应该知道的：

0x00007C00 - 0x00007DFF是引导扇区加载位置

0x00000000 - 0x000fffff :在启动过程中会多次使用，比如BIOS等等（1MB）

0x00100000 - 0x00267fff:用于保存软盘的内容(1440KB)

0x00268000 - 0x0026f7ff:空（30KB）

0x270000~0x27ffff是GDT的位置(2KB)

0x26f800~0x26ffff是IDT的位置(64KB)

0x280000~0x2fffff是bootpack.heb(512KB)  
0x00000000-0xffffffff是数据段

0x00280000-0x002fffff是代码段

0x00300000 - 0x003fffff:栈即其他(1MB)

0x00400000 - :空

----------------------------------------------------------  
struct SEGMENT\_DESCRIPTOR {

short limit\_low; // 段界限低16位

short base\_low; // 基地址低16位

char base\_mid; // 基地址中间8位

char access\_right; // 访问权限(P|DPL|S|Type)

char limit\_high; // 段界限高4位 + 标志位(G|D/B|L|AVL)

char base\_high; // 基地址高8位

};  
struct GATE\_DESCRIPTOR {

short offset\_low; // 偏移低16位

short selector; // 目标代码段选择子

char dw\_count; // 参数个数

char access\_right; // 门属性(P|DPL|S=0|Type)

short offset\_high; // 偏移高16位

};

----------------------------------------------------------------------

1. 为什么要从外设中断要从0x20开始：

答：因为INT0x00~0x1f是CPU内部产生的中断，而对于外设的中断就只能使用0x20。

1. 对于操作系统引导程序中为什么要涉及到A20GATE的相关解释

答：通过处理A20GATE就能（开启它）使用所有的内存了，因为他关闭的时候只能使用1MB的内存

1. GDTR0、GDT0、LGDT的关系

答：GDTR 0是一个特殊寄存器，存储 GDT 的基址和大小。

LGDT 是指令，用于将 GDT 的基址和限制加载到 GDTR0 寄存器。

GDT0是一种特定的GDT。0号是空区域，不能定义段

1. 内存容量检查的时候（memtest\_sub）为什么要跳着检查

答：memtest\_sub主要目的不是内存检查，因为内存在启动的时候已经被检查过了，这里是检查内存容量

1. 关于鼠标移动到屏幕边界的问题

答：确保刷新区域不超出屏幕边界。例如，如果要求刷新区域右边界 vx1 超过屏幕宽度，则截断到屏幕实际宽度。

1. 鼠标为什么要设置为透明色？

答：其实鼠标并未全部设置为透明色，只有一些部分设置为了透明色，因为鼠标其实是一个矩形，因此只是有一些地方是透明色。另外在图层刷新的时候，透明色的部分不会覆盖下面的图层，这更能体现鼠标的“形状”。

1. 什么有些地方只有关中断？  
   答：有些开中断操作是通过 io\_store\_eflags(eflags) 隐式实现的
2. 1s中有几次时钟中断？  
   答：100次，这个数字来源是通过设置相关硬件完成的，可以看定时器文件中的相关初始化。
3. 在操作系统引导程序中为什么显卡显示模式设置的数值很奇怪（其实大多数硬件相关的数值都很奇怪，这里抛砖引玉一下）。

注：程序在asmhead.nas中显示部分

答：和VBE(VESA协会制定的，几乎可以通用的设定方法，关于显卡设置)有关，这个是显卡规范使用的BIOS，使用它可以使用显卡的高分辨率功能，而之前的那些（14天前）都只能委屈使用320x200的画面

截止第16天  
问题文档：

1. 为什么引导扇区最后要将CH存放在[0x0ff0]中？  
   答：在内核代码的头文件中有#define ADR\_BOOTINFO 0x00000ff0这个信息，且包括bootinfo结构体，而它的第一位恰好是柱面数。（0x0ff0正好在引导扇区之前，属于BIOS数据区的保留区域），而在之后的很多结构体都是用这样的方式来定义相关信息的，甚至专门对齐了格式，保证不会有填充。
2. 对于引导扇区中未被运行到的部分有什么意义

答：实际上定义了一个FAT12文件系统的引导扇区，包含了有关磁盘结构、文件系统、操作系统标识符等重要的元数据。FAT12是一种较旧的文件系统，常用于较小的存储介质，如软盘。通过这些信息，操作系统可以了解磁盘的布局、文件系统类型、根目录大小、FAT区域的起始位置等，并顺利引导系统启动。

1. 为什么在功能汇编中，会使用到ESP栈指针呢

答：这是因为在 32 位 x86 汇编中，函数参数通常是通过栈传递的。ESP 寄存器指向栈的顶部，[ESP + 4] 获取栈上第一个参数（即 port），而 ESP + 4 是因为栈中第一个参数是函数的返回地址（通常是调用指令的下一条指令的位置），紧接着才是传递给函数的参数。

1. 为什么中断输入输出的时候需要设置成原子性操作

答：竞态条件：避免中断进来的时候，CPU刚好要休眠，导致错过中断

数据一致性：FIFO是共享资源，其内部通过读/写指针管理数据（主要原因）确保指针 更新和数据存取的完整性。可以看出生产者与消费者问题

1. 为什么set\_gatedesc(idt + 0x21, (int) asm\_inthandler21, 2 \* 8, AR\_INTGATE32);段选择子为2 \* 8？

答：因为在分段后的内存中，存放在段内中的地址都是偏移地址，在计算真实地址时需要加上段的基址地址。如asm\_inthandler21存放在内存段中，它函数保存的地址即为这个段内的偏移地址，所以这里的段选择子为2\*8，而不是1\*8（即以数据段0x00000000为基址的“真实地址”）。

1. 为什么在设置定时器超时功能初始化的时候需要进行原子操作？

答：因为这里在往超时计数器中赋值，如果不中断的话，刚赋了高十六位就可能会触发中断，同时对于缓冲区的输入也应该进行原子性操作

1. 为什么任务切换的时候要设置一个没有任何跳转地址的任务A或者说为何任务A的TSS中没有显式跳转地址？

答：

任务A的 执行入口是隐式的，其逻辑已嵌入在 HariMain 的主循环中：

TSS初始化：在 task\_init 中，任务A的TSS通过 task\_alloc 自动初始化，但其 eip 并未直接设置。

隐式执行流：当任务A首次被调度时，CPU从当前代码位置（即 HariMain 的入口）开始执行，因为此时 eip 的值由 任务切换前的CPU状态 自动保存到TSS中。

循环即入口：任务A的代码本质是一个无限循环（for (;;) { ... }），无需显式跳转地址。

注1：任务A与输入输出FIFO缓冲区是关联关系，每当缓冲区输入新数据时（鼠标、键盘、定时器中断都会切换到任务A）

注2：任务A在没事的时候会休眠切换到其他任务

------------------------------------------------------------------------------------------------------------------------------