《微机原理A》

第四讲：微处理器的编程结构(二)

主讲老师：王克义

本讲主要内容

- 段寄存器
- 实模式下的存储器寻址
- 堆栈

4.1 段寄存器

- 微处理器寄存器集合中的另一组寄存器为16位的段寄存器，用于与微处理器中的其他寄存器联合生成存储器地址。

- **对于同一个微处理器而言，段寄存器的功能在实模式下和保护模式下是不相同的。**

1. **代码段寄存器CS(Code Segment)**：代码段是一个存储区域，用以保存微处理器使用的代码(程序或过程)。代码段寄存器定义代码段的起始地址。

- **在实模式下工作时，它定义一个64K字节存储器段的起点；**

- **在保护模式下工作时，它选择一个描述代码段起始地址、长度及其他一些必要的属性信息(如可读、可写或可被执行等)的描述符。**

2. 数据段寄存器DS(Data Segment)：

数据段是包含程序所使用的大部分数据的存储区。

- 与代码段寄存器CS类似，数据段寄存器DS用以定义数据段的起始地址。
- 与代码段一样，对于8086～80286，数据段的长度限制为64KB；
- 对于工作在保护模式下的80386及更高型号的微处理器，数据段长度限制为4GB。

3. 附加段寄存器ES(Extra Segment)：

附加段是为某些串操作指令存放目的操作数而附加的一个数据段。

- 附加段寄存器ES用以定义附加段的起始地址。附加段的长度限制与上述代码段及数据段的情况相同。

4. 堆栈段寄存器SS(Stack Segment)：

堆栈是存储器中的一个特殊存储区，用以暂时存放程序运行中的一些数据和地址信息。

- 堆栈段寄存器SS定义堆栈段的首地址。由堆栈段寄存器SS和堆栈指针寄存器(ESP/SP)确定堆栈段内的存取地址。
- 另外,EBP/BP寄存器也可以寻址堆栈段内的数据。

5. 段寄存器FS和GS：

这两个段寄存器仅对80386及更高型号微处理器有效，以便程序访问相应的两个附加的存储器段。

4.2 实模式下的存储器寻址

1. 实模式下的存储器地址空间

在实模式下存储器的地址空间为1M字节单元，其地址范围为**00000H～FFFFFH**。

- 实模式下的存储器地址空间被分为通用和专用两种存储区。

- 从地址**00000H～003FFH**这第一个1024个字节单元是专用的。这个存储区为中断向量表区，专门用来存放256个中断服务程序的入口地址(也称中断向量)，每个入口地址占4个字节单元。

