

汇编语言与逆向工程

北京邮电大学 2019年3月





- □1.寄存器
- □2.x86指令集
- □3.寻址方式
- □4.字节序
- □5.栈
- □6.函数调用约定





- □一种线性的数据结构
 - 〇先入后出,
 - *后入栈的数据会比之前入栈的数据先出栈。
 - ○栈操作主要有两种——压栈和弹栈,
 - ❖每次压栈会把把数据从栈顶压入,栈的长度增加,栈顶位置上升;
 - ❖每次弹栈也是从栈顶弹出一个数据,栈的长度减少,栈顶位置下降。
 - 〇依次向栈中压入1、2、3三个数据, 其弹出顺序为3、2、1。

北邮 网安学院 崔宝江





- @程序栈
- @栈的布局
- @栈的生成与销毁



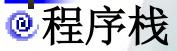




- □栈在程序中起着不可替代的作用
 - ○每一个函数拥有自己的函数栈
 - ○函数栈中保存着函数的局部变量、流控制结构等等
 - ○每次一个函数调用后函数栈会被收回,并再次分配 给其他被调用的函数
- □程序栈是由高地址向低地址生长
 - ○在程序中函数站的栈顶地址是比栈底地址低的







- 〇x86框架为例,支持栈的寄存器包括ESP和EBP,以ESP和EBP作为栈的边界
- OESP是栈寄存器,指示当前栈顶的位置
 - ❖该寄存器在一个函数调用过程中会随着数据的压入弹出不断改变
- OEBP是栈基址寄存器,这个寄存器指向栈底的位置
 - *主要作用是做栈划分
 - ❖在一个函数执行的过程中,该寄存器的值在调用其他函数时暂时改变为其他函数栈的栈底位置,当其他函数调用结束后会恢复为原函数栈栈底位置



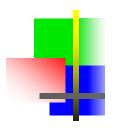




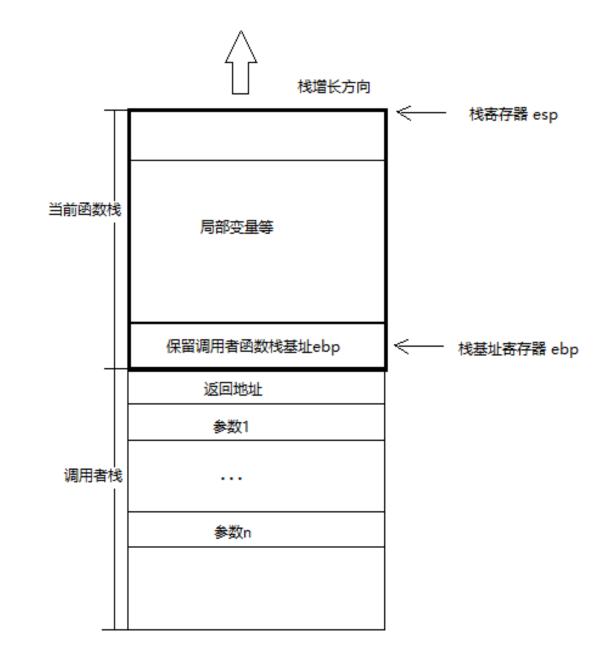
- @程序栈
 - □x86指令集中,与栈操作相关的指令: push、pop、call、leave、ret
 - Opush和pop是栈的基本操作
 - Ocall、leave、ret可以用来构建函数栈和销毁函数 栈。







低地址







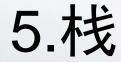




- □栈的生长是从高地址向低地址生长的,下图展 示了栈在内存中是如何布局的。
- □每一次函数的调用会生成一个新的栈帧, 栈帧 可供函数存储局部变量, 或调用其他函数。
- □当函数执行结束后,该函数栈帧会被释放, ESP、EBP会重新赋值为上一栈帧栈顶、栈底 位置。
- □栈帧在内存中会不断的分配、释放







