1. 什么是复杂指令集？什么是精简指令集？80x86采⽤的是哪种？

**复杂指令集，也称为CISC指令集，英文名是CISC，（Complex Instruction Set Computing的缩写）。在CISC微处理器中，[程序](https://baike.baidu.com/item/%E7%A8%8B%E5%BA%8F/13831935?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%A4%8D%E6%9D%82%E6%8C%87%E4%BB%A4%E9%9B%86/_blank)的各条指令是按顺序串行执行的，每条指令中的各个操作也是按顺序串行执行的。**[顺序执行](https://baike.baidu.com/item/%E9%A1%BA%E5%BA%8F%E6%89%A7%E8%A1%8C/332454?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%A4%8D%E6%9D%82%E6%8C%87%E4%BB%A4%E9%9B%86/_blank)的优点是控制简单，但[计算机](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA/140338?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%A4%8D%E6%9D%82%E6%8C%87%E4%BB%A4%E9%9B%86/_blank)各部分的利用率不高，执行速度慢。其实它是英特尔生产的x86系列（也就是IA-32架构）CPU及其兼容CPU，如AMD、VIA的。即使是现在新起的X86-64（也被称为AMD64）都是属于CISC的范畴。

**[精简指令集](https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86/4736552?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)计算机(RISC:Reduced Instruction Set Computer RISC) [3]****是一种执行较少类型[计算机指令](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%8C%87%E4%BB%A4/1490932?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)的[微处理器](https://baike.baidu.com/item/%E5%BE%AE%E5%A4%84%E7%90%86%E5%99%A8/104320?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)，**起源于80年代的MIPS主机（即RISC机），**RISC机中采用的[微处理器](https://baike.baidu.com/item/%E5%BE%AE%E5%A4%84%E7%90%86%E5%99%A8/104320?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)统称[RISC](https://baike.baidu.com/item/RISC/62696?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)处理器。**这样一来，它能够以更快的速度执行操作（每秒执行更多百万条指令，即MIPS）。因为计算机执行每个指令类型都需要额外的晶体管和电路元件，计算机指令集越大就会使微处理器更复杂，执行操作也会更慢。纽约约克镇IBM研究中心的John Cocke证明，**计算机中约20%的指令承担了80%的工作，于1974年，他提出RISC的概念。**许多当前的[微芯片](https://baike.baidu.com/item/%E5%BE%AE%E8%8A%AF%E7%89%87/6908988?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)都使用RISC概念

精简指令集计算机（RISC:Reduced Instruction Set Computer） [3]  是一种[指令长度](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E9%95%BF%E5%BA%A6?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)较短的计算机，其运行速度比CISC要快。RISC的[指令系统](https://baike.baidu.com/item/%E6%8C%87%E4%BB%A4%E7%B3%BB%E7%BB%9F?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)相对简单，它只要求硬件执行很有限且最常用的那部分指令，大部分复杂的操作则使用成熟的[编译技术](https://baike.baidu.com/item/%E7%BC%96%E8%AF%91%E6%8A%80%E6%9C%AF?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E7%B2%BE%E7%AE%80%E6%8C%87%E4%BB%A4%E9%9B%86%E8%AE%A1%E7%AE%97%E6%9C%BA/_blank)，由简单指令合成。

1. 什么是⼩端存储？什么是⼤端存储？80x86采⽤的是哪种？

小端存储（Little Endian）。[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82/1096318?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)或[半字节](https://baike.baidu.com/item/%E5%8D%8A%E5%AD%97%E8%8A%82/7674672?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)的最低位字节（Least Significant Bit，LSB）存放于内存最低位字节地址上。

