王爽汇编第10章,call和ret指令

1. call和ret指令概述:

`call` 和 `ret` 指令都是转移指令，它们都修改IP，或同时修改CS和IP。他们经常被用来实现子程序(函数)的设计。

2. ret和retf

2.1. ret指令

`ret` 指令：用栈中的数据，修改IP的内容，从而实现(`近转移`)；
CPU执行ret指令时，需要进行下面两个步骤：

相当于： `pop IP`

- (1) (IP) = ((ss)*16+(sp))
- (2) (SP)=(sp)+2

2.2. retf指令

`retf` 指令：用栈中的数据，修改CS和IP的内容，从而实现(`远转移`)。

CPU执行retf指令时，需要进行下面四个步骤

相当于： `pop CS,pop IP`

- (1) (IP) = ((ss)*16+(sp))
- (2) (SP) = (sp)+2
- (3) (CS) = ((ss)*16+(sp))
- (4) (SP) = (sp)+2

```
;**********************************
;            ret指令实验            *
;**********************************
assume cs:code,ss:stack

;>>>>>>>>>>>>>>>>>>>>>
;堆栈段
;>>>>>>>>>>>>>>>>>>>>>
stack segment
    db 16 dup(0)
stack ends

```

```
;>>>>>>>>>>>>>>>>>>>>
;代码段
;>>>>>>>>>>>>>>>>>>>>
code segment
          mov ax,4c00h
          int 21h;退出到DOS
main:;程序入口
        mov ax,stack
        mov ss,ax    ;关联堆栈段
        mov sp,16    ;设置栈顶
        mov ax,0
        push ax      ;压栈
      ret          ;pop ip, ->cs:ip cs:0 ,让程序跳到入口处上面
code ends
end main
```

2.3. call 和 ret 的配合使用

call 与 ret 指令共同支持了汇编语言编程中的模块化设计

```
assume cs:code
code segment
start:  mov ax,1
        mov cx,3
        call s ; (1) CPU指令缓冲器存放call指令, IP指向下一条指令 (mov bx, ax) , 执行call指令, IP入栈, jmp

        mov bx,ax   ; (4) IP重新指向这里  bx = 8
        mov ax,4c00h
        int 21h
    s: add ax,ax
       loop s; (2) 循环3次ax = 8
       ret; (3) return : pop IP
code ends
end start
```

3. call指令详解

在X86架构下：call基本都是调用一个函数,比如调用 `MessageBox` ，在汇编中就会写成 `Call MessageBox` ，并且 `call` 经常和 `ret` 搭配使用，下面我们来说说call的原理。

