1. 数据类型和容器

立即数：一个常数，也就是一个普通的数字，不能作为容器。

寄存器：CPU中的临时储存单元，80X86的32位平台平台上CPU寄存器仅有14个，寄存器速度比内存快得多，存储的可以是从0x00000000到0xFFFFFFFF之间的一个整数，一个int类型的存储大小，占32个bit，即一个DoubleWord的长度。

内存：操作系统分配给程序的虚拟内存，并不是真实内存。80X86的32位平台每个进程都有4G的虚拟内存，也可以说是4GB寻址能力，地址从低位到高位分别是0x00000000到0xFFFFFFFF，占32个bit，即一个DoubleWord的长度，总共4GB。

1. 数据储存方式：

在内存中，一个数比如0x11223344会被分个储存，每个被分成的部分是八个bit，也就是一个byte。所以0x11223344会被分成0x11,0x22,0x33,0x44四个部分，数字的低位存在内存的高位，数据的高位存在内存的低位，这种模式被称为“小端模式”。

从内存条上看，数据是这样的：

Address: 0x00000000 | 0x00000001 | 0x00000002 | 0x00000003

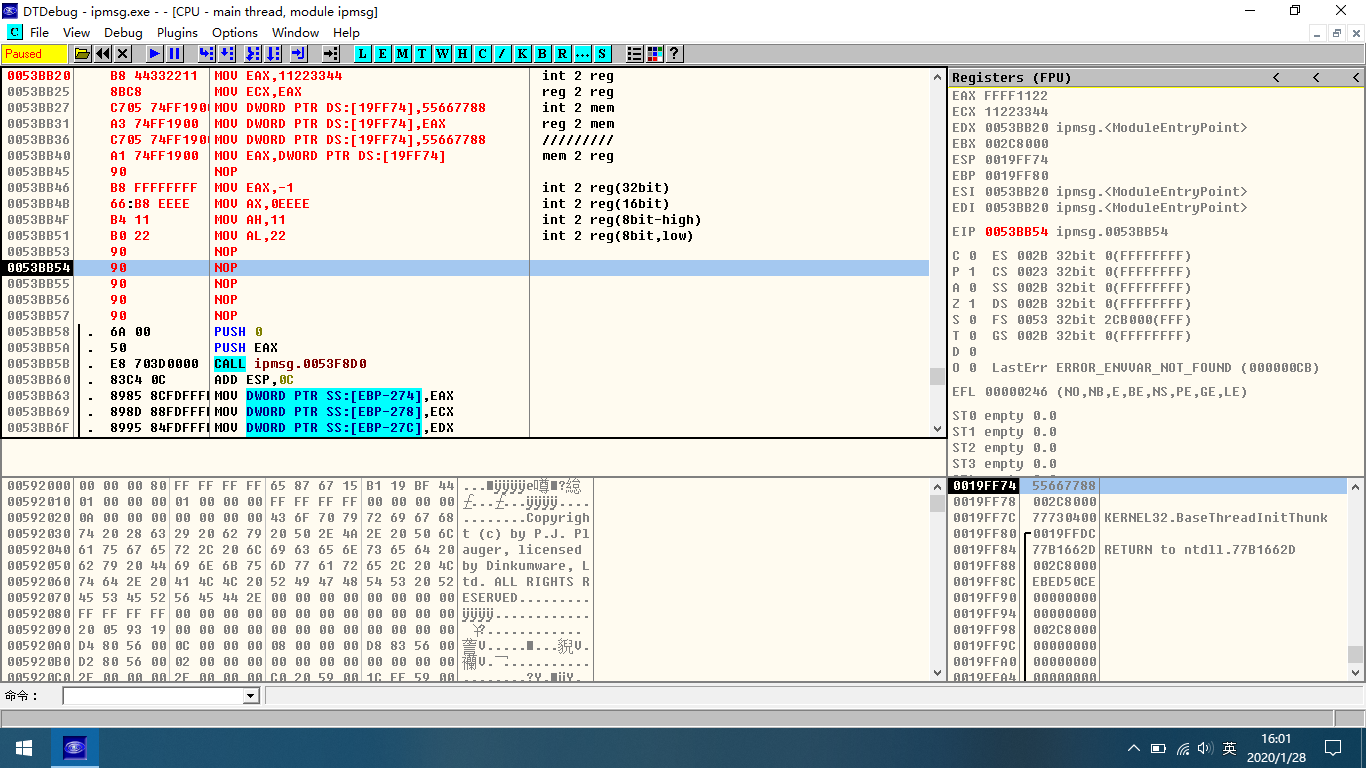
..Value: 0x44.......| 0x33.......| 0x22.......| 0x11......

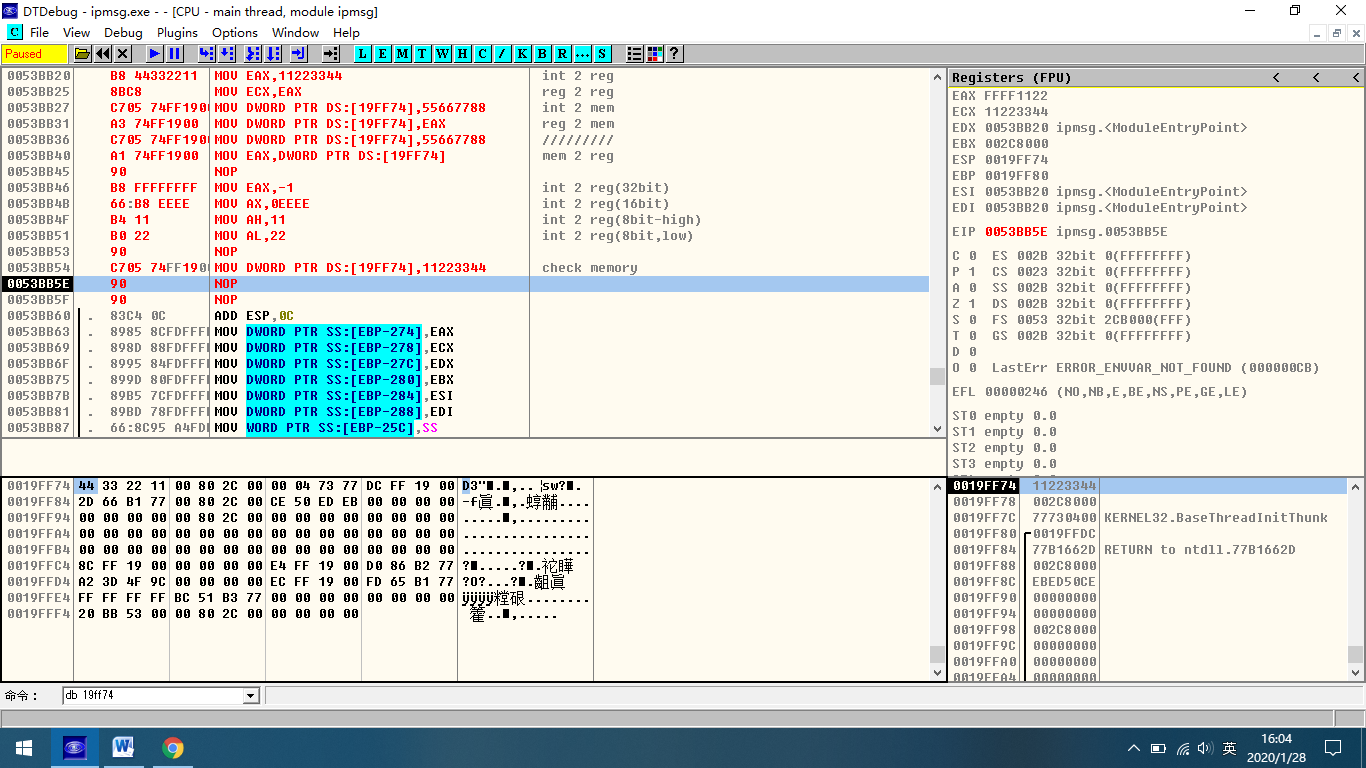
三、80X86汇编指令

*注意：只要没有特别提到，下列这些指令的参数都可以是三十二位立即数、三十二位寄存器、三十二位内存、十六位立即数、十六位寄存器、十六位内存、八位立即数、八位寄存器、八位内存。*

