第3章 寄存器（内存访问）

第3章 寄存器（内存访问）

- 3.1 内存中字的存储
- 3.2 DS和[address]
- 3.3 字的传送
- 3.4 mov、add、sub指令
- 3.5 数据段
- 3.6 栈
- 3.7 CPU提供的栈机制
- 3.8 栈顶超界的问题
- 3.9 push、pop指令
- 3.10 栈段

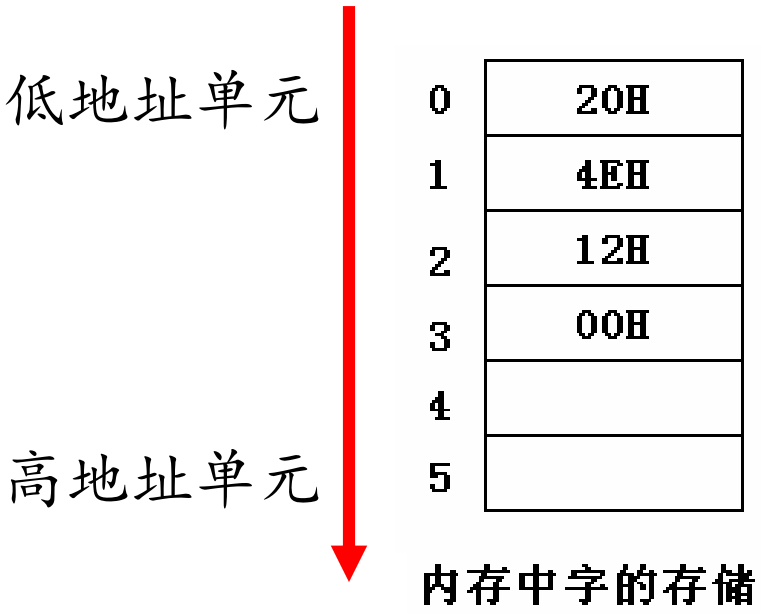
引言

- 在第2章中，从CPU执行指令的角度讲解了8086CPU的逻辑结构、形成物理地址的方法、相关的寄存器以及一些指令。

- 本章将从访问内存的角度继续学习几个寄存器。

3.1 内存中字的存储

- 在0地址处开始存放20000：



3.1 内存中字的存储

N号内存单元 和 N+1号内存单元，

看法1——以字节为单位

地址为0的字节单元0

地址为1的字节单元1

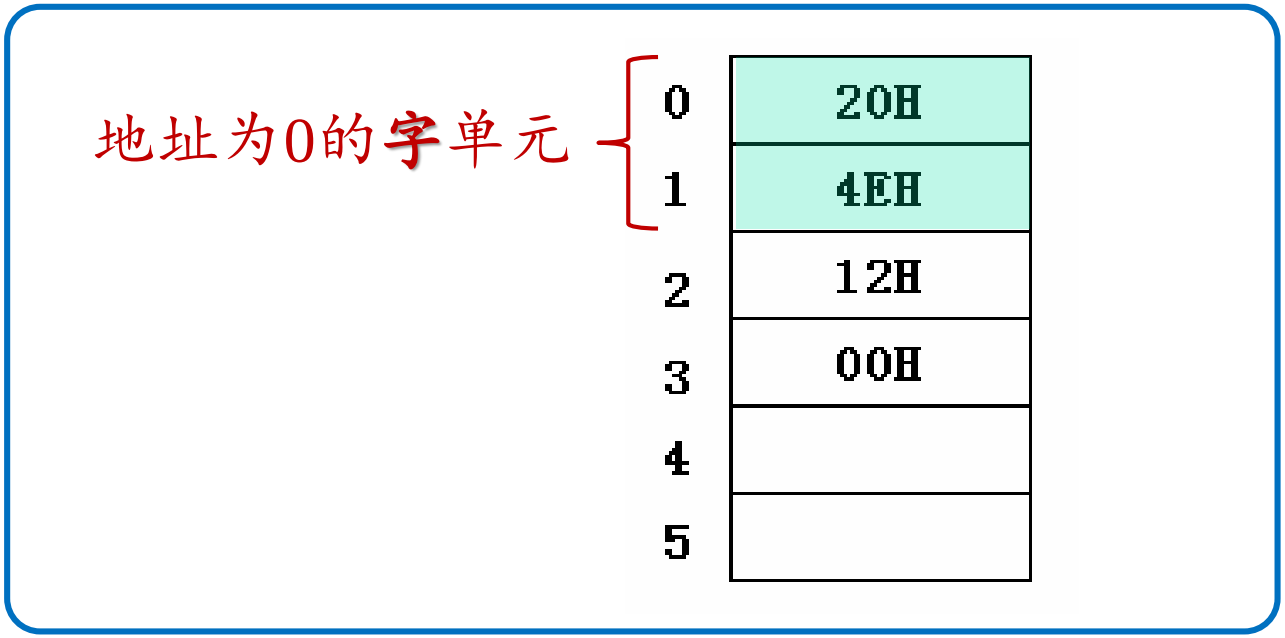
0	20H
1	4EH
2	12H
3	00H
4	
5	

内存-以字节为单位

3.1 内存中字的存储

N号内存单元 和 N+1号内存单元，

看法2——以字为单位



内存-以字为单位

- Little and Big Endian

长整形数(4字节长) 90AB12CDH 在内存中如何保存？

Address	Value
1000	90
1001	AB
1002	12
1003	CD

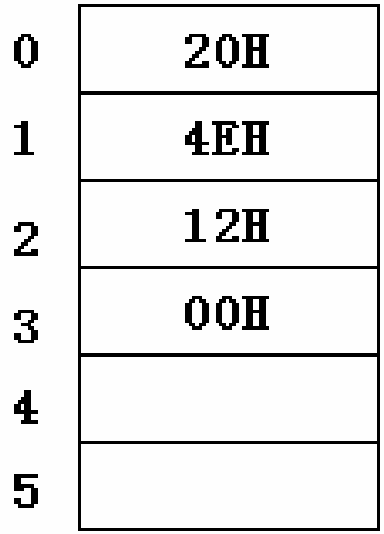
Big Indian
(内存中先存数据高位)
如Sun等计算机系统

Address	Value
1000	CD
1001	12
1002	AB
1003	90

Little Indian
内存中先存储数据地位
如Intel计算机系统

3.1 内存中字的存储

- 问题：
  - 0地址开始的
    - 字节型数据 = 20H
    - 字型数据 = 4E20H
  - 2地址开始的
    - 字节型数据 = 12H
    - 字型数据 = 0012H