- 从地址**FFFF0H～FFFFFH**这16个字节单元保留给系统的初始化代码。

- 当处理器加电或复位(Reset)时，CPU执行的第一条指令就是起始于地址FFFF0H的。

- 通常是在FFFF0H处存放一条无条件转移指令，以转移到系统程序的入口处。

- 通用区域用来存储一般的程序指令和数据。由图4.1可见，它的地址范围为**00400H～FFFEFH**。

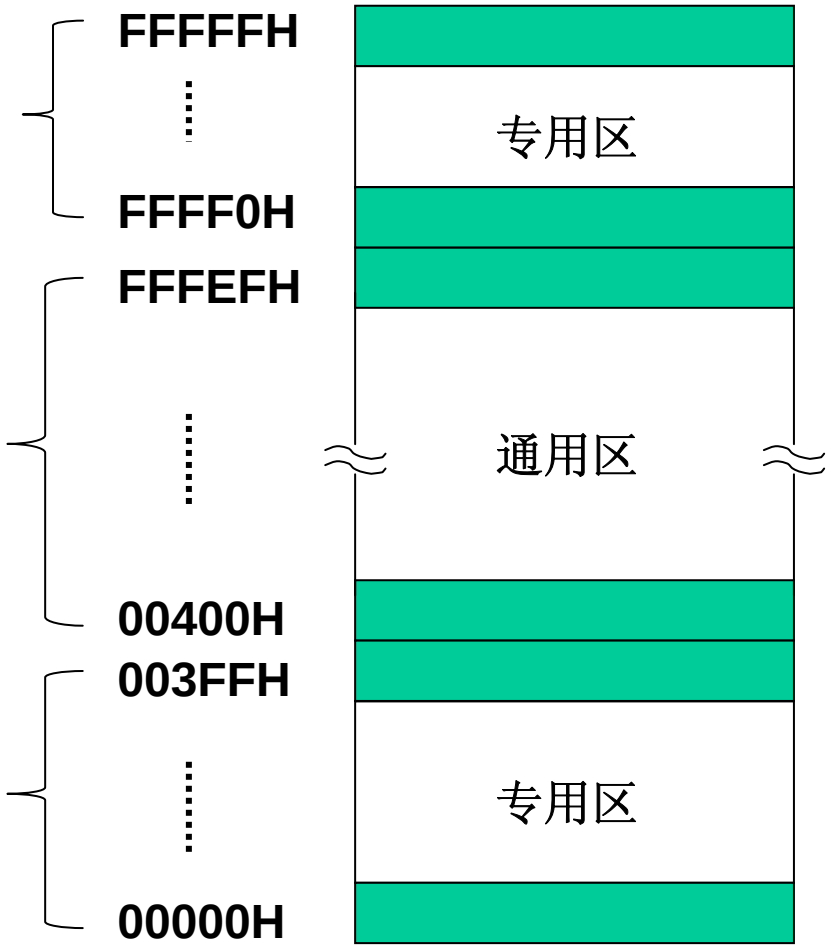


图4.1 实模式下存储器地址空间

2. 存储器分段技术

为什麽要采用存储器“分段”技术？

- 实模式下CPU可直接寻址的地址空间为**2**20＝1M字节单元。CPU需输出20位地址信息才能实现对1M字节单元存储空间的寻址。

- 但实模式下CPU中所使用的寄存器均是16位的，内部ALU也只能进行16位运算，其寻址范围局限在**2**16＝65536(64K)字节单元。

- 为了实现对1M字节单元的寻址，80x86系统采用了存储器分段技术。

- 具体做法是，将1M字节的存储空间分成许多逻辑段，每段最长64K字节单元，可以用16位地址码进行寻址。

- 每个逻辑段在实际存储空间中的位置是可以浮动的,其起始地址可由段寄存器的内容来确定。实际上，段寄存器中存放的是段起始地址的高16位，称之为**“段基值”(segment base value)**。

