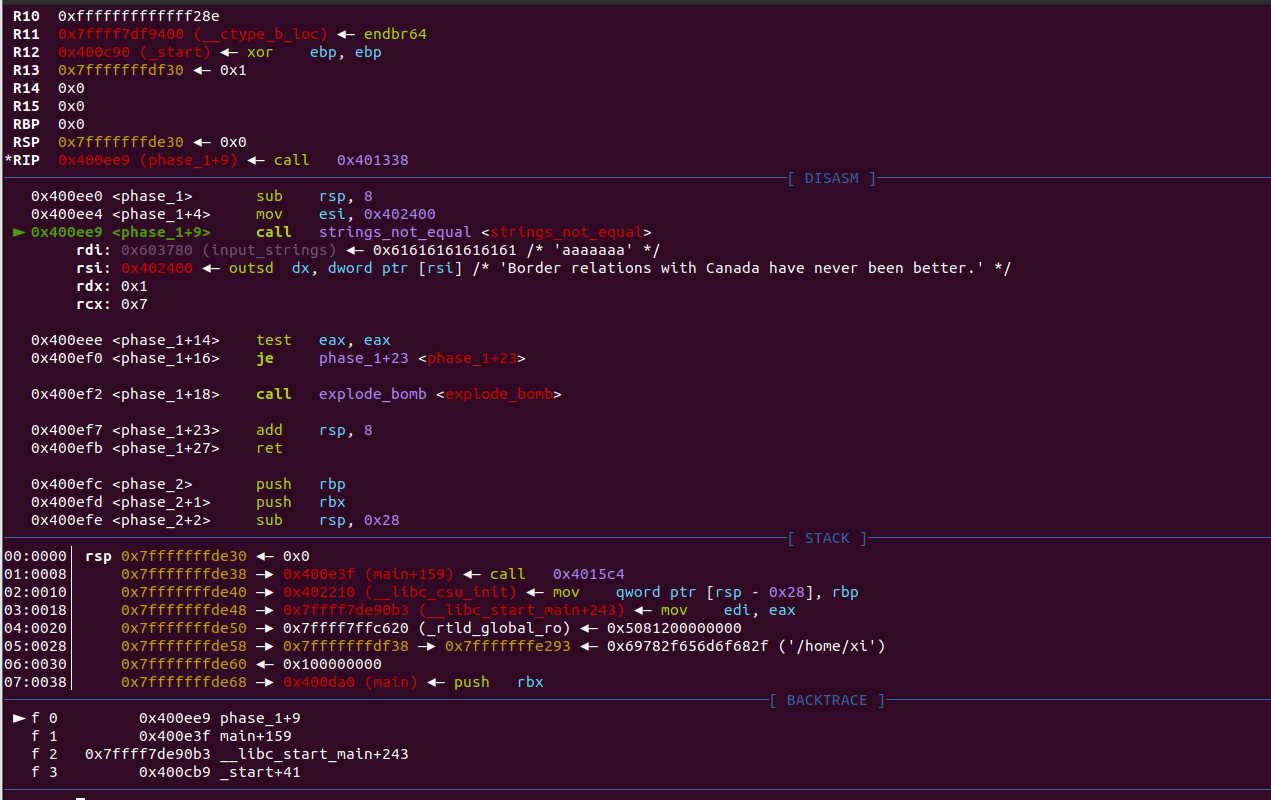
先观察给的.c文件，发现是要输入六组语句并且判断正误的，并且很容易发现判断函数为phase\_i(i=1,2…6)要有一个错误，炸弹即爆炸，那我们就要用到gdb调试了。

