实验问题回答

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CISC，用于X86架构；精简指令集RISC，用于ARM架构；80X86采用复杂指令集

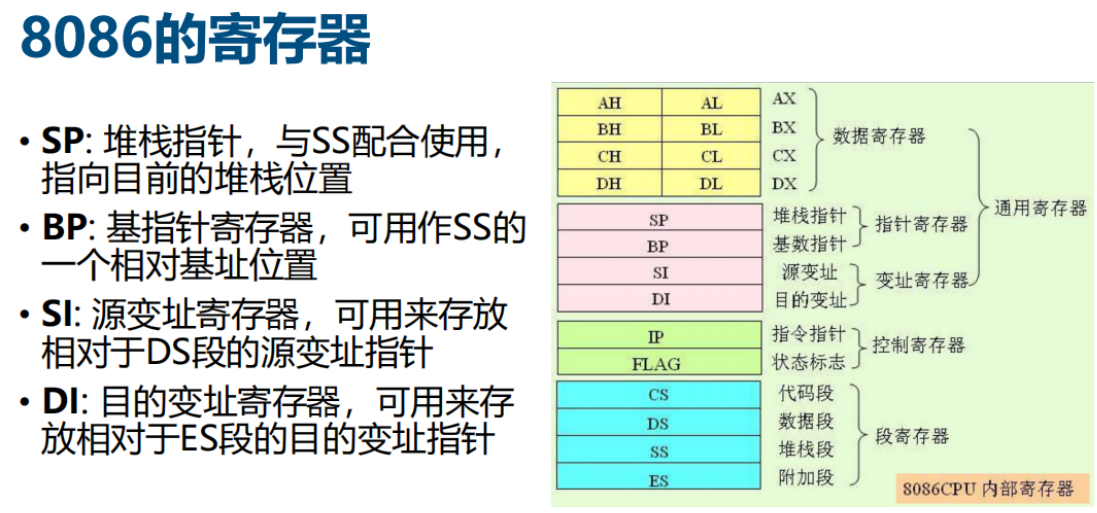
2.说明小端和大端存储的区别，并说明 80x86 系列采用了哪种方式？

- 1)大端模式：Big-Endian就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。其实大端模式才是我们直观上认为的模式，和字符串存储的模式差类似）

  低地址 ----------> 高地址
  0x12 | 0x34 | 0x56 | 0x78
- 2)小端模式：Little-Endian就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。