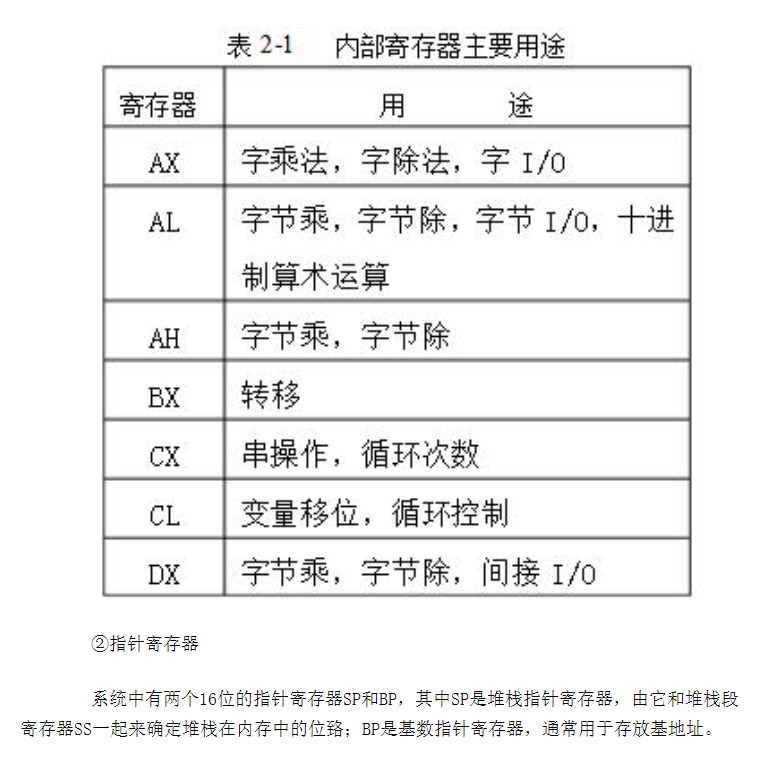
即最低地址存放的最低[字节](https://baike.baidu.com/item/%E5%AD%97%E8%8A%82?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)，一个用十六进制表示的32位数据：0x12345678，存放在[存储字长](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E5%AD%97%E9%95%BF?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)是32位的[存储单元](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E5%8D%95%E5%85%83?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)中，按低字节到高字节的存储顺序为0x78、0x56、0x34和0x12。整个[存储字](https://baike.baidu.com/item/%E5%AD%98%E5%82%A8%E5%AD%97?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E5%B0%8F%E7%AB%AF%E5%AD%98%E5%82%A8/_blank)从低字节到高字节读出的结果就是：0x78563412，为Intel x86 系列等采用。

大端存储则相反

1. 8086有哪5类寄存器？请分别举例说明其作⽤。

①通用寄存器

　　8086有4个16位的通用寄存器（AX、BX、CX、DX），可以存放16位的操作数，也可分为8个8位的寄存器（AL、AH；BL、BH；CL、CH；DL、DH）来使用。其中AX称为累加器，BX称为基址寄存器，CX称为计数寄存器，DX称为数据寄存器，这些寄存器在具体使用上有一定的差别，如表2－1所示。

③变址寄存器

　　系统中有两个16位的变址寄存器SI和DI，其中SI是源变址寄存器，DI是目的变址寄存器，都用于指令的变址寻址方式。

　　④控制寄存器

　　IP、标志寄存器是系统中的两个16位控制寄存器，其中IP是指令指针寄存器，用来控制CPU的指令执行顺序，它和代码段寄存器CS一起可以确定当前所要取的指令的内存地址。顺序执行程序时，CPU每取一个指令字节，IP自动加1，指向下一个要读取的字节；当IP单独改变时，会发生段内的程序转移；当CS和IP同时改变时，会产生段间的程序转移。

　　标志寄存器的内容被称为处理器状态字PSW，用来存放8086CPU在工作过程中的状态。

　　⑤段寄存器

系统中共有4个16位段寄存器，即代码段寄存器CS、数据段寄存器DS、堆栈段寄存器SS和附加段寄存器ES。这些段寄存器的内容与有效的地址偏移量一起，可确定内存的物理地址。通常CS划定并控制程序区，DS和ES控制数据区，SS控制堆栈区。

