

目 录

目 录	1
第一章 基础知识.....	2
1.1 全屏幕编辑程序—EDIT.....	2
1.2 汇编语言上机步骤.....	3
1.3 DEBUG主要命令	4
1.4 Debug 应用实例	10
第二章 实习内容.....	12
实习要求.....	12
实验 1 寻址方式与指令练习	12
实验 2 初级程序的编写与调试.....	13
实验 3 加法程序的编写与调试.....	14
实验 4 数据传送练习	14
实验 5 利用DEBUG 调试程序段	16
实验 6 分支程序实验.....	17
实验 7 循环程序实验.....	19
实验 8 子程序实验（一）	21
实验 9 子程序实验（二）	24
实验 10 字符处理程序实验.....	27
实验 11 输入输出实验.....	30
实验 12 中断程序实验.....	34
实验 13 系统调用程序实验.....	38
实验 14 语言接口实验.....	42
第三章 高级接口.....	45

第一章 基础知识

1.1 全屏幕编辑程序 - EDIT

在编写汇编语言源程序时，可以通过全屏幕程序 EDIT 等建立源程序。

一、EDIT 的功能及应用

EDIT 是基于 DOS 平台下的文本编辑工具，具有常规的操作命令和灵活的键盘功能。其主要功能如下：

- 1、有丰富的编辑命令：可以调入已保存的文件进行编辑，可以对字符串进行查找、替换等操作。
- 2、有区域标记定义和操作功能：可以进行区块的定义、剪切、复制、删除等操作，可以设置前景色及背景色。
- 3、灵活的键盘功能。
- 4、方便的打印功能。

二、EDIT 的启动与退出

1、启动

EDIT 的启动同其他 DOS 外部命令一样，只需在 DOS 状态下键入：

EDIT 【文件名】，其中，【文件名】是一个任选项。如果指定文件名，EDIT 就装入指定的文件以供编辑，如果不指定文件，则 EDIT 就创建一个无名文件，其后用户可在存盘时指定其文件名。

2、存盘及退出

按 ALT 键激活菜单，选择 File 菜单，在选择 Save 或 Save As 或 Exit。分别为保存、另存、为和退出。

三、编辑一个汇编语言源程序

1、在 DOS 状态下进入 EDIT 编辑状态。

EDIT Filename.Asm

2、在编辑状态下键入用汇编语言编写的源程序：

```
data    segment
buf     db  0ah, 0dh, "Hello worild ! $"
data    ends
code    segment
        assume  cs:code,ds:data
start:  mov     ax, data
        mov     ds,  ax
        lea     dx,  buf
        mov     ah,  9
        int     21h
        mov     ah,  4ch
        int     21h
        code    ends
        end     start
```

3、保存源文件：按 ALT 激活菜单，选择 File 菜单，选择 Save 命令保存源文件。

4、退出 EDIT 状态：选择 File 菜单，选择 Exit 命令，退出 EDIT 状态。

1.2 汇编语言上机步骤

一、必备的软件

EDIT.EXE MASM.EXE LINK.EXE DEBUG.COM

二、上机步骤

1、编辑源程序：用全屏幕编辑程序 EDIT 编辑建立和修改源程序。

命令如下： EDIT Filename. Asm

2、编译源程序：由源代码（ASM）生成目标代码（OBJ）。

命令如下： MASM Filename.asm ；

3、连接程序：由目标代码（OBJ）生成可执行文件（EXE）。

4、执行文件：直接键入可执行文件名。

5、DEBUG 调试 EXE 文件。

三、实习例程

此程序为求 $1+2+3+\dots+100$ 的和。

```
data SEGMENT
sum  DW  0
data ENDS
stack SEGMENT stack
      DB 200 DUP(0)
stack ENDS
code SEGMENT
      ASSUME DS:data,SS:stack,CS:code
start: MOV AX,data
      MOV DS,AX
      MOV CX,100
      MOV AX,0
      MOV BX,1
next:  ADD AX,BX
      ; inc bx
      INC BX
      DEC CX
      JNE next
      MOV sum,AX
      MOV AH,4ch
      INT 21h
Code  ENDS
      END start
```

程序执行过程： 1) >masm qh.asm; (编译)

2) >link qh; (连接)

3) >qh (执行)

4) >Debug qh.exe (调试)

1.3 DEBUG 主要命令

DEBUG 是为汇编语言设计的一种高级调试工具，它通过单步、设置断点等方式为汇编语言程序员提供了非常有效的调试手段。

一、DEBUG 程序的调用

在 DOS 的提示符下，可键入命令：

`C:\DEBUG [D:][PATH][FILENAME[.EXT]][PARM1][PARM2]`

其中，文件名是被调试文件的名称。如用户键入文件，则 DEBUG 将指定的文件装入存储器中，用户可对其进行调试。如果未键入文件名，则用户可以用当前存储器的内容工作，或者用 DEBUG 命令 N 和 L 把需要调试的文件装入存储器后再进行调试。命令中的 D 指定驱动器 PATH 为路径，PARM1 和 PARM2 则为运行被调试文件时所需要的命令参数。

在 DEBUG 程序调入后，将出现提示符“_”，此时就可用 DEBUG 命令来调试程序。

二、DEBUG 的主要命令

1、显示存储单元内容的命令 D(DUMP)，格式为：

`_D[address]或_D[range]`

例如，按指定范围显示存储单元内容的方法为：

`_D 100 120`

18E4:0100 c7 06 04 02 38 01 c7 06-06 02 00 02 c7 06 08 02 G..8.G....G..

18E\$:0110 02 02 bb 04 02 e8 02 00-CD 20 50 51 56 57 8B 37 ..;..h..M PQVW.

18E4:0120 8B

其中 100 至 120 是 DEBUG 预显示的存储单元的地址范围，内容有两种表示：左边用十六进制表示每个字节，右边用 ASCII 字符表示每个字节，“•”表示不可显示的字符。这里没有指定段地址，D 命令自动显示 DS 段的内容。如果只指定首地址，则显示从首地址开始的 80 个字节的内容。如果完全没有指定任何地址，则显示上一个 D 命令显示的最后一个单元后的内容。

2、修改存储单元内容的命令有两种。

- 输入命令 E(ENTER)：

有两种格式如下：第一种格式可以用给定的内容表来替代指定范围的存储单元内容。

`_E address [list]`

例如，`_E DS:100 F3'XYZ'8D`

其中 F3, 'X', 'Y', 'Z'和 8D 各占一个字节，该命令可以用这五个字节来替代存储单元 DS: 0100 到 0104 的原来的内容。

第二种格式则是采用逐个单元相继修改的方法。

`_E address`

例如，`-E DS: 100`

则可能显示为：

18E4: 0100 89.-

如果需要把该单元的内容修改为 78，则用户可以直接键入 78，再按“空格”键可接着显示下一个单元的内容，如下：

18E4: 0100 89.78 1B.-

这样，用户可以不断修改相继单元的内容，直到用 ENTER 键结束该命令为止。

- 填充命令 F(FILL)，其格式为：

`-F range list`

例如: -F 4BA:0100 5 F3'XYZ'8D

使 18E4: 0100~0104 单元包含指定的五个字节的内容。如果 list 中的字节数超过指定的范围, 则忽略超过的项; 如果 list 的字节数小于指定的范围, 则重复使用 list 填入, 直到填满指定的所有单元为止。

3、检查和修改寄存器内容的命令 R(register), 它有三种格式如下:

- 显示 CPU 内所有寄存器内容和标志位状态, 其格式为:

-R

例如, -r

AX=0000 BX=0000 CX=010A DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000

DS=18E4 ES=18E4 SS=18E4 CS=18E4 IP=0100 NV UP DI PL NZ NA PO NC

18E4:0100 C70604023801 MOV WORD PTR [0204],0138 DS:0204=0000

- 显示和修改某个寄存器内容, 其格式为:

-R register name

例如, 键入

-R AX

系统将响应如下:

AX F1F4

:

即 AX 寄存器的当前内容为 F1F4, 如不修改则按 ENTER 键, 否则, 可键入欲修改的内容, 如:

-R bx

BX 0369

: 059F

则把 BX 寄存器的内容修改为 059F。

- 显示和修改标志位状态, 命令格式为:

-RF 系统将响应, 如:

OV DN EI NG ZR AC PE CY-

此时, 如不修改其内容可按 ENTER 键, 否则, 可键入欲修改的内容, 如:

OV DN EI NG ZR AC PE CY-PONZDINV

即可, 可见键入的顺序可以是任意的。

4、运行命令 G, 其格式为:

-G[=address1][address2[address3...]]

其中, 地址 1 指定了运行的起始地址, 如不指定则从当前的 CS: IP 开始执行。后面的地址均为断点地址, 当指令执行到断点时, 就停止执行并显示当前所有寄存器及标志位的内容, 和下一条将要执行的指令。

5、跟踪命令 T(Trace), 有两种格式:

- 逐条指令跟踪

-T [=address]

从指定地址起执行一条指令后停下来, 显示所有寄存器内容及标志位的值。如未指定地址则从当前的 CS: IP 开始执行。

- 多条指令跟踪

-T [=address][value]

从指定地址起执行 n 条指令后停下来, n 由 value 指定。

6、汇编命令 A(Assemble), 其格式为:

-A[address]

该命令允许键入汇编语言语句, 并能把它们汇编成机器代码, 相继地存放在从指定地址开始的存储区中。必须注意: DEBUG 把键入的数字均看成十六进制数, 所以如要键入十进制数, 则其后应加以说明, 如 100D。

7、反汇编命令 U(Unassemble)有两种格式。

- 从指定地址开始, 反汇编 32 个字节, 其格式为:

-U[address]

例如:

-u100

```
18E4:0100 C70604023801    MOV     WORD PTR[0204],0138
18E4:0106 C70606020002    MOV     WORD PTR[0206],0200
18E4:010C C70606020202    MOV     WORD PTR[0208],0202
18E4:0112 BBO402          MOV     BX,0204
18E4:0115 E80200          CALL 011A
18E4:0118 CD20            INT     20
18E4:011A 50              PUSH AX
18E4:011B 51              PUSH CX
18E4:011C 56              PUSH SI
18E4:011D 57              PUSH DI
18E4:011E 8B37 MOV SI,[BX]
```

如果地址被省略, 则从上一个 U 命令的最后一条指令的下一个单元开始显示 32 个字节。

- 对指定范围内的存储单元进行反汇编, 格式为:

-U[range]

例如:

-u100 10c

```
18E4:0100 C70604023801 MOV WORD PTR[0204],0138
18E4:0106 C70606020002 MOV WORD PTR[0206],0200
18E4:010C C70606020202 MOV WORD PTR[0208],0202
```

8、命名命令 N(Name), 其格式为:

-N filespecs [filespecs]

命令把两个文件标识符格式化在 CS: 5CH 和 CS: 6CH 的两个文件控制块中, 以便在其后用 L 或 W 命令把文件装入存盘。filespecs 的格式可以是: [d:][path] filename[.ext]

例如,

-N myprog

-L

-

可把 EXE 文件 myprog 装入存储器。

9、装入命令(Load), 有两种功能。

- 把磁盘上指定扇区范围的内容装入到存储器从指定地址开始的区域中。其格式为:

-L[address[drive sector sector]]

- 装入指定文件, 其格式为:

-L[address]

此命令装入已在 CS: 5CH 中格式化了文件控制块所指定的文件。如未指定地址, 则装

入 CS: 0100 开始的存储区中。

10、写命令 W(Write)，有两种功能。

- 把数据写入磁盘的指定扇区。其格式为：
-W address drive sector sector
- 把数据写入指定的文件中。其格式为：
-W[address]

此命令把指定的存储区中的数据写入由 CS: 5CH 处的文件控制块所指定的文件中。如未指定地址则数据从 CS: 0100 开始。要写入文件的字节数应先放入 BX 和 CX 中。

