

## 第六章-比较和条件跳转

通常情况下,比较指令有两个操作数,并根据比较的结果来决定程序是否跳转到后面的分支中。

我们知道,当注册程序要求我们输入序列号的时候,这个时候,程序会执行一条或者多条比较指令,根据比较的结果来判断你输入的序列号是否正确。

接下来,我们将详细介绍比较和跳转指令。

我们知道,某些指令的指令会影响到标志位,最常见的就是零标志位 Z。

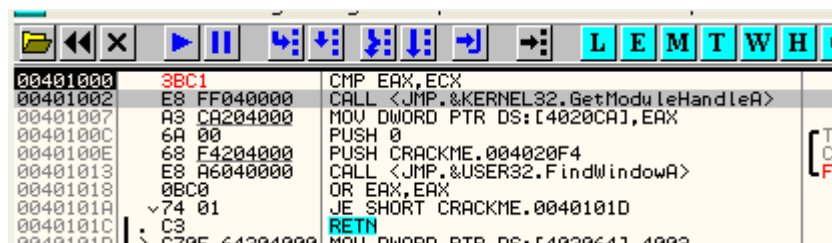
### CMP

该指令是比较两个操作数,实际上,它相当于 SUB 指令,但是相减的结构并不保存到第一个操作数中。只是根据相减的结果来改变零标志位的,当两个操作数相等的时候,零标志位置 1。

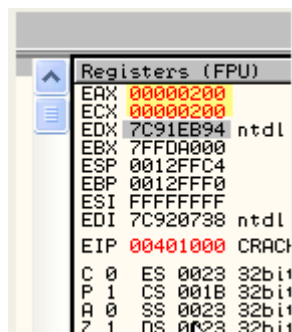
看看下面的例子:

CMP EAX, ECX

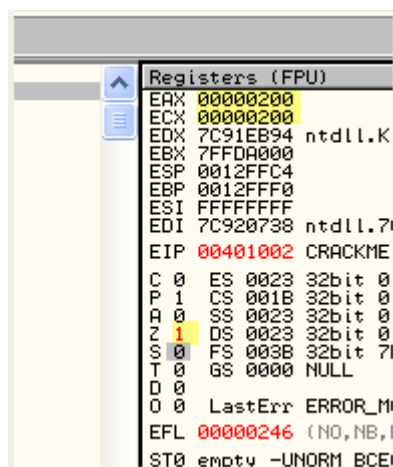
EAX 与 ECX 相减,它们本身的值并不改变,只是根据它们相减的结果来决定零标志位 Z 是否置 1。在 OD 中来看一个例子。



写入 CMP EAX,ECX 指令并修改 EAX 和 ECX 的值,让它们相等。



按下 F7 键,可以看到 EAX 和 ECX 的值并没有改变,但是相减的结果是零标志位 Z 被置为了 1,我们在 OD 来看一个例子。



实际上,我们并不关心相减的确切结果,我们只关心 EAX 和 ECX 是否相等。

虽然我们还没有介绍条件跳转,这里简单提一下,有两种可能性:根据标志的值来决定跳转还是不跳转。最简单的例子就是配合 JZ 指令,如果 Z 标志被置为 1,就跳转,否则,就不跳转。

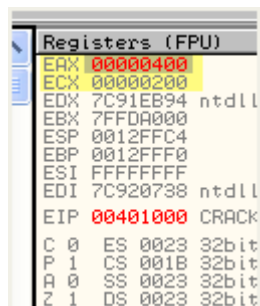
如果有两个序列号做比较,例如 EAX 存放的是你输入的序列号,ECX 存放的是正确的序列号,该程序使用 CMP 指令来比较,如果它们两个是相等的,那么零标志位 Z 就置为 1,后面 JZ 指令就会跳转到注册成功的部分。如果 EAX 不等于 ECX 的话,那么零标志位就会置 0,那么就不会跳转到注册成功的部分。

让我们来看看有关条件跳转的一个更具体的例子。

符号标志位 S 是比较第一个操作数是否大于第二个操作数。

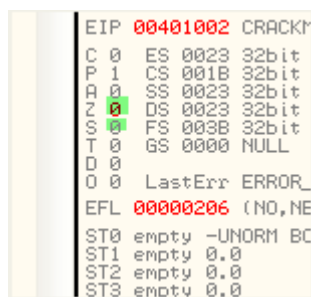
让我们来看下面的例子:

重新写入 CMP EAX,ECX 指令,但是现在要求是 EAX 的值要大于 ECX 的值。



Registers (FPU)	
EAX	00000400
ECX	00000200
EDX	7C91EB94 ntdll
EBX	7FFDA000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401000 CRACK
C 0	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit
Z 1	DS 0023 32bit

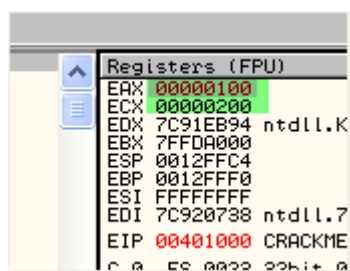
按下 F7 键:



Registers (FPU)	
EIP	00401002 CRACK
C 0	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit
S 0	FS 003B 32bit
T 0	GS 0000 NULL
D 0	
O 0	LastErr ERROR_
EFL	00000206 (NO,NE
ST0	empty -UNORM BC
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0

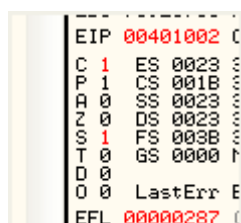
可以看到,零标志位 Z 是 0,所以我们知道,这两个值并不相等,并且符号标志位 S 也等于 0,我们就可以得知 EAX-ECX 结果是正的。也就说明 EAX 大于 ECX。

依然是 CMP EAX,ECX 指令,但这次是 EAX 小于 ECX。



Registers (FPU)	
EAX	00000100
ECX	00000200
EDX	7C91EB94 ntdll.K
EBX	7FFDA000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll.7
EIP	00401000 CRACKME
C 0	ES 0023 32bit

然后按下 F7 键:

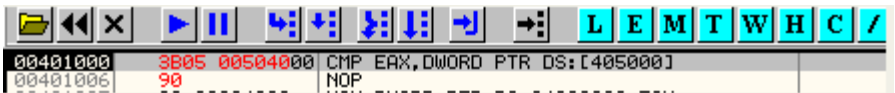


Registers (FPU)	
EIP	00401002 C
C 1	ES 0023 32bit
P 1	CS 001B 32bit
A 0	SS 0023 32bit
Z 0	DS 0023 32bit
S 1	FS 003B 32bit
T 0	GS 0000 32bit
D 0	
O 0	LastErr E
EFL	00000287 C

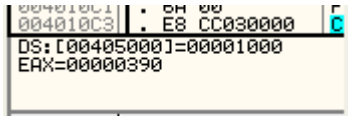
这里我们可以看到,EAX 减去 ECX 的结果是负的,也就是说 ECX 大于 EAX。所以符号标志位 S 被置为了 1。

