面向 FoxBASE 的
汇编语言编程与应用

毕广吉　著

北京大学出版社
北　　京

内 容 提 要

本书是 FoxBASE 编程者的高级读物,专门介绍面向 FoxBASE 的汇编语言编程方法,并给出大约 200 个实用的汇编语言模块。这些汇编语言程序极大地扩充了 FoxBASE 的功能,使 FoxBASE 应用程序升级到新的水平。书中的方法及程序同时也适用于 FoxPro。

本书供学习和使用 FoxBASE, FoxPro 的编程者阅读,也可供学习汇编语言的人员学习,还可作为大、中专院校师生的参考书。

与本书同时出版的还有配套的软件,包含了书中全部的 .ASM, .PRG 源程序及 .BIN 模块,需要的读者请与北京大学出版社软件部联系购买。

图书在版编目(CIP)数据

面向 FoxBASE 的汇编语言编程与应用/ 毕广吉著 .- 北京: 北京大学出版社,1996 .9
ISBN 7-301-03170-X
Ⅰ.面… Ⅱ.毕… Ⅲ.汇编语言 Ⅳ.TP312

书　　名:面向 FoxBASE 的汇编语言编程与应用
著作责任者:毕广吉
责 任 编 辑:沈承凤
标 准 书 号:ISBN 7-301-03170-X/ TP・0305
出　版　者:北京大学出版社
地　　址:北京市海淀区中关村北京大学校内　100871
电　　话:出版部 62752015　发行部 62559712　编辑部 62752032
排　印　者:北京高新特公司盛达激光照排中心
发　行　者:北京大学出版社
经　销　者:新华书店
787×1092 毫米 16 开本　26.75 印张　666 千字
1996 年 10 月第一版 1996 年 10 月第一次印刷
定　　价:35.00 元

前　言

现在 FoxBASE 数据库系统非常流行,相当多的人学习过并且掌握了 FoxBASE 编程,但是其中为数不少的编程者却难以编写出可达商品化水平的应用程序。其主要原因是,FoxBASE 的命令/ 函数集并不能满足我们全部的需要,必须利用汇编语言对其功能进行扩充。本书就是针对这样的读者编写的,专门介绍面向 FoxBASE 的汇编语言编程,是 FoxBASE 的高级读物。利用本书介绍的方法或直接调用本书的程序,可以做出图文并茂、声色俱佳的用户界面和应用程序,使软件增色不少。

尽管目前图书市场上关于 FoxBASE 的书很多,但其中关于汇编语言编程的叙述却极少,往往只是罗列一下调用规则而已,最多也不过举上一个十几行的例子,读者无法从中学习编程方法,更得不到实用的程序。另一方面,很多 FoxBASE 编程者对汇编语言了解不多,再加上汇编语言编程还要涉及 BIOS,DOS,汉字系统、VGA 显示卡、可编程控制器等软、硬件知识,因而本书专门介绍供 FoxBASE 调用的汇编语言编程是十分必要的。

本书首先分析了 FoxBASE 调用汇编语言子程序的规则、调用的过程、FoxBASE 内存变量存储的格式、参数传递方法等必备的基础知识,然后分别具体介绍显示输出、键盘控制、时钟操作、磁盘/ 文件操作、存取屏幕/ 窗口、图形编程、前台/ 后台音乐、位图显示、统计图形、亮度控制、艺术清屏等编程原理和方法,并给出约 200 个实用程序。

编写本书时还特别考虑到了另外一些读者,这部分读者只想获得实用程序,不打算深入学习编程原理和技术细节,那么只要将本书中提供的程序应用于你的系统中,立即会使你的程序得到极大的改善,升级到专业化、商品化的水平。

本书介绍的方法和程序也适用于 FoxPro。通过本书的学习,读者不但可以学习汇编语言编程方法,得到实用程序,而且还能学习到很多关于 VGA 卡、DOS,BIOS 可编程控制器的知识。除此之外,本书还能帮助读者更深一步地了解 FoxBASE 本身。

全书所有程序均在 FoxBASE　2.10,MASM　5.10,DOS　3.30,256K 的 VGA 卡环境下调试通过,所有程序均与具体的汉字系统无关。

由于作者水平所限,谬误之处在所难免,敬请读者赐教指正。

毕 广 吉

1996 年 3 月

第一章　面向 FoxBASE 的汇编语言编程基础

本章主要介绍面向 FoxBASE 的汇编语言编程必备的基础知识。由于篇幅所限，本书不介绍 8088/ 8086 汇编语言指令和伪指令的功能与用法，也不介绍汇编语言编程的一般方法，读者可以自行参考有关的书籍，例如书末所列的参考文献[1]等。

1.1　在 FoxBASE 中调用汇编语言子程序的方法

1.1.1　调用方法

在 FoxBASE 中使用汇编语言子程序有三个步骤：

第一步，用 LOAD 命令将汇编语言子程序模块从磁盘装入内存，准备供 CALL 命令调用，即

LOAD < 二进制文件名 >

装入的程序必须是二进制文件形式，默认的扩展名是 .BIN，每个文件长度不能超过 32K 字节。FoxBASE 最多可装入 16 个二进制文件。文件装入后，文件名便成为模块名，扩展名不再使用。因此，如果新装入的文件名与原先装入的文件名相同，即使它们的扩展名不同，新装入的文件也将覆盖掉内存中原先装入的那个文件。如果在装入文件时不指出扩展名，LOAD 命令认为扩展名是默认的扩展名 .BIN。