内存中字的存储

3.2 DS和[address]

- 物理地址用段地址与偏移地址合成

  物理地址 = 段地址 × 16 + 偏移地址

- 数据段的段地址寄存器：**DS**（Code Segment）

3.2 DS和[address]

- mov指令读取内存的格式：

mov 寄存器名，[address]

其中address为内存单元的**偏移地址**

注意：

- 执行此指令时，8086CPU自动取DS中的数据为内存单元的段地址。
- 用此指令时应保证DS寄存器已保存正确的段地址

3.2 DS和[address]

- 问题：如何用mov指令从10000H中读取1个字节数据到al中？

  答：将物理地址10000H看作

  段地址:偏移地址=1000H:0H

  1. 将1000H放入DS寄存器中 （怎么实现呢？）
  2. 用mov al, [0]完成传送

3.2 DS和[address]

- 以下传送指令对吗？
  - mov bx, 1000H ✓
  - mov ds, 1000H ✗

8086CPU不支持将数据直接送入**段寄存器**的操作，即
数据➔**段**寄存器 ✗

- 如何把1000H送入ds？

数据 ➔ 通用寄存器 ➔ 段寄存器

3.2 DS和[address]

- 问题：如何用mov指令从10000H中读取1个字节数据到al中？

答：将物理地址10000H看作

段地址:偏移地址=1000H:0H

```
mov bx, 1000H      ; 立即数写入通用寄存器
mov ds, bx         ; 通用寄存器间复制到DS
mov al [0]         ; 内存数据写入寄存器
```

3.2 DS和[address]

- 问题：

  如何将al中的数据送入内存单元10000H？

答：这是怎样将数据从寄存器送入内存单元？

结论：

```
mov bx, 1000H
mov ds, bx
mov [0], al      （一种合理的回答）
```

3.3 字的传送

- 8086CPU是数据线宽度为16，一次可传送16位的数据（即一个**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
```

3.3 字的传送

- 问题3.3：已知内存中的情况如右图，写出下面指令执行后寄存器ax，bx，cx中的值。

```
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

3.3 字的传送

- 问题3.3分析

指令	执行后相关寄存器中的内容	说明
mov ax,1000H	ax = 1000H	
mov ds,ax	ds = 1000H	前两条指令的目的是将ds设为1000H
mov ax,[0]	ax = 1123H	1000:0处存放的字型数据送入ax； 1000:1单元处存放的字型数据的高8位：11H； 1000:0单元处存放的字型数据的低8位：23H； 所以1000:0处存放的字型数据为1123H。 指令执行时，字型数据的高8位送入ah，字型数据的低8位送入al，则ax中的数据为1123H
mov bx,[2]	bx = 6622H	以下原理同上
mov cx,[1]	cx = 2211H	
add bx,[1]	bx = 8833H	
add cx,[2]	cx = 8833H	

3.3 字的传送

- 问题3.4：已知内存中的情况如右图，写出下面指令执行后寄存器ax，bx，cx中的值。

```
mov ax,1000H
mov ds,ax
mov ax,11316
mov [0],ax
mov bx,[0]
sub  bx,[2]
mov [2],bx
```

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

内存情况示意

3.3 字的传送

- 问题3.4分析

指令	执行后相关寄存器或内存单元中的内容	说明
mov ax,1000H	ax = 1000H	
mov ds,ax	ds = 1000H	前两条指令的目的是将ds设为1000H
mov ax,11316	ax = 2C34H	十进制11316，十六进制2C34H
mov [0],ax	10000H 34 10001H 2C 10002H 22 10003H 11	ax中的字型数据送到1000:0处： ax中的字型数据是2C34H， 高8位：2CH，在ah中， 低8位：34H，在al中， 指令执行时，高8位送入高地址1000:1单元，低8位送入低地址1000:0单元
mov bx,[0]	bx = 2C34H	
sub bx,[2]	bx = 1B12H	bx = bx中的字型数据 - 1000:2处的字型数据 = 2C34H - 1122H = 1B12H
mov [2],bx	10000H 34 10001H 2C 10002H 12 10003H 1B	bx中的字型数据送到1000:2处

3.4 mov、add、sub指令

- 已学mov指令的几种形式：

  mov 寄存器, 数据

  mov 寄存器, 寄存器

  mov 寄存器, 内存单元

  mov 内存单元, 寄存器

  mov 段寄存器, 寄存器

3.4 mov、add、sub指令

- 根据已知指令进行推测：
  - mov 段寄存器，寄存器
    - ➔ mov 寄存器，段寄存器
  - mov 内存单元，寄存器
    - ➔ mov 内存单元，段寄存器
    - ➔ mov 段寄存器，内存单元

3.4 mov、add、sub指令

- mov 指令

source

	General Register	Segment Register	Memory Location	Constant
General Register	yes	yes	yes	yes
Segment Register	yes	no	yes	no
Memory Location	yes	yes	no	yes

destination

3.4 mov、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

- 它们可以对**段寄存器**进行操作吗？

（请自行在Debug中试验）

3.4 mov、add、sub指令

- add / sub 指令

source

	General Register	Memory Location	Constant
General Register	yes	yes	yes
Memory Location	yes	no	yes

destination

3.5 数据段

- 8086PC机上，可根据需要将一组连续内存单元定义为一个段。
- 可将一组长度为N（N≤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——数据段。
- 如：用123B0H~123B9H这段空间来存放数据：
  - 段地址：123BH
  - 长度：10字节

3.5 数据段

- 如何访问数据段中的数据呢？

- 将一段内存当作数据段，是编程时的一种安排，用 ds 存放数据段的段地址，再用偏移地址访问数据段中的具体单元。

3.5 数据段

- 示例：将123B0H~123BAH的内存单元定义为数据段，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
```

3.5 数据段

- 问题3.5　写几条指令，累加数据段中的前3个字型数据。

答：一个字型数据占两个单元，所以偏移地址是0、2、4。

```
mov ax,123BH
mov ds,ax        ;将123BH送入ds中，作为数据段的段地址。
mov ax, 0        ;用ax存放累加结果
add ax,[0]       ;将数据段第一个字(偏移地址为0)加到ax中
add ax,[2]       ;将数据段第二个字(偏移地址为2)加到ax中
add ax,[4]       ;将数据段第三个字(偏移地址为4)加到ax中
```

3.1节~3.5节 小结

- （1）一个字占两个地址连续字节，字的低位字节存放在低地址单元中，高位字节存放再高地址单元中。

- （2）用 mov 指令访内时，可只给出内存单元的偏移地址，段地址默认在DS寄存器中。

- （3）[address]表示一个偏移地址为address的内存单元。

3.1节~3.5节 小结（续）

- （4）内存和寄存器之间传送字型数据时，高地址单元和高8位寄存器、低地址单元和低8位寄存器相对应。

- （5）mov、add、sub是具有两个操作对象的指令。jmp是具有一个操作对象的指令。

特别提示

- 检测点3.1（p55）