根据比较的不同结果来设置相应的标志位,来决定程序应该走哪个分支。此外,CMP 指令还允许寄存器与 BYTE,

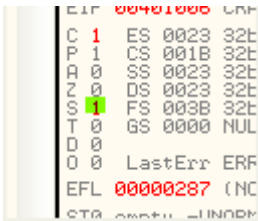
WORD,DWORD 类型的内存单元的值做比较。



这里,该指令是比较的 EAX 和 405000 内存单元的值,跟之前一样,我们在 OD 来看看解释窗口。



这个例子里面,EAX 是小于 405000 内存单元中的值的,405000 内存单元中的值是 1000,相减的结果为负的,所以符号标志位 S 会被置 1。



类似的例子还有

CMP AX,WORD PTR DS:[405000]

和

CMP AL,BYTE PTR DS:[405000]

这两种情况下分别是与 BYTE,WORD 类型的内存单元的值做比较。

#### TEST(逻辑比较)

该指令在一定程度上和 CMP 指令时类似的,两个数值进行与操作,结果不保存,但是会改变相应标志位(比如说,SF,ZF,PF 标志位),程序可以根据结果来决定是否跳转到相应的分支。

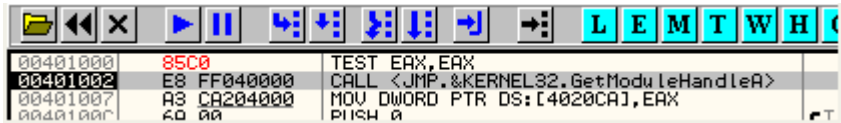
下面有几个例子:

#### TEST EAX,EAX

你会说,如果 EAX 与自己做比较呢? 用这个指令,可以确定 EAX 是否等于 0。

我们在 OD 中写入下面的指令:

#### TEST EAX,EAX



与操作的表如下:

1 and 1 = 1

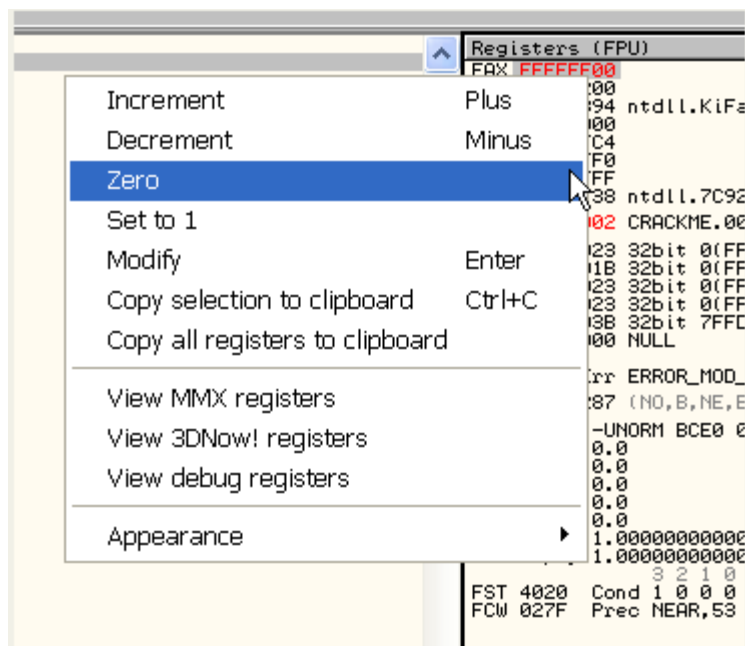
1 and 0 = 0

0 and 1 = 0

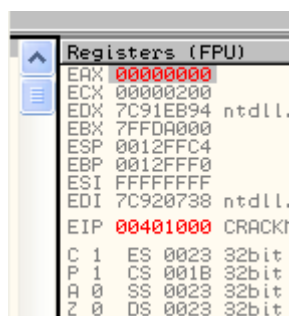
0 and 0 = 0

上表中结果为 0 的只有一种情况,只有当两个数都为 0 的情况(我们并不关心操作数的值是多少,因为我们是 EAX 与自身操作,它们永远是相等的),但是如果 EAX 的二进制某些位为 1 的话,那么运算的结果就不为零。

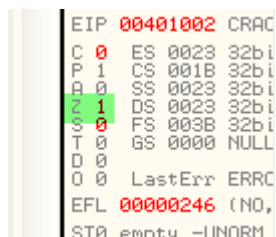
我们将 EAX 修改为 0。



OD 中在寄存器上单击鼠标右键选择零。

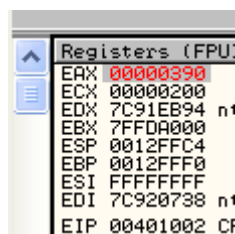


现在按下 F7 键。

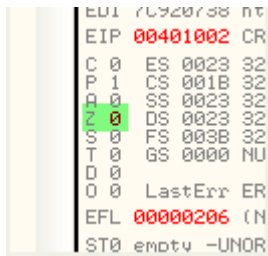


我们看到,零标志位被置 1 了,两个 0 做与操作,结果为 0,所以零标志位被置 1。

如果我们将 EAX 改为非零值,然后重复上面的操作呢。

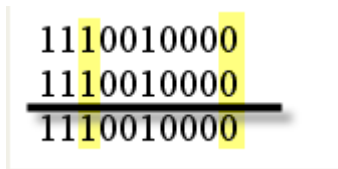


按下 F7 键。



零标志位没有被置 1 的话,就说明结果不等于 0。

如果你使用计算器的话,可以计算出 390 AND 390 结果依然是 390,二进制为 1110010000。



因为位与运算中,如果两个操作数都为 0,结果为 0,如果两个操作数都为 1,结果才为 1,那么在这种情况下,结果将依然是 390,零标志位被置 1。

上面是比较指令的主要部分,下面将介绍跳转指令部分。

## JUMPS

所有的跳转指令都会指向程序将会跳转到的地址。我们在下面的列表中可以看到各种不同类型的跳转指令。

**JMP** - 跳转

**JE, JZ** - 结果为零则跳转

**JNE, JNZ** - 结果不为零则跳转

**JS** - 结果为负则跳转

**JNS** - 结果不为负则跳转

**JP, JPE** - 结果中 1 的个数为偶数则跳转

**JNP, JNPE** - 结果为 1 的个数为奇数则跳转

**JO** - 结果溢出了则跳转

**JNO** - 结果没有溢出则跳转

**JB, JNAE** - 小于则跳转 (无符号数)

**JNB, JAE** - 大于等于则跳转 (无符号数)

**JBE, JNA** - 小于等于则跳转 (无符号数)

**JNBE, JA** - 大于则跳转(无符号数)

**JL, JNGE** - 小于则跳转 (有符号数)

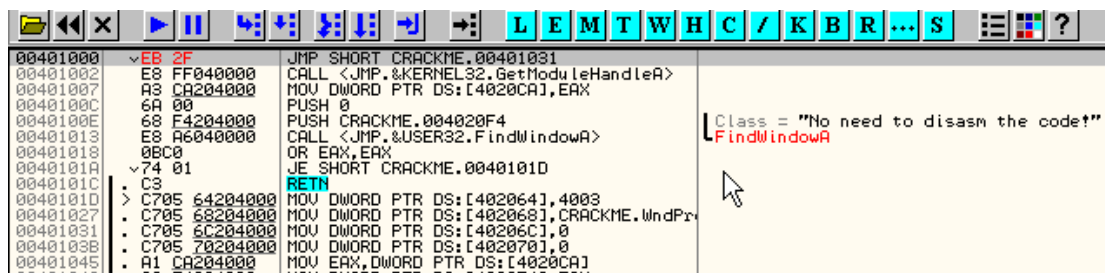
**JNL, JGE** - 大于等于则跳转 (有符号数)

