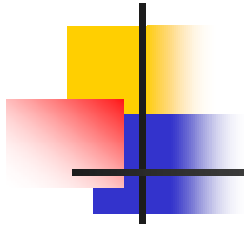


# 微机原理及应用

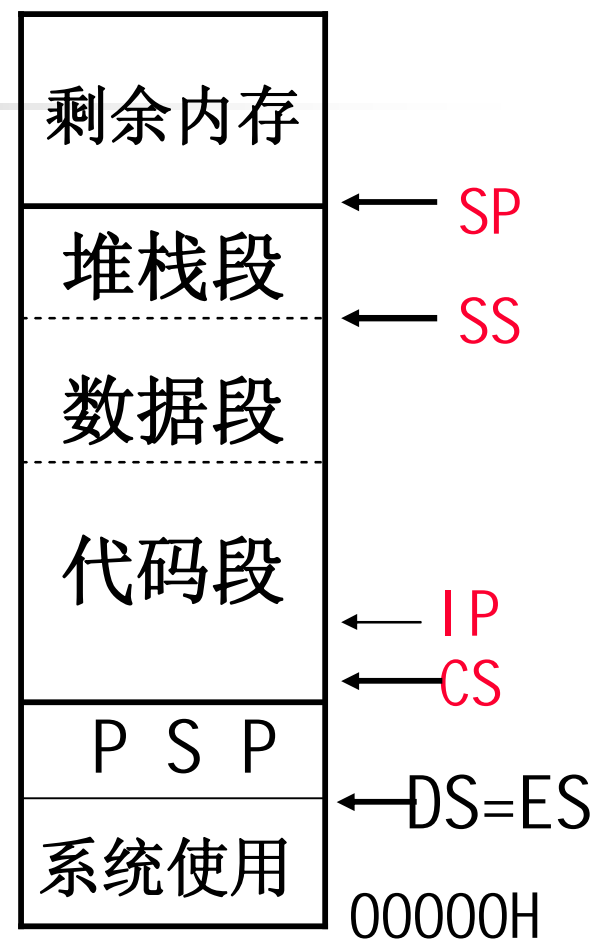
-Debug命令帮助



## n DOS程序结构（补充，不作要求）

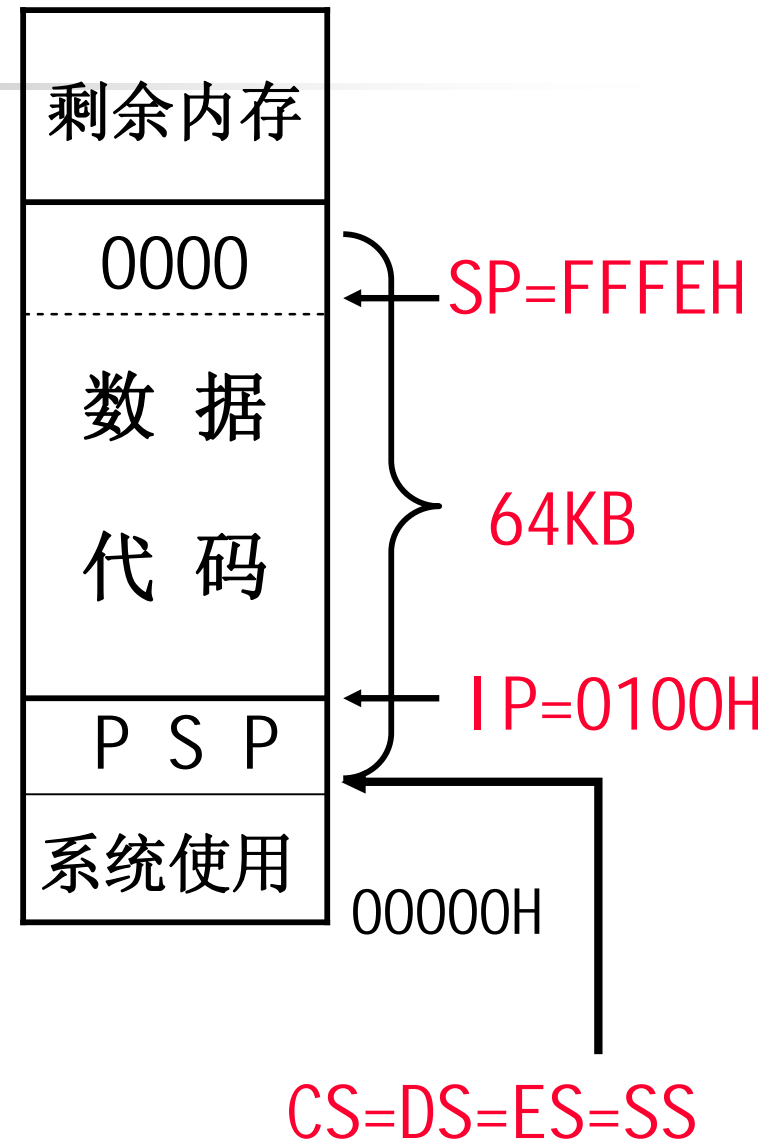
# 1. EXE程序

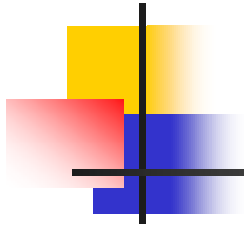
- n 可执行程序(executable program)以.exe为后缀，由多个段组成；长度可大于64KB；（磁盘上的）EXE文件和内存中的.exe文件不大一样，它包括：
  - n 文件头：控制信息、重定位信息；
  - n 装入模块：程序本身。
- n 程序由系统(命令解释器command.com或者debug.exe)装载进内存后，其段分布如图所示，初始化段寄存器内容如下：
  - 1) DS、ES指向PSP(Program Segment Prefix-程序段前缀，包含一些系统信息和命令行参数)段地址。程序中须重新设置，使其指向数据段（和附加段）。
  - 2) CS:IP和SS:SP由连接程序确定；如果不指定堆栈段，则SS=PSP的段地址，SP=100H，堆栈段占用PSP的部分区域，有时也能正常工作。为安全，应设置堆栈段。



## 2. COM程序

- n 代码段、数据段以及堆栈段（附加段）合并成一个段，长度不超过64KB。
- n 磁盘上的COM文件是内存的完全映象，不含附加信息。
- n com程序装载进内存后，
  - 1) 所有段地址都指向PSP的段地址；
  - 2) 执行起点：CS:0100 处。
  - 3) SP自动设为0FFFEH（64K的最后一个字单元。且将0FFFEH和0FFFFH单元设为0。





## n DOS系统功能调用



# DOS和BIOS调用

- n DOS (Disk Operation System)和BIOS(Basic Input and Output System)是为用户提供的两组系统服务程序。
- n BIOS是IBM PC/XT的基本I/O系统，负责管理系统的测试程序、初始化引导程序、一部分中断矢量装入程序及外部设备的服务程序。由于这些程序固化在ROM中，用户可以直接调用。
