

汇编语言讲解（二）

查看cpu和内存，用机器指令和汇编指令编程

预备知识：Debug的使用

之后的讲解中都会使用到Debug程序，首先学习一下他的主要用法

- 什么时Debug?

Debug是Dos、Windows都提供的实模式程序的调整工具，使用它，可以查看cpu各种寄存器中的内容、内存的情况和机器码级跟踪程序的运行

- 我们用到的Debug功能

- 1、用Debug的R命令查看、改变cpu寄存器的内容
- 2、用Debug的D命令查看内存中的内容
- 3、用Debug的E命令改写内存中的内容
- 4、用Debug的U命令将内存中的机器指令翻译成汇编指令
- 5、用Debug的T命令执行一个机器指令
- 6、用Debug的A命令以汇编指令的格式在内存中写入一条机器指令

注意：不区分大小写

- **进入Debug**：Debug是DOS方式下使用的程序，我们在进入Debug前，应先进入到DOS方式。用以下方式可以进入DOS，重启计算机，进入DOS方式属于实模式的DOS；在windows中进入DOS方式，此时进入的虚DOS。以后我们可以使用Debug模式来编写汇编语言。

- 1、安装DOSBox0.74-win32-installer
- 2、在D盘建立名为“debug”的文件夹
- 3、运行桌面的“DOSBox 0.74”

此时会弹出两个窗口，在“dosbox0.74窗口里面输入如下命令”

```
mount c: d:\debug
```

然后输入命令

```
c:
```

最后输入debug进入汇编模式。

```

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=138C ES=138C SS=138C CS=138C IP=0100 NU UP EI PL NZ NA PO NC
138C:0100 0000 ADD [BX+SI],AL DS:0000=CD
-r ax
AX 0000
:1111
-r
AX=1111 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=138C ES=138C SS=138C CS=138C IP=0100 NU UP EI PL NZ NA PO NC
138C:0100 0000 ADD [BX+SI],AL DS:0000=CD
-r cs
CS 138C
:1400
-r ip
IP 0100
:
-r ip
IP 0100
:1111
-r
AX=1111 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=138C ES=138C SS=138C CS=1400 IP=1111 NU UP EI PL NZ NA PO NC
1400:1111 0000 ADD [BX+SI],AL DS:0000=CD

```

实验1

使用Debug将上面的程序段写入内存，逐行执行，观察每条指令执行后，cpu中相关寄存器中内容的变化。

机器码	汇编指令
b8 20 4e	mov ax,4E20H
05 16 14	add ax,1416H
bb 00 20	mov bx,2000H
01 d8	add ax,bx
89 c3	mov bx,ax
01 d8	add ax,bx
b8 1a 00	mov ax,001AH
bb 26 00	mov bx,0026H
00 d8	add al,b1
00 dc	add ah,b1
00 c7	add bh,al
b4 00	mov ah,0
00 d8	add al,b1
04 9c	add al,9CH

提示，可用E命令和A命令以两种方式将指令写入内存，注意用T命令执行时CS:IP的指向。

- 步骤1：使用a命令输入命令：

```

-a
138C:0100 mov ax, 4e20
138C:0103 add ax, 1416
138C:0106 mov bx, 2000
138C:0109 add ax, bx
138C:010B mov bx, ax
138C:010D add ax, bx
138C:010F mov ax, 001A
138C:0112 mov bx, 0026
138C:0115 add al, bl
138C:0117 add ah, bl
138C:0119 add bh, al
138C:011B mov ah, 0
138C:011D add al, bl
138C:011F add al, 9c
138C:0121

```



- 步骤2: 使用-d来查看内存数据, 我们可以使用-d指点要查看的数据

```

-d 073F:0100
073F:0100 B8 20 4E 05 16 14 BB 00-20 01 D8 89 C3 01 D8 BB . N.....
073F:0110 1A 00 BB 26 00 00 D8 00-DC 00 C7 B4 00 00 D8 04 ...&.....
073F:0120 9C 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
073F:0130 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 .....
073F:0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
073F:0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
073F:0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
073F:0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
-u 073F:0100
073F:0100 B8204E      MOV     AX,4E20
073F:0103 051614      ADD     AX,1416
073F:0106 BB0020      MOV     BX,2000
073F:0109 01D8      ADD     AX,BX
073F:010B 89C3      MOV     BX,AX
073F:010D 01D8      ADD     AX,BX
073F:010F B81A00      MOV     AX,001A
073F:0112 BB2600      MOV     BX,0026
073F:0115 00D8      ADD     AL,BL
073F:0117 00DC      ADD     AH,BL
073F:0119 00C7      ADD     BH,AL
073F:011B B400      MOV     AH,00
073F:011D 00D8      ADD     AL,BL
073F:011F 049C      ADD     AL,9C

```

- 步骤3: 上面的看到的073F出现的是机器码, 我们可以使用u来查看汇编格式

```

-u 073F:0100
073F:0100 B8204E      MOV     AX,4E20
073F:0103 051614      ADD     AX,1416
073F:0106 BB0020      MOV     BX,2000
073F:0109 01D8      ADD     AX,BX
073F:010B 89C3      MOV     BX,AX
073F:010D 01D8      ADD     AX,BX
073F:010F B81A00      MOV     AX,001A
073F:0112 BB2600      MOV     BX,0026
073F:0115 00D8      ADD     AL,BL
073F:0117 00DC      ADD     AH,BL
073F:0119 00C7      ADD     BH,AL
073F:011B B400      MOV     AH,00
073F:011D 00D8      ADD     AL,BL
073F:011F 049C      ADD     AL,9C

```

- 步骤4：下面我们开始准备执行命令了，首先我们要改变CS和IP的值才行

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0102  NU UP EI NG NZ NA PO NC
073F:0102 4E          DEC      SI
-r cs
CS 073F
:073F
-r ip
IP 0102          调整cpu位置
:100
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI NG NZ NA PO NC
073F:0100 B8204E      MOV     AX,4E20
- ▲
```

- 步骤5：我们使用t来运算单条机器指令，我们可以观察寄存器的变化

```
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI NG NZ NA PO NC
073F:0100 B8204E      MOV     AX,4E20
-t
AX=4E20 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0103  NU UP EI NG NZ NA PO NC
073F:0103 051614      ADD     AX,1416
-t
AX=6236 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0106  NU UP EI PL NZ NA PE NC
073F:0106 B80020      MOV     BX,2000
-t
AX=6236 BX=2000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0109  NU UP EI PL NZ NA PE NC
073F:0109 01D8      ADD     AX,BX
-t
AX=8236 BX=2000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=010B  OV UP EI NG NZ NA PE NC
073F:010B 89C3      MOV     BX,AX
- ▲
```

使用quit来结束所有的命令。

实验2

将下面3条指令写入从2000: 0开始的内存单元中，利用这3条指令计算2的8次方。

- 先写入3条指令：

