# 实验一　DEBUG 的使用

# 实验报告

x

姓名：赵文亮

学号：2016011452

班级：自 64

日期：2018 年 3 月 26 日

# 目录

# 1 实验目的

1. 练习使用 DEBUG 调试程序。
2. 了解汇编语言上机过程。

# 2 实验准备

1. 自学 DEBUG 的使用。DEBUG 中常用命令及其用法如下：
   - A: 输入汇编语句。
   - D: 显示内存。在后面可以用段基址: 偏移地址的格式指定显示的内存段。
   - G: 执行程序。该命令会从当前的 CS:IP 所指的代码段的指令往下执行，直到遇到中断命令。也可以在 G 后面指定偏移地址，来指定执行结束的代码行。
   - R: 显示寄存器的值。后面不加参数时，会显示所有寄存器的值；后面也可以加入指定的寄存器名称，便可以查看、修改该寄存器的内容。
   - P: 单步执行。运行命令后会显示出所有的寄存器的内容，以及下一步的指令。遇到子程序时不进入，而是将整个调用过程作为整体执行，类似于一般 IDE（如 Visual Studio）的“逐过程调试”。
   - T: 执行并显示（跟踪），单步执行，但是遇到子程序时会进入，类似于一般 IDE 的“逐语句调试”。
   - N: 文件存盘或读取时指定文件名。
   - W: 文件或数据写盘。
   - L: 文件或数据读盘。
   - Q: 退出 DEBUG 环境。
2. 读懂实验内容（一）（二）程序中各条指令，说明程序功能。
   - 任务一的程序利用条件转移语句实现循环，在 DS:0200 起始的 16 个字节中依次填入 00～0F。完整程序和详细分析见 4.1 节。
   - 任务二的程序实现了数据的复制。该程序首先将数据段进行初始化，将 BUFFER1 中初始化为 0～F，将 BUFFER2 中初始化为 10 个字节的 0。在代码段中，使用条件转移语句将 BUFFER1 中的数据依次复制到 BUFFER2 中。完整程序和详细分析见 4.2 节。
3. （选做）按实验内容（三）的要求编写汇编语言源程序并加上必要注释。仿照任务一的思路，基于任务二的框架，不难按照要求实现功能。其中关键的点在于如何 30H～39H 及 41H～46H 这一组并不连续的值。一种思路是分两次循环，每次分别填入一组连续序列；另一个思路是在大循环中进行判断，当此时填完 39H 的时候想办法直接跳到 41H。完整程序和详细分析见 4.3 节。

# 3 实验内容

首先进入 DOS 环境下，并通过 cd 命令修改当前目录[1]（编译、链接、调试工具所在目录）。

## 3.1 DEBUG 的使用练习

实验步骤如下：

### 3.1.1 DEBUG 下程序的输入和存盘

1. 命令行中输入 DEBUG.EXE 启动 DEBUG 程序。
2. 输入 A，进入到程序输入模式。
3. 将 4.1 中的代码依次输入。
4. 在空行处回车即可结束输入，之后返回 DEBUG 命令行。
5. 利用 R 命令修改 BX 和 CX 的值分别为 0000 和 2400，再使用 N 命令指明要保存的文件名，最后使用 W 命令存盘。

### 3.1.2 DEBUG 下程序的读取和运行

1. 使用 N 命令指定要读取的文件名，再使用 L 命令读取程序。
2. 使用 U 命令查看反汇编结果。
3. 使用 G 命令运行程序。程序会在断点处（INT）停止。也可以使用 P、T 进行单步调试。

### 3.1.3 运行结果的查看

可以使用 D 和 R 命令查看寄存器和内存中的内容。

1. 运行前，使用命令“D DS:0200”查询内存中的内容。
2. 运行后，再次执行上述命令。可以看到，以 DS:0200 开始的 16 个字节被写入了 00～0F。

## 3.2 汇编语言上机

1. 使用文本编辑器，输入 4.2 节中的程序，并保存为 PROG2.ASM。
2. 使用 TASM 编译源程序，生成 PROG2.OBJ。
3. 使用链接程序 LINK 连接 OBJ 生成 PROG2.EXE。
4. 运行 DEBUG，使用 L 命令读取 PROG2.EXE。

