时，只能修改当时有关的内存单元内容，因此重新用 debug 装入程序时，仍是原来在磁盘中的内容。操作如下：

```
C:\>debug sample.exe
-g0b

AX=0D31  BX=0000  CX=007A  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=000B     NV UP EI PL ZR NA PE NC
0D36:000B     8D360000                  LEA                 SI,[0000]
DS:0000=6F4D
-d0
0D31:0000   4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20     Move the cursor
```

```
0D31:0010   62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68    backward.Move th
0D31:0020   65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72    e cursor backwar
0D31:0030   64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61    d.Match...$No ma
0D31:0040   74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00    tch!..$.........
0D31:0050   1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D    .+.P.1......6...
0D31:0060   3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB    >........t...;..
0D31:0070   04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1    ...2....!.F..^..
-e29 'forward.'20
-d0
0D31:0000   4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20    Move the cursor
0D31:0010   62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68    backward.Move th
0D31:0020   65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64    e cursor forward
0D31:0030   2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61    . Match...$No ma
0D31:0040   74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00    tch!..$.........
0D31:0050   1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D    .+.P.1......6...
0D31:0060   3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB    >........t...;..
0D31:0070   04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1    ...2....!.F..^..
-g
No match!

Program terminated normally
```

可见这种 e 命令的方式避免使用 ASCII 码进入，对用户是比较方便的。其中最后一个 20 是空格键的 ASCII 码，以补足原来的内容恢复原状，具体如下：

```
-a0d31:29
0D31:0029 db 'backward.'
0D31:0032
-d0
0D31:0000   4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20    Move the cursor
0D31:0010   62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68    backward.Move th
0D31:0020   65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72    e cursor backwar
0D31:0030   64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61    d.Match...$No ma
0D31:0040   74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00    tch!..$.........
0D31:0050   1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D    .+.P.1......6...
0D31:0060   3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB    >........t...;..
0D31:0070   04 8D 16 32 00 B4 09 CD-21 CB 56 FA 89 7E EE 8B    ...2....!.V..~..
-g
Match.
```

由于 a 命令是汇编命令，因此信息是用汇编格式进入的。如果修改的是程序中的语句，方

法也是相同的，下面我们还会看到这类的操作。现在再看一下用 f 命令修改数据区的方法。

```
C:\>debug sample.exe
-g0b

AX=0D31  BX=0000  CX=007A  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=000B   NV UP EI PL ZR NA PE NC
0D36:000B     8D360000                LEA                SI,[0000]
DS:0000=6F4D
-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20   Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68   backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72   e cursor backwar
0D31:0030  64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61   d.Match...$No ma
0D31:0040  74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00   tch!..$.........
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D   .+.P.1......6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB   >........t...;..
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 56 FA 89 7E EE 8B   ...2....!.V..~..
-f29 l9 'forward.'20
-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20   Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68   backward.Move the
0D31:0020  65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64   e cursor forward
0D31:0030  2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61   Match...$No ma
0D31:0040  74 63 21 0D 0A 24 00-00 00 00 00 00 00 00 00      tch!...$.........
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D   .+.P.....6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB   >........t...;..
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 56 FA 89 7E EE 8B   ...2....!.V..~..
-g
No match!

Program terminated normally
-q
```

f 命令中的 29 为修改区的首地址，l9 表示需要修改的长度为 9 个字节。

为进一步说明程序的调试过程，现假设程序编制错误：在源文件中把 jz match 改为 jnz Match。该程序汇编、连接后，进入 debug 调试如下：

```
C>debug sample.exe
-g
No match!
```

```
Program terminated normally
```

结果是错误的（因源文件中的两个字符是相同的）。为检查程序的错误，将断点设在比较串之后。

```
-g19
AX=0D31  BX=0000  CX=0000  DX=0000  SP=FFFC  BP=0000  SI=0019  DI=0032
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=0019   NV UP EI PL ZR NA PE NC
0D36:0019 7506          JNZ     0021
```

此时零标志为 ZR,即 ZF=1，即表示比较结果相等，说明比较结果是正确的。现在可用 p 命令再执行一条指令以观察指令的转向。

```
-p

AX=0D31  BX=0000  CX=0000  DX=0000  SP=FFFC  BP=0000  SI=0019  DI=0032
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=001B   NV UP EI PL ZR NA PE NC
0D36:001B    8D163B00                LEA                DX,[003B]
DS:003B=6F4E
```

为查到 003B 单元的内容，可查数据区如下：

```
-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20   Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68   backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72   e cursor backwar
0D31:0030  64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61   d.Match...$No ma
0D31:0040  74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00   tch!..$.........
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D   .+.P.1......6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 75 06 8D 16 3B 00 EB   >........u...;..
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 46 FA 8B 5E FC C1   ...2....!.F..^..
```

可见 003B 单元的内容为 4E，即 N 的 ASCII 码，后面跟的是 No match!，这说明 jnz 指令使用错误，应该改为 jz。可用 a 命令修改，并用 u 命令检查修改结果。运行结果说明程序修改正确。

```
-a19
0D36:0019 jz 0021
0D36:001B
-u0
0D36:0000 1E            PUSH    DS
0D36:0001 2BC0          SUB     AX,AX
0D36:0003 50            PUSH    AX
0D36:0004 B8310D        MOV     AX,0D31
0D36:0007 8ED8          MOV     DS,AX
0D36:0009 8EC0          MOV     ES,AX
```

```
0D36:000B 8D360000        LEA       SI,[0000]
0D36:000F 8D3E1900        LEA       DI,[0019]
0D36:0013 FC              CLD
0D36:0014 B91900          MOV       CX,0019
0D36:0017 F3              REPZ
0D36:0018 A6              CMPSB
0D36:0019 7406            JZ        0021
0D36:001B 8D163B00        LEA       DX,[003B]
0D36:001F EB04            JMP       0025
-rip
IP 001B
:
-q
```

在这里应该注意，在使用 a 命令修改数据区时，必须给出数据段的地址，而在修改程序区时由于 a 命令的缺省段为代码段，所以直接给出偏移地址就可以了。

在调试过程中，也可以用 t 命令逐条跟踪程序的执行。下面列出断点停在 0b 后，用 f 命令修改数据区中字符串的内容，然后用 t 命令逐条执行指令的情况。

```
-f29 l9 'forward.'20
-d0
0D31:0000  4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20   Move the cursor
0D31:0010  62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68   backward.Move th
0D31:0020  65 20 63 75 72 73 6F 72-20 66 6F 72 77 61 72 64   e cursor forward
0D31:0030  2E 20 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 6D 61   . Match...$No ma
0D31:0040  74 63 68 21 0D 0A 24 00-00 00 00 00 00 00 00 00   tch!..$.........
0D31:0050  1E 2B C0 50 B8 31 0D 8E-D8 8E C0 8D 36 00 00 8D   .+.P.1......6...
0D31:0060  3E 19 00 FC B9 19 00 F3-A6 74 06 8D 16 3B 00 EB   >........t...;..
0D31:0070  04 8D 16 32 00 B4 09 CD-21 CB 8A 46 08 98 50 8B   ...2....!..F..P.
-t

AX=0D31  BX=0000  CX=007A  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=000F     NV UP EI PL ZR NA PE NC
0D36:000F 8D3E1900  LEA       DI,[0019]                    DS:0019=6F4D
-t

AX=0D31  BX=0000  CX=007A  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0019
DS=0D31  ES=0D31  SS=0D31  CS=0D36  IP=0013     NV UP EI PL ZR NA PE NC
0D36:0013 FC              CLD
-t
```