```
mov ax,1
mov ds,ax
mov ax,[0000]        AX=_____
mov bx,[0001]        BX=_____
mov ax,bx            AX=_____
mov ax,[0000]        AX=_____
mov bx,[0002]        BX=_____
add ax,bx            AX=_____
add ax,[0004]        AX=_____
mov ax,0             AX=_____
mov al,[0002]        AX=_____
mov bx,0             BX=_____
mov bl,[000C]        BX=_____
add al,bl            AX=_____
```

- 没有通过检测点

请不要向下学习！

3.6 栈

- 栈的操作规则：LIFO

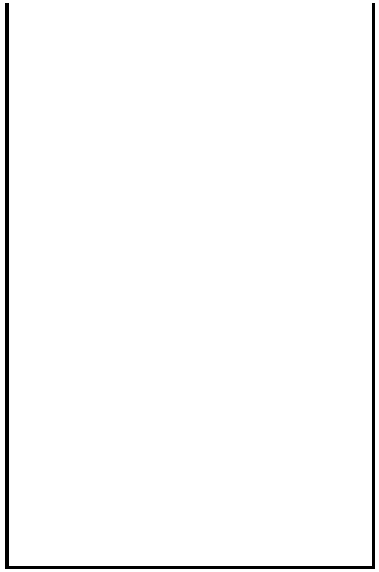
  （Last In First Out，后进先出）

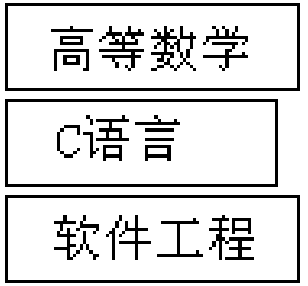
  - 栈的特点：栈顶的元素总是最后入栈，需要出栈时，又最先被从栈中取出。

- 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。

栈的示例

- 可以用一个盒子和3本书来描述栈的操作方式——入栈

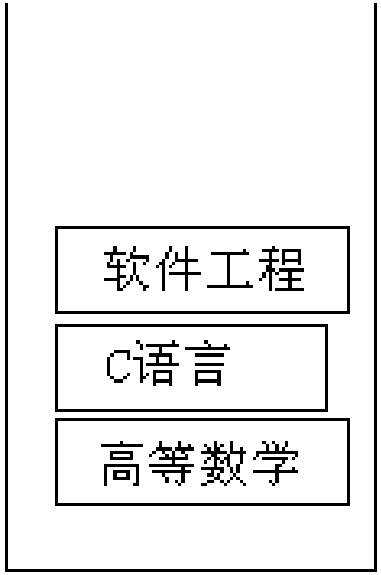




入栈的方式

栈的示例

- 可以用一个盒子和3本书来描述栈的操作方式——出栈



出栈的方式

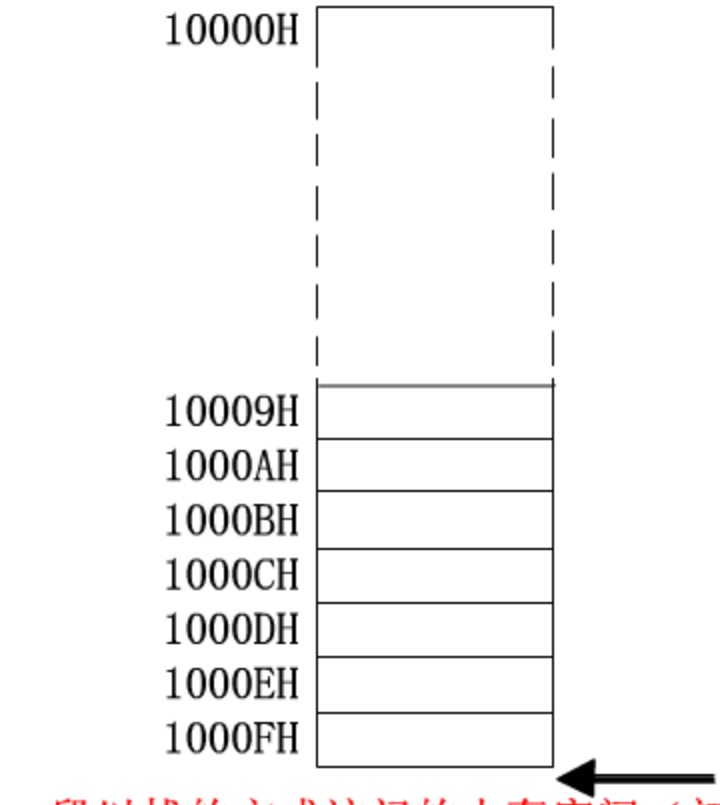
3.7 CPU提供的栈机制

- 8086CPU提供入栈和出栈指令：
  - 入栈指令 PUSH： push ax
    - 作用：将寄存器ax中的数据送入栈中；
  - 出栈指令 POP：pop ax
    - 作用：从栈顶取出数据送入ax

注意：8086CPU的入/出栈操作都是以字为单位进行的——低地址单元放低8 位，高地址单元放高 8 位（Little Indian）。

8086CPU栈操作演示

- 例：将内存10000H~1000FH段当作栈来使用。



指令序列

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

play
step
step
stop

一段以栈的方式访问的内存空间（初始情况）

8086CPU的栈操作