在终端输入gdb bomb

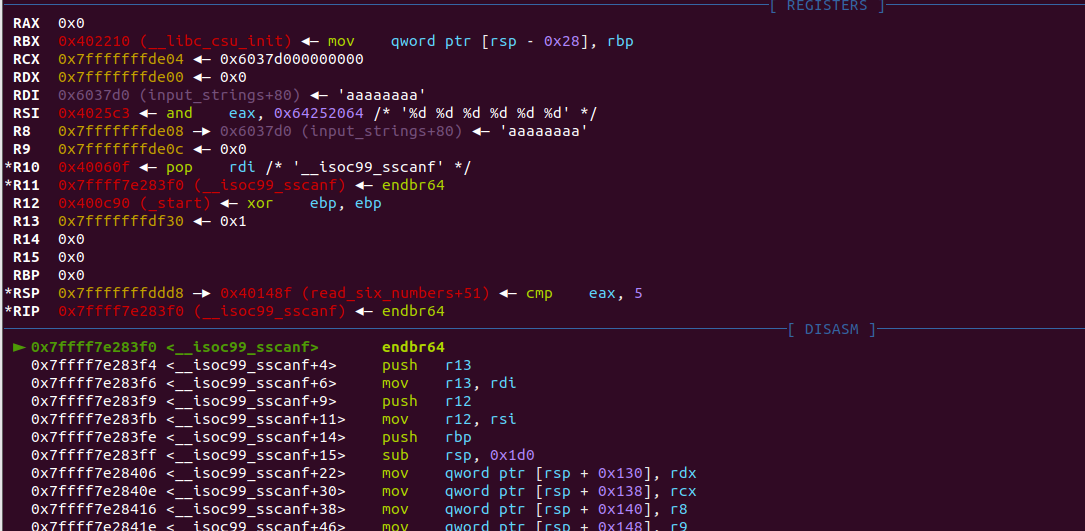
首先在phase\_1处断点（命令：’b phase\_1’）找到phase\_1的拆弹语句。然后’r’运行，先随便输入点东西这里我输入了很多a，断在该处之后单步调试，因为要进入函数内部，我们用命令’step’或者’s’进行单步调试。



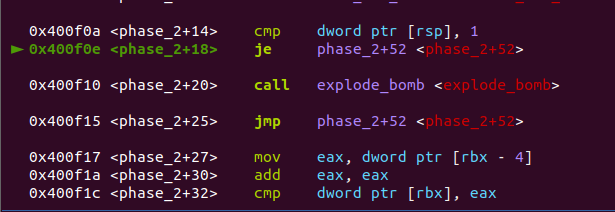
发现strings\_not\_equal函数里面其中一个是我们刚刚输入的很多a，那么显然，另一个就是比较的字符串，所以我们得到拆弹语句：”Border relations with Canada have never been better.”

重新’r’运行之后输入之后发现炸弹没有爆炸，证明phase\_1成功被我们拆掉了，然后同样的道理我们把断点下在phase\_2。在拆掉了第一个炸弹的情况下我们老规矩，输入很多的a，进去’s’单步调试，但是会发现它读取你输入的语句的时候，它的函数名为”read\_six\_numbers”。