```

C:\>debug
-a 2000:0
2000:0000 mov ax,1
2000:0003 add ax,ax
2000:0005 jmp 2000:3
2000:0007
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=073F IP=0100  NU UP EI PL NZ NA PO NC
073F:0100 B8204E          MOV     AX,4E20
-r cs
CS 073F
:2000
-r ip
IP 0100
:0
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0000  NU UP EI PL NZ NA PO NC
2000:0000 B80100          MOV     AX,0001

```

- 使用t命令执行代码

```

-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0000  NU UP EI PL NZ NA PO NC
2000:0000 B80100          MOV     AX,0001
-t
AX=0001 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003
-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0003  NU UP EI PL NZ NA PO NC
2000:0003 01C0          ADD     AX,AX
-t
AX=0004 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0005  NU UP EI PL NZ NA PO NC
2000:0005 EBFC          JMP     0003

```

实验3

查看内存中的内容：pc机主板的ROM中写有一个生产日期，在内存FFF0H~FFFFFH的某几个单元中，请找到这个生产日期并试图修改

```

-r cs
CS 2000
:ffff
^ Error
-r cs
CS 2000
:fff0
-r ip
IP 0005
:0
-d fff0:0
FFF0:0000 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0010 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0020 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0030 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0040 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0050 00 00 00 CF 00 50 B0 20-E6 20 58 CF 00 00 00 00 .....P. . X.....
FFF0:0060 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0070 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....

```

指定范围：（尝试使用e修改，失败）

```

-d fff0:00 ff
FFF0:0000 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0010 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0020 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0030 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0040 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0050 00 00 00 CF 00 50 B0 20-E6 20 58 CF 00 00 00 00 .....P. . X.....
FFF0:0060 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0070 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0080 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:0090 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00A0 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00B0 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00C0 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00D0 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00E0 00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
FFF0:00F0 EA C0 12 00 F0 30 31 2F-30 31 2F 39 32 00 FC 55 .....01/01/92..U

```

实验4

尝试输入下列命令，观察屏幕发生什么变化。

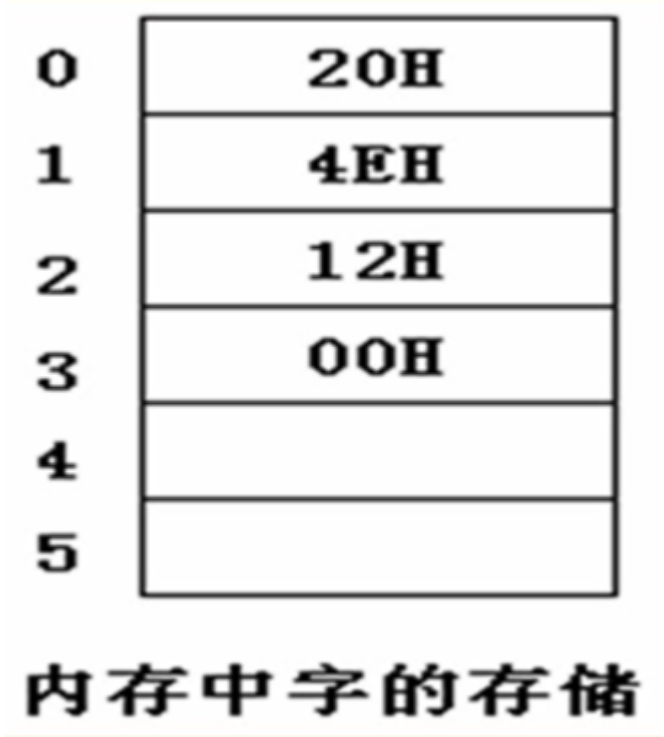
```
FFF0:0FB0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 00 .....
FFF0:0FC0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0FD0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0FE0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0FF0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
-d fff0:00 ff
FFF0:0000  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0010  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0020  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0030  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0040  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0050  00 00 00 CF 00 50 B0 20-E6 20 58 CF 00 00 00 00 .....P. . X.....
FFF0:0060  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0070  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0080  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:0090  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00A0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00B0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00C0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00D0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00E0  00 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00 .....
FFF0:00F0  EA C0 12 00 F0 30 31 2F-30 31 2F 39 32 00 FC 55 .....01/01/92..U
-e b810:0 01 01 02 02 03 03 04 04
-e b810:0 34 43 32 12 12
```

可以看到屏幕中的输出发生了变化，这是因为b810段是显存，我们往里面写什么，就显示什么。

CPU和内存访问

内存中字的存储

在0地址处开始存放20000（4E20H），记住一句话：**高位对应高地址**，如：0号单元是低地址单元，1号单元是高地址单元



根据上图回答下列问题：（注意：字型数据就要读取前后两个数据，字节型表示的是单个数据）

- 1、0地址单元存放的字节型数据是什么？ 答：20H, 32
- 2、0地址字单元中存放的字型数据是多少？ 答：4E20H, 20000
- 3、2地址字单元中存放的字单元中存放的字节型数据是什么？ 答：12H
- 4、2地址单元中存放的字型数据是什么？ 答：0012H
- 5、1地址单元中存放的字型数据是什么？ 答：124EH

结论： 任意两个地址连续的内存单元，N号单元和N+1号单元，可以将他们看成两个内存单元，也可以看成一个地址为N的字单元中的高位字节单元和低位字节单元；

比如上图：1号和2号，可以看成两个内存单元；也可以看成是1地址字单元的高位字节2，1表示其低位字节。

DS和[address]

cpu要读取一个内存单元的时候，必须先给出这个内存单元的地址；在8086pc中，内存地址由段地址和偏移地址组成。8086cpu有一个DS寄存器，通常用来存放要访问的数据的**段地址**。

例如：我们要读取10000H单元的内容可以用如下程序段进行：（**将内存单元的数据读到寄存器中**）

```
mov bx 1000H # 这是因为它是段地址，在经过地址加法器的时候会乘以16
mov ds bx
mov al, [0] # 下面我们会讲解mov的高级用法

