

## 2020/12/31\_16位汇编\_第6课\_masm工具集的使用、CPU数据传输类指令

笔记本： 16位汇编

创建时间： 2020/12/31 星期四 10:42

作者： ileemi

---

- [masm](#)
- [系统调用](#)
  - [功能号 \(DOS中断\)](#)
  - [BIOS中断](#)
- [CPU指令](#)
  - [数据传输类指令](#)
    - [传送指令MOV](#)
    - [交换指令 XCHG \(exchange\)](#)
    - [换码指令 XLAT \(translate\)](#)
  - [堆栈操作指令](#)
    - [PUSH 和 POP](#)
    - [标志寄存器传送指令](#)
    - [标志低字节进出AH指令](#)
  - [地址传送指令 LEA](#)
  - [指针传送指令 LDS](#)
  - [输入输出指令](#)

## masm

masm 是一个集成开发环境，其中 "edit.com" 为 dos年代 的汇编编写工具（有界面）。

汇编调试器（早年带界面的）："TD.exe"

TD 调试起快捷键：

F2：下断点

F9：运行

Ctrl + F：打开文件菜单

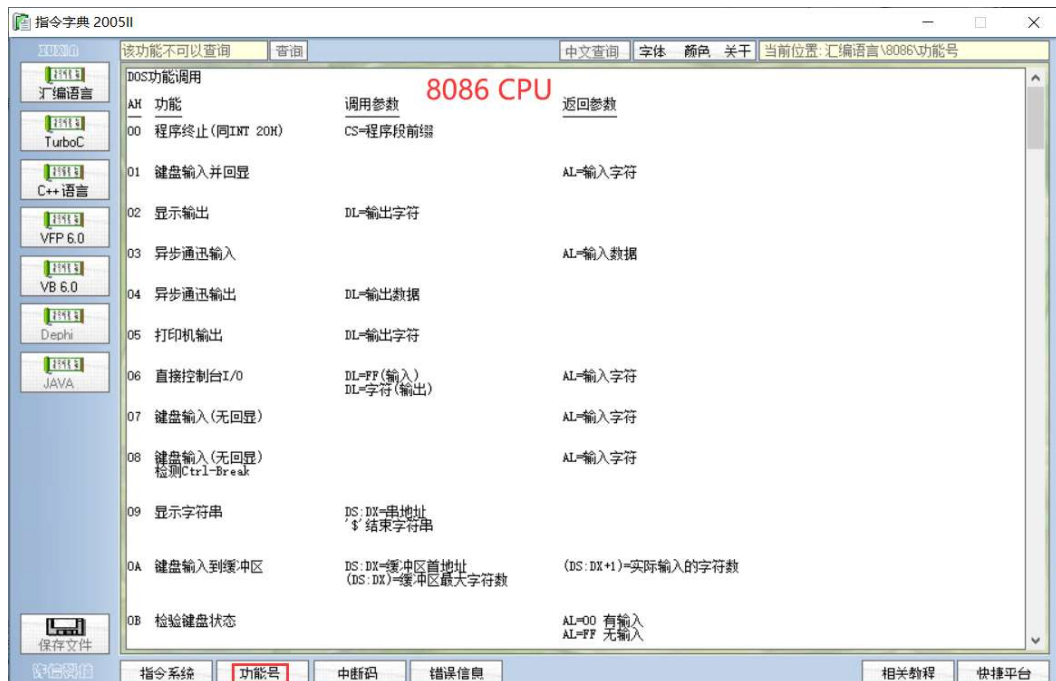
Ctrl + G：在内存窗口，可以查看知道内存地址的数据信息

修改内存数据："红色光标" 移动到指定数据上后，直接输入要修改后的数据即可

## 系统调用

### 功能号 (DOS中断)

Dos 系统提供的所有功能，通过使用对应的中断码，可以省去手动操作硬件的步骤（自己操作硬件就不需要调用中断码）。



mov ah,09h: 系统调用（显示字符串）  
int 21h: 中断

1: 文件操作

MyStack segment stack

db 256 dup(?)

