汇编语言程序设计
lesson 2

上节回顾

- 1、汇编语言是一种低级编程语言，是机器指令的助记符。
- 2、CPU可以直接使用的信息在存储器中存放。
- 3、在存储器中，指令和数据没有任何区别，都是二进制信息。
- 4、1个存储器有很多个存储单元，存储单元从0开始编号。1个存储单元可以存储8个bit，即8位二进制数。

上节回顾

- 5、CPU的3种总线：

  地址总线：总线宽带为N，代表CPU可以最多寻找2的N次方个内存单元；

  控制总线

  数据总线

- 6、微机硬件系统的基本组成。
- 7、CPU访问内存单元时，必须向内存提供内存单元的物理地址。
- 8、8086CPU在内部用段地址和偏移地址移位相加的方法形成最终的物理地址（实模式）。

上节回顾

- 9、上节学过的通用寄存器。
- EAX　　AX（accumulator）累加器
- EBX　　BX （base）基址寄存器
- ECX　　CX （count）计数器
- EDX　　DX （data）数据寄存器

第2章 寄存器

- 2.7 段寄存器
- 2.8 CS和IP
- 2.9 代码段

2.7 段寄存器

- 段寄存器就是提供段地址的寄存器：
  CS、SS、DS、ES、FS、GS。

  8086CPU有4个段寄存器：

  CS（ code segment ）

  DS（data segment ）

  SS（ stack segment ）

  ES （ extra segment ）
- 实模式中，当CPU要访问内存时，由这4个段寄存器提供内存单元的段地址。

2.8 CS和IP

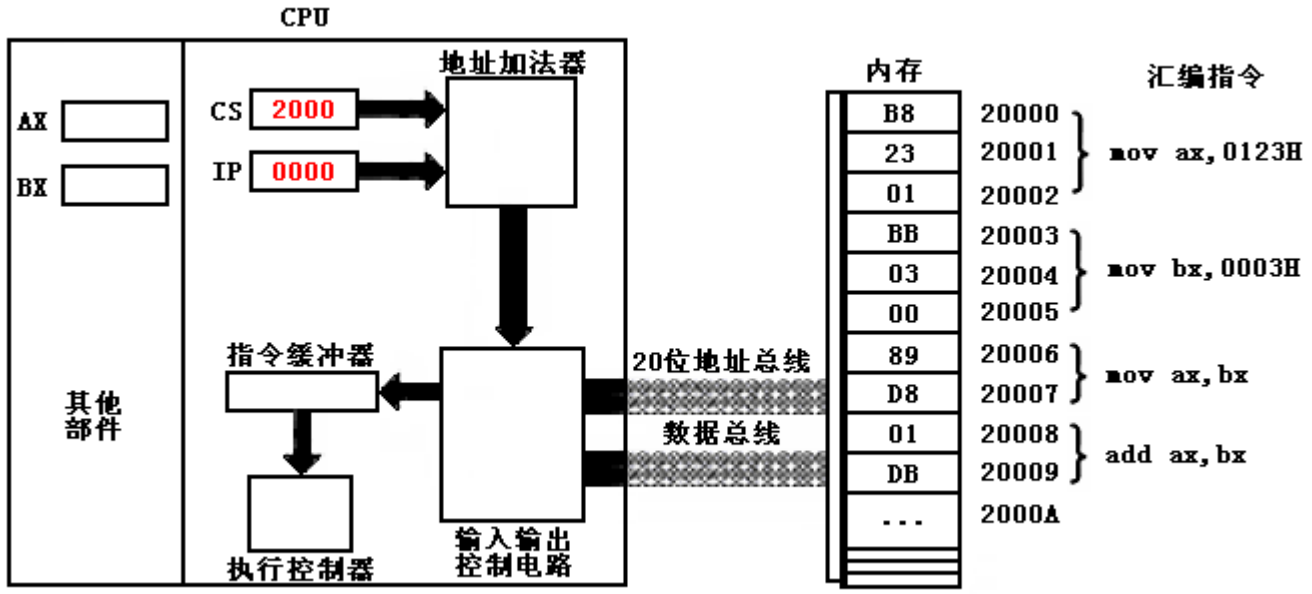
- CS和IP是8086CPU中最关键的寄存器，它们指示了CPU当前要读取指令的地址。

  CS为代码段寄存器；

  IP为指令指针寄存器。

- CPU将CS、IP中的内容当作指令的段地址和偏移地址，用它们合成指令的物理地址，到内存中读取指令码，执行。

8086PC读取和执行指令相关部件



- 8086PC读取和执行指令演示
- 8086PC工作过程的简要描述

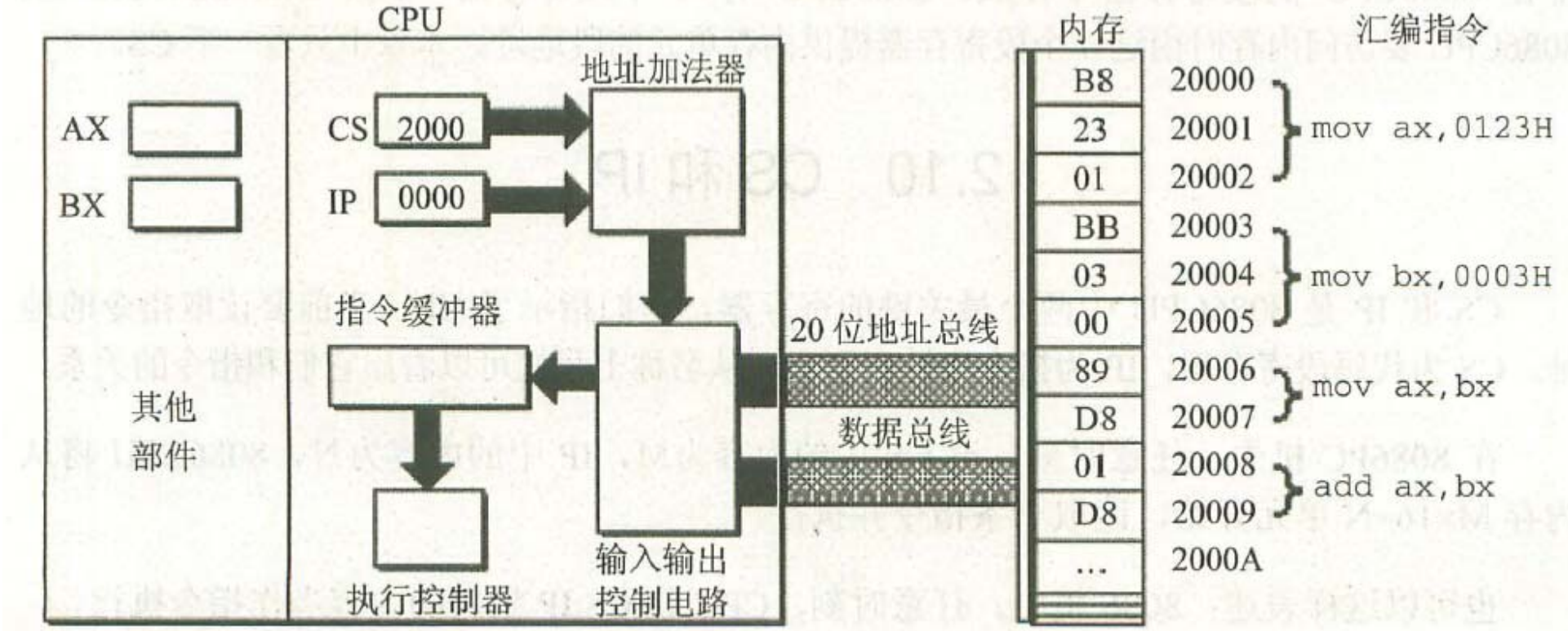


图 2.11　初始状态(CS:2000H，IP:0000H，CPU 将从内存 2000H×16+0000H 处读取指令执行)

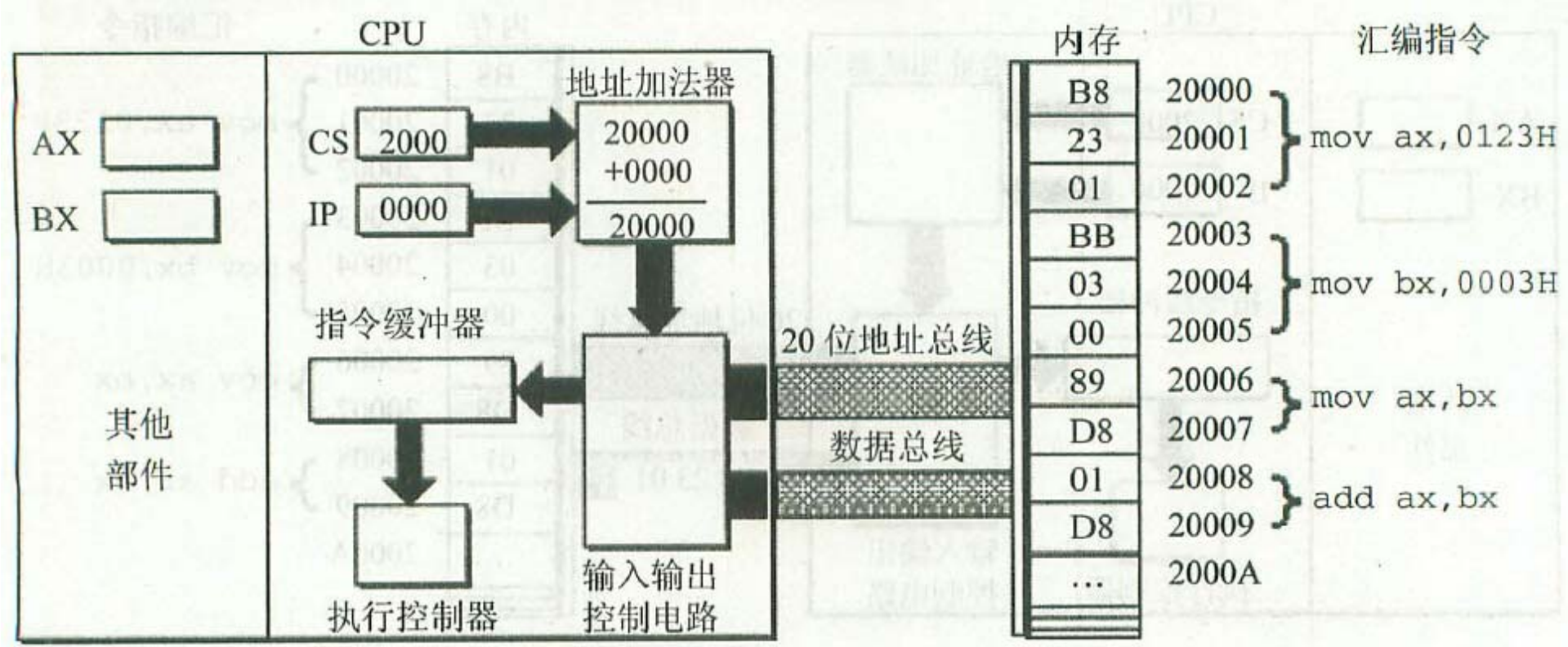


图 2.12　CS、IP 中的内容送入地址加法器(地址加法器完成：物理地址=段地址×16+偏移地址)