**JLE, JNG** - 小于等于则跳转 (有符号数)

**JNLE, JG** - 大于则跳转(有符号数)

## JMP

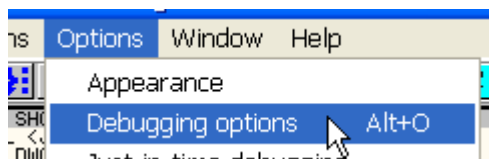
这是一个无条件跳转指令,即总是跳转到指定的地址。让我们在 OD 中来看一个例子:



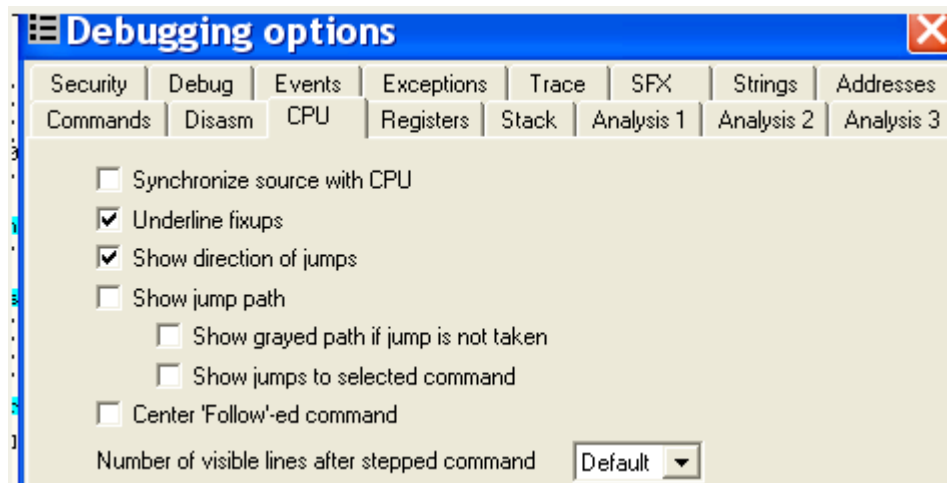
当你执行这条指令时,将跳转到 401031 处,然后从这里继续往下执行。

在 OD 中有几个选项,可以使跳转指令突出显示。

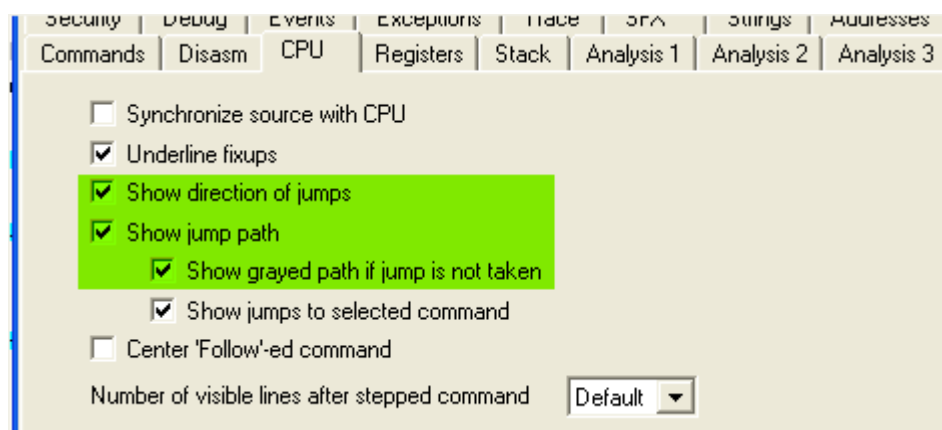
进入 OPTIONS-DEBUGGING OPTIONS:



切换到 CPU 标签页



将 3 个绿色阴影覆盖的复选框勾选上。



可以看到现在所提供的信息更加全面与清晰了。

00401000	EB 2F	JMP SHORT CRACKME.00401031
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 C8204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0
00401045	A1 C8204000	MOV EAX,DWORD PTR DS:[4020CA]
0040104D	A3 74204000	MOV DWORD PTR DS:[402074],EAX

可以看到现在 OD 中的红线展示了将跳转到何处,现在将跳转 401031 处。

按下 F7 键执行这条指令:

00401000	EB 2F	JMP SHORT CRACKME.00401031
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>
00401007	A3 C8204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0
00401045	A1 C8204000	MOV EAX,DWORD PTR DS:[4020CA]
0040104D	A3 74204000	MOV DWORD PTR DS:[402074],EAX

EIP 已经变成了 401031。

EBP	0012FFFF
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401031 CRACKME.00401031
C 1	ES 0023 32bit
P 1	CS 001B 32bit

EIP 将修改为将要跳转的地址,这里是 401031。

JE 或者 JZ

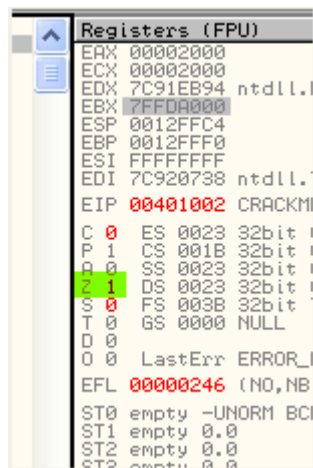
这两个条件跳转指令是等价的,只是书写的形式不同而已。我们可以看到零标志位 Z 被置 1 则跳转。

00401000	3BC1	CMP EAX,ECX
00401002	74 2D	JE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 C8204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0

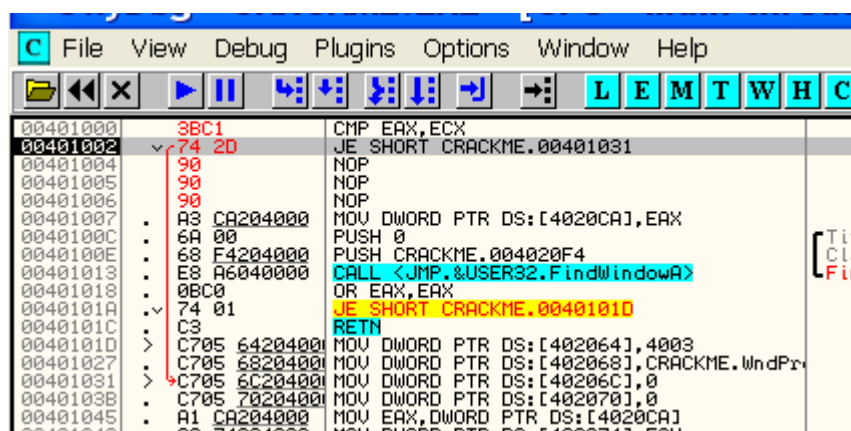
在 OD 中写入下面两条指令,我们验证一下是否会成功跳转,这里我们把 EAX 和 ECX 设置成相等的。