MyStack ends

MyDatas segment

PATH db "C:\CR38\1.txt",0

MyDatas ends

MyCodes segment

MAIN:

```
mov ax,MyDatas ;获取数据段应该获取地址，不应该获取偏移
mov ds,ax
mov es,ax
```

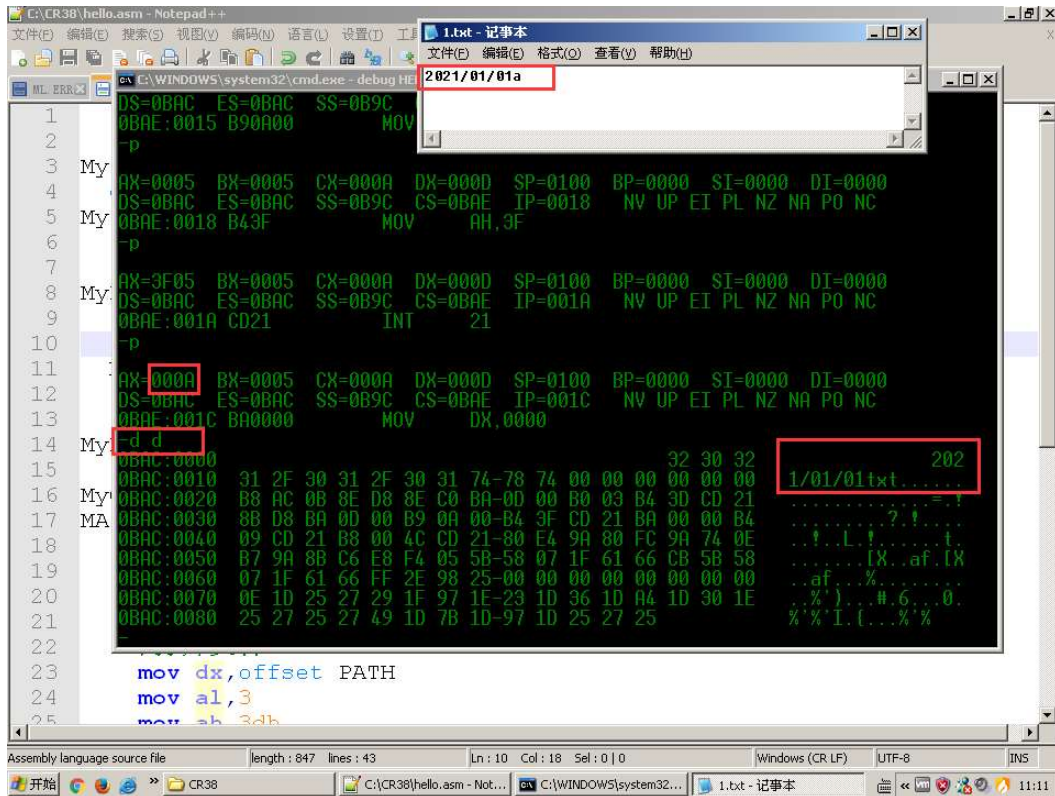
;打开文件

```
mov dx,offset PATH
mov al,3
mov ah,3dh
int 21h
```

;读取文件

```
mov bx,ax ;bx保存文件代号
mov dx,offset PATH ;DX=数据缓冲区地址
mov cx,10 ;读取文件字节数
mov ah,3fh
int 21h
;关闭文件
```

### 操作示例：



不需要操作系统，可以通过对应的中断码操作硬件。代码在内存中，不通过操作系统的情况下，需要将代码放置到磁盘的第一个扇区，BIOS会自动加载代码（类似于操作系统的启动）。BIOS中断，现在使用的不多，多使用 Dos中断。

系统启动后，BIOS中的代码会在内存中驻留，所以这个时候调用BIOS中断也会有对应的效果

```
1: 调整dos窗口分辨率
mov ah;0
mov al,12h ;设置显示方式 (调整dos窗口分辨率)
int 10 ;中断码
```