本书中将把二进制汇编语言模块简称为 .BIN 模块，而不论实际的扩展名是不是 .BIN，事实上也没有太充分的理由不使用 .BIN 作为扩展名。

有一点应该特别指出，如果一个二进制文件被重复装入，FoxBASE 并不会去检查它是否已经装入过，而是再一次装入同一文件，其结果是将上次装入的同一个内容的文件覆盖掉。

通常看来，这样做法不仅浪费装载的时间而且对于一个较复杂的 .BIN 模块来说，可能会造成灾难。例如，用 LOAD 命令将音乐程序 SOUND.BIN 装入内存(见第八章)，然后用 CALL 命令调用后台演奏音乐功能，一首悠扬的乐曲就开始了。如果此时再重复装入 SOUND.BIN 模块，就不再演奏乐曲并且有可能死机。

其原因很简单，在第一次装入程序并调用后台音乐功能时已经改变了 1CH 号时钟中断向量，同时将表示乐曲的音符串填写在内存中，使音乐正常演奏。这时如果突然调入同一程序进行覆盖，纵然二者的程序代码完全相同，但却失去了已经填写好的重要数据，音乐演奏无法进行，更无法恢复原来的 1CH 号中断向量，这就会导致死机。

因为 FoxBASE 并不负责检查 .BIN 模块的重复装入，所以这一责任就只好由程序员自己承担了。

第二步，用 CALL 命令调用汇编语言程序。当 FoxBASE 用 LOAD 命令装入一个 .BIN 程序时，并不立即运行它。要想调用 .BIN 程序，就要使用 CALL 命令。CALL 命令的用法是：

CALL < 二讲制模块名 > [WITH < 字符串表达式 >/ < 内存变量 >]

其中 WITH 子句指明调用参数,当无输入输出参数时,该子句可以省略。传递参数可以用字符串表达式,也可以用内存变量。内存变量可以是字符型 C、数值型 N、日期型 D、逻辑型 L 中的任一种。关于参数的使用,将在 1.3.3 节中详细讨论。

因为 LOAD 命令在装入二进制模块时已经不再使用扩展名,所以 CALL 命令不需要指出扩展名。

第三步,释放 .BIN 模块。RELEASE MODULE 命令用于从内存中释放二进制模块占用的内存,方法是:

RELEASE MODULE < 二进制文件名 >

使用该命令一方面可以使 FoxBASE 使用多于 16 个 .BIN 模块,另一方面,及时释放 .BIN 模块所占用的内存,能让 FoxBASE 有更多的内存空间用于其他工作。

RELEASE MODULE 命令不支持通配符或 ALL 选项。当要释放多个 .BIN 模块时,必须逐个地释放。该命令也不需要指定扩展名,即使二进制文件不是使用默认的扩展名 .BIN 时也是如此。

CLEAR ALL 或 CLOSE ALL 都不能使 FoxBASE 释放二进制模块。除 RELEASE MODULE 命令之外,唯一能让 FoxBASE 释放二进制模块的命令只有 QUIT,QUIT 在退出 FoxBASE 时同时释放二进制模块占用的内存。

使用 RELEASE MODULE 命令或 QUIT 命令释放 .BIN 模块时,必须先恢复由该 .BIN 模块改变了的重要的数据,例如中断向量表等,不然可能会使系统处于瘫痪。

1.1.2 FoxBASE 对汇编语言子程序的要求

FoxBASE 在调用汇编语言子程序时有严格的约定,在编写汇编语言模块时,必须遵守这些约定,只有这样,才能正确无误地调用 .BIN 模块,也只有这样,才能在调用之后顺利地返回 FoxBASE。通常 FoxBASE 手册的说明有如下几点:

(1) 第一个可执行命令必须放在偏移量 0 处;

(2) 配置或使用的内存空间不可超过实际 .BIN 模块的长度,因为 LOAD 命令用文件的大小来确定需要分配的内存容量;

(3) .BIN 程序不能增加或减少作为 CALL…WITH…命令参数的内存变量的长度;

(4) 返回 FoxBASE 之前必须恢复 CS 和 SS 寄存器;

(5) .BIN 程序结束时用远程返回指令 RETF,将控制权返回给 FoxBASE。

下面我们对这些约定做深入的分析和说明。

第(1)条约定让我们在程序开关处写一条伪指令 ORG 0,以便程序从偏移量 0 开始。占据该存储单元的必须是一个可执行的指令,如果程序的开头打算作为数据区,那么此处应该用一个 JMP 指令跳转到程序开始处。不要指望汇编语言的 END START 之类的伪指令为我们安排程序开始的地址,因为 FoxBASE 总是从偏移量 0 处开始执行 .BIN 程序的。

