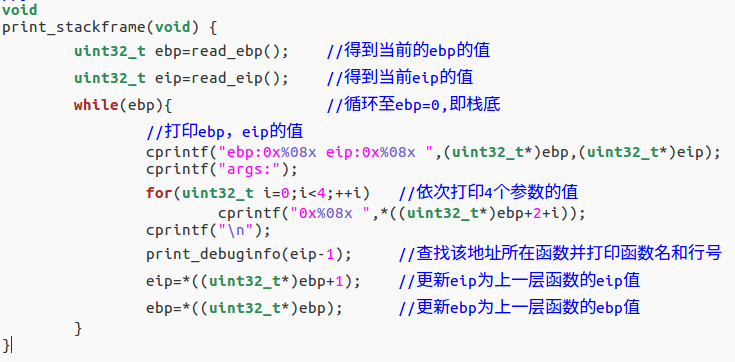
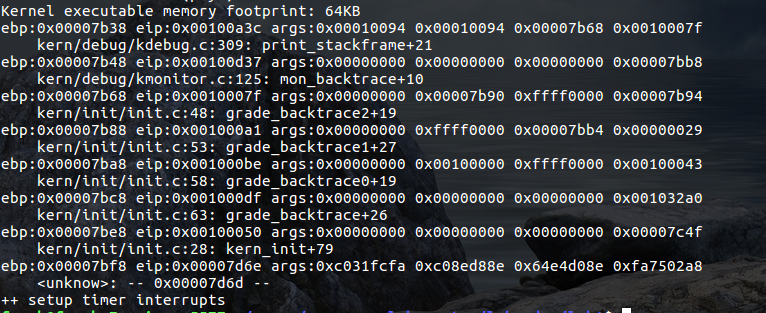
**实验5--实现函数调用堆栈跟踪函数**   
需要完成kdebug.c中函数print\_stackframe的实现，可以通过函数print\_stackframe来跟踪函数调用堆栈中记录的返回地址。   
一．函数堆栈   
　　主要的两点在于栈的结构和ebp寄存器的作用。一个函数调用动作可分解为：零到多个PUSH指令（用于参数入栈），一个CALL指令。CALL指令内部其实还暗含了一个将返回地址（即CALL指令下一条指令的地址）压栈的动作（由硬件完成）。几乎所有本地编译器都会在每个函数体之前插入类似如下的汇编指令：   
　　   
　　这样在程序执行到一个函数的实际命令前，已经有以下数据顺序入栈：参数，返回地址，ebp寄存器。 　　  
　　函数调用的步骤：   
　　　　1.参数入栈：将参数从右向左依次压入栈中。    
    　　　2. 返回地址入栈：call指令内部隐含的动作，将call的下一条指令入栈，由硬件       完成。    
    　　　3. 代码区跳转：跳转到被调用函数入口处。    
    　　　4. 函数入口处前两条指令，为本地编译器自动插入的指令，即将ebp寄存器入栈，       然后将栈顶指针esp赋值给ebp。   
　　相反的，函数返回的步骤为：   
　　　　1.保存返回值，通常将函数返回值保存到寄存器EAX中。    
    　　　2. 将当前的ebp赋给esp。    
    　　　3. 从栈中弹出一个值给ebp。    
    　　　4. 弹出返回地址，从返回地址处继续执行。   
　　并且在函数调用过程中的ebp起着关键作用，从该地址向上（栈底方向）能依次获取返回地址、参数值，向下（栈顶方向）能获取函数局部变量值，而该地址处又存储着上一层函数调用时的ebp的值，于是以此为线索可以形成递归，直至到达栈底。这就是函数调用栈。   
   
二．print\_stackframe函数的实现   
　　由以上知识和源代码文件中的注释实现print\_stackframe():   
 　　  
　　执行’make qemu’指令得到的结果为：   
 　　  
　　可以观察到显示结果与实验指导书上一致。对于最后一行：其对应的是第一个调用堆栈的函数，即bootmain.c中的bootmain函数，因为bootloader设置的堆栈从0x7c00开始，执行’call bootmain’转入bootmain函数。其ebp=0x00007bf8，此时的eip=0x00007d6e，其压入的4个参数分别为0xc031fcfa, 0xc08ed88e, 0x64e4d08e, 0xfa7502a8。