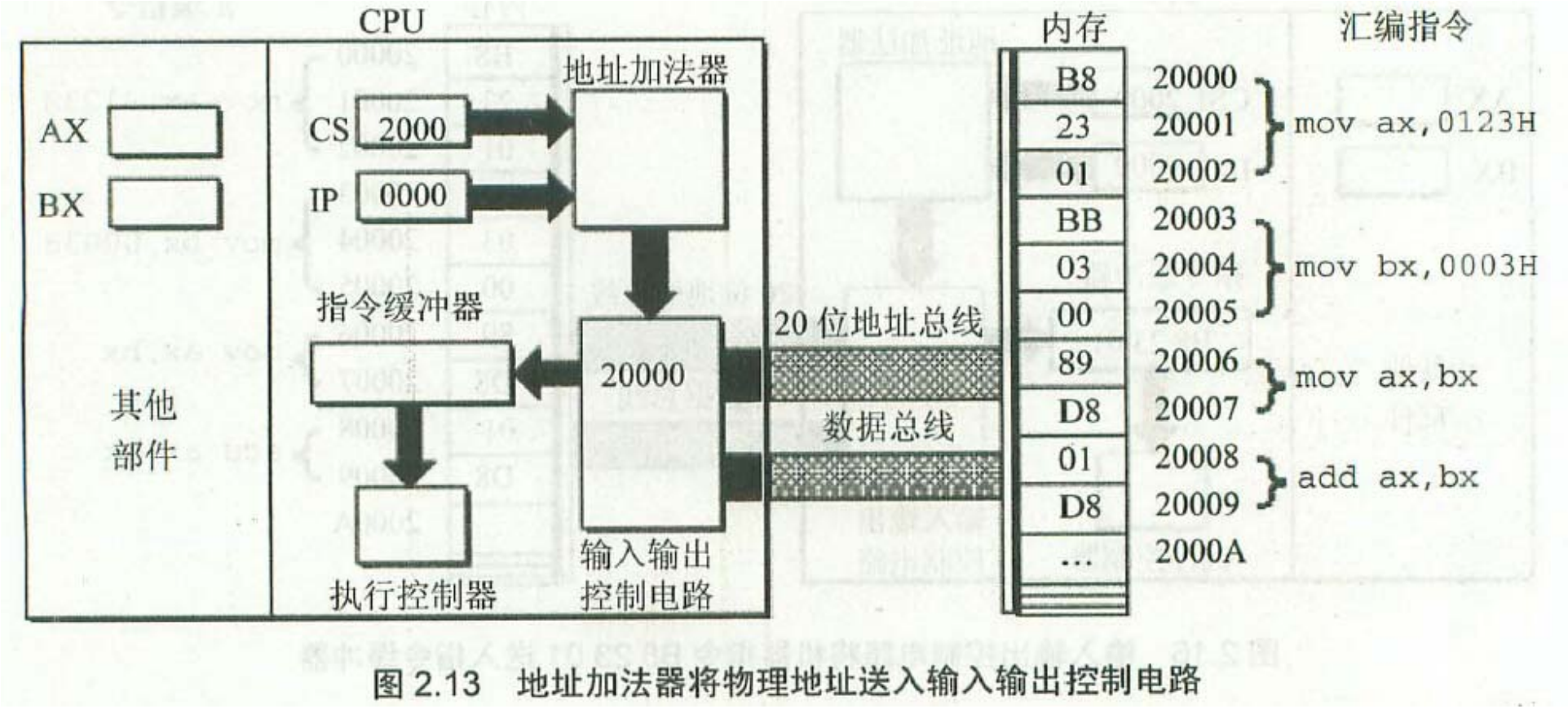


图 2.13　地址加法器将物理地址送入输入输出控制电路

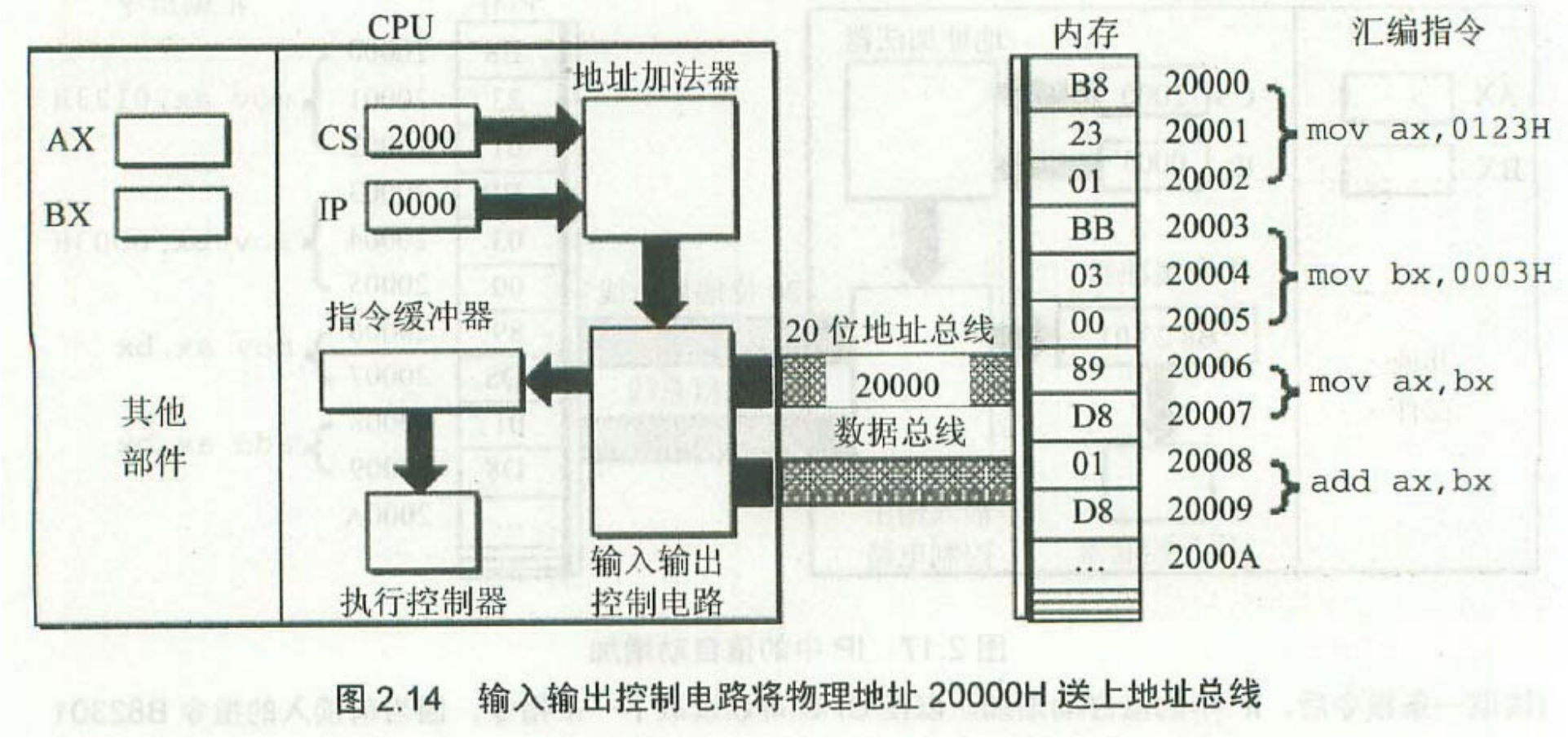


图 2.14　输入输出控制电路将物理地址 20000H 送上地址总线

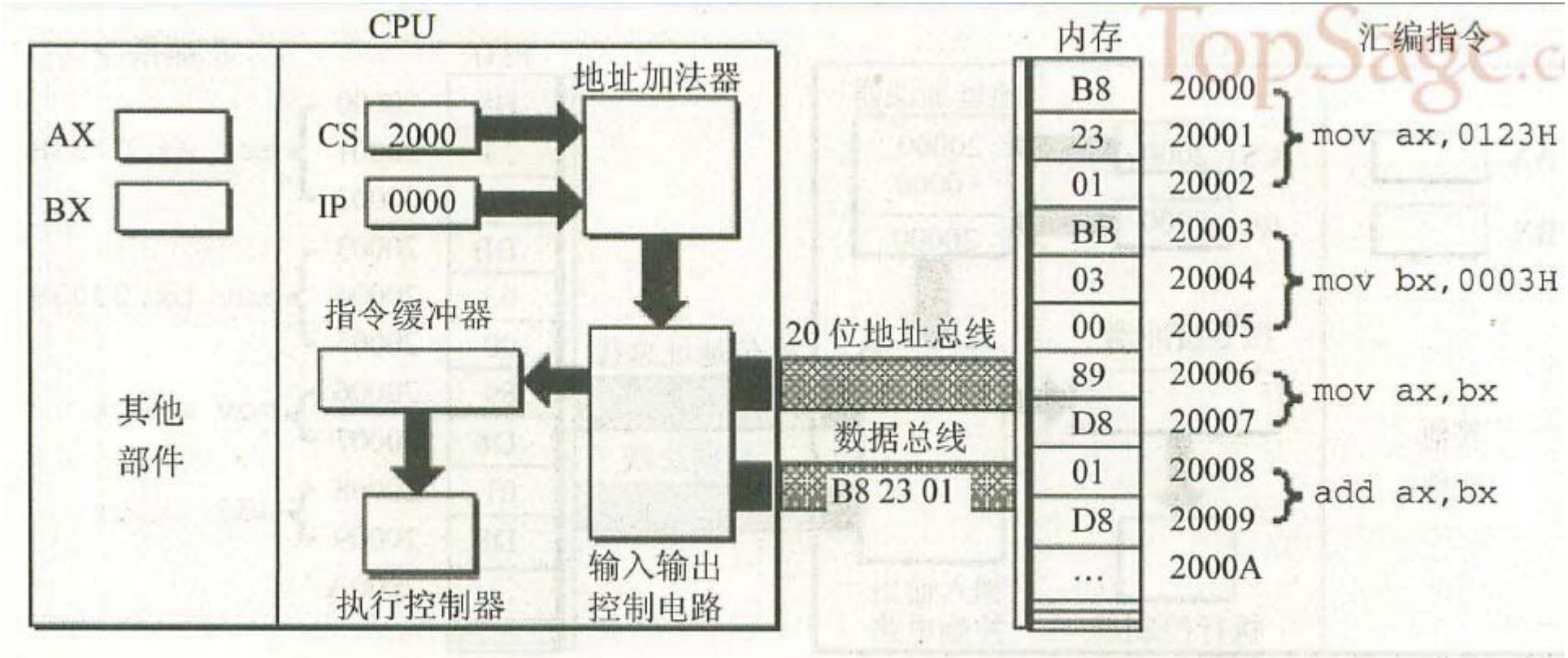


图 2.15　从内存 20000H 单元开始存放的机器指令 B8 23 01 通过数据总线被送入 CPU