11、退出 DEBUG 命令 Q(Quit)，其格式为：

-Q

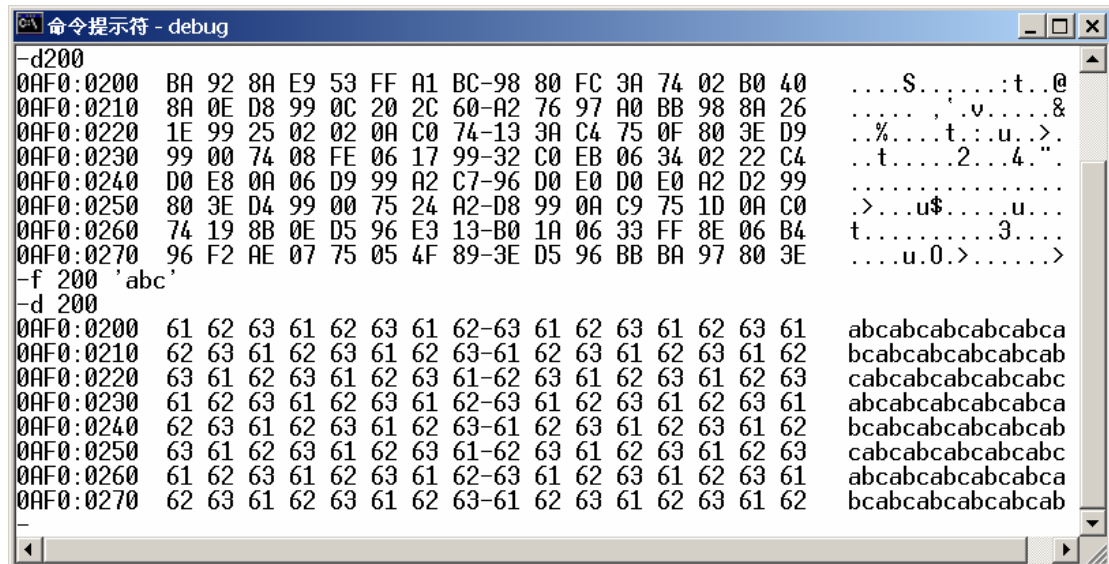
它退出 DEBUG，返回 DOS。本命令并无存盘功能，如需存盘应先使用 W 命令。

12) 内存填充命令 F

格式：F【范围】【单元内容表】

功能：将单元内容表中的内容重复装入内存的指定范围内。

例如：将‘ABC’重复装入从 200H 开始的内存单元中，有：



```
命令提示符 - debug
-d200
0AF0:0200  BA 92 8A E9 53 FF A1 BC-98 80 FC 3A 74 02 B0 40  ....S.....t..@
0AF0:0210  8A 0E D8 99 0C 20 2C 60-A2 76 97 A0 BB 98 8A 26  ....,'.v....&
0AF0:0220  1E 99 25 02 02 0A C0 74-13 3A C4 75 0F 80 3E D9  ..%....t...u...>
0AF0:0230  99 00 74 08 FE 06 17 99-32 C0 EB 06 34 02 22 C4  ..t....2...4..."
0AF0:0240  D0 E8 0A 06 D9 99 A2 C7-96 D0 E0 D0 E0 A2 D2 99  ....
0AF0:0250  80 3E D4 99 00 75 24 A2-D8 99 0A C9 75 1D 0A C0  .>...u$.....u...
0AF0:0260  74 19 8B 0E D5 96 E3 13-B0 1A 06 33 FF 8E 06 B4  t.....3....
0AF0:0270  96 F2 AE 07 75 05 4F 89-3E D5 96 BB BA 97 80 3E  ....u.0.>.....>
-f 200 'abc'
-d 200
0AF0:0200  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcabcabcabca
0AF0:0210  62 63 61 62 63 61 62 63-61 62 63 61 62 63 61 62  bcabcabcabcabcab
0AF0:0220  63 61 62 63 61 62 63 61-62 63 61 62 63 61 62 63  cabcabcabcabcab
0AF0:0230  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcabcabcabca
0AF0:0240  62 63 61 62 63 61 62 63-61 62 63 61 62 63 61 62  bcabcabcabcabcab
0AF0:0250  63 61 62 63 61 62 63 61-62 63 61 62 63 61 62 63  cabcabcabcabcab
0AF0:0260  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcabcabcabca
0AF0:0270  62 63 61 62 63 61 62 63-61 62 63 61 62 63 61 62  bcabcabcabcabcab
```

13) 内存搬家命令 M

格式：M【源地址范围】【目标起始地址】

其中，源地址范围和目标起始地址均为偏移地址，段地址为 DS 的内容。

功能：把源地址范围的内容搬至以目标起始地址开始的存储单元中。

例如：显示 200H—22FH 和 400H—42FH 的单元内容，在用 M 命令将 200H—22FH 单元的内容送到 400H—42FH 存储单元中。

```

C:\masm\BIN>debug
-d 200 22f
0AF0:0200  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcbcabcbcabca
0AF0:0210  62 63 61 62 63 61 62 63-61 62 63 61 62 63 61 62  bcabcbcabcbcabcab
0AF0:0220  63 61 62 63 61 62 63 61-62 63 61 62 63 61 62 63  cabcbcabcbcabcbcab
-d 400 42f
0AF0:0400  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcbcabcbcabca
0AF0:0410  62 8B CE 8B F2 AC E8 7A-E1 74 09 AC 3B F1 72 F5  b.....z.t...;r.
0AF0:0420  59 5E EB 0B 3B F1 72 ED-59 5E 3A 5C FF 74 0E B4  Y^...;r.Y^:\.t..
-m 200 22f 400
-d 400
0AF0:0400  61 62 63 61 62 63 61 62-63 61 62 63 61 62 63 61  abcabcbcabcbcabca
0AF0:0410  62 63 61 62 63 61 62 63-61 62 63 61 62 63 61 62  bcabcbcabcbcabcab
0AF0:0420  63 61 62 63 61 62 63 61-62 63 61 62 63 61 62 63  cabcbcabcbcabcbcab
0AF0:0430  3B CD 21 86 1C 73 95 E8-63 DA E9 91 D7 E9 8B D7  ;.!...s..c.....
0AF0:0440  89 7E 02 80 46 01 0C B8-3F 2E B9 08 00 F3 AA 86  .~..F...?.....
0AF0:0450  C4 AA 86 C4 B1 03 F3 AA-32 C0 AA C3 BE BC 98 BF  .......2.....
0AF0:0460  82 93 B4 60 CD 21 BE 2B-93 BF 82 93 E8 93 E3 C3  ...'.!+.....
0AF0:0470  33 C0 89 3E E6 99 A2 E9-99 A2 EA 99 8A F8 9C 57  3..>.....W

```

14) 比较命令 C

格式：C 【源地址范围】【目的地址】

其中，源地址范围是由起始地址和终止地址支指出的一片连续的存储单元。目的地址为与源地址所指单元相比较的目标地址的起始地址。

功能：从源地址范围起始的地址单元开始，逐个与目标起始地址往后的单元顺序比较每个单元内容，比较到源终止地址为止。比较结果一致，则不显示任何信息，如果不一致，则以

【源地址】【源内容】【目的内容】【目的地址】的形式显示失配单元的地址及内容。

例如：比较 200H—205H 各单元的内容是否与 400H—405H 单元内容是否一致。

```

C:\masm\BIN>debug
-d 200 205
0AF0:0200  BA 92 8A E9 53 FF          ....S.
-d 400 405
0AF0:0400  46 00 01 4E 32 DB          F..N2.
-c 200 205 400
0AF0:0200  BA 46 0AF0:0400
0AF0:0201  92 00 0AF0:0401
0AF0:0202  8A 01 0AF0:0402
0AF0:0203  E9 4E 0AF0:0403
0AF0:0204  53 32 0AF0:0404
0AF0:0205  FF DB 0AF0:0405

```

15) 搜索指定内容的命令 S

格式：S 【地址范围】【表】

功能：在指定地址范围内搜索表中内容，搜索到就显示表中元素所在地址。

例如：搜索地址范围为 100H—150H 中 50H 这个数据所在地址。


```

命令提示符 - debug
0AF0:0205 FF DB 0AF0:0405
-d 100 150
0AF0:0100 35 33 D2 1E 8E 1E B4 96-B4 40 CD 21 1F BA 8C 8A 53.....@.!.!...
0AF0:0110 73 03 E9 4E FF 2B C8 74-D0 F6 06 15 34 00 DF 0A s..N.+..t....4...
0AF0:0120 F6 06 15 99 20 75 0A 80-3E D2 99 00 75 BB 49 74 ....u..>...u.It
0AF0:0130 B8 BA 00 8C EB 23 33 D2-87 D1 B8 01 42 CD 21 A3 .....#3.....B.!.
0AF0:0140 DF 99 89 16 E1 99 80 3E-C5 96 00 74 9C B4 40 CD .....>...t..@.
0AF0:0150 21 !
-s 100 150 50
-s 100 150 00
0AF0:011D
0AF0:012B
0AF0:014A
_

```

16) 写盘命令 W

格式: (1) W[地址][驱动器号][起始扇区号][所写扇区个数]

(2) W[地址]

(3) W

功能: (1) 把在 Debug 状态下调试的程序或数据写入指定的驱动器中, 起始扇区号为逻辑扇区号, 所写扇区数为要占盘中几个扇区。

(2) 当只键入地址参数时, 写盘命令把文件写到当前盘, 当前目录。该文件在用 W 命令之前用 N 命令命名过。

(3) 没有任何参数时, 与 N 命令配合使用进行写盘操作。在用 W 命令之前, 在 BX 和 CX 中应输入文件的字节数。

例如: 在 Debug 状态下汇编一个文件, 并将这个文件存到当前盘上。

```

命令提示符 - debug
-a
0AF0:0100 mov al,32
0AF0:0102 mov dl,37
0AF0:0104 add dl,al
0AF0:0106 sub dl,30
0AF0:0109 mov ah,02
0AF0:010B int 21
0AF0:010D int 20
0AF0:010F
-n ex2
-r cx
CX 0000
:000d
-w
Writing 0000D bytes
-u 100 10f
0AF0:0100 B032 MOV AL,32
0AF0:0102 B237 MOV DL,37
0AF0:0104 00C2 ADD DL,AL
0AF0:0106 80EA30 SUB DL,30
0AF0:0109 B402 MOV AH,02
0AF0:010B CD21 INT 21
0AF0:010D CD20 INT 20
0AF0:010F 8A7303 MOV DH,[BP+DI+03]
_

```

用 W 命令存储文件后, 退出 Debug 查询, 文件 ex2.exe 存在于当前盘中。

1.4 Debug 应用实例

一、例程

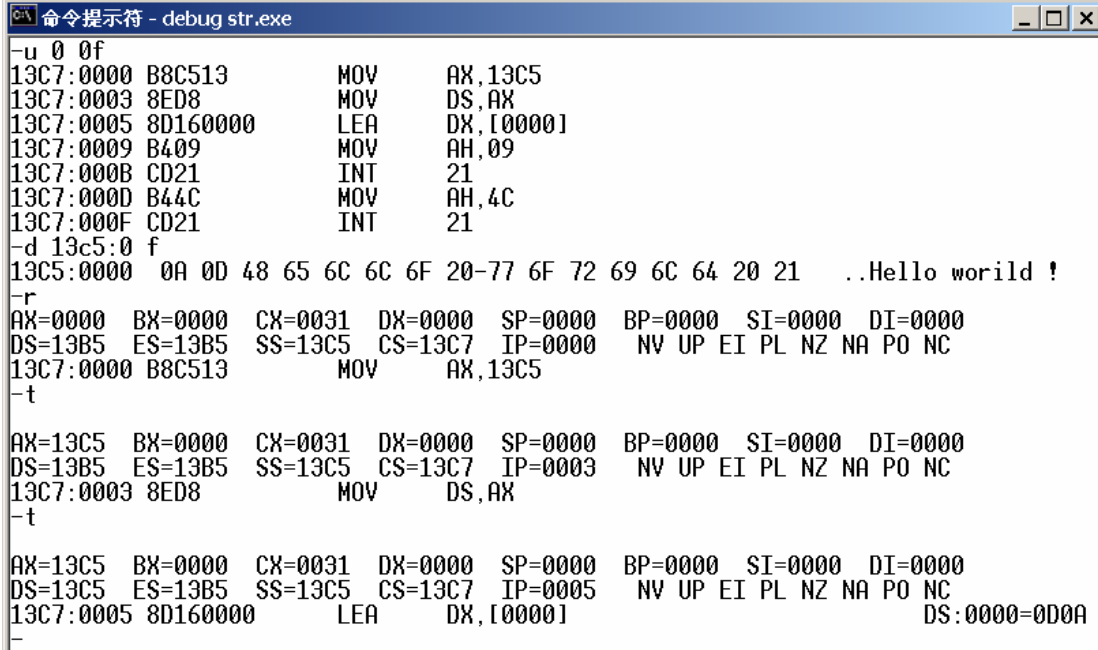
屏幕显示一字符串 “Hello World !”

```
data segment
buf db 0ah, 0dh, "Hello worild ! $"
data ends
code segment
    assume cs:code,ds:data
start: mov ax, data
      mov ds, ax
      lea dx, buf
      mov ah, 9
      int 21h
      mov ah, 4ch
      int 21h
code ends
      end start
```