- n DOS是IBM PC/XT的操作系统，负责管理系统的所有资源，协调微机的操作，其中包括大量的可供调用的服务子程序，完成设备的管理和磁盘文件的管理。
- n 用户控制PC机硬件的方法：  
高级语言à调用DOS程序à使用BIOS程序à直接访问硬件



# DOS 系统功能调用

- n DOS和ROM-BIOS都以中断服务程序的形式向用户提供大量子程序，供用户编程时调用。“系统功能调用”一般是指调用DOS的INT 21H提供的子程序，调用BIOS提供的中断子程序一般称为“BIOS调用”。
- n 系统功能调用格式：
  - n 在AH中设置调用的功能号；
  - n 在指定的寄存器中设置入口参数；
  - n 执行INT 21H指令，调用功能子程序；
  - n 如果需要，分析出口参数。



## 1、单个字符输出

n AH=02; 入口参数: DL=字符的ASCII码

```
MOV AH, 2
```

```
MOV DL, 'A'
```

```
INT 21H ; 可输出控制字符
```

## 2.字符串输出

n AH=09; 入口参数: DS:DX=字符串首地址  
字符串必须以\$ (24H) 结尾

```
string db 'Hello!',0dh,0ah,'$'
```

```
mov ah,9
```

```
mov dx,offset string
```

```
int 21h
```





### 3. 单个字符输入

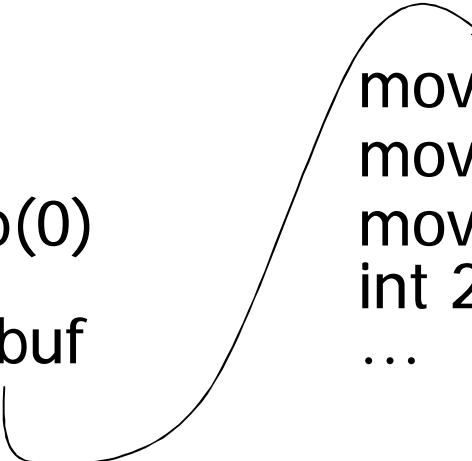
- n AH=1; 入口参数: 无; 出口参数: AL=字符ASCII码  
(系统一直等待, 直到有键按下)

### 4. 字符串的输入

- n AH=0AH; 入口参数: DS:DX=缓冲区首地址  
“回车”表示输入结束; 缓冲区第一个字节添入可能输入的最多字符数(含回车), 第二个字节将存放实际输入的字符数(不含回车), 从第三个字节开始存放输入的字符。

```
buf db 100
    db 0
    db 101 dup(0)
    ...
mov dx, seg buf
```

```
mov ds, dx
mov dx, offset buf
mov ah, 0ah
int 21h
...
```





