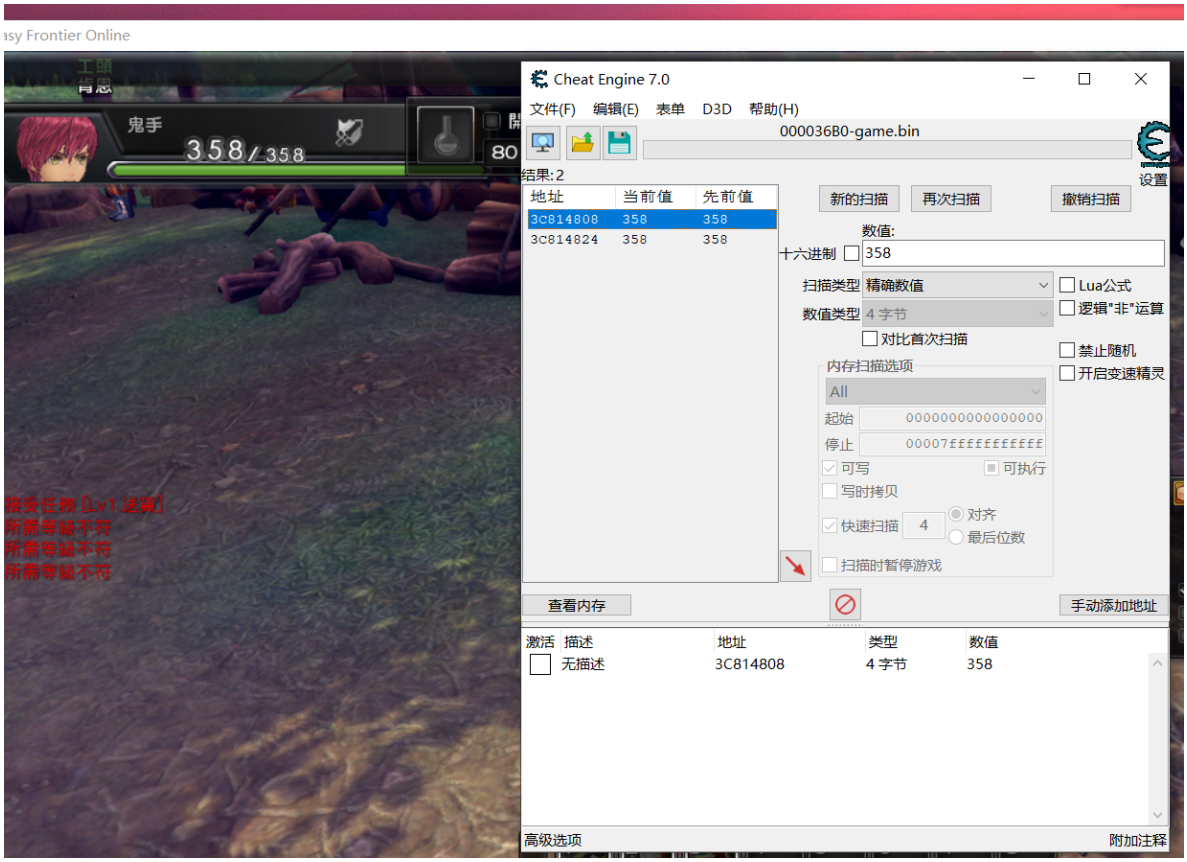
之前我们已经通过人物的血量找到了人物的属性数组，接下来学习一下链表。

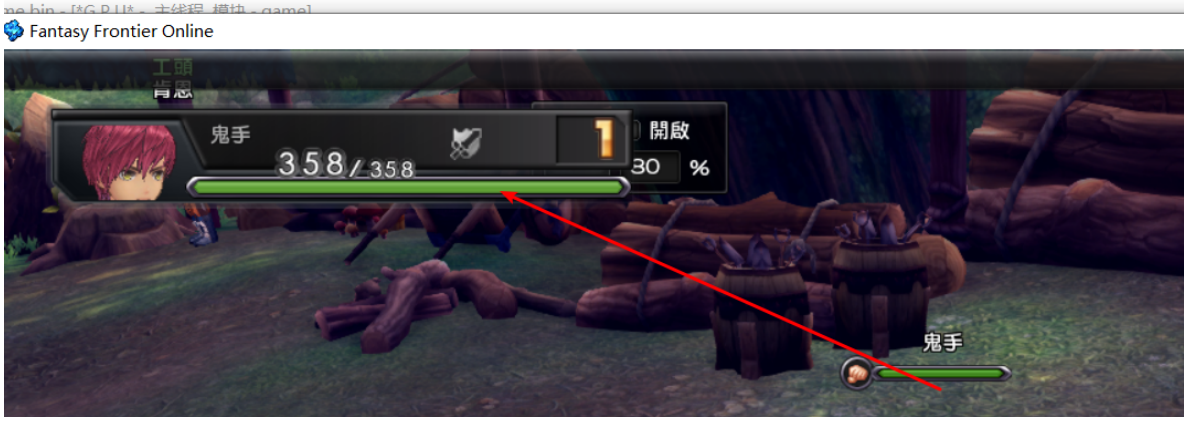
逆向周围对象链表

这一次要找的目标是人物周围的对象链表，包括人物周围的NPC和怪物等等。