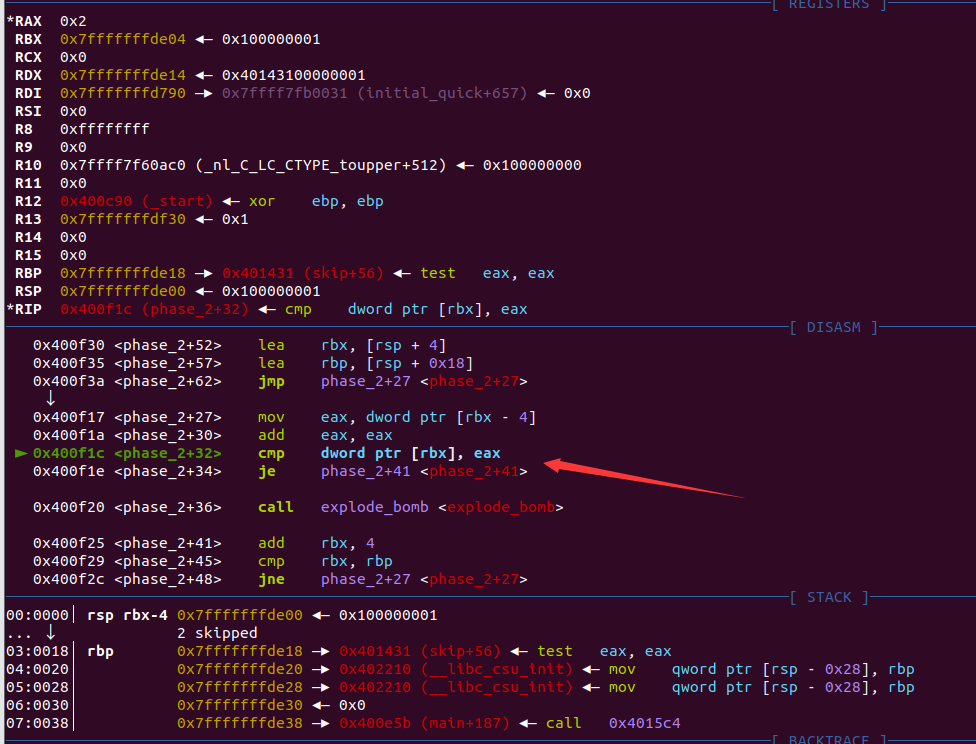
然后不甘心，接着’s’单步调试，发现了寄存器里面一个奇怪的东西。



6个%d，scanf？？这更加确信了这一次输入的不同于第一次，第二次它要输入6个整数，那会是什么整数？那就先猜，就先输入6个0也罢，但此时我们不能’s’单步调试了，因为scanf内部构造很复杂，’s’单步调试会把你键盘按烂的。因此我们用’next’或’n’单步调试，跳过了scanf函数的内部，成功给了6个整数，然后继续调试，发现