第(2)条约定主要是指不要向 .BIN 模块之外的内存空间写数据,不然极有可能造成系统的错误或混乱。实际上,从 .BIN 模块空间之外读取数据总是安全的,而向 .BIN 模块空间之外写数据虽不一定安全,但又是必不可少的。在本书中,经常向显示存储区、BIOS 数据区、甚

至 FoxBASE 空间写数据,问题不是写不写,而是如何将数据写到安全处,绝不可以将 FoxBASE 搞乱。

第(3)条约定是至关重要的,.BIN 程序不能改变作为调用参数的内存变量在 FoxBASE 内存空间中的长度,不然会造成 FoxBASE 内存变量的混乱。关于这一点,在 1.3.2 节中还要仔细分析。

至于约定(4),一般都可以自动满足。FoxBASE 用远调用 CALL FAR 方式调用 .BIN 模块,而 .BIN 模块用远返回 RETF 返回到 FoxBASE,所以 CS 寄存器总是自动恢复的。至于 SS 寄存器,除非很有必要,绝大多数的 .BIN 程序并不切换堆栈段,而是使用 FoxBASE 的堆栈。既然 SS 未被改变过,所以也用不着保存和恢复。当然,使用 FoxBASE 的堆栈时要注意一点,那就是不要使堆栈溢出,这在编程时给予适当的注意就可以了。

最后,约定(5)当然是必须遵守的,就是说要将程序定义成一个远过程,并用 RETF 指令返回到 FoxBASE。

关于 FoxBASE 调用汇编语言子程序的问题在 1.3.1 至 1.3.4 节中还要进一步深入分析。

1.2 汇编语言的上机过程

1.2.1 汇编语言的上机过程

本节简单介绍汇编语言的上机过程,详细内容请读者自己参考有关书籍。

1. 编辑源程序

编辑源程序可以使用行编辑程序 EDLIN.COM 或字处理软件 WORDSTAR, CCED, WPS 等,使用字处理软件时应该用 N 方式,不要用 D 方式,以免 D 方式在源程序中插入软回车、分页符等,使汇编时出错。源程序文件扩展名通常是 .ASM。

2. 汇编

汇编是把 .ASM 源程序转换成供 LINK 使用的目标模块 .OBJ,汇编用 MASM.EXE 完成。汇编时需一个输入文件 .ASM,产生三个输出文件,即 .OBJ,.LST 和 .CRF。其中只有目标文件 .OBJ 是最终生成 .BIN 文件所必须的,其他两个文件可以帮助你调试程序,根据需要决定取舍。

3. 连接

连接程序 LINK.EXE 把汇编生成的目标文件 .OBJ 转换成可执行文件,扩展名为 .EXE。输入文件有两个,一个是 .OBJ 文件,另一个是库文件 .LIB,可以直接回车。输出文件也有两个,即 .EXE 和 .MAP,只有 .EXE 是最终产生 .BIN 文件所必需的。

连接时产生一个

```
LINK: warning L4021: no stack segment
```

(没有堆栈段)的报错信息,可不予理睬。

4. 转换成 .BIN 文件

把 .EXE 文件转换成 .BIN 文件要用 EXE2BIN.EXE 程序,EXE2BIN 要求的输入文件是 .EXE,输出文件默认的扩展名就是 .BIN。

并不是所有的 .EXE 文件都能转换为 .BIN 文件。为了能转换成 .BIN 文件,在编写源程

序.ASM 时必须遵守一定的规则,详细的规定请读者参考汇编语言的有关书籍。本书中所有程序都是符合这些规定的。

1.2.2 第一个汇编语言程序

现在我们用实例说明汇编语言的上机过程。这个实例,就是我们的第一个汇编语言程序——SMALLCAP.ASM。这个程序的功能是将一个字符串中的所有小写字母转换成大写字母,对于非英文字母不做任何改变。该程序的功能类似于 FoxBASE 的 UPPER( )函数,但这是由.BIN 模块完成的。

第一步,用编辑软件(如 WS 等)将程序 SMALLCAP.ASM 输入并存盘。

```
;
;   程序名: SMALLCAP .ASM
;   用   途: 将 FoxBASE 传入的参数变为大写后传回 FoxBASE
;   形   式:  .BIN
;   日   期: 1995 .08
;

CODE        SEGMENT     BYTE PUBLIC          ;定义代码段
            ASSUME      CS:CODE              ;说明代码段
            ORG         0                    ;程序起始偏移量
START:
            PUSH        DS
            POP         ES                   ;ES 与 DS 同段
            MOV         SI,BX                ;初始化 SI
            MOV         DI,BX                ;初始化 DI
SMALLCAP1:
            LODSB                            ;取一字符
            CMP         AL,0                 ;是否结束符
            JZ          SMALLCAP3            ;是结束符,程序结束
            CMP         AL,"a"               ;是否 < "a"
            JB          SMALLCAP2            ;小于,小写字母
            CMP         AL,"z"               ;是否 > "z"
            JA          SMALLCAP2            ;大于,非小写字母
            AND         AL,5FH               ;小写转大写
SMALLCAP2:
            STOSB                            ;送回
            JMP         SHORT SMALLCAP1      ;下一个字符
SMALLCAP3:
            RETF                             ;返回 FoxBASE
CODE        ENDS                             ;代码段结束
            END         START                ;程序结束
```

