实验一问题清单

1. 什么是复杂指令集？什么是精简指令集？80×86 采用的是哪种？

指令系统类型	指令	寻址方式	实现方式	其他
CISC（复杂）	数量多，最多 200 或 300；使用频率差别大；可变长格式	支持多种	微程序控制技术	研制周期长
RISC（精简）	数量少，一般 100 以下；使用频率接近；定长格式；大部分为单周期指令；操作寄存器，只有 Load/Store 操作内存	支持方式少	增加了通用寄存器，硬布线逻辑控制为主，适合采用流水线	优化编译，有效支持高级语言

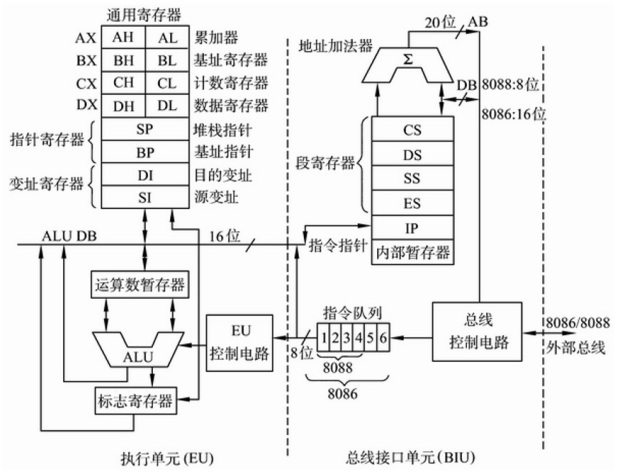
80×86 采用复杂指令集。

2. 什么是小端存储？什么是大端存储？80×86 采用的是哪种？

小端存储是低位字节排放在内存的低地址端，高位字节排放在内存的高地址端；大端存储是高位字节排放在内存的低地址端，低位字节排放在内存的高地址端。

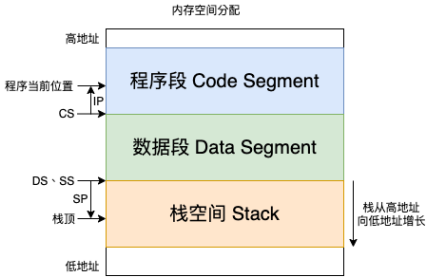
80×86 采用小端存储。

3. 8086 有哪 5 类寄存器？请分别举例说明其作用。



- 数据寄存器
  - AX 累加寄存器：通常用于保存算中间值的操作数，也是与 I/O 设备交互时与外设传输数据的寄存器
  - BX 基址寄存器：通常用于内存寻址时保存地址基址的寄存器，可以配合 DI、SI 提供更复杂的寻址模式
  - CX 计数寄存器：通常用于保存循环次数，也可用于保存用于算数运算、位运算的参数等
  - DX 数据寄存器：通常就是用于存储数据的，偶尔在大数字的乘除运算时搭配 AX 形成一个操作数
- 段寄存器

  - CS 指令段寄存器：用于保存当前执行程序的指令段的起始地址，相当于 `section .text` 的地址
  - DS 数据段寄存器：用于保存当前执行程序的数据段的起始地址，相当于 `section .data` 的地址
  - SS 栈寄存器：用于保存当前栈空间的基址，与 SP(偏移量) 相加得到 SS:SP 可找到当前栈顶地址
  - ES 额外段寄存器：常用于字符串操作的内存寻址基址，与变址寄存器 DI 共用
  - FS、GS 指令段寄存器：80386 额外定义的段寄存器，提供程序员更多的段地址选择
- 指针寄存器：段寄存器用来保存不同区块（段），还需要三个寄存器作为块中指针用来保存偏移量
  - SP 栈指针：与 SS 共用，可通过 SS:SP 找到当前栈顶地址
  - BP 参数指针：与 SS 共用，可通过 SS:BP 找到当前栈底地址



- 变址寄存器：由于寄存器有限，有时候并不会直接保存操作数或是操作数的地址，而是保存操作数的某个相对偏移量，透过间接寻址来操作数据。变址寄存器就是在这个过程中负责保存偏移量的部分。
  - SI 源变址寄存器：通常用于保存源操作数（字符串）的偏移量，与 DS 搭配使用 DS:SI
  - DI 目的变址寄存器：通常用于保存目的操作数（字符串）的偏移量，与 ES 搭配使用 ES:DI
- 控制寄存器
  - IP 指令指针：与 CS 共用，可通过 CS:IP 寻到当前程序执行到的地址
  - FLAG：控制寄存器 FLAG 保存 CPU 运行的状态和一些标识位，每位代表不同的含义：

