Lab1_question

Steps

T1:

```
nasm boot.asm -o boot.bin
dd if=boot.bin of=a.img bs=512 count=1 conv=notrunc
bochs -f bochsrc
```

T2:

```
nasm boot.asm -o boot.bin
nasm loader.asm -o loader.bin
dd if=boot.bin of=boot.img bs=512 count=1 conv=notrunc
sudo mount boot.img /media/ -t vfat -o loop
sudo cp loader.bin /media/
sync
umount /media/
bochs -f bochsrc
```

T3:

```
nasm -f elf main.asm
ld -m elf_i386 main.o -o main
./main
```

Questions

1. 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

--见问题二

•指针寄存器

--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

--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

2. 有哪些段寄存器，它们的作用是什么?

·**CS指令段寄存器**

--保存当前执行程序的指令段的起始地址

·**DS数据段寄存器**

--保存当前执行程序的数据段的起始地址

·**SS栈寄存器**

--保存当前栈空间的基址

·**ES额外段寄存器**

--用于字符串操作的内存寻址基址

·**FS，GS指令段寄存器**

--80386额外定义的段寄存器

3. 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

·寻址方式

--立即寻址

--直接寻址

--寄存器寻址

--寄存器间接寻址

--寄存器相对寻址

--基址加变址

--相对基址加变址

4. 主程序与子程序之间如何传递参数?

·利用寄存器传递参数

·利用约定的地址传递参数

·利用堆栈传递参数

> ·利用CALL后续区传递参数

5. 解释 boot.asm 文件中 org 07c00h 的作用。如果去掉这一句，整个程序应该怎么修改?

> ·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。
>
> .

6. 解释 int 10h 的功能。

   int 10h是一条中断向量指令，用于调用BIOS中提供的显卡相关功能

   INT 10H 是由 BIOS 对屏幕及显示器所提供的服务程序。调用前需要在寄存器AH中存放欲调用的功能号，功能号说明如下表。

7. 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')

8. 解释 bochsrc 中各参数的含义。

> - megs：虚拟机内存大小（MB）
> - display_library：Bochs 使用的 GUI 库
> - 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

9. 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

0. 为什么不让Boot程序直接加载内核，而需要先加载Loader再加载内核?

   这是因为在Boot程序中无法获取内核的完整路径和大小等信息，也无法确定内核加载到内存的哪个位置。因此，需要使用Loader程序来加载内核，Loader程序可以读取内核文件的头部信息，确定内核的位置和大小，并将其正确加载到内存中。

1. 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

2. Kernel的作用有哪些?

kernel是整个操作系统的最底层，它负责整个硬件的驱动，以及提供各种系统所需的核心功能，包括防火墙机制、是否支持LVM或Quota等文件系统等等，如果内核不认识某个最新的硬件，那么硬件也就无法被驱动，你也就无法使用该硬件。

1 这才是真正的操作系统

2 内存管理，进程调度，图像显示，网络访问等等，都是内核的功能。

3 内核的开发使用高级语言

-C语言可以更高效的编写内核

-但我们是在操作系统层面上编写另一个操作系统，于是生成的内核可执行文件是和当前操作系统平台相关的。比如Linux下是ELF格式，有许多无关信息，于是，内核并不能像boot.bin或loader.bin那样直接放入内存中，需要loader从kernel.bin中提取出需要放入内存中的部分