**Report for Lab04-从实模式进入保护模式**

**Stu：金泽文**

**No.PB15111604**

**实验目的：**

学习并掌握bios中断机制。

学习并理解i386保护模式以及对应的gdt，寻址方法，掌握由实模式切换到保护模式的方法。

**实验内容：**

1. 对实验3中的汇编程序进行增量编程，仍用作启动扇区代码
2. 链接脚本不变，shell脚本根据需要修改，仍制作启动软盘，将启动扇区代码安装到软盘的启动扇区(也可以选择其他介质)
3. 启动运行启动扇区代码，最后进入死循环

**相关原理学习：**

1. bios中断的学习。

参考[BIOS中断-wiki](https://zh.wikipedia.org/wiki/BIOS%E4%B8%AD%E6%96%B7%E5%91%BC%E5%8F%AB)，了解BIOS中int10h，int16h，并参考[BIOS下的汇编-fancymore](http://fancymore.com/reading/assembler-bios.html)，得到以下信息：

* 1. 对于INT 10H：
     1. 对于表示“显示模式”的AH=00功能号，AL为设置的模式，由于我们要求80\*25，故AL应置为02。
     2. 对于表示“写字符串”的AH=13h功能号，BL表示显示字符的属性，如颜色等。ES：BP指向字符串的地址。
  2. 对于INT 16H：
     1. 对于表示“等待按键”的AH=10h功能号，AL得到输入的字符。可以通过比较AL是否为0，来判断是否有任意按键，如果等于0，则一直循环，这一模式也称为“轮询”。当然，如果不想等待，还有AH=11h这个选项，但是在这里我们只需要使用AH=10h。

1. i386保护模式的学习。

·从[保护模式-wiki](https://zh.wikipedia.org/wiki/%E4%BF%9D%E8%AD%B7%E6%A8%A1%E5%BC%8F)得到的信息：

保护模式（Protected Mode），是目前大部分x86操作系统所使用的模式。保护模式下，存储器得到保护，可以避免问题程序破坏其他任务或OS核心存储器。I386保护模式地址总线为32位，寻址空间32GB。

·从《Orange’s：一个操作系统的实现》第10章得到的信息：

保护模式下，虽然段值仍然由原来16位的cs、ds寄存器表示，但此时它们仅仅变成了一个索引，这个索引指向一个数据结构的一个表项，表项中详细定义了段的起始地址、界限、属性等内容。这个数据结构就是GDT（或LDT）。GDT的一个表项叫做描述符(Descriptor)。GDT的作用是提供段式存储机制，这种机制由段寄存器和描述符共同提供。

要将实模式的逻辑地址转化为之后的线性地址，需要准备GDT，并用lgdt加载gdtr。之后，由于历史原因，8086的设计者为了不让寻址超过1MB出现异常，所以采用了回卷的策略。而i386保护模式下，需要避免这一策略带来的问题。所以需要关闭这一策略，这一方法就是打开A20地址线。

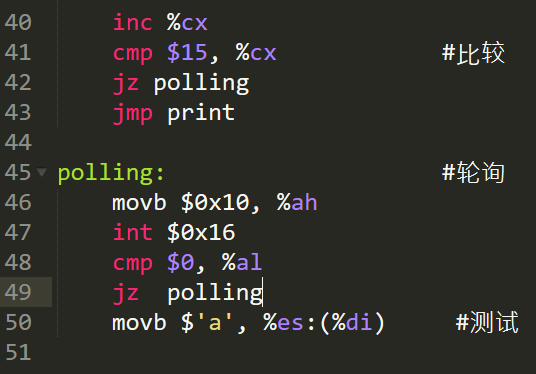
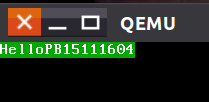
但是在这些前去准备之后，通过什么手段转入到保护模式呢？

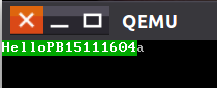
要知道，控制寄存器CR0的PE位即为保护模式与实模式的标志位。0表示实模式，1表示保护模式。只需要将其置为1，就会进入保护模式。不过，还需要将代码段的selector装入cs寄存器中。

还需要注意什么呢？

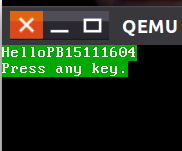
关中断。由于保护模式下，中断处理的机制不同于实模式，如果不关中断，就会出现错误，所以保护模式下的关中断比实模式下更具有必要性。而我们之前代码的第一步即为cli，所以这一步无需设置。