两个疑问

- 两个疑惑
  - CPU如何知道哪段内存空间被当作栈使用？
  - 执行push和pop时，CPU把哪个内存单元当栈顶？

对于两个疑问的分析

- 通过CS:IP, CPU可知当前要执行的指令位置：
  - CS 寄存器，存放当前指令的段地址
  - IP 寄存器，存放当前指令的偏移地址

类似于此，

- 8086CPU中，SS:SP指向栈顶元素：
  - SS 段寄存器，存放栈顶的段地址
  - SP 寄存器，存放栈顶的偏移地址

两个疑问

- 两个疑惑
  - CPU如何知道哪段内存空间被当作栈使用？
  - 执行push和pop时，如何知道哪个单元是栈顶？

- 结论：2个栈专用的寄存器
  - SS: Stack Segment
  - SP: Stack Pointer

牢记：任意时刻，SS:SP指向栈顶元素。

push 指令的执行过程

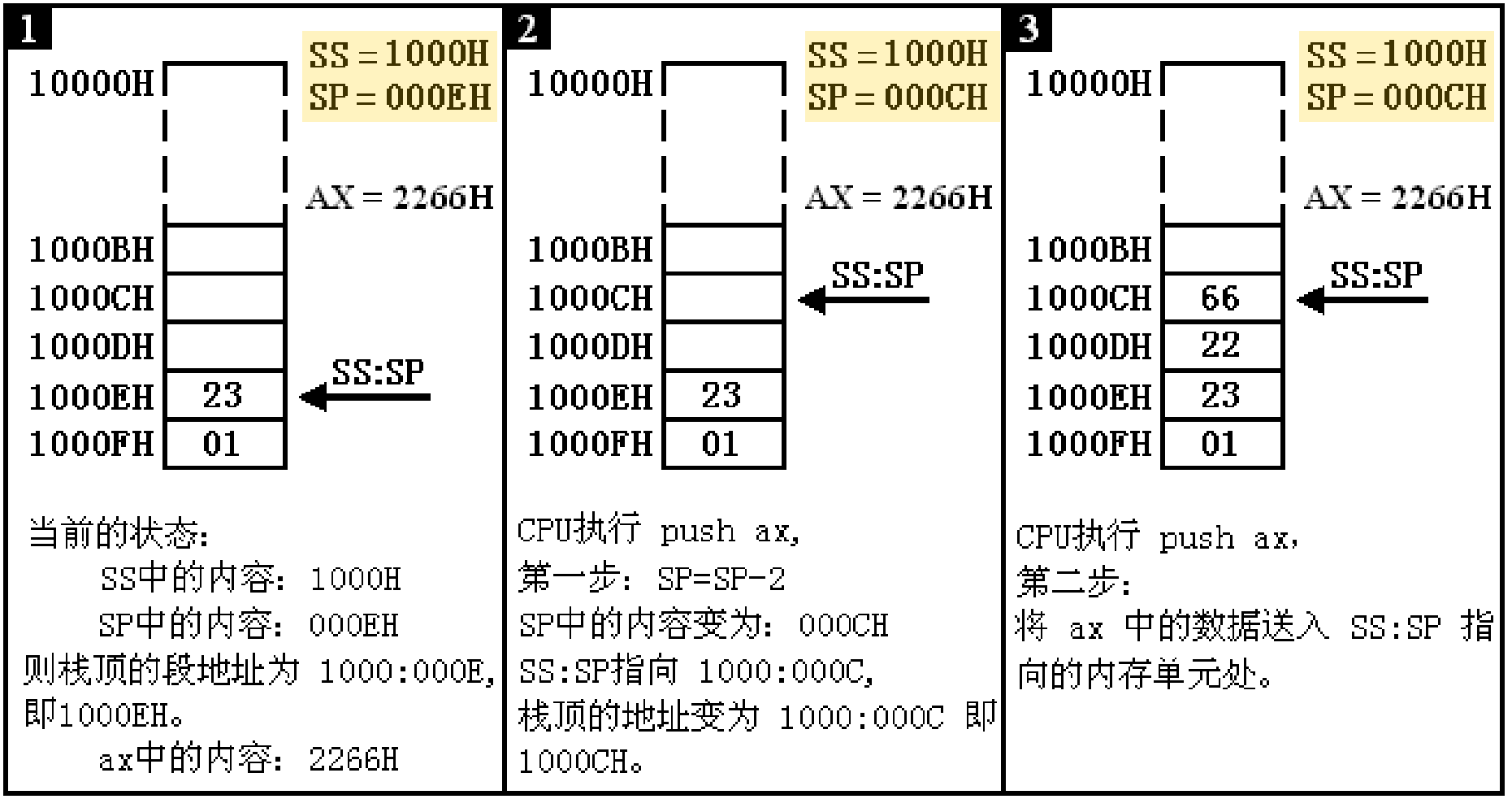
push ax

执行过程

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。

push 指令的执行过程

- 图示　push ax



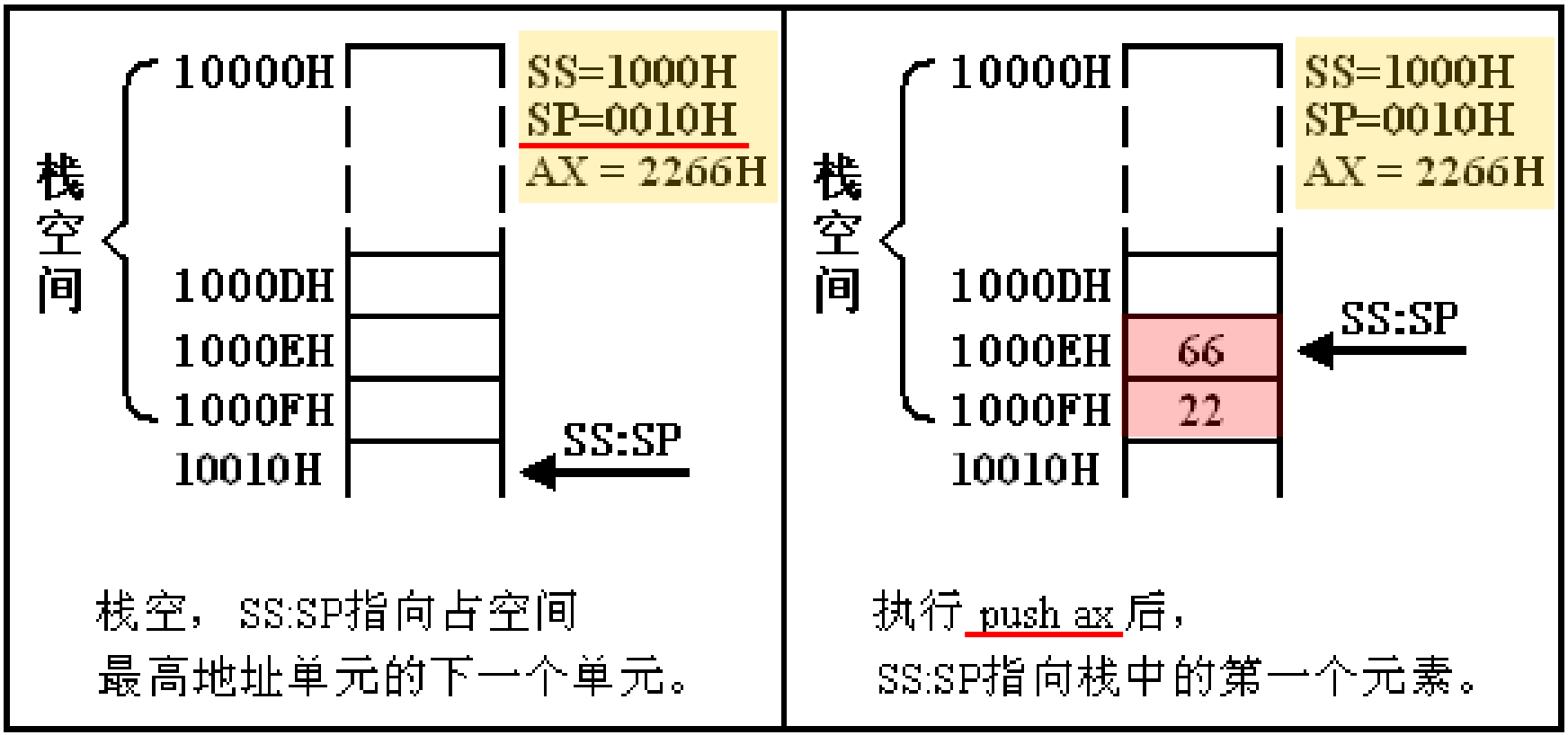
3.7 CPU提供栈的机制

- 问题3.6：

如果将10000H~1000FH 这段空间当作栈，初始时栈的状态为空时，应该如何设置初始的SS：SP？

答：SS=1000H
SP=0010H

问题3.6分析


栈空间
10000H
1000DH
1000EH
1000FH
10010H
SS=1000H
SP=0010H
AX = 2266H
SS:SP
栈空，SS:SP指向占空间
最高地址单元的下一个单元。
栈空间
10000H
1000DH
1000EH
1000FH
10010H
SS=1000H
SP=0010H
AX = 2266H
66
22
SS:SP
执行 push ax 后，
SS:SP指向栈中的第一个元素。

问题3.6

- 将10000H~1000FH 这段空间当作栈段，
  可取栈的段地址SS=1000H，
  因为： 栈底(即只有一个元素时，以字单元）便宜地址SP为000EH 。
  所以： 栈为空时(相当于栈中唯一的元素pop后）， SP=SP+2, 即SP = 0010H 。

结论：初始化栈时，

初始 SP = 栈底(字但愿)地址 + 2

pop 指令的执行过程

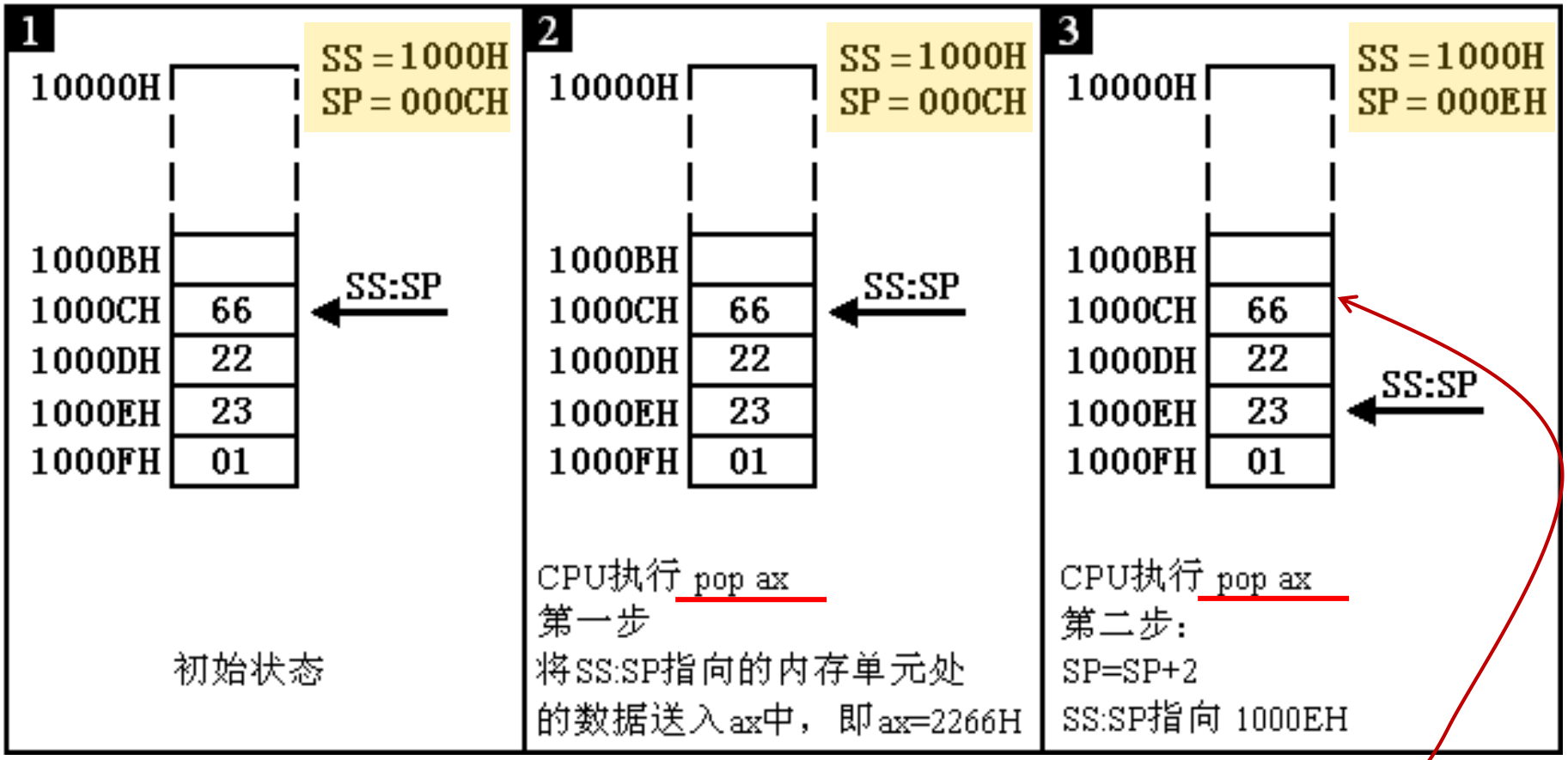
pop ax

执行过程