## 5. 返回DOS

n AH=4CH,入口参数：无；出口参数：无

```
MOV AH,4CH
```

```
INT 21H
```

## 6. 利用INT 20H功能返回DOS

n 需要将主程序定义为DOS系统的子程序（在完整段定义部分已经说明）

```
MYPR PROC FAR
```

```
PUSH DS ;程序初始化时DS执行程序段前缀PSP
```

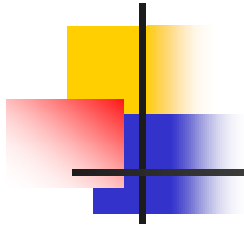
```
SUB AX,AX ;其中的第一条指令就是INT 20h
```

```
PUSH AX ;这三条指令把PSP的地址压入堆栈
```

.....

```
RET ;返回到PSP第一条指令
```

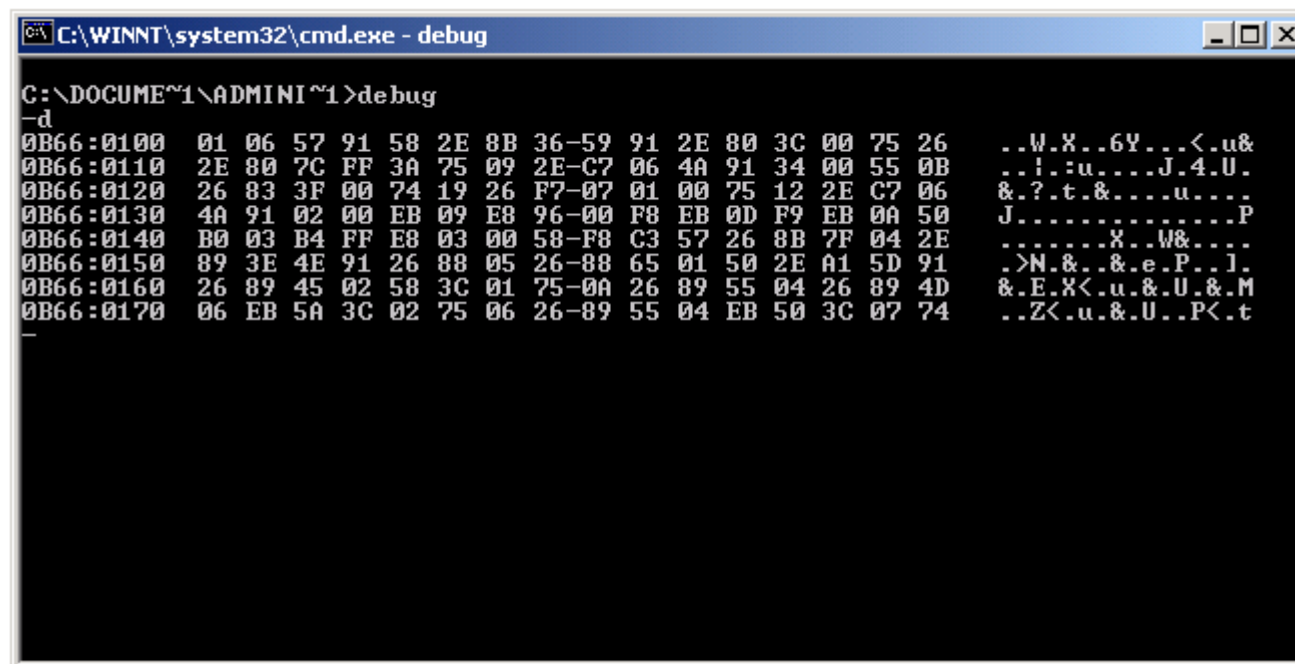
```
MYPR ENDP
```



## n DEBUG 命令解释

# 进入Debug的方法:

- n 桌面快捷栏à开始à运行(R)à输入cmd.exeà出现一黑屏窗口à在命令提示符后输入debug。在提示符-处可输入后面介绍的命令



```
C:\WINNT\system32\cmd.exe - debug
C:\DOCUME~1\ADMINI~1>debug
-d
0B66:0100 01 06 57 91 58 2E 8B 36-59 91 2E 80 3C 00 75 26 ..W.X..6Y...<.u&
0B66:0110 2E 80 7C FF 3A 75 09 2E-C7 06 4A 91 34 00 55 0B ...!.:u....J.4.U.
0B66:0120 26 83 3F 00 74 19 26 F7-07 01 00 75 12 2E C7 06 &?.t.&....u....
0B66:0130 4A 91 02 00 EB 09 E8 96-00 F8 EB 0D F9 EB 0A 50 J.....P
0B66:0140 B0 03 B4 FF E8 03 00 58-F8 C3 57 26 8B 7F 04 2E .....X..W&....
0B66:0150 89 3E 4E 91 26 88 05 26-88 65 01 50 2E A1 5D 91 .>N.&..&.e.P..l.
0B66:0160 26 89 45 02 58 3C 01 75-0A 26 89 55 04 26 89 4D &.E.X<.u.&.U.&.M
0B66:0170 06 EB 5A 3C 02 75 06 26-89 55 04 EB 50 3C 07 74 ..Z<.u.&.U..P<.t
-
```