找这个数据结构的突破口有很多，可以通过人物本身，也可以通过NPC和怪物。

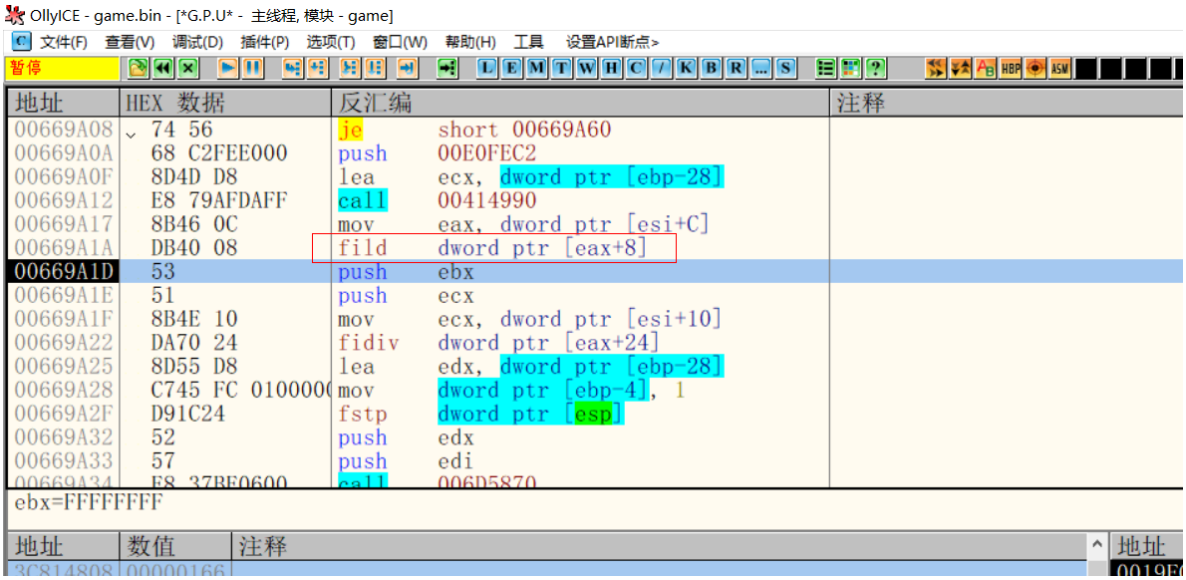
这里还是以人物血量为突破口，之前我们已经用人物血量去找到了属性的数组，但是同一个突破口通过不同的访问代码可以逆向分析出不同的数据。