- @栈的生成与销毁
 - □从一个简单的程序,从汇编语言层面观察栈帧 的生成与销毁
 - □使用ida pro工具对编译好的可执行文件进行反 汇编
 - □看一下关于main函数和foo函数的汇编代码





```
#include<stdio.h>
Int foo(int a){
    printf("argv = %d",a);
    return0;
Int main(){
    int a;
   printf("hello stack \n");
    a=1;
    foo(a);
    return();
                          网安学院 崔宝江
                       北邮
```

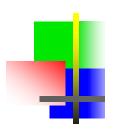


子函数调用过程

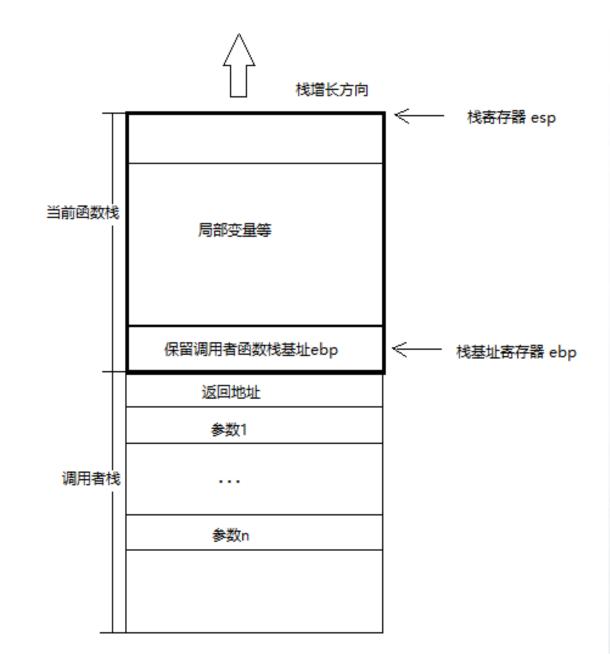
- 当函数调用发生时,新的栈帧被压入栈中,而当函数返回时,栈帧将从栈中弹出。
- @ 第一步: Push 参数1 参数入栈。
 - □ 母函数调用子函数时,在母函数栈帧再往内存低地址方向的邻接 区域,创建子函数的栈帧,首先把参数压入栈中;
- @ 第二步: call 子函数地址
 - □返回地址入栈,将指令寄存器eip中保存的下一条执行指令的地址 做为返回母函数的返回地址压入栈,当子函数调用结束后返回时 ,程序应该按照返回地址跳转到母函数的下一条指令继续执行;
 - □代码区跳转,处理器从当前母函数的代码区跳转到子函数程序的 入口,即程序的控制权转移到被调用的子函数;







低地址





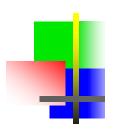


子函数调用过程

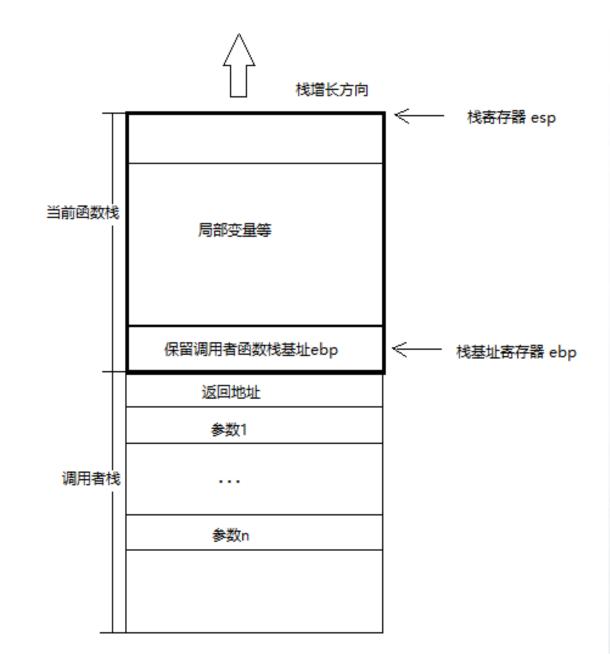
- ◎ 第三步: push ebp ;
 - □ ebp中母函数栈帧基址指针入栈,子函数将基址寄存器ebp中保存的母函数帧栈的基地址指针压入栈中保存;
- 鄭四步: mov ebp esp;
 - □ esp值装入ebp, ebp更新为新栈帧基地址。把esp中保存的最新 栈顶指针拷贝到基址寄存器ebp中,这时ebp中保存的就是正在被 调用子函数的基地址,即子函数的栈帧底部地址;
- ◎ 第五步: sub esp xxx ;
 - □ 给新栈帧分配空间,根据函数需要保存局部变量的空间大小,栈 顶指针esp从子函数的基地址向内存低地址偏移,为局部变量留 出一定空间