# 上面三条指令将10000H(1000:0)中的数据读到al中。
```

注意： 上面的mov al,[0],已知的mov指令可完成的两种传递功能：

- 将数据直接送入寄存器；如 mov ax, 12H
- 将一个寄存器中的内容送入另一个寄存器中；如 mov bx, ax
- 除此之外，mov指令还可以将一个内存单元中的内容送入一个寄存器；如：mov al, [0], 首先我们通过将段地址输入到ds中，然后通过mov al,[0],这里的[0],表示的是偏移地址，将这个偏移地址的数据读取到al中。

从哪个内存单元送到哪个寄存器中那？

mov指令的格式： mov 寄存器名, 内存单元地址

"[...]" 表示一个内存单元，"[...]"中的0表示内存单元的偏移地址。那么内存单元的段地址是多少？

执行指令时，8086cpu自动取DS中的数据为内存单元的段地址。

如何用mov指令从10000H中读取数据？

- 10000H表示为1000:0(段地址:偏移地址)
- 将段地址1000H放入ds
- 用mov al, [0]完成传送（mov 指令中的[]说明操作对象是一个内存单元，[]中的0说明这个内存单元的偏移地址是0，它的段地址默认放在ds中）

能否用 `mov ds,1000H` 这语句，直接将段地址输入到 `ds`？

答：不能，上面的代码演示中是通过 `bx`，间接的将数据给 `ds` 了！原因是 8086cpu 不支持将数据送入段寄存器的操作，`ds` 是段寄存器。但支持将数据输入通用寄存器中，通用寄存器可以将数据给段寄存器。

语句：数据 --> 通用寄存器 --> 段寄存器

问题：写几条指令，将 `al` 的数据送入内存单元 `10000H`？怎样将数据从寄存器送入内存单元？（将寄存器的数据送到内存单元中）

```
1、mov bx, 1000H
2、mov ds,bx
3、mov [0],al # 和之前的相反，因为段地址只能读通用寄存器，所以需要将段地址写道bx中，给ds后，[0]表示偏移地址
```

字的传送

因为 8086cpu 是 16 位结构，有 16 根数据线，所以可以一次性传送 16 位的数据，也就是一次性传送一个字：

```
mov bx,1000H
mov ds,bx
mov ax,[0] ;1000:0处的字型数据送入ax
mov [0],cx ;cx中的16位数据送到1000:0处
```

注意：传送的数据，如果是 16 位的，需要 `ax` 接受，如果是 `ah`、`al` 则接受 8 位数据

例子：内存中的情况如下，写出下面指令执行后寄存器 `ax`, `bx`, `cx` 的值

```

mov ax,1000H
mov ds,ax
mov ax,[0]
mov bx,[2]
mov cx,[1]
add bx,[1]
add cx,[2]

```

10000H	23
10001H	11
10002H	22
10003H	66

内存情况示意

解答步骤：

- 步骤一：将数据写入内存中

```

-a
2000:0012 mov ax,17
2000:0015 mov bx,1000H
                ^ Error
2000:0015 mov bx,1000
2000:0018 mov ds,bx
2000:001A mov [0],ax
2000:001D
-r
AX=0004 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=FFFD IP=0000  NU UP EI PL NZ NA PO NC
FFFD:0000 0000          ADD     [BX+SI],AL          DS:0000=CD
-r cs
CS FFFD
:2000
-r ip
IP 0000
:0012
-r
AX=0004 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0012  NU UP EI PL NZ NA PO NC
2000:0012 B81700          MOV     AX,0017

```

- 步骤二：切换CS、IP，执行语句

```

-r
AX=0004 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0012  NU UP EI PL NZ NA PO NC
2000:0012 B81700      MOV     AX,0017
-t

AX=0017 BX=0000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0015  NU UP EI PL NZ NA PO NC
2000:0015 BB0010      MOV     BX,1000
-t

AX=0017 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=073F ES=073F SS=073F CS=2000 IP=0018  NU UP EI PL NZ NA PO NC
2000:0018 8EDB      MOV     DS,BX
-t

AX=0017 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=2000 IP=001A  NU UP EI PL NZ NA PO NC
2000:001A A30000      MOV     [0000],AX          DS:0000=0000
-t

AX=0017 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=2000 IP=001D  NU UP EI PL NZ NA PO NC
2000:001D 0000      ADD     [BX+SI],AL          DS:1000=00
- ▲

```

- 步骤三：

```

-d 1000:0
1000:0000 17 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-e 1000:0 23      修改值
-d 1000:0
1000:0000 23 00 00 00 00 00 00 00-00 00 00 00 00 00 00 #.....
1000:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-

```

- 写入数据，最终方案：

```

1000:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
-e 1000:0 23 11 22 66      这种方式更方便，且偏移0会只增
-d 1000:0
1000:0000 23 11 22 66 00 00 00 00-00 00 00 00 00 00 00 #.'f.....
1000:0010 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0020 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0030 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0040 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0050 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0060 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
1000:0070 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 .....
- ▲

```

- 准备开始输入指令

```

-a 1380:100
1380:0100 mov ax,1000
1380:0103 mov ds,ax
1380:0105 mov ax,[0]
1380:0108 mov bx
      ^ Error
1380:0108 mov bx,[2]
1380:010C mov cx,[1]
1380:0110 add bx,[1]
1380:0114 add cx,[2]
1380:0118
-r cs
CS 2000
:1380
-r ip
IP 001D
:100
-r
AX=0017 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0100  NU UP EI PL NZ NA PO NC
1380:0100 B80010      MOV     AX,1000
-

```

- 执行的结果

```

AX=1000 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0103  NU UP EI PL NZ NA PO NC
1380:0103 8ED8      MOV     DS,AX
-t
AX=1000 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0105  NU UP EI PL NZ NA PO NC
1380:0105 A10000      MOV     AX,[0000]                      DS:0000=1123
-t
AX=1123 BX=1000 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0108  NU UP EI PL NZ NA PO NC
1380:0108 8B1E0200     MOV     BX,[0002]                      DS:0002=6622
-t
AX=1123 BX=6622 CX=0000 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=010C  NU UP EI PL NZ NA PO NC
1380:010C 8B0E0100     MOV     CX,[0001]                      DS:0001=2211
-t
AX=1123 BX=6622 CX=2211 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0110  NU UP EI PL NZ NA PO NC
1380:0110 031E0100     ADD     BX,[0001]                      DS:0001=2211

```

- 结果:

```

-t
AX=1123 BX=8833 CX=2211 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0114 0V UP EI NG NZ NA PE NC
1380:0114 030E0200 ADD CX,[0002] DS:0002=6622
-t
AX=1123 BX=8833 CX=8833 DX=0000 SP=00FD BP=0000 SI=0000 DI=0000
DS=1000 ES=073F SS=073F CS=1380 IP=0118 0V UP EI NG NZ NA PE NC
1380:0118 0000 ADD [BX+SI],AL DS:8833=00

```

小结

以学mov指令的几种形式：

- 1、mov 寄存器，数据 如：mov ax, 6
- 2、mov 寄存器，寄存器 如：mov ax, bx
- 3、mov 寄存器，内存单元 如：mov ax, [0]
- 4、mov 内存单元，寄存器 如：mov [0], ax
- 5、mov 段寄存器，寄存器 如：mov ds, ax
- 6、mov 寄存器，段寄存器 如：mov ax, ds

add、sub指令

add和sub指令同mov一样，都有两个操作对象

add 寄存器，数据	比如：add ax, 8
add 寄存器，寄存器	比如：add ax, bx
add 寄存器，内存单元	比如：add ax, [0]
add 内存单元，寄存器	比如：add [0], ax
sub 寄存器，数据	比如：sub x, 9
sub 寄存器，寄存器	比如：sub ax, bx
sub 寄存器，内存单元	比如：sub ax, [0]
sub 内存单元，寄存器	比如：sub [0], ax

数据段

对于8086pc机，我们可以根据需要将一组内存单元定义为一个段（可以是代码段、数据段等），也就是说如果是CS、IP指向则是代码段，如果是DS指向则是数据段。

定义： 我们可以将一组长度为 $N(N \leq 64K)$,地址连续，起始地址为16的倍数的内存单元当作专门存储数据的内存空间，从而定义了一个数据段。

如何访问数据段中的数据那？

将一段内存当作数据段，是我们在编程时的一种安排，我们可以在具体操作的时候，用ds存放数据段的段地址，再根据需要，用相关指令访问数据段中的具体单元。

例子：

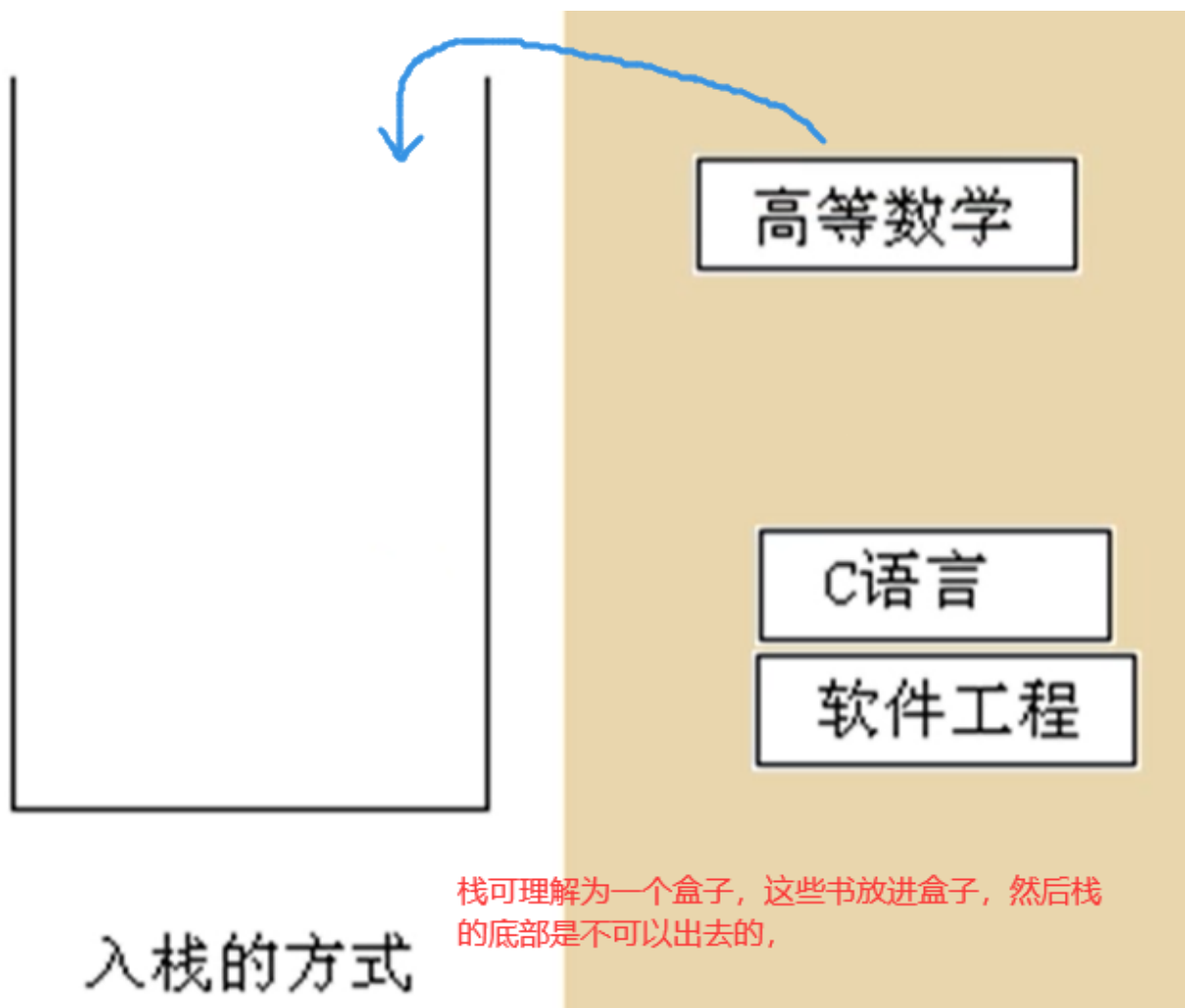
我们将123B0H~123BAH的内存单元定义为数据段，我们现在要累加这个数据段中的前3个单元中的数据，代码如下：