和之前的方式一样，用CE找到当前的人物血量，然后在这个位置下硬件访问断点



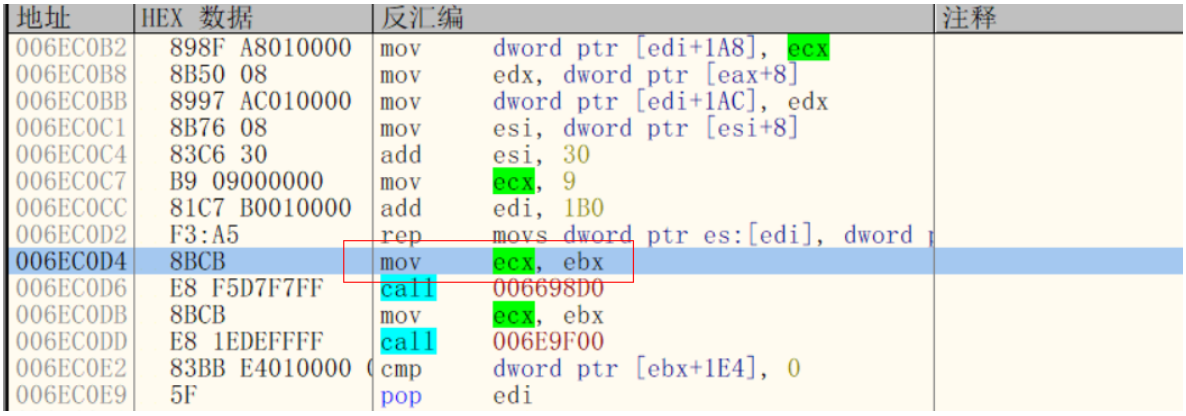
点击当前的人物血量



断点断下，[eax+8]是人物的血量，追eax的数据来源，eax来自[esi+0xC]，继续往上追esi

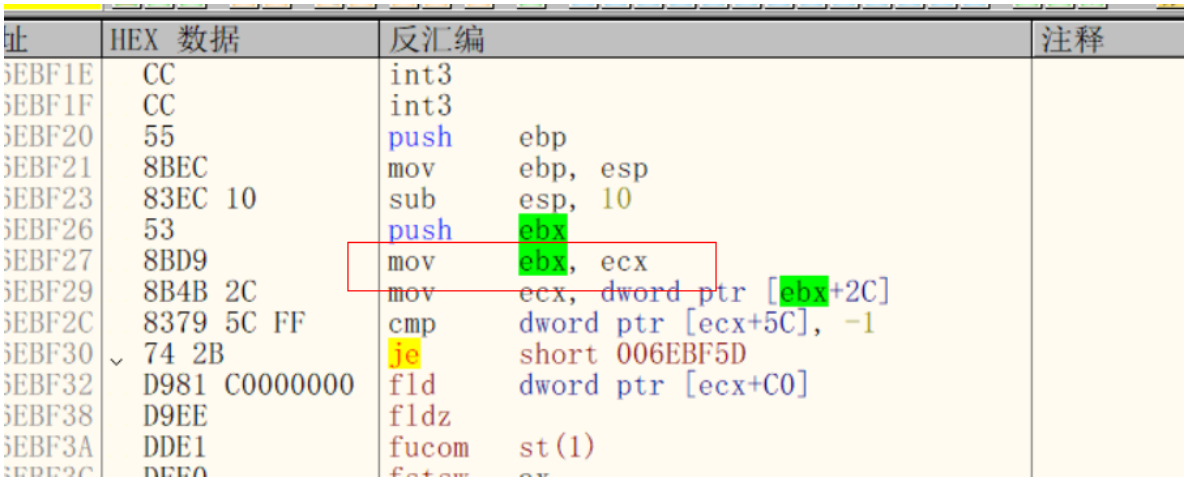
```
血量=[esi+0xC]+8
```

esi来自于ecx，返回上层找ecx



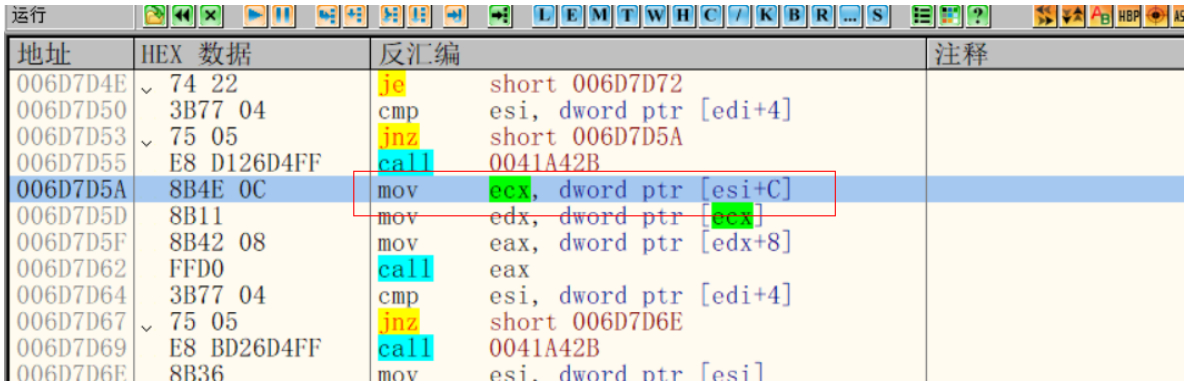
ecx又来源于ebx

```
血量=[ebx+0xC]+8
```



址	HEX 数据	反汇编	注释
5EBF1E	CC	int3	
5EBF1F	CC	int3	
5EBF20	55	push ebp	
5EBF21	8BEC	mov ebp, esp	
5EBF23	83EC 10	sub esp, 10	
5EBF26	53	push ebx	
5EBF27	8BD9	mov ebx, ecx	
5EBF29	8B4B 2C	mov ecx, dword ptr [ebx+2C]	
5EBF2C	8379 5C FF	cmp dword ptr [ecx+5C], -1	
5EBF30	74 2B	je short 006EBF5D	
5EBF32	D981 C0000000	fld dword ptr [ecx+C0]	
5EBF38	D9EE	fldz	
5EBF3A	DDE1	fucom st(1)	