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。

pop 指令的执行过程



出栈后，1000CH 处的值相当于垃圾数据，
再次push后新数据会覆盖掉原数据

3.8 栈顶超界的问题

- 两个疑问
  - 当**栈满**的时再push会怎样？ **栈上溢**
  - 当**栈空**的时再pop会怎样？ **栈下溢**
- 栈顶指针超出栈空间，即栈顶超界（溢出）问题——危险。

栈上溢

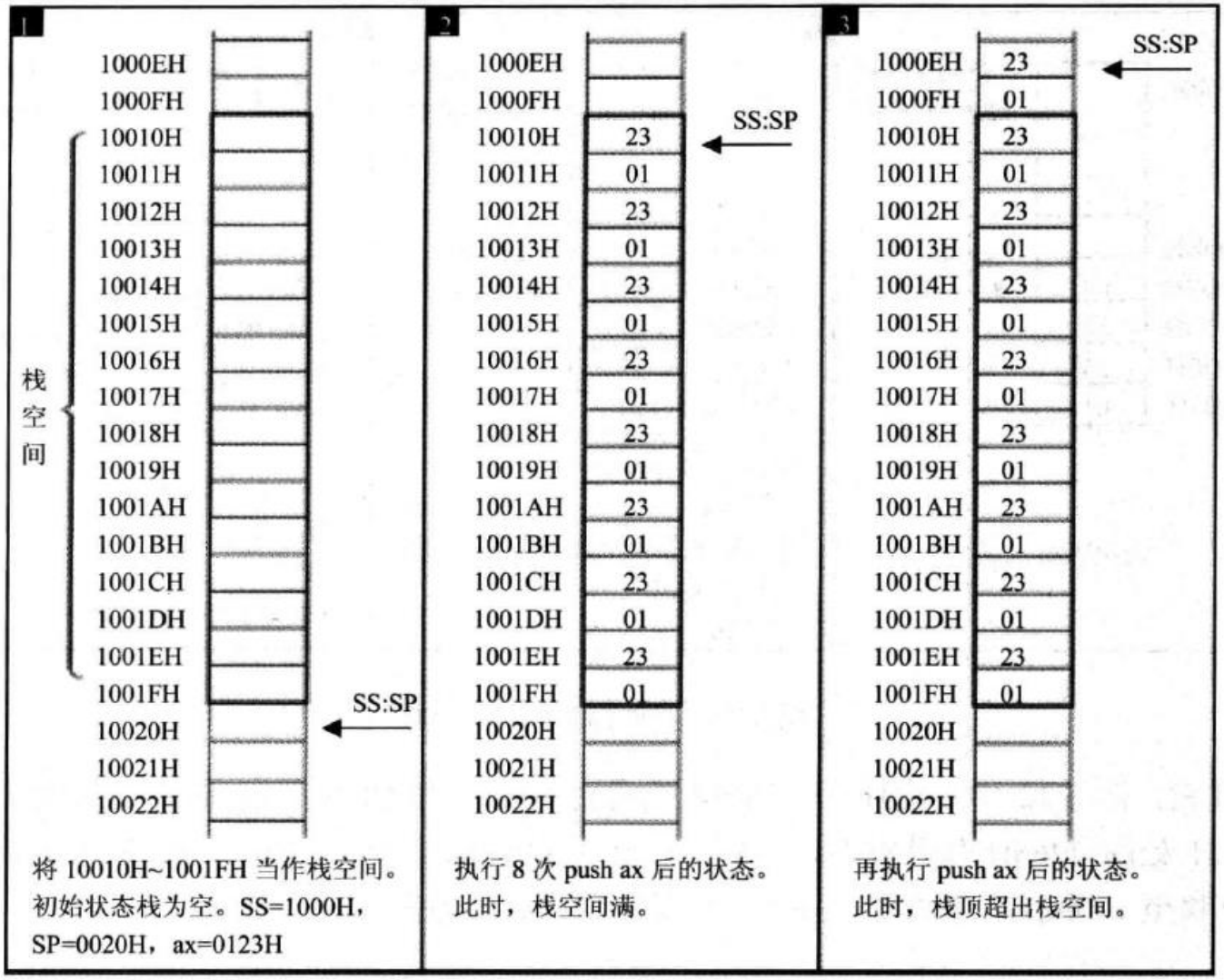


图 3.13　执行 push 后栈顶超出栈空间

栈下溢

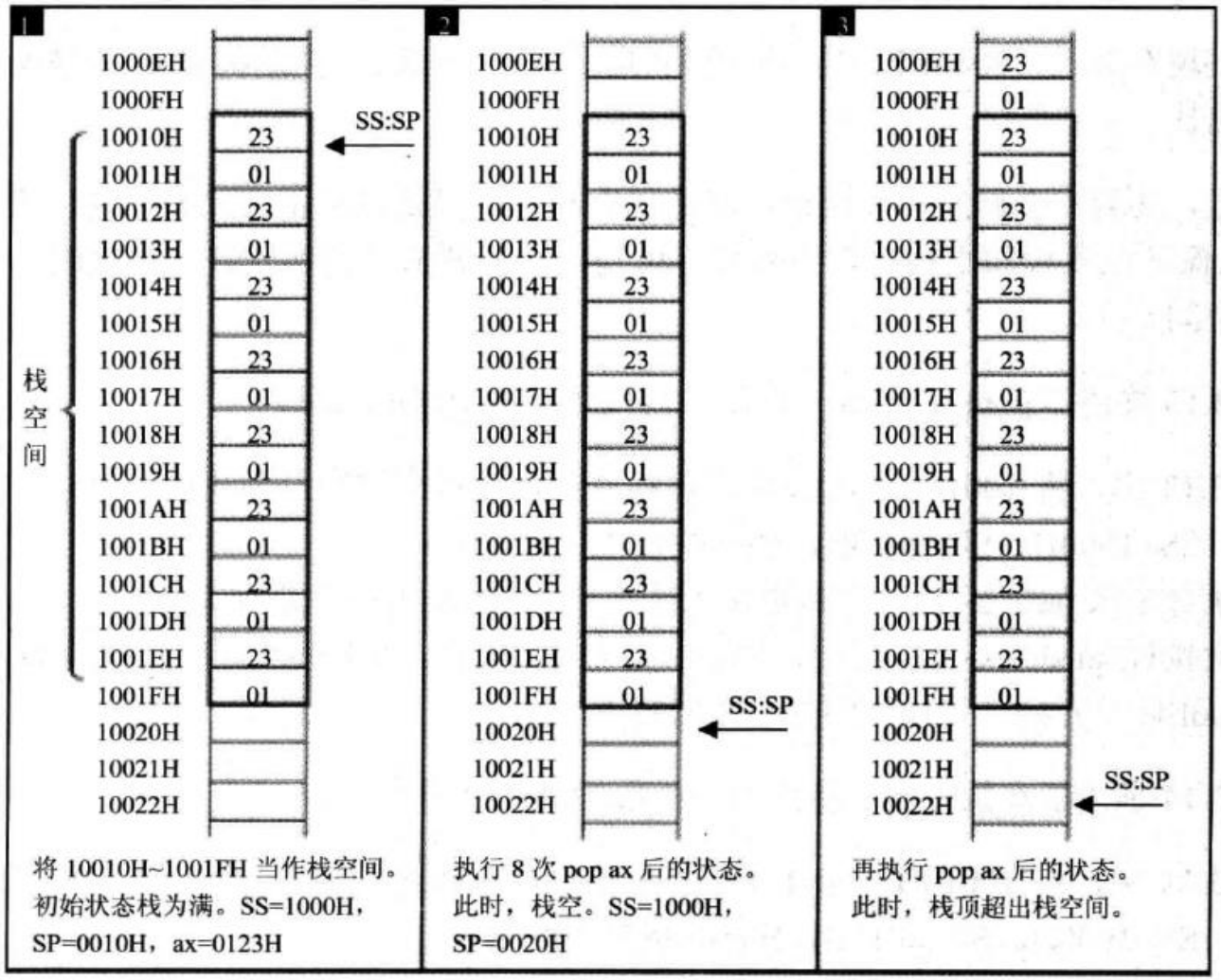
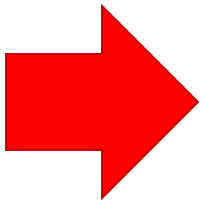


图 3.14　执行 pop 后栈顶超出栈空间

3.8 栈顶超界的问题

- 栈顶超界是危险的：

若在栈之外的空间里若存放了具有其他用途的数据、代码。



栈时将这些有用数据、代码意外地改写，将会引发一连串的错误。

3.8 栈顶超界的问题

- 8086CPU**不保证**对栈的操作不会超界。
  - 由SS:SP CPU可知栈顶内存位置。
    （不知栈的大小）
  - 由CS:IP CPU 可知当前要执行的指令内存位置。
    （不知道指令有多少）

3.8 栈顶超界的问题

- 如何能够保证在栈操作不会越界？
  - 思路1：CPU 可否解决堆栈越界问题？
    - 有的CPU会支持，但是8086CPU中不行
  - 思路2：程序员自行解决 （如何实现？）

3.8 栈顶超界的问题

- 程序员解决堆栈溢出问题
  - 编程时要按可能用到的最大栈空间来设置栈大小，防止入栈的数据太多而导致超界；
  - 执行出栈操作时也要注意，以防栈空时因继续出栈而导致超界。

3.9 push、pop指令

- push和pop：

  在寄存器和内存的栈段之间传送数据。

- 8086CPU中，push和pop的操作数可以是
  - 通用寄存器
  - 段寄存器
  - 内存

3.9 push、pop指令

- push和pop指令——格式1
  - push 寄存器

    将寄存器中的数据入栈
  - pop 寄存器

    用一个寄存器接收出栈数据

- 例如：

  push ax

  pop bx

  push al 指令对吗？

3.9 push、pop指令

- push和pop指令——格式2
  - push 段寄存器

    将一个段寄存器中的入栈
  - pop 段寄存器

    用一个段寄存器接收出栈的数据

- 例如：