```
mov ax,123BH
mov ds,ax      ;将123BH送入ds中，作为数据段的段地址。
mov al,0       ;用al存放累加结果
add al,[0]     ;将数据段第一个单元（偏移地址为0）中的数值加到al中
add al,[1]     ;将数据段第二个单元（偏移地址为1）中的数值加到al中
add al,[2]     ;将数据段第三个单元（偏移地址为2）中的数值加到al中
```

栈

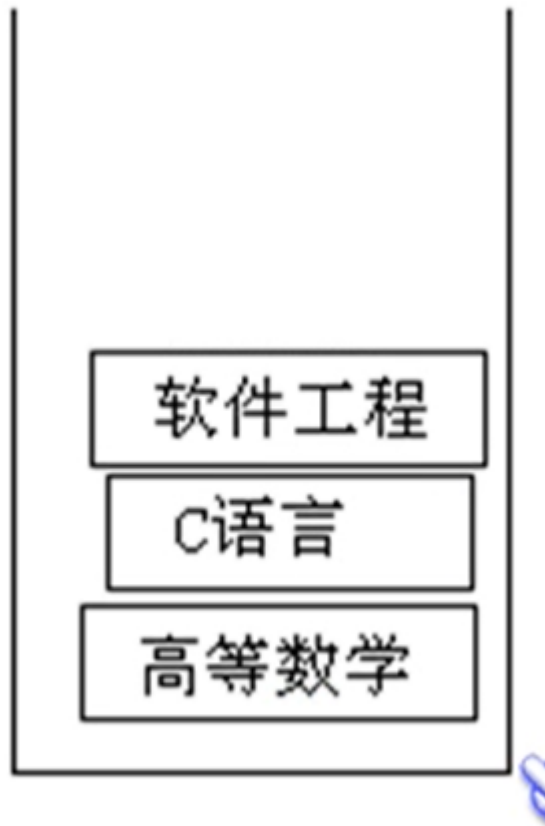
我们研究栈的角度：**栈是一种具有特殊的访问方式的存储空间**，它的特殊性就在于，最后进入这个空间的数据，最先出去（FILO），先进后出，最后进入的最先出去。

- 入栈的方式：



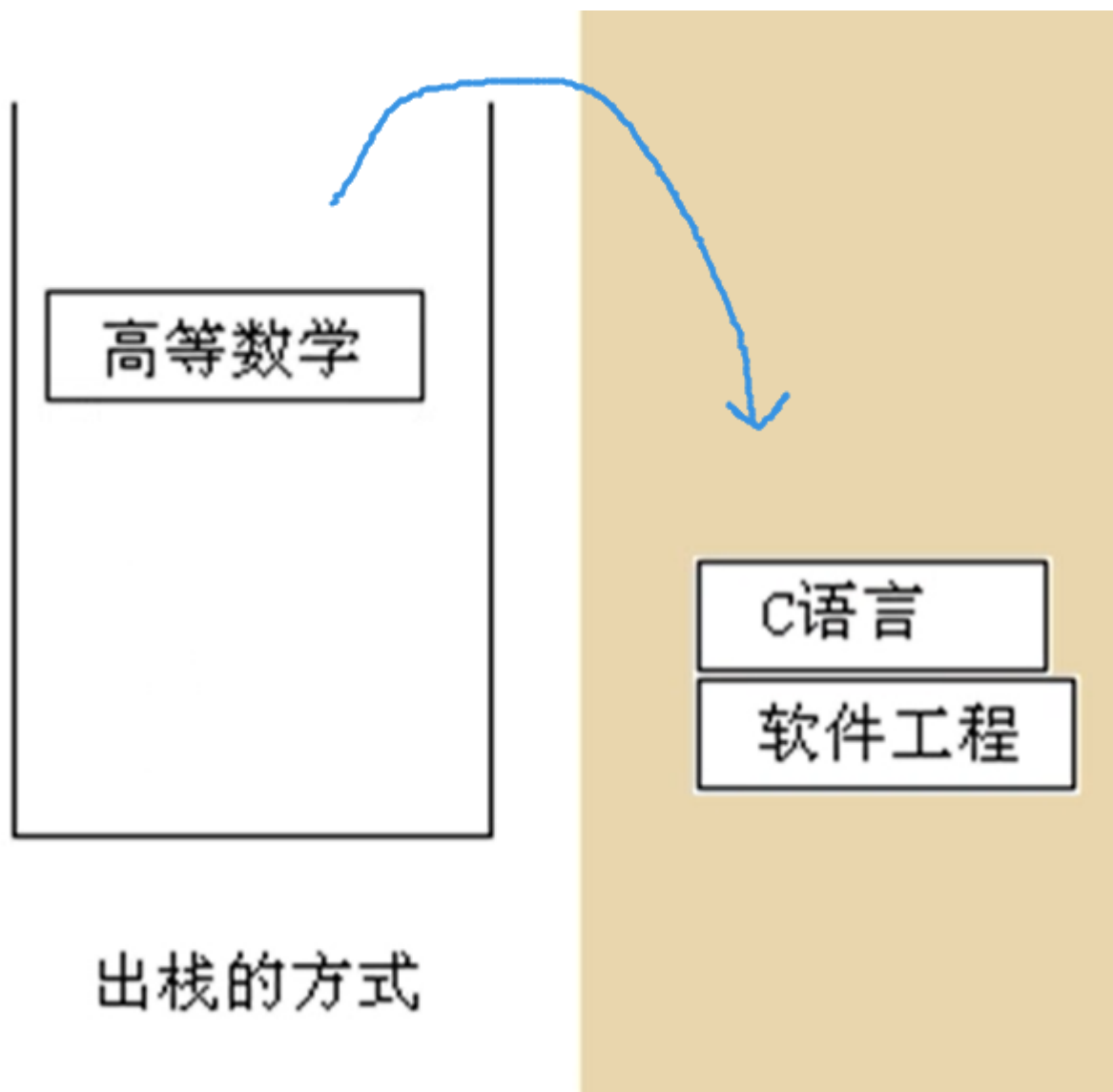
栈可理解为一个盒子，这些书放进盒子，然后栈的底部是不可以出去的，

- 入栈后：



入栈的方式

- 出栈后，出栈的方式是高等数学最后出来，因为**后进先出**



栈有两个基本的操作：入栈和出栈

- **入栈**：将一个新的元素放到栈顶
- **出栈**：从栈顶取出一个元素

栈顶的元素总是最后一个入栈，需要出栈是，又是最先被从栈中取出的

栈的操作规则：**LIFO**

现在的cpu中都有栈的设计，8086cpu提供相关的指令来以栈的方式访问内存空间，这意味着，我们在基于8086cpu编程的时候，可以将一段内存当作栈来使用。

8086cpu提供入栈和出栈指令：（最基本的）**PUSH**（入栈）、**POP**（出栈）

```
push ax  # 将寄存器ax中的数据送入栈中
pop ax   # 从栈顶取出数据送入ax中，注意8086cpu的入栈和出栈操作都是以字为单位进行的
```

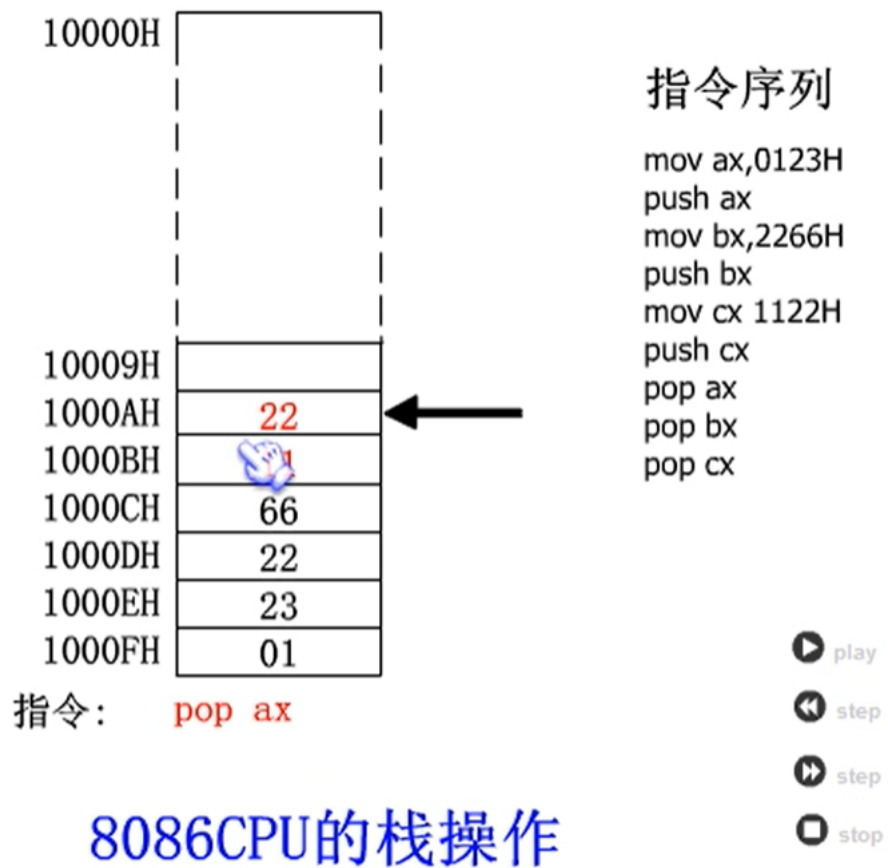
下面举例说明，我们可以将10000H~1000FH这段内存当作栈来使用，下面一段指令的执行过程：