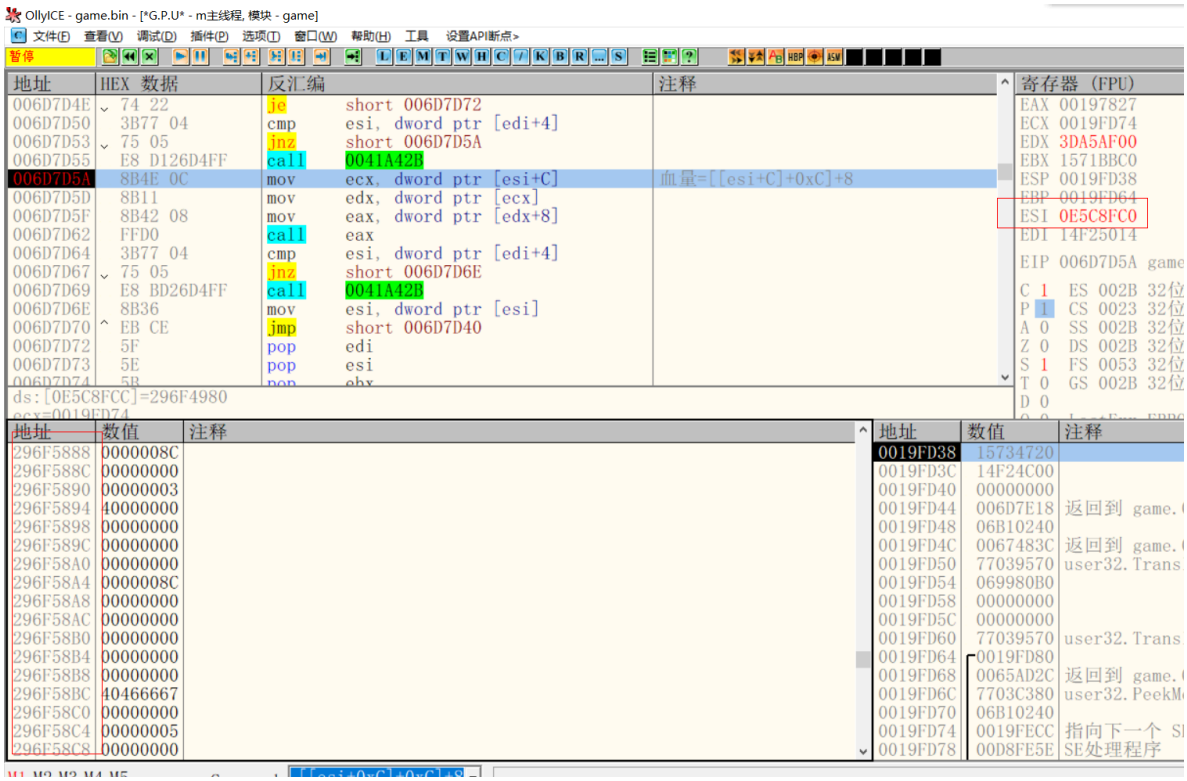
ebx来源ecx



地址	HEX 数据	反汇编	注释
006D7D4E	74 22	je short 006D7D72	
006D7D50	3B77 04	cmp esi, dword ptr [edi+4]	
006D7D53	75 05	jnz short 006D7D5A	
006D7D55	E8 D126D4FF	call 0041A42B	
006D7D5A	8B4E 0C	mov ecx, dword ptr [esi+C]	
006D7D5D	8B11	mov edx, dword ptr [ecx]	
006D7D5F	8B42 08	mov eax, dword ptr [edx+8]	
006D7D62	FFD0	call eax	
006D7D64	3B77 04	cmp esi, dword ptr [edi+4]	
006D7D67	75 05	jnz short 006D7D6E	
006D7D69	E8 BD26D4FF	call 0041A42B	
006D7D6E	8B36	mov esi, dword ptr [esi]	

ecx来自[esi+C]