1. MOV a,b

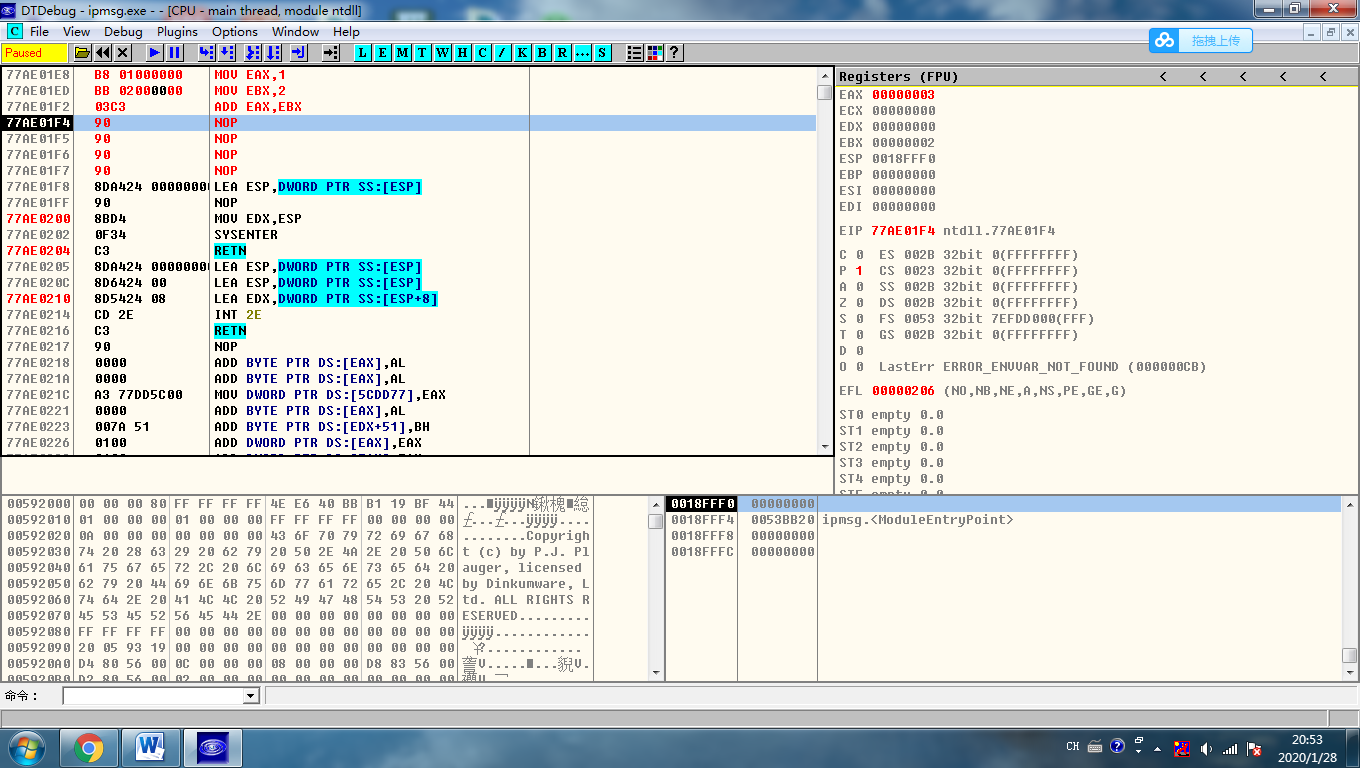
把b写入到a



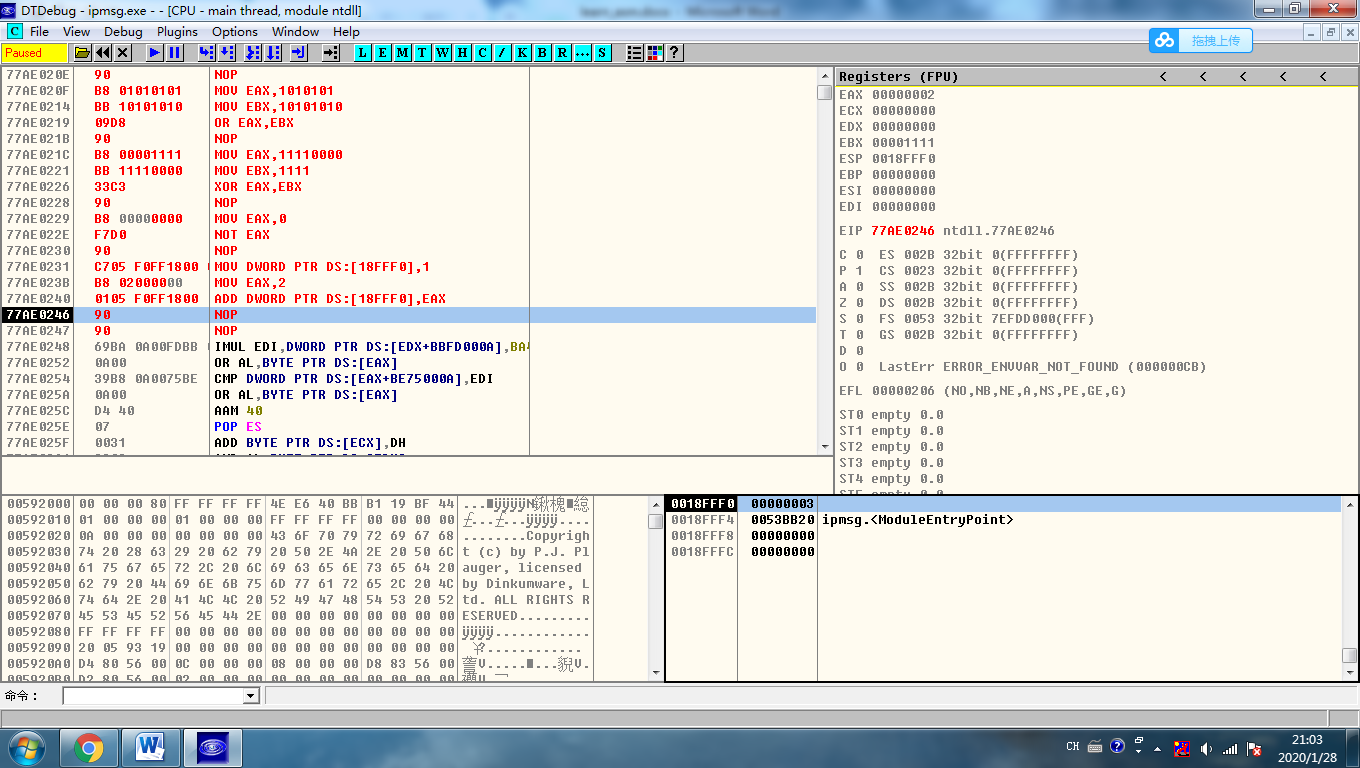


1. ADD a,b

a加b，结果存到a

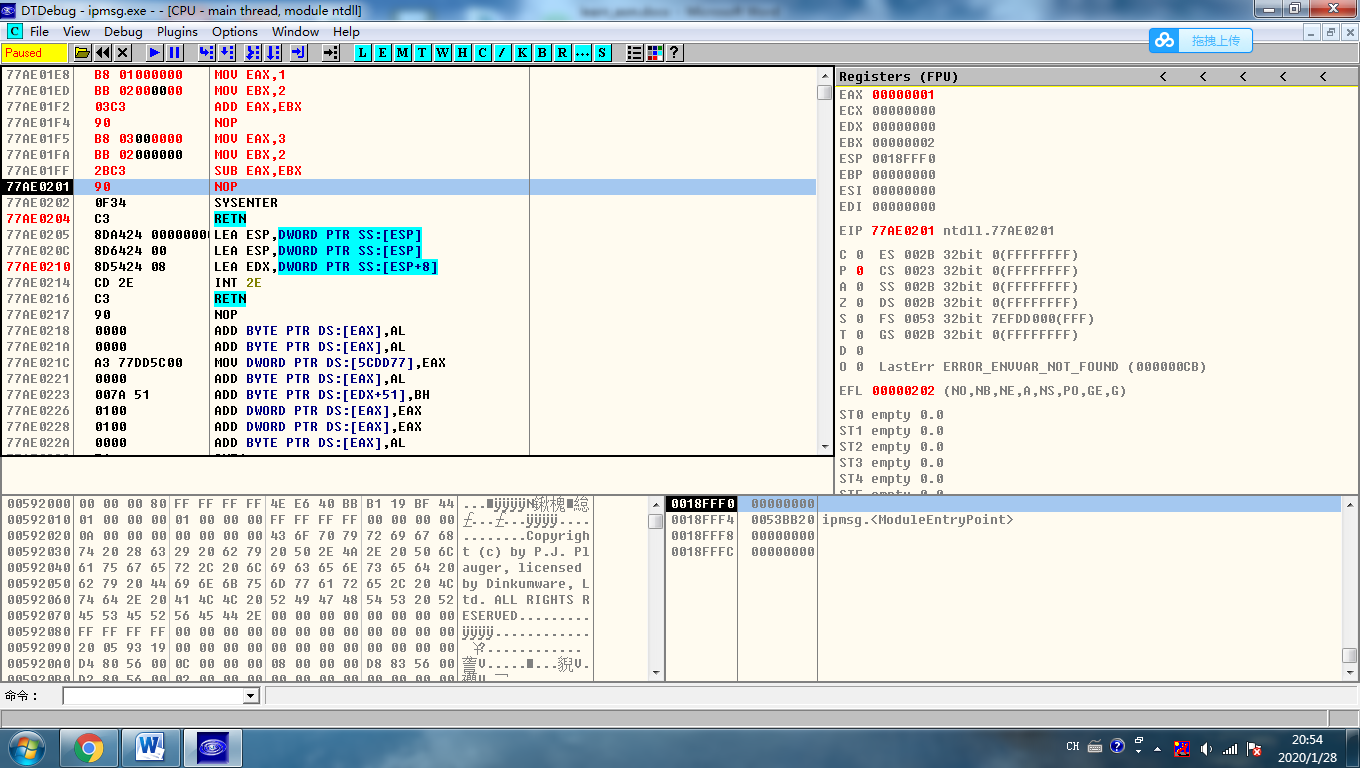


PS:使用内存的ADD。之后的指令如果没有特别说明，也可以有一个参数使用内存地址，但不能两个参数都用内存地址。



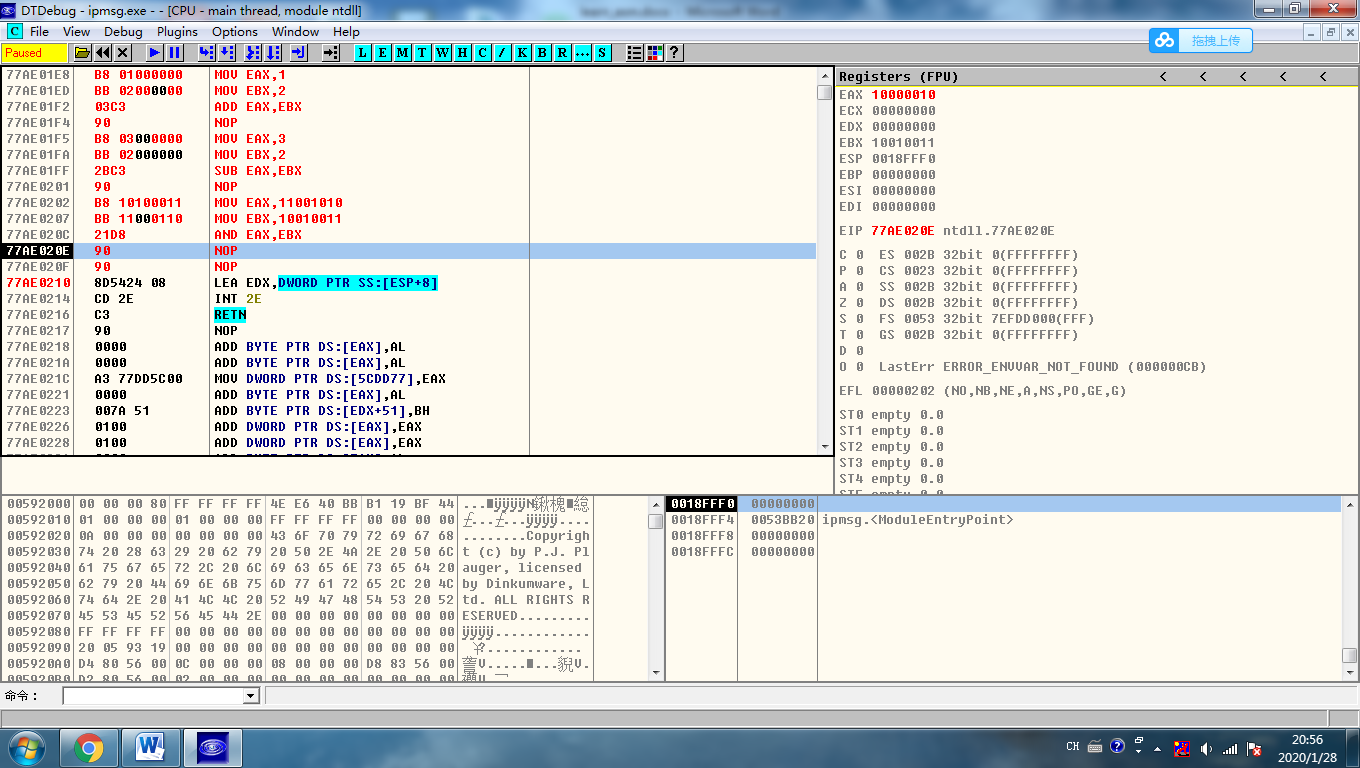
1. SUB a,b （SUB即Subtract的简写）

a减b，结果存到a



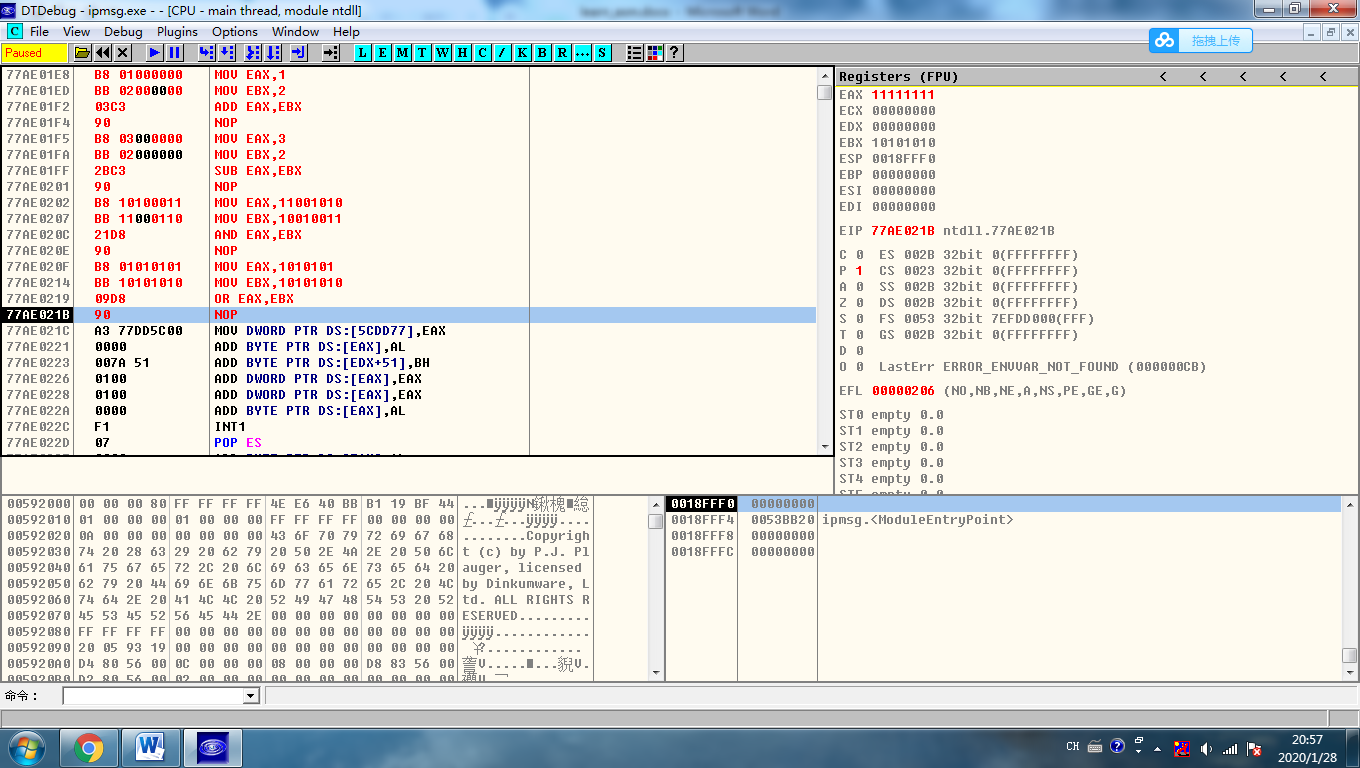