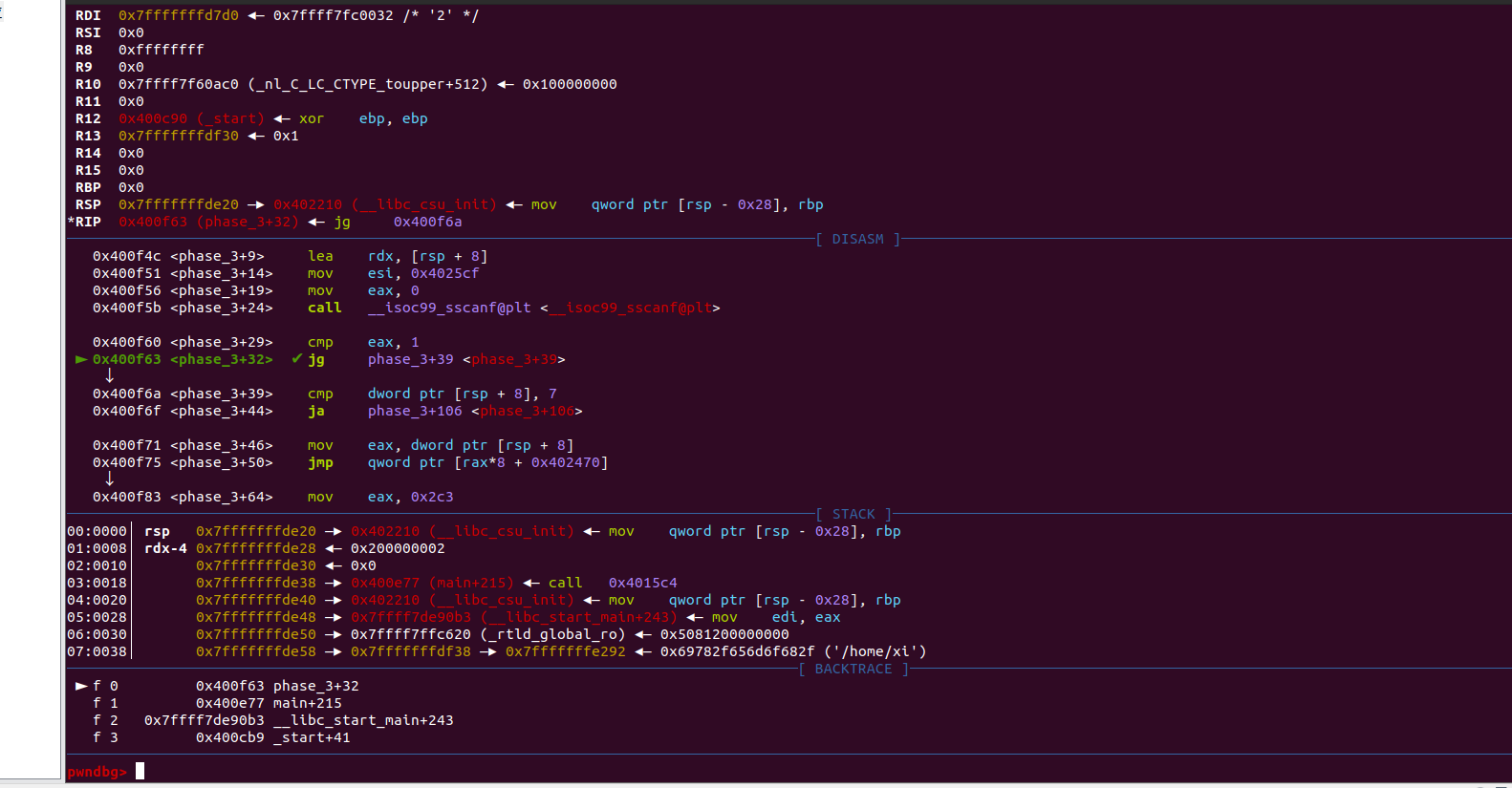


很明显，如果cmp比较的两个数没有相等，那么不触发je后面的跳转下面会直接call explode\_bomb函数，即炸弹就爆炸了，首先看看cmp比较中，有一个立即数是1，那就说明rsp所指向的值应该是1，想想之前那个scanf，他输入了六个整型值栈顶的值应该是先输入的（菜鸡也不知道这跟传参机制有没有关系，从右到左依次入栈，因此栈顶为第一个输入的值）,那么重新运行，到这里我们把第一个值改成1，其它的照常不变，发现第一个数躲过了爆炸，那么说明我们的方案可行，接下来只需要把剩下五个数判断完了就可以了。继续’n’单步调试，发现第二个测试的数是

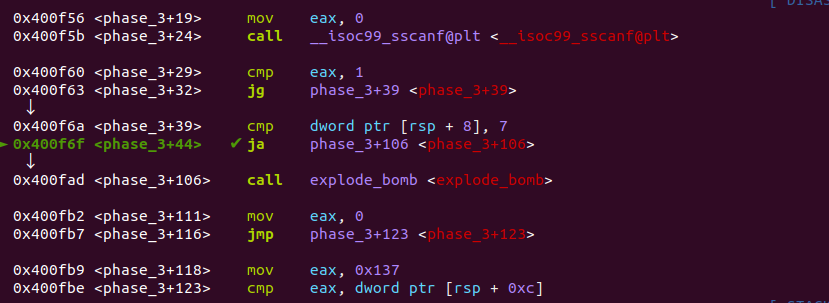


这里不是立即数了，而是一个寄存器eax，那么我们可以通过寄存器窗口查看rax值为2，（eax为rax的低32位，因此rax=2→eax=2，自证应该不难）然后比较的又是rbx寄存器所含地址的，从前面我们也可以看出，rbx=rsp+4，即栈顶往栈底偏移4个字节，也就是输入的下一个int，那么就很容易得出第二个值应该是2，到现在我们已经推出了2个值，同理把正确的值输入剩下的随便输入继续调试，很容易得到这一步输入的语句就是1 2 4 8 16 32这6个数字。’r’重新运行之后输入发现没有爆炸，那么第二个语句也成功被拆除。