第二步,汇编,做法是:

```
C:\ >MASM
Microsoft (R) Macro Assembler Version 5 .10
Copyright (C) Microsoft Corp 1981, 1988 . All rights reserved .

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

  49786 + 307041 Bytes symbol space free

    0 Warning Errors
```

```
  0 Severe Errors
```

汇编时共提问四个文件名,第一个是源文件名,回答 SMALLCAP,扩展名 .ASM 可以省略。第二个是目标文件名,系统已经给出了一个缺省的文件名 SMALLCAP.OBJ,所以直接回车即可。第三、四个文件名分别为列表文件和交叉引用文件名,默认的是空文件 NUL.LST 和 NUL.CRF,即不要这两个文件,所以通常是直接回车。然后开始进行汇编,并显示信息。

也可以用命令行方式进行汇编,方法是:

```
C:\ > MASM SMALLCAP;
Microsoft (R) Macro Assembler Version 5 .10
Copyright (C) Microsoft Corp 1981, 1988 . All rights reserved .

  49786 + 307041 Bytes symbol space free

    0 Warning Errors
    0 Severe Errors
```

其中 MASM 后的第一个参数是源文件名,紧跟着一个分号表示其余三个参数(即 .OBJ 文件、.LST 文件和 .CRF 文件)取缺省值。

如果汇编过程没有发生错误,则继续进行第三步——连接;若出现报错信息,则返回到第一步,根据报错信息修改源程序 SMALLCAP.ASM,然后重新汇编。

第三步,汇编正确之后就可以进行连接,由目标文件 .OBJ 生成可执行文件 .EXE,方法是:

```
C:\ > LINK

Microsoft (R) Overlay Linker Version 3 .64
Copyright (C) Microsoft Corp 1983 - 1988 . All rights reserved .

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

连接时首先提问目标文件名,输入 SMALLCAP,扩展名 .OBJ 可以省略。接着提问可执行文件名,系统给出的默认值是 SMALLCAP.EXE,故可直接回车。提问的第三个文件名是连接程序列表文件,缺省值是空文件 NUL.MAP,即不要该文件,通常是直接回车取缺省值。第四提问库文件名,本书中并不使用库文件,所以直接回车。回答完文件名后开始进行连接工作,生成可执行文件 SMALLCAP.EXE。

连接程序也可以用命令行方式运行,方法是:

```
C:\ > LINK SMALLCAP;
Microsoft (R) Overlay Linker Version 3 .64
Copyright (C) Microsoft Corp 1983 - 1988 . All rights reserved .

LINK : warning L4021: no stack segment
```

其中命令行中的第一个参数为目标文件名,分号表示其余的三个参数取缺省值。

通常连接产生的 .EXE 文件可以在 DOS 提示符下键入并运行。但我们的第一个汇编语言程序 SMALLCAP.ASM 生成的可执行文件 SMALLCAP.EXE 却不能在 DOS 下正确运行,这是因为我们的源程序 SMALLCAP.ASM 是按 .BIN 文件的要求编写的,而不是按

.EXE 文件的要求编写的。所以对于我们来说,.EXE 文件只是一个中间产物,最终我们要的是.BIN 文件,因此还要进行第四步,转换成.BIN 文件。关于.EXE 文件和.BIN 文件的不同之处,在第 1.4.3 节中还要加以说明。

第四步,转换成.BIN 文件,方法是:

```
C:\ > EXE2BIN SMALLCAP
```

命令行中的第一个参数是.EXE 文件名,扩展名可以省略;第二个参数是转换后的文件名,缺省时转换后的文件名主名与.EXE 文件主名相同,扩展名为.BIN,我们就用这种缺省的方式。

由于我们的源程序 SMALLCAP.ASM 是按照.BIN 文件的格式编写的,所以转换不会出错,不然可能会报告一些出错信息。

至此,产生.BIN 文件的全部过程已经完成,看一看目录:

```
C:\ > DIR SMALLCAP.*
 Volume in drive C is 12345678901
 Directory of C:\

SMALLCAP BIN        25      3-26-96     6:12a
SMALLCAP OBJ        86      3-26-96     6:12a
SMALLCAP ASM      1264      3-26-96     6:08a
SMALLCAP EXE       537      3-26-96     6:12a
        4 File(s)     5218304 bytes free
```

其中 SMALLCAP.OBJ,SMALLCAP.EXE 是中间产物,可以删去。

1.2.3 .BIN 程序的调用方法

调用.BIN 程序要在 FoxBASE 中进行,以 SMALLCAP.BIN 为例,方法如下:

```
C:\ > MFOXPLUS
. LOAD SMALLCAP
. A = "123ABCmnop;$"
. CALL SMALLCAP WITH A
. ? A
123ABCMNOP;$
RELEASE MODULE SMALLCAP
. QUIT
C:\ >
```

可见 SMALLCAP.BIN 达到了预定的设计目的,将小写字母转换为大写字母,对其他字符没做任何变换。

SMALLCAP.BIN 是一个带有返回参数的.BIN 程序,如果把它作为一个函数,则调用起来更加方便。下面是自定义函数 SMALLCAP( ):

