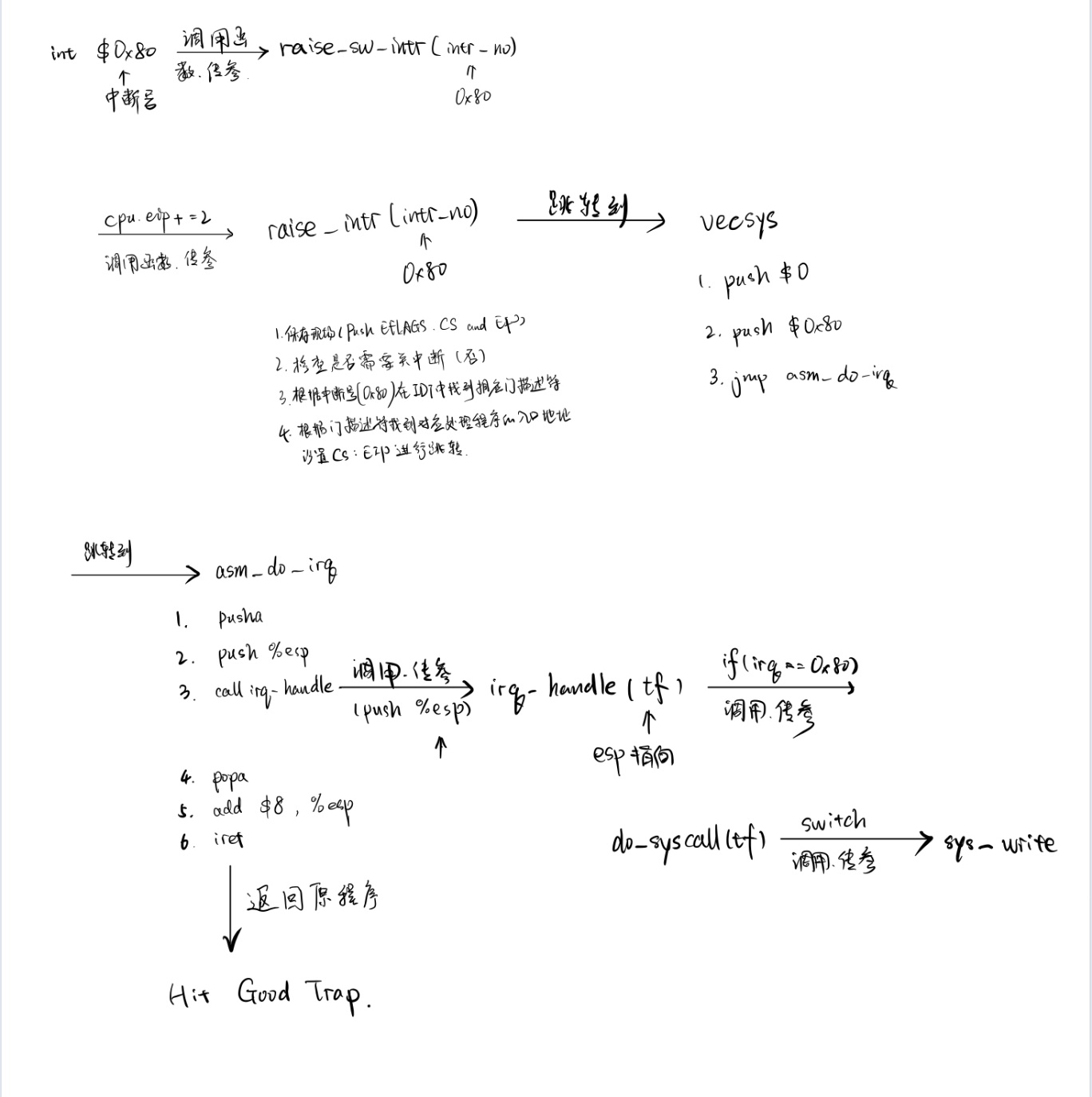
**\* 实验报告要求**

除了上述代码实验，在实验报告中还需要回答如下问题：

**§4-1.3.1 通过自陷实现系统调用**

详细描述从测试用例中的int $0x80开始一直到HIT\_GOOD\_TRAP为止的详细的系统行为（完整描述控制的转移过程，即相关函数的调用和关键参数传递过程），可以通过文字或画图的方式来完成；