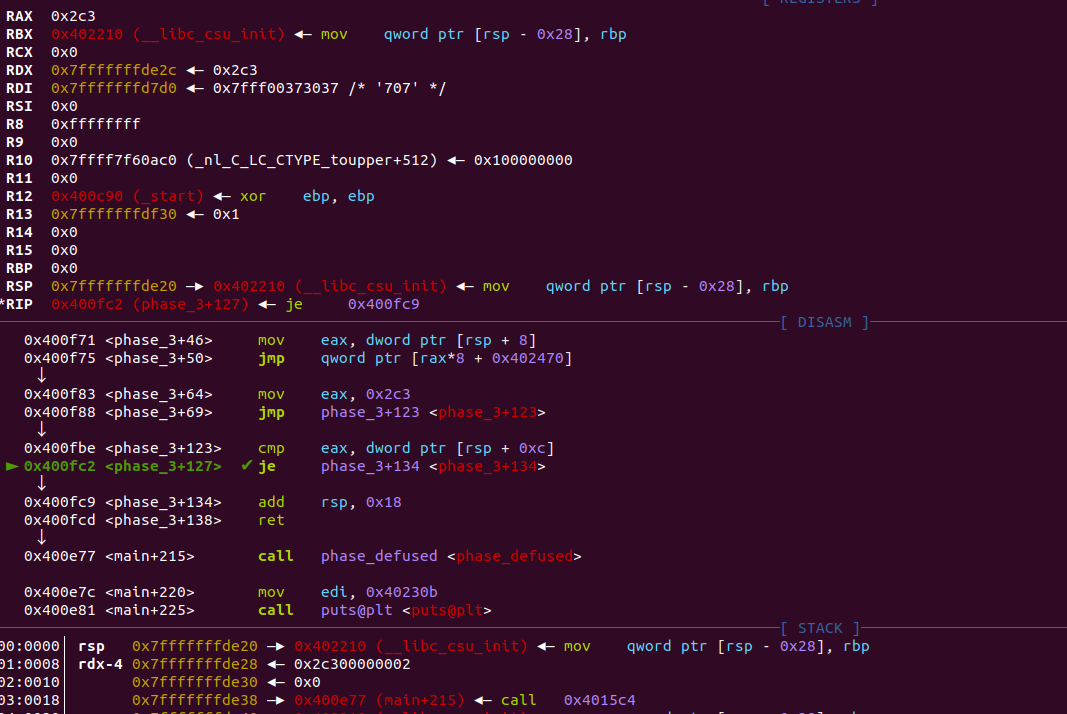
然后断点下在phase\_3，接着运行到那边。运行到scanf发现格式化字符串为”%d %d”，那就是两个整数，并且看到后面有一个cmp eax,1，并且要求eax>1，否则就执行爆炸函数了，eax在scanf之后获得了scanf函数的返回值，scanf的返回值就是输入数的个数。那我们就随便输入俩数看看。