1. AND a,b

a与b，结果存到a



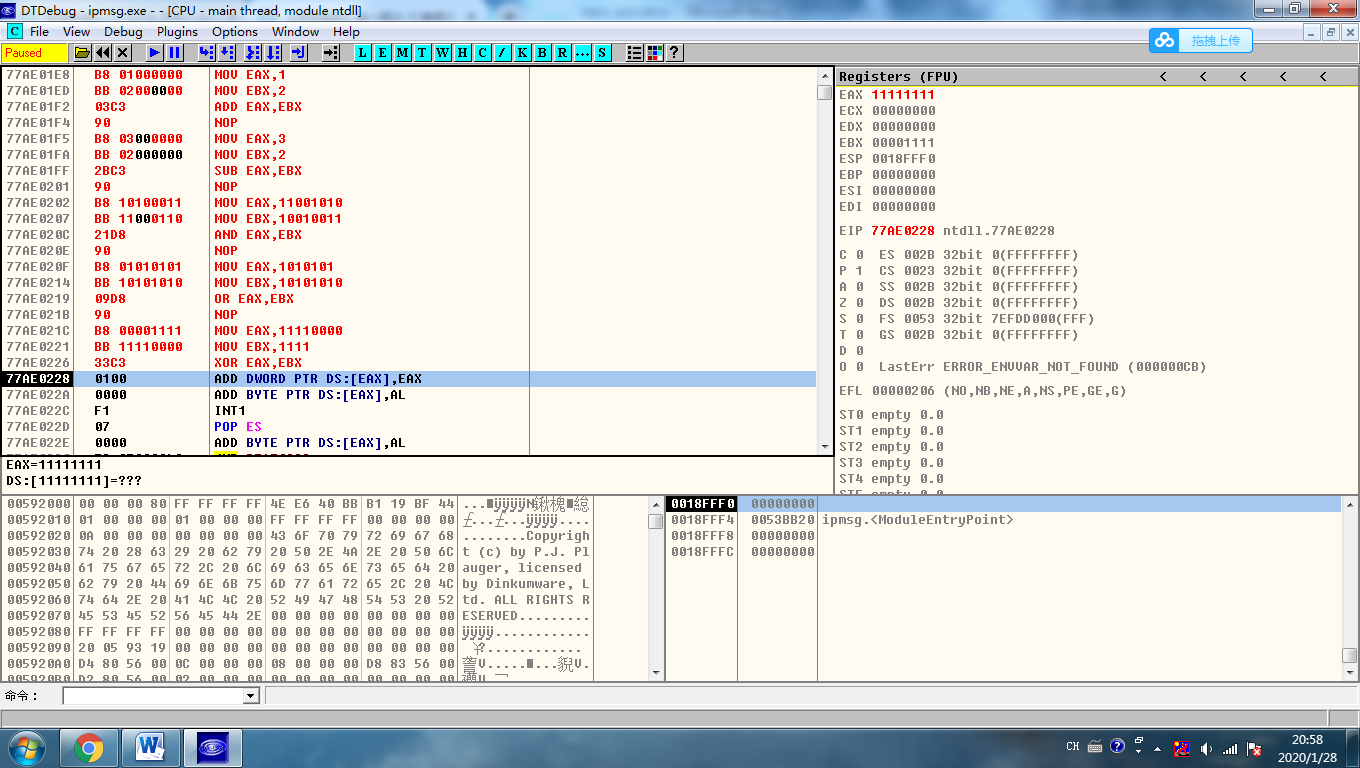
1. OR a,b

a或b，结果存到a



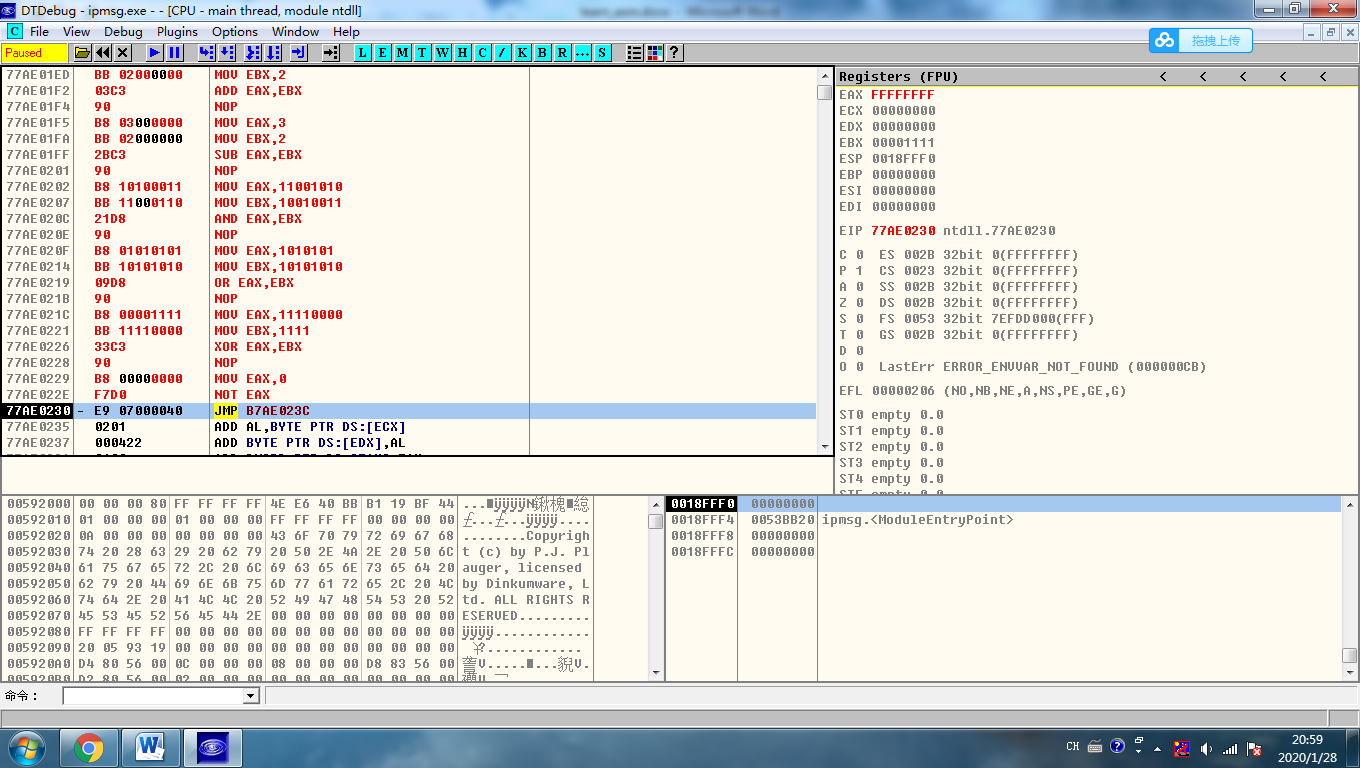
1. XOR a,b

a异或b，结果存到a



1. NOT a

非a，结果存到a



1. MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI] 全等于 MOVSB

MOVS WORD PTR ES:[EDI],WORD PTR DS:[ESI] 全等于 MOVSW

MOVS DWORD PTR ES:[EDI],DWORD PTR DS:[ESI] 全等于 MOVSD

MOVS命令将内存的数据复制到内存的数据。

这个命令很特殊，只有上面这几种写法，也就是说ESI和EDI是不能写成内存地址或寄存器的，只能够写成这两个寄存器。

每执行一句这个命令，ESI和EDI的值都会**增加或减少**一个**BYTE/WORD/DWORD**的长度，也就是**1/2/4**这三个数。

决定到底长度是是**BYTE**、**WORD**还是**DWORD**的因素是MOVS指令后面跟着的是BYTE、WORD还是DWORD；

而决定是增加还是减少的因素，则是EFL（EFlag）寄存器从右往左数的第10个位（也叫DF）的值。如果EFL的第十个位（DF）是0，那就增加；如果第十个位是1，那就减少。