- 逻辑段在物理存储器中的位置如图4.2所示。

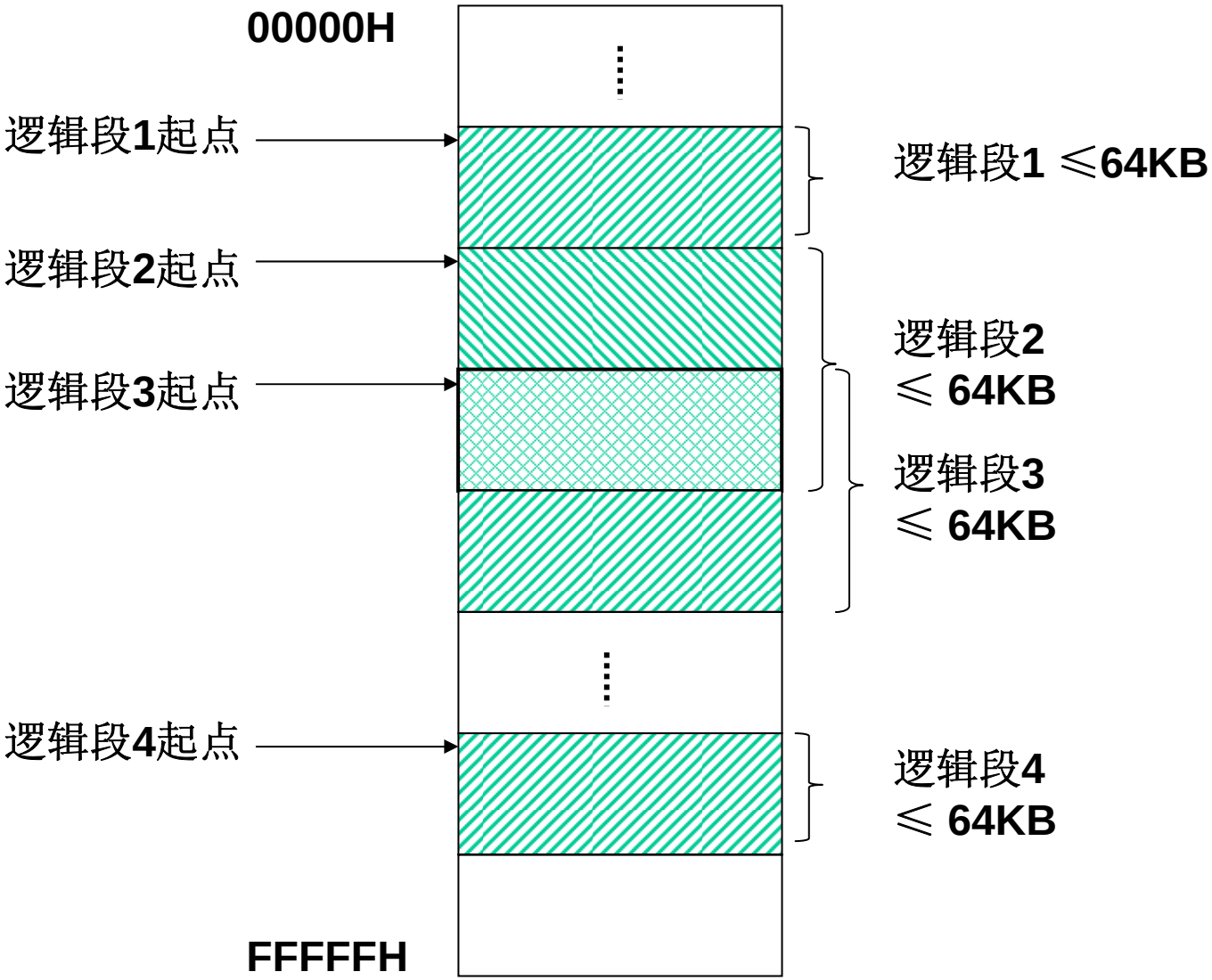


图4.2　逻辑段在物理存储器中的位置

- **各个逻辑段在实际的存储空间中可以完全分开，也可以部分重叠，甚至完全重叠。**

- 段的起始地址的计算和分配通常是由操作系统完成的，并不需要普通用户参与。

3. 实模式下的存储器寻址

(1) 物理地址与逻辑地址

在有地址变换机构的计算机系统中，每个存储单元可以看成具有两种地址：**物理地址和逻辑地址**。

- **物理地址**是信息在存储器中实际存放的地址，它是CPU访问存储器时实际输出的地址。

  例如，实模式下的80x86/Pentium系统的物理地址是20位，存储空间为220＝1M字节单元，地址范围从00000H到FFFFFH。

- CPU和存储器交换数据时所使用的就是这样的物理地址。

- **逻辑地址是编程时所使用的地址。**

- 或者说程序设计时所涉及的地址是逻辑地址而不是物理地址。

- 编程时不需要知道产生的代码或数据在存储器中的具体物理位置。这样可以简化存储资源的动态管理。

- 在实模式下的软件结构中，**逻辑地址**由“**段基值**”和“**偏移量**”两部分构成。

- **“段基值”**是段的起始地址的高16位。
- **“偏移量”(offset)也称偏移地址，**它是所访问的存储单元距段的起始地址之间的字节距离。
- 给定段基值和偏移量，就可以在存储器中寻址所访问的存储单元。
- 在实模式下，“段基值”和“偏移量”均是16位的。
- “段基值”由段寄存器**CS、DS、SS、ES、FS和GS**提供；
- “偏移量”由**BX、BP、SP、SI、DI、IP**或以这些寄存器的组合形式来提供。