继续调试发现，如果ptr[rsp+8]大于7会发生跳转不妨先输入大于7的值看看会不会跳转，输入之后，好的，成功爆炸，躲不掉的那种哦。

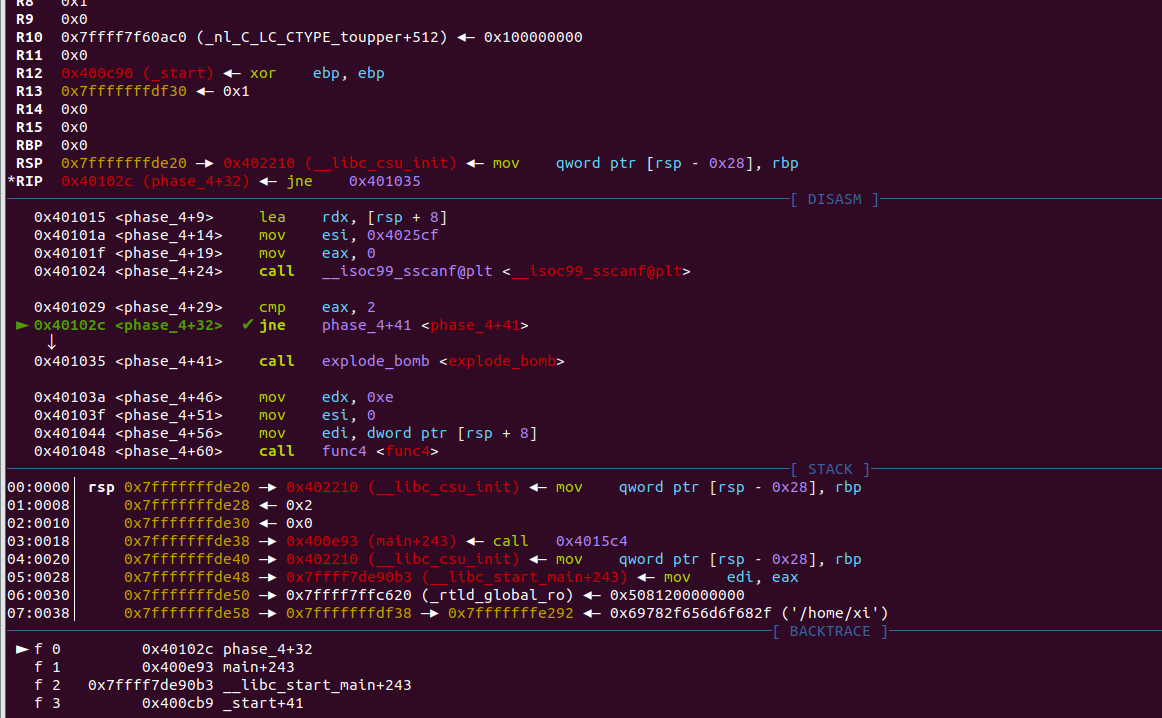


那我们知道了，我们输入的第一个值只能小于等于7，你应该知道，负数在里面可能会成为很大的正整数，所以老老实实，就在0-7里面输入，目前测试是什么都可以的，每个第一个值都有唯一的第二个值与之对应，对应上了就可以拆掉了。这里我输入了2 707，这些自己测试应该不难。



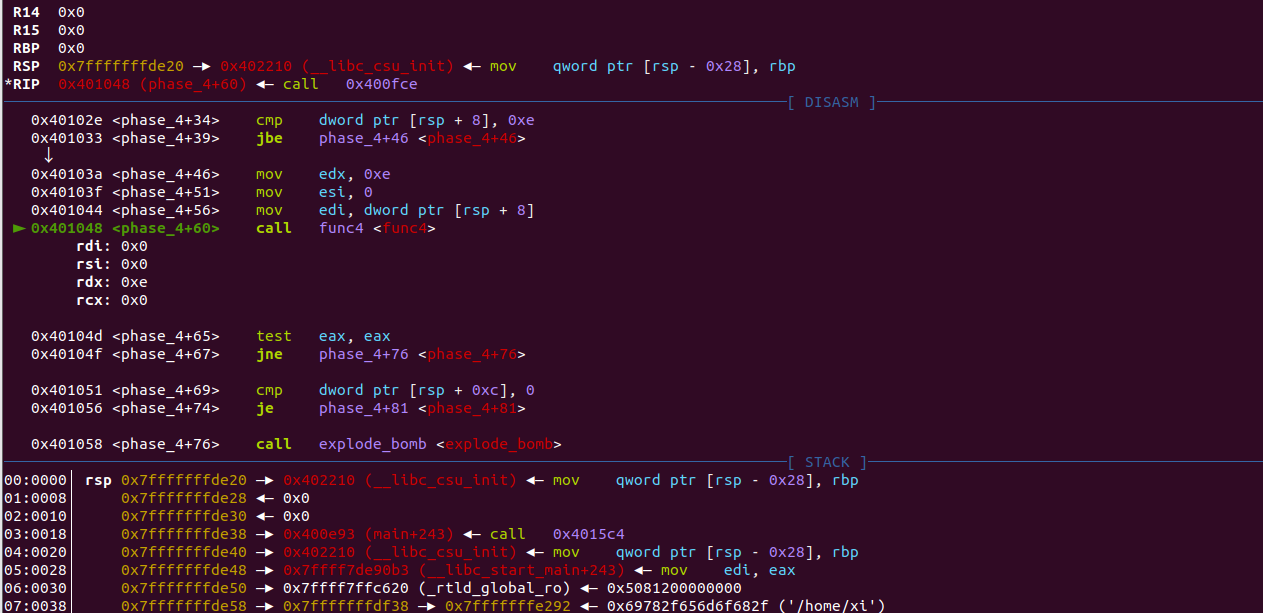
之后发现phase\_3函数成功返回了，自此，我们拆弹完成了一半。

在phase\_4处断点，依次输入前三条拆弹语句，第四句老规矩输入很多a，’s’单步调试进入scanf，发现语句依然是%d %d，好嘛，又是两个整型，那重来，先0 0，’n’单步调试进去发现有一个语句cmp eax,2 jne，jne为jump not equal，不相等则跳转，那不妨重来跟进去看看会跳到哪。



