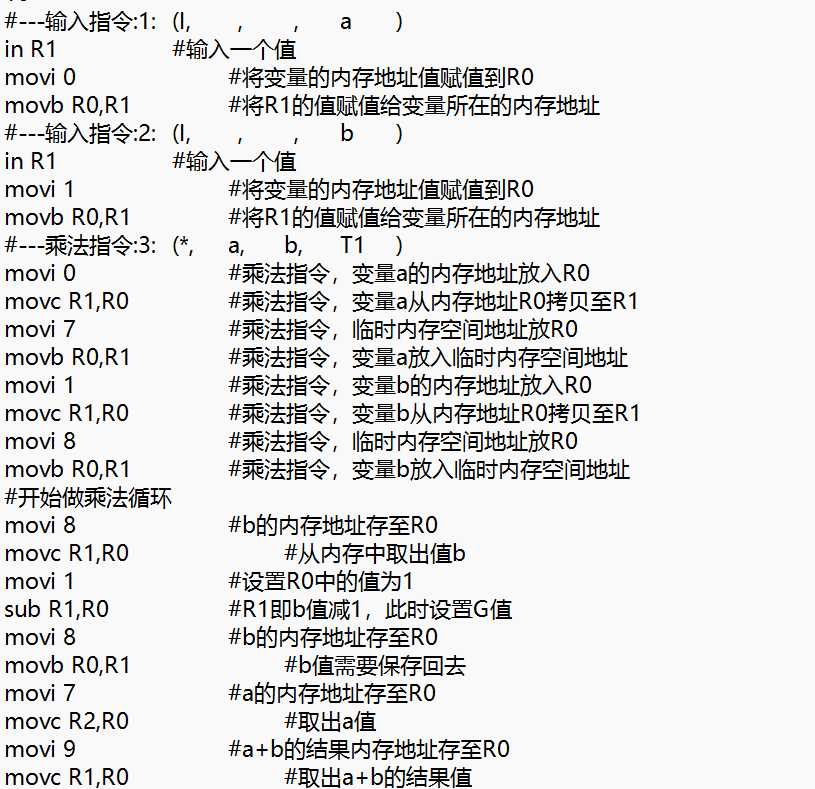
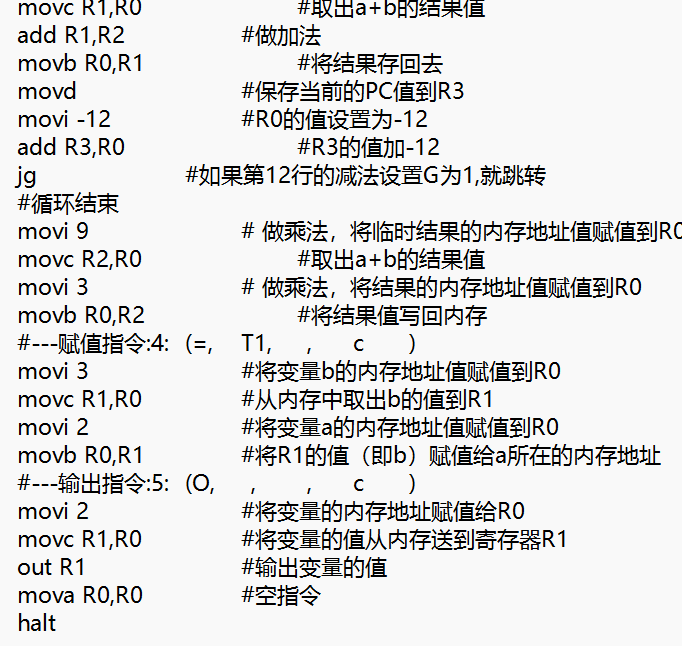
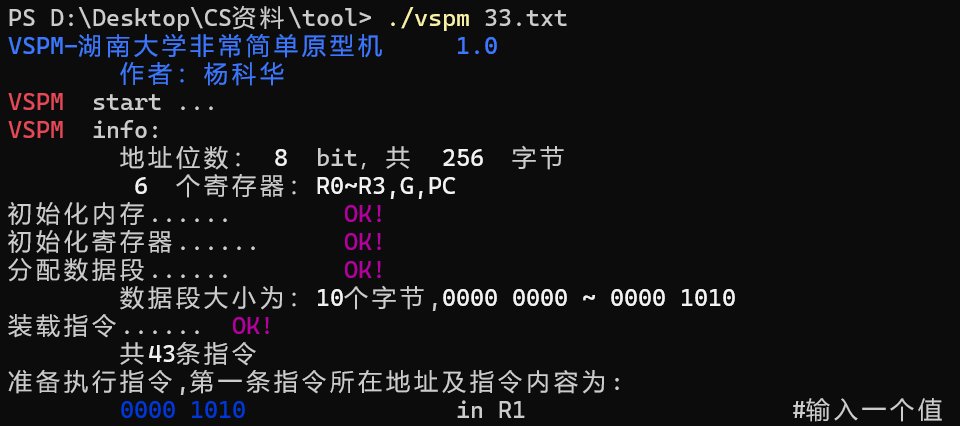
33\_3实现两个正整数相乘。