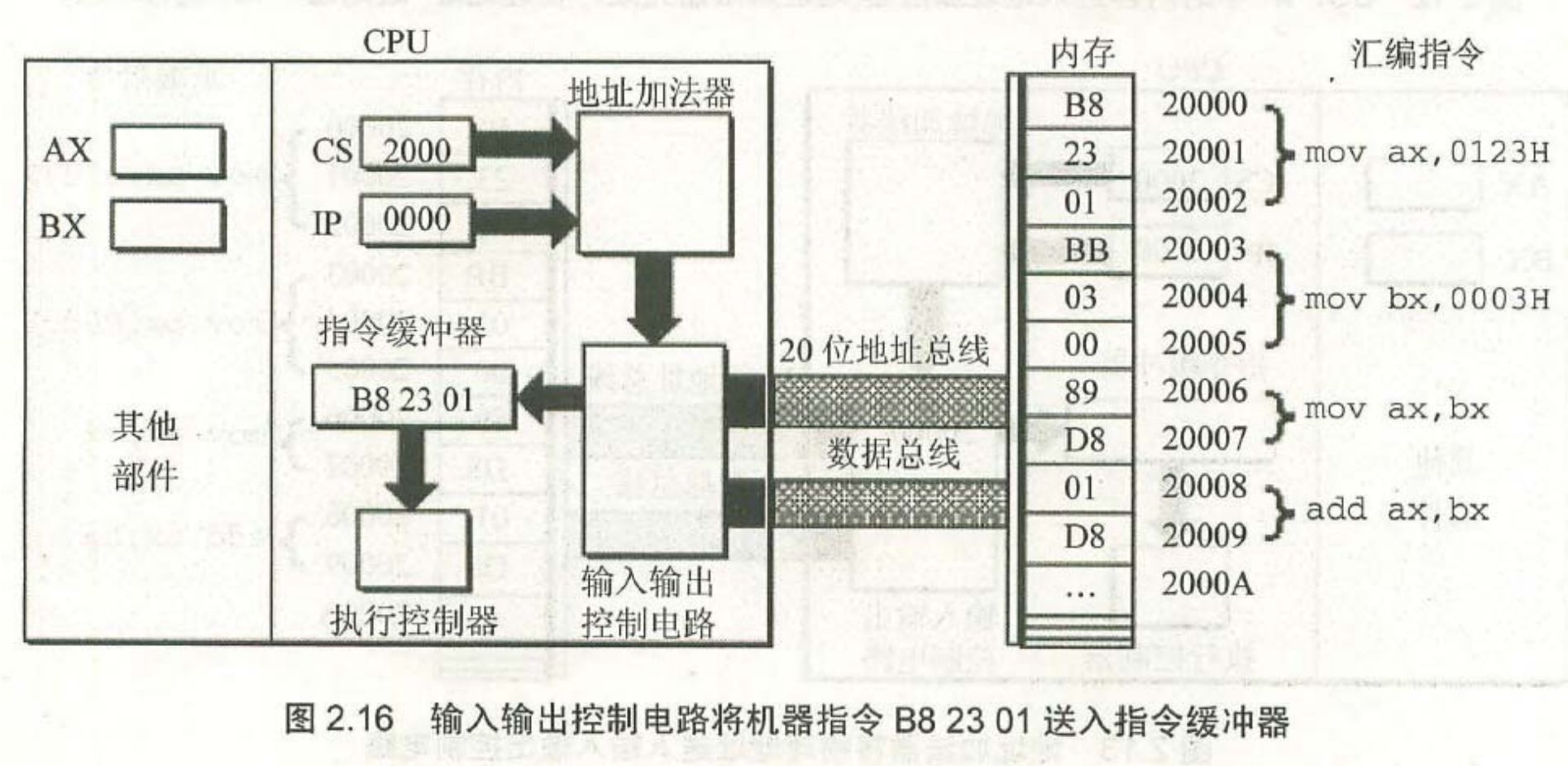


图 2.16　输入输出控制电路将机器指令 B8 23 01 送入指令缓冲器

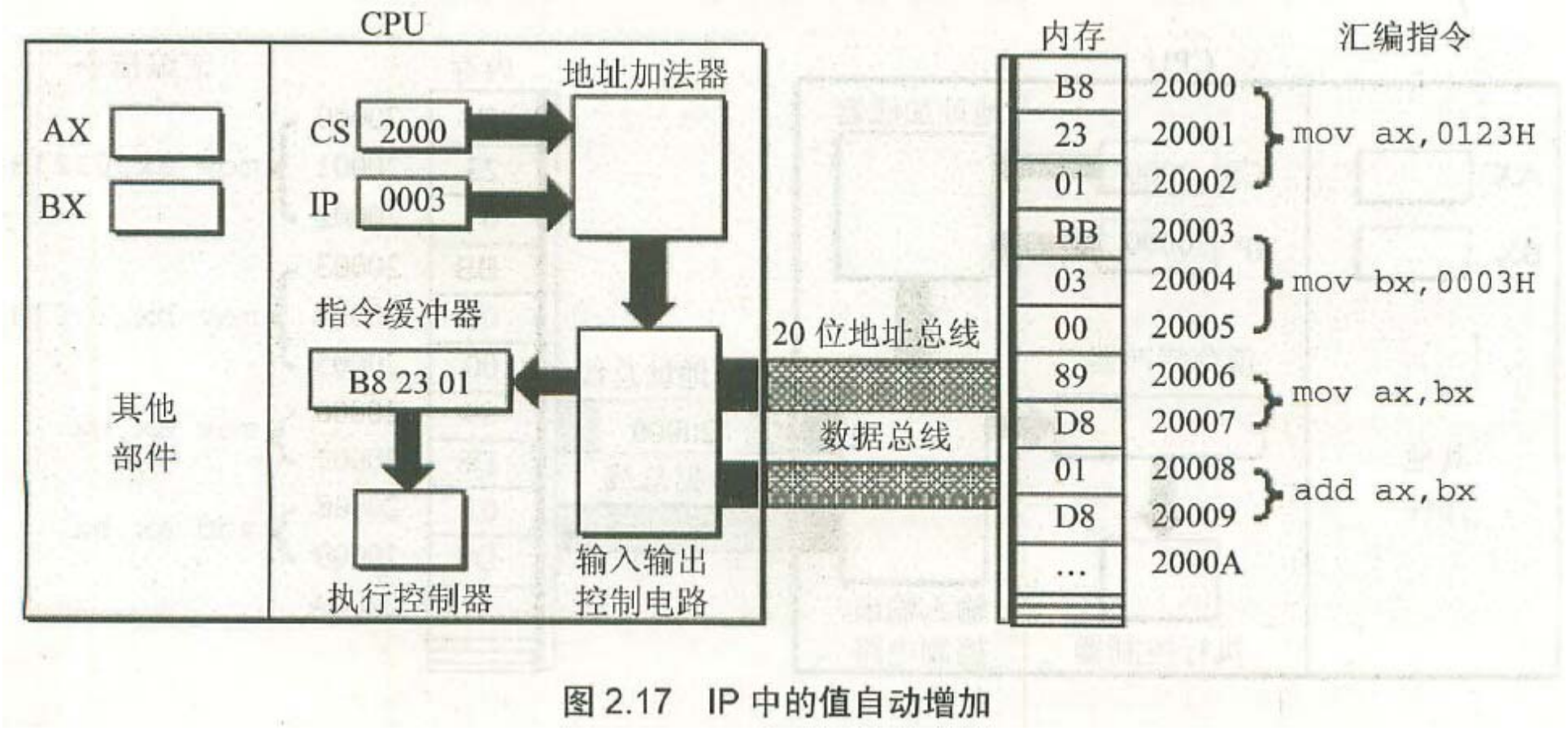


图 2.17　IP 中的值自动增加

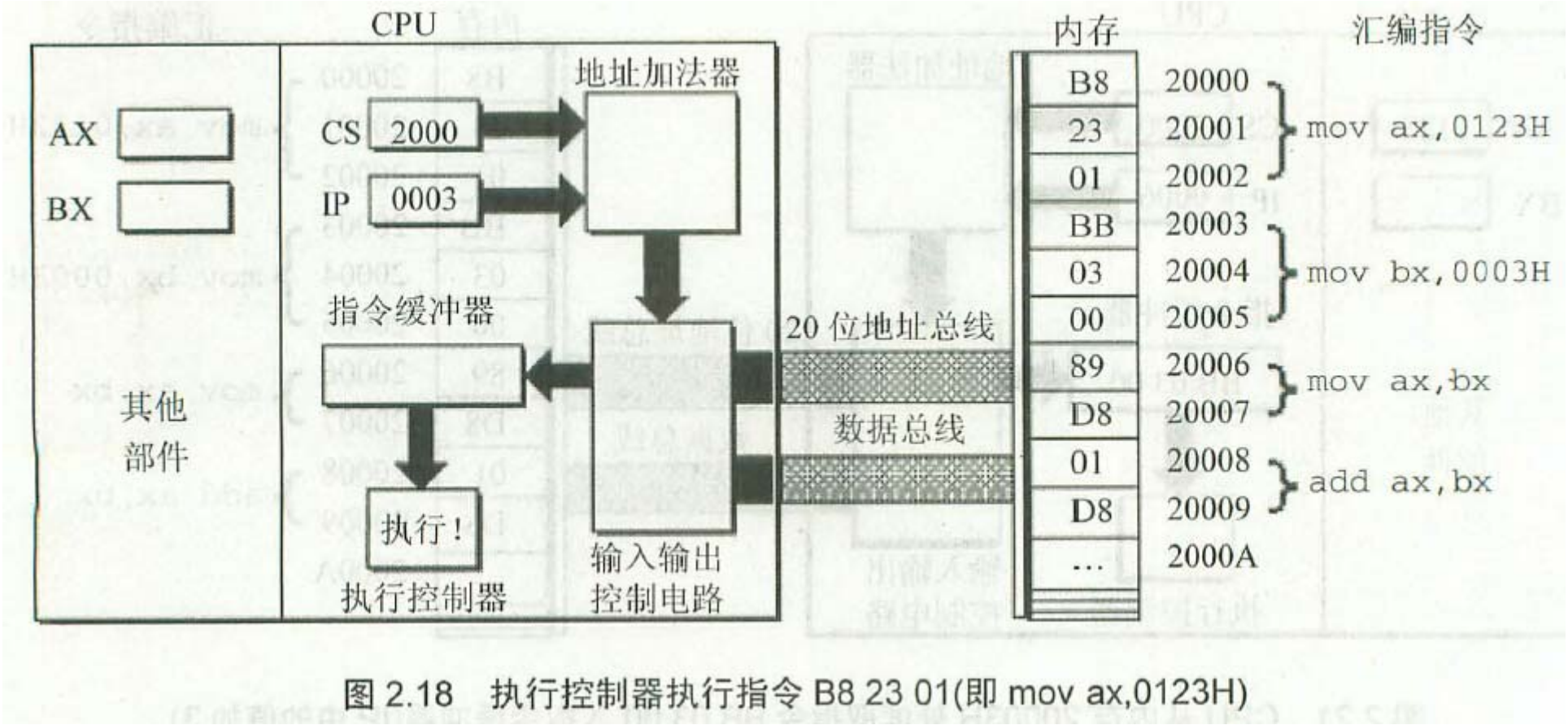


图 2.18　执行控制器执行指令 B8 23 01(即 mov ax,0123H)