## Debug:A(汇编)

- n 该命令从汇编语言语句创建可执行的机器码。

- n a [address]

- n 参数

**address** 指定键入汇编语言指令的位置。如果不指定地址，**a** 将在它上次停止处开始汇编

- n 输入指令，最后以Ctrl+C结束输入

- n 支持db、dw伪指令

**db 1,2,3,4,"THIS IS AN EXAMPLE"**

**dw 1000,2000,3000,"BACH"**



# Debug:C (比较)

---

n 比较内存的两个部分。

c range address

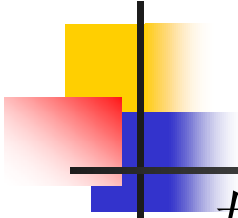
n 参数

range 指定要比较的内存第一个区域的起始和结束地址，或起始地址和长度。

1. [segr:]offset1 offset2 例如 cs:100 10F

2. [segr:]offset1 / length cs:100 / 10

n Address 指定要比较的第二个内存区域的起始地址

- 
- n 如果 range 和 address 内存区域相同，Debug 将不显示任何内容而直接返回到 Debug 提示符。如果有差异，Debug 将按如下格式显示：

address1 byte1 byte2 address2

- n 举例 c100,10f 300

对 100h 到 10Fh 的内存数据块与 300h 到 30Fh 的内存数据块进行比较

197F:0100 4D E4 197F:0300

197F:0101 67 99 197F:0301

197F:0102 A3 27 197F:0302

197F:0103 35 F3 197F:0303

○ ○ ○ ○



# DEBUG:D(显示内存内容)

---

显示内存单元的内容

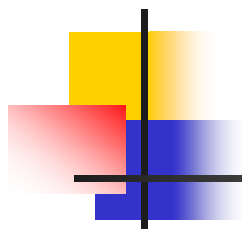
d [range]

n 参数

range

指定要显示其内容的内存区域的起始和结束地址，或起始地址和长度。如果不指定 range，Debug 程序将从以前 d 命令中所指定的地址范围的末尾开始显示 128 个字节的内容，范围仅限于同一段。





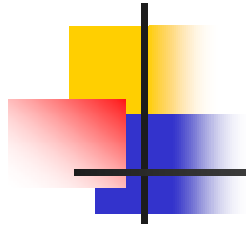
## 十六进制部分, 16 字节/行

## ASCII 码部分

```
-d 0
08B7:0000  CD 20 FF 9F 00 9A F0 FE-1D F0 42 02 64 05 70 02  M ....p~.pB.d.p.
08B7:0010  64 05 56 01 15 04 64 05-01 01 01 00 02 FF FF FF  d.U...d.....
08B7:0020  FF FF FF FF FF FF FF FF-FF FF FF FF 66 08 C6 2A  .....f.F*
08B7:0030  64 05 14 00 18 00 B7 08-FF FF FF FF 00 00 00 00  d....7.....
08B7:0040  05 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
08B7:0050  CD 21 CB 00 00 00 00 00-00 00 00 00 00 20 20 20  M!K.....
08B7:0060  20 20 20 20 20 20 20 20-00 00 00 00 00 20 20 20  .....
08B7:0070  20 20 20 20 20 20 20 20-00 00 00 00 00 00 00 00  .....
-d
08B7:0080  00 0D 61 69 6E 31 2E 65-78 65 0D 73 79 73 74 65  ..ain1.exe.syste
08B7:0090  6D 33 32 5C 64 6F 73 78-0D 0D 6F 72 20 28 6C 6F  m32\dosx..or <lo
08B7:00A0  61 64 20 62 65 66 6F 72-65 20 64 6F 73 78 2E 65  ad before dosx.e
08B7:00B0  78 65 29 0D 69 6F 6E 27-73 20 50 49 46 2E 0D 73  xe).ion's PIF..s
08B7:00C0  73 20 61 0D 00 00 00 00-00 00 00 00 00 00 00 00  s a.....
08B7:00D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
08B7:00E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
08B7:00F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  .....
```