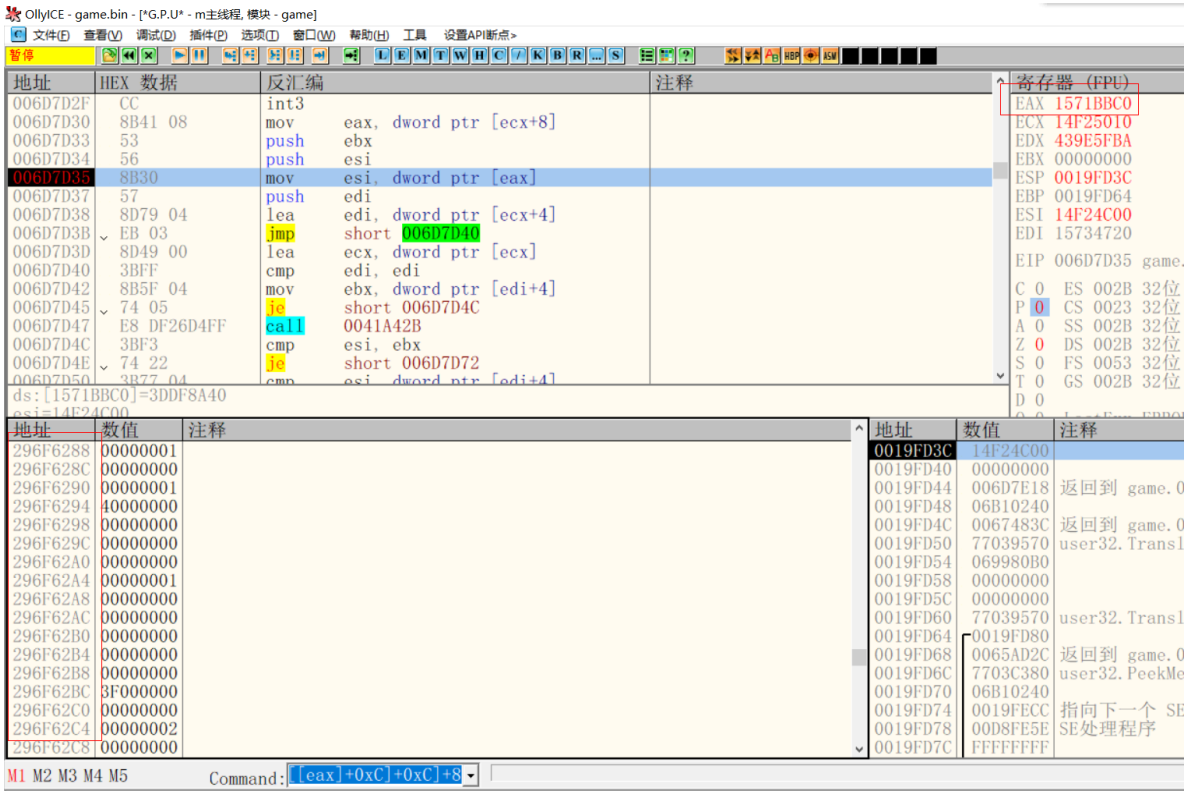
```
血量=[[esi+C]+0xC]+8
```



在这里查看一下血量的值，每一次断点断下，存储血量的地址和esi的值都会不断发生变化。

这个地方的地址一直发生变化说明是已经来到了一个数据结构，如果这个地方单纯的存放的是人物血量的话，就是基址+偏移的方式，不会发生改变。

继续往上追esi



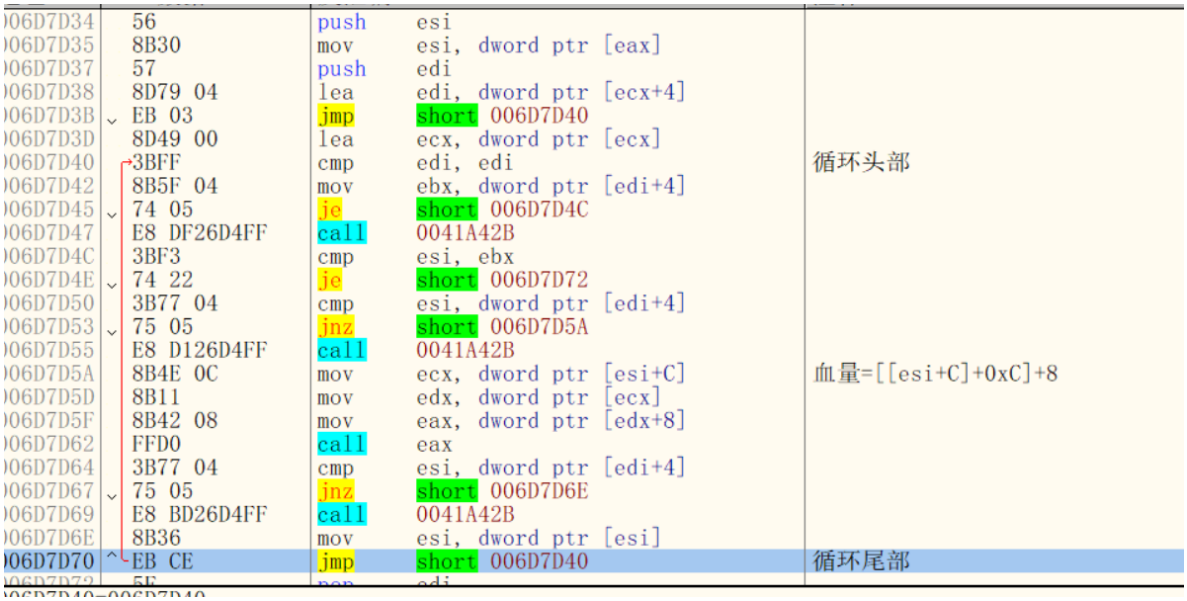
esi来自[eax]

```
血量=[[[eax]+C]+0xC]+8
```