(2) 实模式下物理地址的产生

实模式下CPU访问存储器时的20位物理地址可由逻辑地址转换而来。

- **具体方法是，将段寄存器中的16位“段基值”左移4位(低位补0)，再与16位的“偏移量”相加，即可得到所访问存储单元的物理地址，**如图4.3所示。

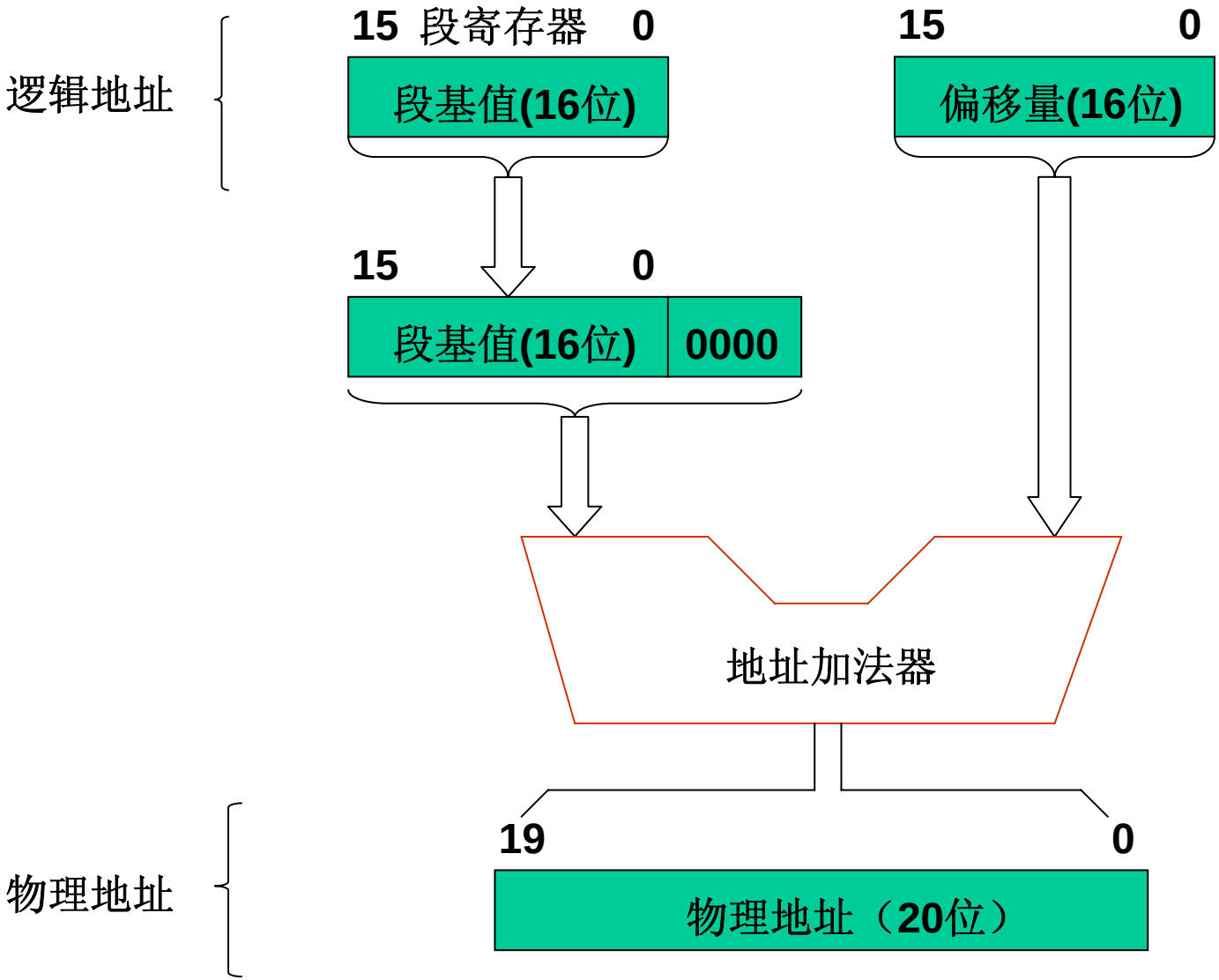


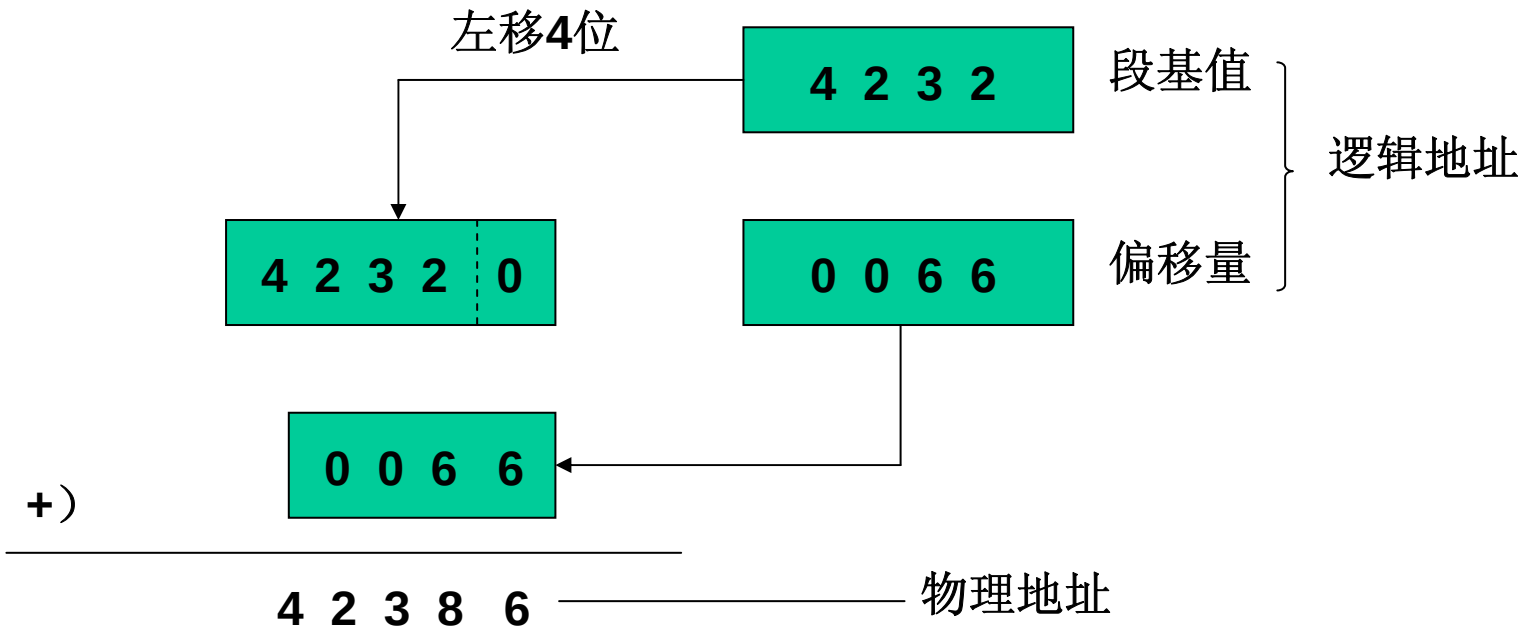
图4.3 实模式下物理地址的产生

- 上述由逻辑地址转换为物理地址的过程也可以表示成如下计算公式：

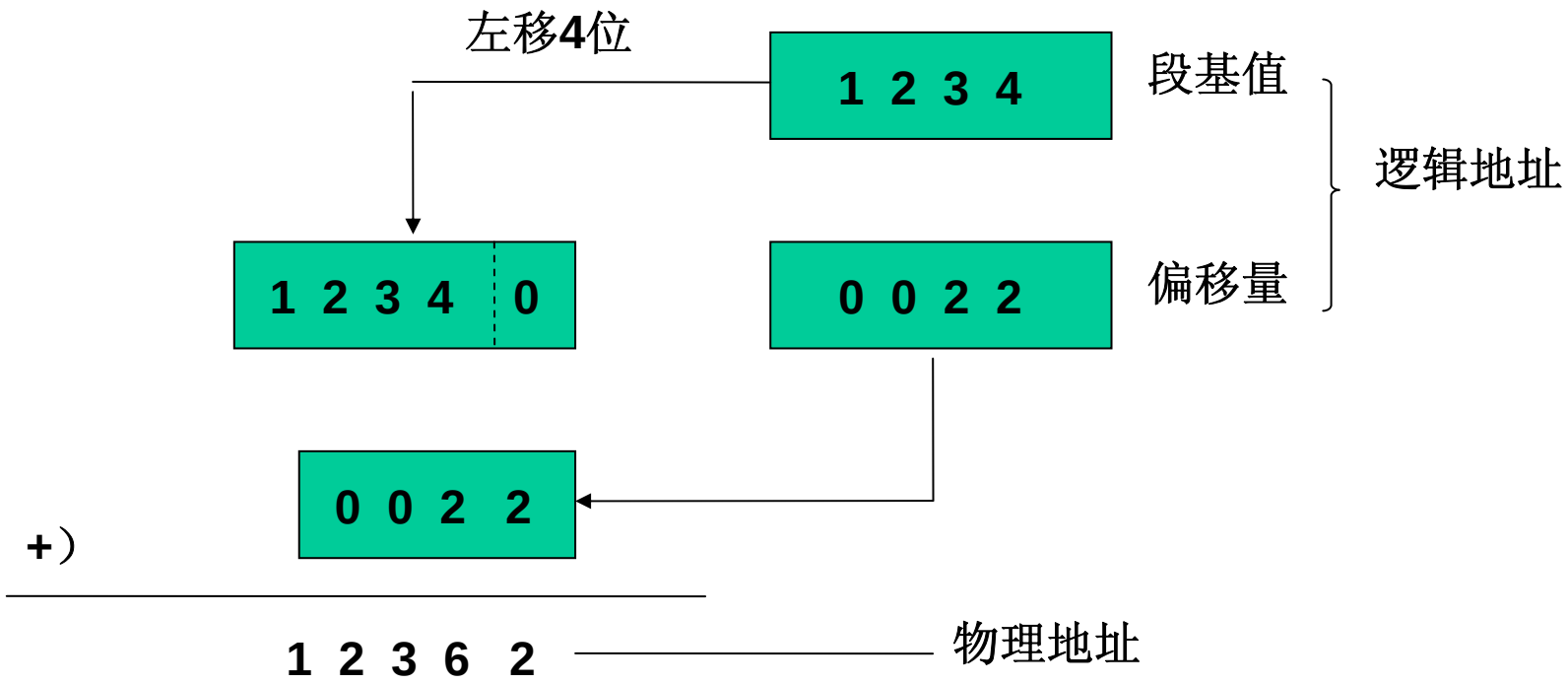
  物理地址＝段基值×16+偏移量

- 上式中的“段基值×16”在微处理器中是通过将段寄存器的内容左移4位(低位补0)来实现的，与偏移量相加的操作则由地址加法器来完成。

例 2.1 设代码段寄存器CS的内容为4232H，指令指针寄存器IP的内容为0066H，即CS＝4232H，IP＝0066H，则访问代码段存储单元的物理地址计算如下：



例 2.2 设数据段寄存器DS的内容为1234H，基址寄存器BX的内容为0022H，即DS＝1234H，BX＝0022H，则访问数据段存储单元的物理地址计算如下：



例 2.3 若段寄存器内容是002AH，产生的物理地址是002C3H，则偏移量是多少?

解: 将段寄存器内容左移4位，低位补0得：

002A0H。

从物理地址中减去上列值得偏移量为：

002C3H-002A0H＝0023H。

- 同一个物理地址可以由不同的逻辑地址来构成。
- 例如，段基值为0020H，偏移量为0013H，构成的物理地址为00213H；
- 若段基值改变为0021H，配以新的偏移量0003H，其物理地址仍然是00213H，如图4.4 所示。