连字符

每个非打印字符



**n d cs:100 10F**

显示范围从 CS 段的 100h 到 10Fh 中所有字节的内容

08D0:0100 70 61 67 65 73 20 68 61-76 65 20 62 65 65 6E 20 pages have been

**n d [ds:]100 10F**

显示范围从 DS 段的 100h 到 10Fh 中所有字节的内容

08B7:0100 48 65 72 65 20 61 72 65-20 74 77 65 6E 74 79 2D Here are twenty-

**n d 100 l 20**

从 DS:100 开始显示 20h 个字节的内容

08B7:0100 48 65 72 65 20 61 72 65-20 74 77 65 6E 74 79 2D Here are twenty-

08B7:0110 73 69 78 20 65 6E 67 6C-69 67 68 20 6C 65 74 74 six english lett



## DEBUG -E (键入)

---

- n 将数据输入到内存中指定的地址。  
可以按十六进制或 **ASCII** 格式键入数据。以前存储在指定位置的任何数据全部丢失。

**e address [list]**

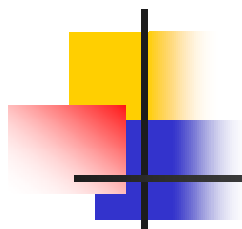
参数

**address**

指定输入数据的第一个内存位置。

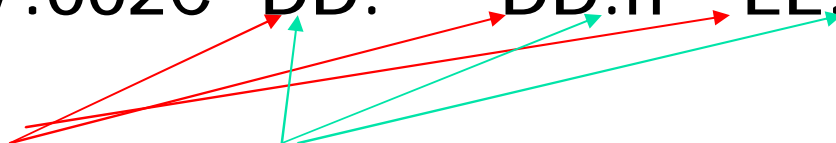
**list**

指定要输入到内存的连续字节中的数据。



List为空则进入一个字节一个字节输入过程

08B7:002C DD. DD.ff EE.



原内容 . 新输入

输入可以是：

SPACEBAR（空格键）——进入下一个字节

键入新值 ——更改该字节中的值(0~F 否则不回显)

HYPHEN 键 (-) ——返回到前一个字节

ENTER 键 ——停止执行 e 命令



## 使用 list 参数

如果指定 **list** 参数的值，随后的 **e** 命令将使用列表中的值替换现有的字节值。如果发生错误，将不更改任何字节值。

**List** 值可以是十六进制字节或字符串。使用空格、逗号或制表符来分隔值。必须将字符串包括在单或双引号中。

```
-e 2c 'Here are twenty-six english letters'
-d 2c
08B7:002C  48 65 72 65                                Here
08B7:0030  20 61 72 65 20 74 77 65-6E 74 79 2D 73 69 78 20  are twenty-six
08B7:0040  65 6E 67 6C 69 73 68 20-6C 65 74 74 65 72 73 00  english letters.
```



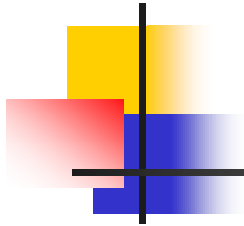
# Debug:F (填充)

---

- n 使用指定的值填充指定内存区域中的地址。  
可以指定十六进制或 **ASCII** 格式表示的数据。  
任何以前存储在指定位置的数据将会丢失。

## **-f range list**

- n 如果 **range** 包含的字节数比 **list** 中的数值大，  
**Debug** 将在 **list** 中反复指派值，直到 **range** 中的  
所有字节全部填充。如果 **list** 包含的数值多  
于 **range** 中的字节数，**Debug** 将忽略 **list** 中额  
外的值。



n -f 2c|40 48 65 72 65

n -d 2c

```
-f 2c|40 48 65 72 65
-d 2c
08B7:002C 48 65 72 65 Here
08B7:0030 48 65 72 65 48 65 72 65-48 65 72 65 48 65 72 65 HereHereHereHere
08B7:0040 48 65 72 65 48 65 72 65-48 65 72 65 48 65 72 65 HereHereHereHere
08B7:0050 48 65 72 65 48 65 72 65-48 65 72 65 48 65 72 65 HereHereHereHere
08B7:0060 48 65 72 65 48 65 72 65-48 65 72 65 00 20 20 20 HereHereHere.
```

n 从 ds:2c 之后40H的单元重复填充48 65 72 65这四个字节



# Debug:G (转向)

---

**n** 运行当前在内存中的程序。

**g [=address] [breakpoints]**

**参数 =address**

指定当前在内存中要开始执行的程序地址。如果不指定 **address**，将从 **CS:IP** 寄存器中的当前地址开始执行程序。

**Breakpoints** 指定可以设置为 **g** 命令的部分的 1 到 10 个临时断点。断点为指令地址





## Debug:H (十六进制)

- n 对指定的两个参数执行十六进制运算。

h value1 value2

参数 value1/2 代表从 0 到 FFFFh 范围内的任何十六进制数字。

- n Debug 首先将指定的两个参数相加，然后第一个参数中减去第二个参数。这些计算的结果显示在一行中：先计算和，然后计算差。



## Debug:I (端口输入)

---

**n** 从指定的端口读取并显示一个字节值。

**i port**

参数

**port**

按地址指定输入端口。地址可以是 **16** 位的值。



# Debug:L (从磁盘加载)

- n 将某个文件或特定磁盘扇区的内容加载到内存。
  - n *l* [address] , 文件名由n命令预先指定
- 要从磁盘文件加载 **BX:CX** 寄存器中指定的字节数内容到 **address**。如果是.exe文件, **Debug** 将文件重新定位到 .exe 文件的标题中指定的加载地址, 因此忽略 **address** 参数; 如果不带参数, 则缺省使用地址 **CS:100** 。 **Debug** 同时将 **BX** 和 **CX** 寄存器设置为加载的字节数
- n *l* address drive start number
- 略过 文件系统并直接加载特定的扇区
- n **drive** 指定包含读取指定扇区的磁盘的驱动器。该值是数值型: 0 = A, 1 = B, 2 = C 等。
  - n **Start** 指定要加载其内容的第一个扇区的十六进制数。
  - n **Number** 指定要加载其内容的连续扇区的十六进制数