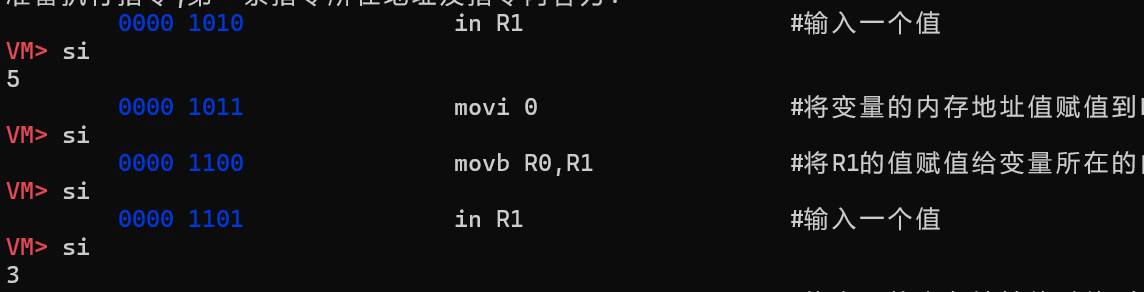




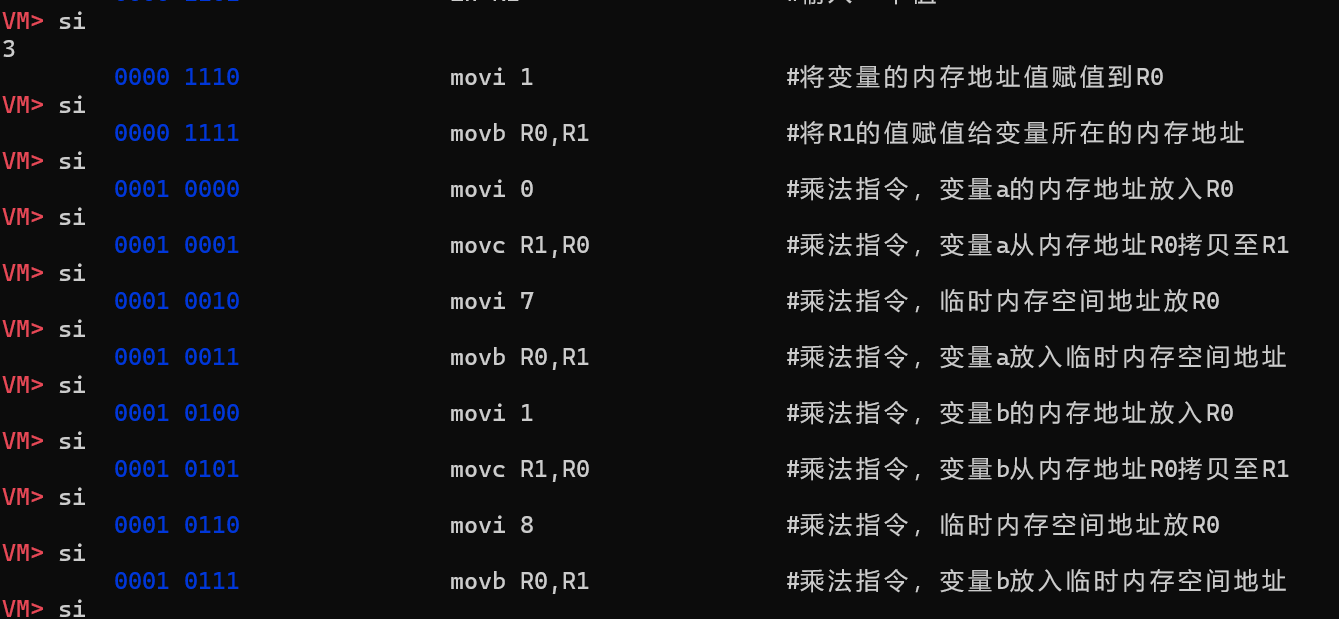
在系统中执行该文件：



文件在原型机中打开的样子。



输入两个乘数：5和3

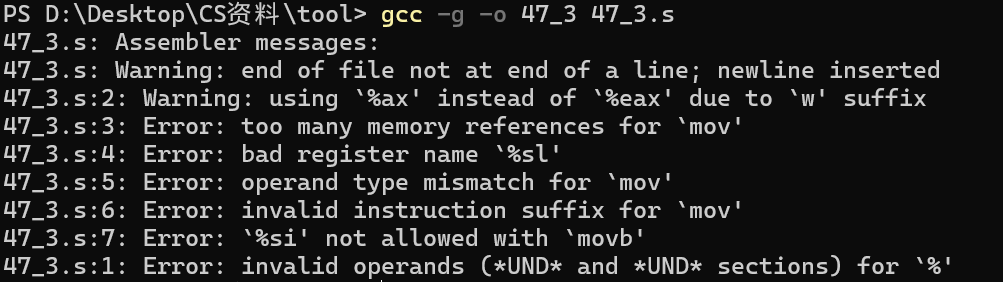


执行乘法指令



最后我们得到结果输出：15

47\_3



第一行是使用 % 符号的方式不符合语法要求。

可以看到第二行的错误是对于movw，我们应该使用%ax而不是使用%eax，因为“w”是对于16位的操作而%eax是三十二位寄存器。

第三行的错误是对于mov，使用了两个内存引用。



我们看到两个寄存器都带有括号，我们需要将（%eax）改成%eax：



第四行的错误是%sl是一个名字错误的寄存器，参照寄存器表，这里的sl正确拼写应该为si，同时注意到%si为十六位而%al是八位，我们需要改%al为%ax，：



第五行的错误是mov 指令的源和目的操作数类型不匹配

“movl %eax, $0xFFFFFFFF”,指令中不能将一个寄存器中的数给一个立即数，我们需要将两者位置互换：



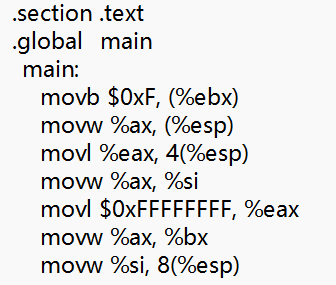
第六行的错误是mov 指令的源和目的操作数类型不匹配：

