由一个vc内嵌asm的BUG引出的思考 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022__E7_94_B1_E4_B8_80_E4_B8_AAv_c98_138940.htm 在语法上, 我们通常认为以下的两条语句是等价的: mov ecx, offset DATA_LABLE //其中DATA_LABLE是数据定义标签 lea ecx, DATA_LABLE 而更进一步, 我们也会认为以下两句是等价的: mov ecx, ebp-8 lea ecx, [ebp-8] 第一种, 用的是存储器寻址方式. 而第二种, 用的是寄存器寻址和寄存器间接寻址方式. 让我意想不到的是, 在第二种情况下, vc的处理并没有让寄存器寻址和寄存器间接寻址方式的mov和lea两者之间实现等价. 在使用 _asm{} 的方式将"mov ecx, ebp-8"这条语句括起来编译之后, 很遗憾地, 我在vc的反汇编窗口发现它变成了这样的一条语句: "mov ecx, ebp". 啊哦, 我的"-8"竟然不翼而飞了! 到目前为止, 我尚没有查到造成这种现象的原因所在, 我只能暂时将它归为vc的bug了. 对gcc下会不会存在这个问题呢? 为更进一步证实, 我使用gcc重新写了这句代码: "mov ecx, ebp-8", 但重写后的代码由当初的一句变成了这样的两句: movl , subl $8, 之所以改写成这样的两句, 是因为我发现在ATamp.T语法对寄存器间接寻址方式的支持没有intel asm更具人性化, 但我猜想AT&amp.T之所以采用这样的方法, 可能一定程度上也是为了提高微指令级的执行效率. 当然, "mov ecx, ebp-8"这句也可以改写成这样的两句: subl $8, movl , 但一般不会这么作, 道理是很显然的, ebp通常会作为函数内的基址寄存器, 用于存放函数入口点的堆栈首地址, 这个值的改变会直接影响其后语句对局部变量以及函数参数的引用发生变化, 所以, 在函数首部之后的执行体中, ebp通常是不允许被改变的,

这也是我们设计自己的汇编代码时所应该遵循的原则. 不知道vc为什么会将ebp之后的立即数作丢弃处理, 这显然是没有道德的行为. 这让我想起了这样的一句话: 不要试图帮助用户去纠正错误, 而是当错误发生时去提醒用户, 因为程序再聪明也不会始终明白设计者的真正意图 ,我们所需要作的就是"为异常捕获提供尽可能详细的日志, 并及时通知用户这种异常, 试图纠正异常的作法从方法上就是错误和愚蠢的".