# Debug:M (移动)

n 将一个内存块中的内容复制到另一个内存块中。

m range address

n 参数

**range** 指定要复制内容的内存区域的起始和结束地址，或起始地址和长度。

**Address** 指定要将 **range** 内容复制到该位置的起始地址。



## Debug:N (名称)

**n** n [drive:][path] filename

指定 Debug l (加载) 或 w (写入) 命令的可执行文件的名称

**n** n file-parameters

程序装载后, 要指定程序参数, 如同命令行输入参数

```
>prog param1 param2
```

```
>debug prog.com  
-nparam1 param2  
-g
```

```
>Debug  
-nprog.com  
-l  
-nparam1 param2  
-g
```



## Debug:O（端口输出）

---

n 将字节值发送到输出端口。

o port byte-value

参数

port

通过地址指定输出端口。端口地址可以是  
16 位值。

byte-value

指定要指向 port 的字节值。



## Debug:P (执行)

n p [= address] [number]

n 和g非常类似，只是运行中止由number（执行指令数）控制，除此之外，如果address处是“循环、重复字符串指令、软件中断或者完成了指定地址的子例程为止”也可将控制权交还debug。

n 举例：假定正在测试的程序在地址 CS:143F 处包含一个 call 指令。要运行 call 目标位置的子程序然后将控制返回到 Debug，请键入以下命令：

p=143f



## Debug:Q (退出debug)

- n 停止 Debug 会话，不保存当前测试的文件。  
键入 q 以后，控制返回到 DOS 的命令提示符。

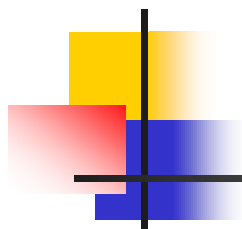
q  
该命令不带参数。





## Debug:R (寄存器)

- n 显示或改变一个或多个 CPU 寄存器的内容。  
r [register-name]
- n 如果在没有参数的情况下使用，则 r 命令显示所有寄存器的内容以及寄存器存储区域中的标志。
- n register-name  
指定要显示其内容的寄存器名。



n R

```
-r
AX=0000 BX=0000 CX=0100 DX=0000 SP=0028 BP=0000 SI=0000 DI=0000
DS=08B7 ES=08B7 SS=08CD CS=08D0 IP=0000  NU UP DI PL NZ NA PO NC
08D0:0000 B8C708          MOV     AX,08C7
```

n Rf 查看标志的状态

溢出	方向	中断	正负	零	辅助进位	奇偶校	验进位
ov	dn (减)	ei (启用)	ng (负)	zr	ac	pe (偶校验)	cy
nv	up (增)	di (禁用)	pl (正)	nz	na	po (奇校验)	nc

n -r ip

n IP 0000

n :100



# Debug:S (搜索)

- n 在某个地址范围搜索一个或多个字节值的模式。

s range list

- n 举例: -s f000:0 | 0 'AMIBIOS '

- n F000:F400

- n F000:F500

- n -d f000:f500

```
F000:F500  41 4D 49 42 49 4F 53 20-28 43 29 31 39 39 39 20  AMIBIOS (C)1999
F000:F510  41 6D 65 72 69 63 61 6E-20 4D 65 67 61 74 72 65  American Megatre
F000:F520  6E 64 73 20 49 6E 63 2E-2C 20 20 20 20 20 20 20  nds Inc.,
F000:F530  0D 0A 36 56 58 43 37 2D-34 58 20 2F 20 36 56 58  ..6UXC7-4X / 6UX
F000:F540  43 37 2D 34 58 2D 50 20-46 43 20 20 20 20 20 20  C7-4X-P FC
```



## Debug:T (跟踪)

- n 执行程序，但和g不同，它逐条执行指令并显示寄存器内容
- n -t [=address] [value]
- n 参数
- n =address 起始执行地址,忽略从当前CS:IP开始
- n Value 指示多少条指令被执行时，忽略等效为1，即单步运行

