华 中 科 技 大 学

课 程 实 验 报 告

课程名称：汇编语言程序设计实验

实验名称：实验一

实验时间：2019.9.16

实验地点：南一楼116

指导教师：鲁宏伟

专业班级：IS1802

学 号：U201814867

姓 名：王证儒

报告日期：2019.9.22

**成绩评定**

|  |  |  |
| --- | --- | --- |
| 实验完成质量（70分） | 报告撰写质量（30分） | 总成绩 |
| 实验步骤清晰、详细、深入，实验记录真实完整等 | 报告规范、完整、通顺、详实 |
|  |  |  |

**目 录**

[1.实验目的与要求 3](#_Toc22498140)

[2.实验内容 3](#_Toc22498141)

[3.实验过程 5](#_Toc22498142)

[3.1 任务 1 5](#_Toc22498143)

[3.2 任务2 7](#_Toc22498144)

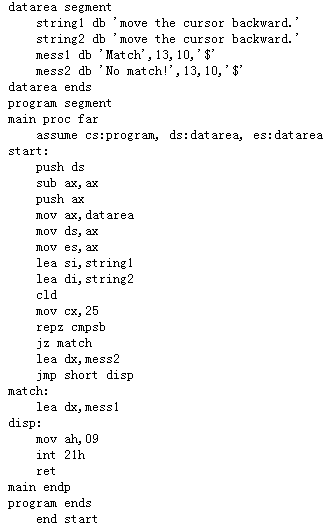
[4.总结与体会 12](#_Toc22498145)

# 1.实验目的与要求

1. 掌握汇编源程序编辑工具、汇编程序、连接程序、调试工具 TD 的使用；
2. 理解数、符号、寻址方式等在计算机内的表现形式；
3. 理解指令执行与标志位改变之间的关系；
4. 熟悉常用的 DOS 功能调用；
5. 熟悉分支、循环程序的结构及控制方法，掌握分支、循环程序的调试方法；
6. 加深对转移指令及一些常用的汇编指令的理解。

# 2.实验内容

任务 1. 阅读以下程序，根据指令的执行流程，说明程序实现的目标，并采用汇编程序对改代码进行编辑、编译、连接和调试。



要求：

1. 查阅资料，说明指令“cld”和“repz cmpsb”的功能；
2. 查阅资料，说明“int 21h”的功能；
3. 借助TD工具，观察指令“repz cmpsb”完成后，哪些标志位发生了变化。
4. 程序中的两条“push”指令的作用是什么，有没有感觉代码中缺少与“push”指令相关的操作指令？如果有，尝试完善该段代码，并观察程序运行效果。
5. 运行程序，说明当前程序运行的结果，然后修改字符串string2，让程序产生不同的结果。

任务 2. 阅读下列程序, 并指出程序执行之后, 以BUF2、BUF3、BUF4 为首址的3 个字节存储区中存放的数据。

|  |  |
| --- | --- |
|  |  |

要求：

1. 分别记录执行到“ MOV CX，10”和“INT 21H”之前的(BX)、(BP)、(SI)、(DI)各是多少。
2. 记录程序执行到退出之前数据段开始，40个字节的内容，指出程序运行结果是否与设想的一致。
3. 在标号LOPA前加上一段程序，实现新的功能：先显示提示信息“Press any key to begin!”, 然后，在按了一个键之后继续执行 LOPA 处的程序。

操作提示：使用 TD.EXE 调试程序时，应先单步执行各个语句，每执行一条语句，都应观察数据段中的内容以及相应寄存器的变化。首先注意观察对DS寄存器的赋值过程，并在TD的数据窗口定位待观察的数据区位置。其次，单步执行循环体两遍且正确理解了循环体语句的含义后，可在“MOV AH, 4CH”处设置断点，然后直接执行到断点处，回答(1)和(2)的问题。完成（3）的内容，涉及到“INT 21H”相关的输入和输出操作，参考任务1中显示字符串的方法。