movw %eax, %bx，%eax是三十二位寄存器而movw和%bx都是十六位，所以要把%eax改成%ax：



第七行的错误是%si和movb不匹配，因为%si是16位寄存器但是movb的操作数应该是八位数据。我们将movb改成movw。

至此，所有的错误都被改正：





再进行操作不会报错。

48\_6

指令1：  
addl %ecx,(%eax) 将%ecx的值加到%eax值对应的地址中的值上面，我们看题目上面的表知道，此时%eax=0x100，而地址0x100对应的值为0xFF，%ecx的值为0x1，两者相加得到更新后的地址0x100的值为0x100.

指令2：

Subl %edx,4(%eax) 指令的目的是将%edx从 %eax值对应寄存器中的值加上偏移量4后对应的地址中的值减去。%edx此时的值为0x3，%eax值对应地址中的值为0x100，得到的地址为0x100+4等于0x104,查表得到该地址对应的值为0xAB。故更新后的地址0x104的值为0xA8.

指令3：

Imull $16,(%eax,%edx,4)，指令的目的是从(%eax+%edx\*4)得到地址，将地址对应的值乘以16之后写回原地址中。%eax=0x100,%edx=0x3,计算得到的地址为0x10C，查表得到该地址对应的值为0x11，计算后得到更新后的地址0x10C对应值为0x176.