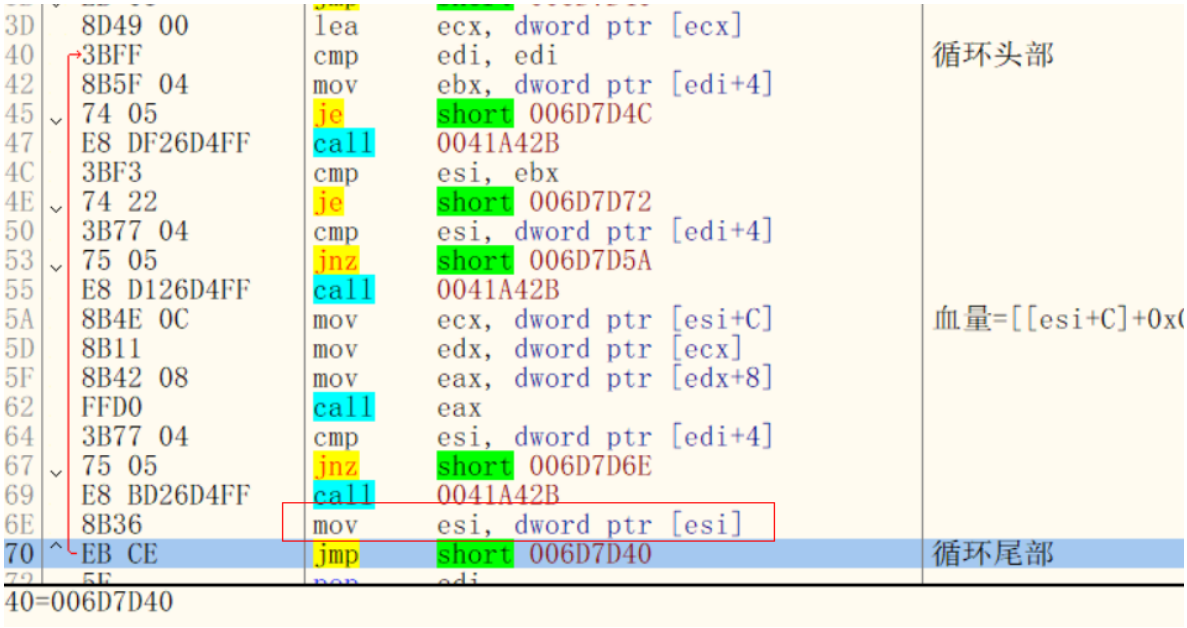
eax这个时候只有两个值在发生变化，而且偏移表达式的地址，不管断下多少次都没有发生改变。

从这个地方的变化说明esi的值不来源于eax，也就是说当前的代码不是顺序执行的，有可能是进到了循环里面。

那么就有必要分析一下当前的这段代码



这里有一个向上的跳转是当前的循环尾部，jmp跳转的目标地址是循环头部，中间的代码相当于是循环体了。

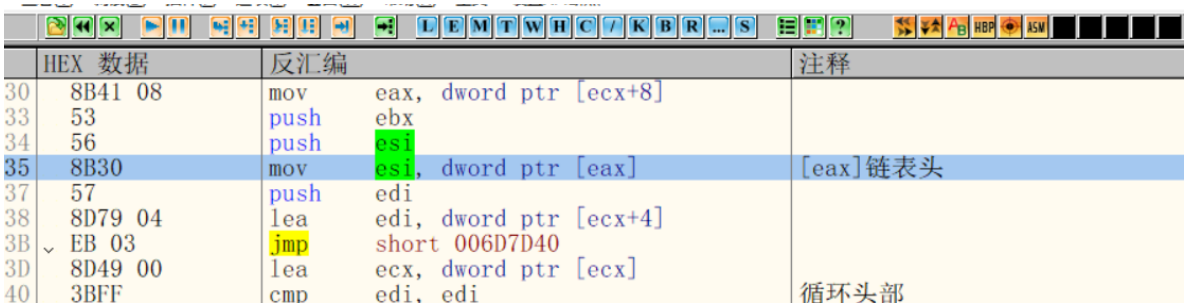


esi不来自循环体外的eax，也就说明是来自循环体内的[esi]，esi再往上找还是没有发现来源，说明当前的这一句代码一直在循环取值。

如果和数据结构联想到一起的话，很明显这个地方是一个单向链表，C++代码如下

```
class Node
{
public:
    Node<T> *next;
    T data;
```

上面这段代码就是通过对Node节点的循环遍历，来找到想要的目标节点。跳出循环体外



esi来源于[eax]，eax就是当前的链表头，这里可以对当前的链表进行取值

```
第一个对象血量:[[[[eax]]+C]+0xC]+8
第二个对象血量:[[[[[eax]]]+C]+0xC]+8
第三个对象血量:[[[[[[eax]]]]+C]+0xC]+8
```

我们已经找到了链表的数据结构，这个时候就没有必要继续往上追了，意义不大。剩余的对象属性各位可以自己去分析

总结

对于链表的逆向来说，需要时刻关注追数据过程中每一步的数据结构变化，特别是变化比较明显的那一种。识别链表的标识在于是否有汇编代码在遍历当前的链表不断进行取值，例如：

```
mov eax,[eax];
```

最后，附上Github地址，里面有游戏下载链接和相关工具，需要请自取：
https://github.com/TonyChen56/GameReverseNote