```
* 自定义函数 SMALLCAP .PRG
* 调用前先 LOAD SMALLCAP
PARAMETERS X
CALL SMALLCAP WITH X
RETURN X
```

调用方法是:

```
?SMALLCAP("123ABCmnop;$")
```

或

```
A = "123ABCmnop;$"
B = SMALLCAP(A)
 ?B
123ABCMNOP;$
```

有些读者可能会产生这样的疑问:既然 SMALLCAP.BIN 与函数 UPPER( )功能相同,又何必编这样一个程序呢?至少有两个理由,其一,这是我们的第一个汇编语言程序,着重用以说明汇编语言上机过程,程序本身的功能当然不能太复杂。其二,相当重要的,UPPER( )是 FoxBASE 系统提供的内部函数,你无法改变它的功能,而 SMALLCAP.BIN 是我们自己开发的,它可以完成我们希望的功能。例如,如果我们愿意,可以输入一个"1",返回一个"男",输入一个"2",返回一个"女",输入一个"bjdx"返回一个"北京大学"等等,可以说是"心想事成"。无论 FoxBASE 的命令/函数有多少,总不会满足程序员的所有要求,要扩充 FoxBASE 的功能,就靠.BIN 模块。本书的目的就是向读者介绍如何进行.BIN 格式的汇编语言编程。

1.2.4 用批文件完成汇编

汇编、连接、转换工作总是反复进行的,特别是源程序有错误时,就要多次地编辑、汇编、连接、转换、试运行。汇编、连接、转换工作可以用一个批文件来完成:

```
@ ECHO OFF
REM ASMTOBIN.BAT
IF %1! = = !GOTO NOFILENAME
IF NOT EXIST %1.ASM GOTO NOASM
MASM/Z %1;
IF NOT EXIST %1.OBJ GOTO NOOBJ
LINK %1;
IF NOT EXIST %1.EXE GOTO NOEXE
EXE2BIN %1
IF EXIST %1.BAK DEL %1.BAK
DEL %1.OBJ
DEL %1.EXE
DIR %1.*
GOTO END
:NOFILENAME
ECHO NO ASM FILENAME
ECHO Usage: ASMTOBIN FILENAME
GOTO END
:NOASM
ECHO %1.ASM NOT EXIST
GOTO END
:NOOBJ
ECHO %1.OBJ NOT EXIST (MASM ERROR ?)
GOTO END
:NOEXE
ECHO %1.EXE NOT EXIST (LINK ERROR ?)
:END
```

在批文件 ASMTOBIN.BAT 的第 5 行 MASM/Z %1;中的开关/Z 告诉汇编程序 MASM.EXE 当汇编出错时,不但显示报错信息,而且显示错误所在源程序行和行号。

批文件 ASMTOBIN.BAT 的用法是(以 SMALLCAP.ASM 为例):

```
ASMTOBIN SMALLCAP
```

其中命令行中的参数必须省略扩展名.ASM。该批文件将顺序依次完成汇编、连接、转换、删

除中间文件的工作,并具有错误处理能力。

1.3 FoxBASE 调用汇编语言子程序的深入分析

要想编写好供 FoxBASE 调用的汇编语言子程序,还需要对 FoxBASE 调用汇编语言子程序的过程、FoxBASE 内存变量存储格式以及 FoxBASE 与 .BIN 模块间参数的传递方法进行更深入的分析。

1.3.1 FoxBASE 是如何调用汇编语言子程序的

FoxBASE 调用 .BIN 模块的过程如下:

在用 LOAD 命令装入 .BIN 模块时,将程序名(去掉扩展名)及其装入内存后所在的 < 段地址:偏移量 > 存放在一张表中,供 CALL 命令使用。

当使用 CALL 命令执行汇编语言子程序时,FoxBASE 首先根据模块名在上述表中查找,从而得到被调用程序所在的 < 段地址:偏移量 > ,并对其输入参数进行一定的处理,然后通过下面一段指令来完成汇编语言程序的执行过程:

```
U4980:0000,0010
4980:0000 55          PUSH    BP
4980:0001 8BEC        MOV     BP,SP
4980:0003 56          PUSH    SI
4980:0004 57          PUSH    DI
4980:0005 1E          PUSH    DS
4980:0006 C55E0A        DS      X,[BP+0A]       ;传递参数指针
4980:0009 FF5E06      CALL    FAR [BP+06]       ;调用 .BIN 模块
4980:000C 1F          POP     DS
4980:000D 5F          POP     DI
4980:000E 5E          POP     SI
4980:000F 5D          POP     BP
4980:0010 CB          RETF
```

从这段程序可以看出,对 .BIN 模块的调用,是用 CALL FAR 的远程调用完成的,而参数指针是用 DS:BX 传递的,这与 FoxBASE 手册的叙述完全一致。

从这段程序还可以知道,调用 .BIN 模块时,FoxBASE 自己保存了 DS,DI,SI,BP 寄存器;而在 1.1.2 节中,我们已经指出 CS,IP 寄存器总是自动恢复,而 SS,SP 寄存器只要在 .BIN 模块中未切换堆栈段,也是能自动维持堆栈平衡的。于是只剩下 AX,BX,CX,DX,ES 以及 FLAG 需要保护。实践证明,FoxBASE 对这些寄存器的改变毫不在乎。所以,我们得到一个结论:在进入 .BIN 后返回到 FoxBASE 的调用过程中,并不要求 .BIN 模块保护寄存器。这就简化了汇编语言子程序的设计。

有的读者可能奇怪,上面那一段程序是怎样得到的呢?事实上得到这一段程序并不难,而且恰好可以用汇编语言子程序去得到它。

由于 FoxBASE 总是先用 LOAD 命令装入 .BIN 模块,再用 CALL 命令调用它,可以猜测 .BIN 模块不会装在 FoxBASE 的代码段中,因而 FoxBASE 对 .BIN 模块的调用很可能是一个远程间接调用 CALL FAR。根据汇编语言,在 CALL FAR 调用时,总是将下一条指令的 CS,IP 先压入堆栈,即保存返回地址,待 .BIN 程序执行到 RETF 时再弹出。所以,如果我们

在.BIN 模块中从堆栈指针 SP 以及 SP+2 处取出两字,则正好是 IP 和 CS。依据这个思路编了汇编语言程序 GETCODE.ASM,汇编、连接、转换后成为 GETCODE.BIN。

GETCODE.ASM 的程序流程很简单,首先将堆栈指针送 BP,然后从 SS:[BP+2]处取出 CS,从 SS:[BP]处取出 IP,利用子程序 AXTOHEX 和 BINTOASC 把二进制转换成十六进制的字符串,最后用 DOS 的显示字符串功能把 CS:IP 显示出来。

```
PAGE        50,132
;
;   程序名:GETCODE .ASM
;   用    途:侦察 FoxBASE 是如何调用 .BIN 模块的
;   形    式: .BIN
;   日    期:1995 .10
;
;——主程序开始————————————————————————————————————
CODE        SEGMENT     BYTE PUBLIC
            ASSUME      CS:CODE,DS:CODE
            ORG         0

GETCODE     PROC        FAR
START:
            JMP         GETCODE1
;——数据区————————————————————————————————————————
CODESTR0    DB          13,10
CODESTR     DB          " :",13,10,"$"
ASCII       DB          "0123456789ABCDEF"
;——子程序区————————————————————————————————————————
;
;   子程序:BIN2ASC
;   功    能:AL 中的四位二进制数化成 16 进制 ASCII 字符
;   输    入:AL = 二进制数(低四位有效)
;   输    出:AL = 16 进制 ASCII 字符
;
BIN2ASC     PROC        NEAR
            AND         AL,0FH
            LEA         BX,ASCII
            XLAT        ASCII
            RETN
BIN2ASC     ENDP

;
;   子程序:AXTOHEX
;   功    能:将 AX 中的二进制数转换成十六进制数
;   输    入:AX = 二进制数,ES:DI = 目址
;   输    出:转换成 4 位 16 进制字符存放在 ES:DI,DI 自动调整
;
AXTOHEX     PROC        NEAR
            MOV         CX,AX
            ;AX 的最高 4 位
            SHR         AH,1
            SHR         AH,1
            SHR         AH,1
            SHR         AH,1
            MOV         AL,AH
            CALL        BIN2ASC
            STOSB
            ;AX 的 11 - 8 位
            MOV         AL,CH
            CALL        BIN2ASC
            STOSB
            ;AX 的 7 - 4 位
            MOV AL,CL
```

```
                SHR             AL,1
                SHR             AL,1
                SHR             AL,1
                SHR             AL,1
                CALL            BIN2ASC
                STOSB
                ;AX的最低4位
                MOV             AL,CL
                CALL            BIN2ASC
                STOSB
                RETN
AXTOHEX         ENDP
;——主程序开始————————————————————————————————————————————
GETCODE1:
                MOV             BP,SP
                PUSH            CS
                POP             DS
                PUSH            CS
                POP             ES
                MOV             DI,OFFSET CODESTR
                MOV             AX,[BP+2]
                CALL            AXTOHEX
                MOV             AX,[BP]
                INC             DI
                CALL            AXTOHEX
                MOV             DX,OFFSET CODESTR0
                MOV             AH,9
                INT             21H
                RETF
GETCODE         ENDP

CODE            ENDS
                END             START
```

```
;
;   程序结束
;
```

现在让我们利用 GETCODE.BIN **来找到本节开头给出的那一段代码**:

```
 .LOAD GETCODE
 .CALL GETCODE
4890:000C
RUN DEBUG
 - U4980:000C
4980:000C 1F                    POP     DS
4980:000D 5F                    POP     DI
4980:000E 5E                    POP     SI
4980:000F 5D                    POP     BP
4980:0010 CB                    RETF
4980:0011 00558B                ADD     [DI-75],DL
4980:0014 EC                    IN      AL,DX
4980:0015 C45E06                LES     BX,[BP+06]
4980:0018 26                    ES:
4980:0019 8A6701                MOV     AH,[BX+01]
4980:001C 2AC0                  SUB     AL,AL
4980:001E 26                    ES:
4980:001F 8A0F                  MOV     CL,[BX]
4980:0021 2AED                  SUB     CH,CH
4980:0023 03C1                  ADD     AX,CX
4980:0025 5D                    POP     BP
4980:0026 CB                    RETF
```