[1]事实上，如果已经将编译工具等的路径加入了环境变量中，则不必修改目录

5. 使用 U 命令，查看反汇编结果。此时要注意第一行语句。例如 MOV AX, 076A 表示 DATA 的段基址为 076A[2]。

6. 执行命令“D 076A:0000”查看该数据段内存。如图 1 所示。可见该数据段已经按照代码中所示进行正确的初始化。

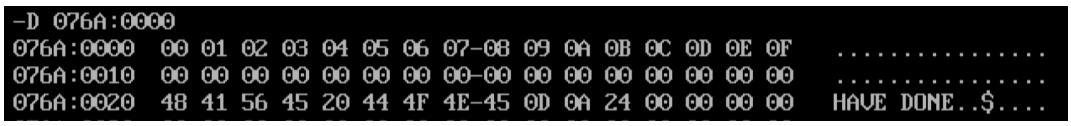


图 1: 任务二运行前内存内容

7. 使用 G 命令运行程序。

8. 再次执行命令“D 076A:0000”查看内存，如图 2 所示。可见已经成功将 BUFFER1 中的内容复制到 BUFFER2 中。

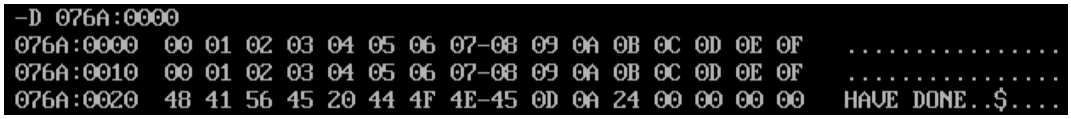


图 2: 任务二运行后内存内容

### 3.3 选做内容

1. 用文本编辑器输入 4.2 中所示程序并保存。

2. 与任务二相似，加载程序，反汇编查看 DATA 的段基址为 076A。

3. 运行前执行“D 076A:0000”查看内存。如图 3 所示。可见已经将对应数据初始化。

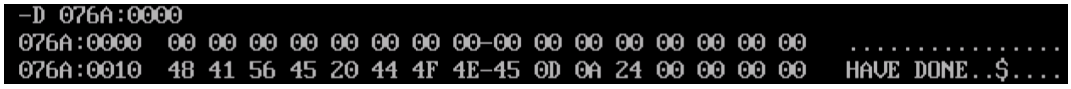


图 3: 任务三运行前内存内容

4. 使用 G 命令运行程序。

5. 执行“D 076A:0000”查看内存，如图 4 所示。其中前 16 个字节中被填入 ASCII 字符 0~F。

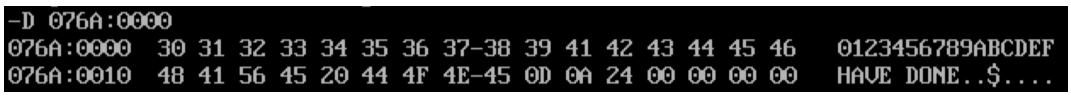


图 4: 任务三运行后内存内容

[2]不同电脑上的结果可能不同

# 4 实验程序及说明

## 4.1 任务一

```
1 MOV SI, 200
2 MOV CX, 10
3 MOV AL, 0
4 MOV [SI], AL
5 INC SI
6 INC AL
7 DEC CX
8 JNZ 108
9 INT 3
```

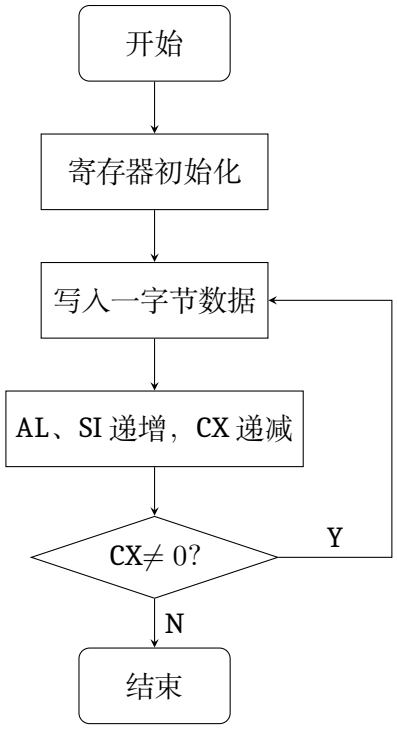


图 5: 任务一流程图

- 1-3：寄存器初始化。

  指定了初始的偏移地址、循环次数、填入数据的初值。

- 4：将 AL 中数据装入 SI 所指内存的一个字节。

- 5-7：数据、目标地址递增；循环次数递减。

- 8：条件转移，实现循环操作。

- 9：用于程序的中断，便于在 DEBUG 模式下正常执行。

代码思路比较清晰，通过循环实现功能。其流程图如图 5 。

## 4.2 任务二

```
1  NAME MY_PROGRAM          ;程序模块名
2  DATA SEGMENT             ;数据段开始
3  BUFFER1 DB 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