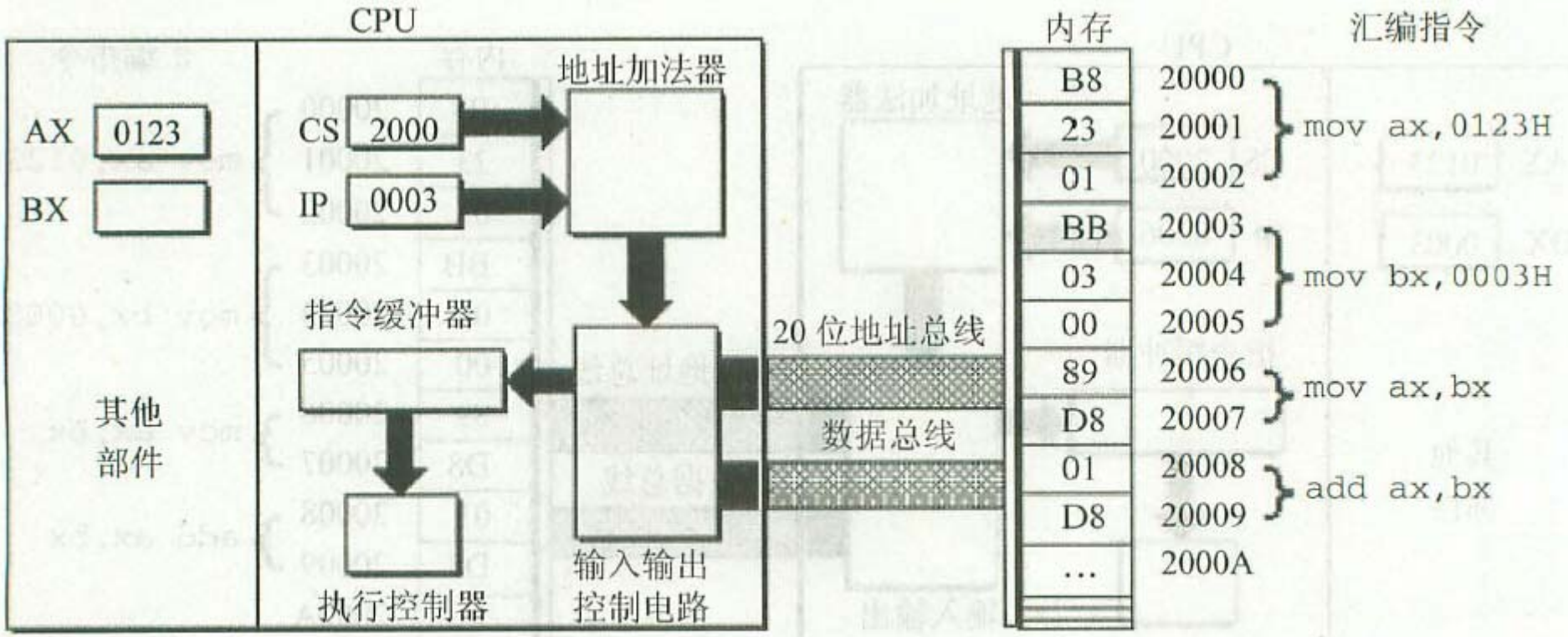


图 2.19　指令 B8 23 01 被执行后 AX 中的内容为 0123H

(此时，CPU 将从内存单元 2000:0003 处读取指令。)

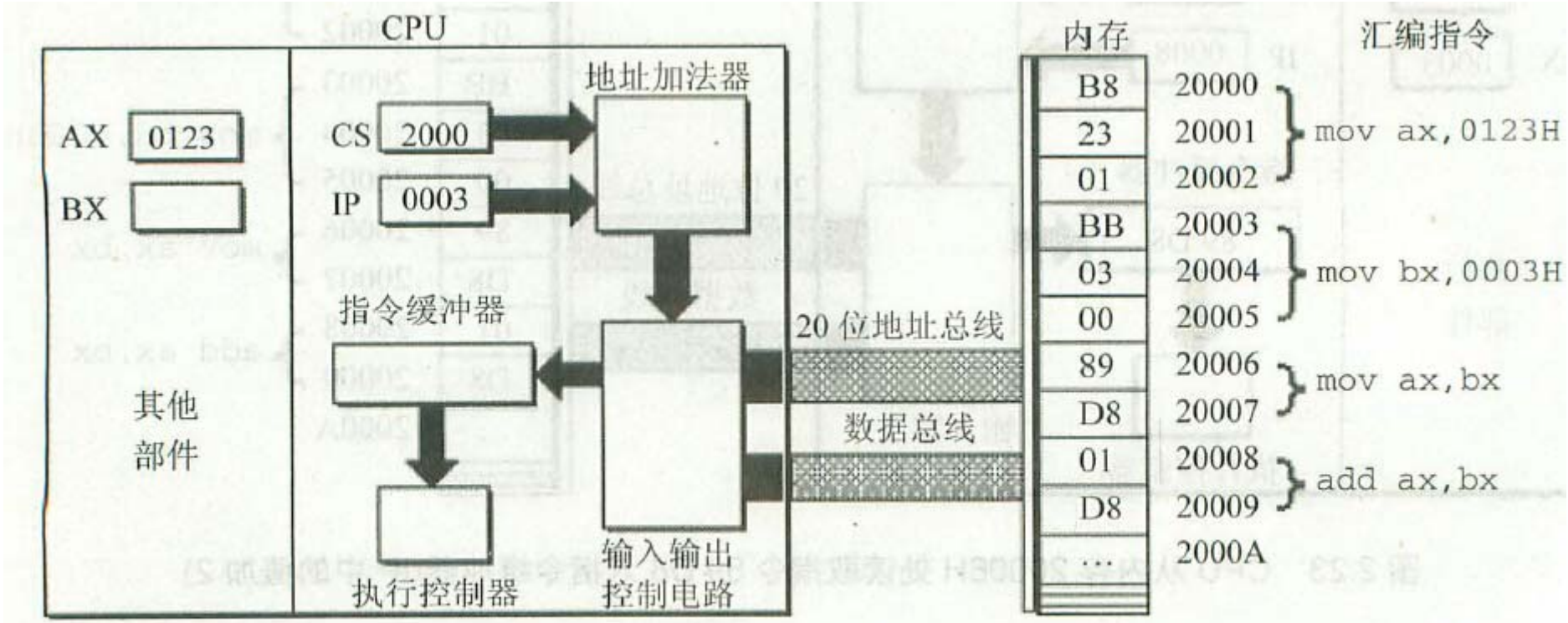


图 2.20　CS:2000H，IP:0003H(CPU 将从内存 2000H×16+0003H 处读取指令 BB 03 00)

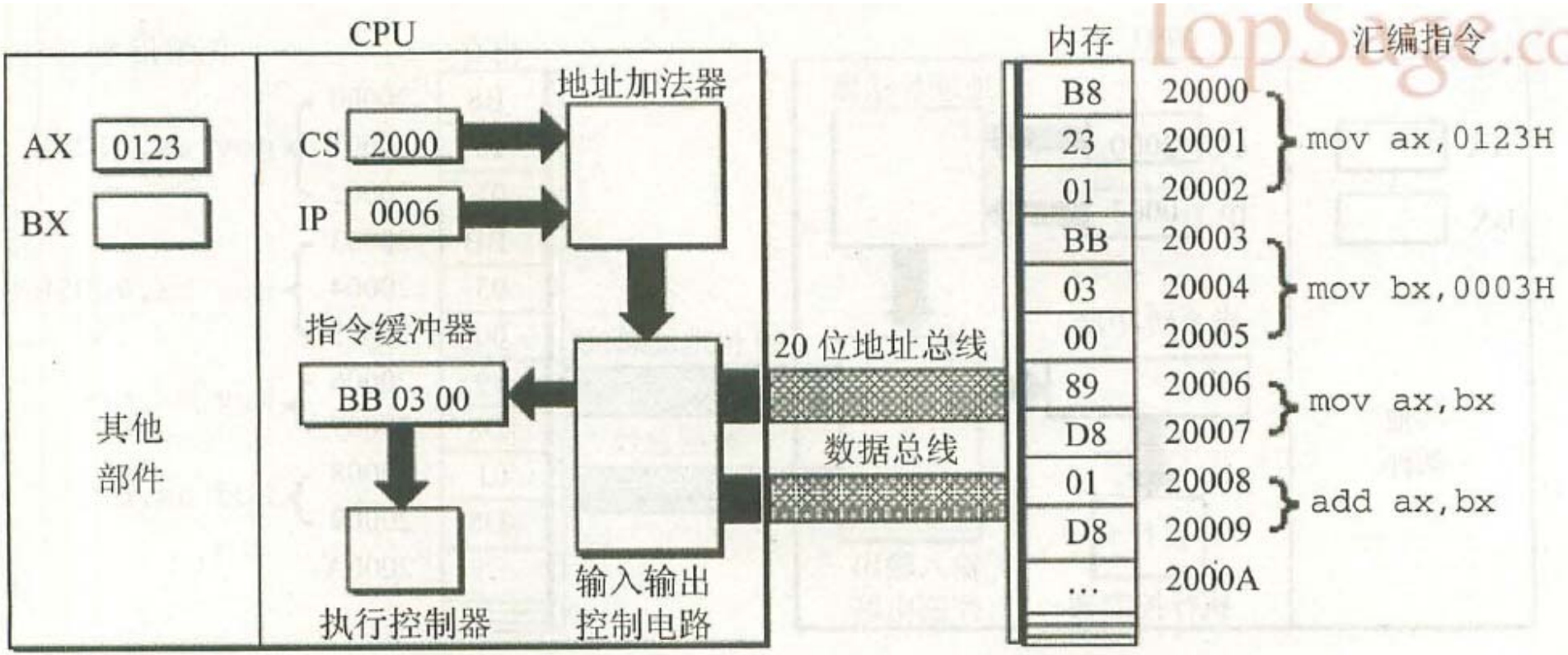


图 2.21　CPU 从内存 20003H 处读取指令 BB 03 00 入指令缓冲器(IP 中的值加 3)