二、汇编过程

- 1) >masm str.asm; (编译)
- 2) >link str; (连接)
- 3) >str (执行)
- 4) >Debug str.exe (调试)

三、Debug 调试过程



```
命令提示符 - debug str.exe
-u 0 0f
13C7:0000 B8C513      MOV     AX,13C5
13C7:0003 8ED8        MOV     DS,AX
13C7:0005 8D160000     LEA     DX,[0000]
13C7:0009 B409        MOV     AH,09
13C7:000B CD21        INT     21
13C7:000D B44C        MOV     AH,4C
13C7:000F CD21        INT     21
-d 13c5:0 f
13C5:0000 0A 0D 48 65 6C 6C 6F 20-77 6F 72 69 6C 64 20 21  ..Hello worild !
-r
AX=0000 BX=0000 CX=0031 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13B5 ES=13B5 SS=13C5 CS=13C7 IP=0000 NV UP EI PL NZ NA PO NC
13C7:0000 B8C513      MOV     AX,13C5
-t
AX=13C5 BX=0000 CX=0031 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13B5 ES=13B5 SS=13C5 CS=13C7 IP=0003 NV UP EI PL NZ NA PO NC
13C7:0003 8ED8        MOV     DS,AX
-t
AX=13C5 BX=0000 CX=0031 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13C5 ES=13B5 SS=13C5 CS=13C7 IP=0005 NV UP EI PL NZ NA PO NC
13C7:0005 8D160000     LEA     DX,[0000]          DS:0000=0D0A
-
```

```
命令提示符
13C7:0005 8D160000 LEA DX,[0000] DS:0000=0D0
-r bx
BX 0000
:1111
-r
AX=13C5 BX=1111 CX=0031 DX=0000 SP=0000 BP=0000 SI=0000 DI=0000
DS=13C5 ES=13B5 SS=13C5 CS=13C7 IP=0005 NV UP EI PL NZ NA PO NC
13C7:0005 8D160000 LEA DX,[0000] DS:0000=0D0
-e 13c5:2 5 'heppy'
-d 13c5:0 f
13C5:0000 0A 0D 05 68 65 70 70 79-77 6F 72 69 6C 64 20 21 ...heppyworild !
-g
*heppyworild !
Program terminated normally
-q
C:\masm\BIN>
```

```
命令提示符 - debug
-a 100
1371:0100 db '1234567890'
1371:010A cld
1371:010B mov si,100
1371:010E mov di,200
1371:0111 mov cx,a
1371:0114 rep movsb
1371:0116
-g=10a 116

AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=010A DI=020A
DS=1371 ES=1371 SS=1371 CS=1371 IP=0116 NV UP EI PL NZ NA PO NC
1371:0116 43 INC BX
-d100 la
1371:0100 31 32 33 34 35 36 37 38-39 30 1234567890
-d es:200 la
1371:0200 31 32 33 34 35 36 37 38-39 30 1234567890
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=010A DI=020A
DS=1371 ES=1371 SS=1371 CS=1371 IP=0116 NV UP EI PL NZ NA PO NC
1371:0116 43 INC BX
-
```

```
命令提示符 - debug
C:\masm\BIN>debug
-n str.exe
-l
-u
13C7:0000 B8C513 MOV AX,13C5
13C7:0003 8ED8 MOV DS,AX
13C7:0005 8D160000 LEA DX,[0000]
13C7:0009 B409 MOV AH,09
13C7:000B CD21 INT 21
13C7:000D B44C MOV AH,4C
13C7:000F CD21 INT 21
```

第二章 实习内容

实习要求

上机实习的次数暂定为 14 次，可结合实际情况适当增减，但不少于 8 次。

1、实验辅导的主要内容

实验辅导内容包括每个实验的目的；实验内容；实验的算法及实验方法的必要说明；实验准备；实验步骤；实验报告要求；实验程序及参考框图。开始的实验介绍较细，后面的实验简要介绍。

2、实验的软硬件要求

关于汇编语言程序设计的硬件要求不高，有 IBM-PC/XT 即可，但应有彩色显示器以便进行图形实验。软件方面应有 MASM.EXE V5.0 版(包括 LINK.EXE)，与 MS-DOS 版本配套的 DEBUG 程序和 EDIT.EXE 编辑软件（其它编辑软件也可以）。

3、加强实践能力的培养

实验目的不光是为了验证书本理论，更重要的是实践能力的培养。其中包括：

实际调试程序的能力，例如修改程序参数的能力，查看结果的能力，设置断点调试运行的能力等；

开发汇编语言应用程序的能力，例如应用有关汇编软件的能力，进行系统调用和 BIOS 功能调用的能力，进行模块程序设计的能力等；

对某一问题用不同的程序实现的能力，例如我们为每个实验提供了参考程序（或程序段），目的是让每个实验者参照样本程序将实验成功地实现，在掌握其方法后，自己改变程序或部分改变程序加以实现。

实验 1 寻址方式与指令练习

[实验目的]

通过实验掌握各种寻址方式，理解指令的功能，熟悉 DEBUG 的调试命令。

[实验内容]

(1) 假设 (DS) = 2000H, (ES) = 2100H, (SS) = 1500H, (SI) = 00A0H, (BX) = 0100H, (BP) = 0010H, 请通过实验指出下列指令的源操作数是什么寻址方式？它们的物理地址是多少？

Mov ax, 0ab

Mov ax, bx

Mov ax, [100]

Mov ax, [bx]

Mov ax, es:[bx]

Mov ax, [bp]

Mov ax, [si+10]

Mov ax, [bx][si]

Mov ax, 10[bx][si]

(2) 自选已学过的指令，在 Debug 下用 A 命令写入，如：add [bx], ax 等，用 T 命令跟踪，观察执行结果以及结果的存放位置。

[实验要求]

本实验只要求在 Debug 状态下进行。

[实验步骤提示]

- 1、首先进入 Debug 状态，用 R 命令修改各寄存器的内容，以满足题目的要求。
- 2、用汇编命令 A 写入要实验的 9 条语句。
- 3、用 T 命令单步跟踪执行，观察运行的结果，进一步理解各种寻址方式。
- 4、用 T 命令后，再用 D 命令观察内存单元的内容和运行结果。

[实验报告]

- 1、写出上机调试步骤，调试过程中所遇到的问题是怎样解决的。对调试过程中的问题进行分析，对执行结果进行分析。
- 2、写出实验体会及对寻址方式的理解。

实验 2 初级程序的编写与调试

[实验目的]

- 1、熟练掌握 Debug 的常用命令，学会用 Debug 调试程序。
- 2、深入理解数据在存储器中的存取方法，以及堆栈中数据的压入与弹出。

[实验内容]

- 1、设堆栈指针 SP=2000H，AX=3000H，BX=5000H，请编一程序段将 AX 的内容和 BX 的内容进行互换。用堆栈作为两寄存器内容的中间存储单元，用 Debug 调试程序进行汇编与调试。
- 2、设 DS=当前段地址，BX=0300H，SI=0002H，请用 Debug 命令将存储器偏移地址 300H 到 304H 连续 5 个存储单元顺序装入 0AH, 0BH, 0CH, 0DH, 0EH。在 Debug 状态下键入下面程序段，并用单步跟踪方法，分析每条指令源操作数地址的形成过程？当数据传送完毕时，AX 中的内容是什么？

[程序清单如下]

```
mov ax, bx
mov ax, 304
mov ax, [304]
mov ax, [bx]
mov ax, 01[bx]
mov ax, [bx][si]
mov ax, 01[bx][si]
hlt
```

[实验要求]

实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法、对程序结果的分析。本实验只要求在 Debug 状态下进行，包括汇编程序、调试程序、执行程序。

[实验内容 1]

将两个寄存器的内容进行交换时，必须有一个中间寄存器才能进行内容的交换。如果用堆栈作为中间存储单元，必须遵循后进先出的原则。注意观察堆栈指针 SP 的变化。

[实验内容 2]

其中数据段寄存器中的段地址为进入 Debug 状态后系统自动分配的段地址，SI 和 BX 的初值可在 Debug 状态下，用 R 命令装入，也可以在程序中用指令来完成。

用 T 命令执行程序，可进行单步跟踪执行，每执行一条指令就可以看到寄存器的状

态，也可用 R 命令直接调出寄存器，来检查各寄存器的内容是否正确。

在执行程序前，可用 E 命令将偏移地址 300H—304H 中送入 0AH, 0BH, 0CH, 0DH, 0EH。

[实验报告]

- 1、说明上机调试的情况：上机调试步骤、调试过程中所遇到的问题是怎样解决的、对调试过程中的问题进行分析、对执行结果进行分析。
- 2、写出程序框图。
- 3、写出程序清单和执行过程。

实验 3 加法程序的编写与调试

[实验目的] 了解指令对标志位的影响，掌握多字节的运算方法。

[实验内容]

对两个 32 位数做加法运算。这两个数从地址 10050H 开始存放，结果放在其后。假设这两个数分别为 12345678H 和 2468ACE0H。

[实验要求]

- 1、实验前要做好充分准备，包括汇编程序清单、调试步骤、调试方法、对程序结果的分析。实验过程注意观察标志位的变化。
- 2、本实验只要求在 Debug 状态下进行。

[实验提示]

- 1、在 Debug 状态下，用 E 命令把两个数写入内存。注意低位在低字节。
- 2、编程时注意：高位相加时，要加上低位相加所产生的进位。
- 3、用 D 命令观察程序的运行结果。

[实验报告]

- 1) 说明上机调试的情况：上机调试步骤、调试过程中所遇到的问题是怎样解决的、对调试过程中的问题进行分析、对执行结果进行分析。
- 2) 写出程序框图。
- 3) 写出程序清单和执行过程。

[参考程序段]

假设在内存 BUFFER1, BUFFER2 开始的两个存区中分别存放有两个四字节数，现求这两个数之和，并把结果存入以 BUFFER3 为首址的存区中。

```
Mov ax,buffer1
Add ax,buffer2
Mov buffer3,ax
Mov ax,buffer1+2
Acd ax,buffer2+2
Mov buffer3+2,ax
```

实验 4 数据传送练习

[实验目的] 了解 Debug 中 M 命令的使用方法，掌握数据成片传送的程序编写。

[实验内容]

用汇编语言编写一个字符串传送程序：数据区 DATA1 有一组数据，如 30h, 31h, 32h, 33h, 34h, 35h, 36h, 37h, 38h, 39h。编程把这组数据传送至 DATA2 中。

[实验要求]

- 1、分析题目，将程序中的原始数据和最终结果的存放方式确定好。
- 2、写出用 M 命令的执行过程。
- 3、写出算法和流程图。
- 4、对程序结果进行分析，并准备好上机调试过程。
- 5、编写完整的汇编语言程序，要求在 EDIT 下写程序，通过汇编（MASM）、连接（LINK）生成执行文件，并进行跟踪调试（Debug）。

[编程提示]

- 1、要有一个完整的程序框架，参考程序如下：

```
data segment
data1 db 30h, 31h, 32h, 33h, 34h, 35h, 36h, 37h, 38h, 39h
data2 db 10dup(?)
data ends
code segment
    assume ds:data, cs:code
start: mov ax, data
        mov ds, ax
        mov es, ax
        cld
        lea si, data1
        lea di, data2
        mov cx, 10
        rep movsb
        mov ah, 4ch
        int 21h
code ends
end start
```

- 2、以文件名 LX1.ASM 存盘。
- 3、对源程序进行汇编，如 MASM LX1.ASM；生成了目标文件 LX1.OBJ。
- 4、连接生成可执行文件，如 LINK LX1；生成了执行文件 LX1.EXE。
- 5、用 Debug LX1.EXE 跟踪运行，观察结果。

[M 命令使用提示]

- 1、在使用 M 命令之前，用 E 命令在存储单元中写入数据。
- 2、注意源数据到目标数据的移动地址。
- 3、用 D 命令随时观察数据的移动过程。

[实验报告]