# 3.实验过程

## 任务 1

1. DF(Direction Flag)是方向标志位，作用是表示DI 以及 SI这两个自增（减）的偏移地址寄存器的自增（减）方向。当DF为1时，DI与SI自增，当DF为0时，DI与SI自减。CLD的作用就是对DF取反

REPZ CMPSB的作用是如果DS:SI和ES:DI所指向的两个字节相等，则SI和DI都自增1，然后继续比较下一位，否则SI和DI都自增1

1. INT 21H是DOS的系统调用，目的是减轻程序员负担，具体作用要结合上一条指令和参考系统API文档确定，相当于调用一段子程序
2. 没有标志位发生了变化

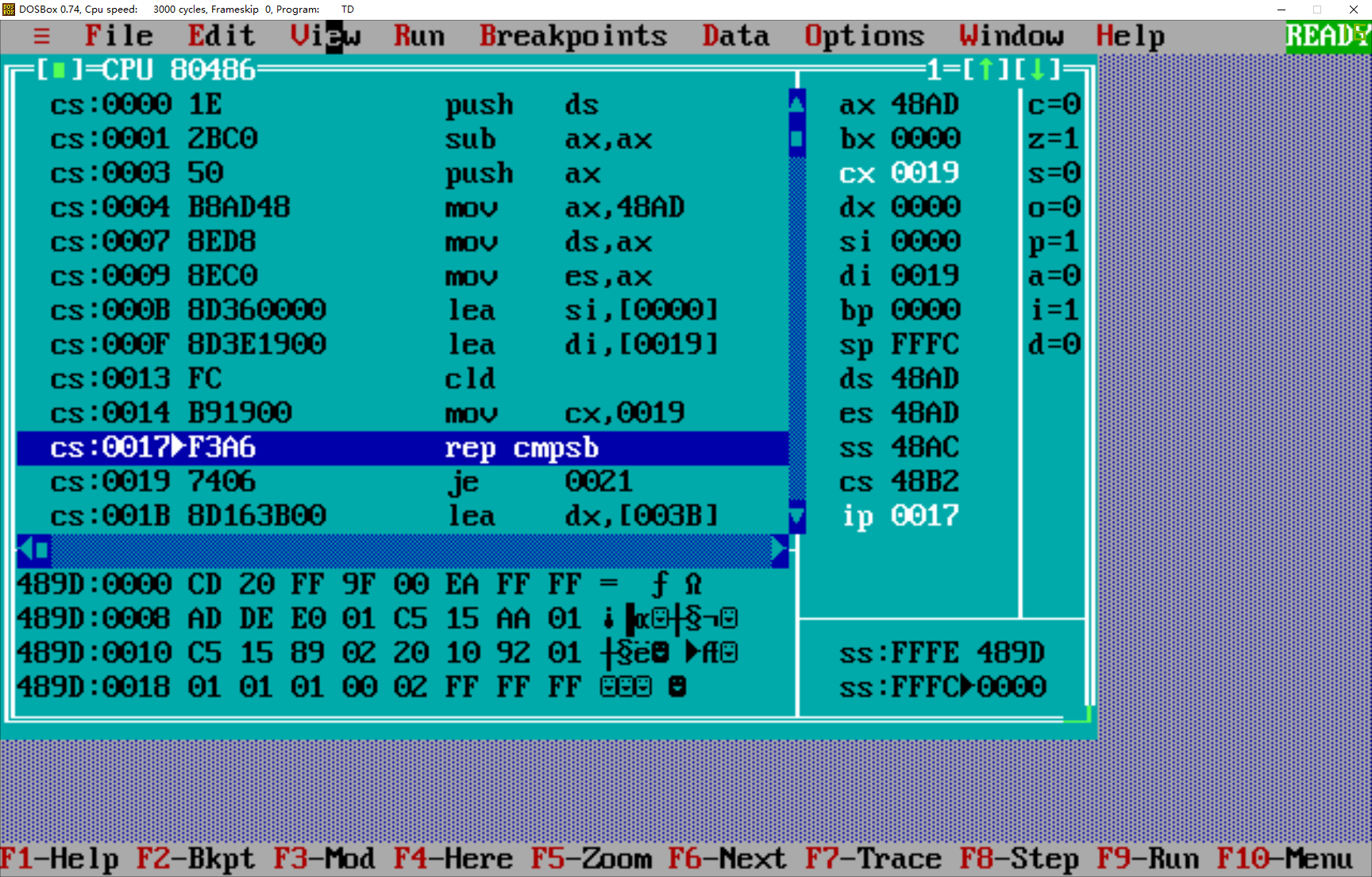


图1.1.3-1 rep语句执行之前

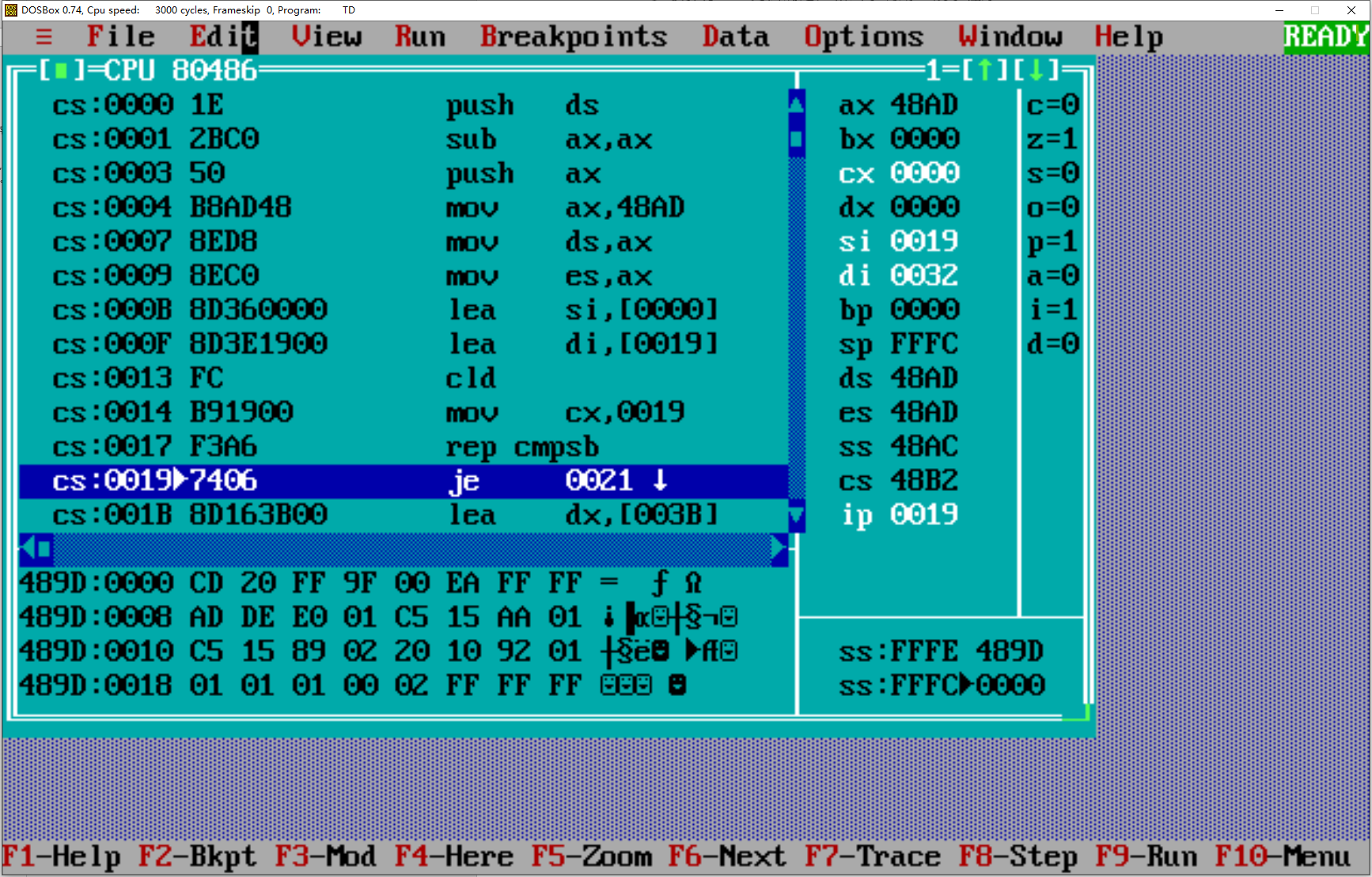


图1.1.3-2 rep语句执行之后

1. PUSH的作用是入栈。虽然没有直接的POP出栈指令，但是程序最后的RET会执行POP类似的作用。汇编程序开始时的push ds; sub ax, ax; push ax; 这三条语句是为后面的ret做准备,ret即执行pop ip; pop cs，执行完ip的值为0，cs的值为ds。程序转移到程序PSP的起始地址，起始地址的前两个字节放的是int 20指令,用于返回DOS。不过不推荐这样做，现在常用mov ax 4c00h;int 21h;终止程序，返回值为al