1. 有哪些段寄存器，它们的作⽤是什么？

同上

1. 什么是寻址？8086有哪些寻址⽅式？

寻址方式就是**寻找指令中操作数的方式,或寻找指令转移目的地址的方式。**

8086/8088微处理器的寻址方式分为**数据寻址方式和程序转移寻址方式**。数据寻址方式主要包括**立即数寻址方式、寄存器寻址方式、直接寻址方式、寄存器间接寻址方式、寄存器相对寻址方式、基址变址寻址方式和相对基址变址寻址方式**;程序转移寻址方式主要包括**段内直接寻址、段内间接寻址、段间直接寻址、段间间接寻址。**

1. 什么是直接寻址？直接寻址的缺点是什么？

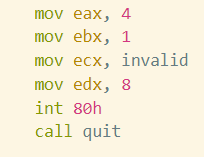
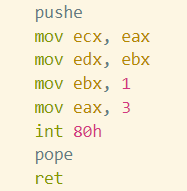
在指令格式的地址的字段中直接指出操作数在内存的地址。

A的位数限制了操作数的寻址范围，而且必须修改A的值，才能修改操作数的地址

1. 主程序与⼦程序之间如何传递参数？

利用寄存器传递参数，利用约定的存储单元(内存变量)传递参数，利用堆栈传递参数，利用CALL后续区传递参数。

1. 如何处理输⼊和输出？你的代码中在哪⾥体现的？



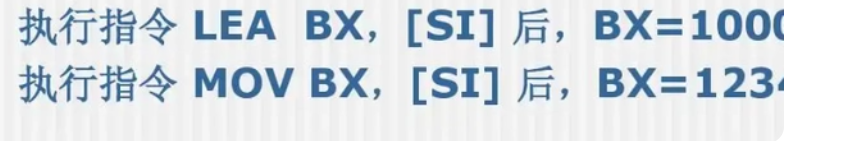
1. 通过什么寄存器保存前⼀次的运算结果？你的代码中在哪⾥体现的？

通用寄存器(个人一般用的eax)

1. 请分别简述 MOV 指令和 LEA 指令的⽤法和作⽤。

传送指令MOV（Move）把一个字节、字或双字的操作数从源位置传送到目的位置，源操作数的内容不变。可以实现立即数到通用寄存器或主存的传送，通用寄存器与通用寄存器、主存或段寄存器之间的传送，主存与段寄存器之间的传送。该操作属于复制性质，不属于搬家性质。

LEA指令是一个计算机指令，可以将有效地址传送到指定的的寄存器。



1. 解释 boot.asm ⽂件中 org 07c00h 的作⽤。

计算机开机后会自动跳转到07c00h处执行指令，如果我们的程序中有跳转指令等（需要一些地址信息时），就需要增加org字段，以告诉[编译器](https://so.csdn.net/so/search?q=%E7%BC%96%E8%AF%91%E5%99%A8&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/lh6868/article/details/_blank)当前段的起始地址是什么，编译器据此即可以计算在程序运行中的真正的地址

1. 解释 boot.asm ⽂件中 times 510-($-$$) db 0 的作⽤。

填充剩下的空间，使生成的二进制代码恰好为512字节

$，代表当前地址。

$-$$可能会经常被用到，它表示本行距离程序开始处的相对距离。  
times 510-($-$$)  db  0  的意思就是将0这个字节重复510-($-$$)遍，知道程序有510B为止。这样，加上结束标志0XAA55占用2B，恰好是512B

1. 解释 bochsrc 中各参数的含义。

megs:32

floppya: 1\_44=a.img, status=inserted

boot: floppy

1. boot.bin 应该放在软盘的哪⼀个扇区？为什么？
2. Loader的作⽤有哪些？