4          DB 0AH, 0BH, 0CH, 0DH, 0EH, 0FH
5  BUFFER2 DB 10H DUP(0)
6  MESS    DB 'HAVE DONE', 13, 10, '$'
7  DATA    ENDS             ;数据段结束
8  STACK   SEGMENT PARA STACK
9          DB 100 DUP(?)
10 STACK   ENDS             ;堆栈段结束
11 CODE    SEGMENT          ;代码段开始
12         ASSUME CS: CODE, DS: DATA, ES:DATA, SS:STACK
```

```
13
14 START:  MOV AX, DATA
15        MOV DS, AX        ;建立用户数据段
16        MOV ES, AX
17        LEA SI, BUFFER1
18        LEA DI, BUFFER2
19        MOV CX, 10H
20 NEXT:  MOV AL, [SI]
21        MOV [DI], AL
22        INC SI
23        INC DI
24        DEC CX
25        JNZ NEXT
26        LEA DX, MESS      ;指向提示字符串
27        MOV AH, 9         ;显示字符串的功能号
28        INT 21H           ;DOS功能调用
29        MOV AH, 4CH       ;退出用户程序的功能号
30        INT 21H           ;DOS功能调用
31 CODE   ENDS              ;代码段结束
32 END    START             ;整个源程序结束，并指明第一条执行语句
```

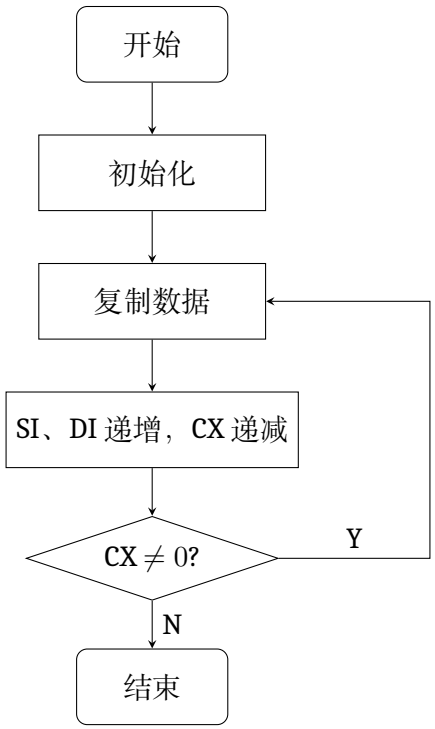


图 6: 任务二流程图

- 2-7：数据段初始化。在内存中开辟一段空间（以 DATA 为基址），并以此写入 BUFFER1、BUFFER2、MESS 中的内容。
- 8-10：堆栈段初始化，深度为 100 字节。
- 12：指明各个段基址。
- 14-16：初始化段基址。
- 17-18：初始化偏移地址，
- 19：初始化循环次数，总共循环 16 次。
- 20-21：将数据从 [SI] 中赋值到 [DI] 中。
- 22-24：递增变址指针，递减循环次数。
- 25：条件转移，实现循环操作。
- 26-30：DOS 命令操作，用于显示提示字符串和退出程序。