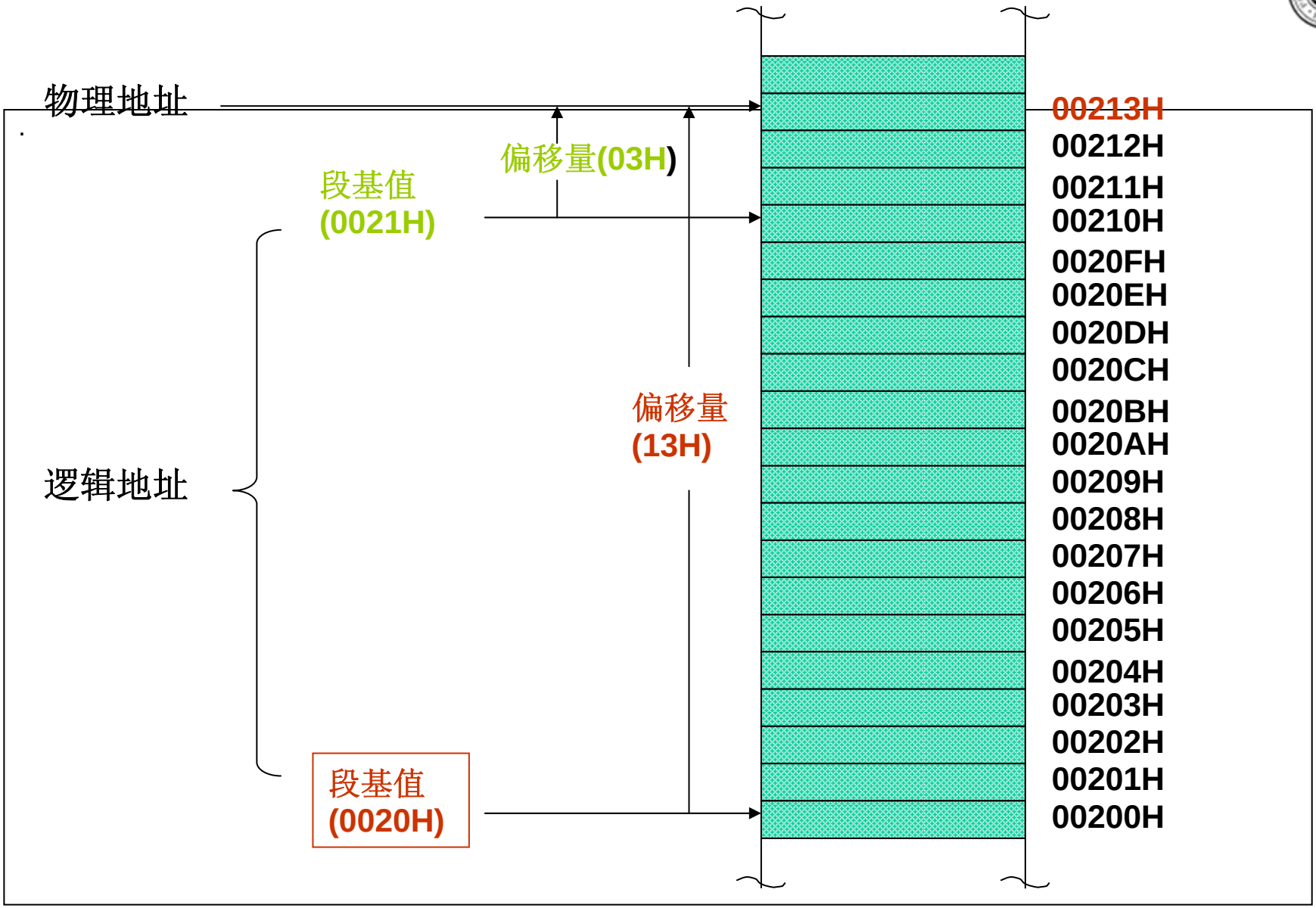


图4.4 一个物理地址对应多个逻辑地址

4. “段加偏移”寻址

- 上述由段基值(段寄存器的内容)和偏移量相结合的存储器寻址机制也称为“**段加偏移**”寻址机制，所访问的存储单元的地址常被表示成“**段基值：偏移量**”的形式。

例如，若段基值为2000H，偏移量为3000H，则所访问的存储单元的地址为2000H：3000H。

- 图4.5进一步说明了这种“段加偏移”的寻址机制如何选择所访问的存储单元的情形。

- 这里段寄存器的内容为1000H，偏移地址为2000H。图中显示了一个64KB长的存储器段，该段起始于10000H，结束于1FFFFH。

- 图中也表示了如何通过段基值(段寄存器的内容)和偏移量找到存储器中被选单元的情形。**偏移量(offset)**也称偏移地址，正如图中所示，它是自段的起始位置到所选存储单元之间的**距离(或跨度)**。