  低地址 ----------> 高地址
  0x78 | 0x56 | 0x34 | 0x12
- 80x86采用小端

3.8086 有哪五类寄存器，请分别举例说明其作用？



4.段寄存器的种类和作用：

1、代码段寄存器CS（Code Segment）

存放当前正在运行的程序代码所在段的段基址，表示当前使用的指令代码可以从该段寄存器指定的存储器段中取得，相应的偏移量则由IP提供。

2、数据段寄存器DS（Data Segment）

指出当前程序使用的数据所存放段的最低地址，即存放数据段的段基址。

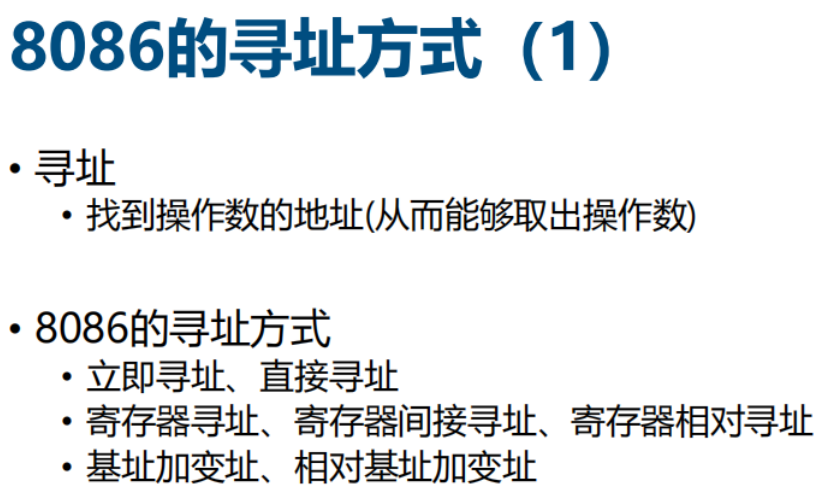
3、堆栈段寄存器SS（Stack Segment）

指出当前堆栈的底部地址，即存放堆栈段的段基址。

4、附加段寄存器ES（Extra Segment）

指出当前程序使用附加数据段的段基址，该段是串操作指令中目的串所在的段。

5.什么是寻址？8086有那些寻址方式？



6.什么是直接寻址？直接寻址的缺点是什么？

3.直接寻址

指令字的形式地址地址A就是操作数的真实地址EA，即EA=A。

直接寻址的优点是简单，指令在执行阶段仅访问一次主存，不需要专门计算操作的地址；缺点是A的位数决定了该指令操作数的寻址范围，操作数的地址不易修改。

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？

- 寄存器法
  - 主程序中把要传递的参数放在某一指定的寄存器中，然后从子程序中取出指定的寄存器参数
  - 缺点：能传递的参数有限，因为寄存器有限
- 约定单元法
- 堆栈法

8.如何处理输入和输出？你的代码中在哪里体现的？

系统调用输入输出指令

9.通过什么寄存器保存前一次的运算结果，在代码中哪里体现出来。

一般是目的寄存器

也有特殊，比如除法

> 无符号除法指令DIV(DIVision) DIV OPRD ；
>
> 除数OPRD决定是8位除法还是16位除法;
>
> OPRD=8位,则被除数默认在AX中,AX除以OPRD的商保存在AL中,余数保存在AH中;

> OPRD=16位,则被除数默认在DX与AX中,结果的商保存在AX中,余数保存到DX中

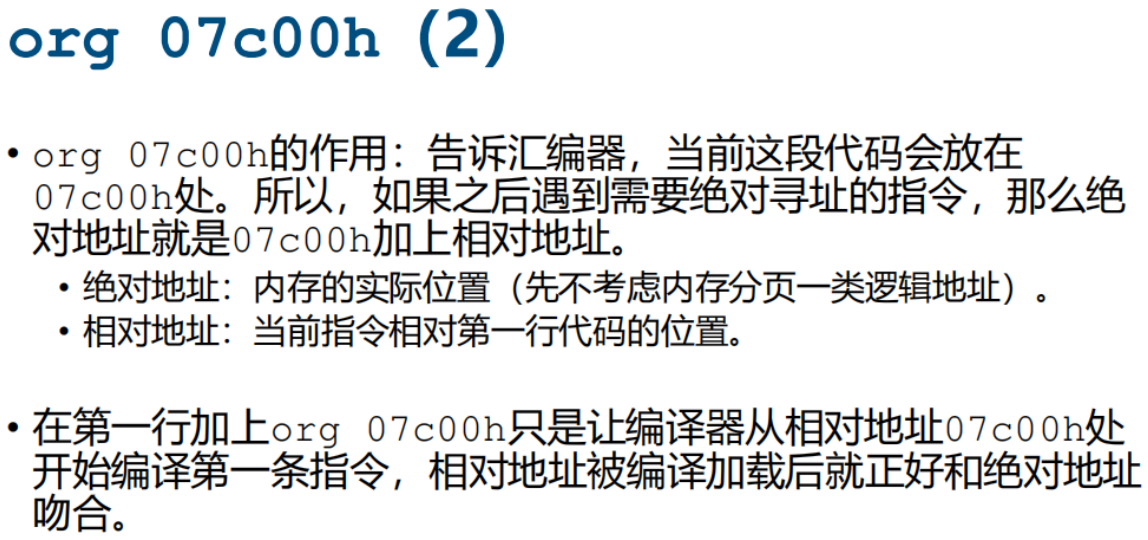
10.请分别简述 MOV 指令和 LEA 指令的用法和作用?

lea是"load effective address"的缩写（加载有效地址），将源操作数的地址加载到目的寄存器中，简单的说，lea指令可以用来将一个内存地址直接赋给目的操作数，例如：
lea eax,[ebx+8]就是将ebx+8这个值直接赋给eax，而不是把ebx+8处的内存地址里的数据赋给eax。