低地址









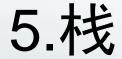
- □栈顶指针esp恢复到栈底位置,也就是将分配的栈空间全部释放,将分配的栈空间全部释放
- @第七步: pop ebp
 - □ 将ESP会指向的栈帧中母函数栈帧基址指针 弹出保存在ebp
- ◎第八部: ret
 - □按照返回地址指向的位置,返回main函数下一 步指令





- □foo函数的调用步骤如下:
 - ○1.在.text 40109c处,执行了mov eax, [ebp+var_4]指令,ebp+var_4位置的偏移是变量a 在栈帧中的位置,继续执行了push eax指令,导致 a被压入栈中,ESP向低地址增长4字节。
 - ○2.在.text 4010a0执行了call foo指令,call指令会将调用函数结束的返回地址,即函数中下一条指令add esp,4的地址0x4010a5压入栈中,程序进入foo函数中执行。





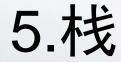
@ foo函数的调用步骤如下:

- ○3.进入foo函数后,首先执行的是push ebp指令,该指令的目的是保存上一函数栈的栈基址位置,便于函数结束后恢复上一函数栈。指令执行后,EBP中的数据被压入栈中,ESP向低地址增长4字节。
- ○4.在.text 401021执行了mov ebp, esp指令,该指令将EBP寄存器的值改写为当前ESP寄存器的值,导致栈底上移,新栈开始分配。
- ○5.在.text 401023执行了sub esp, 40h的指令,该指令均导致ESP寄存器向低地址增长了64字节,也就是说栈的长度为64字节,新栈分配到此结束。



- @ foo函数的调用步骤如下:
 - 〇6.从.text 401023至.text 401053为函数调用了 printf函数,是函数正常的执行逻辑及编译器添加的检查函数。
 - ○7.在.text 401058处开始执行了mov esp, ebp和 pop ebp两条操作,在某些编译器中这两条也被整合成为leave指令。
 - ❖首先mov esp, ebp指令将栈顶指针esp恢复到栈底位置 ,也就是将分配的栈空间全部释放。
 - ❖再执行pop ebp指令,由之前的步骤可知,当前ESP寄存器指向空间的内容是main函数栈的栈基址位置,指令结束后,ESP会指向返回地址位置,EBP寄存器会恢复到main函数函数栈的栈底位置№邮 网安学院 崔宝江





@ foo函数的调用步骤如下:

8.在.text 40105b处执行了ret指令,该指令内容可以理解为pop eip,执行后程序跳转回main函数,并且栈帧也恢复回到main函数执行call foo之前的状态。