```
push ds
pop es
```

3.9 push、pop指令

- push和pop指令——格式3
  - push 内存单元

    将一个内存单元处的字入栈
  - pop 内存单元

    用一个内存字单元接收出栈的数据
- 例如：

  push [0]

  pop [2]

  在 push、pop 指令中给出内存单元的偏移地址，CPU从ds中取得段地址。

3.9 push、pop指令

- 问题3.7

编程：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。

- 思考后看分析。

问题3.7分析

```
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
```

3.9 push、pop指令

- 问题3.8

编程：

（1）将10000H~1000FH这段空间当作栈，初始状态是空的；

（2）设置AX=001AH，BX=001BH；

（3）将AX、BX中的数据入栈；

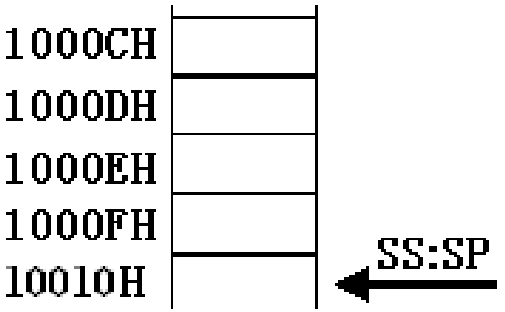
（4）然后将AX、BX清零；

（5）从栈中恢复AX、BX原来的内容。

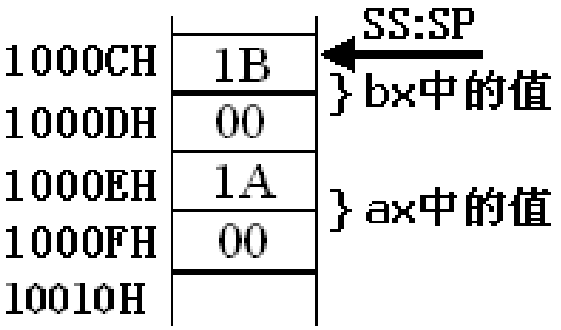
- 思考后看分析。

问题3.8分析