```
4980:0027 90                NOP
4980:0028 55                PUSH    BP
4980:0029 8BEC              MOV     BP,SP
4980:002B 83EC06            SUB     SP,+06
```

可以看到,这里的第一行反汇编代码正是 CALL FAR 下面的一行,要想得到全部的程序,可以使反汇编地址往前提一些,键入 U4980:0000 试一试,成功了,请看:

```
- U4980:0000
4980:0000 55                PUSH    BP
4980:0001 8BEC              MOV     BP,SP
4980:0003 56                PUSH    SI
4980:0004 57                PUSH    DI
4980:0005 1E                PUSH    DS
4980:0006 C55E0A            LDS     BX,[BP+0A]
4980:0009 FF5E06            CALL    FAR [BP+06]
4980:000C 1F                POP     DS
4980:000D 5F                POP     DI
4980:000E 5E                POP     SI
4980:000F 5D                POP     BP
4980:0010 CB                RETF
4980:0011 00558B            ADD     [DI-75],DL
4980:0014 EC                IN      AL,DX
4980:0015 C45E06            LES     BX,[BP+06]
4980:0018 26                ES:
4980:0019 8A6701            MOV     AH,[BX+01]
4980:001C 2AC0              SUB     AL,AL
4980:001E 26                ES:
4980:001F 8A0F              MOV     CL,[BX]
- Q
.
```

GETCODE 程序的成功给我们一些启示:.BIN 模块不仅能扩充 FoxBASE 的功能,也能用来探察 FoxBASE 自身。如果你有兴趣,不妨把 GETCODE.ASM 中 MOV AX,[BP+2] 改为 MOV AX,[BP+10],把 MOV AX,[BP]改为 MOV AX,[BP+8],然后重新试一下,看看你得到了什么?得到的是调用 4980:0000 这一段程序那一条指令的下一条指令。因为在 4980:0009 之前有 4 个 PUSH 指令,使 SP 减小了 8,所以使 BP+10 和使 BP+8。

在第 2.4.5 节还指出了另一种可以侦察程序运行的方法。

1.3.2 FoxBASE 内存变量存储格式分析

为了使.BIN 模块与 FoxBASE 之间的参数传递更清楚,本节将对 FoxBASE 内存变量存储格式进行分析。

FoxBASE 将内存变量名、变量类型、变量值的地址等存放在一个内存块中,形成一张表,而将变量值存放在另一个内存块中,二者是分开存放的。由于.BIN 程序不想也没有必要去改变 FoxBASE 内存变量名及变量类型等,所以我们真正关心的只是变量值的存储地址和数据格式。

1. 字符型变量的存储格式

字符型内存变量占用内存的字节数不是一个常数,而由该变量本身的长度来决定。从 DS:BX 指向的字节处开始,用若干个字节来存储字符串的内容,最后一个字节以 0 作为结束

标记,所以,含 5 个字符的字符串占用 6 个字节的存储空间。因为 0 作为字符串结束标记使用,使得 FoxBASE 不允许在字符串中含有 CHR(0)字符,这可以用如下方法验证:

```
. A = "ABC" + CHR(0) + "DEF"
. ? A
ABCDEF
. ? LEN( A)
6                    (而不是 7)
.
```

可见 FoxBASE 会自动将 CHR(0)剔除掉。明确这一点对于 .BIN 编程者是有用的,因为 FoxBASE 不对 .BIN 程序的行为负责,所以程序员必须自己保证不在字符串变量的存储区中间插入一个 0,也不能将字符串存储区末尾的 0 去掉,不然会造成 FoxBASE 对内存变量管理的混乱。

2. 数值型变量

数值型变量在存储区中占用固定的长度,是用 IEEE 标准格式存储的。在处理数值型(特别是浮点数)数据时,汇编语言不如 FoxBASE 方便,所以在汇编语言子程序中常常尽量避免对数值型变量的处理。

3. 日期型变量

日期型变量在内存中也占用固定的长度,但存储的并不是年、月、日,而是经过一定变换的数据,所以汇编语言用起来并不方便。

4. 逻辑型变量

逻辑型变量在内存中的存储长度也是固定的,但只有第一个字节有用,用来表示逻辑值,0 表示 .F.,1 表示 .T.,其余字节不用。当调用 .BIN 模块时,BX 指向有用的第一个字节处。

下面的汇编语言程序 GETADDR.ASM 是用来得到内存变量的地址的,编程原理与 GETCODE.ASM 基本相同,不过这里返回的是 DS:BX,而不是 CS:IP,程序中有两个子程序 AXTOHEX 和 BIN2ASC 与 GETCODE.ASM 也是相同的,用以将二进制数转换成十六进制的字符串。

```
PAGE        50,132
;
;   程序名:GETADDR .ASM
;   用    途:得到 FoxBASE 内存变量的地址
;   形    式: .BIN
;   日    期:1995 .10
;
;——主程序开始————————————————————————————————————
CODE        SEGMENT     BYTE PUBLIC
            ASSUME      CS:CODE,DS:CODE
            ORG         0

