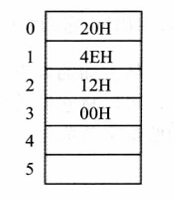
\_03\_寄存器（内存访问）

**1.内存中字的存储（小端字节序）**

将一个字存放在内存中，需要占用两个字节的连续空间，按照小端方式存储。

小端字节序：将数据拆分成字节，低地址存放低位字节，高地址存放高位字节。

e.g. 将字4E20和0012先后存入内存中，结果如下：



（先将4E20拆分成字节4E、20，之后低地址存放低位20，高地址存放高位4E）

（0012的存放同理）

**2.DS和[偏移地址]**

①DS（Data Segment）：数据段寄存器

功能：用于存储当前要访问的数据所在的段地址。

②[偏移地址]：表示一个内存空间

功能：表示当前DS段中，偏移地址所对应的内存空间。

（当用[xxx]操作内存时，8086CPU会自动取DS中的值作为其段地址）

e.g. 若DS为1000，则[1]表示地址为1000:1的内存空间。

③对比CS、DS和SS

CS:IP，表示将要执行的下一条指令的地址。

[偏移地址]，表示(DS:偏移地址)所对应的内存空间。

SS:SP，表示指向栈顶元素的指针。

**3.mov、add、sub指令**

**（1）mov指令**

语法：mov dest, src

功能：将src中的值拷贝到dest中。

用法： mov 寄存器, 立即数 （如：mov ax,8）

mov 寄存器, 寄存器 （如：mov ax,bx）

mov 寄存器, 内存单元 （如：mov ax,[0]）

mov 内存单元, 寄存器 （如：mov [0],ax）

mov 段寄存器, (通用)寄存器 （如：mov ds,ax）[mov不能为cs赋值]