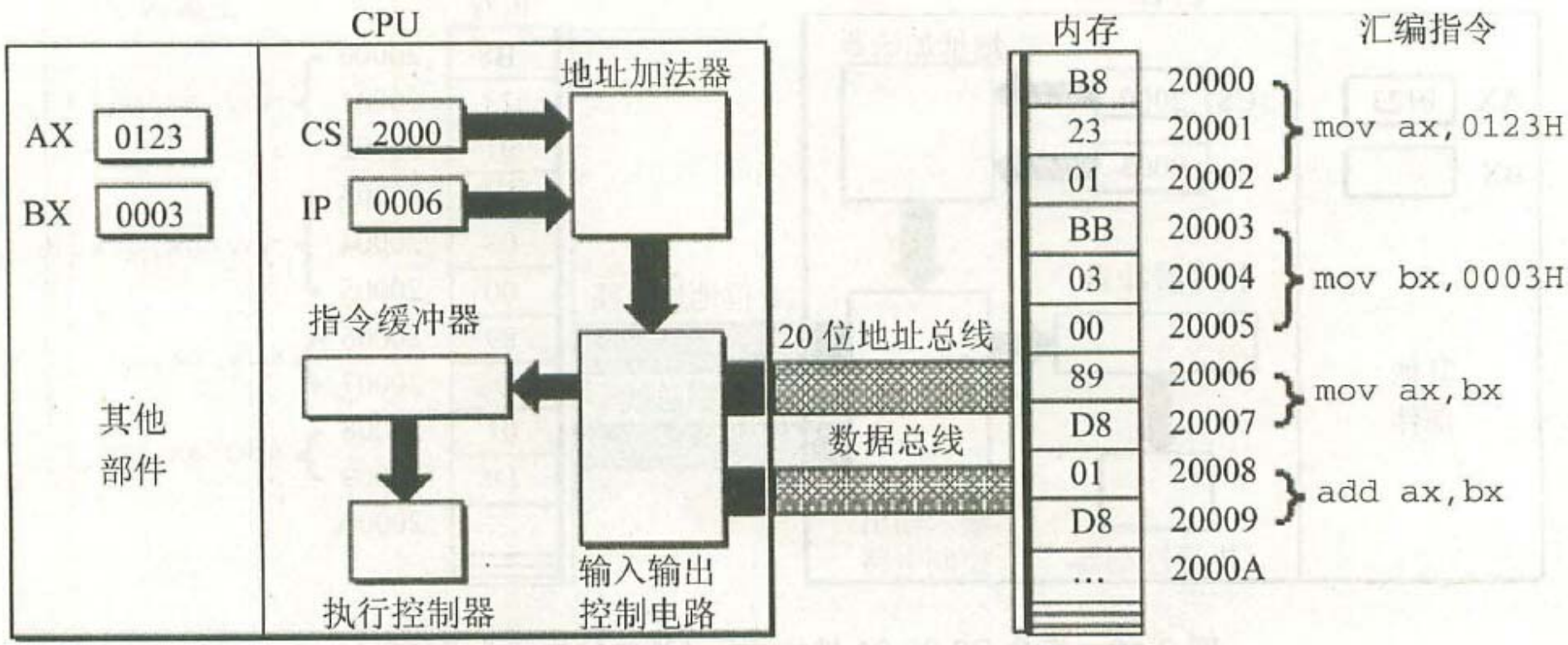


图 2.22　执行指令 BB 03 00(即 mov bx,0003H)

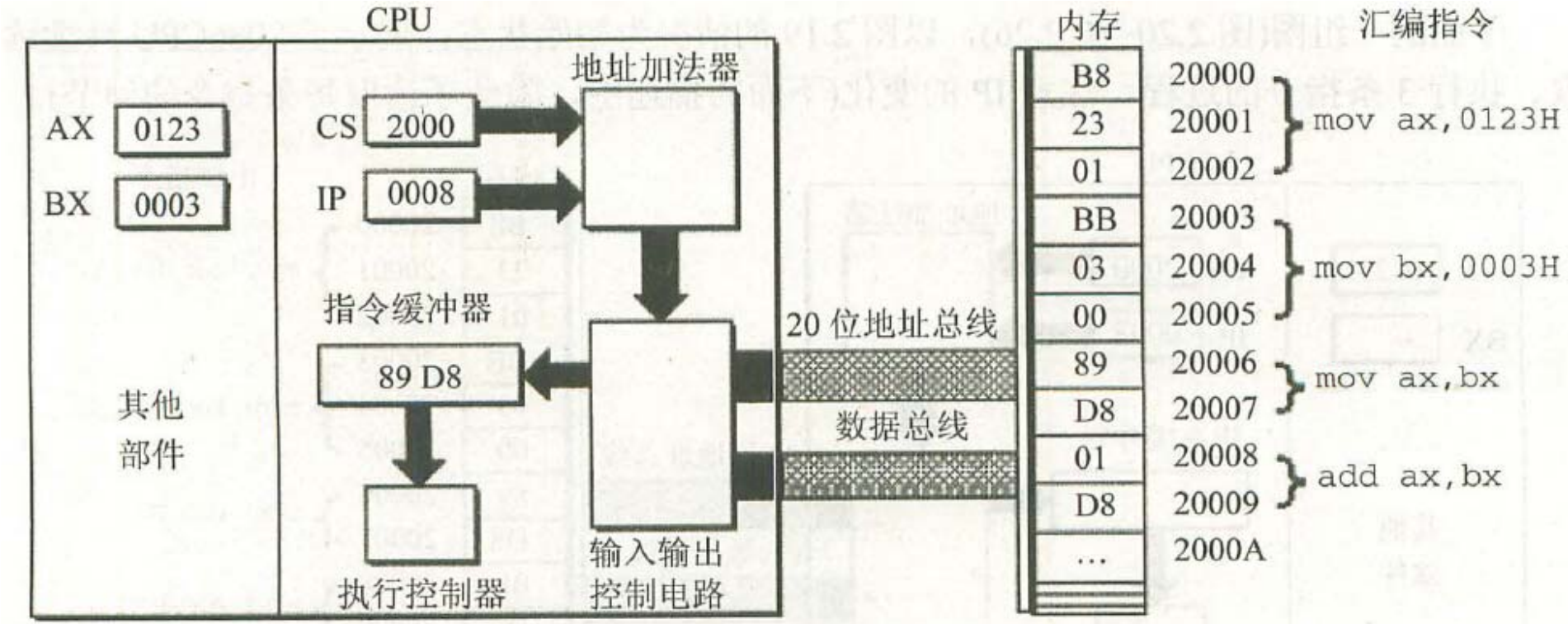


图 2.23　CPU 从内存 20006H 处读取指令 89 D8 入指令缓冲器(IP 中的值加 2)