EAX	00002000
ECX	00002000
EDX	7C91EB94 ntdll
EBX	7FFDA000
ESP	0012FFC4
EBP	0012FFF0
ESI	FFFFFFFF
EDI	7C920738 ntdll
EIP	00401031 CRACKME.00401031
C 1	ES 0023 32bit
P 1	CS 001B 32bit

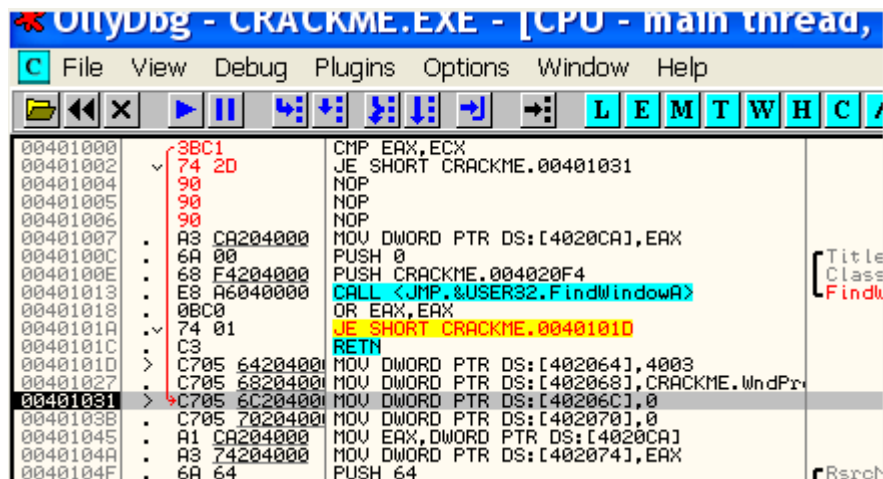
我们将两操作数相减,因为两操作数相等,结果为 0,零标志位 Z 被置 1。



我们来看看下面这个条件跳转:



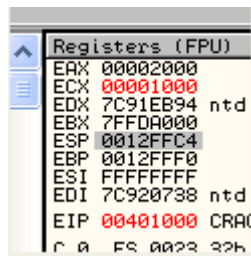
OD 提醒我们零标志位 Z 为 1,红色线显示的跳转将被执行。如果它是灰色的,那么将不会跳转。按 F7 键。



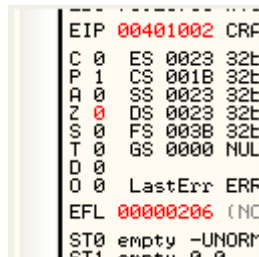
跳转已经执行了,现在 EIP 是 401031。

现在我们重新将 EAX 和 ECX 设置成不同的值。

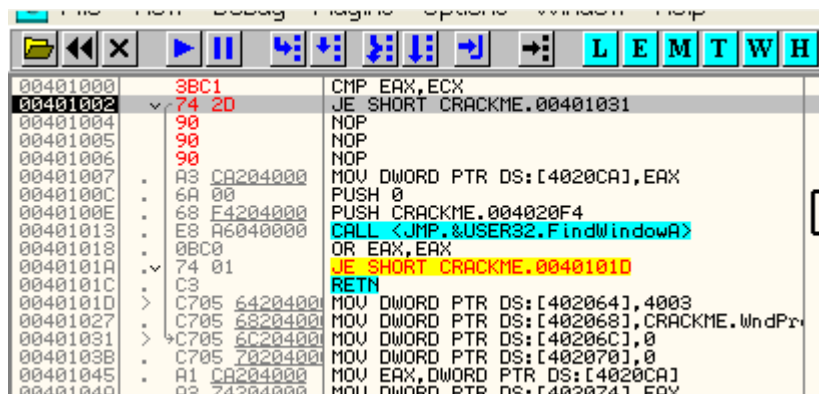




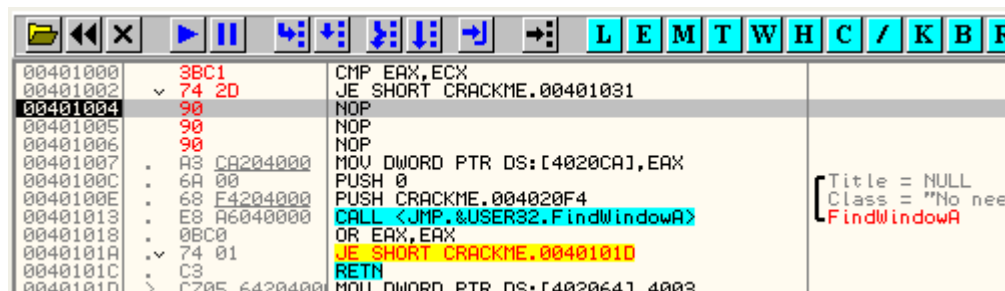
按下 F7 键,因为第一条指令的结果不为 0,所以零标志位 Z 并不会置为 1。



我们继续来 OD 中

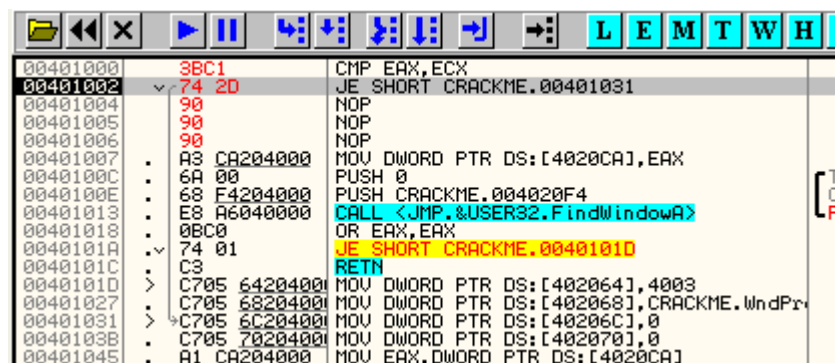


由于跳转不会发生,所以指向箭头是灰色的。继续按 F7 键。

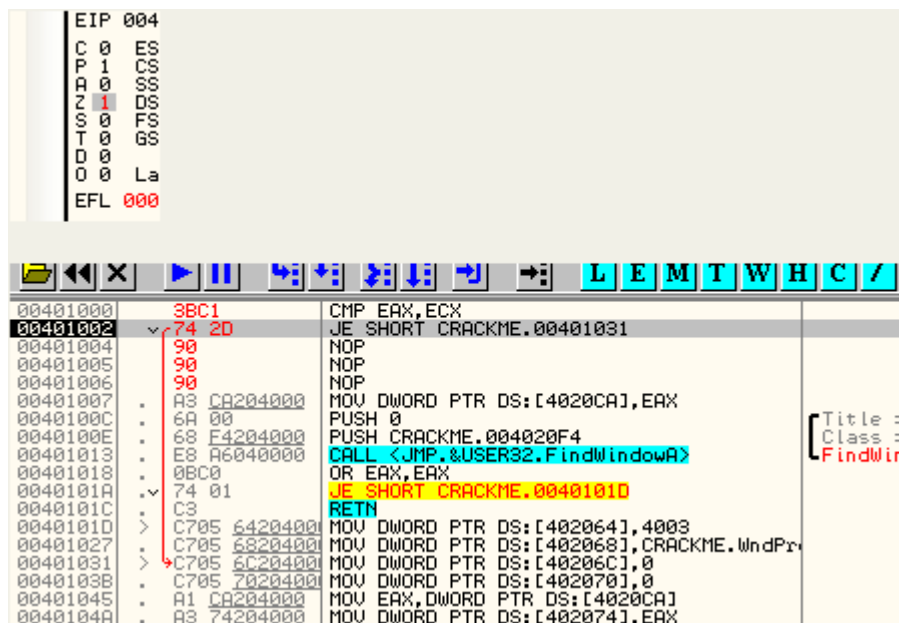


可以看到跳转并没有发生,程序继续执行到了 401004。这种比较指令在所有的程序中都是如此。

重复上面的例子,在跳转指令这里,但是我们暂时不执行。



我们知道,零标志位 Z 为 0,跳转不会成功。现在,我们双击零标志位 Z,并修改为 1 看看会发生什么?