3.1. call原理

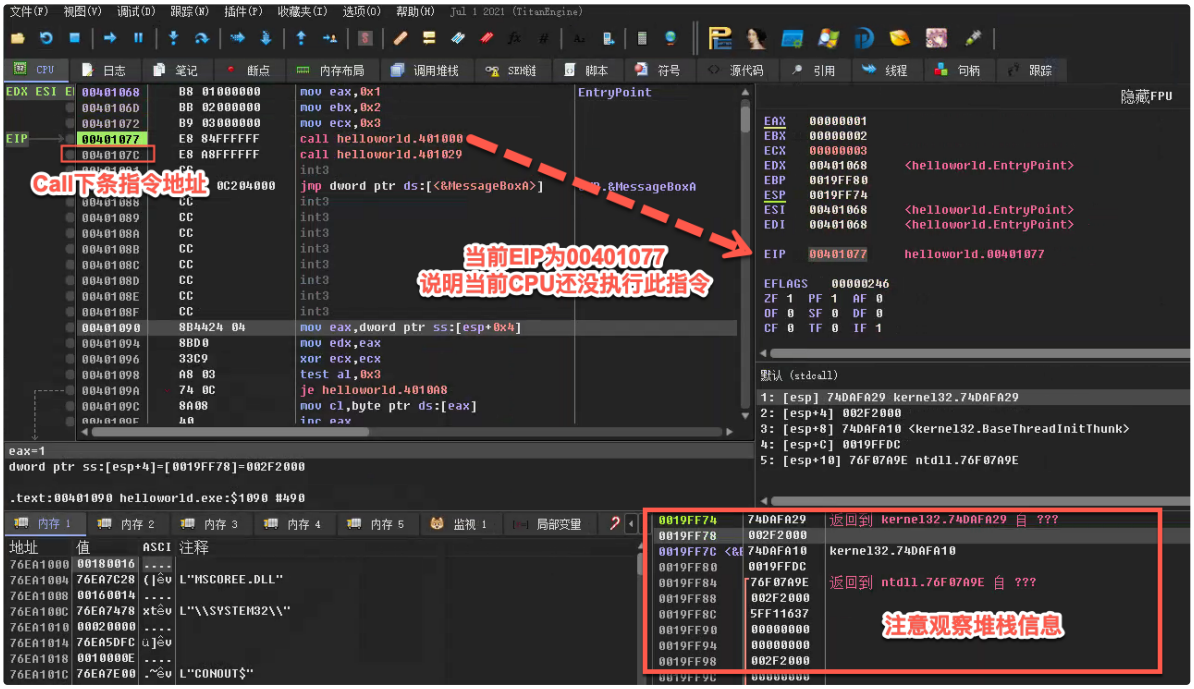
CPU执行call指令时，会进行如下两个步骤：

- (1) 将当前的IP或者CS:IP压入栈中;
- (2) 转移指令(jmp)

我们这里先来看看 `x86` 架构下EXE执行call的流程。

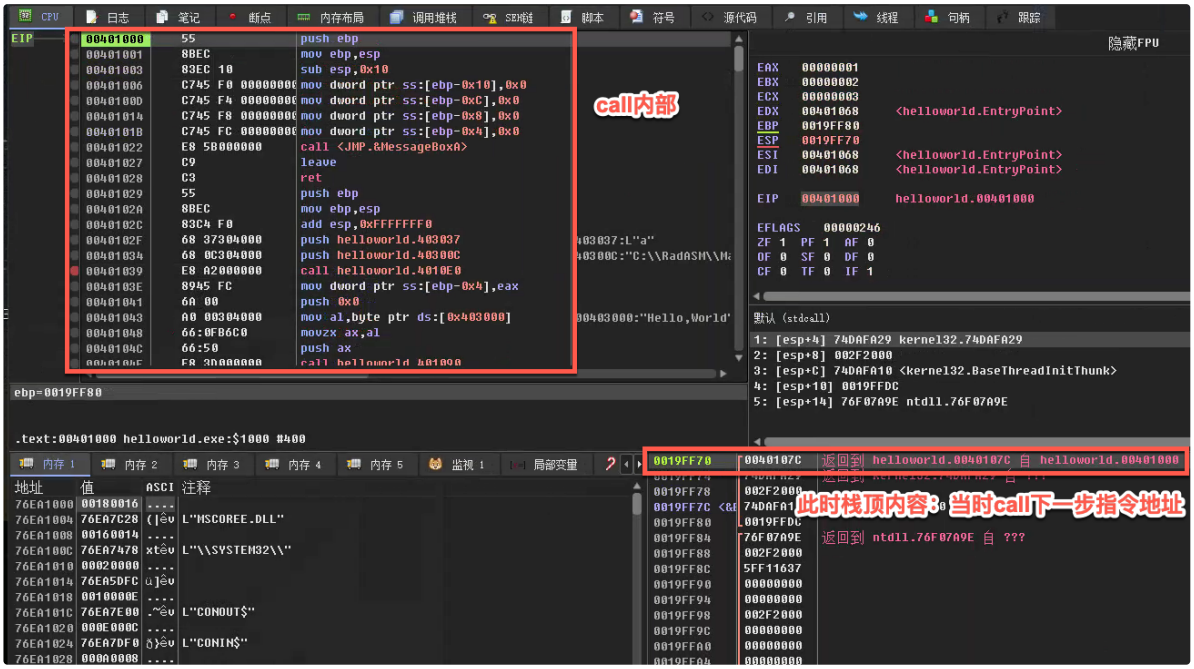
1. 用 `x32dbg` 打开 `HelloWorld.exe` ，然后反汇编视图如下：

   此时我们还没执行Call，然后我们先记一下Call下条指令的地址 `0x0040107C` ，然后接着第2步骤按 `F7` 进入到call内部，然后此时注意观察堆栈的信息。



2. 我们按 `F7` 进入到call内容。

   第一步里面我们记下来的 `0x0040107C` 地址，被push到了堆栈里面，并且 `jmp` 到了函数的地址 `00401000` ，所以验证了之前CPU执行call指令会有两个步骤(push ip，并且jmp 到call 函数的地址)。