```
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。
```



(a)栈初始化的情况



（b）ax、bx入栈的情况

用栈来暂存以后需要恢复的寄存器中的内容时，
出栈的顺序要和入栈的顺序相反

3.9 push、pop指令

- 问题3.9

  编程：

  （1）将10000H~1000FH 这段空间当作栈，初始状态是空的；

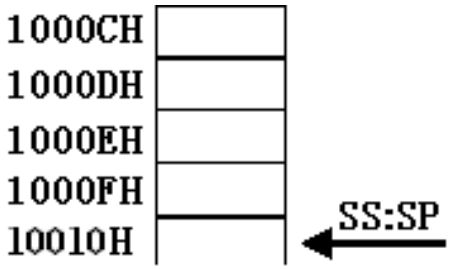
  （2）设置AX=002AH，BX=002BH；

  （3）利用栈，交换 AX 和 BX 中的数据。

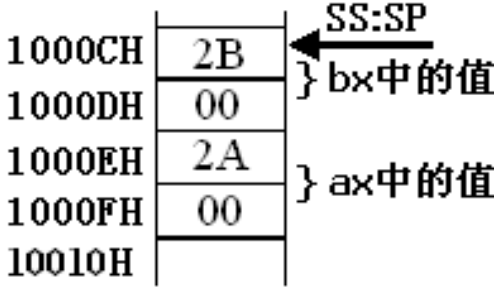
- 思考后看分析。

问题3.9分析