**实验过程（以修改代码为主）：**

1.等待输入。由“相关原理学习”部分，得到如下修改：

并且输入之后

测试成功。

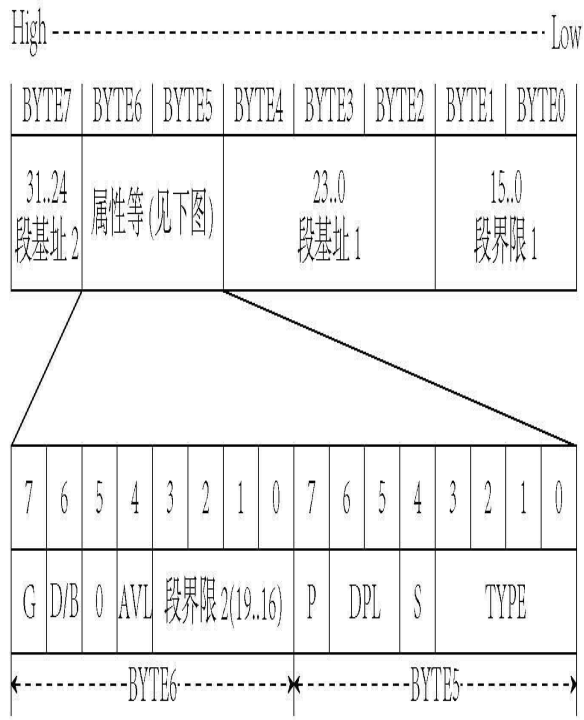
2.实模式下的换行。只需要在输出下一行字符串前将%di设置为下一行开头即可。由于代码很短，所以可以手动设置%di为160,320等。如右所示。

3.进入保护模式。

这一部分是本实验的难点。由“相关原理学习”部分知，要进入保护模式，需要以下五步：

准备GDT，用lgdt加载gdtr，打开A20，置cr0的PE位，跳转，进入保护模式。

i.准备GDT：准备GDT，即准备描述符。我们只需要准备数据段和代码段描述符。

描述符格式如图。

·两个段界限一起表示segment最大长度，一共20位，如果下面的G为1，则将其长度左移12位，否则不左移。

·G位表示段界限粒度，0表示字节，1表示4KB，我们置为1。

·D/B较复杂。Code segment Descriptor中，1表示指令默认使用32位地址，0表示16位；Data segment Descriptor中，1表示段的上界为4GB，0表示64KB。