2: 输入字符

mov ah,01

int 21h

Dos中断码:

ah = 06: 直接控制台I/O

DL=FF(输入)、AL=输入字符: 表示从键盘缓冲区中获取一个字符, 存放到AL中。

DL=字符(输出): 输入的字符同时可以进行输出

## CPU指令

32位汇编最高支持到80686, 80686之后就是64位CPU。

**mov 指令的语法格式**

**mov 指令可以影响到的标志寄存器**

**介绍**

**操作码**

Operands	8086	286	386	486	Size Bytes
reg, reg	2	2	2	1	2
mem, reg	9+EA	3	2	1	2-4 (W88-13+EA)
reg, mem	8+EA	5	4	1	2-4 (W88-12+EA)
mem, imm8	10+EA	3	2	1	3-6 (W88-14+EA)
reg, imm8	4	2	2	1	2-3
mem, accum	10	3	2	1	3 (W88-14)
accum, mem	10	5	4	1	3 (W88-14)
segreg, reg16	2	2	2	3	2
segreg, mem16	8+EA	5	5	9	2-4 (W88-12+EA)
reg16, segreg	2	2	2	3	2
mem16, segreg	9+EA	3	2	3	2-4 (W88-13+EA)
reg32, CR0/CR2/CR3	-	-	-	6	4
CR0, reg32	-	-	10	16	3
CR2, reg32	-	-	4	4	3
CR3, reg32	-	-	5	4	3
reg32, DR0/DR1/DR2/DR3	-	-	22	10	3
reg32, DR6/DR7	-	-	22	10	3
DR0/DR1/DR2/DR3, reg32	-	-	22	11	3
DR6/DR7, reg32	-	-	16	11	3
reg32, TR6/TR7	-	-	12	4	3
TR6/TR7, reg32	-	-	12	4	3
reg32, TR3	-	-	-	3	3
TR3, reg32	-	-	-	6	3

**CPU每条指令的运算速度, 单位为: Clocks (指令周期)**

**操作码对应的的语法:**

- r8: 8位寄存器
- r16: 16位寄存器
- m8: 访问内存中的8位数据 (byte ptr [xxx])
- m16: word ptr [xxx]

**mov r8,r8**

**mov m8,r8**

8E / r MOV Sreg,r/m16\*\* Move r/m16 to segment register

immed: 立即数 (常量)

Sreg: 段寄存器

accum: 累加器 (ax寄存器), mov [bx],ax 要比 mov [bx],cx 快一两个指令周期。

在 8086 CPU 中, 将数据传输到内存时, 尽量将数据放到累加器ax 中 (传输速度相对其它寄存器传输的速度要快)。

## 数据传输类指令

1. 数据传送是计算机中最基本、最重要的一种操作
2. 传送指令也是最常使用的一类指令
3. 传送指令把数据从一个位置传送到另一个位置
4. 除标志寄存器传送指令外, 均不影响标志位

重点掌握: MOV XCHG XLAT PUSH POP LEA

## 传送指令MOV

把一个字节或字的操作数从源地址传送至目的地址。

语法：

- MOV reg/mem,imm ;立即数送寄存器或内存
- MOV reg/mem/seg,reg ;寄存器送（段）寄存器或内存
- MOV reg/seg,mem ;内存送（段）寄存器
- MOV reg/mem,seg ;段寄存器送寄存器或内存

非法传送：

- 两个操作数的类型不一致
- 两个操作数不能都是存储器
- 不允许立即数传送给段寄存器：MOV DS,100H
- 不允许直接改变CS值：MOV CS,[SI]
- 不允许段寄存器之间的直接数据传送：MOV DS,ES

## 交换指令XCHG (exchange)

把两个地方的数据进行互换：XCHG reg,reg/mem

代码示例：

```
mov ax,1  
mov bx,2  
xchg ax,bx
```

## 换码指令XLAT (translate)

将BX指定的缓冲区中、AL指定的位移处的一个字节数据取出赋给AL。类似于数组查询下标，数组首地址对应bx，下标对应al，之后再将目标存放到al中。

- 换码指令执行前：  
在主存建立一个字节量表格，内含要转换成的目的代码，表格首地址存放于BX，AL存放相对表格首地址的位移量。
- 换码指令执行后：  
将AL寄存器的内容转换为目标代码。

使用时，XLAT即可，其等价：MOV AL,[BX+AL]

代码示例：

```
MyStack segment stack  
    db 256 dup(?)  
MyStack ends  
MyDatas segment  
    TABLE1 db 03,04,05,06  
MyDatas ends  
MyCodes segment
```

MAIN:

```
mov ax,MyDatas ;获取数据段应该获取地址，不应该获取偏移
mov ds,ax
mov es,ax
```

;两数交换

```
mov ax,1
mov bx,2
xchg ax,bx
```

;换码指令 查表

```
mov bx,offset TABLE1
mov al,1
xlat
```

```
mov ax,4c00h ;强制退出进程，不需要在使用ret
int 21h
```