完善后的代码：

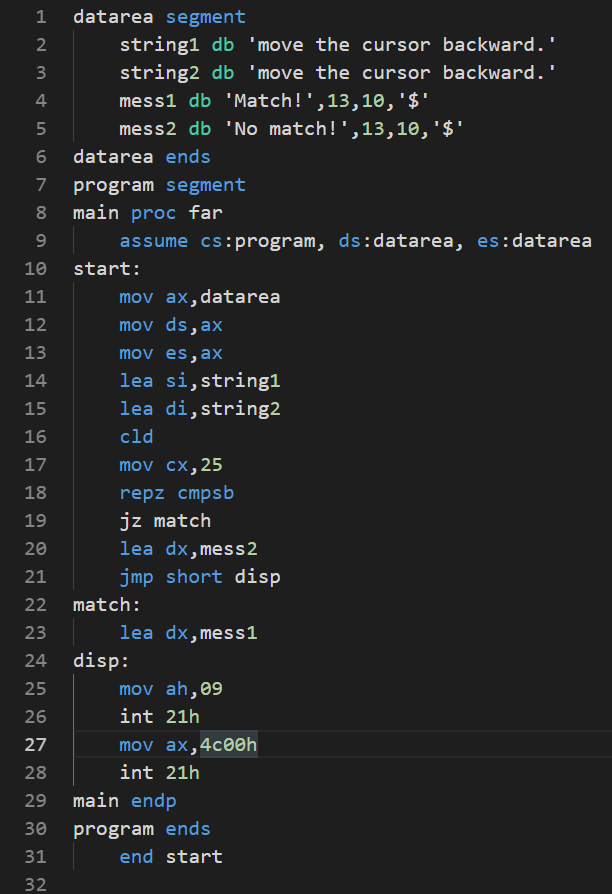


图1.1.4-1 完善后的代码

完善后的代码运行结果：

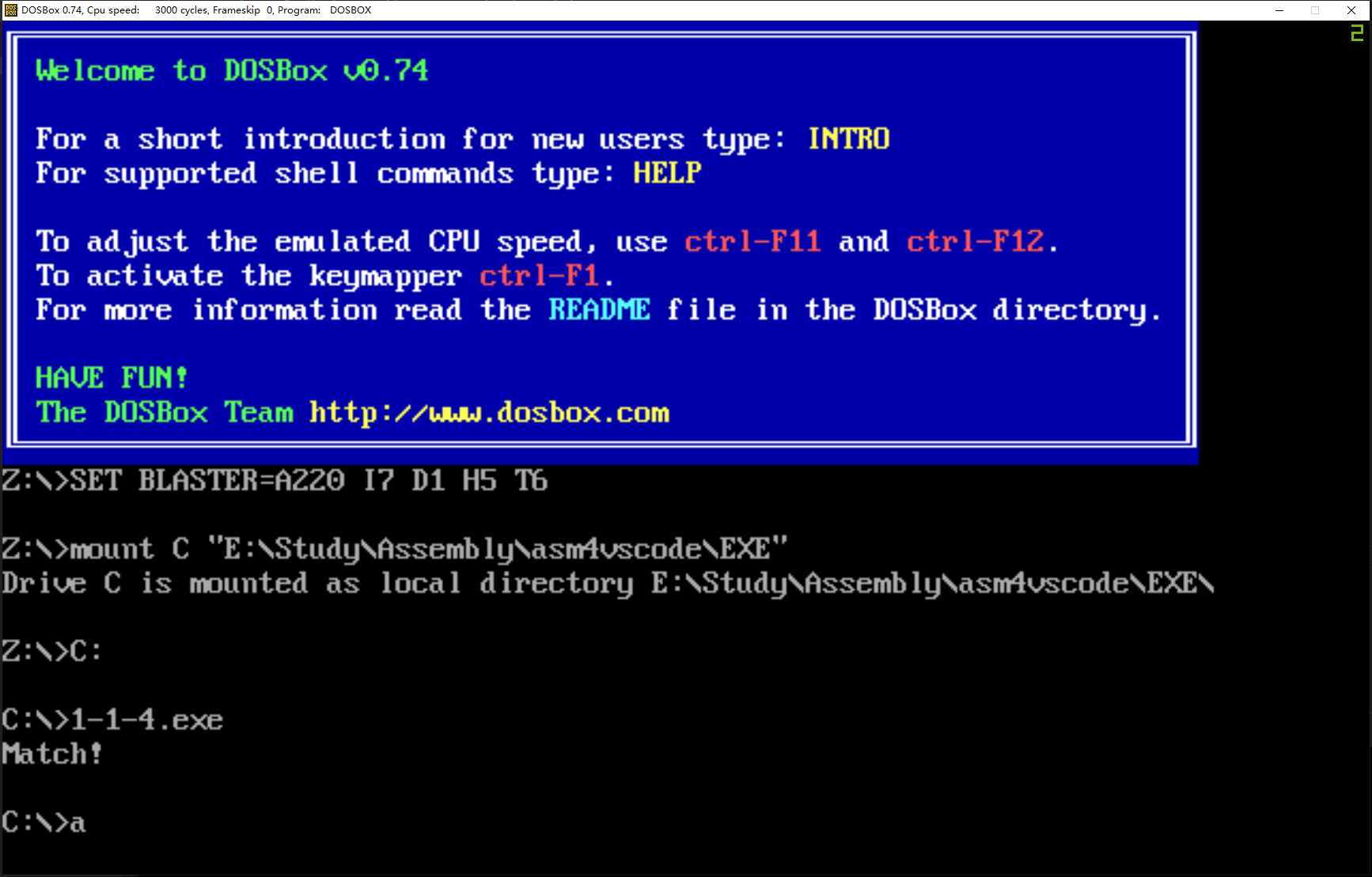


图1.1.4-2 完善后的代码运行结果

## 任务2

1. 执行到mov cx, 10之前：