```

mov ax,0123H
push ax
push bx,2266H
push bx
mov cx,1122H
push cx
pop ax
pop bx
pop cx

```

通过push入栈后的数据如下：



上面的操作之后发现ax为1122H，bx为2266H，cx为0123H；注意：字型数据用两个单元存放，高地址单元放高8位，低地址放低8位

cpu如何指导当前要执行的指令所在的位置？

答：寄存器CS和IP中存放着当前指令的段地址和偏移地址。

8086cpu中，有两个寄存器：

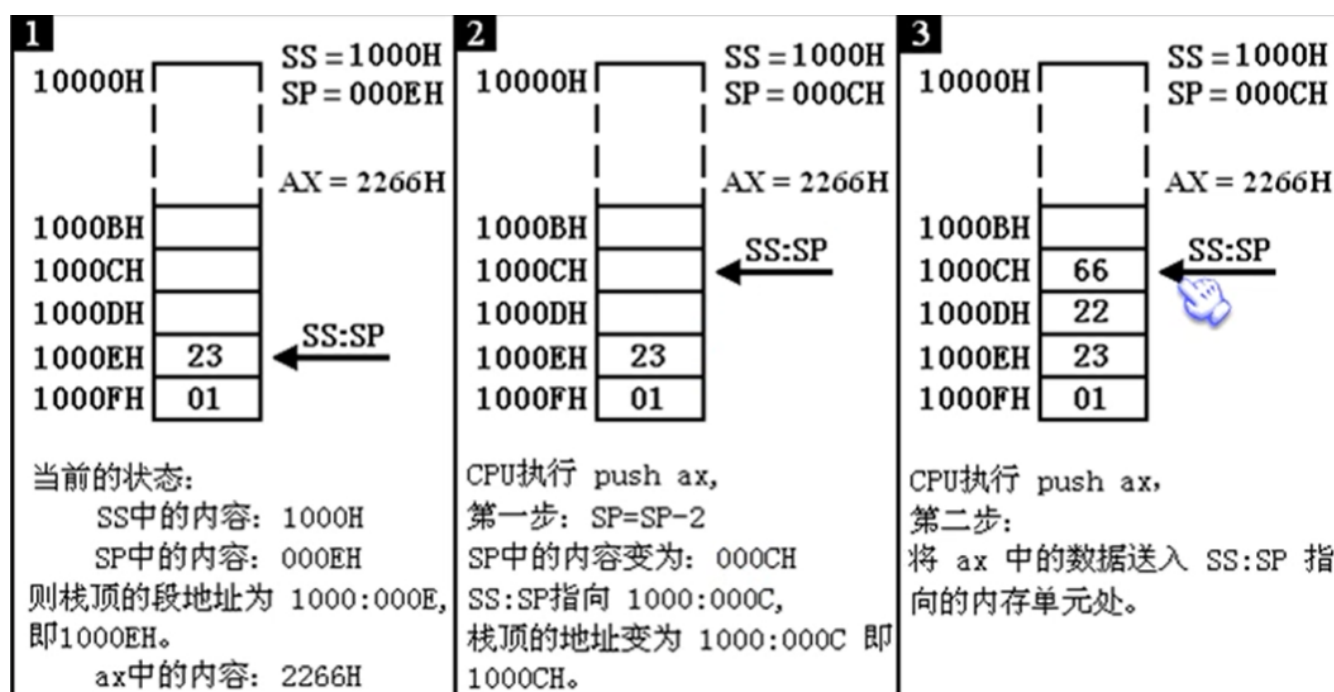
段寄存器SS (Stack Segment) 存放栈顶的段地址
寄存器SP (Stack Pointer) 存放栈顶的偏移地址

结论： 任意时刻，SS: SP指向栈顶元素。最开始的时候指向F

CPU是怎么知道指针的移动的？

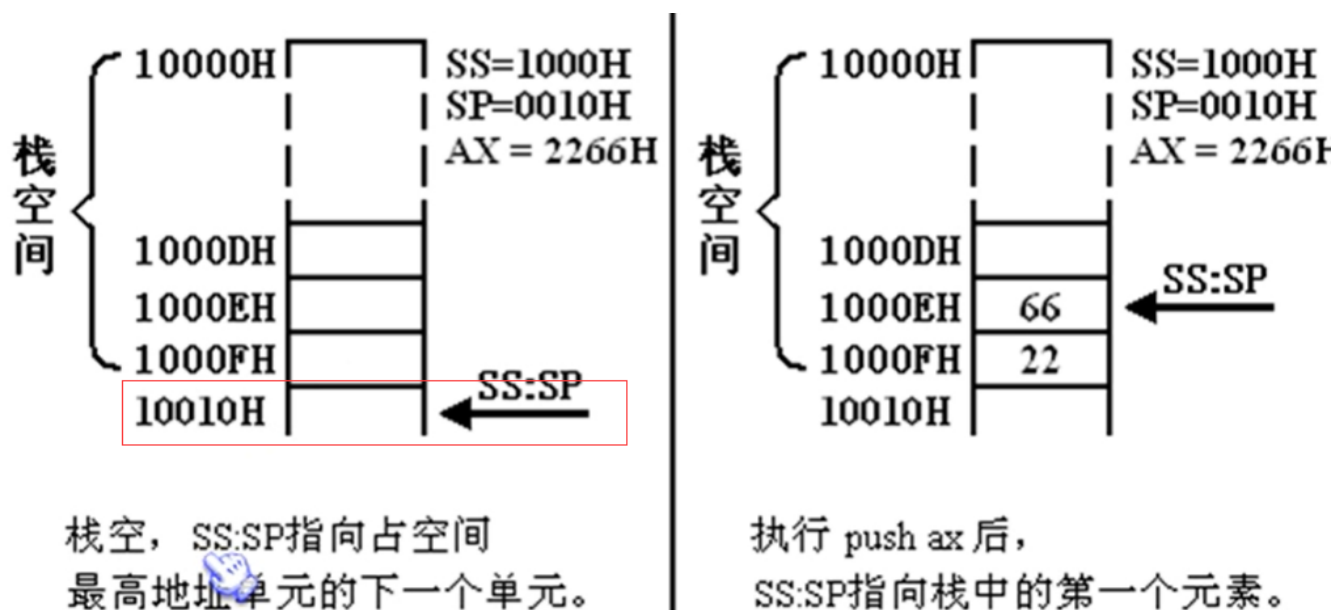
push ax

- 1、SP=SP-2
- 2、将ax中的内容送入SS: SP指向的内存单元处，SS: SP此时指向新栈顶



问题：如果我们将10000H~1000FH这段空间当作栈，初始状态是空的，此时，SS=1000H，SP=？

答：10010H

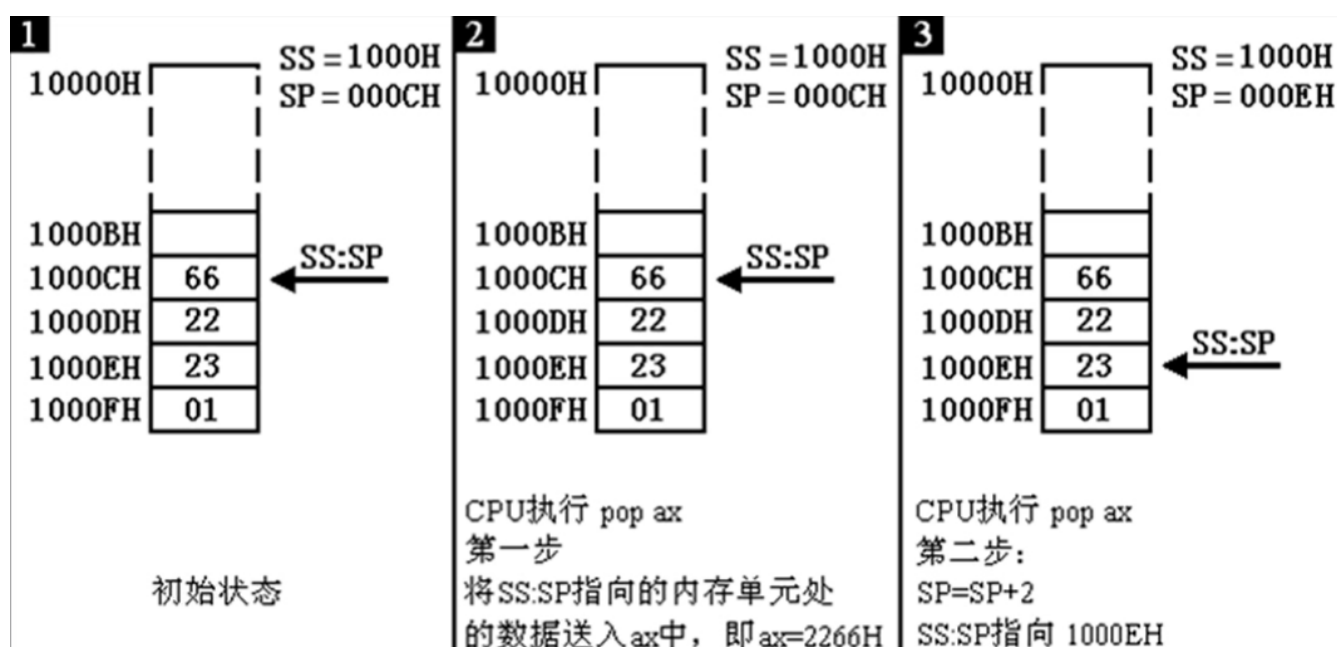


总结：

- 任意时刻，SS: SP指向栈顶元素，当栈为空的时候，栈中没有元素，也就不存在栈顶元素；所以SS: SP只能指向栈的最底部单元下面的单元，该单元的偏移地址为栈最底部的字单元的偏移地址+2；栈最底部字单元的地址为1000: 000E，所以栈空时，SP=0010H。

pop ax

- 1、将SS: SP指向的内存单元处的数据送入ax中；