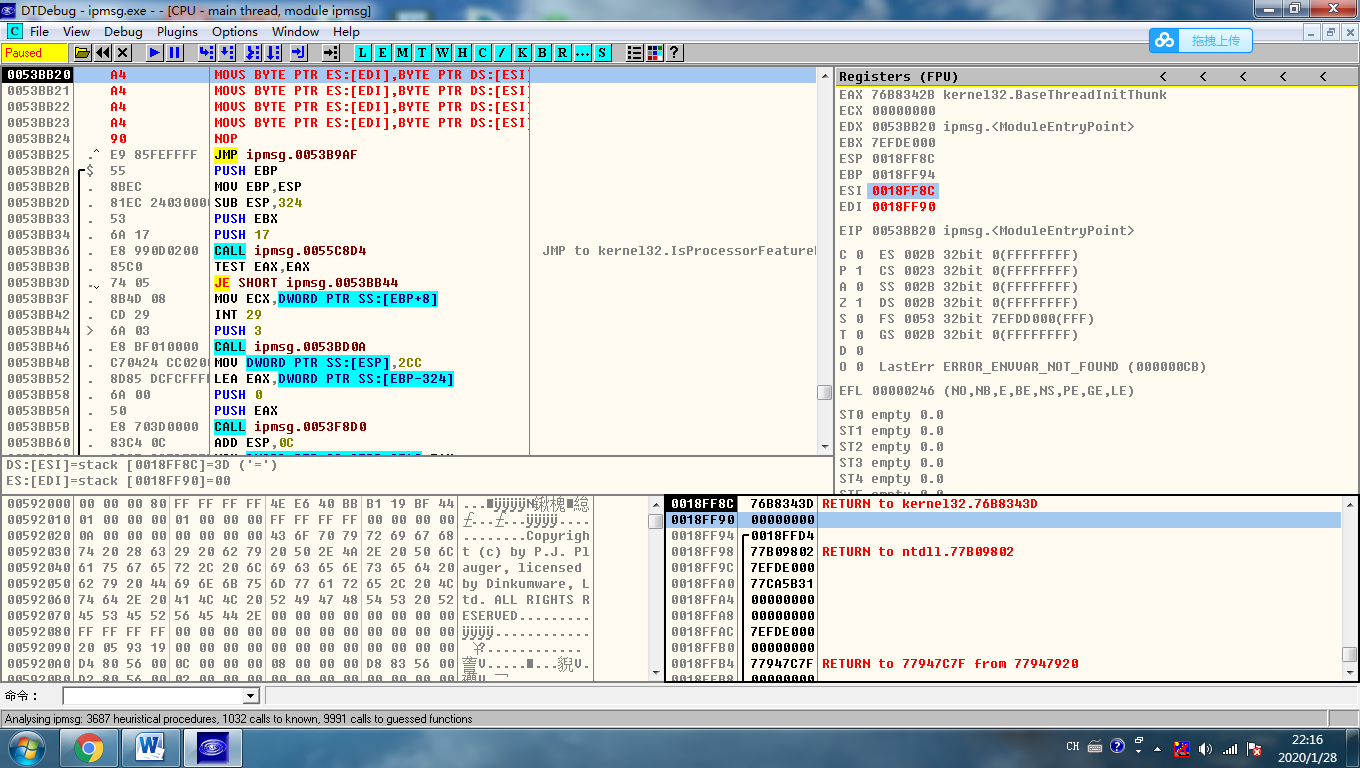
注意！第十个位是要从0开始数的，所以他就是普通人说的第十一个位！！！

Address: 31 30 29 28 …… 11 10 9 8 …… 6 5 4 3 2 1 0

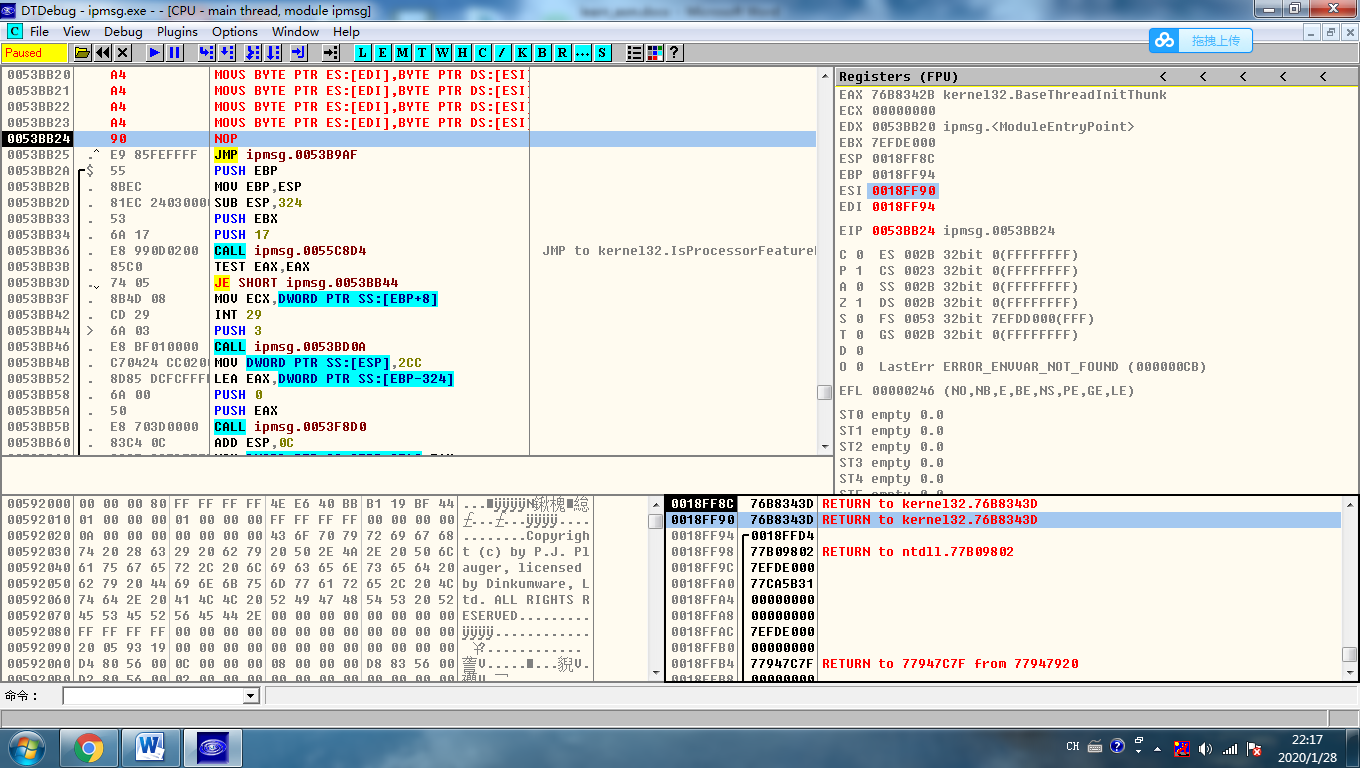
..Value: .0 .0 .0 .0 …… .0 .0 0 0 …… 0 0 0 0 0 0 0

关于MOVSB这样的简写，编译器会自动把他编译成MOVS BYTE PTR ES:[EDI],BYTE PTR DS:[ESI]这句话，他们在编译后的二进制文件中是一样的。

