1、boot.asm 中，org 0700h 的作用：

告诉编译器，这段程序要被加载到内存偏移地址 0x0700h 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。

在第一行加上 org 07c00h 只是让编译器从相对地址 07c00h 处开始编译第一条指令，相对地址被编译加载后就正好和绝对地址吻合

2、为什么要把 boot.bin 放在第一个扇区？直接复制为什么不行？

计算机从软盘启动时，会检查其 0 面 0 磁道 1 扇区，若发现其以 0xAA55 结束，则 BIOS 认为其是一个引导扇区，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。直接复制不能发挥其引导的作用。

具体来讲：当计算机的电源被打开时，它会先进行加电自检，然后寻找启动盘，如果是选择从软盘启动，计算机就会检查软盘的 0 面 0 磁道 1 扇区，如果发现它是以 0xAA55 结束，则 BIOS 认为它是一个引导扇区。

硬盘的 0 柱面、0 磁头、1 扇区称为主引导扇区，也叫主引导记录 MBR，该记录占用 512 个字节，它用于硬盘启动时将系统控制权转给用户指定的、在分区表中登记了某个操作系统分区。MBR 的内容是在硬盘分区时由分区软件写入该扇区的，MBR 不属于任何一个操作系统，不随操作系统的不同而不同，即使不同，MBR 也不会夹带操作系统的性质，具有公共引导的特性。但安装某些多重引导功能的软件或 LINUX 的 LILO 时有可能改写它，它先于所有的操作系统被调入内存并发挥作用，然后才将控制权交给活动主分区内的操作系统。所以在操作系统内我们无法直接将文件复制到第一个扇区。

3、Loader 的作用有哪些

bootLoader:初始化硬件设备、建立内存空间的映射图，从而将系统的软硬件环境带到一个合适的状态，以便为最终调用操作系统内核准备好正确的环境

Loader 是操作系统的一部分，负责程序的加载。它是程序运行中不可或缺的一个步骤，加载器会将程序置放在存储器中，让它开始运行。加载程序的步骤包括，读取可执行文件，将可执行文件的内容写入存储器中，之后开展其他所需的准备工作，准备让可执行文件运行。当加载完成之后，操作系统会将控制权交给加载的代码，让它开始运作。

4、L1、L6 各标识了一个字节（8bit）的数据，eax 是一个 16 位的寄存器，说明下面这行代码的意思

行号 代码

1 mov al, [L1]——将 L1 中的内容传给 al

2 mov eax, L1——将 L1 的地址传给 eax

3 mov [l1], ah——将 L1 中的内容赋值为 ah

4 mov eax, [L6]——将 L6 中的内容传给 eax

5 add eax, [L6]——将 L6 中的内容加到 eax 上去

6 add [L6], eax——将 eax 中的内容加到 L6 存储的内容上去

7 mov al, [L6]——将 L6 中的内容赋值给 al

5、times 510-($-$$) db 0中的数字为什么是 510? $ 和 $$ 分别表示什么？不用 times 指令怎么写（等价命令）？

一个扇区 512 字节，减去结束标志 0xAA55 占用的两个字节，正好剩下 510 个字节

$表示当前行被汇编后的地址

$$表示一个节(section)的开始处被汇编后的地址。一个节表示一段代码。这里我们的程序只有一段代码，所以$$实际上就表示程序被编译后的开始地址，也就是 0x7c00

times 510-($-$$) db 0 表示将0这个字节重复510-($-$$)遍，也就是在剩下的空间中不断地填充0，知道程序有510个字节为止。加上结束标志0xAA55占用的2个字节，攻击512个字节。

```
%assign i 1
%rep 510-($-$$)
    db 0
%assigni i+1
%endrep
```

‘虽然 NASM 的'TIMES'前缀非常有用，但是不能用来作用于一个多行宏，因为它是在 NASM 已经展开了宏之后才被处理的。所以，NASM 提供了另外一种形式的循环，这回是在预处理器级别的：'%rep'。

操作符'%rep'和'%endrep'('%rep'带有一个数值参数，可以是一个表达式；'%endrep'不带任何参数)可以用来包围一段代码，然后这段代码可以被复制多次，次数由预处理器指定。

```
%assign i 0
%rep   64
     inc    word [table+2*i]
%assign i i+1
%endrep
```

这段代码会产生连续的 64 个'INC'指令，从内存地址'[table]'一直增长到'[table+126]'。

6、解释 db 命令： L10 db “w” ， “o” ， “r” ， “d” ，0 这条语句的意义，并且说明数字 0 的作用。

L10 存储表达式第一字节的地址。这条语句声明一个 word 字符串，0 用来标记字符串结尾的位置。

对 C 语言或者汇编语言的字符串来说，只用一个基本数据类型去表示一个字符串，这会存在一个问题，那就是无法保存这个字符串的长度，换句话说我们不知道一个字符串是在哪里结束的。