实模式存储器

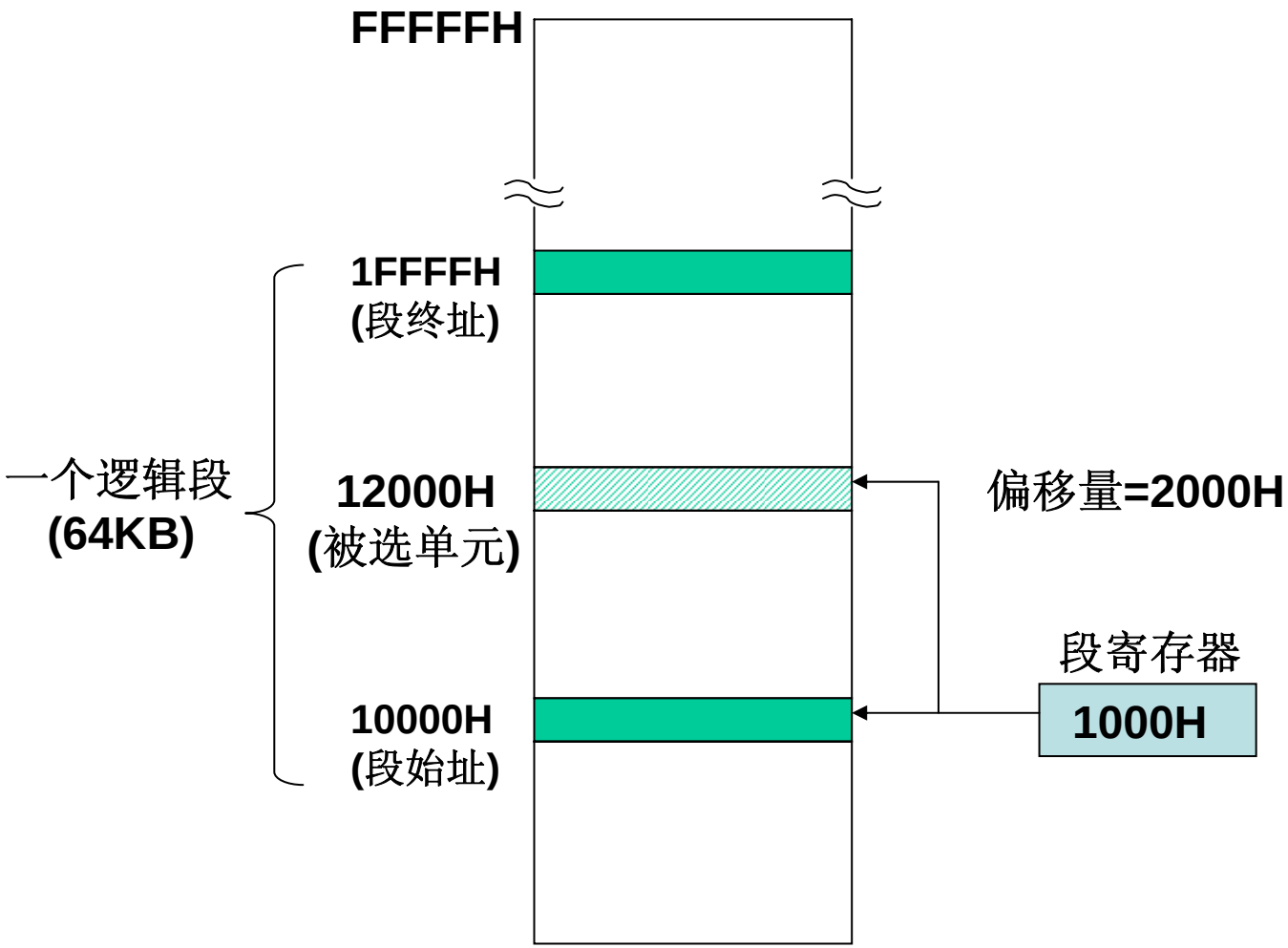


图4.5　实模式下存储器寻址机制——“段加偏移”

- 图4.5中段的起始地址10000H是由段寄存器内容1000H**左移4位低位补0(或在1000H后边添加0H)**而得到的。

- 段的结束地址1FFFFH是由段起始地址10000H与段长度FFFFH(64K)相加之结果。

- 在这种“段加偏移”的寻址机制中，由于是将段寄存器的内容左移4位(相当于乘以十进制数16)来作为段的起始地址的，所以**实模下各个逻辑段只能起始于存储器中16字节整数倍的边界**。

- 这样可以简化实模式下CPU生成物理地址的操作。通常称这16字节的小存储区域为“分段”或“节”(**paragraph**)。

5. 默认的段和偏移寄存器

- 在“段加偏移”的寻址机制中，微处理器有一套用于定义各种寻址方式中段寄存器和偏移地址寄存器的组合规则。如表4-1和表4-2所示。

段寄存器	偏移地址寄存器	主要用途
CS	IP	指令地址
SS	SP或BP	堆栈地址
DS	BX、DI、SI、8位或16位数	数据地址
ES	串操作指令的DI	串操作目的地址

表4-1默认的16位“段+偏移”寻址组合

4.3 堆栈

- **堆栈定义：**

  堆栈是存储器中的一个特定的存储区，它的一端(栈底)是固定的，另一端(栈顶)是浮动的，信息的存入和取出都只能在浮动的一端进行，并且遵循后进先出(Last In First Out)的原则。