好奇心真不能太重，直接爆了，那它这句就是判断你输入的值是否为2，不为2直接爆，那好，回去老老实实输入两个值。0 0

单步调试到这一步发现在调用函数了



函数有三个参数第一个参数和第二个参数是固定的0xe和0，第三个参数来源于我们输入的值。64位程序调用函数，在call命令执行之前会先从左到右将参数依次传入rdi,rsi,rdx,rcx,r8,r9寄存器，若参数大于六个，剩下的参数从右到左依次入栈。再看看后面有一个判断不相等的跳转语句，这里有人可能想问了，test eax,eax不是肯定相等嘛，那么我们想想cmp如何工作的，cmp rax,rbx就是直接两个一减，然后那个值保存到一个地方，做条件转移的时候就用到那个值，test eax,eax就是一个判断eax是否为0的语句，那么jne其实它也是判断两个值相减是否为0，不为说明不相等。所以这个意思可以理解为eax为0则跳转，不为0则跳转，那么，试过之后发现跳转的那个语句直接爆炸（别问为什么，所有的分支咱都试过了），没办法，那我们只能让eax尽量为0，也就是它的函数返回值(当然这个函数可能没有返回值)。然后下面还有一个跳转语句就是让ptr[rsp+0xc]=0，才能让它跳转而不执行explode函数。那么关键还是看这个函数了，我们’s’单步调试进入这个函数看看它的逻辑关系



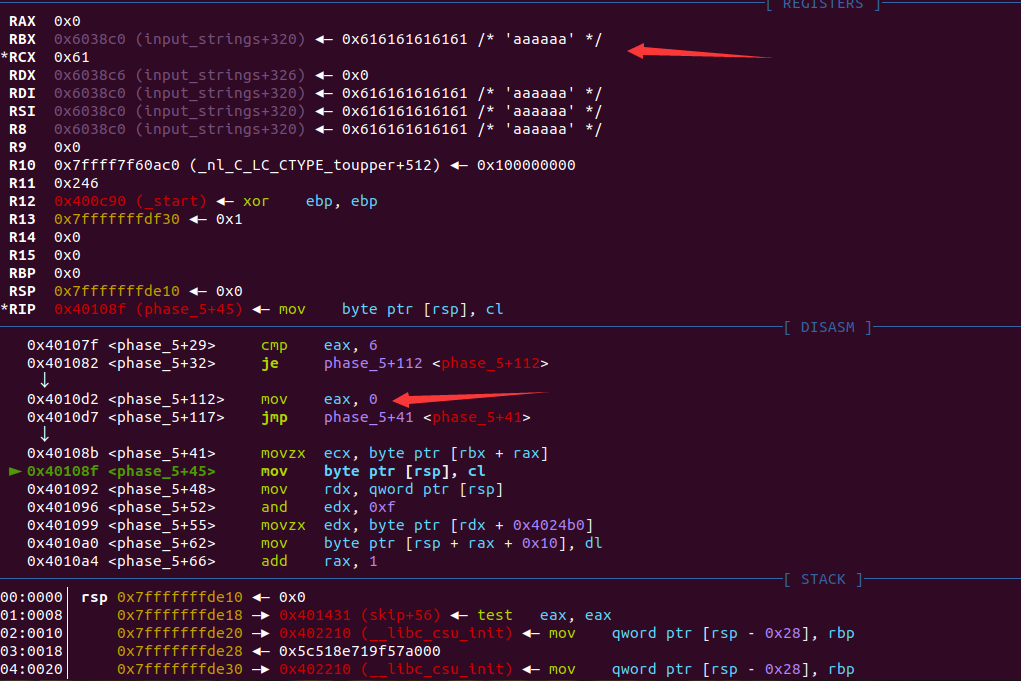
先不妨设这个函数为void func4(int a,int b,int c);

……

然后我也没有细细的分析了，我只知道我0 0就过了qwq；

那我们来看到phase\_5,

一开始随意输入了两个数字，发现这题是要正儿八经的字符串啊，然后下面还检测字符串长度，不等于0直接爆了？行吧，那就先输入6个a试试看。



剩下的待更新