- 1、说明上机调试的情况：上机调试步骤、调试过程中所遇到的问题是怎样解决的、对调试过程中的问题进行分析、对执行结果进行分析。
- 2、写出程序框图。
- 3、写出程序清单和执行过程。
- 4、写出实验体会以及对汇编语言上机过程的理解。

实验 5 利用 DEBUG 调试程序段

1. 实验目的

- (1) 熟悉 DEBUG 有关命令的使用方法。
- (2) 利用 DEBUG 掌握有关指令的功能。
- (3) 利用 DEBUG 运行简单的程序段。

2. 实验内容

- (1) 进入和退出 DEBUG 程序。
- (2) 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。
- (3) 利用 DEBUG，验证乘法、除法、加法、减法、带进位加、带借位减、堆栈操作指令、串操作指令的功能。
- (4) 计算表达式 $(V - (X * Y + Z - 540)) / X$ 的值。

3. 实验准备

- (1) 仔细阅读有关 DEBUG 命令的内容，对有关命令，都要事先准备好使用的例子。
- (2) 作为例子，准备用 A 命令，输入在显示器上显示字符“S”的系统调用程序段。
- (3) 阅读将 AX 左移的程序段。

4. 实验步骤

- (1) 在 DOS 提示符下，进入 DEBUG 程序。在 DOS 子目录下启动 DEBUG。
- (2) 详细记录每一步所用的命令，以及查看结果的方法和具体结果。
- (3) 表达式计算结果存放在那里？

5. 实验报告要求

- (1) 如何启动和退出 DEBUG 程序。
- (2) 整理每个 DEBUG 命令使用的方法，实际示例及执行结果。
- (3) 启动 DEBUG 后，要装入某个 .EXE 文件，应通过什么命令去实现？

参考程序清单

```
data segment
x dw 10 ;?
y dw 100 ;?
z dw 1000;?
v dw 2000;?
data ends
code segment
    assume cs:code, ds:data
start:mov ax, data
    mov ds, ax
    mov ax, x
    imul y
    mov cx, ax
    mov bx, dx
    mov ax, z
    cwd
    add cx, ax
```



```

        adc bx, dx
        sub cx, 540
        sbb bx, 0
        mov ax, v
        cwd
        sub ax, cx
        sbb dx, bx
        idiv x
        mov ah, 4ch
        int 21h
code    ends
        end start

```

实验 6 分支程序实验

1. 实验目的

- (1) 掌握分支程序的设计方法。
- (2) 掌握利用 DEBUG 修改参数. 检查结果的方法。
- (3) 掌握汇编语言源程序的编辑. 汇编. 连接及调试过程。

2. 实验内容

- (1) 编一程序, 显示 AL 寄存器中的两位十六进制数。
- (2) 编一程序, 判别键盘上输入的字符; 若是 0—9 字符, 则显示之; 若为 A—Z 或 a—z 字符, 均显示 “c”; 若是回车字符<CR>(其中 ASCII 码为 0DH), 则结束程序, 若为其它字符则不显示, 继续等待新的字符输入。

3. 实验准备

- (1) 编写实验内容要求的两个程序。
- (2) 写出调试以上程序, 即修改程序的参数, 检查结果的操作方法。
- (3) 熟悉源程序汇编. 连接命令的使用方法及要回答的内容。

4. 实验步骤

- (1) 用 EDIT 或其它编辑软件, 编写. ASM 源程序, 例如 HEXASC. ASM 及 DSPKEY. ASM。
- (2) 对其进行汇编及连接, 产生. EXE 文件。
- (3) 对. EXE 文件进行调试运行。
 - ①用 DEBUG 调试运行, 学会修改 AL 内容的方法。
 - ②对 DSPKEY. EXE 键入不同的字符, 分别进行调试。
 - ③在 MS-DOS 下运行这两个. EXE 文件。

5. 实验报告要求

- (1) 画出两个程序的流程图, 若要独立编写程序, 应列出程序清单。
- (2) 说明本实验是如何利用 DEBUG 进行调试的。

6. 参考程序清单

```

(1) 显示 AL 中十六进制数程序;
        ; DISPHEX. ASM
CODE    SEGMENT
        ASSUME CS:CODE
START:  MOV AL, 3EH

```

```

Mov    bl, al
MOV    DL , AL
MOV    CL , 4
SHR    DL, CL
CMP    DL , 9
JBE    NEXT1
ADD    DL , 7
NEXT1: ADD    DL , 30H
MOV    AH, 2
INT    21H                ; 显示高位 ASCII 码
MOV    DL, bL
AND    DL , 0FH
CMP    DL, 9
JBE    NEXT2
ADD    DL, 7
NEXT2: ADD    DL, 30H
MOV    AH, 2
INT    21H                ; 显示低位 ASCII 码
MOV    AH, 4CH
INT    21H                ; 返回 DOS
CODE   ENDS
        END START

```

(2) 显示键入字符程序:

; DISPKEY. ASM

```

CODE    SEGMENT
        ASSUME CS:CODE
START:   MOV    AH, 1
          INT    21H                ; 等待键入字符, 送 AL
          CMP    AL, 0DH            ; 是否是回车符?
          JZ     DONE              ; 是则转 DONE 退出程序
          CMP    AL, '0'
          JB     NEXT
          CMP    AL, '9'
          JA     CHRUP
          MOV    DL, AL
          MOV    AH, 2
          INT    21H                ; 显示 0 ~ 9
          JMP    START
CHRUP:   CMP    AL, 41H
          JB     NEXT
          CMP    AL, 5AH
          JA     CHRDN
DISPC:   MOV    DL, ' C'
          MOV    AH, 2

```

```

                INT 21H
NEXT:           JMP START
CHRDN:          CMP AL, 61H
                JB NEXT
                CMP AL, 7AH
                JA NEXT
                JMP DISPC
DONE:           MOV AH, 4CH
                INT 21H                ; 返回 DOS
CODE            ENDS
                END START

```

实验 7 循环程序实验

1. 实验目的

- (1) 掌握循环程序的设计方法。
- (2) 进一步熟悉用 DEBUG 工具修改程序参数的方法，并检查和验证结果的正确性。
- (3) 学会针对不同的问题，选用不同的组织循环的方法。

2. 实验内容

- (1) 将字符串 STRN (以 “\$” 结尾) 中的每一个字符均加上偶校验位，并统计有多少个字符因含有奇数个 “1” 而加上了校验位。统计结果存于 N 单元中。
- (2) 存储 DAT1 及 DAT2 中各有 10 个字节的二进制数，高位字节放在高位地址中，试编写一程序将这两个数据相加，结果存放在从 DAT3 开始的单元中。

3. 实验准备

- (1) 预习循环程序的两种基本结构及应用场合，学会正确地组织循环。
- (2) 结合参考程序，画出程序流程图。
- (3) 思考用不同的程序去实现同一个功能。

4. 实验步骤

- (1) 编写、汇编和连接源程序，产生可执行文件 .EXE。
- (2) 程序调试成功后，修改有关参数进行调试运行并验证结果的正确性。
- (3) 对多字节二进制数加法，用另一种程序加以实现。

5. 实验报告要求

- (1) 列出程序清单，画出程序流程图。如果是自己编写的程序，则要首先列出。
- (2) 总结循环程序的结构和组织循环的方法。
- (3) 总结一下两个多字节二进制数的加法程序可用几种方法实现，并说明如果实现这两个数的减法、两个 BCD 数的加法及减法，程序应作哪些修改？

6. 参考程序清单

(1) 偶校验程序清单

```

                ; EX56. ASM
DSEG           SEGMENT
STRN            DB ' ABCDEfghi jkLMNOPQuvw', '$'
N               DB?
DSEG            ENDS
CSEG            SEGMENT

```

```

                ASSUME CS: CSEG, DS: DSEG
START:  MOV AX, DSEG
        MOV DS, AX
        LEA SI, STRN
        MOV DL, 0
AGAIN:  MOV AL, [SI]
        CMP AL, ' $ '
        JE DONE
        TEST AL, 0FFH
        JPE NEXT
        OR AL, 80H
        MOV [SI], AL
        INC DL
NEXT:   INC SI
        JMP AGAIN
DONE:   MOV N, DL
        MOV AH, 4CH
        INT 21H
CSEG    ENDS
        END START

```

以上是条件控制的循环。因为事先不知道字符串 STRN 共有多少个字符，故采用条件控制的循环程序实现。

(2) 加法程序清单：

```

                ; ADDHEX. ASM
CODE    SEGMENT
                ASSUME CS:CODE, DS:CODE
START:  MOV AX, CS
        MOV DS, AX
        LEA SI, DAT1
        LEA DI, DAT2
        LEA BX, DAT3
        MOV CX, 10
        XOR AL, AL                ; 0→CF
AGAIN:  MOV AL, [SI]
        ADC AL, [DI]
        MOV [BX], AL
        INC SI
        INC DI
        INC BX
        LOOP AGAIN
        MOV AH, 4CH
        INT 21H
DAT1    DB 70H, 80H, 90H, A0H, A1H, A2H, A3H
        DB 74H, 65H, 56H

```

```

DAT2      DB 45H, 67H, 89H, 1AH, 2BH, 3CH
           DB 4DH, 5EH, 6FH, 04H
DAT3      DB 10 DUP(?)
CODE      ENDS
           END START

```

本题中，DS 及 CS 同处一个逻辑段 CODE 中，所以，应将 CS 值送至 DS 中。

实验 8 子程序实验（一）

本实验的目的在于让读者掌握同一模块内的子程序调用的方法。

1. 实验目的

- (1) 掌握主程序与子程序之间的调用关系及其调用方法。
- (2) 掌握子程序调用过程中近程调用与远程调用的区别。
- (3) 掌握通过堆栈传送参数的方法。

2. 实验内容

(1) 将 BUF 开始的 10 个单元中的二进制数转换成两位十六进制数的 ASCII 码，并在屏幕上显示出来。要求码型转换通过子程序 HEXASC 实现，在转换过程中，通过子程序 DISP 实现显示。

(2) 编写一个主程序，从键盘接收若干个字符，然后用远调用的方法，调用子程序统计该字符串中字符“b”的个数。子程序的参数是字符串的首地址 TABLE、字符长度 N 及字符“b”。子程序返回字符“b”的个数。参数传送采用堆栈实现。主程序在子程序返回后，显示字符“b”及其个数（设为一位十六进制数）。

3. 实验说明

(1) 第一个实验程序用子程序的近程调用实现。由于在调用 HEXASC 子程序时，子程序又调用 DISP 子程序，这叫做子程序的嵌套调用。实验过程中可以从堆栈的内容看到两个子程序的返回地址值。由于是近调用，地址值只包括返回地址的段内偏移量。在每个子程序的执行中，检查 CS 值是不变的。

(2) 第二个程序是利用远调用的方法调用子程序的。在远调用情况下，主程序与子程序处在不同的逻辑代码段中，可在子程序执行中查看 CS 值，它与主程序中的 CS 值是不同的。子程序调用后，堆栈中保留了返回地址的段地址及段内偏移量。

(3) 第二个程序中，主程序与子程序之间参数的传送是由堆栈实现的。一段是将参数（此处是串首址 TABLE，串的长度 N 及待统计的字符“b”）顺序压入堆栈，在子程序调用后，通过 BP 指针对堆栈中的参数访问，并将统计的结果通过堆栈返回。有关该方法的原理此处不再介绍。

4. 实验准备

- (1) 预习子程序设计的基本方法，根据实验内容要求，画出主程序及子程序的流程图。
- (2) 熟悉键盘输入字符串及用堆栈传送参数的程序段编制方法。

5. 实验步骤

- (1) 编辑、汇编两个源程序，生成相应的可执行文件（.EXE）。
- (2) 用 DEBUG 的 R 命令、T 命令或 G 命令和 D 命令检查远程调用及近程调用时堆栈的变化。特别是通过堆栈传送的参数和子程序取出的参数是返回参数的详细过程。
- (3) 检查程序执行的结果是否正确。