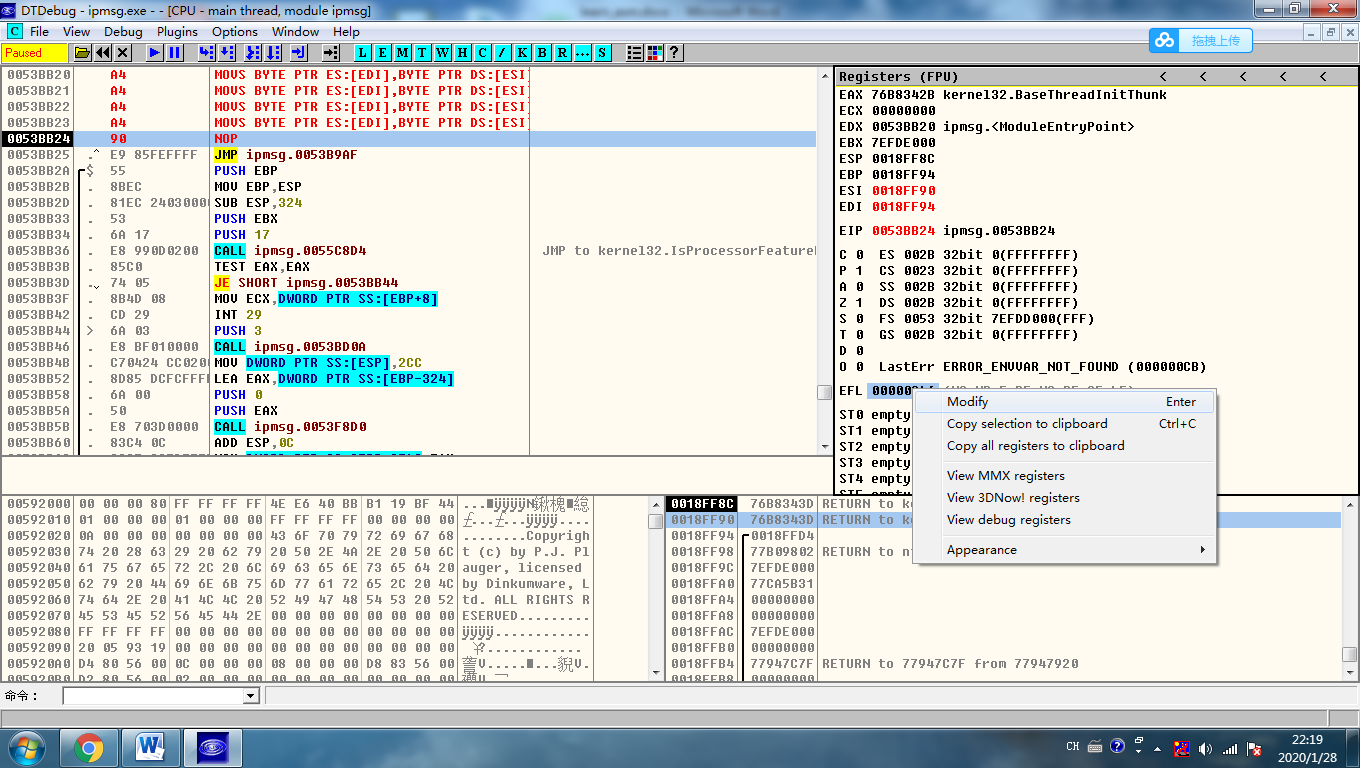
(1).将esi和edi的值设好，输入命令后单步调试



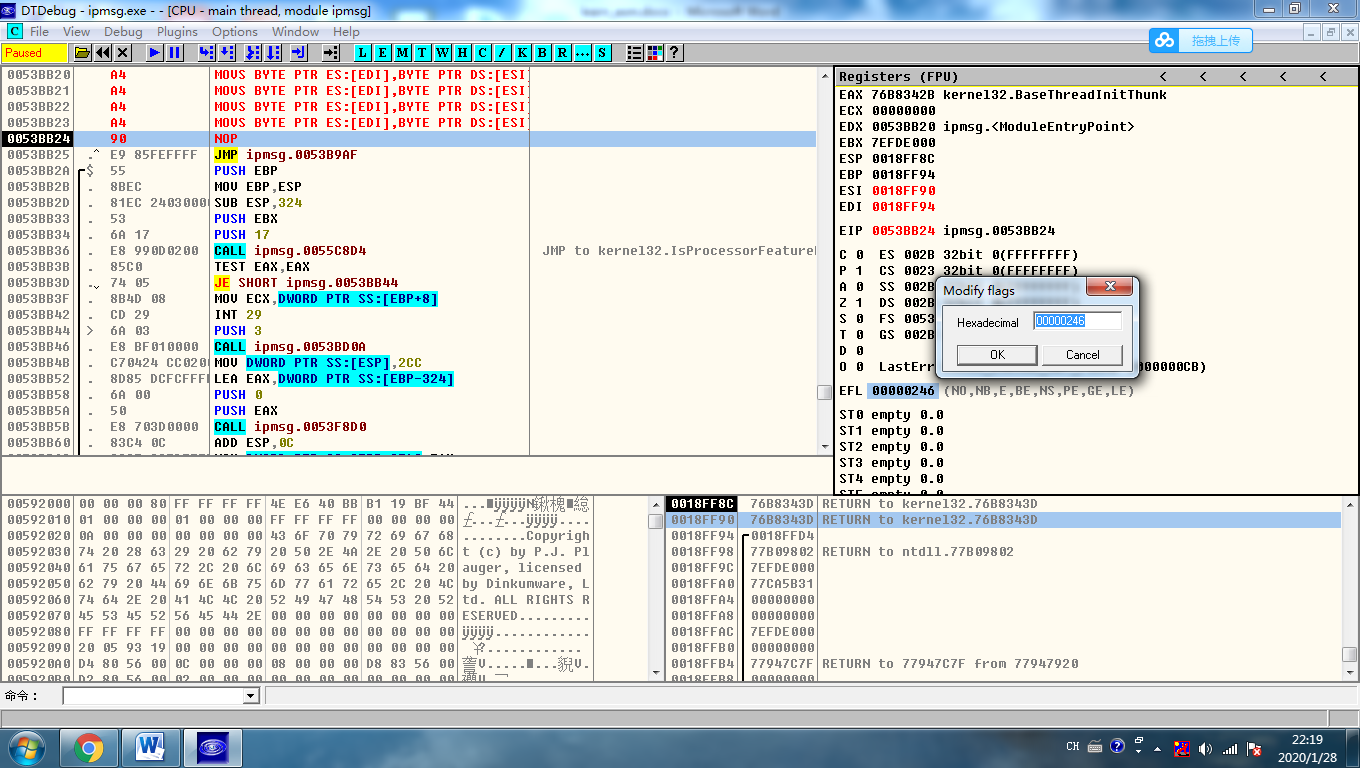
(2).运行完四条命令后，可以看到内存已经成功被复制，ESI和EDI也已经自动增加了四个byte的长度