3.2. call指令所有写法

`8086` CPU架构中：

- call 标号（近转移）

```
push ip
jmp near ptr 标号
```

- call far ptr 标号（段间转移）

```
push cs
push ip
jmp fat ptr 标号
```

- call 16位寄存器

```
push ip
jmp ax
```

- call word ptr 内存单元地址

```
mov ax, 0123h
mov ds:[0],ax
call word ptr ds:[0]
;等于
push ip
jmp word ptr ds:[0]
```

- call dword ptr 内存单元地址

```
mov ds:[0],ax
mov word ptr ds:[2],0
call dword ptr ds:[0]
;等于 cs:ip 0:0123h
push cs
push ip
jmp dword ptr ds:[0]
```

`X86` CPU架构中：

先来看一张 `Intel X86架构手册` 的图片。

CALL—Call Procedure

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
E8 *cw*	CALL *rel16*	B	N.S.	Valid	Call near, relative, displacement relative to next instruction.
E8 *cd*	CALL *rel32*	B	Valid	Valid	Call near, relative, displacement relative to next instruction. 32-bit displacement sign extended to 64-bits in 64-bit mode.
FF /2	CALL *r/m16*	B	N.E.	Valid	Call near, absolute indirect, address given in *r/m16*.
FF /2	CALL *r/m32*	B	N.E.	Valid	Call near, absolute indirect, address given in *r/m32*.
FF /2	CALL *r/m64*	B	Valid	N.E.	Call near, absolute indirect, address given in *r/m64*.
9A *cd*	CALL *ptr16:16*	A	Invalid	Valid	Call far, absolute, address given in operand

- E8 cw(w表示word的意思|代表后面要跟两个字节) - 近转移 位移

```
!注意! CPU在实模式下, 0xE8才接受2字节操作数
0xE8 0x04 0x00  call 标号 偏移:0x04
```

- E8 cd(d表示dword的意思)

```
在保护式下，0xE8接受4字节操作数
0xE8 0x04 0x03 0x02 0x01  call 偏移:0x01234
```

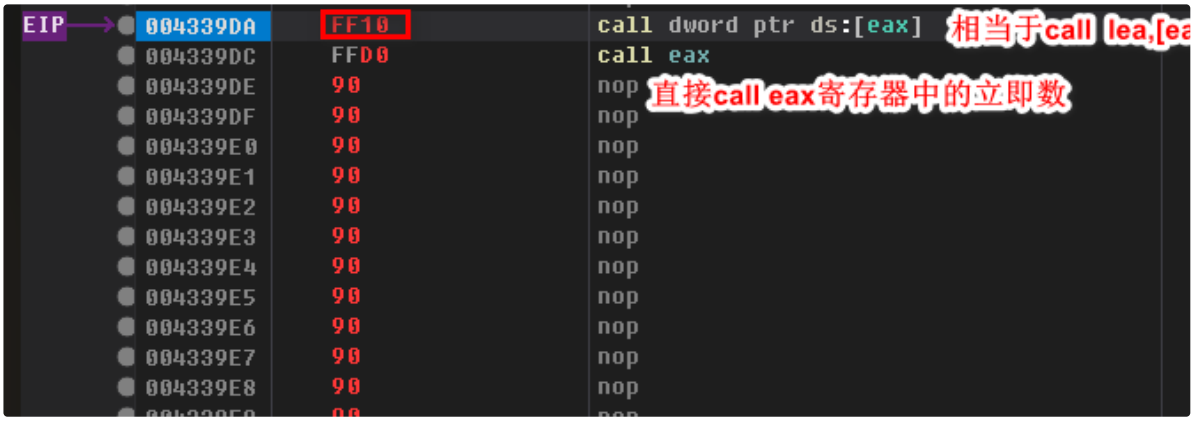
- FF /2 (r/m32)