6. 实验报告要求

- (1) 分析远程调用与近程调用的区别，在用 DEBUG 有关命令观察时，执行过程有何不同。

- (2) 说明用堆栈传送参数的过程及具体的方法。
 (3) 分析实验结果及所遇到的问题，并说明解决的方法。

7. 参考程序清单

(1) 码型转换程序清单:

```

                                ; CONV.ASM
DATA        SEGMENT
BUF          DB 0ABH, 0CDH, 0DEH, 01H, 02H, 03H
             DB 3AH, 4BH, 5CH, 6FH
DATA        ENDS
CODE        SEGMENT
             ASSUME CS: CODE , DS : DATA
START:      MOV AX , DATA
             MOV DS , AX
             MOV CX , 10
             LEA BX , BUF
AGAIN:      MOV AL , [BX]
             CALL HEXASC
             INC BX
             LOOP AGAIN
             MOV AH , 4CH
             INT 21H
HEXASC      PROC NEAR
             MOV DL , AL
             PUSH CX
             MOV CL , 4
             SHR DL , CL
             POP CX
             CALL DISP                ; 显示高位 HEX 数
             MOV DL , AL
             AND DL , 0FH
             CALL DISP                ; 显示低位 HEX 数
             RET
HEXASC      ENDP
DISP        PROC
             CMP DL , 9
             JBE NEXT
             ADD DL , 7
NEXT:      ADD DL, 30H                ; 将一位 HEX 数转为 ASCII 码
             MOV AH, 2
             INT 21H                ; 显示
             RET
DISP        ENDP
CODE        ENDS
             END START

```

(2) 统计并显示某键入字符的个数的程序：

```

                ; COUNTER . ASM
DATA            SEGMENT
CHAR            DB ' b'
BUF             DB 50H, ? 50H DUP (?)
DATA            ENDS
MCODE           SEGMENT
                ASSUME CS: MCODE, DS: DATA
START:          MOV AX, DATA
                MOV DS, AX
                LEA DX, BUF
                MOV AH, 9
                INT 21H
                LEA SI, BUF
                MOV CL, [SI+1]
                MOV CH, 0                ; CX 中为字符串的长度
                INC SI
                INC SI                ; SI 指向串首址 TABLE
                MOV AL, CHAR
                MOV AH, 0                ; AX 中为待查字符
                PUSH SI
                PUSH CX
                PUSH AX                ; 参数送堆栈
                CALL CHECK
                POP AX                ; 统计个数在 AX 中
                MOV DL, CHAR
                MOV AH, 2
                INT 21H                ; 显示待检字符
                MOV DL, AL
                AND DL, 0FH
                CMP DL, 9
                JBE NEXT
                ADD DL, 7
NEXT:           ADD DL, 30H
                MOV AH, 2
                INT 21H                ; 显示统计个数
                MOV AH, 4CH
                INT 21H
MCODE           ENDS
SCODE           SEGMENT
                ASSUME CS: SCODE
CHECK           PROC FAR
                PUSH BP
                MOV BP, SP

```

```

        MOV SI, [BP+10]
        MOV CX, [BP+8]
        MOV AX, [BP+6]
        XOR AH, AH
AGAIN:   CMP AL, [SI]
        JNE NEXT1
        INC AH
NEXT1:   INC SI
        LOOP AGAIN
        MOV AL, AH
        MOV [BP+10], AX
        POP BP
        RET 4
CHECK   ENDP
        END START

```

实验 9 子程序实验（二）

本实验目的在于使读者掌握模块间调用子程序的编写方法。

1、实验目的

- (1) 了解多模块程序设计方法。
- (2) 学会使用 PUBLIC 和 EXTRN 伪指令解决模块间的符号（如变量名、标号等）通信问题。

2、实验内容

- (1) 编写一个子程序，将主程序设定的内存中字符串的小写字母转换成大写字母并显示出来。主程序用另一个模块编写。
- (2) 编写一个子程序，将主程序指定的字符所在的地址值返回给主程序，字符串与主程序在同一个模块。

3、实验准备

- (1) 仔细阅读教材中有关模块间通信的方法及模块程序设计的方法。
- (2) 弄清伪指令 PUBLIC 和 EXTRN 的功能及用法。

4、实验步骤

- (1) 分别对实验 1 和 2 的主、子模块进行汇编，在连接时，将它们装配成一个以 .EXE 为扩展名的可执行文件。观察汇编及连接过程中有无错误。
- (2) 对 .EXE 文件进行调试及运行。
- (3) 将实验 1 中的源字符串改为由键盘输入，然后由子模块将源串中的小写字母转换成大写字母，并将源串与转换后的两个字符串分两行显示出来（设键入字符串长度小于 80 个字符）。

5、实验报告要求

- (1) 对源程序清单，画出相应的程序流程图。
- (2) 扼要总结多模块程序的特点和编写方法。

6、参考程序清单

- (1) 转换字符串小写字母为大写字母的程序清单：
； EXCHAR.ASM 为主模块的文件名


```

EXTRN DNTOUP: FAR
PUBLIC  STRING1, STRING2
DATA    SEGMENT
STRING1 DB 'This is a Book', '$'
STRING2 DB 80 DUP (?)
DATA    ENDS
CODE    SEGMENT
        ASSUME CS: CODE, DS: DATA
START:  MOV AX, DATA
        MOV DS, AX
        CALL DNTOUP
        MOV AH, 4CH
        ICN 21H
CODE    ENDS
        END START
; DNTOUP.ASM 为子模块的文件名
PUBLIC  DNTOUP
EXTRN  STRING1: BYTE, STRING2: BYTE
CODE    SEGMENT
DNTOUP  PROC FAR
        MOV BX, 0
CYCLE:  MOV AL, STRING1[BX]
        CMP AL, 24H          ; 是否是 STRING1 结尾
        JZ  DONE             ; 是 "$", 转 DONE
        CMP AL, 61H          ; 是小写字母吗?
        JB  NEXT             ; 不是转 NEXT
        CMP AL, 7AH
        JA  NEXT
        SUB AL, 20H          ; 转为大写字母
NEXT    MOV STRING2[BX], AL
        INC BX
        JMP CYCLE
DONE:   MOV AL, '$'
        MOV STRING2[BX], AL; 补一个$
        MOV DL, 0DH
        MOV AH, 2
        INT 21H
        MOV DL, 0AH
        MOV AH, 2
        INT 21H             ; 显示回车, 换行
        LEA DX, STRING1
        MOV AH, 9
        INT 21H             ; 显示源串内容
        MOV DL, 0DH

```

```

        MOV AH, 2
        INT 21H
        MOV DL, 0AH
        MOV AH, 2
        INT 21H           ; 回车换行
        MOV DX, OFFSET STRING2
        MOV AH, 9
        INT 21H           ; 显示转换后的大写串
        RET
DNTOUNP ENDP
CODE     ENDS
        END

```

(2) 查找指定字符，并返回地址值的程序清单：

； MAINP.ASM 为主模块程序名

```

EXTRN    FINDC: FAR
PUBLIC   STRN
DATA     SEGMENT
STRN     DB' LINK DISPLAY SUBROUTINE $'
ADDR     DW?
DATA     ENDS
CODE     SEGMENT
        ASSUME CS: CODE, DS: DATA
STARTR:  MOV AX, DATA
        MOV DS, AX
        MOV AH, 1
        INT 21H
        CALL FINDC
        MOV ADDR, DI
        MOV AH, 4CH
        INT 21H
CODE     ENDS
        END START
;FINDC.ASM 为子模块程序名
PUBLIC   FINDC
EXTRN    STRN: BYTE
CODE     SEGMENT
        LEA DI, STRN
AGAIN:   CMP BYTE PTR[DI], '$'
        JZ  DONE
        CMP AL, [DI]
DONE1:   RET
NEXT:    INC DI
        JMP AGAIN
DONE:    MOV DI, 0FFFFH   ; 找不到返回 0FFFFH

```

```

                JMP DONE1
FINDC          ENDP
CODE           ENDS
                END

```

(3) 若由键盘输入字符串，则在程序 1 的主模块的 DATA 段中加一行提示信息 IMAGE：

```
IMAGE DB 'INPUT A STRING PLEASE: $'
```

然后在主模块的 CALL 指令前插入以下程序段：

```

                LEA DX, IMAGE
                MOV AH, 9
                INT 21H
                LEA DX, STRING2+1
                MOV AH, 0AH
                INT 21H
                LEA SI, STRING2+2
                MOV CL, STRING2+1
                MOV CH, 0
                LEA DI, STRING1
AGAIN:  MOV AL, [SI]
                MOV [DI], AL
                INC SI
                INC DI
                LOOP AGAIN
                MOV BYTE PTR[DI], '$' : 补一个$

```

说明：此处先将键入字符存入 STRING2，然后再将其字符串部分传送到 STRING1 中，当然在定义 STRING2 时，应为：

```
STRING2 DB 80, ? 80 DUP (?)。
```

实验 10 字符处理程序实验

1. 实验目的

- (1) 熟悉串操作指令的功能与应用。
- (2) 掌握串操作指令的寻址方式及使用方法，编写常用的字符串处理程序。

2. 实验内容

(1) 字符串统计。自 STRN 开始的存储区中，有一个字符串，统计其中含有小写字母的个数，将统计结果以两位十进制数形式显示在屏幕上。

(2) 在给定的字符串中，删除重复的字符，其余的字符向前递补。

3. 实验准备

- (1) 熟悉字符处理的方法和字符处理程序的设计。
- (2) 认真预习有关串操作的指令及其寻址方式的特点，能够正确使用串操作指令，并准备好数据。

(3) 按正常的方法将删除字符程序编成子程序，规定子程序的入口和出口参数。

4. 实验步骤

(1) 用 1 号系统调用从键盘输入一个字符串，然后统计其中小写字母的个数。程序每次执行，都能得到不同的结果。

(2) 实验 2 可参考教材例题练习的解法，但要编写一个在同一个字符串中删除字符，并将其余字符向前递补的程序。

5. 实验报告要求

(1) 对照参考程序，画出程序流程图。

(2) 总结字符串处理程序的编程方法，提出改进和完善此类程序设计方案。

6. 参考程序清单

(1) 统计小写字母个数的程序清单：

```
; COUNTDC.ASM
DATA SEGMENT
    STRN DB 80DUP(?)
DATA ENDS
CODE    SEGMENT
    ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV DS, AX
        LEA DI, STRN
        MOV CL, 0
AGAIN:  MOV AH, 1
        INT 21H
        CMP AL, 0DH
        JZ  DONE
        MOV [DI], AL
        INC DI
        INC CL
        JMP AGAIN
DONE:   LEA SI, STRN
        MOV CH, 0
        MOV BL, 0
        CLD
CYCLE:  LODSB
        CMP AL, 61H
        JB NEXT
        CMP AL, 7AH
        JA NEXT
        INC BL
NEXT:   LOOP CYCLE
        MOV AL, BL
        MOV AH, 0
        MOV CL, 10
        DIV CL
        XCHG AH, AL
        MOV DL, 0AH
        OR DL, 30H
        MOV AH, 2
```

```

        INT 21H
        MOV DL, AL
        OR DL, 30H
        INT 21H
        MOV AH, 4CH
        INT 21H
CODE    ENDS
        END START

```

(2) 删除字符串中重复字符的源程序清单:

```

; DELD.ASM
DATA    SEGMENT
STRN    DB    80    DUP(?)
LEN      DB    ?
DATA    ENDS
CODE    SEGMENT
        ASSUME CS:CODE, DS:DATA, ES:DATA
START:  MOV AX , DATA
        MOV DS , AX
        MOV ES , AX
        LEA SI , STRN
        MOV CL , 0
AGAIN:  MOV AH , 1
        INT 21H
        CMP AL , 0DH
        JZ  DONE
        MOV [SI] , AL
        INC SI
        INC CL
        JMP AGAIN
DONE:   MOV CH , 0
        MOV LEN , [CL]
REPEAT: PUSH SI
        PUSH CX
        PUSH CX
        POP DX
        DEC CX
        JE NEXT1
        MOV AL , [SI]
        CMP AL , 0
        JE NEXT1
COMP:   INC SI
        CMP AL , [SI]
        JNE GOON
        MOV BYTE PTR[SI] , 0