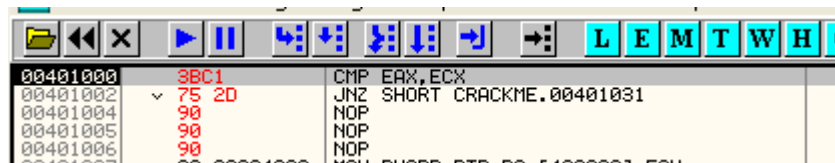


可以看到跳转线变成红色了,跳转将成功执行,不管比较的结果如何,你还是可以通过直接修改标志位的值来改变跳转的流程。

我们来简要的介绍一下其他跳转的例子。

#### JNE 或 JNZ

这条指令与上面一个指令刚好相反:如果零标志位 Z 为 0 则跳转,即,要求操作的结果非零。



这里我写的 JNZ 指令,如果 EAX 跟 ECX 相等,则零标志位 Z 置 1。

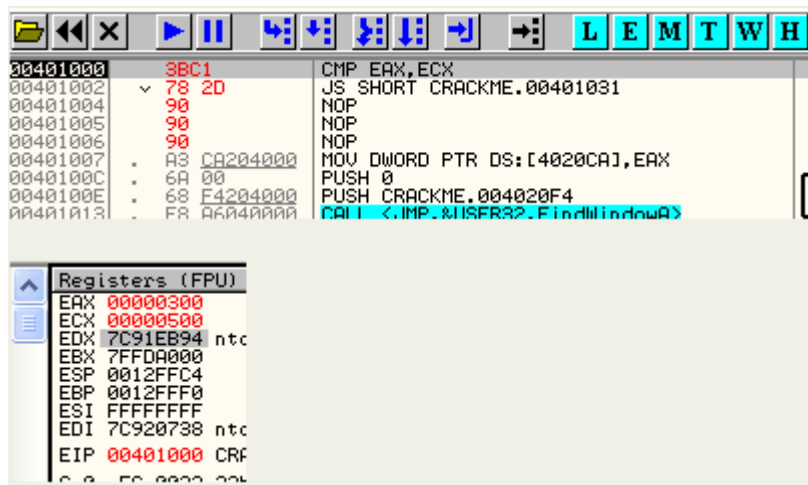


不像 JZ 指令,JZ 指令是当零标志位 Z 为 1 时跳转,JNZ 与 JZ 刚好相反,当零标志位为 0 才跳转。

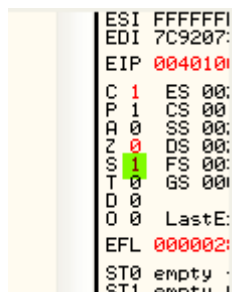
可以这么理解,如果 EAX 跟 ECX 的值不同,则运算的结果将不为 0,则零标志位为 0,就会跳转,反过来,将不会跳转。

## JS

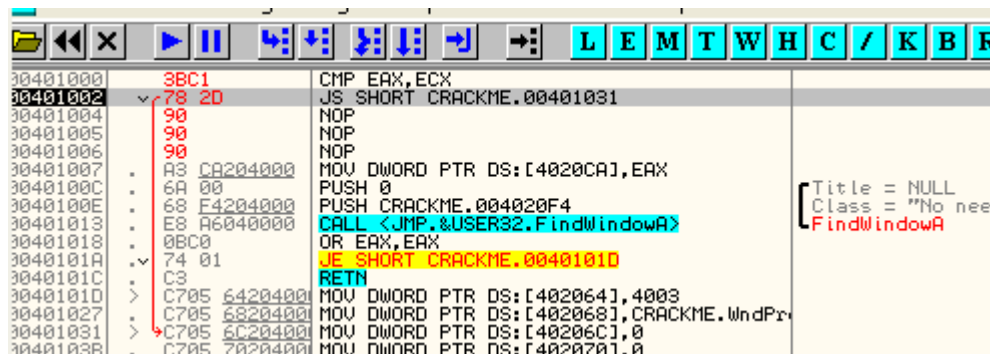
从上面的表中可以看出,当比较的结果为负时将跳转,即,按前面的例子的话就是 EAX 小于 ECX。



按 F7 键:



符号标志位 S 为 1,所以将发生跳转。



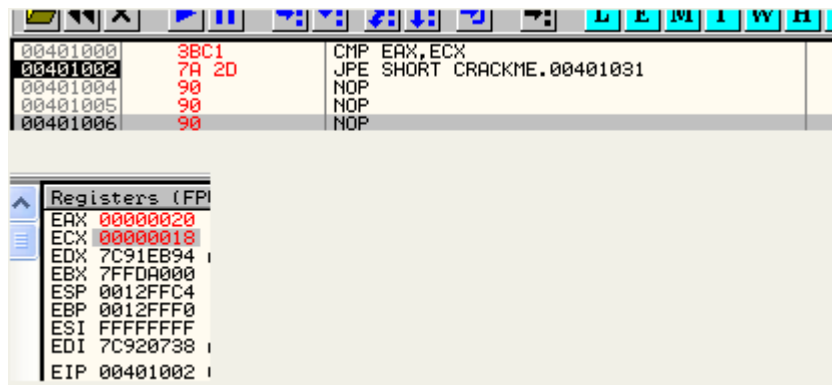
看到红线显示就表明将发生跳转。如果 EAX 大于 ECX,则符号标志位 S 为 0,结果为正,将不会发生跳转。

## JNS

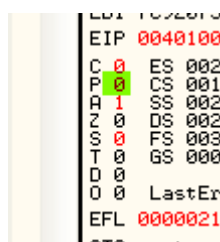
这个跳转指令与 JS 刚好相反。当零标志位 S 为 0 的时候跳转,也就是说之前例子中,EAX 大于 ECX 的时候跳转。