MyCodes ends

end MAIN

## 堆栈操作指令

1. 堆栈是一个“后进先出FILO”（或说“先进后出FILO”）的主存区域，位于堆栈段中；
2. SS段寄存器记录其段地址，堆栈只有一个出口，即当前栈顶；
3. 用堆栈指针寄存器SP指定，栈顶是地址较小的一端（低端），栈底不变。

没有堆栈前，16位汇编中，8个通用寄存器使用比较频繁，其中保存的数据在下次使用后就不存在了，所以就需要将使用前寄存器中保存的数据进行保存。可行操作如下：

MyStack segment stack

```
db 256 dup(?) ;申请的最大堆栈空间
```

MyStack ends

MyDatas segment

```
;db 256 dup(11h)
```

```
TABLE1 db 03,04,05,06
```

;等价于全局变量

```
NUM1 dw 0ffffh ;用于临时存储通用寄存器中的数据，内存初始化为0
```

```
;org 512 ;在 Hello World! 后面申请512字节，默认初始化为0
```

MyDatas ends

MyCodes segment

MAIN:

```
mov ax,MyDatas ;获取数据段应该获取地址，不应该获取偏移
mov ds,ax
mov es,ax
```

```

;两数交换
mov ax,1
mov bx,2
xchg ax,bx

;保存数据,存在临时空间不适用无法释放问题
;mov word ptr ds:[NUM1],bx
;手动申请堆栈空间
sub sp,2 ;抬栈
mov bp,sp
mov [bp],bx

;换码指令 查表
mov bx,offset TABLE1
mov al,1
xlat

;还原数据
;mov bx,word ptr ds:[NUM1]

;手动释放堆栈空间
mov bx,[bp]
add sp,2 ;不释放堆栈会溢出,时刻保持堆栈平衡
mov ax,4c00h ;强制退出进程,不需要在使用ret
int 21h
MyCodes ends
end MAIN

```

## PUSH 和 POP

存放临时数据, 栈两个字节对齐。push bl -->不可取

push、pop 指令直接操作 sp, 不操作bp。bp 保存某一时刻的栈地址（可以理解为栈底）。堆栈中没有空间的情况下, 继续push 数据可能会将操作系统的代码进行覆盖。

不能 PUSH 立即数, 如果需要 PUSH 立即数, 需要使用寄存器进行中转一下。

汇编代码示例:

```

MyCodes segment
MAIN:
    mov ax,MyDatas ;获取数据段应该获取地址, 不应该获取偏移
    mov ds,ax
    mov es,ax

;两数交换
mov ax,1

```

```

mov bx,2
xchg ax,bx

;保存数据,存在临时空间不适用无法释放问题
;mov word ptr ds:[NUM1],bx
;手动申请堆栈空间
;sub sp,2 ;抬栈
;mov bp,sp
;mov [bp],bx

;使用push指令, 等价上面三行指令
push bx

;换码指令 查表
mov bx,offset TABLE1
mov al,1
xlat

;还原数据
;mov bx,word ptr ds:[NUM1]

;手动释放堆栈空间
;mov bx,[bp]
;add sp,2 ;不释放堆栈会溢出, 时刻保持堆栈平衡

; 还原数据, 等价上面两行指令, 会自动调整堆栈
pop bx
mov ax,4c00h ;强制退出进程, 不需要在使用ret
int 21h
MyCodes ends
end MAIN

```

## 标志寄存器传送指令

当前面的汇编指令执行后, 将对应的标志寄存器数值进行的改变, 可能会影响到后面指令的执行。可以将对应的标志寄存器先进行保存(可以将其保存到堆栈中)。单独操作标志寄存器需要使用: **PUSHF**、**POPF**。

- **PUSHF**: PUSHF指令将标志寄存器的内容压入堆栈, 同时栈顶指针SP减2
- **POPF**: POPF指令将栈顶字单元内容送标志寄存器, 同时栈顶指针SP加2

## 标志低字节进出AH指令

- **LAHF**: 将标志寄存器的低字节送寄存器AH (AH<--FLAGS的低字节), SF/ZF/AF/PF/CF状态标志位分别送入AH的第7/6/4/2/0位, 而AH的第5/3/1位任意。



- **SAHF**: SAHF将AH寄存器内容送FLAGS的低字节（FLAGS的低字节<--AH），用AH的第7/6/4/2/0位相应设置SF/ZF/AF/ PF/CF标志。