FF /2，是 0xFF 后面跟着一个 /digit 表示的东西。如下图 2 对应的就是DL带头的那一列，标红的这32个值代表了32种寻址方式。

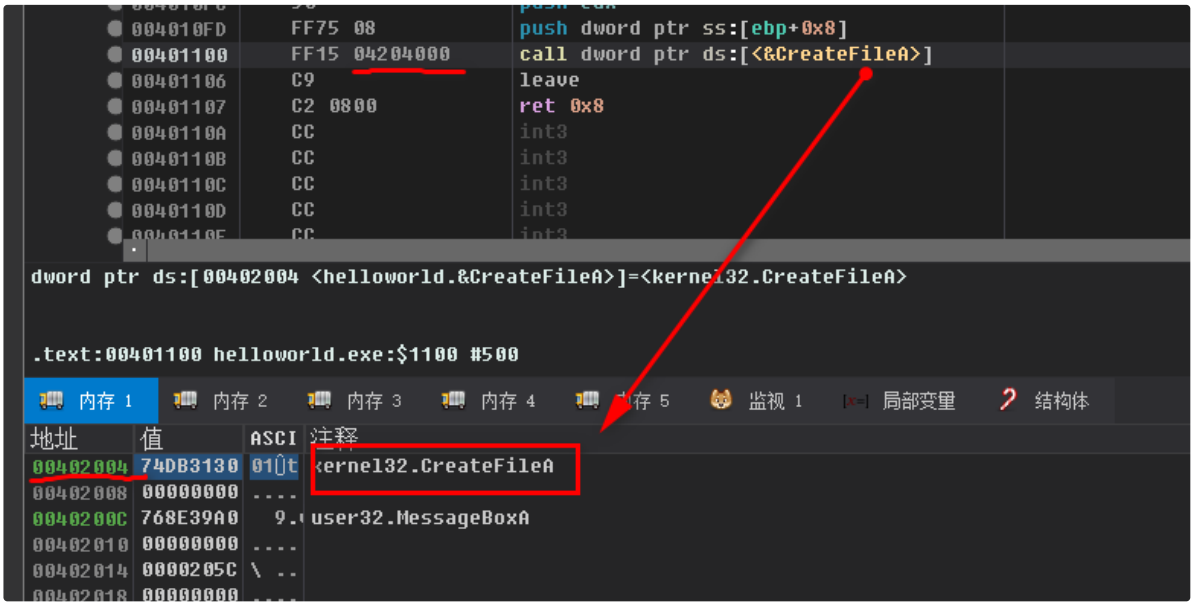
Table 2-2. 32-Bit Addressing Forms with the ModR/M Byte

r8(/r) r16(/r) r32(/r) mm(/r) xmm(/r) (In decimal) /digit (Opcode) (In binary) REG =			AL AX EAX MM0 XMM0 0 000	CL CX ECX MM1 XMM1 1 001	DL DX EDX MM2 XMM2 2 010	BL BX EBX MM3 XMM3 3 011	AH SP ESP MM4 XMM4 4 100	CH BP EBP MM5 XMM5 5 101	DH SI ESI MM6 XMM6 6 110	BH DI EDI MM7 XMM7 7 111
Effective Address	**Mod**	**R/M**	**Value of ModR/M Byte (in Hexadecimal)**							
[EAX]	00	000	00	08	10	18	20	28	30	38
[ECX]		001	01	09	11	19	21	29	31	39
[EDX]		010	02	0A	12	1A	22	2A	32	3A
[EBX]		011	03	0B	13	1B	23	2B	33	3B
[--][--][1]		100	04	0C	14	1C	24	2C	34	3C
disp32[2]		101	05	0D	15	1D	25	2D	35	3D
[ESI]		110	06	0E	16	1E	26	2E	36	3E
[EDI]		111	07	0F	17	1F	27	2F	37	3F
[EAX]+disp8[3]	01	000	40	48	50	58	60	68	70	78
[ECX]+disp8		001	41	49	51	59	61	69	71	79
[EDX]+disp8		010	42	4A	52	5A	62	6A	72	7A
[EBX]+disp8		011	43	4B	53	5B	63	6B	73	7B
[--][--]+disp8		100	44	4C	54	5C	64	6C	74	7C
[EBP]+disp8		101	45	4D	55	5D	65	6D	75	7D