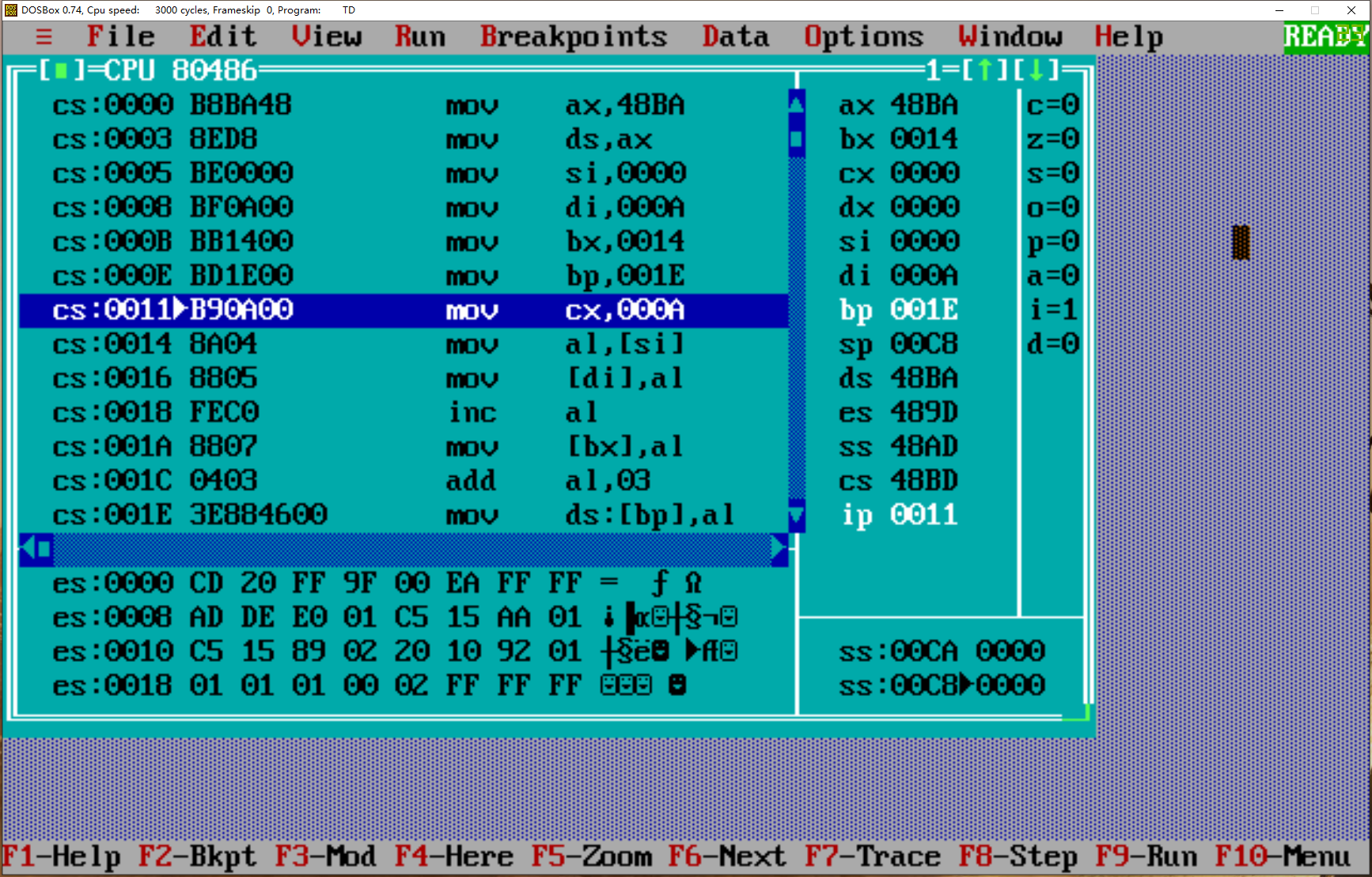


图1.2.1-1 各寄存器数值

此时bx=0014H, bp=001EH, si=0000H, di=000AH

执行到int 21h之前：

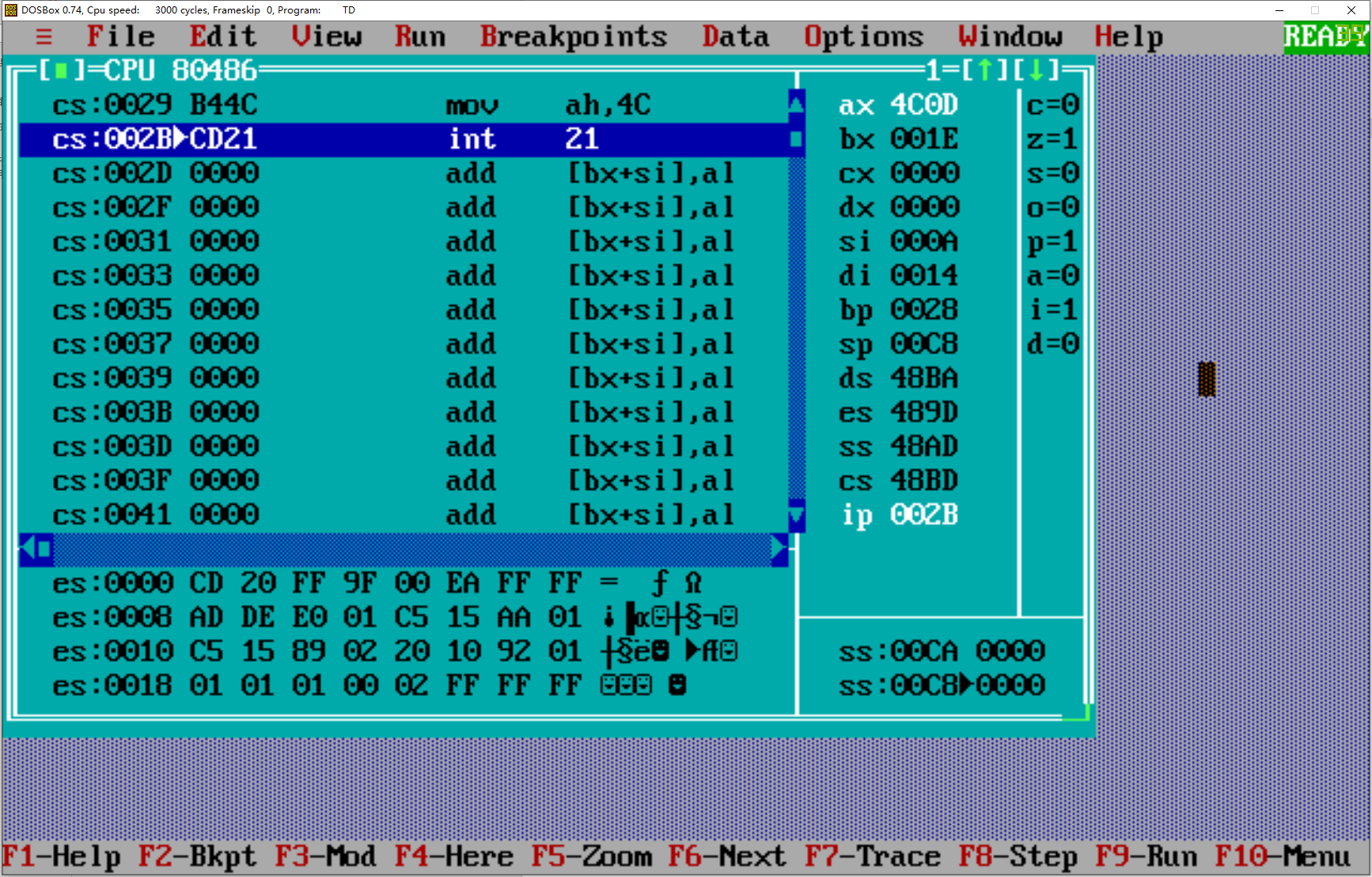


