

## 建立运行汇编语言程序

### 一、实验目的:

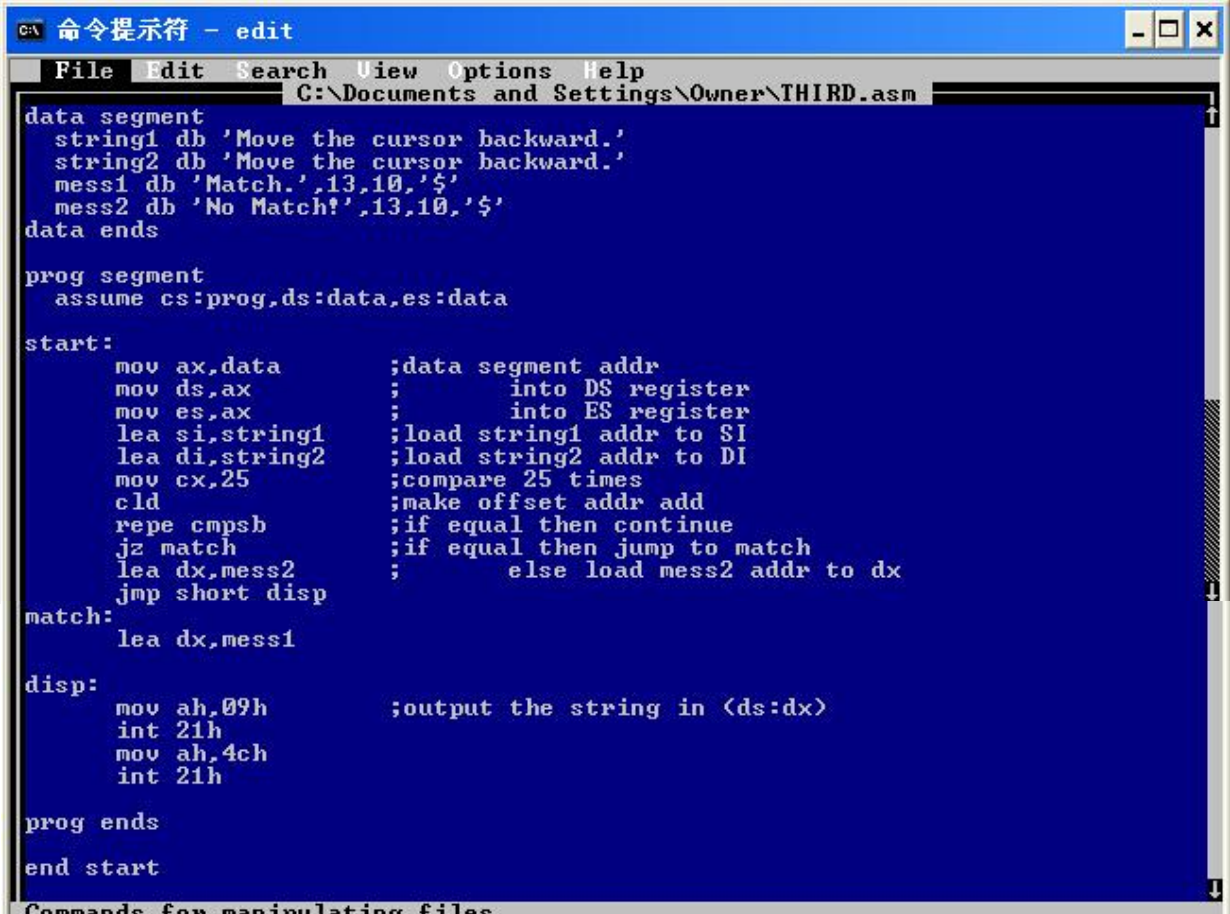
- 1.掌握汇编语言程序格式及其它命令的作用。
- 2.掌握汇编、连接、运行汇编程序的全过程，并能检查修改程序简单错误。

### 二、实验内容:

- 1.编写程序，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'No match!'。
- 2.编辑、汇编、连接这个汇编语言源程序，形成.EXE 文件。
- 3.通过 DEBUG 调试运行该可执行文件，查看运行结果。

### 三、实验主要步骤:

- 1.根据要求编制汇编源程序。使用字处理软件(Edit.exe)编辑该源文件。



```
命令提示符 - edit
File Edit Search View Options Help
C:\Documents and Settings\Owner\THIRD.asm

data segment
    string1 db 'Move the cursor backward.'
    string2 db 'Move the cursor backward.'
    mess1 db 'Match.',13,10,'$'
    mess2 db 'No Match!',13,10,'$'
data ends

prog segment
    assume cs:prog,ds:data,es:data

start:
    mov ax,data           ;data segment addr
    mov ds,ax             ; into DS register
    mov es,ax             ; into ES register
    lea si,string1         ;load string1 addr to SI
    lea di,string2         ;load string2 addr to DI
    mov cx,25              ;compare 25 times
    cld                   ;make offset addr add
    repe cmpsb             ;if equal then continue
    jz match               ;if equal then jump to match
    lea dx,mess2            ; else load mess2 addr to dx
    jmp short disp

match:
    lea dx,mess1

disp:
    mov ah,09h             ;output the string in <ds:dx>
    int 21h
    mov ah,4ch
    int 21h

prog ends
end start

Commands for manipulating files
```

2.使用 Masm、Link 程序汇编、连接该文件，形成可执行的.EXE 文件。并执行.exe 文件。

```
C:\DOCUME~1\Owner>MASM
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

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

49328 + 449245 Bytes symbol space free

0 Warning Errors
0 Severe Errors
```

Masm  
汇编

```
C:\DOCUME~1\Owner>LINK

Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

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

Link  
连接

```
C:\DOCUME~1\Owner>THIRD
Match.
```

运行.EXE，结果是 Match。正确。

3.使用 Debug 调试运行这个.EXE 文件，并查看运行结果。对实验内容修改两个字符串的内容，使它们互不相同，再观察程序的运行结果

```
C:\DOCUME~1\Owner>DEBUG THIRD.EXE
Microsoft (R) Symbolic Debug Utility Version 4.00
Copyright (C) Microsoft Corp 1984, 1985. All rights reserved.