[ESI]+disp8		110	46	4E	56	5E	66	6E	76	7E
[EDI]+disp8		111	47	4F	57	5F	67	6F	77	7F
[EAX]+disp32	10	000	80	88	90	98	A0	A8	B0	B8
[ECX]+disp32		001	81	89	91	99	A1	A9	B1	B9
[EDX]+disp32		010	82	8A	92	9A	A2	AA	B2	BA
[EBX]+disp32		011	83	8B	93	9B	A3	AB	B3	BB
[--][--]+disp32		100	84	8C	94	9C	A4	AC	B4	BC
[EBP]+disp32		101	85	8D	95	9D	A5	AD	B5	BD
[ESI]+disp32		110	86	8E	96	9E	A6	AE	B6	BE
[EDI]+disp32		111	87	8F	97	9F	A7	AF	B7	BF
EAX/AX/AL/MM0/XMM0	11	000	C0	C8	D0	D8	E0	E8	F0	F8
ECX/CX/CL/MM/XMM1		001	C1	C9	D1	D9	E1	E9	F1	F9
EDX/DX/DL/MM2/XMM2		010	C2	CA	D2	DA	E2	EA	F2	FA
EBX/BX/BL/MM3/XMM3		011	C3	CB	D3	DB	E3	EB	F3	FB
ESP/SP/AH/MM4/XMM4		100	C4	CC	D4	DC	E4	EC	F4	FC
EBP/BP/CH/MM5/XMM5		101	C5	CD	D5	DD	E5	ED	F5	FD
ESI/SI/DH/MM6/XMM6		110	C6	CE	D6	DE	E6	EE	F6	FE
EDI/DI/BH/MM7/XMM7		111	C7	CF	D7	DF	E7	EF	F7	FF


```
;call的是取寄存器地址的值
FF 10 call dword ptr [eax]
FF 11 call dword ptr [ecx]
FF 12 call dword ptr [edx]
FF 13 call dword ptr [ebx]
;call的是寄存器的值
FF D0 call eax
FF D1 call ecx
FF D2 call edx
FF D3 call ebx
```




```
FF 15 [地址] 常见于调用Windows的导出表比如:
call dword ptr ds:[<&CreateFileA>]
```