图1.2.1-2 各寄存器数值

此时bx=001EH, bp=0028H, si=000AH, di=0014H

（2）退出之前DS段开始40个字节的内容：

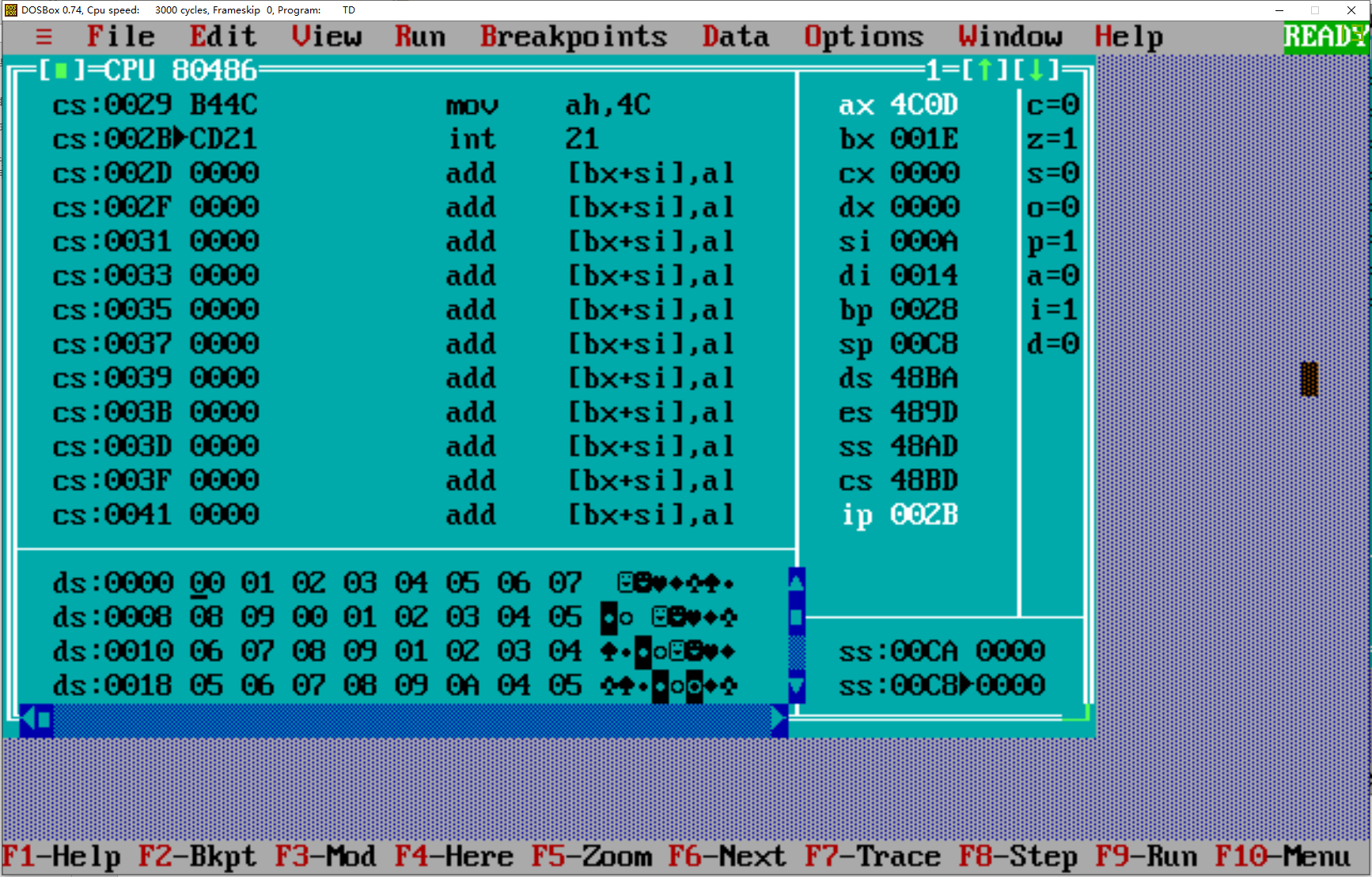
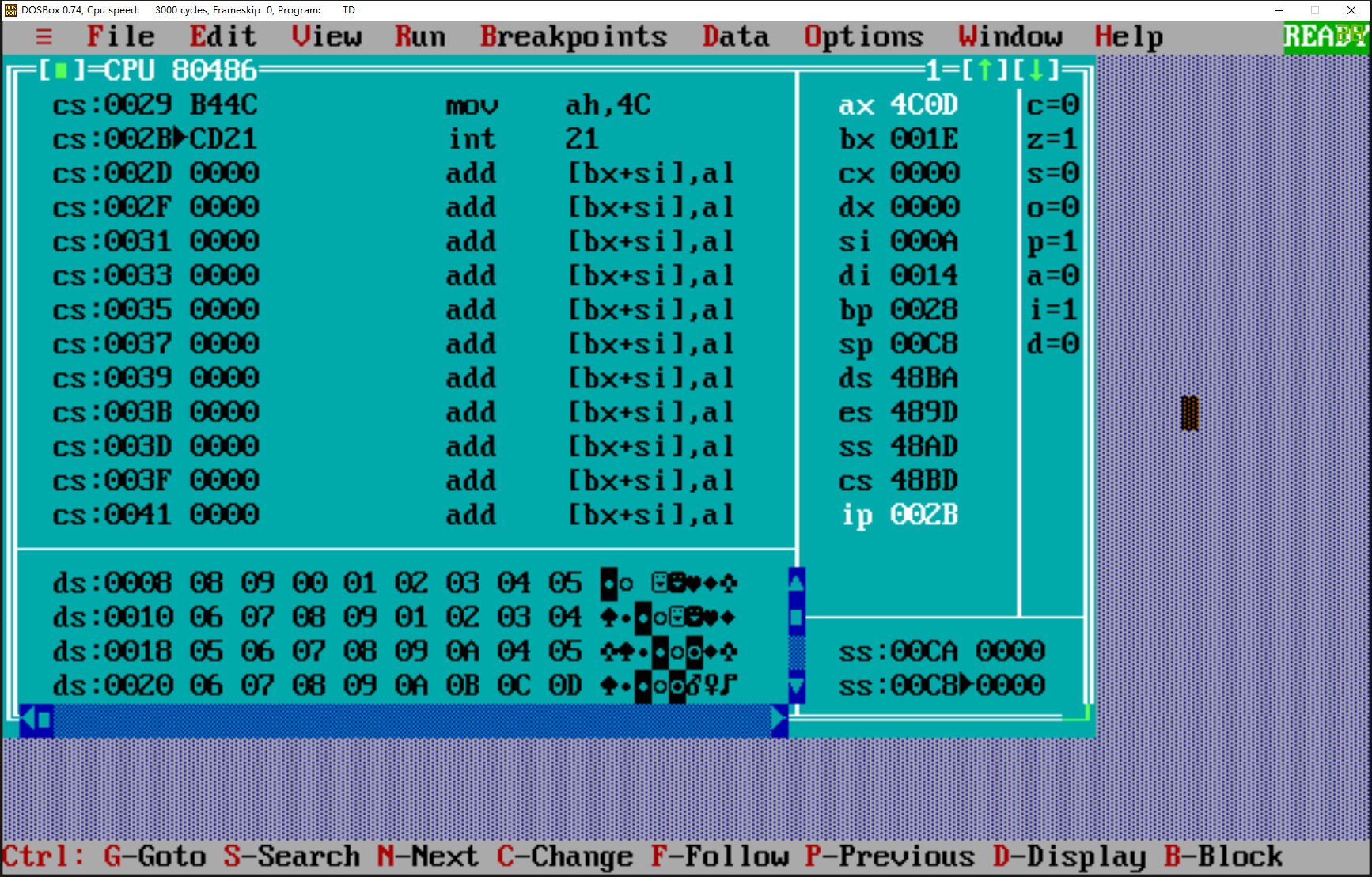