- 2、SP=SP+2，SS: SP指向当前栈顶下面的单元，以当前栈顶下面的单元为新的栈顶，数据只会被覆盖，并没有删除



栈顶超界问题

SS和SP只记录了栈顶的地址，依靠SS和SP可以保证在入栈和出栈时找到栈顶。可是，如何能够保证在入栈、出栈时，栈顶不会超出栈空间。

push入栈时，如果栈满了会发生什么事，**溢出**

1000EH	23
1000FH	01
10010H	23
10011H	01
10012H	23
10013H	01
10014H	23
10015H	01
10016H	23
10017H	01
10018H	23
10019H	01
1001AH	23
1001BH	01
1001CH	23
1001DH	01
1001EH	23
1001FH	01
10020H	
10021H	
10022H	

栈空间

SS:SP 再执行 push ax 后的状态
此时，栈顶超出栈空间

push ax

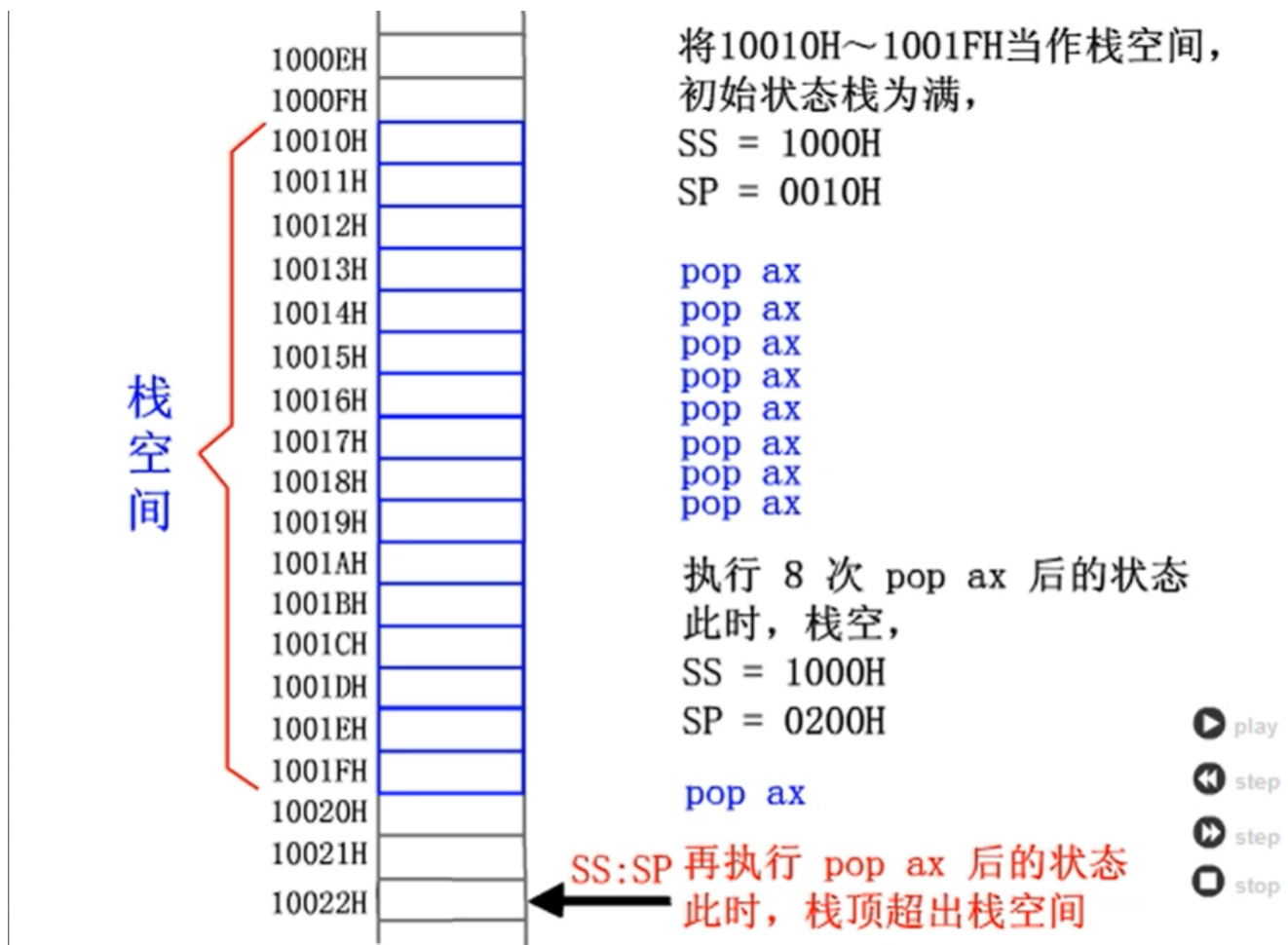
执行 8 次 push ax 后的状态
此时，栈空间满

push ax
push ax
push ax
push ax
push ax
push ax
push ax
push ax

将10010H~1001FH当作栈空间，
初始状态栈为空，
SS = 1000H
SP = 0020H
AX = 0123H

play
step
step
stop

pop出栈时，如果栈空了再出栈会发生什么？也属于**溢出**



注意：栈顶超界是危险的

因为我们既然将一段空间安排为栈，那么在栈空间之外的空间里很可能存放了，具有其他用途的数据，代码等，这些数据、代码可能是我们自己的程序中的，也可能是别的程序中的。（毕竟一个计算机系统并不是只有我们自己的程序在运行）

栈与内存

栈空间当然也是内存空间的一部分，它只是一段可以以一种特殊的方式进行访问的内存空间

push和pop指令的格式