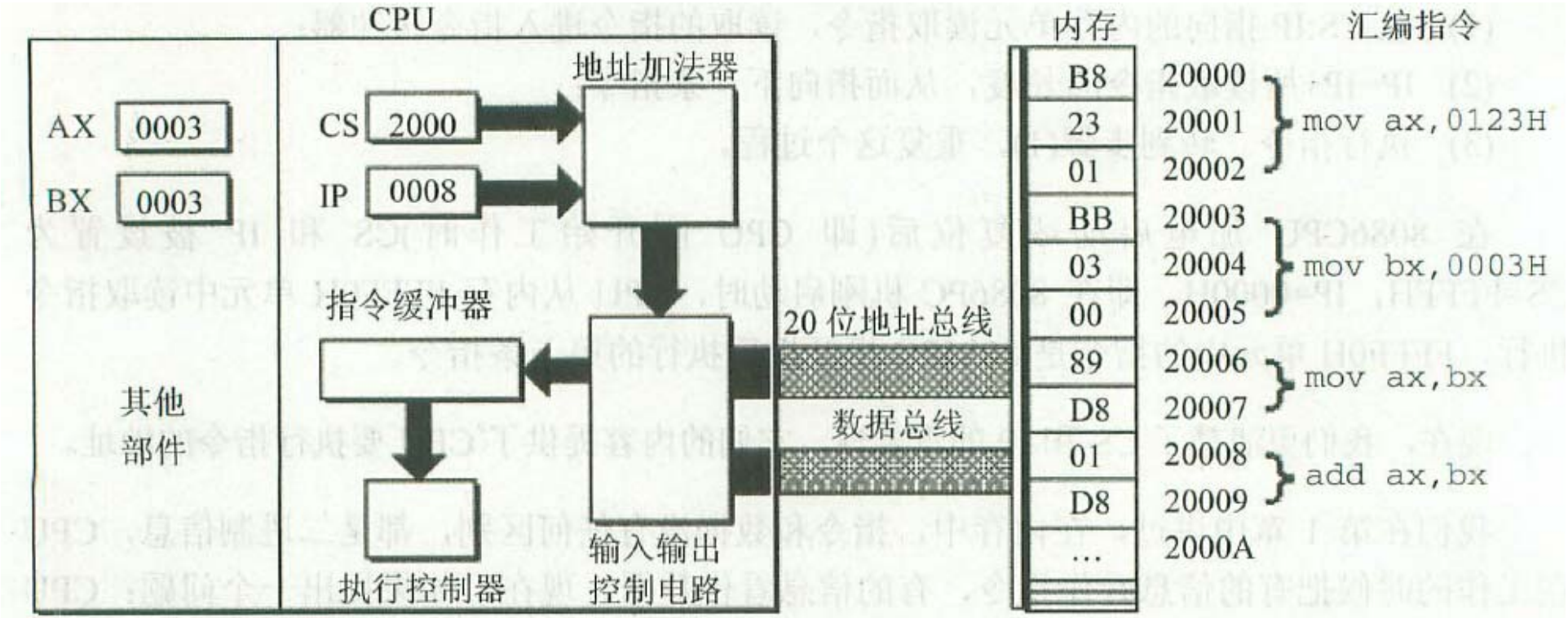


图 2.24　执行指令 89 D8(即 mov ax,bx)后，AX 中的内容为 0003H

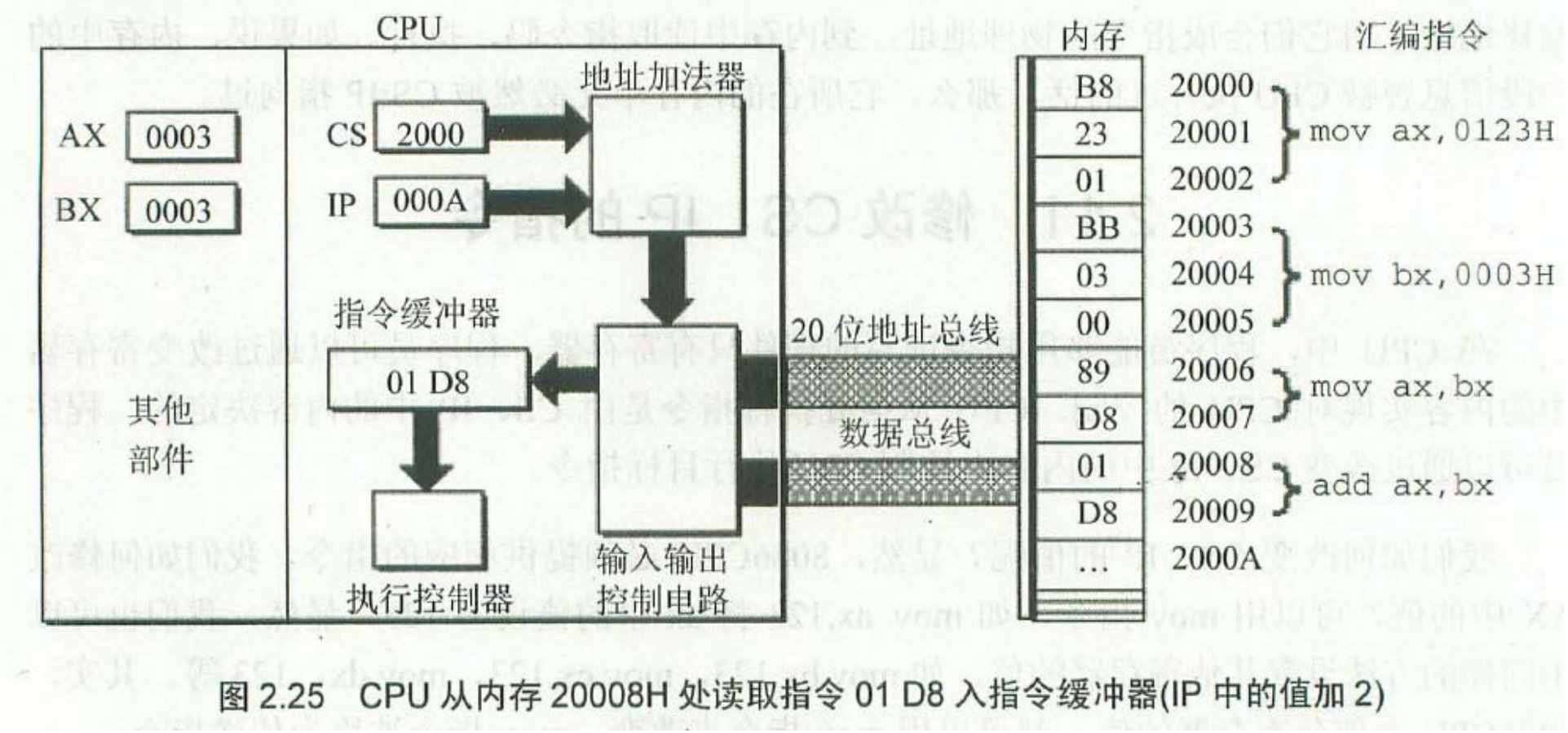


图 2.25　CPU 从内存 20008H 处读取指令 01 D8 入指令缓冲器(IP 中的值加 2)

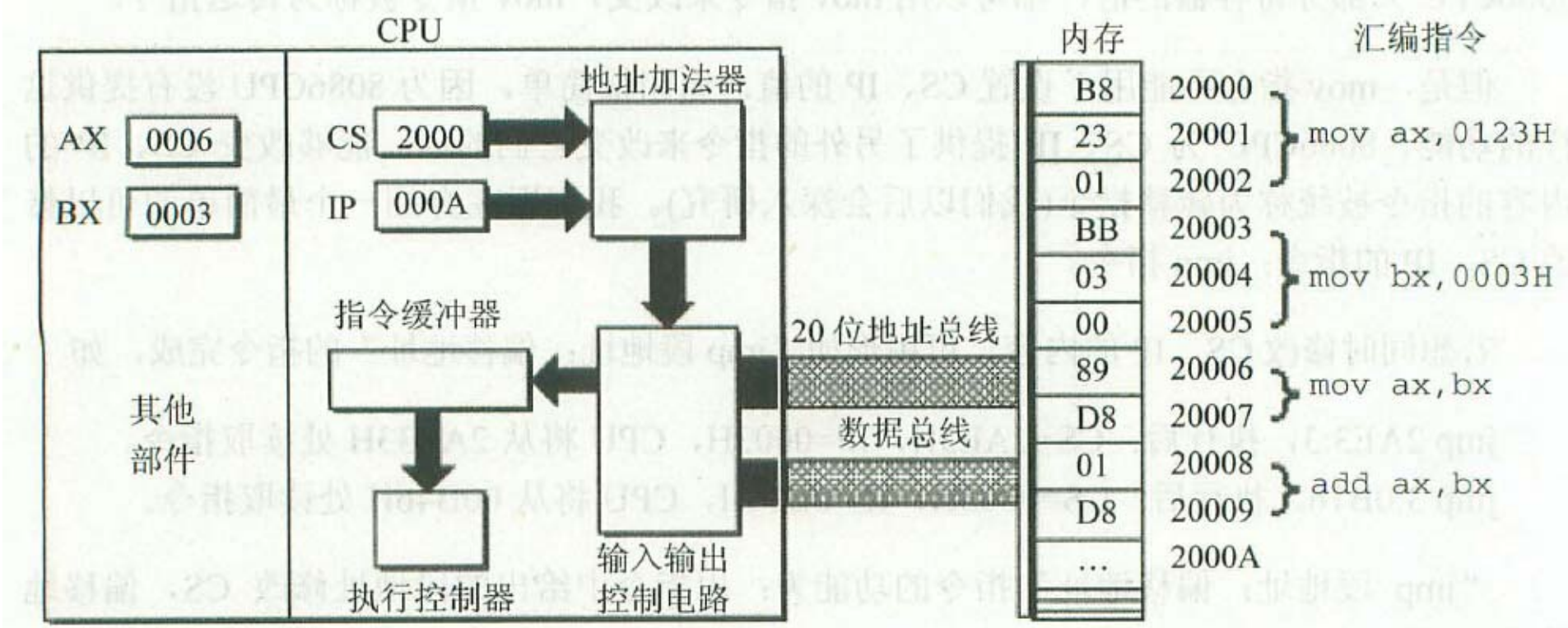


图 2.26　执行指令 01 D8(即 add ax,bx)后，AX 中的内容为 0006H

8086PC工作过程的简要描述

- （1）从CS:IP指向内存单元读取指令，读取的指令进入指令缓冲器；
- （2）IP = IP + 所读取指令的长度，从而指向下一条指令；
- （3）执行指令。 转到步骤 （1），重复这个过程。

8086PC工作过程的简要描述

- 在 8086CPU 加电启动或复位后（ 即CPU刚开始工作时）CS和IP被设置为CS=FFFFH，IP=0000H，即在8086PC机刚启动时，CPU从内存FFFF0H单元中读取指令执行，FFFF0H单元中的指令是8086PC机开机后执行的第一条指令。

2.8 CS和IP-修改

- 如何修改CS、IP的值呢？
- 通过转移指令。
  - 同时修改CS、IP的内容：

    jmp 段地址：偏移地址

    jmp 2AE3:3

    jmp 3:0B16

    功能：用指令中给出的段地址修改CS，偏移地址修改IP。

2.8 CS和IP-修改

- 仅修改IP的内容：

  jmp 某一合法寄存器

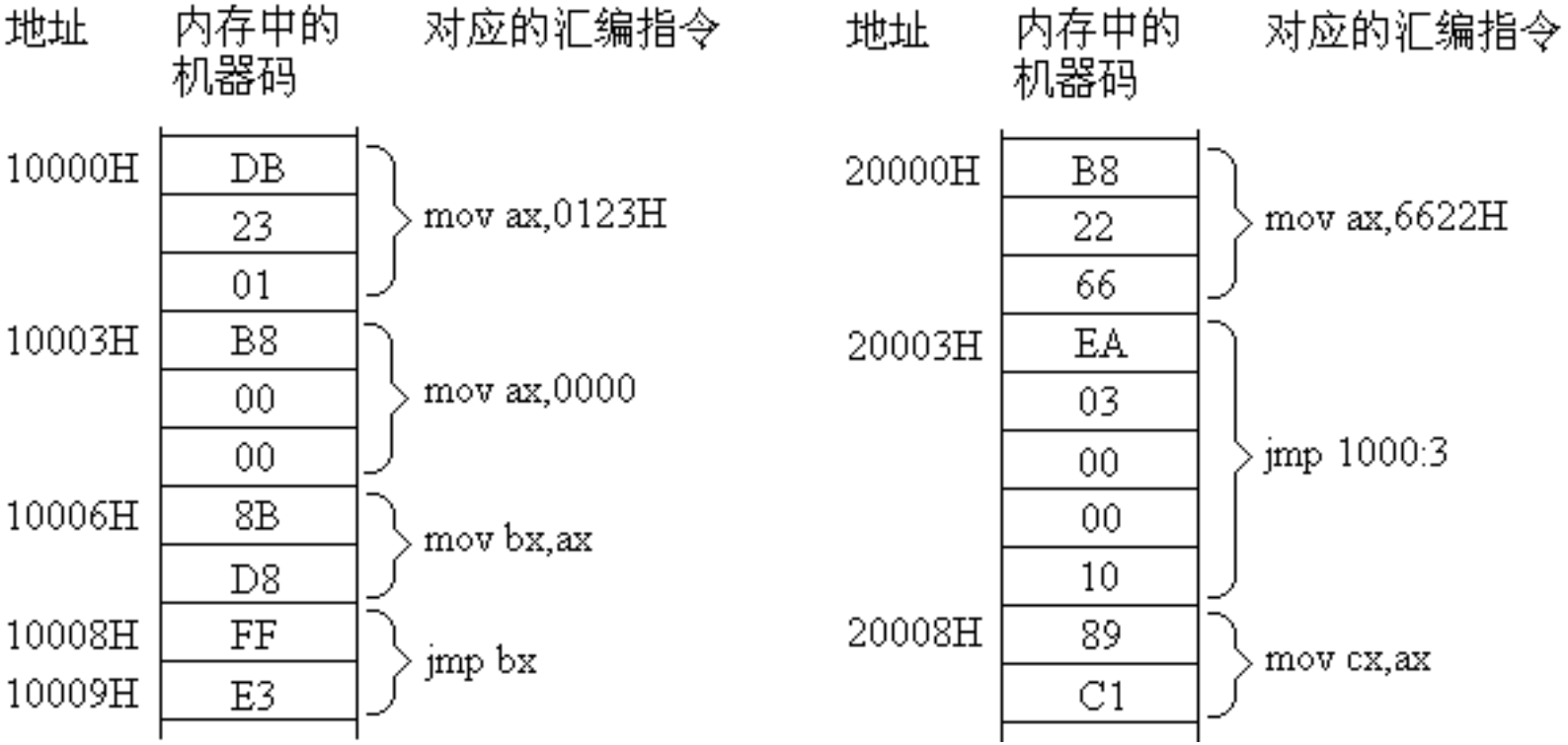
  jmp ax （类似于 mov IP,ax）

  jmp bx

  功能：用寄存器中的值修改IP。

问题分析

- 内存中存放的机器码和对应汇编指令情况：
  （初始：CS=2000H，IP=0000H）



- 请写出指令执行序列：

问题分析结果：

- （1）mov ax,6622
- （2）jmp 1000:3
- （3）mov ax,0000
- （4）mov bx,ax
- （5）jmp bx
- （6）mov ax,0123H
- （7）转到第（3）步执行