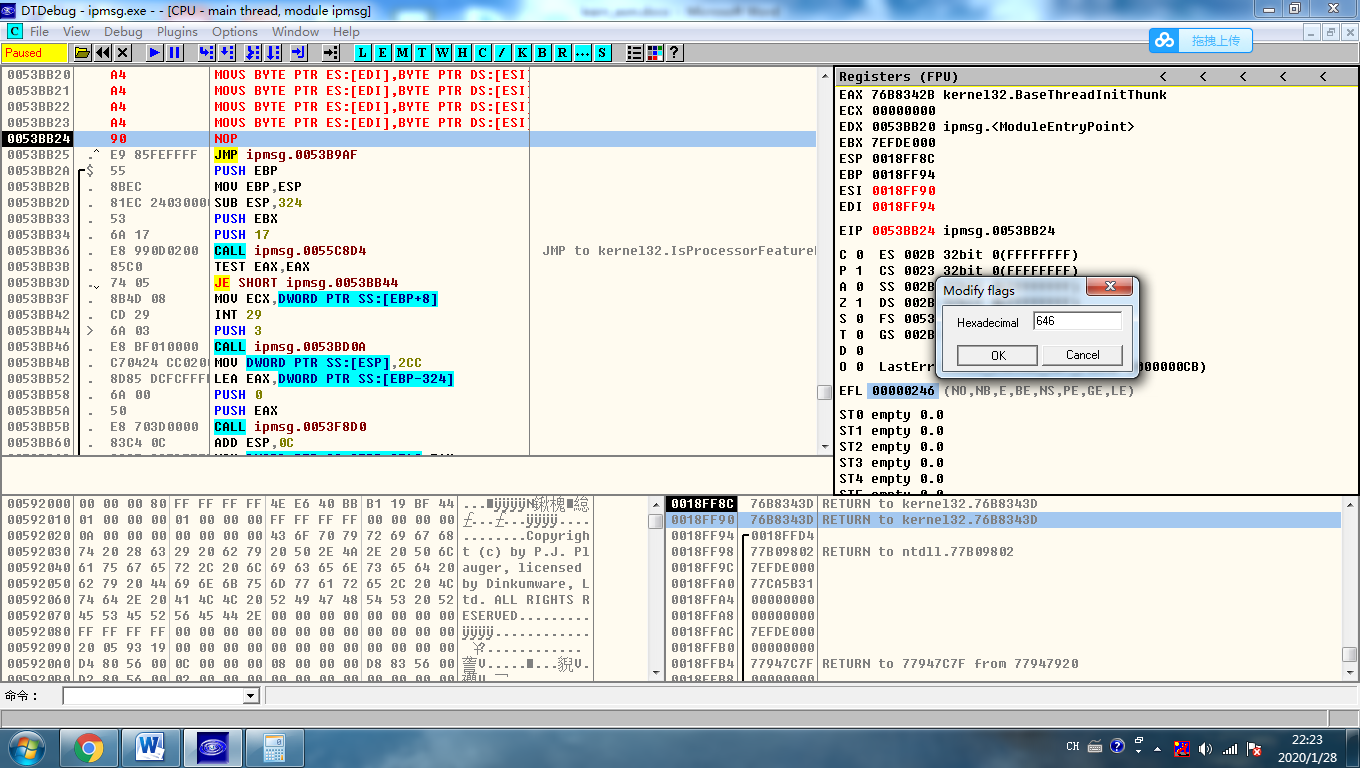
(3).对EF进行Modify



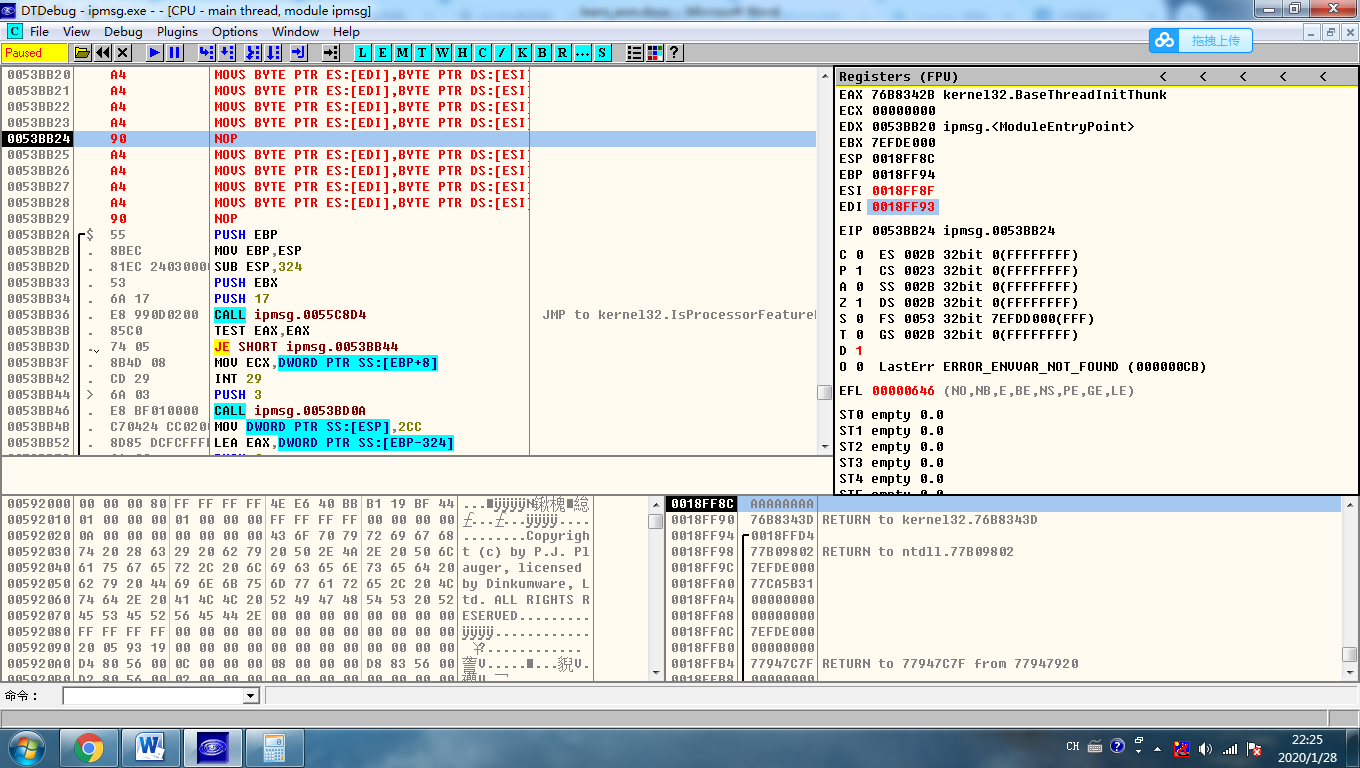
(4).可以看到当前的值是0x00000246，转换成二进制是0000 0000 0000 0000 0000 0010 0100 0110



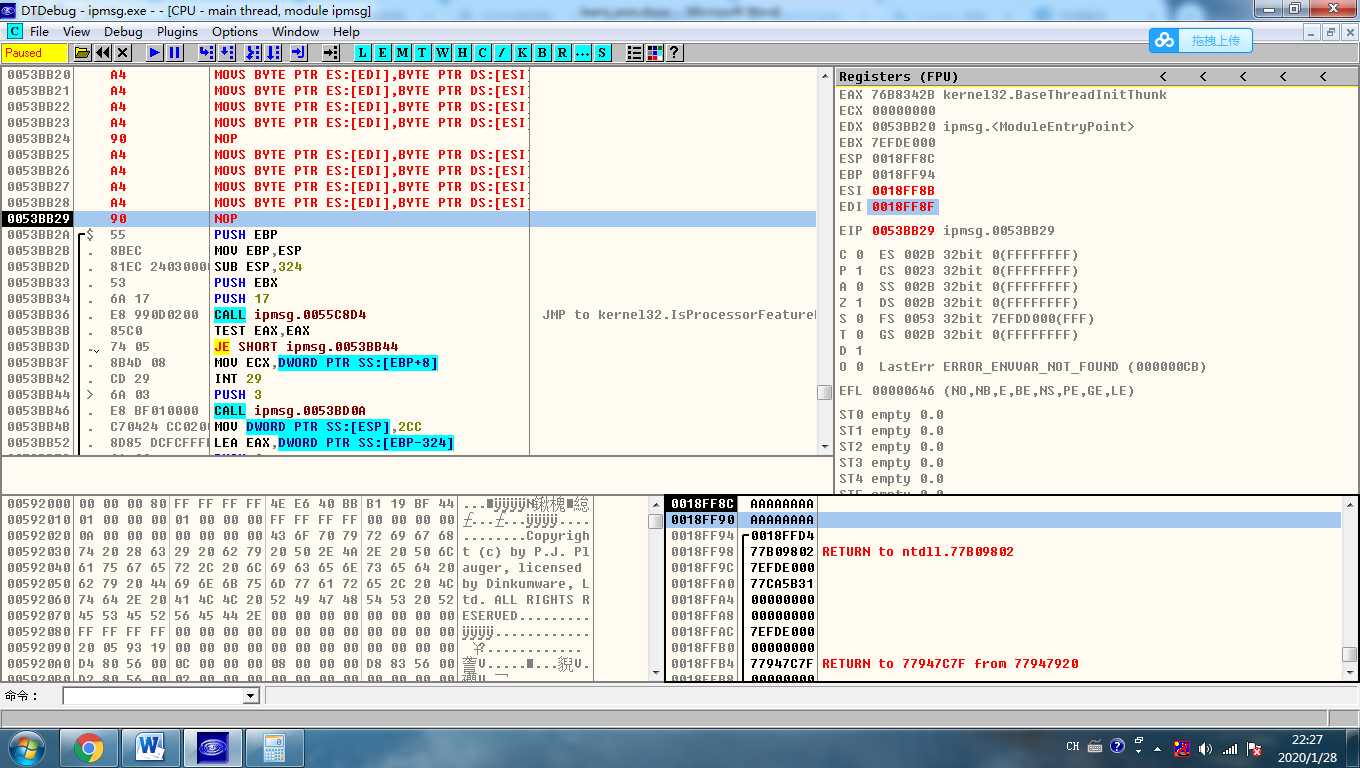
(5). 0000 0000 0000 0000 0000 0010 0100 0110从右往左数十个的位是0，把它改成1得到0000 0000 0000 0000 0000 0110 0100 0110，转换成十六进制是646，把它填进去：



(6).在把ESI和EDI各减1，这是他们指向刚才那块内存的最后一个byte的位置，把ESI所在内存的值设为0xAAAAAAAA，观察MOVS复制的效果



(7).MOVS把内存倒着复制过去了，ESI和EDI也都减了4个byte的长度，搞定！



1. XX指令