```
push 寄存器    # 将一个寄存器中的数据入栈
pop  寄存器    # 出栈，用一个寄存器接受出栈的数据
```

如：push ax

```
push 内存单元  # 将一个内存单元处的字入栈（栈操作都是以字为单元）
pop 内存单元  # 出栈，用一个内存字单元接收出栈的数据，指令执行时，cpu要知道内存单元的地址，可以在push、pop指令中给出内存单元的偏移地址，段地址在指令执行时，cpu从ds中取得
```

如：push [0]
pop [2]

编程题

1、将10000H~1000FH这段空间当作栈，初始状态是空的，将AX、BX、DS中的数据入栈


```
mov ax,1000H
mov ss,ax      ;设置栈的段地址，SS=1000H，不能直接向段寄存器SS送入
               ;数据，所以用ax中转。

mov sp,0010H   ;设置栈顶的偏移地址，因为栈为空，所以SP=0010H。如果
               ;对栈为空时SP的设置还有疑问，复习3.7节、问题3.6。

               ;上面三条指令设置栈顶地址。编程中要自己注意栈的大小

push ax
push bx
push ds
```

2、将10000H~1000FH这段空间当作栈，初始状态是空的；设置AX=001AH,BX=001BH;将AX，BX中的数据入栈；然后将AX、BX清零；从栈中恢复AX、BX原来的内容。

```
mov ax,1000H
mov ss,ax
mov sp,0010H   ;初始化栈顶，栈的情况如图a所示
mov ax,001AH
mov bx,001BH
push ax        
push bx        ;ax、bx入栈，栈的情况如图b所示
sub ax,ax      ;将ax清零，也可以用mov ax,0，
               ;sub ax,ax的机器码为两个字节，
               ;mov ax,0，的机器码为3个字节。

sub bx,bx
pop bx         ;从栈中恢复ax、bx原来的数据，当前栈顶的内容是bx
pop ax        ;中原来的内容：001BH，ax中原来的内容001AH在栈顶
               ;的下面，所以要先pop bx，然后再pop ax。
```

3、将10000H~1000FH这段空间当作栈，初始状态是空的；设置AX=002AH，BX=002BH；利用栈，交换AX和BX中的数据


```

mov ax,1000H
mov ss,ax
mov sp,0010H      ; 初始化栈顶，栈的情况如图a所示
mov ax,002AH
mov bx,002BH
push ax
push bx            ; ax、bx入栈，栈的情况如图b所示
pop ax             ; 当前栈顶的数据是bx中原来的数据：002B；
                  ; 所以先pop ax， ax=002BH；
pop bx            ; 执行pop ax后，栈顶的数据为ax原来的数据；
                  ; 所以再pop bx， bx=002AH。

```

栈段

对于8086pc机，在编程时，我们可以根据需要，将一组内存单元定义为一个段。我们可以将长度为N（ $N \leq 64K$ ）的一组地址连续，起始地址为16的倍数的内存单元，当作栈来用，从而定义了一个栈段。

问题：如果我们将10000H~1FFFFH这段空间当作栈段，初始状态是空的，此时，SS=1000H，SP=?

答:10010H

我们将10000H~1FFFFH这段空间当作栈段，SS=1000H，栈空间大小为64KB，栈最底部的字单元地址为1000:FFFE，任意时刻，SS：SP指向栈顶，当栈中只有一个元素的时候，SS=1000H，SP=FFFEH。