- FF /2 (r/m64)

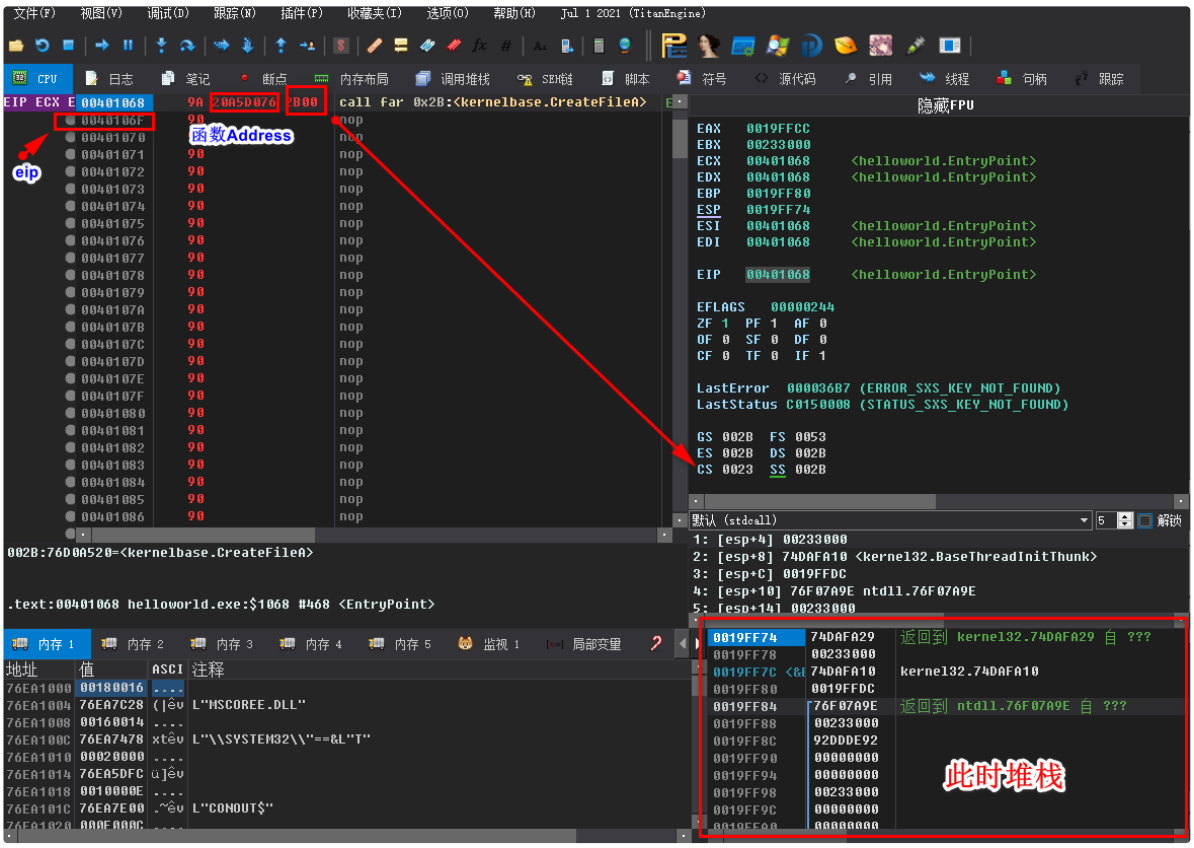
```
同32位，只是寄存器和地址都是64位的
```

- 9A cd(d表示dword的意思) -- call far ptr 标号(段间转移)

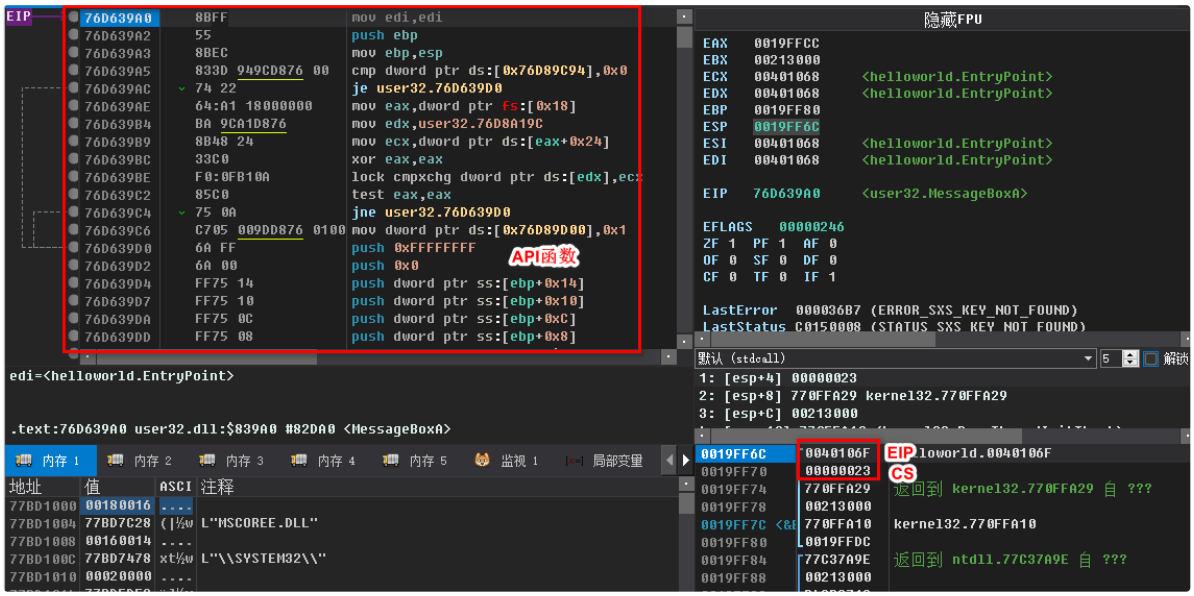
```
类似8086中的call far ptr
9A xx xx xx xx xx xx
其中最后两个xx xx = 段地址
9A后面4个xx       = 要call的地址

push cs
push eip
jmp xx xx xx xx
```