2.9 代码段

- 对于8086PC机，在编程时，可以根据需要，将一组内存单元定义为一个段。
- 可以将长度为 N（ N≤64KB ）的一组代码，存在一组地址连续、起始地址为 16 的倍数的内存单元中，这段内存是用来存放代码的，从而定义了一个代码段。

- 例如

2.9 代码段

mov ax,0000	(B8 00 00)
add ax,0123	(05 23 01)
mov bx,ax	(8B D8)
jmp bx	(FF E3)

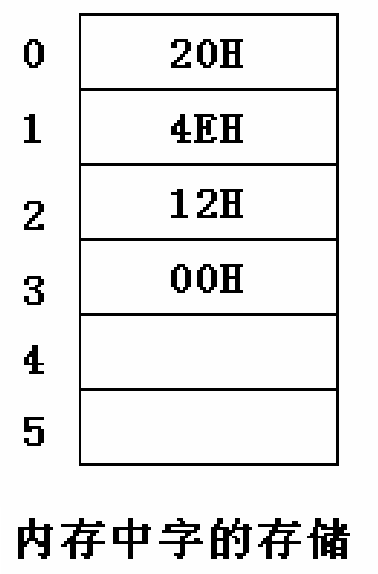
- 这段长度为 10 字节的字节的指令，存在从123B0H~123B9H的一组内存单元中，我们就可以认为，123B0H~123B9H这段内存单元是用来存放代码的，是一个代码段，它的段地址为123BH，长度为10字节。
- 需要将CS:IP指向所定义的代码段中的第一条指令的首地址。使得代码段中的指令被执行呢？

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

3.1 内存中字的存储

- 在0地址处开始存放20000：

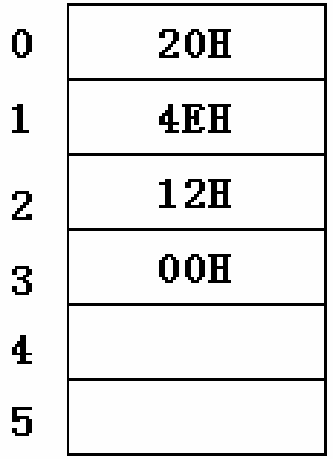


内存中字的存储

- 0号单元是低地址单元，1号单元是高地址单元。

3.1 内存中字的存储

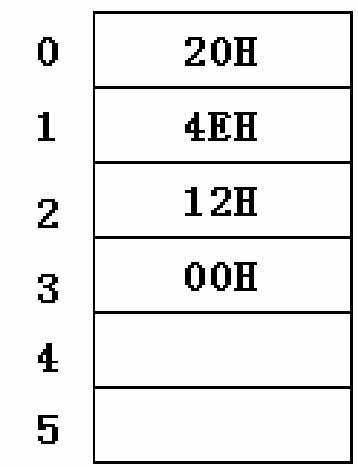
- 问题：
  - （1）0地址单元中存放的字节型数据是多少？
  - （2）0地址字单元中存放的字型数据是多少？
  - （3）2地址单元中存放的字节型数据是多少？



内存中字的存储

3.1 内存中字的存储

- 问题（续）：
  - （4）2地址单元中存放的字型数据是多少？
  - （5）1地址字单元中存放的字型数据是多少？



内存中字的存储

- 结论

3.1 内存中字的存储

- 结论：

  任何两个地址连续的内存单元，N号单元和 N+1号单元，可以将它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。

- 8086CPU中段寄存器DS通常用来存放要访问的数据的段地址。

3.2 DS和[address]

- 例如：我们要读取10000H单元的内容可以用如下程序段进行：

```
mov bx,1000H
mov ds,bx
mov al,[0]
```

- 上面三条指令将10000H（1000:0）中的数据读到al中。

3.2 DS和[address]

mov指令小结：

- 已知的mov指令可完成的两种传送功能：
- （1）将数据直接送入寄存器；
- （2）将一个寄存器中的内容送入另一个寄存器中。
- mov 指令 还可以将一个内存单元中的内容送入一个寄存器。
- mov指令的格式：

  mov 寄存器名，内存单元地址

3.2 DS和[address]

- 是否可以把1000H直接送入ds?
  - 传送指令 mov ax,1
  - 相似的方式 mov ds,1000H?
  - 8086CPU不支持将数据直接送入段寄存器的操作，ds是一个段寄存器。

    （硬件设计的问题）
  - mov ds,1000H 是非法的。
  - 数据➔一般的寄存器➔段寄存器

3.2 DS和[address]

- 问题：

  怎样将数据从寄存器送入内存单元？

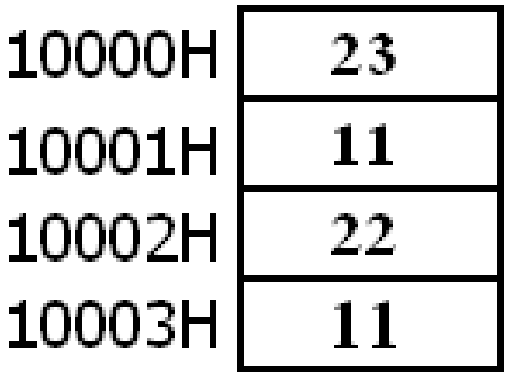
  8086CPU是16位结构，有16根数据线，所以，可以一次性传送16位的数据，也就是一次性传送一个字。

```
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
```

3.3 字的传送

- 问题3.4：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
```



内存情况示意

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

- 根据已知指令进行推测

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？（验证）
  - 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？

验证（Debug）

- mov 段寄存器，寄存器

  ➔ mov 寄存器，段寄存器

```
C:\>debug
-a
073F:0100 mov ax,ds
073F:0102
-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   NV UP EI PL NZ NA PO NC
073F:0100 8CD8          MOV     AX,DS
-t

AX=073F  BX=0000  CX=0000  DX=0000  SP=00FD  BP=0000  SI=0000  DI=0000
DS=073F  ES=073F  SS=073F  CS=073F  IP=0102   NV UP EI PL NZ NA PO NC
073F:0102 0000          ADD     [BX+SI],AL                      DS:0000=CD
-
```

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）

3.5 数据段

- 我们可以将一组长度为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，从而定义了一个数据段。
- 比如我们用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？
- 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
```

3.6 栈

- 栈是一种具有特殊的访问方式的存储空间。最后进入这个空间的数据，最先出去。
- 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入栈，需要出栈时，又最先被从栈中取出。
- 栈的操作规则：LIFO

  （Last In First Out，后进先出）

3.7 CPU提供的栈机制

- 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。
- 8086CPU的入栈和出栈操作都是以字为单位进行的。