4. 有哪些段寄存器，它们的作用是什么？

- CS 指令段寄存器：用于保存当前执行程序的指令段的起始地址，相当于 `section .text` 的地址
- DS 数据段寄存器：用于保存当前执行程序的数据段的起始地址，相当于 `section .data` 的地址
- SS 栈寄存器：用于保存当前栈空间的基址，与 SP(偏移量) 相加得到 SS:SP 可找到当前栈顶地址
- ES 额外段寄存器：常用于字符串操作的内存寻址基址，与变址寄存器 DI 共用
- FS、GS 指令段寄存器：80386 额外定义的段寄存器，提供程序员更多的段地址选择

5. 什么是寻址？8086 有哪些寻址方式？

寻址是指找到操作数的地址，从而能够取出操作数。

8086 的寻址方式有：

- 立即寻址：`MOV AX 1234H`
- 直接寻址：`MOV AX [1234H]`

- 寄存器寻址：MOV AX BX，操作数在寄存器里，给出寄存器名即可取走操作数
- 寄存器间接寻址：MOV AX [BX]，操作数有效地址在寄存器之中（SI、DI、BX、BP）
- 寄存器相对寻址：MOV AX [SI+3]
- 基址加变址：MOV AX [BX+DI]，把一个基址寄存器（BX、BP）的内容，加上变址寄存器（SI、DI）的内容
- 相对基址加变址：MOV AX [BX+DI+3]

6. 什么是直接寻址？直接寻址的缺点是什么？

直接寻址是指指令直接包含有操作数的有效地址（偏移地址），例如 MOV AX [1234H] 直接给出操作数位于地址 1234H 。

直接寻址的缺点是操作数的位数决定了寻址范围，即有限的地址空间。

7. 主程序与子程序之间如何传递参数？你的实验代码中在哪里体现的？

- 利用寄存器传递参数
- 利用约定的地址传递参数
- 利用堆栈传递参数（常用）
- 利用 CALL 后续区传递参数

8. 如何处理输入和输出？你的代码中在哪里体现的？

采用系统调用的形式，例如函数 sprint 和 getline

9. 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

10. 请分别简述 MOV 指令和 LEA 指令的用法和作用。

MOV 指令能够实现数据传送，LEA 指令的功能是将源操作数，即存储单元的有效地址（偏移地址）传送到目的操作数。例如 lea eax,[ebx+8] 是将 ebx+8 这个值直接赋给 eax，而 mov eax,[ebx+8] 则是把内存地址为 ebx+8 处的数据赋给 eax。

11. 解释 boot.asm 文件中 org 07c00h 的作用

告诉汇编器，当前这段代码会放在 07c00h 处。所以，如果之后遇到需要绝对寻址的指令，那么绝对地址就是 07c00h 加上相对地址。

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

BIOS 会将 512 字节的数据加载到内存中，所以需要将不足 512 字节的部分写满 0。是 510 不是 512 的原因是最后一行指令 dw 0xaa55 是两个字节。

$ 代表当前指令的地址，$$ 代表一个节的开始处被汇编的地址

其等价命令是 db 510-($-$$) dup('0')

13. 解释 bochsrc 中各参数的含义。

```
megs: 32
display_library: sdl2
floppya: 1_44=a.img, status=inserted
boot: floppy
```

- `display_library`：Bochs 使用的 GUI 库
- `megs`：虚拟机内存大小（MB）
- `floppya`：虚拟机外设，软盘为 a.img 文件
- `boot`：虚拟机启动方式，从软盘启动

14. boot.bin 应该放在软盘的哪一个扇区？为什么？

放在第一个扇区。BIOS 程序检查 0 面 0 磁道 1 扇区，如果扇区以 `0xaa55` 结束，就认为是引导扇区，将其 512 字节的数据加载到内存 `07c00` 处，然后设置 PC，跳到内存 `07c00` 处开始执行代码。

15. Loader 的作用有哪些？

为了突破 512 字节的限制，引入另外一个重要的文件 loader.asm，引导扇区只负责把 loader 加载入内存并把控制权交给他，这样将会灵活得多。最终，由 loader 将内核 kernel 加载入内存，才开始了真正操作系统内核的运行。

- 跳入保护模式
- 启动内存分页
- 从 kernel.bin 中读取内核，并放入内存，然后跳转到内核所在的开始地址，运行内核