```

; CX 为内循环计数
; DX 为外循环计数
; 是最后一个字符, 不再找重复字符
; 若为空字符, 则跳过
; 为非重复字符

```

GOON:  LOOP COMP                ; 查处与该字符重复的所有字符，并用 0 替换
NEXT1:  POP DX
        POP SI
        DEC DX
        MOV CX , DX
        JNZ REPEAT              ; 对所有重复字符都进行查找
        LEA SI, STRN            ; 删除重复字符
        MOV CL, LEN
        MOV CH, 0
CYCLE:  MOV AL, [SI]
        CMP AL, 0                ; 是重复字符吗？
        JNZ NEXT                ; 不是，查找下一个字符
        DEC CX                    ; 递补子串长度送 CX
        JZ DONE                  ; 若是最后字符不递补
        CALL DELE                ; 删除该空字符并递补
        DEC SI                    ; 对递补子串从头处理
NEXT:   INC SI
        LOOP CYCLE
        MOV AH, 4CH
        INT 21H
DELE:   PROC
        PUSH SI
        PUSH CX
        CLD
        MOV DI, SI                ; 空字符处为目的首址
        INC SI                    ; 下一个地址为源串首址
        REP MOVSB
        POP CX
        POP SI
        RET
DELE    ENDP
CODE    ENDS
        END START

```

实验 11 输入输出实验

1. 实验目的

- (1) 掌握输入输出程序设计的概念和方法。
- (2) 了解 PC 机外围芯片 8255、8259 的功能。
- (3) 学习如何在 PC 机上编写具有输入输出功能的程序，包括对 8255、8259 芯片的使用方法。

2. 实验说明

本实验要求自行编写一个键盘输入处理程序，它可以完成键盘字符的读入并进行屏幕显示。本实验要利用 IBM-PC 系统的硬件结构，分别使用外围芯片 8255 及 8259。

在本例中，利用 8255A 的 A 端作数据输入，对应的端口地址为 60H；利用 B 端口作控制输出，端口地址为 61H。8255A 的控制端口地址为 63H。

本例中的 8259 中断控制器，其 IRQ1 端用于键盘中断请求线。键盘通过它可以向 CPU 发出中断请求。8259 的 I/O 端口地址为 21H，可以写入中断屏蔽字，以对 8 个中断源是否允许中断进行控制。在每次中断结束时，要通过 I/O 端口地址 20H 写回一个中断结束命令 EOI，使 8259 可以清除此次中断。

因此本实验既属于输入输出实验，也属于中断实验。

有关说明将详细地附在参考程序中。

3. 实验内容

利用 PC 机键盘，编写一个读入并显示键盘输入的演示程序。该程序只接受常规字符（包括回车键及退格键），对特殊功能键不进行处理。在程序中设置这些特殊功能键对应 0 编号即可，凡是检测到键盘位置编号为 0 值时，均忽略对它们的处理。

有键按下时，送出扫描码的 D7 位为 0，当键抬起时，扫描码的 D7 位为 1，以判定键是否被按下。

4. 实验准备

（1）预习输入输出程序设计的特点和方法。

（2）仔细阅读参考程序，弄清外围芯片接口初始化的意义和方法。

5. 实验步骤

（1）建立源文件，并通过汇编和连接，产生可执行文件。

（2）运行程序，观察常规字符键及特殊功能键按下时程序的反应。

6. 实验报告要求

（1）给出程序框图，包括主程序框图、中断处理程序框图。

（2）说明输入输出程序设计的特点。

（3）讨论：将左 shift 键及右 shift 也进行判别处理，它们的扫描码为 42 及 54。为记录 shift 键的按动状态，可设一个标志单元 KBFLAG，右 shift 按下，KBFLAG 的 D0 位置 1，左 shift 按下，KBFLAG 的 D1 位置 1，放下左右的 shift 键，KBFLAG 的相应位恢复为 0。

当程序工作时，就能显示上档键的字符。

7. 参考程序清单

;KEYPRG..ASM 为键盘输入处理子程序

```
STACK      SEGMENT PARA STACK 'STACK'
            DB 256 DUP(0)
```

```
STACK      ENDS                                ;设置 256 个字节堆栈区
```

```
DATA      SEGMENT PARA PUBLIC 'DATA'
```

```
BUFFER    DB 10 DUP(0)                        ;定义 10 个字节的键盘缓冲区
```

```
BUFFER1   DW 0                                ;指向键盘缓冲区的起点
```

```
BUFFER2   DW 0                                ;指向键盘缓冲区的终点
```

```
;注意当 BUFFER1=BUFFER2 时,表明缓冲区空
```

```
SCANTAB   DB 0,0,'1234567890-=',8,0
            DB ' QWERTYUIOP[]',0DH,0
            DB ' ASDFGHJKL;',0,0,0,0
            DB ' ZXCVBNM,./',0,0,0
            DB ' ',0,0,0,0,0,0,0,0,0,0,0
            DB 0,0,'789-456+1230.'
```

```
DATA      ENDS
```

```

        EXTRN KBINT;FAR                ; 外部引用说明
CODE    SEGMENT PATA PUBLIC 'CODE'
        ;主程序段
        ASSUME CS:CODE
START:  MOV AX, DATA
        MOV DS, AX
        ASSUME DS:DATA
        ;建立自行设计的中断服务程序
        CLI                            ;关中断, 以防引起混乱
        MOV AX, 0                      ;确定中断类型号 9 的物理地址
        MOV ES, AX
        MOV DI, 24H
        MOV AX, OFFSET KBINT
        ;写自行设计的中断处理程序入口偏移量到中断适量表中
        CLD
        STOSW
        MOV AX, CS
        STOSW                          ;再写入中断处理程序段基址
        MOV AL, 0FCH                   ;允许定时器和键盘中断的控制字送 8259
        OUT 21H, AL
        STI                            ;开中断
        ;读键盘并显示字符
FOREVER:CALL KBGET1                    ;读缓冲区字符
        PUSH AX
        CALL DISPCHAR                  ;显示接收字符
        POP AX
        CMP AL, 0DH                    ;是回车符吗?
        JNZ FOREVER                   ;不是, 再接收
        CALL DISPCHAR                  ;是, 换行
        JMP FOREVER                    ;接收下一个字符
        ;KBGET1 取缓冲区字符->AL (返回参数)
KBGET1  PROC NEAR
KBGET:   PUSH BX
        CLI
        MOV BX, BUFFER1
        COM BX, BUFFER2
        JNZ KBGET2                     ; 缓冲区不空, 转 KBGET2
        STI                            ; 开中断
        POP BX
        JMP KBGET
        ; 取缓冲区字符
KBGET2:  MOV AL, [BUFFER+BX]            ; 取缓冲区字符
        INC BX                          ; 首指针加 1
        CMP BX, 10                      ; 到缓冲区尾吗?

```



```

                JC KBGET3                ; 未到，转 KBGET3
                MOV BX, 0                ; 到，指向缓冲区首
KBGET3:  MOV BUFFER1, BX                ; 保护首指针
                STI
                POP BX                  ; 恢复 BX 寄存器
                RET
KBGET1  ENDP
                ; DISPCHAR 字符显示子程序
DISPCHAR PROC NEAR
                PUSH BX
                MOV BX, 0                ; 显示零页
                MOV AH, 14H             ; 写功能
                INT 10H                 ; 调显示器中断调用
                POP BX
                RET
DISPCHAR ENDP
CODE        ENDS
                ENDS START

```

从 FOREVER 开始，程序便进入循环，反复读缓冲区字符，并将其显示出来。而键盘缓冲区中的内容，是通过键盘中断 9 的中断处理程序 KBINT 写入的。KBINT 程序如下：

```

; KBINT 键盘中断程序
SEG SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:SEG
        PUBLIC KBINT
KBINT    PROC FAR
                PUSH BX
                PUSH AX
;读键盘数据，并发确认信号
                IN AL, 60H                ; 读 8255A 口，即键盘扫描码
                PUSH AX
                IN AL, 61H                ; 为置确认信号
                                                ; 读 8255A B 口
                OR AL, 80H                ; 置键盘确认信号
                OUT 61H, AL              ; 置 PB7 位并送键盘
                AND AL, 7FH              ; 恢复键盘确认信号
                OUT 61H, AL
                ;将接收的扫描码译为 ASCII 码
                POP AX
                TEST AL, 80H              ; 检查键是否按下？
                JNZ KBINT2                ; 键抬起放弃本次输入
                MOV BX, OFFSET SCANTAB
                XLAT SCANTAB              ; 查表找对应 ASCII 码
                CMP AL, 0                  ; 是有效字符键吗？
                JZ KBINT2                 ; 不是，放弃

```

```

; 存 ASCII 码字符到缓冲区
MOV BX, BUFFER2 ; 取缓冲区尾指针
MOV [BUFFER+BX], AL ; 存字符
INC BX ; 尾指针加 1
CMP BX, 10 ; 是否到区尾?
JC KBINT3 ; 不到, 转
MOV BX, 0 ; 到, 置新尾址
KBINT3: CMP BX, BUFFER1 ; 缓冲区满?
JZ KBINT2 ; 满, 转
MOV BUFFER2, BX ; 不满, 存尾指针
; 结束中断处理
KIBNT2: MOV AL, 20H ; 发 EOI, 结束 8259 中断
OUT 20H, AL
POP AX
POP BX
IRET ; 中断返回
KIBNT ENDP
SEG ENDS
END

```

实验 12 中断程序实验

实验目的

- 1、掌握中断通信程序的设计方法。
- 2、了解串行通信芯片 8250 的编程方法。

实验内容

1) 8250 初始化

8250 共有 10 个可访问的寄存器。对其中的 5 个寄存器，只需要在主程序的开始处用输出指令将它们初始化即可。要想实现单机实验，要对调制解调器控制寄存器 3FCH 正确地初始化。它的 D0 位等于 1 时，表示实际的数据终端已准备好调制解调器信号；D1 位等于 1，表示实际的请求发送的调制解调器信号，D2 位（QUT1）是辅助的用户设计输出，不使用；D3 位是辅助的用户设计的输出，为将 8250 产生的中断信号经系统总线传送到 8259A 中断控制器的输入端 IRQ4，此位必须为 1；D4 位若为 1，8250 的输出被回送到 8250 自己的输入中。利用这个特点，可以编写程序，不需任何附加设备，直接测试 8250 的工作是否正常；D5、D6、D7 位强迫置 0。

2) 8259 的初始化

为实现分别允许或禁止各中断源，可以通过把屏蔽字节写入中断屏蔽寄存器 IMR（端口地址为 21H）来实现。

例如希望禁止所有外设中断，只保留键盘中断（连 8259 的 IRQ1），可编程如下：

```

MOV AL, 0FDH; 只有 IRQ1 未屏蔽
OUT 21H, AL

```

当然，中断信号能否被 CPU 所接收，还需由标志位 IF 来决定。IF=1，CPU 允许中断。在 CPU 执行完中断服务程序是，应当把这一信息通知 8259。其方法是向 20H 端口送一个中断结束命令字节 EOI，其内容为 20H。将 EOI 送至 8259，通知本次外设中断已结束。

收发程序的实现

发送程序的数据来自键盘输入，它将接收到的数据发送到 8250 串行口，并在屏幕上显示。接收程序接收到数据后，也在屏幕上显示出来。因此，程序正常执行后，从键盘输入一个数字，在屏幕上显示两个同样的数据，其中一个发送程序发送的数据，另一个是接收程序接收到的字符数据。