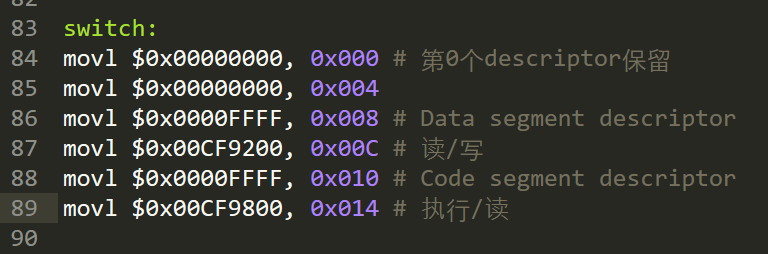
·AVL为保留位。

·第S位“1”表示该段在内存中存在。

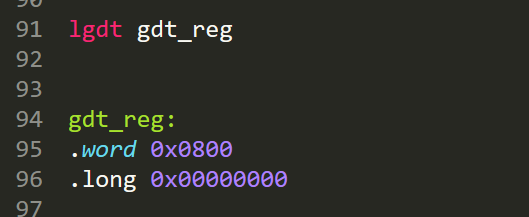
·DPL描述特权级，我们置为0。

·S位描述数据段/代码段还是系统段/门，我们置为1。

·TYPE描述读写执行的类型，我们将ds对应的设置为2，cs对应的设置为8，分别表示读/写，执行/写。

 在实现中，我选择将gdt内容存储在绝对地址0x0处。并根据以上格式，得到如下代码：

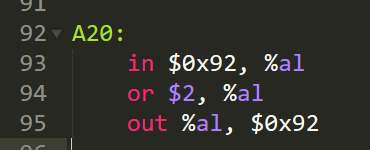
ii.加载gdtr

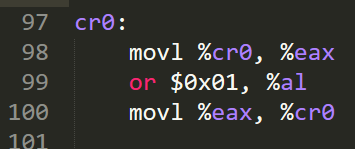
 这一步得到如下代码：

我将基地址设置为0x0，同时将limit设置为0x800。

iii.打开A20

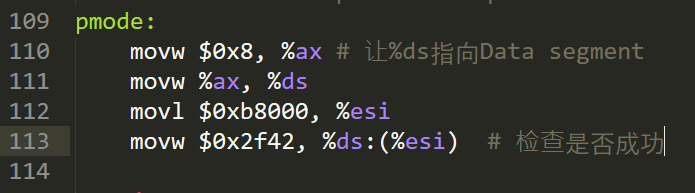
得到如下代码：

 iv.CR0

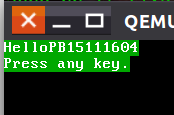
 得到如下代码：

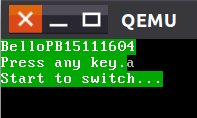
v.跳转

 得到如下代码：

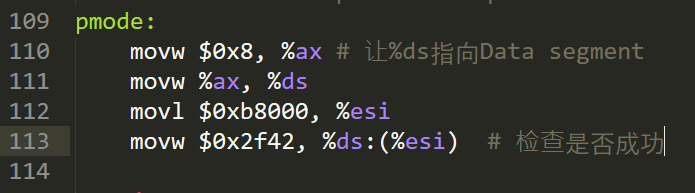
 就此，进入保护模式部分的代码完成。为了检验，得到：

模拟之后得到如下结果：

等待输入：

输入之后：，可以注意到第一个字符变为了‘B’，检查通过。（右边的‘a’是上面为了检查等待过程。）

4.保护模式下的输出

保护模式下的寻址，不同于实模式，不再是 段基址：段内偏移，而是selector：段内偏移。回顾上面检验切换到保护模式时输出字符‘B’的代码

 由于设置的ds的基地址为0x0，故将esi置为0xb8000，即可直接通过%ds:(%esi)得到vga显存地址。下面通过这个原理，得到输出字符串的代码：

**要求的内容**：

1. 如何在实模式下利用BIOS实现任意键的获取？

利用BIOS 16h中断。对于表示“等待按键”的AH=10h功能号，AL得到输入的字符。可以通过比较AL是否为0，来判断是否有任意按键，如果等于0，则一直循环，这一模式也称为“轮询”。当然，如果不想等待，还有AH=11h这个选项，但是在这里我们使用AH=10h。

1. 什么是保护模式？要进入保护模式需要进行哪些准备？

对第一个问题，【相关原理学习】中的保护模式部分有详细的解答。

对第二个问题，【实验过程】中第3点“进入保护模式”部分有详细的解答。

1. 【附加】如何在实模式下利用BIOS实现VGA显示？能不能在保护模式下利用BIOS实现IO？能或者不能都说出你的理由。

实模式下，可以利用bios中断机制，通过int 10h等，实现vga的显示，根据输出的需求，设置功能号%ah的值，比如%ah=0x13表示写字符串，%ah=0x0e表示写字符等。

而在保护模式下，严格来说是可以通过bios中断实现的。正常情况下的保护模式虽然不能利用bios，因为bios中断所使用的是16位实模式环境。但是在保护模式下，可以利用“虚拟8086模式”来模拟一个虚拟的较为逼真的实模式环境。在这个虚拟8086模式中，可以通过bios中断来实现IO。虚拟8086模式作为任务运行在保护模式下，所以虚拟8086模式下的bios中断实现，当然可以看做是保护模式下的实现。

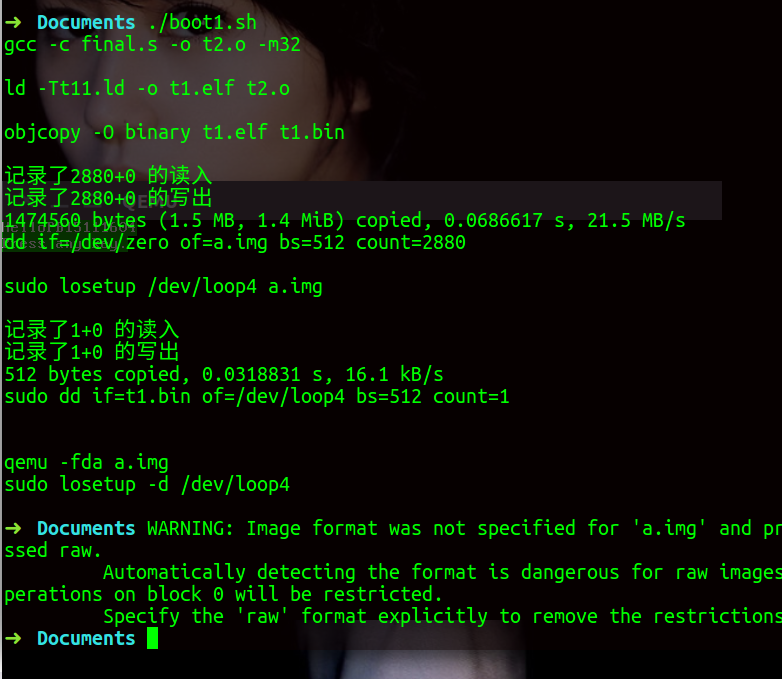
1. 保护模式下写VGA缓存和实模式下写VGA缓存有什么不一样？

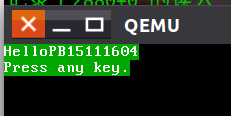
寻址方式会略有不同。比如，虽然我用的都是movsb这个伪指令，但是在实模式下，这个指令所存取的 %ds:(%si)和%es:(%di)都是实模式的寻址方式。ds，es分别代表的就是段基址。而在保护模式下，%ds:(%esi)和%es:(%edi)都是保护模式的寻址方式，ds，es分别代表的是selector，也就是选择子，是通过gdt下的描述符的index实现的，而不是实模式下左移4位实现的。

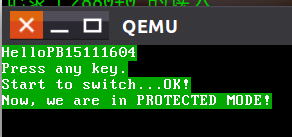
1. 你是如何实现回车功能的？

实模式下，只需要在输出下一行字符串前将%di设置为下一行开头即可。由于代码很短，所以可以手动设置%di为160,320等。

1. 给出代码运行关键时刻的截屏并加以说明

运行脚本：

等待输入：

进入pmode：

1. 给出你所有代码的流程图