注意事项【重点】：

①汇编指令中，一般是目标操作数在左边，源操作数在右边。

（a=1，是将右边的值赋给左边，汇编指令按照这个习惯使用操作数）

②汇编指令的源操作数、目标操作数的长度要匹配。

dest、src二者都是字节/都是字。

③mov指令中，两个操作数必须至少有一个是寄存器。

目的是为了保证代码运行的速度（寄存器的读写速度快）。

④mov指令中，左边（目标操作数）不能是立即数。

⑤若用mov指令为段寄存器（ds、ss、es）赋值，其源操作数最好是通用寄存器（ax、bx、cx、dx）。

⑥mov指令不能为cs代码段寄存器、ip指令指针寄存器赋值。

cs、ip寄存器需要用jmp指令赋值。

（2）add、sub指令

语法：add dest, src

sub dest, src

功能：add=>将两个操作数相加，结果存储在目的操作数中（dest=dest+src）

sub=>将两个操作数相减，结果存储在目的操作数中（dest=dest-src）

注意事项：

（add、sub的操作数使用规则和mov大致相同）

①两个操作数中，至少有一个是寄存器。

②dest不能是立即数。

③两个操作数都不能是段寄存器。

**4.栈**

**（1）栈顶指针 SS和SP**

①SS（Stack Segment）：栈段寄存器

功能：用于存储栈顶元素的段地址。

②SP（Stack Pointer）：栈指针寄存器

功能：用于存储栈顶元素的偏移地址。

注意：在元素入栈/出栈时，SP的值会动态变化：

PUSH入栈时，SP=SP-2；

POP出栈时，SP=SP+2；

③SS和SP

**SS:SP表示栈顶指针，指向栈顶元素。**

**（2）8086CPU中栈的机制【重点】**

①栈底、栈顶、栈顶上限

**8086CPU提供了SS:SP表示栈顶指针。**

**8086CPU并没有提供用于记录栈底和栈顶上限的寄存器。**

因此，8086CPU只知道栈顶在何处，但是并不知道我们安排的栈空间有多大。在编程时，我们需要自己操心CS:IP的值，避免栈顶指针越界。

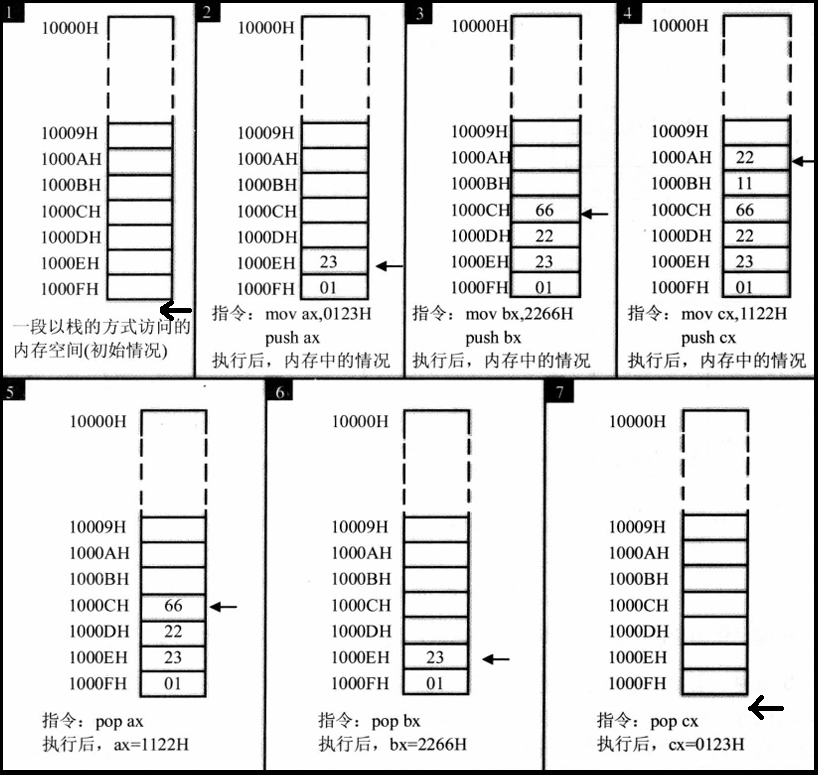
②入栈和出栈

若我们把一段内存当作栈，**规定栈底在高地址处，栈顶上限在低地址处。**

**栈顶指针初始指向栈底的下一位。**

8086CPU规定，**只能入栈/出栈16位（1字）的数据。**

**当元素入栈，栈顶指针-=2；当元素出栈，栈顶指针+=2。**



（3）PUSH和POP指令

①PUSH指令

语法：PUSH 寄存器/内存单元

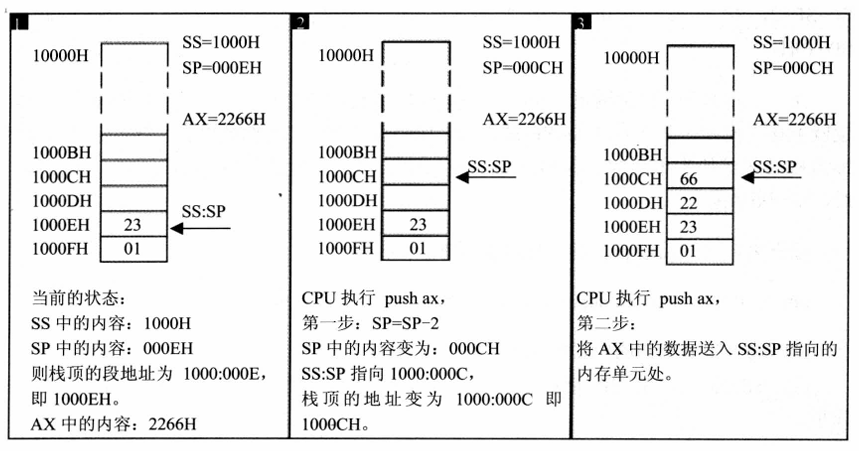
功能：**SP=SP-2，之后将寄存器/内存单元中的值压入栈中。**

例如： PUSH AX

PUSH DS（段寄存器的值可以入栈[包括CS]）

PUSH [0]

注意事项：PUSH指令的操作数，长度必须是16位。



②POP指令

语法：POP 寄存器/内存单元

功能：**将栈顶的值拷贝到寄存器/内存单元中，之后SP=SP+2**

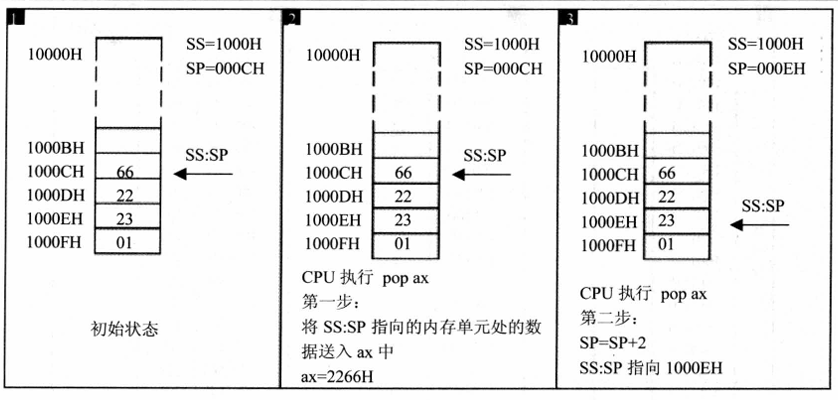
例如： POP AX

POP DS（可以出栈到段寄存器中[不能出栈到CS]）

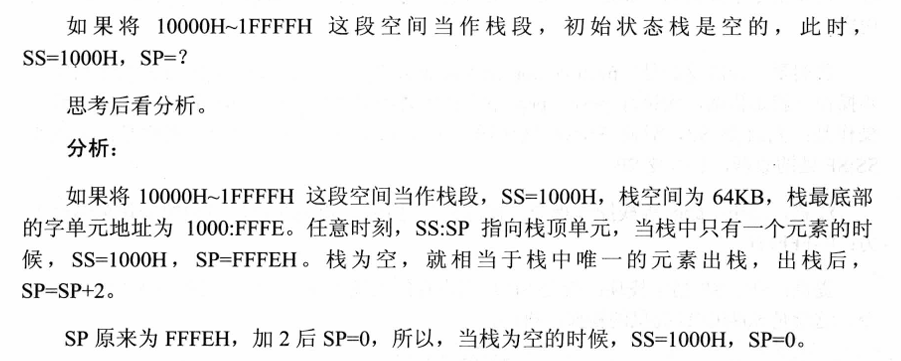
POP [0]

注意事项：①POP指令的操作数，长度必须是16位。

②POP指令通过SP+=2的方式，从逻辑上将栈顶元素弹出。但是，之前栈顶的值仍然会保留在内存之中，当下次使用PUSH指令时才会将其覆盖。



（4）知识点补充：栈的循环性



如上，假设栈中只有一个字，则其地址为1000:FFFE。将其出栈，SP=SP+2，运行结果为1000:0000，表示栈空时的栈顶指针。

栈的循环：栈空时，栈顶指针为1000:0000。当元素入栈，SP=SP-2，栈顶指针变为1000:FFFE、1000:FFFC......。最终，当栈顶元素达到上限，此时栈顶指针为1000:0000，栈是满的。若再向栈中添加元素，则SP又会循环到FFFE，上一轮的栈元素会被覆盖。

以上栈循环的原因：①CPU只知道SS:SP表示的栈顶指针，而不知道整个栈空间的范围； ②PUSH和POP指令只能改变SP的值，不能改变SS的值，SP越界则会取余，然后循环。