接收数据是通过接收中断处理程序实现的。

实验准备

(1) 预习关于中断程序设计的有关内容。

(2) 熟悉 8250 初始化的方法，8259 有关命令字和中断向量表的设置等，熟悉中断程序的结构。

实验步骤

(1) 建立源文件，汇编、连接产生可执行文件。

(2) 运行程序，使发送的字符串能被正确地接受。

实验报告要求

(1) 由参考程序，画出收发通信的程序框图。

(2) 对 8250 及 8259 芯片，弄清程序是如何实现对它们进行初始化和控制的，写出有关的初始化控制字节并说明其意义。

参考程序清单

```
; SINT. ASM
STACK . SEGMENT PARA STACK 'STACK'
    DB 256 DPU (0)
STACK ENDS
CODE SEGMENT PARA PUBLIC 'CODE'
RTATS PROC FAR
    ASSUME CS:CODE
STRAT: PUSH DS
        XOR AX , AX
        PUSH AX                ; 保存返回地址
        CLI
        MOV AX , 0
        MOV ES , AX            ; ES 指向中断向量表
        MOV DI , 0CH*4        ; 串行口中断型号为 0CH
        MOV AX , OFFSET RECEIVE
        CLD
        STOSW                  ; RECEIVE 为接受中断入口
        MOV AX , SEG RECEIVE
        STOSW                  ; 以上四条指令完成置 0CH 的中断向量
        MOV AL , 2CH
        OUT 21H , AL           ; 允许串口、键盘及时钟中断，初始化为 8259
        MOV DX , 3FBH         ; 通信线控制端口
        MOV AL , 80H
        OUT DX , AL           ; 初始化波特率寄存器
        MOV DX , 3F8H         ; 将波特率初始化为 1200 波特，分别送 3F8H 及
                                ; 3F9H
```

```

MOV AL , 60H
OUT DX , AL
MOV AL , 0H
MOV DX , 3F9H
OUT DX , AL
MOV DX , 3FBH ; 奇校验, 1 位停止位
MOV AL , 0AH ; 7 位代码
OUT DX, AL
MOV DX , 3FCH ; 设置 MODEM 寄存器的控制信号, 设置为单机
; 方式

MOV AL , 1BH
OUT DX , AL
MOV DX , 3F9H ; 允许数据接受中断
MOV AL , 01H
OUT DX , AL
STI ; 开中断
FOREVER : MOV DX , 3FDH ; 读线路状态寄存器
IN AL , DX
TEST AL , 20H ; 是否允许发送
JZ FOREVER ; 否则继续查询
MOV AH , 1 ; BIOS 16H 中断, 判有无键盘输入
INT 16H
JZ FOREVER ; 无键盘输入继续查询
MOV AH , 0 ; 读入键盘输入
INT 16H
INC SI ; 发送次数加 1
CMP SI , 11 ; 超过 10 个字符结束
JE GOOD
MOV BX , 0
MOV AH , 14 ; 显示键入的字符
INT 10H
MOV DX , 3F8H
OUT DX , AL ; 将字符发送出去
JMP FOREVER ; 循环, 随时可产生接受中断
GOOD : RET
RECEIVE PROC NEAR
PUSH AX
PUSH BX
PUSH DX
PUSH DS
MOV DX , 3FDH ; 读线路状态寄存器
IN AL , DX
TEST AL , 1EH
JNZ ERROR ; 有错误转错误处理

```

```

MOV AL , DX
MOV DX , 3F8H
    IN  AL , DX                ; 输入数据
    AND AL , 7FH
    PUSH AX                    ; 以下显示输入字符
    MOV BX , 0
    MOV AH , 14
    INT 10H
    POP AX
    CMP AL , 0DH
    JNZ RTN
    MOV AL , 0AH                ; 显示回车换行
    MOV AH , 14
    MOV BX , 0
    INT 10H
RTN:    MOV AL , 20H            ; 向 8259 发 EOI 指令
    OUT 20H , AL
    POP DS
    POP DX
    POP BX
    POP AX
    STI                        ; 开中断
    IRET                       ; 中断返回
ERROR : MOV DX , 3F8H
    IN  AL , DX                ; 从数据寄存器中读一字节，一清除超时错误
    MOV AL , ' ? '            ; 显示问号错误指标
    MOV BX , 0
    MOV AH , 14
    INT 10H
    JMP RTN                    ; 返回中断程序
RECEIVE ENDP
RTATS   ENDP
CODE    ENDS
        END START

```

系统中预制的 DOS 或 BIOS 中断调用，给用户编制程序带来极大的方便，但在系统配置有专用设备，或需要扩充原中断例行程序的功能，或需要设备执行完全不同于原中断例行程序的功能时，就需要用户自编中断处理程序。

这类程序应解决好以下问题：

1) 将原中断向量保存在堆栈中或自设的存储单元中。

若中断类型为 INT-TYPE 并将中断类型号对应的中断向量保存在 KEEP-IP 及 KEEP-CS 地址中，则以下程序段可以实现该操作：

```

MOV AH , 35H
        ;DOS 系统调用 35 号功能，可将原中断向量段址 → ES，偏移
        ; 地址    BX → 中

```

```

MOV AL , INT-TYPE
INT 21H
MOV KEEP - IP, BX
MOV KEEP - CS, ES

```

- 2) 将自编的中断处理程序入口地址置入中断向量表中。该操作可以用 DOS 系统调用 25H 实现，这时应将自编中断处理程序的段内偏移地址送 DX，断基址送 DS，中断类型号送 AL 中作为入口参数。相应的程序段如下：

```

PUSH DS
MOV DX , OFFSET ROUTINE
MOV AX , SEG ROUTINE
MOV DS , AX
MOV AL , INT - TYPE
MOV AH , 25H
INT 21H
POP DS

```

当用 25H 功能改变中断向量时，会自动禁止硬件中断，因此在设置新的中断向量时，硬件中断不会使用中断向量。

- (3) 在中断处理程序结束时，必须恢复原来的中断向量，否则后继程序不能正确使用系统提供的例行程序。相应的程序段如下：

```

PUSH DS
MOV DX , KEEP-IP
MOV AX , KEEP-CS
MOV DS , AX
MOV AL , INT - TYPE
MOV AH , 25H
INT 21H
POP DS

```

中断处理子程序中，应由 IRET 指令从中断处理程序返回。

实验 13 系统调用程序实验

1. 实验目的

学习借助于系统功能调用程序设计方法，并掌握系统功能调用的方法。

学习 BIOS 功能调用的方法，并掌握用 BIOS 功能调用编制应用程序的方法。

2. 实验内容

(1) 利用 INT 16H 的 0 号功能，判断键入字符的 ASCII 值是否为 0，若为 0，则显示其扫描码，否则就显示该字符。若该字符为“q”，则结束程序的执行。

(2) 利用 0EH 及 19H 系统调用，获取系统当前逻辑驱动器的数目，并显示最后一个有的驱动器字母。

(3) 将文件属性改变为隐含文件。

(4) 在当前驱动器的当前目录下建立一个新的子目录 NEW_DIR，并使之成为新的当前目录。

实验准备

- (1) 仔细阅读教材中关于 DOS 系统功能及 BIOS 中断调用的内容，弄清与实验有关的系统调用及 BIOS 中断调用的使用方法，弄清它们需要预置的入口参数及调用后的返回参

数。

(2) 认真阅读参考程序，

实验报告要求

1、画出参考程序流程图。

2、说明程序中所使用的系统功能调用及 BIOS 中断调用的方法及需要的参数设置。

参考程序清单

； KEYTEST.ASM 查看并显示键代码程序

```
cseg segment
    assume cs:cseg
    org 0100h
main proc far
start: push ds
      sub ax,ax
      push ax
again: mov ah,0
      int 16h
      cmp ax,0
      jz next
      cmp al,0dh
      jz next
      cmp al, ' q'
      jz exit
      mov bx,ax
      xchg bl,bh
      call prtkey
next: mov dl,0dh
      mov ah,2
      int 21h
      MOV DL, 0AH
      MOV AH, 2
      INT 21H
      JMP AGAIN
EXIT:    RET
MAIN    ENDP
PRTKEY  PROC NEAR
        PUSH CX
        PUSH DX
        MOV CH, 4
ROTATE: MOV CL, 4
        ROL BX, CL
        MOV AL, BL
        AND AL, 0FH
        ADD AL, 30H
        CMP AL, 3AH
```

```

                JL DISPLAY
                ADD AL, 07H
DISPLAY:  MOV AH, 2
                INT 21H
                DEC CH
                JNZ ROTATE
                POP DX
                POP CX
                RET
PRTKEY    ENDP
CSEG      ENDS
                END START

```

此处是将 ASCII 码及扫描码以 4 位十六进制数同时显示出来。若按原题意要求，PRTKEY 可修改如下：

```

PRTKEY    PROC NEAR
                PUSH CX
                PUSH DX
                CMP BH, 0
                JZ ROTATE
                MOV DL, BH
                MOV AH, 2
                INT 21H          ; 显示 ASCII 码
                JMP RETURN
ROTATE:     MOV CL, 4
                MOV AL, BL
                SHR BL, CL
                ADD BL, 30H
                CMP BL, 3AH
                JL  DISPLAY
                ADD BL, 07H
DISPLAY:    MOV DL, BL
                MOV AH, 2
                INT 21H
                ADD AL, 0FH
                CMP AL, 3AH
                JA  DISPLAY1
                ADD AL, 07H
DISPLAY1:   MOV DL, AL
                MOV AH, 2
                INT 21H
RETURN:     POP DX
                POP CX
                RET
PRTKEY    ENDP

```



```

CSEG      ENDS
          END START
;LASTDR, ASM 获取系统逻辑驱动器号的程序
SSEG      SEGMENT PARA STACK 'STACK'
SSEG      ENDS
DSEG      SEGMENT WORD PUBLIC 'DATA'
MSG        DB 'LASTDRIVE='
DLETTER    DB ?
          DB 0DH, 0AH, '$'
DSEG      ENDS
CSEG      SEGMENT WORD PUBLIC 'CODE'
          ASSUME CS:CSEG, DS:DSEG, SS:SSEG
MAIN      PROC
START:     MOV AX, DSEG
          MOV DS, AX
          MOV AH, 19H
          ;取驱动器号系统调用, 缺省
          ;驱动器号--AL

          ;0EH 调用, 把 DL 中指定的驱动器号置为
          ;当前驱动器号, 并将逻辑驱动器号回送
          ;AL

          INT 21H
          MOV DL, AL
          MOV AH, 0EH
          INT 21H
          MOV BL, AL ;将 AL--BL
          ADD AL, 'A'-1 ;将 AL 驱动器内容换成驱动器字母号, AL 为 0 对应 A 驱
          MOV BDLETTER, AL ;写入 MSG 串中
          MOV DX, OFFSET MSG
          MOV AH, 9
          INT 21H
          MOV AH, 4CH
          INT 21H
MAIN      ENDP
CSEG      ENDS
          END START
;RENFILE. ASM 将文件属性改为隐藏属性, 设程序中的文件为 LASTDR. ASM
DATA      SEGMENT
N          DB 'LASTDR. ASM', 0, 0
DATA      ENDS
CODE      SEGMENT
          ASSUME CS:CODE, DS:DATA
START:     MOV AX, DATA

```

```

MOV DS, AX
MOV AH, 43H          ;43H 为 DOS 取/置文件属性的系统调用
mov al, 01H
MOV DX, OFFSET N;
MOV CX, 02H          ;AL=01 为置属性
INT 21H              ;DX 指向文件路径
MOV AH, 4CH          ;名 ASCII 串首址, CX 为文件属性, 02 为隐藏属性
INT 21H
CODE ENDS
END START

```

若要改为可读属性, 则只要将程序 RENFILE。ASM 的 CX 值, 由 02H 改为 00H 即可
;MDOS.ASM 在当前目录下, 新建一个子目录 NEW-DIR, 并置为当前目录

```

CSEG SEGMENT
ASSUME CS: CSEG, DS, CSEG
ORG 0100H
START: MOV AH, 39H
MOV DX, OFFSET NEW-DIR
INT 21H
JZ ERROR
MOV AH, 3BH
MOV DX, OFFSET NEW-DIR
INT 21H
JC ERROR
MOV DX, OFFSET MSGOK
JMP NEXT
ERROR: MOV DX, OFFSET MSGERR
NEXT: MOV AH, 9
INT 21H
MOV AH, 4CH
INT 21H
MSGOK DB 0DH, 0AH, 'OK', 0DH, 0AH, '$'
MSGERR DB 0DH, 0AH, 'ERROR', 0DH, 0AH, '$'
NEW-DIR DB 'NEW-DIR', 0
CSEG ENDS
END START

```

以上 4 个程序可分为两次完成。有关 DOS 的系统调用以及 BIOS 中断调用还可以自行设计程序, 例如对于文件的建立, 打开, 读, 写等功能调用, 10H 视频显示中断调用, 13H 磁盘 I/O 中断调用等。

实验 14 语言接口实验

1 实验目的

- (1) 掌握高级语言与汇编语言接口方法
- (2) 正确编写混合语言程序

2 实验内容

用高级语言 C 编写主程序，完成两个十进制数的相加运算，其中加法运算要求用汇编语言编写，实现高级语言对汇编语言的调用。

3 实验步骤

(1) 在 C 语言环境下，调用汇编程序

(2) 在 C 语言程序中，嵌入汇编语句

要求对以上两个程序进行汇编、编译、连接后，生成.exe 可执行文件。

4.实验报告要求

(1) 画出参考程序的框图。

(2) 说明 C 语言调用汇编语言程序的具体方法和过程。

(3) 说明在 C 语言中嵌入汇编语句的方法、格式和过程。

5. 参考程序清单

```
/*CPROG1.C 在 C 语言中调用汇编子程序*/
#include <stdio.h>
int ascadd(int, int, int*);
main()
{int i, j;
int k;
printf("please input i, j=?");
scanf("%d, %d", &i, &j);
ascadd(i, j, &k);
printf("the %d Add %d is %d\n", i, j, k);
}
PUBLIC _ascadd
_TEXT SEGMENT BYTE PUBLIC 'CODE'
DGROUP GROUP _DATA, _BSS
_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS
_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS
ASSUME CS: _TEXT, DS: DGROUP, SS: DGROUP
_ascadd proc near
    push bp
    move bp, sp
    push si
    move ax, [bp+4]
    move bx, [bp+6]
    cld
    add al, bl
    daa
    xchg al, ah
    adc al, bh
    daa
    xchg al, ah
```

```

        mov si, [bp+8]
        mov [si], ax
        pop si
        pop bp
        ret
_ascadd endp
_TEXT   ENDS
        END