而mov指令则恰恰相反，例如：
mov eax,[ebx+8]则是把内存地址为ebx+8处的数据赋给eax。

> 如LEA EAX, [EBX + ECX]，它相当于计算EBX和ECX的值，将这个值保存到EAX寄存器中。原因：由于EBX+ECX计算出来的值是该内存地址，而通过[EBX+ECX]得到的是内存地址保存的值，而LEA命令是加载该值的有效地址并且保存到目标寄存器中，也就是将EBX+ECX的值保存到EAX寄存器中
> `由于加载的是有效地址，而不是实际地址，所以EAX中保存的是EBX+ECX，而不是ds:EBX+ECX`

11. 解释 boot.asm 文件中，org 0700h 的作用



14. boot.bin 应该放在软盘的哪一个扇区？为什么?

0面0道1扇区 512字节

1. ROM-BIOS完成之前，最后一件事是从外存储设备读取更多的指令交给处理器执行
2. 如果计算机设置从硬盘启动，ROM-BIOS会读取主引导扇区的内容，将他加载到逻辑地址0x0000:0x7c00处
3. 判断是否有效（最后两个字节是0x55和0xAA）
4. 如果有效，jmp 0x0000:0x7c00继续执行

BIOS

- 开机，从ROM运行BIOS程序，BIOS是厂家写好的。
- BIOS程序检查软盘0面0磁道1扇区，如果扇区以0xaa55结束，则认定为引导扇区，将其512字节的数据加载到内存的07c00处，然后设置PC，跳到内存07c00处开始执行代码。
- 以上的0xaa55以及07c00都是一种约定，BIOS程序就是这样做的，所以我们就需要把我们的OS放在软盘的第一个扇区，填充，并在最末尾写入0xaa55。

```
17  times   510-($-$$)  db  0    ; 填充剩下的空间，使生成的二进制代码恰好为512字节
18  dw  0xaa55                   ; 结束标志
```

15. loader 的作用有哪些?

因为boot只能是512字节.这么小的空间没法做啥.只能作为跳板所以写个loader用来加载内核

> 一个操作系统从开机到开始运行，大致经历“引导—>加载内核入内存—>跳入保护模式—>开始执行内核”这样一个过程。也就是说，在内核开始执行之前不但要加载内核，而且还有准备保护模式等一系列工作，如果全都交给引导扇区来做，512字节很可能是不够用的，所以不妨把这个过程交给另外的模块来完成，我们把这个模块叫做Loader。引导扇区负责把Loader加载入内存并且把控制权交给它，其他工作放心地交给Loader来做，因为它没有512字节的限制，将会灵活得多

Loader

- 跳入保护模式
  - 最开始的x86处理器16位，寄存器用ax，bx等表示，称为实模式。后来扩充成32位，eax，ebx等，为了向前兼容，提出了保护模式
  - 必须从实模式跳转到保护模式，才能访问1M以上的内存。
- 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
  - 跟boot类似，使用汇编直接在软盘下搜索kernel.bin
  - 但是，不能把整个kernel.bin放在内存，而是要以ELF文件的格式读取并提取代码。

12. 解释语句 `times 510-($-$$) db 0`，为什么是 `510`? `$` 和 `$$` 分别表示什么?

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位512字节

- 一个有效的主引导扇区，最后两个字节应当是0x55和0xAA，所以是510，最后两个字节填充`dw 0xaa55`
- `$`当前行的地址
- `$$`当前节（section）的开始处的地址

13. 解释配置文件 bochsrc 文件中各参数的含义

```
# 第一步，首先设置Bochs在运行过程中能够使用的内存，本例为32MB。
# 关键字为：megs 单位为MB
# 虚拟机内存大小 (MB)
megs: 32

# 设置Bochs所使用的磁盘，软盘的关键字为floppy。
# 若只有一个软盘，则使用floppya即可，若有多个，则为floppya，floppyb...
# 这是用来设置软盘的相关属性。类型为1.44M容量的软盘， 镜像文件名为a.img
floppya: 1_44=a.img, status=inserted

# 选择启动盘符，启动方式
boot: floppy

# Bochs使用的GUI库
display_library: sdl2
```

启动虚拟机（1）

- 启动还需要Bochs的配置文件。告诉Bochs，你希望你的虚拟机是什么样子的，如内存多大，硬盘映像和软盘映像都是哪些文件等内容。
  - 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

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