指令4：

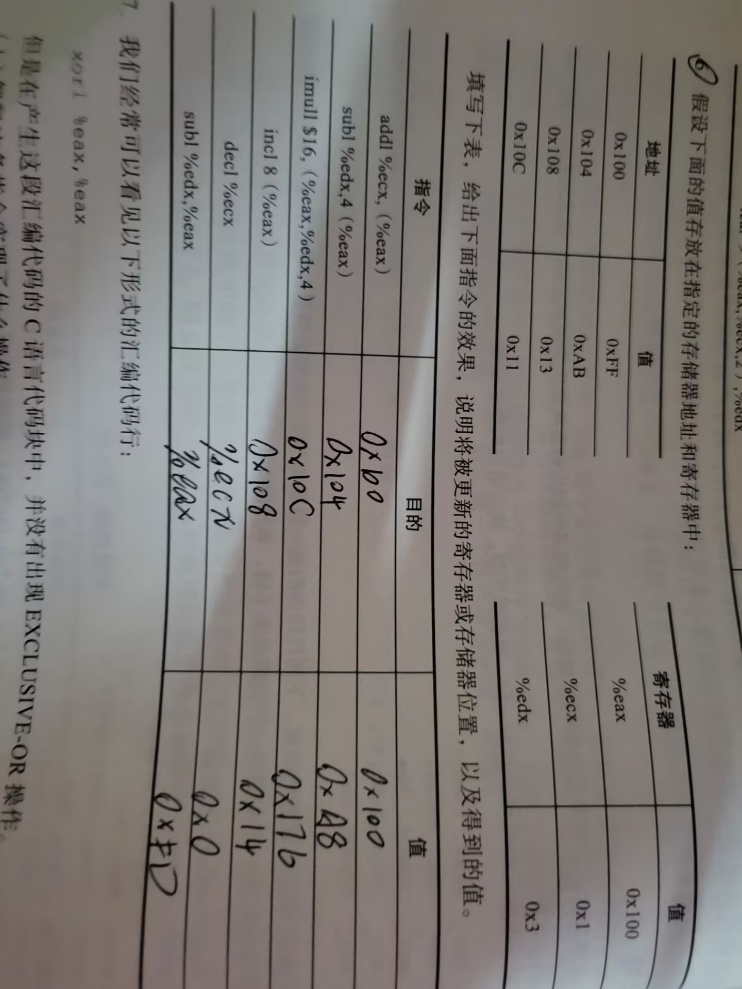
Incl 8 (%eax)，指令目的是将%eax值加8后对应地址中的值加1，地址为0x100+8=0x108，查表得到该值为0x13，加一后更新地址0x108的值为0x14.

指令5：

Decl %ecx，目的是将%ecx的值减一，更新后的%ecx的值为0x0.

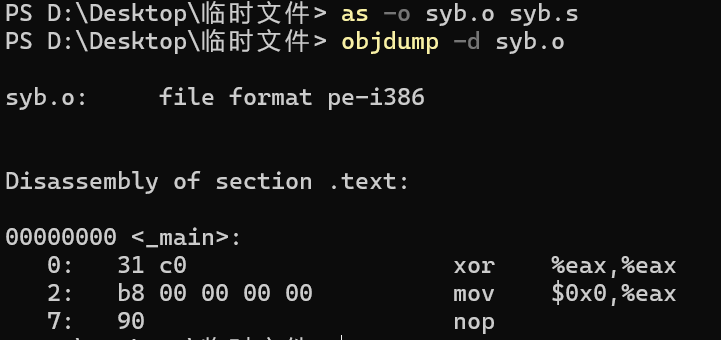
指令6：

Subl %edx，%eax 指令目的是将%edx从%eax中减去，得到更新后的%eax的值为0x100-0x3=0XFD。



47\_7:

1. 这条指令实现了操作将%eax和自己做异或，每一位和自己异或后都变为了0，从而将其值变成0.
2. 更直接地表达这个操作的汇编代码为将%eax直接设置成0：movl $0,%eax。



1. 我们可以看到使用异或只需要2个字节而使用直接赋值需要5个字节。