### 4.3 任务三

任务三的代码是以任务二为框架，任务一为原理修改而来。为了实现将 30H～39H 及 41H～46H 一次性赋值到对应内存区间，我在循环中使用了一个判断。如第 19 行所示，每次循环都要比较一下当前 AL 的值于 3AH 的大小，如果比较结果为 0，则说明刚刚将 39H 写入，此时应该将 41H 赋给 AL，下次将 41H 写入内存。我使用条件转移语句，当比较结果不为 0 时就跳过 “MOV AL, 41H” 这行代码。由此实现了功能。其他代码与任务二类似，在 4.2 中已经做过详细的解释。流程图如图 7 所示。

## 5 完成情况和心得体会

本次实验由于预习十分充分，我很顺利地在 10min 之内就完成了实验。预习时，我在本机上安装了 TASM 的 Win10 版本（配有 DOSBOX 虚拟环境），在此环境下即可使用 DEBUG 程序进行调试。

一开始，由于对命令不是特别熟悉，对着代码一头雾水，不知道如何查看代码的效果。后来仔细分析后，我知道了使用 D 命令查看内存，而内存的段基址和偏移地址需要分析代码的内容来获得。

在我不断地尝试中，我对汇编和 DEBUG 逐步熟悉。后来做选做任务的时候我也很轻松就写出了代码。我的第一版代码是使用两个循环，第一个循环写入 0～9，第二个循环写入 A～F。后来，我觉得这种做法不够优雅，代码量有点大，于是我就改成了现在的版本，即只通过一个循环，其中加入判断来实现。

上面的工作我在预习阶段就做的很好，实验上机的时间里我很顺利就完成了。完成任务后，我还为周围的同学解答问题。在帮助同学的过程中，我对 DEBUG 的理解更加深刻了。例如，每次使用 A 命令写入一段汇编程序前一定要先退出 DEBUG，否则存盘的过程中会保存本次进入 DEBUG 后输入的所有代码，很多同学在这个问题上没有注意，导致反汇编结果不对；DATA 的段基址取决于电脑，不同的电脑不一定相同，需要通过反汇编方式查看。

本次实验使我收获了很多，也增加了我对汇编语言的兴趣。感谢老师和助教的指导和付出！

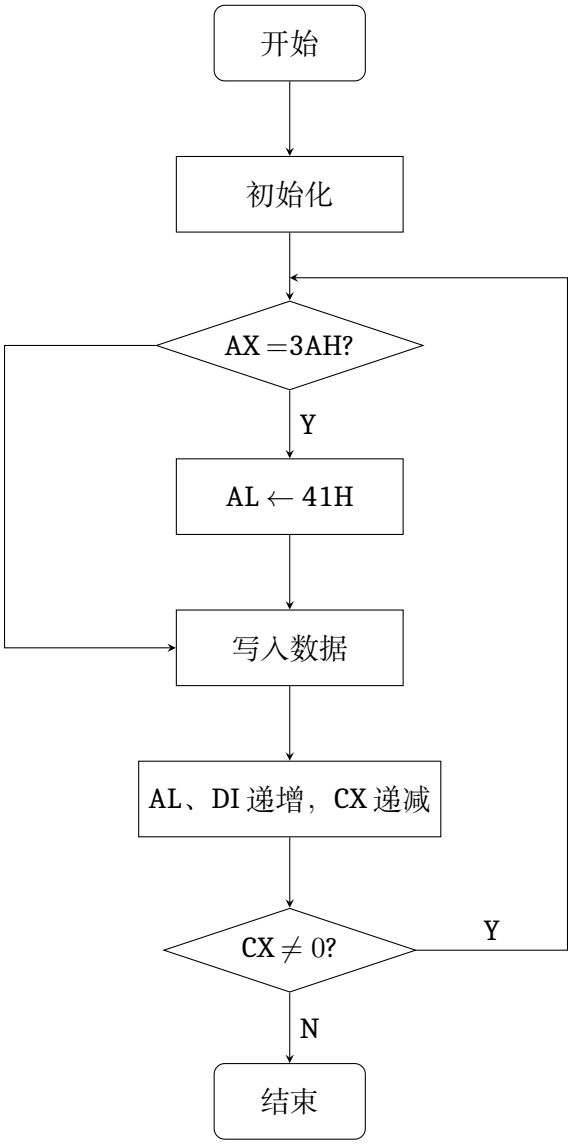


图 7: 任务三流程图