## JP 或 JPE

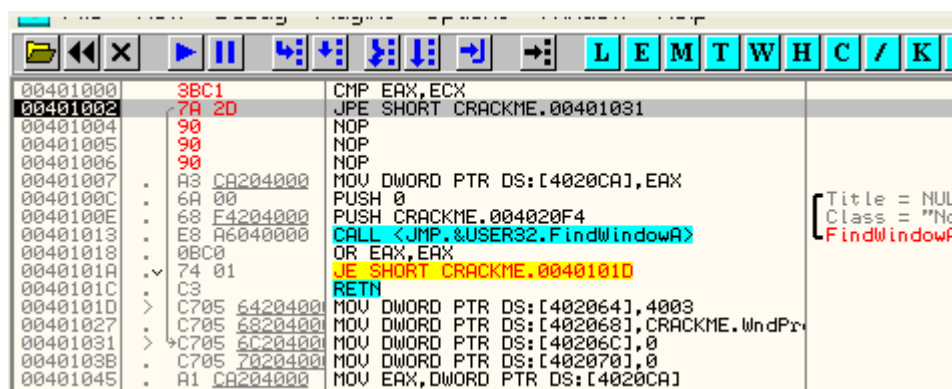
这个跳转指令时当奇偶标志位 P 置 1 的时候才会发生,也就是比较的结果中 1 的个数要是偶数。



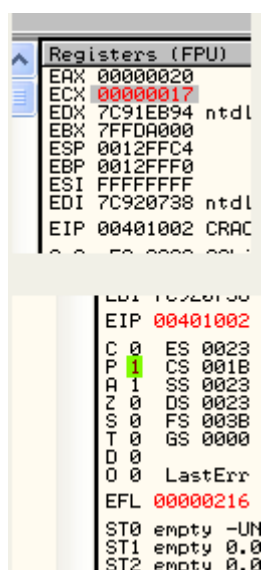
我们这里将 EAX 设置为 20,ECX 设置为 18,按 F7 键:



这里 EAX 与 ECX 的差值是 2,转化为二进制是 10,只有一个 1,所以包含奇数个 1,所以奇偶标志位为 0,JPE 指令就不会发生跳转。



现在将 ECX 的值修改为 17,然后按 F7:



可以看到结果为 3,二进制形式为 11,1 的个数为偶数个,所以奇偶标志位置 1,使用 JPE 指令就会发生跳转。

Address	Disassembly	Comment
00401000	3BC1	CMP EAX,ECX
00401002	7A 2D	JPE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX,EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064],4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068],CRACKME.WndPr
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C],0
0040103B	C705 70204000	MOV DWORD PTR DS:[402070],0
00401045	A1 CA204000	MOV EAX,DWORD PTR DS:[4020CA]

## JNP 或 JNPE

这条指令刚好与上一条指令刚好相反,当奇偶标志位 P 为 0 的时候跳转。即结果中 1 的个数为奇数的时候。在上面的例子中,当结果为 2 的时候发生跳转,当结果为 3 的时候不会发生跳转。

## JO

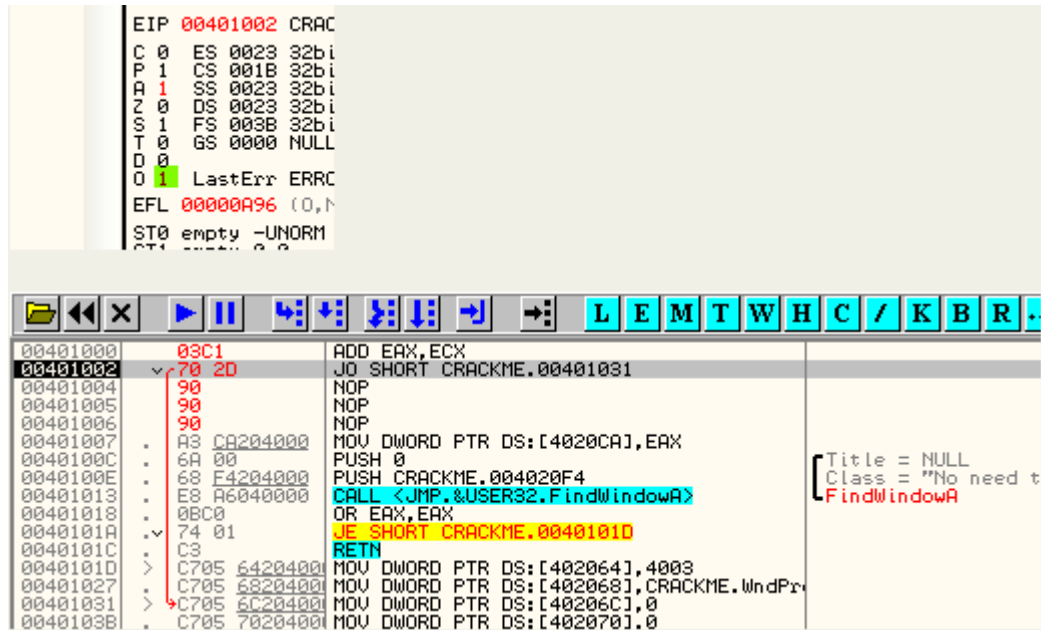
当发生溢出时,即溢出标志位 O 置 1 的时候跳转。

Address	Disassembly	Comment
00401000	03C1	ADD EAX,ECX
00401002	70 2D	JO SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA],EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4

这里我们需要修改一下指令,因为我们需要将溢出标志位 O 置位,即溢出发生,这里我们可以通过加法指令来实现。

Register	Value	Comment
EAX	7FFFFFFF	
ECX	00000001	
EDX	00000000	
EBX	7FFDA000	
ESP	0012FFC4	
EBP	0012FFF0	
ESI	FFFFFFFF	
EDI	7C920738	ntdll
EIP	00401000	CRACKME
CS	0023	32bit
SS	0023	32bit

按 F7 键:



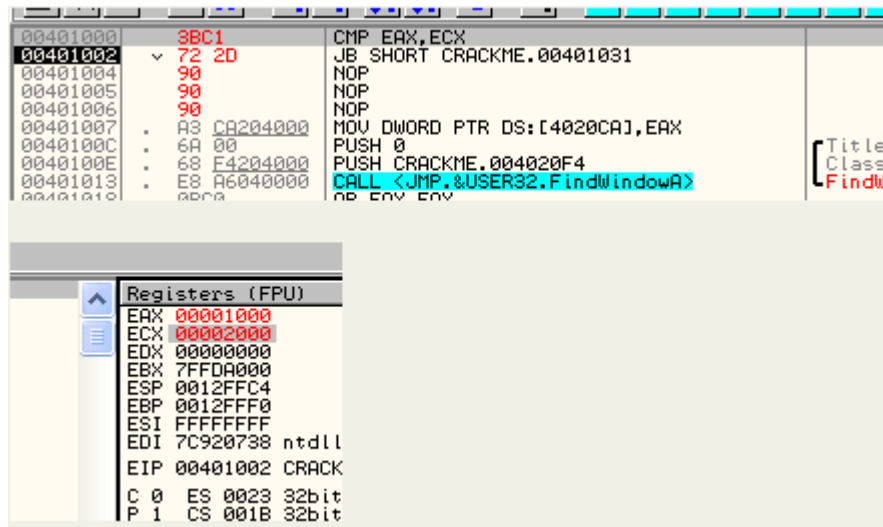
这里跳转将执行,因为溢出标志位 O 被置为了 1。

## JNO

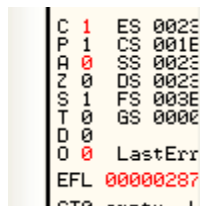
跟上一条指令相反,这里是当溢出标志位 O 为 0 时跳转,即溢出没有发生时。

## JB

如果第一个操作数小于第二个操作数的时候跳转。这里我们来看一个例子。



可以看到 EAX 小于 ECX,所以会发生跳转。按 F7 键:



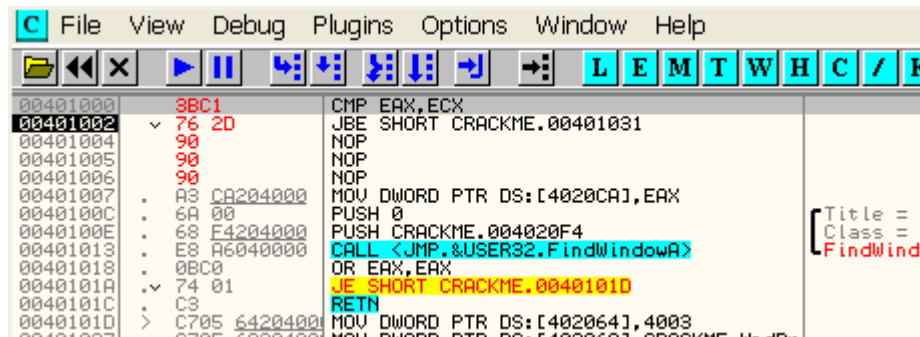
进位/借位标志位置 1,当两个操作数的差值为负的时候,该标志位将被置 1,这里我们将可以得出 EAX 小于 ECX。

JNB

和 JB 指令相反,这个指令是当进位/借位标志位为 0 的时候跳转,也就是说,结果为正的时候跳转。在前面的例子中,因为 EAX 小于 ECX,所以不会发生跳转。

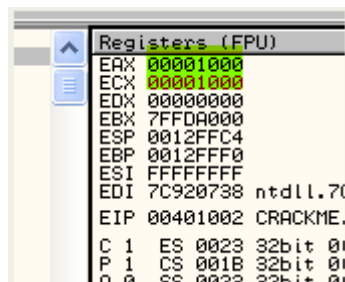
JBE

这个指令是小于或者等于的时候跳转,这是判断两个标志位的,当进位/借位标志位置 1 或者零标志位置 1 的时候将发生跳转,也就是说,EAX 要小于或者等于 ECX 才会发生跳转。



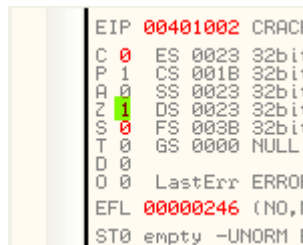
```
00401000 3BC1 CMP EAX,ECX
00401002 76 2D JBE SHORT CRACKME.00401031
00401004 90 NOP
00401005 90 NOP
00401006 90 NOP
00401007 A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C 6A 00 PUSH 0
0040100E 68 F4204000 PUSH CRACKME.004020F4
00401013 E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018 0BC0 OR EAX,EAX
0040101A 74 01 JE SHORT CRACKME.0040101D
0040101C C3 RETN
0040101D C705 64204000 MOV DWORD PTR DS:[402064],4003
00401027 C705 20204000 MOV DWORD PTR DS:[402060],CRACKME.WndProc
```

我们首先让 EAX 和 ECX 相等。



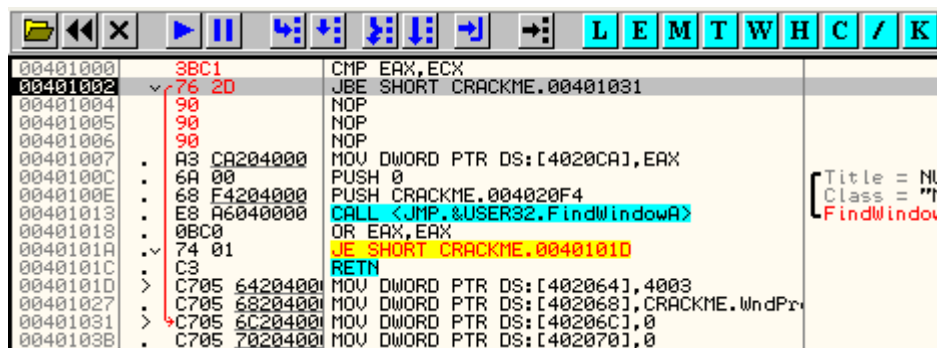
```
Registers (FPU)
EAX 00001000
ECX 00001000
EDX 00000000
EBX 7FFDA000
ESP 0012FFC4
EBP 0012FFF0
ESI FFFFFFFF
EDI 7C920738 ntdll.7C920738
EIP 00401002 CRACKME.00401002
C 1 ES 0023 32bit 00000000
P 1 CS 001B 32bit 00000000
D 0 SS 0023 32bit 00000000
```

按 F7 键。



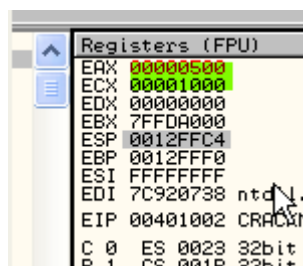
```
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 00000000
P 1 CS 001B 32bit 00000000
A 0 SS 0023 32bit 00000000
Z 1 DS 0023 32bit 00000000
S 0 FS 003B 32bit 00000000
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR
EFL 00000246 (NO,INTERRUPT)
ST0 empty _UNORM E
```

可以看到零标志位 Z 置 1 了。

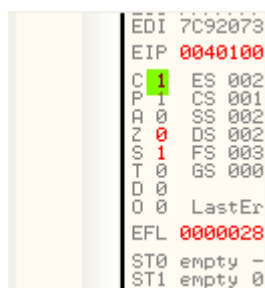


```
00401000 3BC1 CMP EAX,ECX
00401002 76 2D JBE SHORT CRACKME.00401031
00401004 90 NOP
00401005 90 NOP
00401006 90 NOP
00401007 A3 CA204000 MOV DWORD PTR DS:[4020CA],EAX
0040100C 6A 00 PUSH 0
0040100E 68 F4204000 PUSH CRACKME.004020F4
00401013 E8 A6040000 CALL <JMP.&USER32.FindWindowA>
00401018 0BC0 OR EAX,EAX
0040101A 74 01 JE SHORT CRACKME.0040101D
0040101C C3 RETN
0040101D C705 64204000 MOV DWORD PTR DS:[402064],4003
00401027 C705 68204000 MOV DWORD PTR DS:[402068],CRACKME.WndProc
00401031 C705 6C204000 MOV DWORD PTR DS:[40206C],0
0040103B C705 70204000 MOV DWORD PTR DS:[402070],0
```

现在让 EAX 小于 ECX:

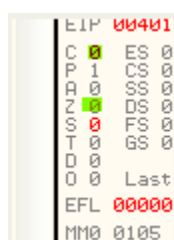


按 F7 键:

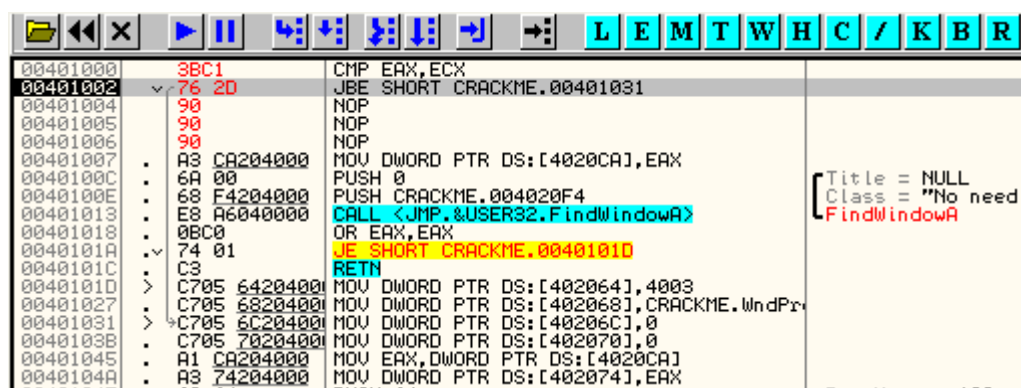


在这种情况下, 进位/借位标志位置 1, 因为结果是负的, 也就是说 EAX 小于 ECX。

最后一个例子, 让 EAX 大于 ECX, 然后按 F7 键



进位/借位标志位 C 与零标志位 Z 都为 0, 所以跳转不会发生。



也就是说, 这个例子中我们要 EAX 小于或者等于 ECX 的时候才会发生跳转。

## JNBE

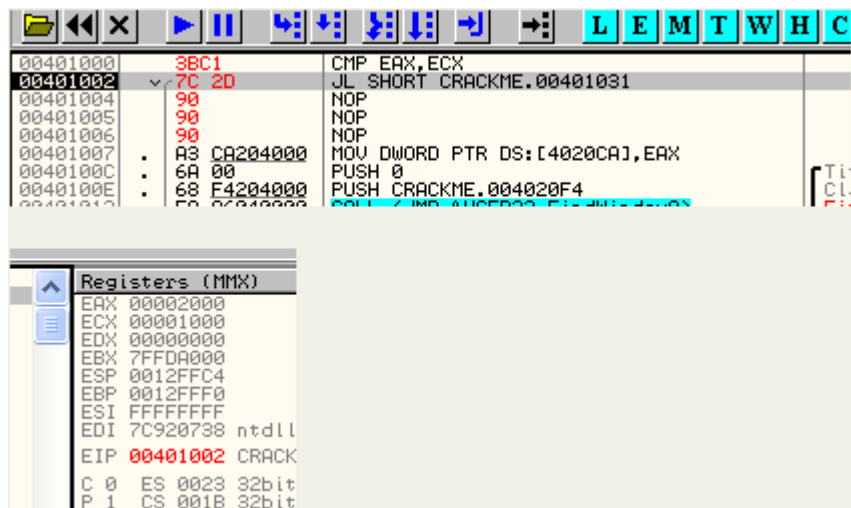
这个指令跟 JBE 刚好相反, 当进位/借位标志位 C 与零标志位 Z 都为 0 时候才会发生跳转。

## JL

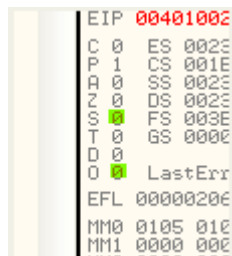
这个指令当小于的时候跳转, 但是与前面的 JB 稍微有点不同。这个指令时根据符号标志位 S 来决定是否跳转。

来看看这样一个例子, 这里 EAX 和 ECX 都是整数, 并且 EAX 大于 ECX。

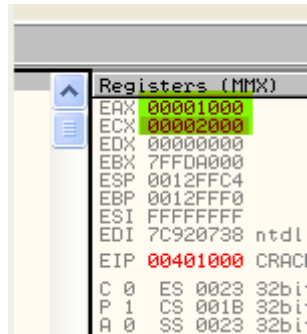




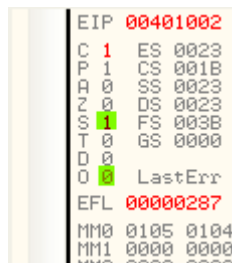
当你按下 F7 键的时候将不会发生跳转,因为 EAX 与 ECX 的差值是正数,所以符号标志位 S 和溢出标志位 O 将不会置 1。



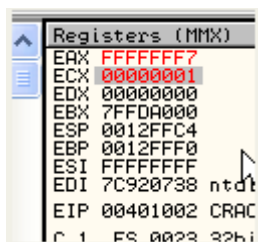
还是上面的例子, 这里我们让 EAX 小于 ECX,并且两者依然是正数。



按 F7 键:

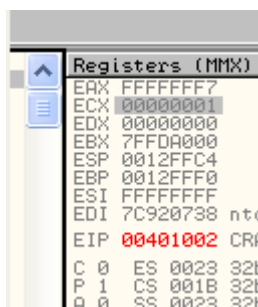


这里是操作数 1 小于操作数 2,并且两者都是正数,所以溢出标志位 O 与符号标志位 S 不同,所以跳转将发生,我们再来看另外一个例子。

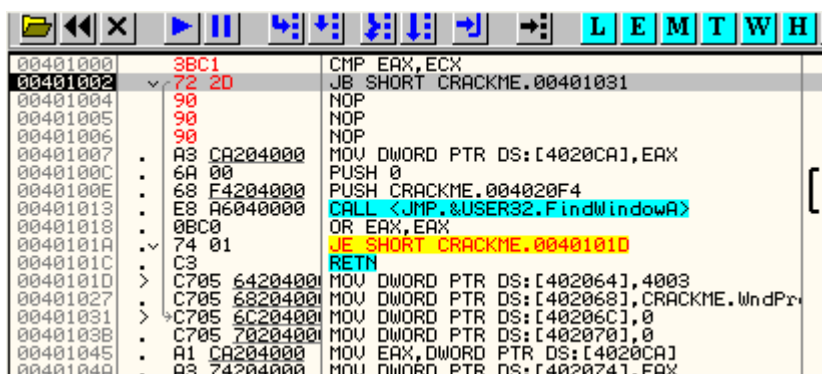


这种情况下,EAX 小于 ECX,并且 EAX 为负数。看看将发生什么。

这里跳转将执行,现在我们尝试相同的值用 JB 指令来试试。



按 F7 键。



可以看到 JB 指令并不会发生跳转,因为 JB 比较两个数的时候,将它们两个都看做是正数,即认为它们是无符号数,但是 JL 指令要考虑符号,这就是这两条指令的主要区别。

Перейти, если	Сравнения совершаются с	
	Числа со знаком	Числа без знака
Больше / Больше чем	JA	JG
Равно	JE	JE
Не равно	JNE	JNE
Меньше / Меньше чем	JB	JL
Меньше или Равно / Меньше чем или Равно	JBE	JLE
Больше или Равно / Больше чем или Равно	JAE	JGE

(此表由于是图片,就不翻译了)

可以看到,这些条件跳转指令被分为了两类:那些我们需要考虑符号的,那些不需要考虑符号的。

JA,JB,JBE,JAЕ 的两个操作数都是正数(无符号数),而 JG,JL,JLE,JGE 把两个操作数都看成有符号数。

我相信,比较和条件跳转指令的介绍将凸显出它们在程序中是如何使用的。

接下来的部分将介绍 `call` 和 `ret` 指令。要多点耐性,嘿嘿。