Processor is [80286]
-R
AX=0000 BX=0000 CX=0079 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=0E70 ES=0E70 SS=0E80 CS=0E85 IP=0000 NU UP EI PL NZ NA PO NC
0E85:0000 B8800E MOV AX,0E80
-U
0E85:0003 8ED8 MOV DS,AX
0E85:0005 8EC0 MOV ES,AX
0E85:0007 8D360000 LEA SI,[0000]
0E85:000B 8D3E1900 LEA DI,[0019]
0E85:000F B91900 MOV CX,0019
0E85:0012 FC CLD
0E85:0013 F3 REPZ
0E85:0014 A6 CMPSB
```

```

-U
0E85:0015 7406          JZ      001D
0E85:0017 8D163B00        LEA     DX,[003B]
0E85:001B EB04          JMP     0021
0E85:001D 8D163200        LEA     DX,[0032]
0E85:0021 B409          MOV     AH,09
0E85:0023 CD21          INT     21
0E85:0025 B44C          MOV     AH,4C                ;'L'
0E85:0027 CD21          INT     21
-G 000F
AX=0E80 BX=0000 CX=0079 DX=0000 SP=0000 BP=0000 SI=0000 DI=0019
DS=0E80 ES=0E80 SS=0E80 CS=0E85 IP=000F  NU UP EI PL NZ NA PO NC
0E85:000F B91900          MOV     CX,0019
-D DS:0 L40
0E80:0000 4D 6F 76 65 20 74 68 65-20 63 75 72 73 6F 72 20  Move the cursor
0E80:0010 62 61 63 6B 77 61 72 64-2E 4D 6F 76 65 20 74 68  backward.Move th
0E80:0020 65 20 63 75 72 73 6F 72-20 62 61 63 6B 77 61 72  e cursor backward
0E80:0030 64 2E 4D 61 74 63 68 2E-0D 0A 24 4E 6F 20 4D 61  d.Match...$No Ma
-G
Match.

```

使用 edit.exe 把 string1 改成 'This is my 3th assembly.' 后再汇编连接，执行结果如下：

```

C:\DOCUMENT~1\Owner>third
No Match!

```

结果正确：

屏幕显示：No Match! 用 debug 调试执行，结果如下：

使用 debug 调试如下：

```

-g 000f
AX=0E80 BX=0000 CX=0079 DX=0000 SP=0000 BP=0000 SI=0000 DI=0018
DS=0E80 ES=0E80 SS=0E80 CS=0E85 IP=000F  NU UP EI PL NZ NA PO NC
0E85:000F B91900          MOV     CX,0019
-d ds:0 l40
0E80:0000 54 68 69 73 20 69 73 20-6D 79 20 33 74 68 20 61  This is my 3th a
0E80:0010 73 73 65 6D 62 6C 79 2E-4D 6F 76 65 20 74 68 65  ssembly.Move the
0E80:0020 20 63 75 72 73 6F 72 20-62 61 63 6B 77 61 72 64  cursor backward
0E80:0030 2E 4D 61 74 63 68 2E 0D-0A 24 4E 6F 20 4D 61 74  .Match...$No Mat
-g
No Match!

Program terminated normally (36)
_

```

此时查看 ds:0 L40 的内容可知 string1 内容发生了变化。

结果是：No Match!

#### 四、实验结果与分析：

把 exe 文件转载到 debug 后，先反汇编，然后用 G 命令执行汇编指令使得 ds:si 和 ds:di 分别指向 string1 和 string2，然后查看此时 ds:0，长度了 40h 的内容，会发现此时 ds:00 的内容对应的 ASCII 码正是 Sring1 的内容，而紧接着是 string2，之后是 mess1 和 mess2 的内容，就是因为 mov 和 lea 指令让 ds 段地址指向了定义的 data 段的首地址，所以在 data segment 里的内容才会被显示。若不执行第一个 lea 指令，则 ds:0

的内容将是不可预知的。也就是说我们定义 `data segment` 其实就是提供一个入口的地址。在程序代码中必须要让对应的寄存器指向它，才能达到我们预期的内容。`Lea` 指令就能很好地满足这个需求。

同时发现在 `debug` 模式下，`G` 命令的使用格式不同，执行的结果也不同。如果单独使用 `G` 则完整执行程序，并退出，返回没有装载任何程序的 `debug`；若使用 `G=[地址]`，则执行到当 `CS:[地址]` 时，退出并返回没有装载任何程序的 `debug`；但是，如果使用 `G [地址]`，那么执行到当 `CS:[地址]` 时，`debug` 依然是装载着当前程序的，并显示当前寄存器的情况，此时 `G` 的功能才是相当于多步 `T` 命令的功能。所以我们为了避免使用多步 `T` 而用 `G` 命令时，一定要注意是没有“=”的。

当然，程序中可把 `repe` 改成 `repne`，只是跳转指令也要必须跟着改。

通过该实验，进一步强化了对 `debug` 调试工具的认识。同时更加熟悉段定义和串处理指令的使用，深刻体会到使用串处理、跳转指令时要时刻注意此时程序执行哪一步，理清思路才能编写好正确的程序。