7、L1 db 0

L2 dw 1000

L1、 L2 是连续存储的吗？即是否 L2 就存储在 L1 之后？

是连续存储。是的

8、要是不知道 L6 标识的是多大的数据，下面这句话对不对？

mov [L6], 1

不对。使用 MOV 指令必须遵循：源操作数和目的操作数类型要一致，即同时为字节或字，不能一个是字一个是字节。不然会报错：error: operation size not specified

9、如何处理输入输出？在代码中哪里体现出来？

1、单个字符：

输入：调用 int 21h 的 01H 功能，即

```
mov ah,01h （带回显键盘输入）
int 21h,
```

入口参数：无

出口参数：AL=读到字符的代码（ASCII 码）

输出：调用 int 21h 的 02H 功能，即

```
mov ah,02h （显示输出）
```

入口参数：DL=要输出的字符（ASCII 码）

出口参数：无

2、多个字符：

可以考虑循环输入单个字符的方式，但更可以用下面的方式：

输入：调用 int 21h 的 0AH 功能，将键盘输入的字符串读入缓冲区，即

```
mov ah,0AH
int 21h
```

入口参数：DS:DX=缓冲区首地址

出口参数：接收到的输入字符串在缓冲区中

输出：调用 int 21h 的 09H 功能

```
mov ah,09h
int 21h
```

入口参数：DS:DX=需要输出的字符串的首地址，字符串以字符’$’为结束标志

出口参数：无

10、通过什么来保存前一次的运算结果？在代码中哪里体现出来？

① 通过寄存器来保存前一次运算的结果

```
DispStr:
        mov ax, BootMessage
        mov bp, ax ; es:bp = 串地址
```

② 通过栈

```
push ax
……
pop ax
```

11、随机选择代码段，说明作用。

```
        org 07c00h ; 告诉编译器程序加载到 7c00处
        mov ax, cs
        mov ds, ax
        mov es, ax
        call DispStr ; 调用显示字符串例程
        jmp $ ; 无限循环
DispStr:
        mov ax, BootMessage
        mov bp, ax ; es:bp = 串地址
        mov cx, 16 ; cx = 串长度
        mov ax, 01301h ; ah = 13, al = 01h
        mov bx, 000ch ; 页号为 0(bh = 0) 黑底红字(bl = 0Ch,高亮)
        mov dl, 0
        int 10h ; 10h 号中断
        ret
BootMessage:
        db "Hello, OS!"
        times 510-($-$$) db 0 ; 填充剩下的空间，使生成的二进制代码恰好为
        dw 0xaa55 ; 结束标志
```

12、有哪些段寄存器？

四个段寄存器：

代码段寄存器 CS(Code Segment)

数据段寄存器 DS(Data Segment)

堆栈段寄存器 SS(Stack Segment)

附加段寄存器 ES(Extra Segment)

13、8086/8088 存储单元的物理地址长， CPU 总线的数量，可以直接寻址的物理地址空间。

存储单元的物理地址长为：20 位，范围是 00000H 至 FFFFFH

CPU 有 20 根地址总线

可以直接寻址的物理地址空间是 1M 字节（$=2^{20}$）

14、如何根据逻辑地址计算物理地址？

1M 字节地址空间最多可划分 64K 个，最少 16 个逻辑段

所以段起始地址可以表示成 XXXX0（16 进制）——XXXX 称为段值

于是，存储单元的逻辑地址可以表示为 段值：偏移

物理地址=段值*16+偏移（二进制左移四位，十六进制左移 1 位）

15、寄存器的寻址方式（知道如何计算）。

立即寻址：

MOV AX,1234H

立即数可以是 8 位，也可以是 16 位（高高低低）

主要用于给寄存器和存储单元赋初值的场合

寄存器寻址：

MOV SI,AX

MOV AL,DH

16 位操作数寄存器是 AX, BX, CX, DX, SI, DI, SP, BP

8 位操作数寄存器是 AL, AH, BL, BH, CL, CH, DL, DH

直接寻址：

直接寻址的地址放在方括号中

寄存器间接寻址：

寄存器存放的是要的数值的地址

假设有指令：MOV BX, [DI]，在执行时，(DS)=1000H，(DI)=2345H，存储单元 12345H 的内容是 4354H。问执行指令后，BX 的值是什么？

解：根据寄存器间接寻址方式的规则，在执行本例指令时，寄存器 DI 的值不是操作数，而是操作数的地址。该操作数的物理地址应由 DS 和 DI 的值形成，即：

PA=(DS)*16+DI=1000H*16+2345H=12345H。

寄存器相对寻址：

指定的寄存器内容，加上指令中给出的位移量（8 位或 16 位），并以一个段寄存器为基准，作为操作数的地址。指定的寄存器一般是一个基址寄存器或变址寄存器。

假设指令：MOV BX, [SI+100H]，在执行它时，(DS)=1000H，(SI)=2345H，内存单元 12445H 的内容为 2715H，问该指令执行后，BX 的值是什么？