- @ foo函数的调用步骤
 - @ IDA的示例
 - @ OllyDbg的动态调试过程





@foo函数的调用步骤如下:

```
; CODE XREF: _mainîj
.text:00401070 main 0
                                proc near
.text:00401070
.text:00401070 var 44
                                = byte ptr -44h
.text:00401070 var 4
                                = dword ptr -4
.text:00401070
.text:00401070
                                        ebp
                                push
.text:00401071
                                mov
                                        ebp, esp
.text:00401073
                                sub
                                        esp, 44h
.text:00401076
                                push
                                        ebx
.text:00401077
                                push
                                        esi
.text:00401078
                                        edi
                                push
.text:00401079
                                lea
                                        edi, [ebp+var 44]
.text:0040107C
                                mov
                                        ecx, 11h
.text:00401081
                                        eax, OCCCCCCCCh
                                mov
.text:00401086
                                rep stosd
                                        offset aHelloStack; "hello stack \n"
.text:00401088
                                push
.text:0040108D
                                call
                                         printf
                                        esp, 4
.text:00401092
                                add
.text:00401095
                                         [ebp+var_4], 1
                                MOV
.text:0040109C
                                        eax, [ebp+var_4]
                                MOV
.text:0040109F
                                push
                                        eax
                                        sub_401005
                                call
.text:004010A0
.text:004010A5
                                add
                                        esp, 4
.text:004010A8
                                        eax, eax
                                xor
.text:004010AA
                                        edi
                                pop
                                        esi
.text:004010AB
                                pop
.text:004010AC
                                        ebx
                                pop
.text:004010AD
                                add
                                        esp, 44h
.text:004010B0
                                CMP
                                        ebp, esp
.text:004010B2
                                call
                                         chkesp
.text:004010B7
                                        esp, ebp
                                MOV
.text:004010B9
                                pop
                                        ebp
.text:004010BA
                                retn
.text:004010BA main 0
                                endp
```



崔宝江



@foo函数的调用步骤如下:

```
.text:00401020 ; Attributes: bp-based frame
.text:00401020
                                                       ; CODE XREF: sub_4010051j
.text:00401020 sub_401020
                               proc near
.text:00401020
= byte ptr -40h
.text:00401020 arg 0
                               = dword ptr 8
.text:00401020
.text:00401020
                               push
                                      ebp
.text:00401021
                                       ebp, esp
                               mov
                                      esp, 40h
.text:00401023
                               sub
.text:00401026
                                      ebx
                               push
.text:00401027
                                      esi
                               push
.text:00401028
                               push
                                      edi
.text:00401029
                               1ea
                                      edi, [ebp+var_40]
.text:0040102C
                               mov
                                       ecx, 10h
                                      eax, OCCCCCCCCh
.text:00401031
                               MOV
.text:00401036
                               rep stosd
                                      eax, [ebp+arg_0]
.text:00401038
                               MOV
.text:0040103B
                               push
                                      eax
                                                      ; "argv = %d"
.text:0040103C
                               push
                                       offset aArqvD
                               call
                                      printf
.text:00401041
.text:00401046
                               add
                                      esp, 8
.text:00401049
                                       eax, eax
                               xor
                                      edi
.text:0040104B
                               pop
.text:0040104C
                                      esi
                               pop
.text:0040104D
                                      ebx
                               pop
.text:0040104E
                               add
                                      esp, 40h
                                       ebp, esp
.text:00401051
                               CMP
.text:00401053
                               call
                                      chkesp
.text:00401058
                               MOV
                                      esp, ebp
.text:0040105A
                               pop
                                       ebp
.text:0040105B
                               retn
                               endp
```







- @foo函数的调用步骤如下:
 - @ OllyDbg的动态调试过程





- □1.寄存器
- □2.x86指令集
- □3.寻址方式
- □4.字节序
- □5.栈
- □6.函数调用约定

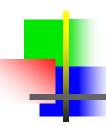




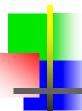
- @函数调用约定
 - □对函数调用时如何传递参数的一种约定
 - ○例如:使用VC++ 6.0对源代码进行编译,在函数定义时可以选择函数的调用约定
 - □主要解决的是问题就是函数的参数是如何传递的,以及函数执行结束后,参数应该如何处理
 - □主要的函数约定由三种
 - **OCdecl**
 - **OStdcall**
 - **O**Fastcall







```
#include<stdio.h>
void cdecl foo cdecl(int a,int b,int c) {
     printf("argv %d %d %d \n",a,b,c);
void stdcall foo stdcall(int a,int b,int c) {
     printf("argv %d %d %d \n",a,b,c);
void fastcall foo fastcall(int a,int b,int c,int d) {
     printf("argv %d %d %d %d \n",a,b,c,d);
intmain(){
     foo cdecl(1,2,3);
     foo stdcall(1,2,3);
     foo fastcall(1,2,3,4);
     return0;
                              北邮
                                  网安学院 崔宝江
```