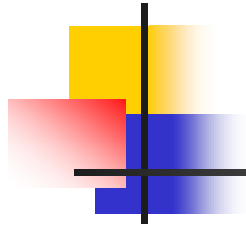


## Debug:U (反汇编)

- n 反汇编字节并显示相应的原语句，其中包括地址和字节值。反汇编代码看起来象已汇编文件的列表。

**u [range]**

如果在没有参数的情况下使用，则 **u** 命令分解 **20h** 字节（默认值），从前面 **u** 命令所显示地址后的第一个地址开始。



n -u 0 l 8

n 08D0:0000 B8C708          MOV    AX,08C7

n 08D0:0003 8ED8          MOV    DS,AX

n 08D0:0005 8EC0          MOV    ES,AX

n 08D0:0007 9A0000D108    CALL   08D1:0000

n -u 1 l 8

n 08D0:0001 C7088ED8      MOV    WORD PTR[BX+SI],D88E

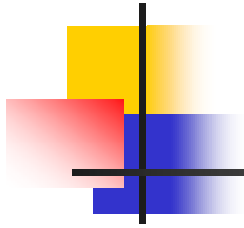
n 08D0:0005 8EC0          MOV    ES,AX

n 08D0:0007 9A0000D108    CALL   08D1:0000



# Debug:L (从磁盘加载)

- n 将内存内容写为某个文件或特定磁盘扇区。
- n w [address] , 文件名由n命令预先指定  
将在address起始, BX:CX 寄存器中指定字节数的内容写入磁盘文件。 .exe涉及重定位问题, 不能用该命令写入 .exe ; 如果不带参数, 则缺省使用地址 CS:100 。写前需设置 BX 和 CX 寄存器。
- n w address drive start number  
略过文件系统并直接写入特定的扇区。参数说明同l命令
- n 略过操作系统的文件处理机制, 写入特定的分区非常危险。如果键入错误的值, 则磁盘文件结构很容易被损坏。



---

- n 可视化的调试器

- n Borland Turbo Debugger

- n Microsoft CodeView



# Turbo Debugger 3.0 for Dos

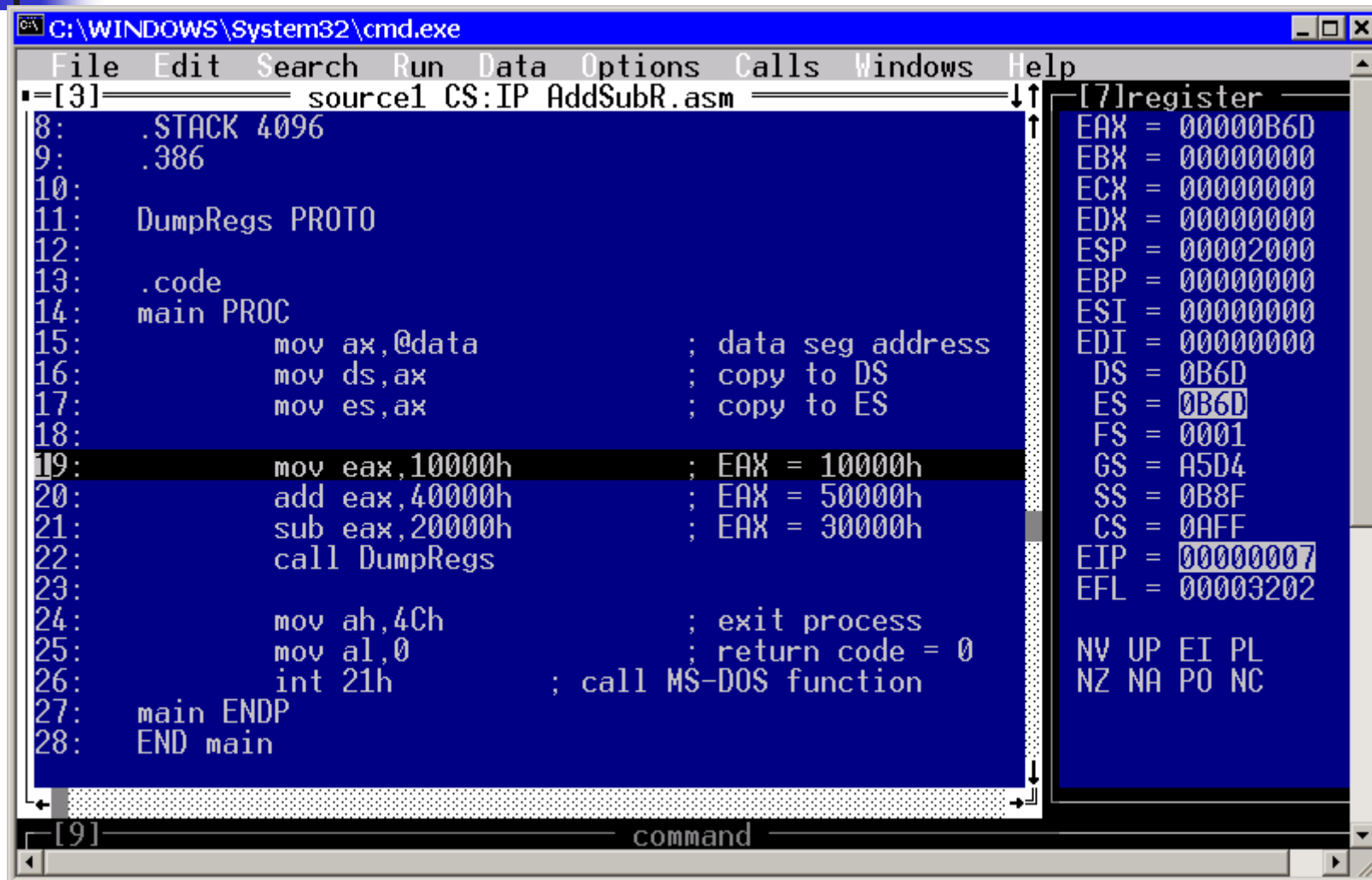
The screenshot displays the Turbo Debugger 3.0 for DOS interface. The menu bar at the top includes File, Edit, View, Run, Breakpoints, Data, Options, Window, and Help. The status bar at the bottom shows function key shortcuts: F1-Help, F2-Bkpt, F3-Mod, F4-Here, F5-Zoom, F6-Next, F7-Trace, F8-Step, F9-Run, and F10-Menu.