图1.2.2-1退出之前DS段的内容（1）

图1.2.2-2退出之前DS段的内容（2）

运行结果与设想的一致

1. 更改后的程序如下：

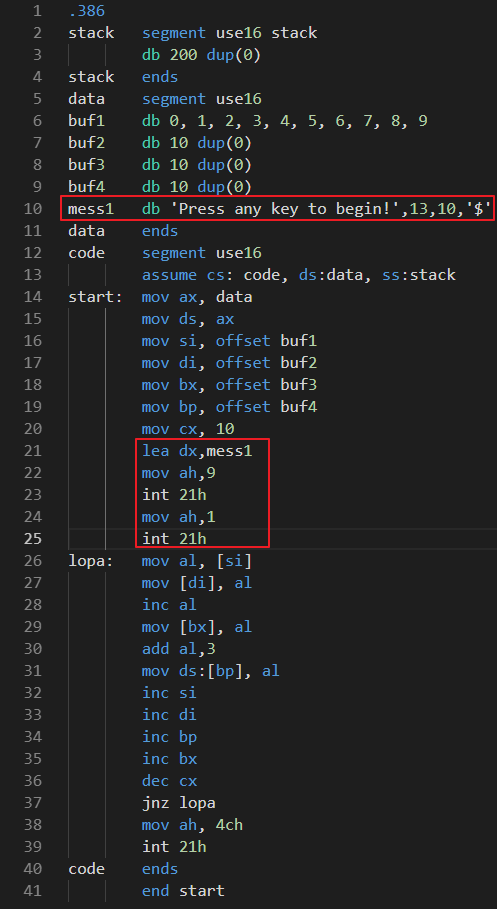


图1.2.3-1 修改后的程序截图

运行结果：

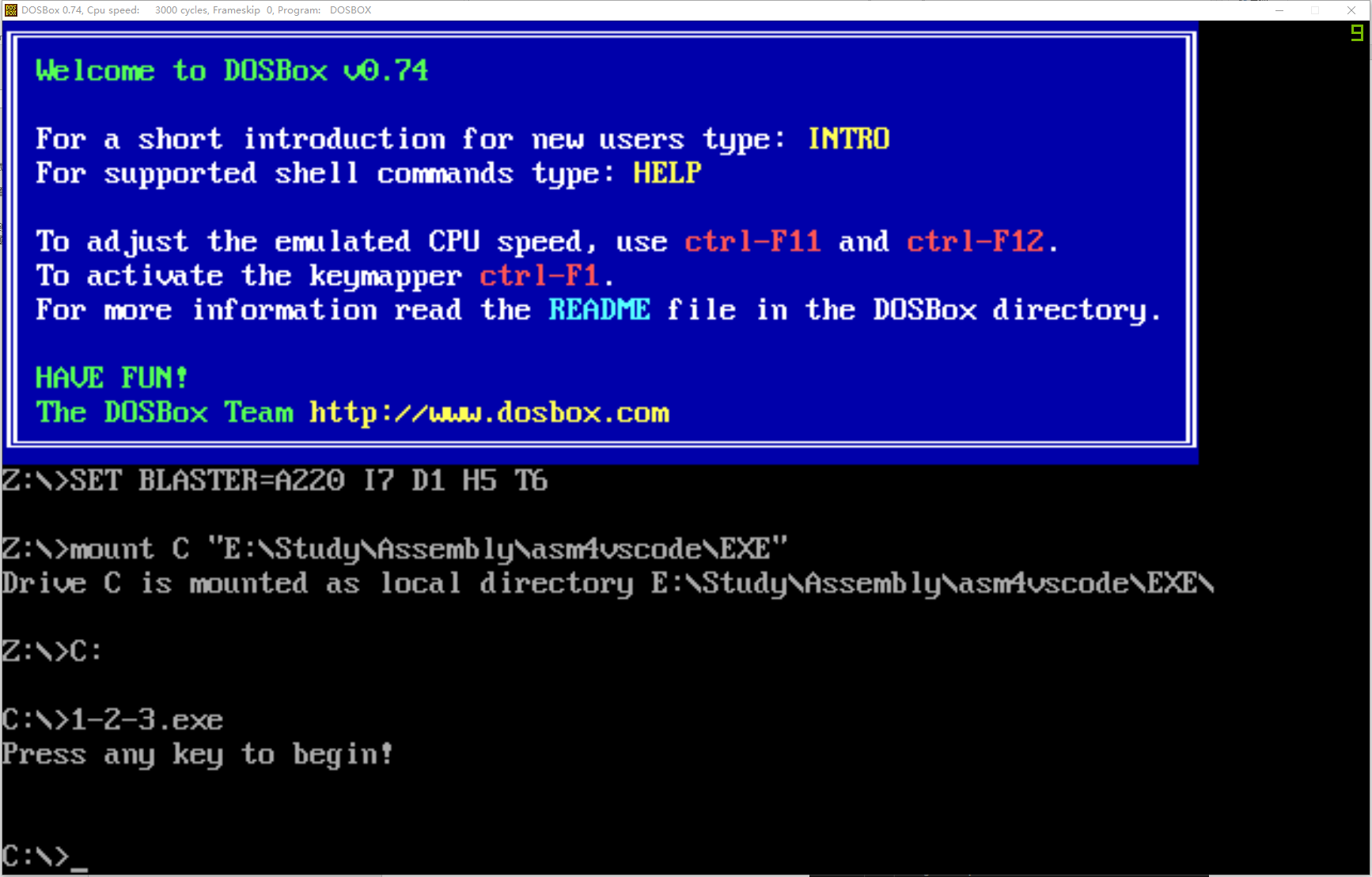


图1.2.3-2 修改后的程序运行结果

程序执行之后，BUF2、BUF3、BUF4为首地址的3个字节存储区中存放的数据如图所示：

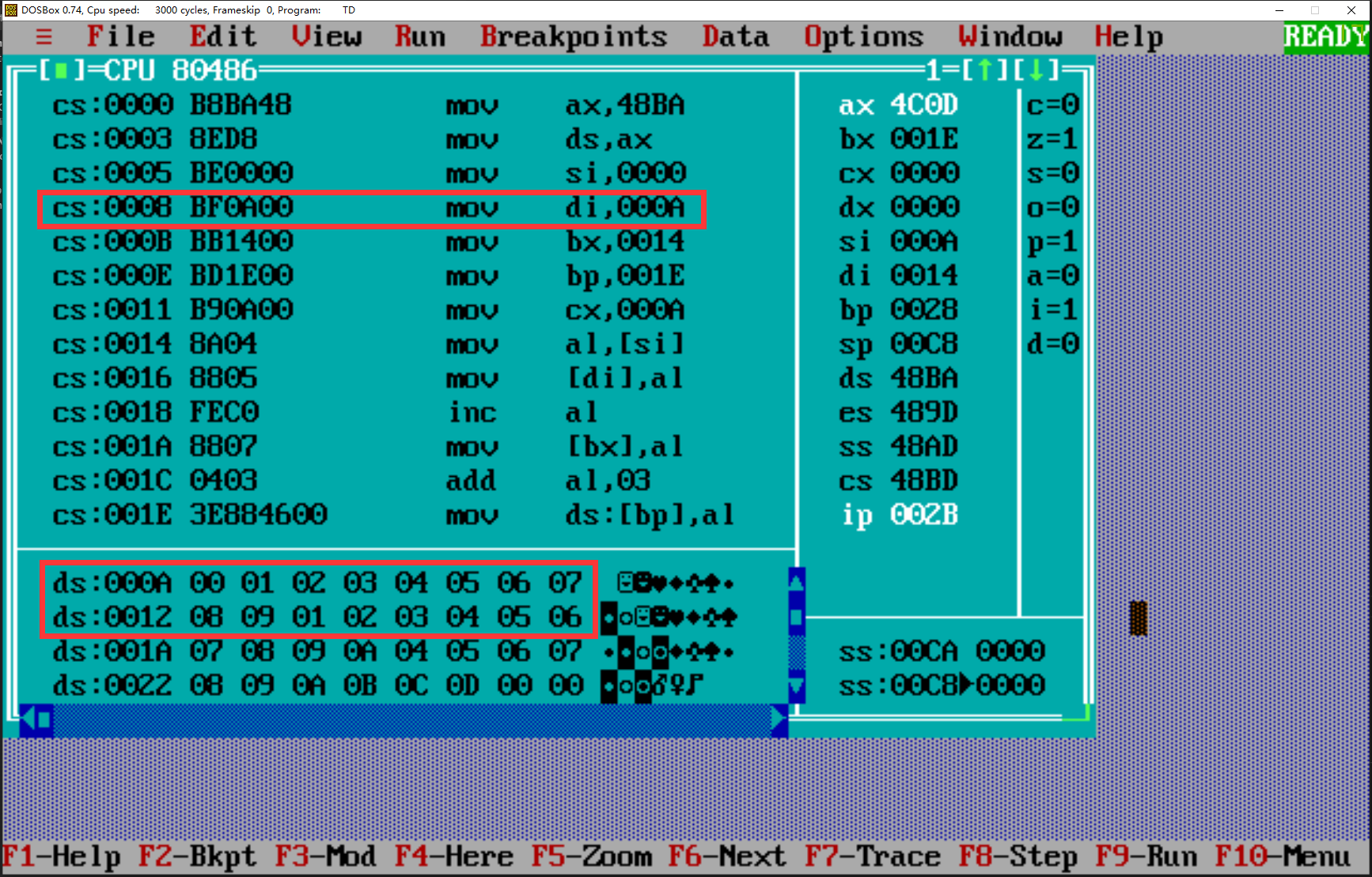


图1.2-1 程序执行后BUF2开始的数据

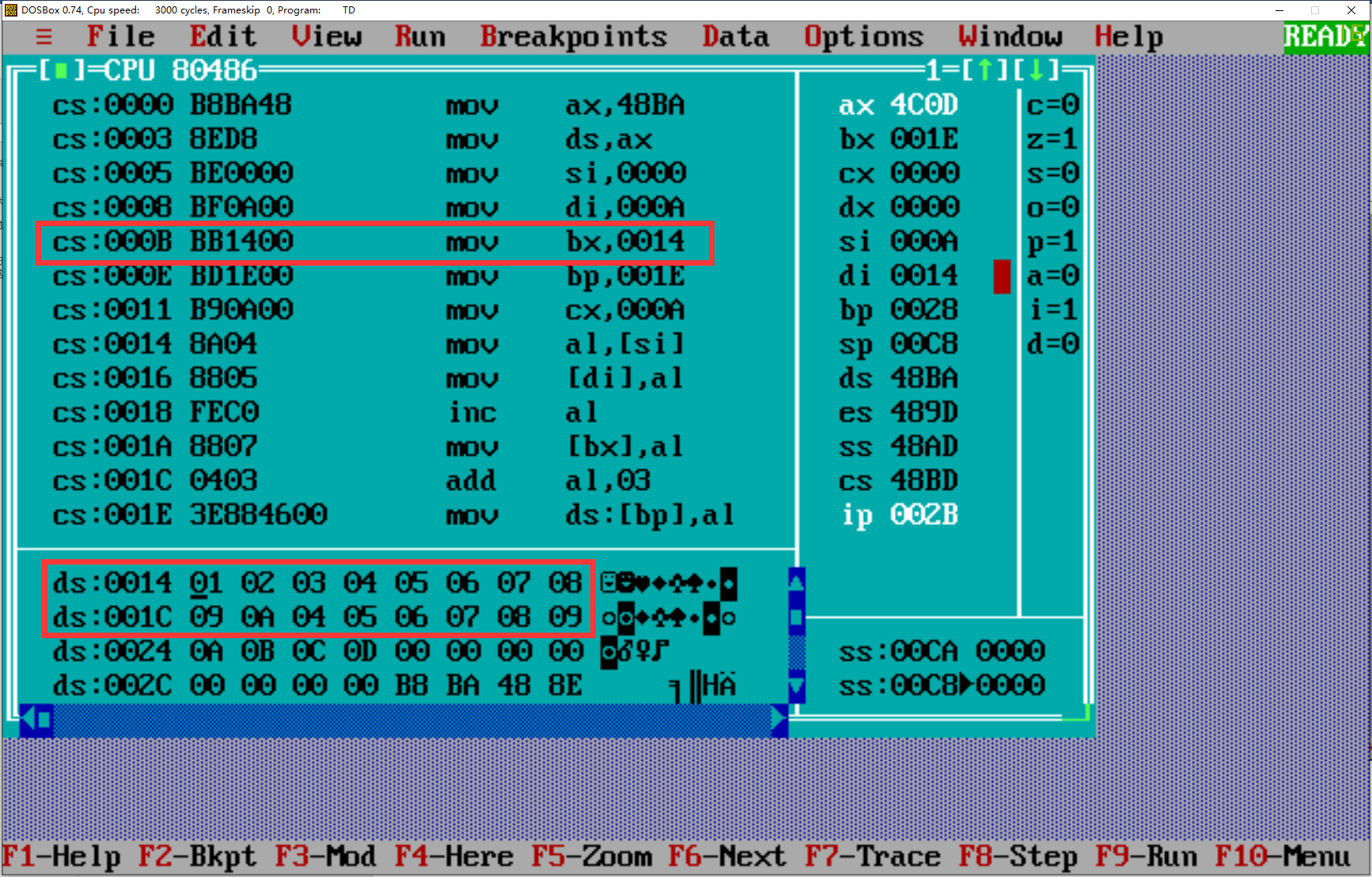