GETADDR     PROC        FAR
START:
            JMP         GETADDR1
;——数据区————————————————————————————————————————
ADDRSTR0    DB          13,10
ADDRSTR     DB          " :",13,10,"$"
ASCII       DB          "0123456789ABCDEF"
;——子程序区————————————————————————————————————————
```

```
;
;     子程序:BIN2ASC
;     功    能:AL 中的四位二进制数化成 16 进制 ASCII 字符
;     输    入:AL = 二进制数(低四位有效)
;     输    出:AL = 16 进制 ASCII 字符
;
BIN2ASC     PROC            NEAR
            AND             AL,0FH
            LEA             BX,ASCII
            XLAT            ASCII
            RETN
BIN2ASC     ENDP
;
;     子程序:AXTOHEX
;     功    能:将 AX 中的二进制数转换成十六进制数
;     输    入:AX = 二进制数,ES:DI = 目址
;     输    出:转换成 4 位 16 进制字符存放在 ES:DI,DI 自动调整
;
AXTOHEX     PROC            NEAR
            MOV             CX,AX
            ;AX 的最高 4 位
            SHR             AH,1
            SHR             AH,1
            SHR             AH,1
            SHR             AH,1
            MOV             AL,AH
            CALL            BIN2ASC
            STOSB
            ;AX 的 11 - 8 位
            MOV             AL,CH
            CALL            BIN2ASC
            STOSB
            ;AX 的 7 - 4 位
            MOV             AL,CL
            SHR             AL,1
            SHR             AL,1
            SHR             AL,1
            SHR             AL,1
            CALL            BIN2ASC
            STOSB
            ;AX 的最低 4 位
            MOV             AL,CL
            CALL            BIN2ASC
            STOSB
            RETN
AXTOHEX     ENDP
;——主程序开始————————————————————————————————————————————
GETADDR1:
            MOV             DX,BX
            MOV             AX,DS
            PUSH            CS
            POP             DS
            PUSH            CS
            POP             ES
            MOV             DI,OFFSET ADDRSTR
            CALL            AXTOHEX
            MOV             AX,DX
            INC             DI
            CALL            AXTOHEX
            MOV             DX,OFFSET ADDRSTR0
            MOV             AH,9
            INT             21H
            RETF
GETADDR     ENDP
```

```
CODE        ENDS
            END          START
;
;    程序结束
;
```

GETADDR. ASM 汇编、连接、转换成 GETADDR. BIN 文件后,可用来得到字符型、数值型、日期型、逻辑型内存变量的地址,进而用 DEBUG 研究它们的存储情况,举例如下:

```
 .LOAD GETADDR
 .A = "ABCDEFGHIJK"
ABCDEFGHIJK
 .B = .T .
 .T .
 .CALL GETADDR WITH A
7A83:4618
 .CALL GETADDR WITH B
7A83:342C
 .RUN DEBUG
 - D7A83:4610L20
7A83:4610  00 00 70 17 00 00 12 00 - 41 42 43 44 45 46 47 48    ..p.....  BCDEFGH
                                     |————————→                  |————→
                                          变量内容
7A83:4620  49 4A 4B 00 00 00 16 00 - 5E 17 16 00 00 00 00 00    IJK ....^.......
           ←————|  |_|                                           ←—|
               结束符
 - D7A83:3420L20
7A83:3420  00 00 04 00 4C 80 00 00 - 01 00 00 00 01 00 00 00    ....L...........
                                                 |————→
                                           变量内容,01 表示 .T .
7A83:3430  00 00 00 00 00 00 00 00 - 00 00 00 00 00 00 00 00    ................
           ←————|
 - Q
```

1.3.3 参数的传递

FoxBASE 在调用 . BIN 模块时有两种方式传递参数,即用字符串表达式或用内存变量。用字符串表达式的方式可以实现 FoxBASE 向 . BIN 模块的参数传递,但无法实现 . BIN 模块向 FoxBASE 传回参数。因而这种方式只适用于无返回参数的情况。

当采用第二种方式,用内存变量传递参数时,可以实现双向参数传递。内存变量的类型原则上用字符型、数值型、日期型、逻辑型均可。但根据 1.3.2 节的分析,从实用的角度看,用数值型和日期型内存变量是不方便的。所以在 . BIN 实用程序中几乎不用这两种内存变量传递参数,需要时,可以用 STR( )函数及 DTOC( )函数转换成字符型,待从 . BIN 返回时再用 VAL( )函数和 CTOD( )函数转换回来。

从逻辑型内存变量的存储结构来看,逻辑型变量是容易操作的,可用来进行双向参数传递,可惜的是逻辑型内存变量只有 . F. 和 . T. 两个值,作为输入参数显得输入值个数太少,难以实用。但 . F. 或 . T. 作为输出量却非常有用,它们可以直接作为 IF, IIF 的判断条件,控制程序流程。

字符型内存变量便于 . BIN 程序处理,而且数值型、日期型、逻辑型内存变量也都可以转