3.6 栈

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

ax=1122H
bx=2266H
cx=0123H

两个疑问

- 1、CPU如何知道一段内存空间被当作栈使用？
- 2、执行push和pop的时候，如何知道哪个单元是栈顶单元？
- 分析

结论：8086CPU中，有两个寄存器：

段寄存器SS　　存放栈顶的段地址

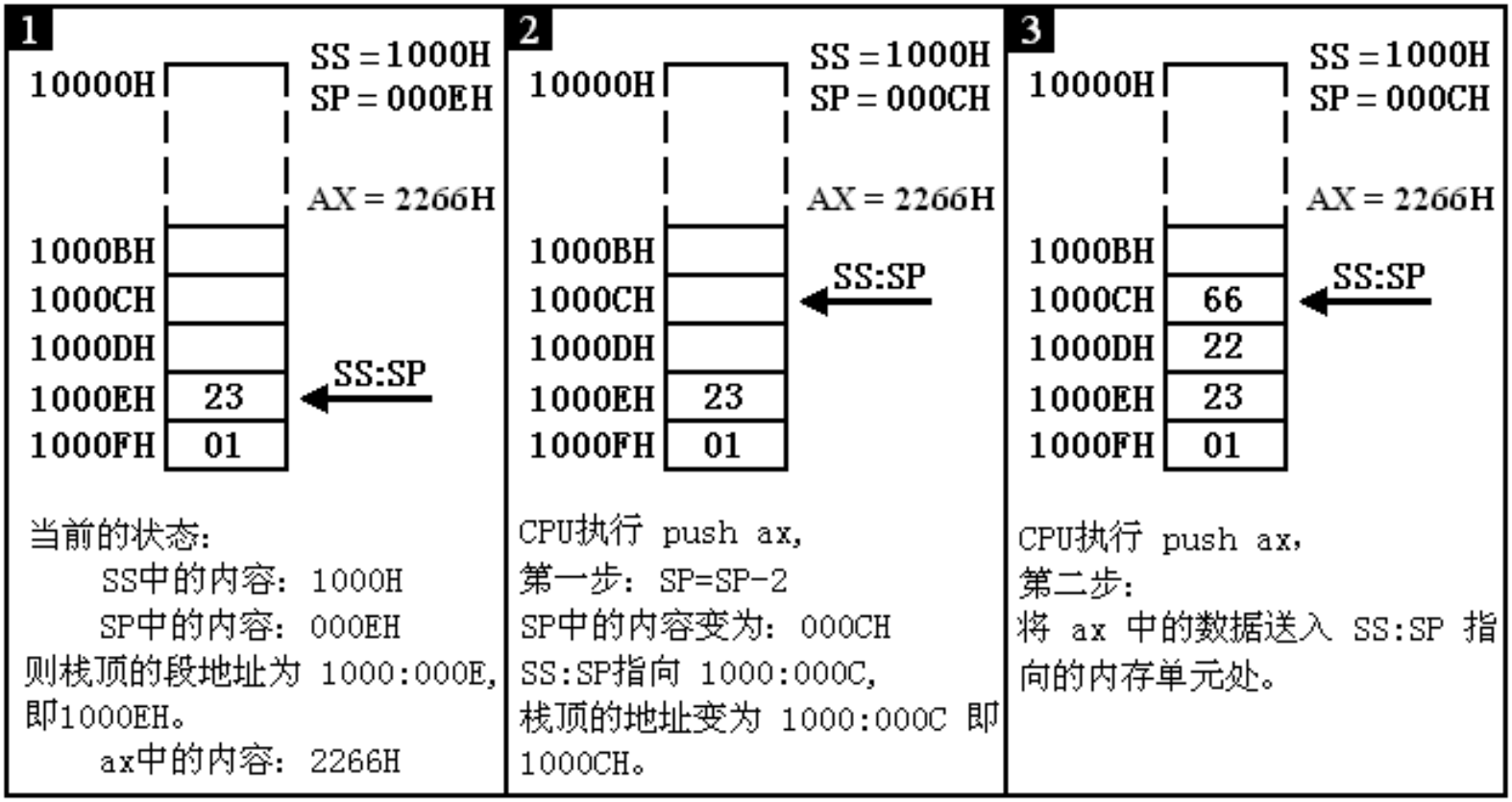
寄存器SP　　存放栈顶的偏移地址

任意时刻，SS:SP指向栈顶元素

push 指令的执行过程

- 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。
  - 图示

push 指令的执行过程



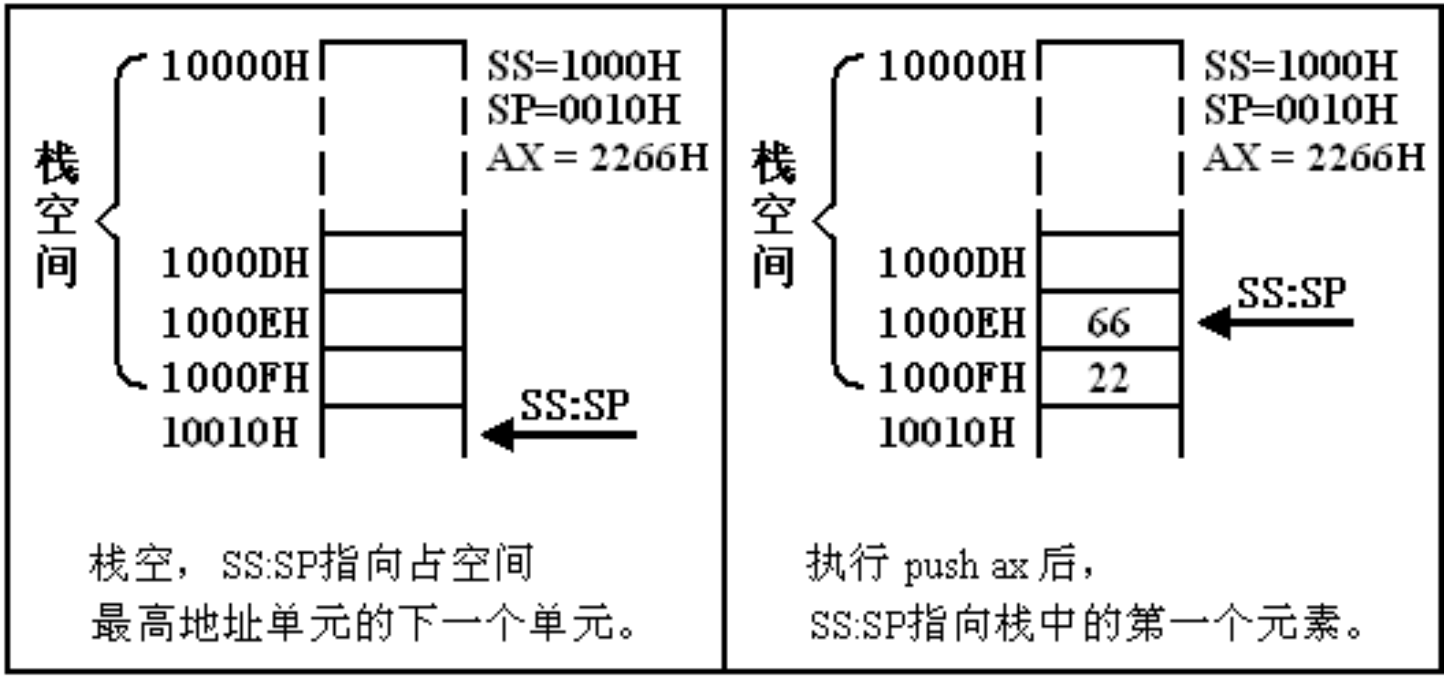
3.6 栈

- 问题3.6：如果我们将10000H~1000FH 这段空间当作栈，初始状态栈是空的，此时，SS=1000H，SP=？

- 思考后看分析。

问题3.6分析

- SP = 0010H



问题3.6分析（续）

- 栈为空，就相当于栈中唯一的元素出栈，出栈后，SP=SP+2，SP 原来为000EH，加 2 后SP=10H，所以，当栈为空的时候，SS=1000H，SP=10H。
- 换个角度看

问题3.6分析（续）

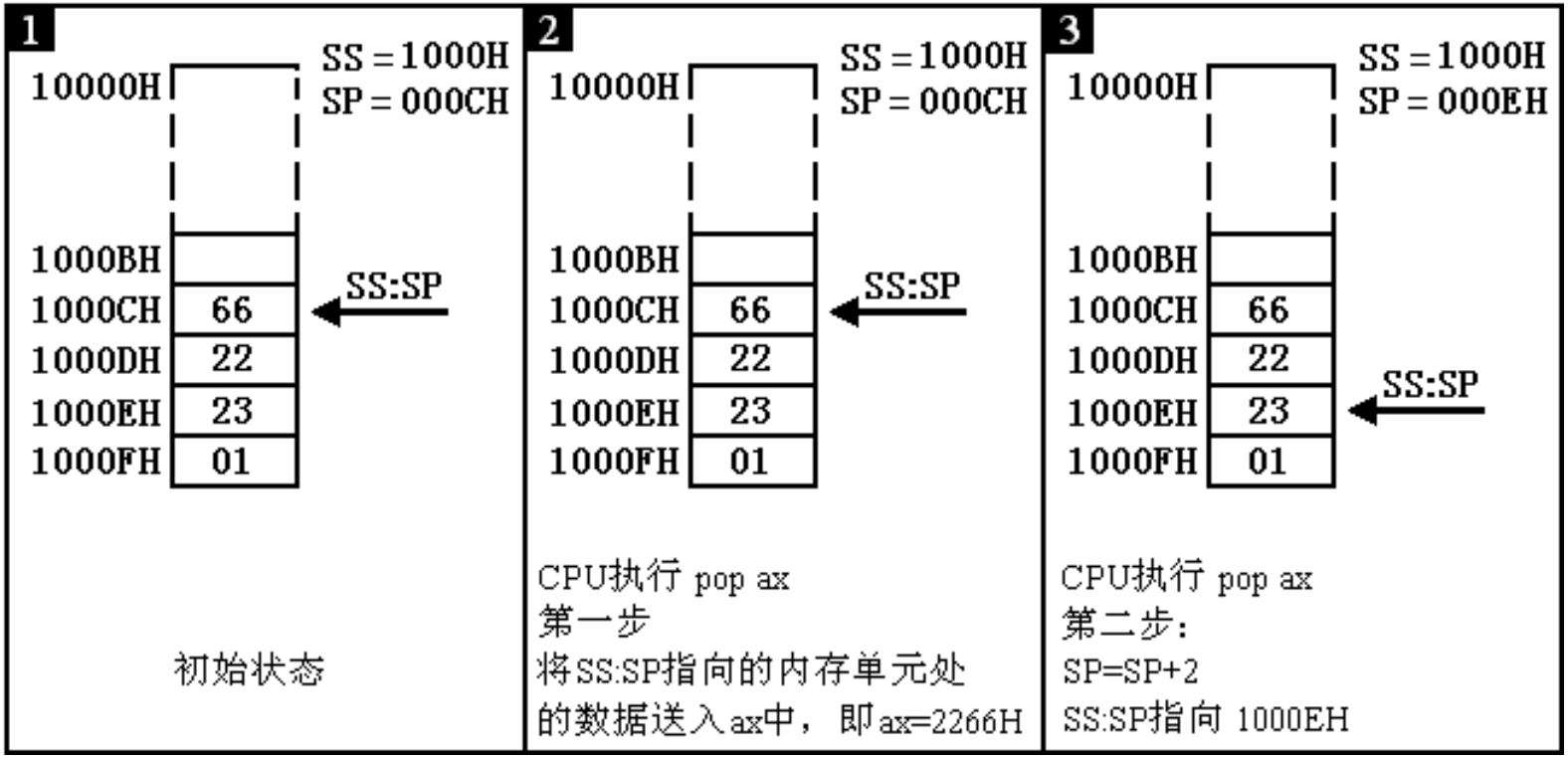
- 换个角度看：

  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 指令的执行过程

- 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。
  - 图示

pop 指令的执行过程



- 注意

pop 指令的执行过程

- 注意：
  - 出栈后，SS:SP指向新的栈顶 1000EH，pop操作前的栈顶元素，1000CH 处的2266H 依然存在 ，但是，它已不在栈中。
  - 当再次执行push等入栈指令后，SS:SP移至1000CH，并在里面写入新的数据，它将被覆盖。

3.8 栈顶超界的问题

- 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。
- 当栈满的时候再使用push指令入栈，
  栈空的时候再使用pop指令出栈，
  都将发生栈顶超界问题。

3.9 push、pop指令

- push和pop指令是可以在寄存器和内存之间传送数据的。
- push和pop指令的格式

3.9 push、pop指令

- push和pop指令的格式（1）
  - push 寄存器：将一个寄存器中的数据入栈
  - pop寄存器：出栈，用一个寄存器接收出栈的数据

- 例如：push ax
  pop bx

3.9 push、pop指令

- push和pop指令的格式（2）
  - push 段寄存器：将一个段寄存器中的数据入栈
  - pop段寄存器：出栈，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指令的格式（3）
  - 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位）
  - pop 内存单元：出栈，用一个内存字单元接收出栈的数据
- 例如：push [0]

  pop [2]

  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。

3.9 push、pop指令

- 问题3.9

  编程：

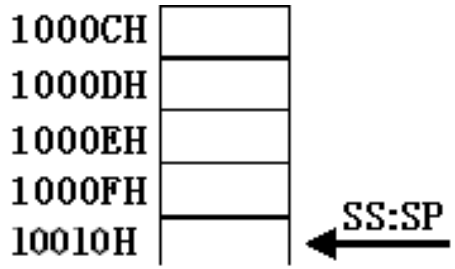
  （1）将10000H~1000FH 这段空间当作栈，初始状态是空的；

  （2）设置AX=002AH，BX=002BH；

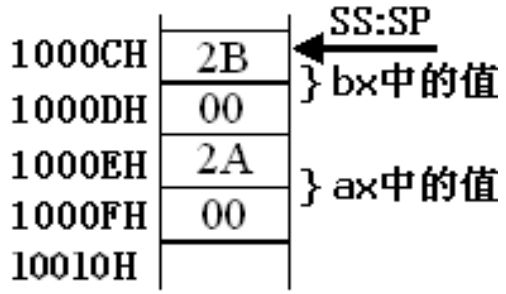
  （3）利用栈，交换 AX 和 BX 中的数据。
- 思考后看分析。

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

栈的综述

- （1）8086CPU提供了栈操作机制，方案如下：
  在SS，SP中存放栈顶的段地址和偏移地址；
  提供入栈和出栈指令，他们根据SS:SP指示的地址，按照栈的方式访问内存单元。
- （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，栈空间的大小我们要自己管理。
- （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

- 我们可以将长度为 N（N ≤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。
- 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。

小结