修改所有标志寄存器数据:

pushf ;保存标志寄存器

popf ;恢复

lahf

sahf

mov ax,0ffffh

push ax

popf ;从堆栈中拿两个字节数据给对应的标志寄存器

```

C:\WINDOWS\system32\cmd.exe - debug TEST.EXE
0B9C:0160  B4 09 CD 21 B8 00 4C CD-21 00 00 00 00 00 00 00 00  ...!..L!.....
0B9C:0170  0E 1D 25 27 29 1F 97 1E-23 1D 36 1D                ..%')...#.6.
-p
AX=FFFF BX=0001 CX=0169 DX=0000 SP=00FC BP=00FE SI=0000 DI=0000
DS=0BAC ES=0BAC SS=0B9C CS=0BAF IP=001E  NV UP EI PL NZ AC PO NC
0BAF:001E 50                PUSH    AX
-d ss:fc
0B9C:00F0                01 00 01 00                ....Hello World!
0B9C:0100  03 04 05 06 48 65 6C 6C-6F 20 57 6F 72 6C 64 21      $C:\CR38\1.txt..
0B9C:0110  24 43 3A 5C 43 52 33 38-5C 31 2E 74 78 74 00 FF      .....
0B9C:0120  FF 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00      .....
0B9C:0130  B8 AC 0B 8E D8 8E C0 B8-01 00 BB 02 00 93 83 EC      .....^..S.....P.
0B9C:0140  02 8B EC 89 5E 00 53 9C-9D 9F 9E B8 FF FF 50 9D      .....^.....I.
0B9C:0150  BB 00 00 B0 01 D7 8B 5E-00 83 C4 02 5B BA 04 00      .....
0B9C:0160  B4 09 CD 21 B8 00 4C CD-21 00 00 00 00 00 00 00      ...!..L!.....
0B9C:0170  0E 1D 25 27 29 1F 97 1E-23 1D 36 1D                ..%')...#.6.
-p
AX=FFFF BX=0001 CX=0169 DX=0000 SP=00FA BP=00FE SI=0000 DI=0000
DS=0BAC ES=0BAC SS=0B9C CS=0BAF IP=001F  NV UP EI PL NZ AC PO NC
0BAF:001F 9D                POPF
-p
AX=FFFF BX=0000 CX=0169 DX=0000 SP=00FC BP=00FE SI=0000 DI=0000
DS=0BAC ES=0BAC SS=0B9C CS=0BAF IP=0023  OV DN EI NG ZR AC PE CY
0BAF:0023 B001                MOV     AL,01

```

## 地址传送指令 LEA

地址传送指令将存储器单元的逻辑地址送至指定的寄存器。注意：不是获取存储器单元的内容。lea 可在一定程度上代替 add（指令周期要比 add快）。

代码示例:

MyStack segment stack

db 256 dup(?) ;申请的最大堆栈空间

MyStack ends

MyDatas segment

TABLE1 db 03,04,05,06

MyDatas ends

MyCodes segment

MAIN:

mov ax,MyDatas ;获取数据段应该获取地址，不应该获取偏移

mov ds,ax

mov es,ax

;取内容



MyCodes segment

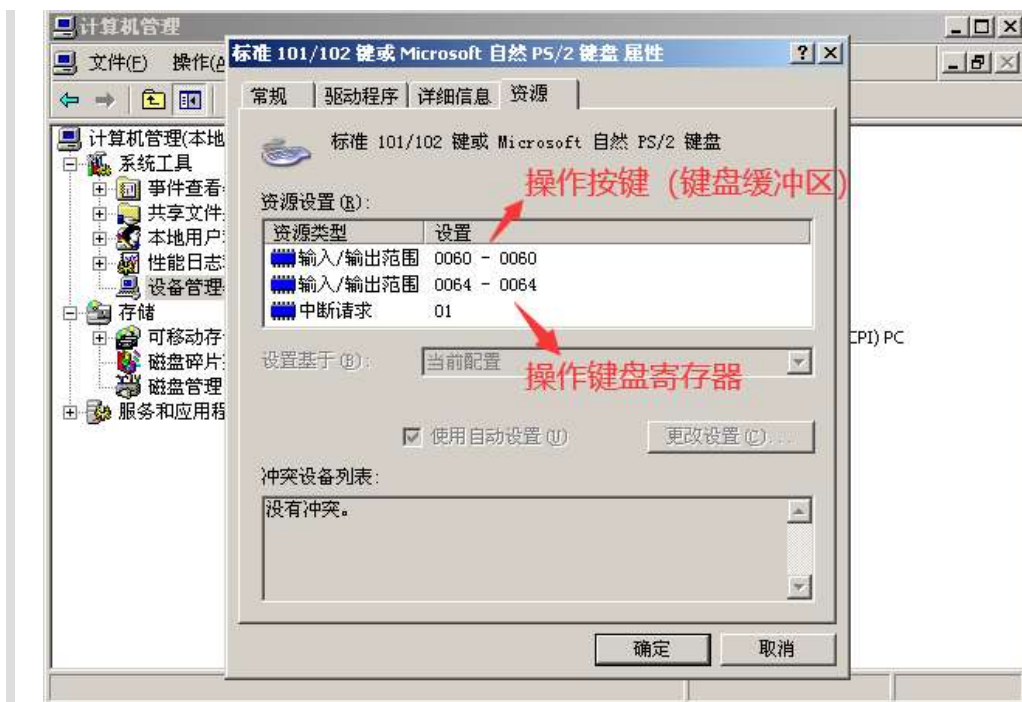
MAIN:

```
mov ax,MyDatas ;获取数据段应该获取地址，不应该获取偏移
mov ds,ax
mov es,ax
;指针传送指令
mov ax,MyDatas2
push ax ;压入段地址
mov ax,0
push ax ;压入偏移
mov bp,sp
lds dx,[bp] ;切换数据段，修改ds
les dx,[bp] ;修改es
add sp,4 ;释放堆栈
;
mov ax,bx
mov ax,4c00h ;强制退出进程，不需要在使用ret
int 21h
MyCodes ends
end MAIN
```

```
C:\WINDOWS\system32\cmd.exe - debug TEST.EXE
DS=0BAC ES=0BAC SS=0B9C CS=0BB0 IP=000F NV UP EI PL NZ NA PO NC
0BB0:000F 8BEC MOV BP,SP
-p
AX=0000 BX=0000 CX=019A DX=0000 SP=00FC BP=00FC SI=0000 DI=0000
DS=0BAC ES=0BAC SS=0B9C CS=0BB0 IP=0011 NV UP EI PL NZ NA PO NC
0BB0:0011 C55600 LDS DX,[BP+00] SS:00FC=0000
-d ss:fc
0B9C:00F0
0B9C:0100 03 04 05 06 48 65 6C 6C-6F 20 57 6F 72 6C 64 21 ....Hello World!
0B9C:0110 24 43 3A 5C 43 52 33 38-5C 31 2E 74 78 74 00 FF $C:\CR38\1.txt..
0B9C:0120 FF 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0B9C:0130 12 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
0B9C:0140 B8 AC 0B 8E D8 8E C0 B8-AE 0B 80 B8 00 00 50 8B .....P...P.
0B9C:0150 EC C5 56 00 C4 56 00 83-C4 04 B8 01 00 BB 02 00 ..V..V.....
0B9C:0160 93 BB 00 00 8B 47 02 8D-47 02 83 C3 08 8B C3 83 .....6..6.....
0B9C:0170 EC 02 8B EC 83 5E 00 53-9C 9D 9F 9E .....^..S....
-p
AX=0000 BX=0000 CX=019A DX=0000 SP=00FC BP=00FC SI=0000 DI=0000
DS=0BAF ES=0BAC SS=0B9C CS=0BB0 IP=0014 NV UP EI PL NZ NA PO NC
0BB0:0014 C45600 LES DX,[BP+00] SS:00FC=0000
-p
AX=0000 BX=0000 CX=019A DX=0000 SP=00FC BP=00FC SI=0000 DI=0000
DS=0BAF ES=0BAF SS=0B9C CS=0BB0 IP=0017 NV UP EI PL NZ NA PO NC
0BB0:0017 83C404 ADD SP,+04
```

## 输入输出指令

in（从硬件中读取数据）、out（将数据输入到硬件中）。需要给每个硬件设定一个端口号，如下图所示：



操作硬件（键盘，鼠标，显示器），使用 in、out 需要对硬件有一定的了解程度。