1. 在描述过程中，回答kernel/src/irq/do\_irq.S中的push %esp起什么作用，画出在call irq\_handle之前，系统栈的内容和esp的位置，指出TrapFrame对应系统栈的哪一段内容。

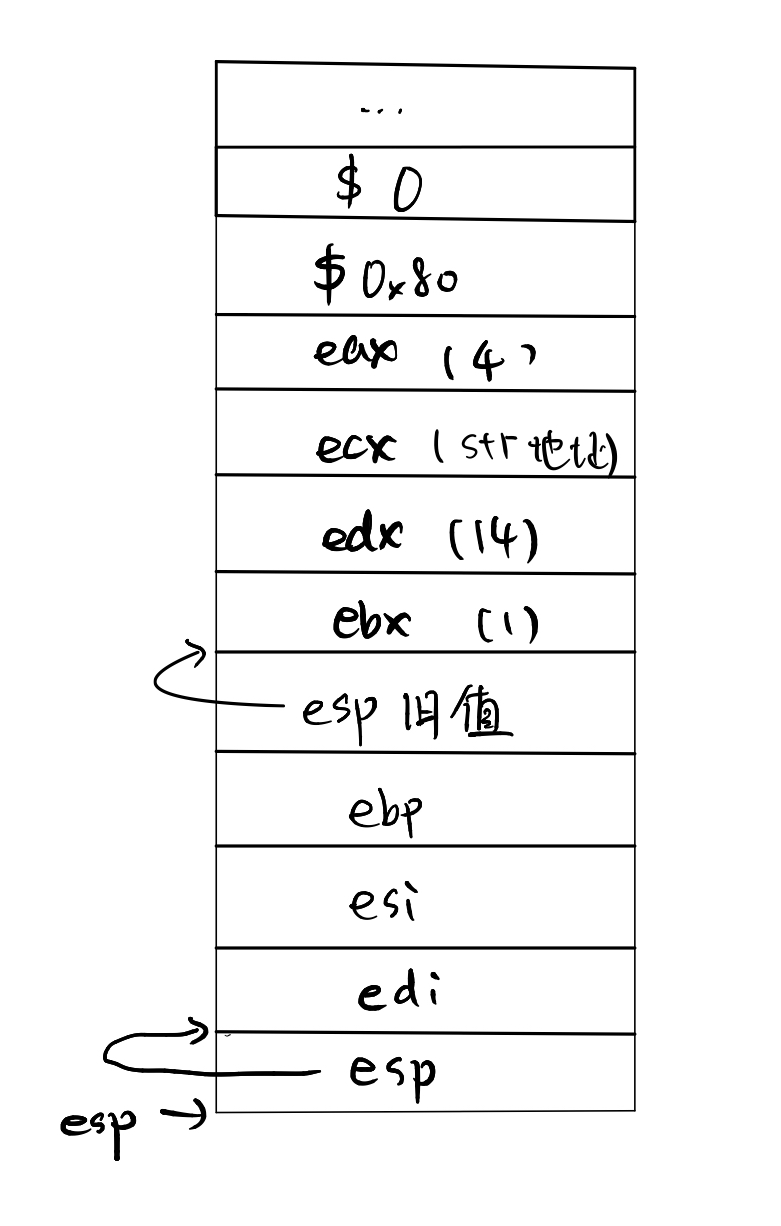
前面调用了pusha，然后esp也就是前面push了栈顶地址，把这个地址压栈，就是把栈中信息的地址作为指针进行传参，刚好参数是TrapFrame类型。

typedef struct TrapFrame {

uint32\_t edi, esi, ebp, xxx, ebx, edx, ecx, eax;

int32\_t irq; uint32\_t error\_code; uint32\_t eip, cs, eflags; } TrapFrame;

系统栈的部分内容：



**§4-1.3.2 响应时钟中断**

1. 详细描述NEMU和Kernel响应时钟中断的过程和先前的系统调用过程不同之处在哪里？相同的地方又在哪里？可以通过文字或画图的方式来完成。

不同之则处主要在

1.系统调用由内部的指令直接中断并直接得到中断号（如0x80）而响应时钟中断是执行每一条指令之后检查中断，并根据中断控制器（i8259）得到中断号。

2.系统调用不用考虑关闭中断的问题，而时钟中断需要在raise\_intr中关中断。

3.系统调用需要提前传入系统调用号到eax中，而时钟中断则不是。

相同之处在其获取中断号后的一系列处理过程是类似的，与4-1.3.1 通过自陷实现系统调用中的过程基本一致。