执行 `call far ptr 标号` 前的数据

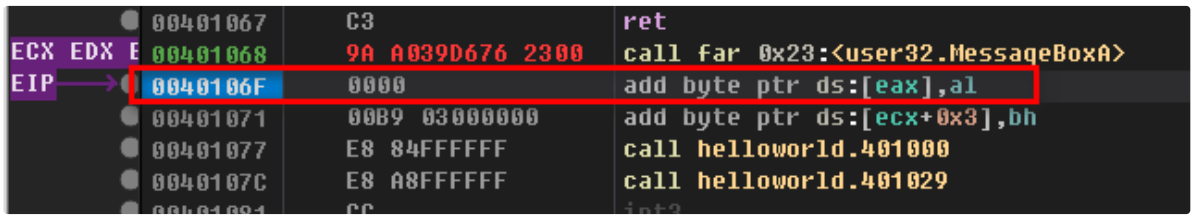


跟入call后的数据，注意堆栈的内容。



执行ret后

返回到了call调用处



3.3. call 指令大全图表

表格

指令	二进制格式
call rel32	E8 xx xx xx xx
call dword ptr [EAX]	FF 10
call dword ptr [ECX]	FF 11
call dword ptr [edx]	FF 12
call dword ptr [ebx]	FF 13
call dword ptr [REG * SCALE+BASE]	FF 14 xx
call dword ptr [Address]	FF 15 xx xx xx xx
call dword ptr [ESI]	FF 16
call dword ptr [EDI]	FF 17
call dword ptr [EAX+xx]	FF 50 xx
call dword ptr [ECX+xx]	FF 51 xx
call dword ptr [EDX+xx]	FF 52 xx
call dword ptr [EBX+xx]	FF 53 xx
call dword ptr [REG*SCALE+BASE+offset8]	FF 54 xx xx
call dword ptr [EBP+xx]	FF 55 xx
call dword ptr [ESI+xx]	FF 56 xx
call dword ptr [EDI+xx]	FF 57 xx
call dword ptr [EAX+xxxxxxxx]	FF 90 xx xx xx xx
call dword ptr [ECX+xxxxxxxx]	FF 91 xx xx xx xx
call dword ptr [EDX+xxxxxxxx]	FF 92 xx xx xx xx
call dword ptr [EBX+xxxxxxxx]	FF 93 xx xx xx xx
call dword ptr [REG*SCALE+BASE+offset32]	FF 94 xx xx xx xx xx
call dword ptr [EBP+xxxxxxxx]	FF 95 xx xx xx xx
call dword ptr [ESI+xxxxxxxx]	FF 96 xx xx xx xx
call dword ptr [EDI+xxxxxxxx]	FF 97 xx xx xx xx
call eax	FF D0
call ecx	FF D1
call edx	FF D2
call ebx	FF D3
call esp	FF D4
call ebp	FF D5
call esi	FF D6
call edi	FF D7

指令	二进制格式
call FAR seg16:Address	9A xx xx xx xx xx xx

图

指令	二进制形式
CALL rel32	E8 xx xx xx xx
CALL dword ptr [EAX]	FF 10
CALL dword ptr [ECX]	FF 11
CALL dword ptr [EDX]	FF 12
CALL dword ptr [EBX]	FF 13
CALL dword ptr [REG*SCALE+BASE]	FF 14 xx
CALL dword ptr [abs32]	FF 15 xx xx xx xx
CALL dword ptr [ESI]	FF 16
CALL dword ptr [EDI]	FF 17
CALL dword ptr [EAX+xx]	FF 50 xx
CALL dword ptr [ECX+xx]	FF 51 xx
CALL dword ptr [EDX+xx]	FF 52 xx
CALL dword ptr [EBX+xx]	FF 53 xx
CALL dword ptr [REG*SCALE+BASE+off8]	FF 54 xx xx
CALL dword ptr [EBP+xx]	FF 55 xx
CALL dword ptr [ESI+xx]	FF 56 xx
CALL dword ptr [EDI+xx]	FF 57 xx
CALL dword ptr [EAX+xxxxxxxx]	FF 90 xx xx xx xx
CALL dword ptr [ECX+xxxxxxxx]	FF 91 xx xx xx xx
CALL dword ptr [EDX+xxxxxxxx]	FF 92 xx xx xx xx
CALL dword ptr [EBX+xxxxxxxx]	FF 93 xx xx xx xx
CALL dword ptr [REG*SCALE+BASE+off32]	FF 94 xx xx xx xx xx
CALL dword ptr [EBP+xxxxxxxx]	FF 95 xx xx xx xx
CALL dword ptr [ESI+xxxxxxxx]	FF 96 xx xx xx xx
CALL dword ptr [EDI+xxxxxxxx]	FF 97 xx xx xx xx
CALL EAX	FF D0
CALL ECX	FF D1
CALL EDX	FF D2
CALL EBX	FF D3
CALL ESP	FF D4
CALL EBP	FF D5
CALL ESI	FF D6
CALL EDI	FF D7
CALL FAR seg16:abs32	9A xx xx xx xx xx xx

> 参考资料:
> https://zhuanlan.zhihu.com/p/68588184 CALL指令有多少种写法
> https://blog.csdn.net/qq_39654127/article/details/88698911 王爽《汇编语言》笔记（详细）