□cdcel是C/C++默认方式,参数从右向左入栈

```
.text:00401150
                                 push
                                         ebp
.text:00401151
                                 mov
                                         ebp, esp
.text:00401153
                                 sub
                                         esp,
                                              40h
.text:00401156
                                         ebx
                                 push
.text:00401157
                                 push
                                         esi
.text:00401158
                                         edi
                                 push
.text:00401159
                                 lea.
                                         edi, [ebp+var 40]
.text:0040115C
                                 mov
                                         ecx, 10h
.text:00401161
                                         eax, OCCCCCCCCh
                                 mov
tovt • 881/81166
                                 ran ctacd
.text:00401168
                                 push
                                         3
text:0040116A
                                         2
                                 push
text:0040116C
                                 push
text:0040116E
                                         i foo cdecl
                                 call
.text:00401173
                                 add
                                         esp, OCh
.text:00401176
                                 push
                                         3
.text:00401178
                                         2
                                 push
.text:0040117A
                                 push
                                         j foo stdcall@12 ; foo stdcall(x,x,x)
.text:0040117C
                                 call
.text:00401181
                                 push
.text:00401183
                                 push
.text:00401185
                                 mov
                                         edx, 2
.text:0040118A
                                 mov
.text:0040118F
                                         i @foo fastcall@12 ; foo fastcall(x,x,x)
                                 call
.text:00401194
                                 xor
                                         eax, eax
.text:00401196
                                 pop
                                         edi
.text:00401197
                                         esi
                                 pop
.text:00401198
                                         ebx
                                 pop
```

○查看main函数汇编代码





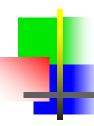
□参数入栈顺序

- ○参数是被压入栈中传递的,并且压入栈中的顺序是 从右向左,即从最后一个参数开始压栈
- 〇此时相比未压栈时,ESP指针向减小了3*4=12(0x0c)。

□参数清除

○当执行过callfoo_cdecl后,ESP指针是会恢复到压入参数后的位置,下一条执行add esp, 0x0c就会把ESP指针增加12字节,也就恢复到了压入参数之前的位置,从而实现了参数清除





```
.text:00401030 arq 0
                               = dword ptr
.text:00401030 arg_4
                               = dword ptr
                                             0Ch
.text:00401030 arg 8
                               = dword ptr
                                             10h
.text:00401030
.text:00401030
                               push
                                        ebp
.text:00401031
                               mov
                                        ebp, esp
                                        esp, 40h
.text:00401033
                               sub
.text:00401036
                               push
                                        ebx
                                        esi
.text:00401037
                               push
                                        edi
.text:00401038
                               push
.text:00401039
                               1ea
                                        edi, [ebp+var_40]
.text:0040103C
                               mov
                                        ecx, 10h
.text:00401041
                               mov
                                        eax, OCCCCCCCCh
.text:00401046
                               rep stosd
.text:00401048
                               mov
                                        eax, [ebp+arg_8]
.text:0040104B
                               push
.text:0040104C
                               mov
                                        ecx, [ebp+arq 4]
.text:0040104F
                               push
                                        ecx
.text:00401050
                               mov
                                        edx, [ebp+arg 0]
.text:00401053
                               push
                                        edx
.text:00401054
                               push
                                        offset Format
                                                       ; "arqv %d %d %d \n"
.text:00401059
                               call
                                        printf
.text:0040105E
                               add
                                        esp, 10h
.text:00401061
                                        edi
                               pop
.text:00401062
                                        esi
                               pop
.text:00401063
                               pop
                                        ebx
.text:00401064
                                        esp, 40h
                               add
.text:00401067
                               CMP
                                        ebp, esp
                                        __chkesp
.text:00401069
                               call
.text:0040106E
                               mov
                                        esp, ebp
.text:00401070
                                        ebp
                               pop
.text:00401071
                               retn
endp
```

foo_cdecl反汇编代码中





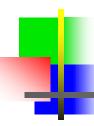
- □在foo_cdecl反汇编代码中,可以看到函数是利用的是EBP寄存器与参数位置的相对关系顺序传递参数的
- □参数自右向左传递的目的也是保证生成汇编语言时,这些参数相对于EBP指向的栈位置的偏移量是固定的。