解：根据寄存器相对寻址方式的规则，在执行本例指令时，源操作数的有效地址 EA 为：

EA=(SI)+100H=2345H+100H=2445H

该操作数的物理地址应由 DS 和 EA 的值形成，即：

PA=(DS)*16+EA=1000H*16+2445H=12445H。

基址+变址寻址：

假设指令：MOV BX, [BX+SI]，在执行时，(DS)=1000H，(BX)=2100H，(SI)=0011H，内存单元 12111H 的内容为 1234H。问该指令执行后，BX 的值是什么？

解：根据基址加变址寻址方式的规则，在执行本例指令时，源操作数的有效地址 EA 为：

EA=(BX)+(SI)=2100H+0011H=2111H

该操作数的物理地址应由 DS 和 EA 的值形成，即：

PA=(DS)*16+EA=1000H*16+2111H=12111H

相对基址+变址寻址：

假设指令：MOV AX, [BX+SI+200H]，在执行时，(DS)=1000H，(BX)=2100H，(SI)=0010H，内存单元 12310H 的内容为 1234H。问该指令执行后，AX 的值是什么？

解：根据相对基址加变址寻址方式的规则，在执行本例指令时，源操作数的有效地址 EA 为：

EA=(BX)+(SI)+200H=2100H+0010H+200H=2310H

该操作数的物理地址应由 DS 和 EA 的值形成，即：

PA=(DS)*16+EA=1000H*16+2310H=12310H

16、几个常用指令的作用（如 MOV， LEA 等）。

MOV:传送指令

MOV DST,SRC

把一个字或字节从源操作 SRC 数送至目的操作数 DST

内部寄存器之间的数据传送，立即数送至通用寄存器或存储单元，寄存器与存储器的数据传送

遵守以下规定：

源操作数和目的操作数类型要一致，即同时为字节或字，不能一个是字一个是字节

除了串操作指令外，源操作数和目的操作数不能同时是存储器操作数

把 CS 的内容送到 DS

```
MOV AX,CS
MOV DS,AX
```

LEA：地址传送指令

LEA REG,OPRD

把操作数 OPRD 的有效地址传送到操作数 REG

操作数必 OPRD 必须是一个存储器操作数

```
LEA AX,BUFFER  ;BUFFER 是变量名
LEA DX,[BX+3]
LEA SI,[BP+DI+4]
```

堆栈操作指令

加减运算指令

逻辑运算指令：NOT,AND,OR,XOR,TEST

无条件转移指令：JMP

循环指令:LOOP,LOOPE,LOOPZ,LOOPNE,LOOPNZ,JCXZ

17、主程序与子程序的几种参数传递方式。

1. 利用寄存器传递参数

把参数放到约定的寄存器中。

优点：

实现简单和调用方便。

缺点：

由于寄存器的个数是有限的，且寄存器往往还要存放其他数据，所以只适用于要传递的参数较少的情况

2. 利用约定存储单元传递参数

在传递参数较多的情况下，可利用约定的内存变量来传递参数。

优点：

子程序要处理的数据或送出的结果都有独立的存储单元，编写程序时不易出错。

缺点：

但是这种方法要占用一定的存储单元。

通用性较差。为了传递较多的参数，又要保持良好的通用性，通常把参数组织成一张参数表，存放在某个存储区，然后把这个存储区的首地址传送给子程序。（既可以利用寄存器传递首地址，也可利用堆栈方法传递首地址）

3.利用堆栈传递参数：

如果使用堆栈传递参数，那么主程序在调用子程序之前，把需要传递的参数依次压入堆栈，子程序从堆栈中取入口参数；如果使用堆栈传递出口参数，那么子程序在返回前，把需要返回的参数存入堆栈，主程序在堆栈中取出口参数。

优点：

利用堆栈传递参数可以不占用寄存器，也无需使用额外的存储单元。

缺点：

由于参数和子程序的返回地址混杂在一起，有时还要考虑保护寄存器，所以较为复杂。

通常堆栈传递入口参数，而利用寄存器传递出口参数。

4.利用 CALL 后续区传递参数

CALL 后续区是指位于 CALL 指令后的存储区域。

主程序在调用子程序之前，把入口参数存入 CALL 指令后的存储单元中，子程序根据保存在堆栈中的返回地址找到入口参数——这种传递参数的方法成为 CALL 后续区传递参数法。

利用 CALL 后续区传递参数的子程序必须修改返回地址。由于这种方法把数据和代码混在一起，所以在 x86 系列汇编语言程序中使用不多。