图1.2-2 程序执行后BUF3开始的数据

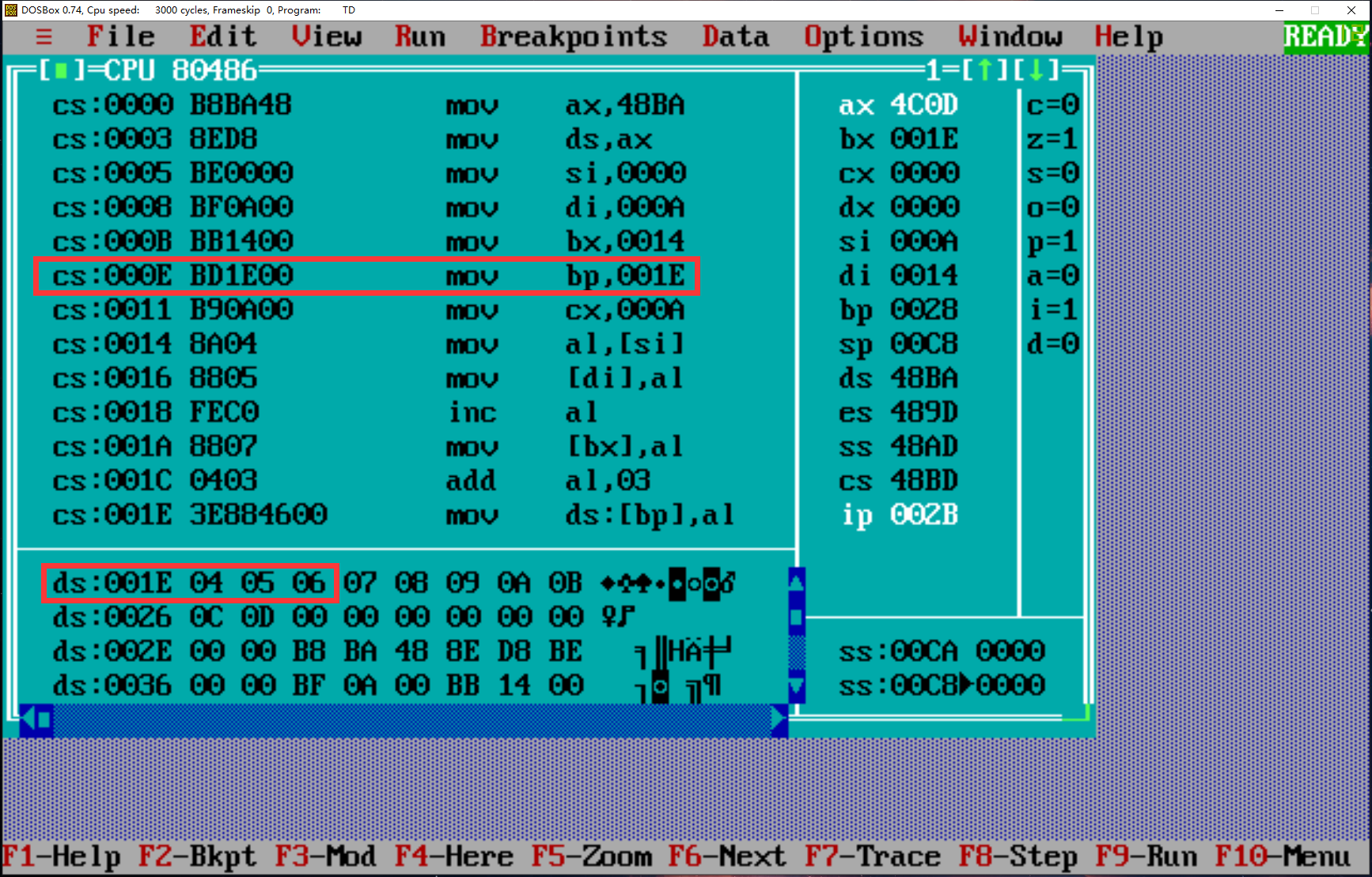


图1.2-3 程序执行后BUF4开始的数据

# 4.总结与体会

本次汇编语言程序设计课程实验是我学习汇编语言进行的第一次实验。在这之前我几乎没有怎么接触过完整的汇编代码，也没有完整地运行和调试过汇编代码。平时课堂上我们大多是“纸上谈兵”，只学习到了各种各样、花样繁多的指令，学习它们的用法、各种语法。而只有通过真正的上机实验，才能将这些理论知识用于实践。

汇编实验，是让我目前为止最接近硬件的实验，因为它总是和特定的机器硬件绑定在一起。在C语言发明之前，过去的程序员先辈们几乎都在用汇编语言来写软件、写操作系统、写驱动程序。虽然现代程序开发很少很少再采用汇编语言，但这不代表学习汇编语言是对我们没有用的。恰恰相反，正是因为现代的大多数程序员用着很多的高级语言（比如Java, Python），他们对硬件、对系统底层的了解比原来更少了，而了解一个系统的底层，有时候可以让我们从一个更全面的角度来考虑程序，比如我们以后在写C语言程序的时候，脑子里自然会有一些汇编指令与之对应，而我们也可以从汇编角度来对一些程序进行底层优化——而这种优化，往往是高级语言所不具备的——它们往往抽象了很多的底层细节，往往连指针和地址的概念都不存在。汇编可以让程序员自由掌控几乎一切程序的细节——当然，这带来的代价就是开发时间和代码量的增大。

本次实验让我复习了课上所学的一些指令，同时对于课上没讲过的部分，也都通过查找资料等方式圆满地完成了任务。同时我也初步掌握了DOSBOX和TD调试的用法。了解了一些DOS的基础知识。