```
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。
```



(a)栈初始化的情况



(b) ax、bx入栈的情况

3.9 push、pop指令

- 问题3.10 代码1实现了在10000H处写入字型数据2266H功能。

代码1

```
mov ax, 1000H
mov ds, ax
mov ax, 2266H
mov [0], ax
```

代码2

```
_____
_____
_____
mov ax,2266H
push ax
```

补全代码2，完成相同功能。

要求：不能使用“mov [内存单元], 寄存器”

问题3.10分析

mov ax,1000H

mov ss,ax

mov sp,2

mov ax,2266H

push ax

用push 写内存10000H，所以要设置好栈的CS：SP，若CS=1000H则，SP=0002H

问题3.10分析（续）

- push/pop 与 mov的比较
  - push、pop 实质上就是一种内存传送指令，可以在寄存器和内存之间传送数据，与mov指令不同的是，push和pop指令访问的内存单元的地址不是在指令中给出的，而是由SS:SP指出的。
  - push和pop指令会改变 SP 中的内容。

问题3.10分析（续）

- push和pop指令同mov指令不同：
  - 执行mov指令只需一步操作——传送
  - 执行push、pop指令却需要两步操作。
    - push：先改变SP，后向SS:SP处传送。
    - pop：先读取SS:SP处的数据，后改变SP。

注意

- push、pop 等栈操作指令，修改的只是SP。也就是说，栈顶的变化范围最大为：0~FFFFH。

- 8086CPU提供的栈操作机制：
  - SS:SP指示栈顶；
  - 入栈指令push: 改变SP后写栈内存；
  - 出栈指令pop：读栈内存后改变SP。

栈的综述

- （1）8086CPU提供了**栈**操作机制：

  栈顶的段地址和偏移地址：SS:SP ；

  push / pop 对栈操作

- （2）push指令的执行步骤：

  1）SP=SP-2；

  2）向SS:SP指向的字单元中送入数据。

- （3）pop指令的执行步骤：

  1）从SS:SP指向的字单元中读取数据；

  2）SP=SP+2。

栈的综述（续）

- （4）任意时刻，SS:SP指向栈顶元素。
- （5）8086CPU只记录栈顶，栈空间大小由程序员管理。
- （6）用栈来暂存以后需要恢复的寄存器的内容时，寄存器出栈的顺序要和入栈的顺序相反。
- （7）push、pop实质上是一种内存传送指令，注意它们的灵活应用。
- 栈是一种非常重要的机制，一定要深入理解，灵活掌握。

3.10 栈段

- 8086PC机编程时，程序员可将起始地址为16的倍数一组连续的长度为 N（N ≤64K ）的内存单元定义为一个段——用作栈时，即为一个栈段。
  - 如 10010H~1001FH 内存的 段地址为1000H，大小为16字节。

3.10 栈段

- 如何使的如push、pop 等栈操作指令访问我们定义的栈段呢？

  将SS:SP指向我们定义的栈段。

3.10 栈段

- 问题3.11
- 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=？

- 思考后看分析。

问题3.11分析

分析：

10000H~1FFFFH当作栈段，若SS=1000H，栈底部的字单元的偏移地址为FFFEH，

初始 PS = 栈底偏移地址 + 2

所以 PS = 0000H。（最高位进位超过16位宽）

问题3.12

- 一个栈段最大可以设为多少？为什么？

- 思考后看分析。

问题3.12分析

- 一个栈段最大可以设为多少？为什么？

答：

- 因push、pop等在执行时只修改SP，所以栈顶的变化范围是0~FFFFH
- 从栈空时候的SP=0，一直入栈到栈满。栈满后再次入栈出现什么问题？
  - 栈满时SP = 0，栈顶将环绕，再次入栈覆盖了原来栈中的内容。
- 所以一个栈段的容量最大为64KB。

段的综述

- 将一段内存定义为一个段，用段地址指示段，用偏移地址访问段内单元。
- 如何定义段在内存中的位置，完全是程序员的安排。

- 需要哪些段？
  - 一个段存放数据——“数据段”；
  - 一个段存放代码——“代码段”；
  - 一个段当作栈——“栈段”；

段的综述（续）

- 我们可以这样安排，但若要让CPU按照我们的安排来访问这些段，就要：

  - 对于数据段，
    - DS 中存放段地址

用mov、add、sub等访问内存单元的指令时，CPU就将我们定义的数据段中的内容当作数据段来访问；

段的综述（续）

- 对于代码段
  - CS中存放段地址
  - IP中存放段中**第一条指令**的偏移地址

这样CPU就将执行程序员定义的代码段中的指令，

- jmp 指令会修改IP或者CS:IP；

段的综述（续）

- 对于栈段，
  - SS 中存放段地址，
  - SP 中存放**栈顶单元**的偏移地置 ，

这样CPU在需要进行push、pop 时， 将程序员定义的栈段当作栈空间来用。

- push、pop 会修改SP的值

段的综述（续）

- 代码段、数据段、栈段在内存中的位置都是程序员的安排。
- 要让CPU按安排运行，需先设置对应寄存器：
  - CPU 将内存中的某段内存当作代码，是因为设置好CS:IP指向了那里；
  - CPU将某段内存当作栈，是因为 SS:IP 指向了那里；
  - CPU将某段内存当做数据，时因为DS指向数据段的段地址。

段的综述（续）

- 若将10000H~1001FH为代码段，代码如下：

```
mov ax,1000H
mov ss, ax
mov sp,0020H
mov ax, cs
mov ds, ax
mov ax, [0]
add ax, [2]
mov bx, [4]
add bx, [6]
push ax
push bx
pop ax
pop bx
```

;初始化栈顶
10000H~1001FH 安排为栈段

;设置数据段段地址
10000H~1001FH 安排为数据段

10000H~1001FH这段内存，既是代码段，又是栈段和数据段。

段的综述（续）

- 一段内存，可以既是代码的存储空间，又是数据的存储空间，还可以是栈空间，也可以什么也不是。

- 关键在于CPU中寄存器的设置，即：

  CS、IP、SS、SP、DS的指向。

特别提示

- 检测点3.2 （p66）
- 没有通过检测点请不要向下学习！

Quiz

- 补全代码，使内存10000H~1000FH中的8个字，逆序复制到20000H~2000FH中

```
mov ax,2000H
mov ds,ax

_____

_____

_____

pop [E]
pop [C]
pop [A]
pop [8]
pop [6]
pop [4]
pop [2]
pop [0]
```