- 堆栈是一种后进先出型数据结构。
- 堆栈是插入删除操作都只能在一端进行的线性表。

堆栈的用途：

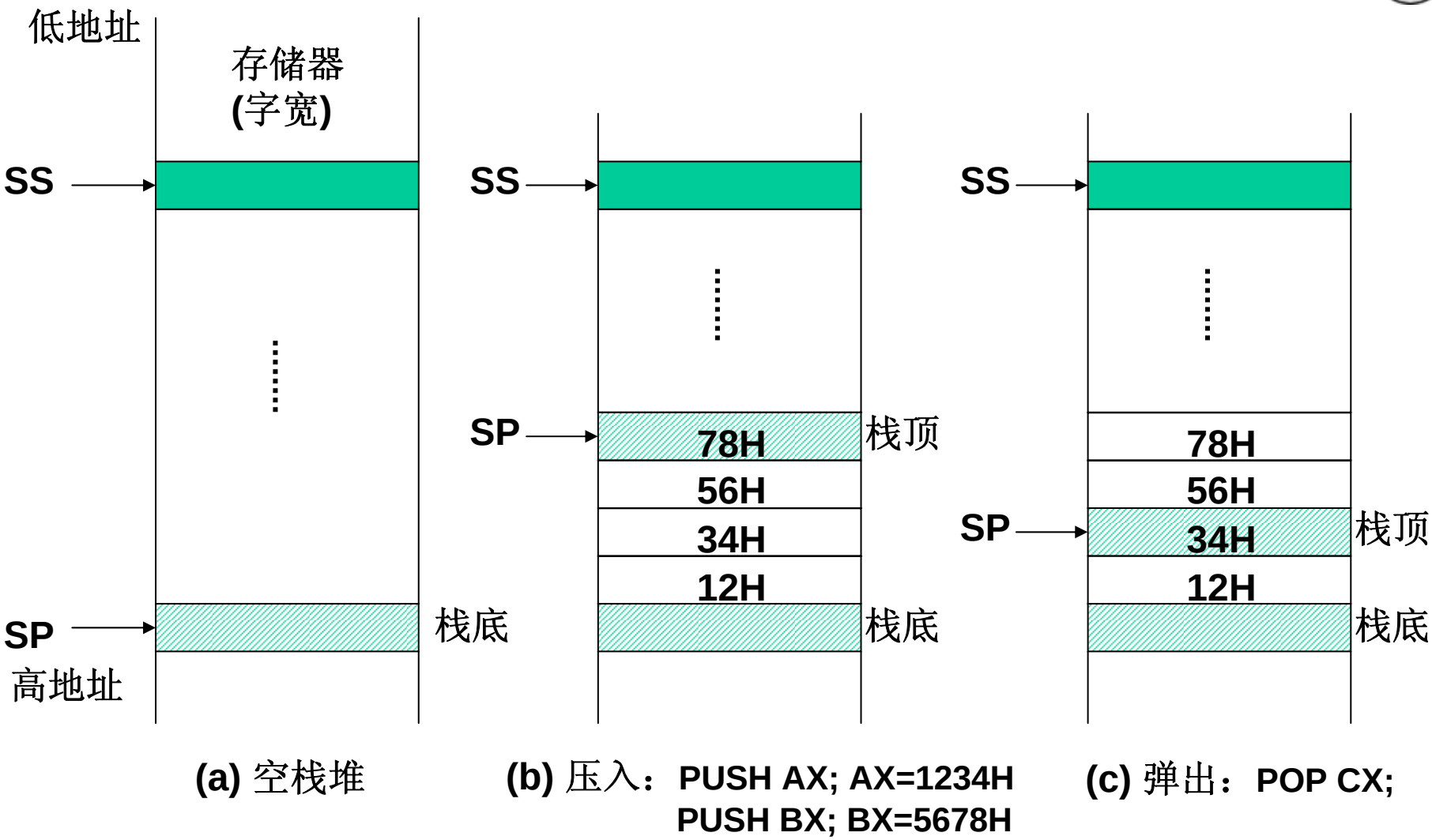
- 堆栈主要用来暂时保存程序运行时的一些地址或数据信息。
- 当CPU执行调用(Call)指令时，用堆栈保存程序的返回地址(亦称断点地址)；
- 在中断响应及中断处理时，通过堆栈“保存现场”和“恢复现场”；
- 有时也利用堆栈为子程序传递参数。

堆栈的结构：

- 堆栈是在存储器中实现的，并由堆栈段寄存器SS和堆栈指针寄存器SP来定位。

- SS寄存器中存放的是堆栈段的段基值，它确定了堆栈段的起始位置。

- SP寄存器中存放的是堆栈操作单元的偏移量，SP总是指向栈顶。
- 图4.6给出了堆栈的基本结构及操作示意图。



(a) 空栈堆

(b) 压入：PUSH AX; AX=1234H
PUSH BX; BX=5678H

(c) 弹出：POP CX;

图4.6　堆栈的结构与操作

- 值得注意的是，这种结构的堆栈是所谓“**向下生长的**”，即栈底在堆栈的高地址端，当堆栈为空时SP就指向栈底。

- 堆栈段的段基址(由SS寄存器确定)并不是栈底。

堆栈的操作特点：

- 实模式下的堆栈为16位宽(字宽)，堆栈操作指令(PUSH指令或POP指令)对堆栈的操作总是以字为单位进行。
- 要压栈(执行PUSH指令)时，先将SP的值减2，然后将16位的信息压入新的栈顶。
- 要弹栈(执行POP指令)时，先从当前栈顶取出16位的信息，然后将SP的值加2。
- 可概括为：**“压栈时， 先修改栈指针后压入”**
  “弹栈时， 先弹出后修改栈指针”。

习题（四）

- 教材P160

  13.　14.