```

其中，_TEXT 为代码段；

_DATA 存放已初始化全局数据和静态数据。

_BSS 存放未初始化的全局数据和静态数据。栈和数据段共组，也称为小模式。

6. 操作方法

(1) 在 TC 集成环境下编辑成一个源文件。

(2) 在 TC 子目录中应有 TASM.EXE 文件，即将 MASM V5.0 中的 MASM.EXE 改名为 TASM.EXE 并拷备过来。

(3) 在 TC 子目录下，键入以下命令：

```

TCC
/*CPROG2.C, C 语言与汇编混合编程*/
main()
{
    int i, j, k;
    printf("please input i, j=?");
    scanf("%d, %d", &i, &j);
    #asm
    mov ax, i
    mov bx, j
    cld
    add al, bl
    daa
    xchg al, ah
    adc al, bh
    daa
    xchg al, ah
    mov k, ax
    #endasm
    printf("the result is: %d", k);
}
/*CPROG3.C, 与 CPROG2.C 等价*/
main()
{
    int i, j;
    printf("please input i, j=?");
    scanf("%d, %d", &i, &j);
    asm mov ax, i

```

```

asm mov bx, j
asm add al, bl
asm daa
asm xchg al, ah
asm adc al, bh
asm daa
asm mov j, ax
printf("the result is:%d", j);
}

```

第三章 高级接口

汇编语言和高级语言的接口内容属于汇编语言的高级应用，对于初学者来讲，只作一般了解即可。由于目前比较流行汇编语言与 C 语言的混合编程，这两种语言的相互调用仍有着重要应用。作为汇编语言的上机指导，此处结合实例作一点专门介绍。

汇编语言是直接对计算机系统硬件进行操作的语言，要求编程者不仅对汇编语言有较深入的了解，而且要求他们对计算机硬件的工作原理也有相当的了解。即使如此，要编写复杂的计算程序远没有使用高级语言那样简便。而 C 语言提供了极为丰富的编程函数，其中包括对 MS—DOS 系统调用和中断调用的函数。如果我们把 C 语言编程和汇编语言编程结合起来，将会使编程功能更加强大，它主要体现在以下 3 个方面：

(1) 可以提高程序运行的速度和效率，如在显示及需要反复使用系统资源等场合。

(2) 为了实现某些 C 语言中不具备的功能，但又必须适合不同机种所特有的功能，如配接外部设备时，仅仅为用户提供汇编语言使用硬设备的程序。

(3) 利用 C 语言通用框架程序，嵌入可利用的大量的汇编例程，可以提高编程的速度。

把汇编程序块和用户的 C 语言程序结合起来，又两种常用的简便方法，第一种是在 C 语言程序中嵌入你所需要的汇编例程；第二种是在 C 语言程序中调用汇编例程，只要在自动生成的汇编语言框架程序中，填写一些我们所需要的语句就可以了。

一、在 C 语言程序的语句行间嵌入汇编程序

当所需使用汇编程序较短时，可以在汇编语句行之前加上 `asm` 关键字，使汇编语句直接成为 C 语言程序的一部分；另外可用两个预处理程序的伪指令 `#asm` 和 `#endasm`。`#asm` 伪指令指出一个汇编程序块的开始，而 `#endasm` 伪指令则指出汇编程序块的程序。

例 1：嵌入汇编部分语句的 C 语言例程。

```

main()
{
    int i,j;
    char *s;
    printf("please input:i=");
    scanf("%d",&i);
    #asm
    mov ax,i
    mov cl,2
    mul cl

```

```

mov j,ax          /*乘积送 C 变量 j, 直接寻址*/
#endasm          /*汇编块结束预处理*/
printf("the result is:&d*2=%d",i,j);
getch();
}

```

要注意 #asm 和 #endasm 之间的语句是汇编语句，其结尾不加 C 语句的结尾符 “;”。

操作方法：

(1)在 TC 集成环境下输入源程序，编辑成一个文件。

(2)把 MASM.EXE 改名为 TASM.EXE 复制到 TC 子目录下。

(3)用 TCC.EXE 编译连接，即：

TCC—B—L(库文件路径)文件名 库文件名(其中 TC 含有的库可省略)。

例 2 利用 BIOS 调用功能画色板的 C 语言和汇编语言的混合编程。

```

#include<dos.h>
#include<graphics.h>
main()
{
    int i=256;
    int gdriver=DETECT,gmode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&gdriver,&gmode,"c:\\TC");
    bar3d(100,200,200,300,100,1);
    printf("Lu Turbo c VEA:640*480,16color");
    sleep(3);
    asm mov bl,i;
    asm mov ah,0;
    asm mov al,13h;
    asm int 10h;
    asm mov cx,290;      /*CX 中为列号*/
lop:   asm mov dx,199;    /*DX 中为行号*/
lop0:  asm mov,ah,0ch;    /*INT10H 写像素点*/
    asm mov al,bl;      /*BL 中确定点的颜色*/
    asm mov bh,0
    asm int 10h;        /*屏幕写点中断*/
    asm dec dx;         /*行号减 1*/
    asm cmp dx,0;       /*判行结尾*/
    asm jne lop0;       /*未结束继续写电*/
    asm dec cx;         /* 列号减 1*/
    asm dec bl;         /*换一种颜色号*/
    asm cmp bl,0;       /*颜色号用完否?*/
    asm jne lop;        /*未完继续显示*/
    printf("Assembly VGA 320*200,256");
    getch();
    funl();
    getch();
}

```

```

closegraph();          /*关闭屏幕图形方式显示*/
printf("The end");
}

fun1()
{
    asm mov ah,0;        /*屏幕显示模式*/
    asm mov al,12h;      /*VGA: 640×480, 16 色*/
    asm int 10h;         /*屏幕中断*/
    asm mov dx,240;      /*DX 中为行号*/
lop1:asm mov cx,440; /*CX 中为列号*/
lop2:asm mov ah,0ch;     /*0ch 为写像素点*/
    asm mov al,4;        /*颜色 4 为红色(RED)*/
    asm mov bh,0;
    asm int 10h;
    asm dec cx;
    asm cmp cx,200;      /*列号为 200 否?*/
    asm jne lop2;
    asm dec dx;
    asm cmp dx,120;
    asm jne lop1;
    printf("Assembly VGA: 640*480,16color");
}

```

例 2 中采用 `asm` 标识符开头,在不用预处理伪指令 `#asm` 时,每一句汇编都要写上 `asm`。值得注意的是,如果有标号,则标识符 `asm` 应在标号之后。而每行结尾都应按 C 语言规则加上“;”号。这里利用汇编语言 BIOS 调用例程,它可以直接放在 C 语言中,也可以作为一个子程序,让 C 语言调用,像 `fun1()` 函数那样。

上机操作方法:

- (1)在 TC 集成环境下编辑成一个源程序文件。
- (2)TC 子目录下仍要有 TASM.EXE。
- (3)TCC—B—L C: \TC\LIB 文件名。

(因为 TC\LIB 中有 GRAPHICS.LIB 可省略后面的库文件名。)

二、编写被 C 语言调用的汇编子程序

以这种形式编写程序时要注意 C 语言向汇编传送的数据,以及返回的数据的特点

例 3

```

#include<stdio.h>
int ascmuyl(int,int,long*)
main()
{
    int i,j;
    long k;
    printf("please input i,j=?");
    scanf("%d,%d",&i,&j);
    ascmul(i,j,&k);
    printf("the %d times %d is %Ld\n",i,j,k);
}

```

```
}
```

能够被上述 C 语言调用的汇编程序，名为 ascmul(i,j,&k),内容如下：

```
PUBLIC _ascmul
_TEXT SEGMENT BYTE PUBLIC 'CODE'
DGROUP GROUP _DATA, _BSS
_DATA SEGMENT WORD PUBLIC 'DATA'
_DATA ENDS
_BSS SEGMENT WORD PUBLIC 'BSS'
_BSS ENDS
ASSUME CS:_TEXT,DS:DGROUP,SS:DGROUP
_ascmul proc near
    push bp
    mov bp,sp
    push si
    mov ax,[bp+4]    ; 数据 i  → AX
    mov bx,[bp+6]    ; 数据 j  → BX
    mul bx           ; 计算 AX*BX
    mov si,[bp+8]    ; 取&k 地址送 SI
    mov [si],ax
    inc si
    inc si
    mov [si],dx
    pop si           ; 恢复 SI
    pop bp           ; 恢复 BP
    ret
_ascmul endp
_TEXT ENDS
END
```

C 语言可调用的汇编语言程序格式是固定的，其含义可解释为：

`_TEXT` 段是一个代码段，以 `_TEXT SEGMENT` 开始，后跟 `BYTE PUBLIC 'CODE'`，即以字节定界，与 `'CODE'` 段组合在一起，构成一个物理段，以 `_TEXT ENDS` 结束该代码段。

`_DATA` 段用来存放已初始化的全局数据和静态数据，以 `_DATA ENDS` 结束该段。

`_BSS` 段是存放未初始化的全局数据和静态数据，以 `_BSS ENDS` 结束该段。

从上面的程序可以看出，编程中的内容管理是栈、数据段共组，这就是通常所说的小模式，现在介绍的方法使用于小模式，其格式就如例程中的汇编部分，而且这些标识符不能随意改动。

关于数据传送的内存管理规律，根据上面实例可得到证实。其中 C 语言程序调用 `ascmul` 函数，`ascmul(i,j,&k)` 将传送 3 个数据，它们都在栈中。

调用函数时，按 `i,j,&k` 顺序将它们压入堆栈，在执行与 `CALL` 相似的调用函数时，近调用将返回地址压入堆栈，进入汇编子程序后，用 `MOV BP, SP` 将 `SP` 保存到 `BP` 中然后用 `BP+4` 指向 `i` 参数，将 `BP+6` 指向 `j` 参数，用 `BP+8` 指向 `&k` 参数。

由于 `&k` 被定义为长整数的地址指针，故用指令 `MOV SI, [BP+8]` 将 `&k` 参数送 `si` 后，将 32 位的乘积的低 16 位存入 `si` 所指的两个单元中去，将乘积的高 16 位存入 `si+2` 所指的两个单元中去。

例 4 用 C 语言编程设定人机界面及工作参数，再调用汇编和硬件直接发生关系。在这种情况下，使用例 3 的设计形式是很合适的。现有一个通用发声程序 GENSOUND，它利用定时器发出指定频率。设定发音时间，即可演奏歌曲。具体原理可参阅 PC 机接口硬件资料。

通用程序如下：

```

PUBLIC GENSOUND
    CSEG    SEGMENT PARA 'CODE'
        ASSUME CS:CSEG
    GENSOUND PROC FAR
        PUSH AX
        PUSH BX
        PUSH CX
        PUSH DI
        MOV AL,0B6H; 定时器工作方式初始化，将定时器
        OUT 43H,AL; 2# 设置为方式 2，产生方波
        MOV DX,120H;定时器 2#初值
        MOV AX,533H*896
        DIV DI
        OUT 42H,AL ;先送初值低 8 位
        MOV AL,AH
        OUT 42H,AL ;再送初值高 8 位
        IN AL,61H ;读取 61H 口状态
        MOV AH,AL;保存在 AH 中
        OR AL,3;61H 的 D0, D1 置 1，允许定时器
            ;2# 技术且允许扬声器发声
        OUT 61H,AL
        MOV BX,100 ;10ms×100 设定延时常数
    DELAY: MOV CX,2801
    WAITER : LOOP WAITER
        DEC BX
        JNZ DELAY
        IN AL,61H ; 关闭扬声器
        AND AL,0FCH;
        OUT 61H,AL
        RET
    GENSOUND ENDP
    CSEG ENDS
    END

```