The main window is divided into several panes:

- Assembly Pane (Top Left):** Displays assembly code for a CPU Pentium. The current instruction is highlighted: `cs:0100 80740883 xor byte ptr [si+08],83`. Other instructions include `mov byte ptr [bp+si],E2`, `in al,F9`, `jmp 010C`, `clc`, `pop di`, `pop si`, `pop ds`, `pop cx`, `pop bx`, `pop ax`, `ret`, `push cx`, `push si`, and `push ds`.
- Registers Pane (Top Right):** Shows the current values of registers: `ax 0000`, `bx 0000`, `cx 0000`, `dx 0000`, `si 0000`, `di 0000`, `bp 0000`, `sp 0080`, `ds 29ED`, `es 29ED`, `ss 29ED`, `cs 29ED`, and `ip 0100`. On the far right, a column shows flags: `c=0`, `z=0`, `s=0`, `o=0`, `p=0`, `a=0`, `i=1`, and `d=0`.
- Memory Pane (Bottom Left):** Displays memory contents starting from `ds:0000`. The first few lines are: `ds:0000 CD 20 00 9E 00 9A F0 FE = R 0`, `ds:0008 1D F0 8D 1D 1A FE 0F 07`, `ds:0010 02 25 78 01 13 23 E5 24`, `ds:0018 01 01 01 00 02 08 FF FF`, and `ds:0020 FF FF FF FF FF FF FF FF`.
- Memory Pane (Bottom Right):** Displays memory contents starting from `ss:0082`. The first few lines are: `ss:0082 0000`, `ss:0080 0D00`, `ss:007E 0000`, `ss:007C 0000`, and `ss:007A 0000`.

The status bar at the top right indicates the debugger is in a **READY** state.

# CodeView



The screenshot shows the CodeView debugger window. The title bar indicates the file is `C:\WINDOWS\System32\cmd.exe`. The menu bar includes File, Edit, Search, Run, Data, Options, Calls, Windows, and Help. The main window is divided into three panes. The left pane shows the source code for `source1 CS:IP AddSubR.asm`. The middle pane shows the assembly code with comments. The right pane shows the register values. The bottom pane shows the command line.

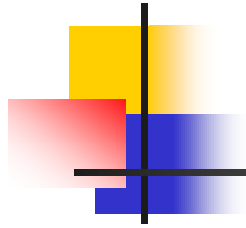
```
File Edit Search Run Data Options Calls Windows Help
[3] source1 CS:IP AddSubR.asm
8: .STACK 4096
9: .386
10:
11: DumpRegs PROTO
12:
13: .code
14: main PROC
15:     mov ax,@data           ; data seg address
16:     mov ds,ax              ; copy to DS
17:     mov es,ax              ; copy to ES
18:
19:     mov eax,10000h          ; EAX = 10000h
20:     add eax,40000h          ; EAX = 50000h
21:     sub eax,20000h          ; EAX = 30000h
22:     call DumpRegs
23:
24:     mov ah,4Ch              ; exit process
25:     mov al,0                ; return code = 0
26:     int 21h                 ; call MS-DOS function
27: main ENDP
28: END main
```

[7] register

EAX	=	00000B6D
EBX	=	00000000
ECX	=	00000000
EDX	=	00000000
ESP	=	00002000
EBP	=	00000000
ESI	=	00000000
EDI	=	00000000
DS	=	0B6D
ES	=	0B6D
FS	=	0001
GS	=	A5D4
SS	=	0B8F
CS	=	0AFF
EIP	=	00000007
EFL	=	00003202

NV UP EI PL  
NZ NA PO NC

[9] command



## n 其他常用调试工具

n Soft-ICE

n WinDbg

n Gdb