- □stdcall调用者未对传入参数进行清理
 - Ostdcall是windows API默认方式,参数从右向左入 栈
 - ○通过对比在main函数foo_stdcall和foo_cdecl两种的函数调用代码,发现调用foo_stdcall相比foo_cdecl,缺少了add esp, 0ch指令,也就是说调用者未对传入参数进行清理



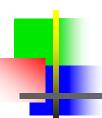


```
.text:00401150
                                push
                                         ebp
.text:00401151
                                mov
                                         ebp, esp
.text:00401153
                                sub
                                         esp, 40h
.text:00401156
                                         ebx
                                push
.text:00401157
                                push
                                         esi
.text:00401158
                                         edi
                                push
.text:00401159
                                lea.
                                         edi, [ebp+var 40]
.text:0040115C
                                mov
                                         ecx, 10h
.text:00401161
                                         eax, OCCCCCCCCh
                                mov
tovt • 881/81166
                                ran ctacd
text:00401168
                                push
                                         3
text:0040116A
                                push
text:0040116C
                                push
text:0040116E
                                         j__foo_cdecl
                                call
.text:00401173
                                add
                                         esp, OCh
.text:00401176
                                push
                                         3
.text:00401178
                                         2
                                push
.text:0040117A
                                push
                                         j foo stdcall@12 ; foo stdcall(x,x,x)
.text:0040117C
                                call
.text:00401181
                                push
.text:00401183
                                push
.text:00401185
                                mov
                                         edx, 2
.text:0040118A
                                mov
.text:0040118F
                                         j_@foo_fastcall@12 ; foo_fastcall(x,x,x)
                                call
.text:00401194
                                xor
                                         eax, eax
.text:00401196
                                pop
                                         edi
.text:00401197
                                         esi
                                pop
.text:00401198
                                pop
                                         ebx
```

○查看main函数汇编代码







@foo stdcall反汇编代码

```
.text:00401090
                               push
                                       ebp
.text:00401091
                               mov
                                       ebp, esp
.text:00401093
                               sub
                                       esp, 40h
.text:00401096
                               push
                                       ebx
                                       esi
.text:00401097
                               push
.text:00401098
                               push
                                       edi
.text:00401099
                                       edi, [ebp+var 40]
                               lea.
.text:0040109C
                               MOV
                                       ecx, 10h
.text:004010A1
                               mov
                                       eax, OCCCCCCCCh
.text:004010A6
                               rep stosd
.text:004010A8
                               mov
                                       eax, [ebp+arg 8]
.text:004010AB
                               push
                                       eax
.text:004010AC
                                       ecx, [ebp+arg 4]
                               mov
.text:004010AF
                               push
                                       ecx
.text:004010B0
                                       edx, [ebp+arq 0]
                               mov
.text:004010B3
                               push
                                       edx
                                       offset Format
                                                       ; "argv %d %d %d \n"
.text:004010B4
                               push
.text:004010B9
                               call
                                       printf
.text:004010BE
                               add
                                       esp, 10h
.text:004010C1
                                       edi
                               pop
.text:004010C2
                                       esi
                               pop
.text:004010C3
                                       ebx
                               pop
.text:004010C4
                               add
                                       esp, 40h
.text:004010C7
                               CMP
                                       ebp, esp
.text:004010C9
                               call
                                        chkesp
.text:004010CE
                               mov
                                       esp, ebp
text.004010D0
                                        ehn
                               non
.text:004010D1
                                        0Ch
                               retn
```





- □foo_stdcall采用的是retn 0ch的指令,而非retn。
 - ○该指令的作用是retn + pop 0x0c字节,即返回后使 ESP增加12个字节,这与foo_cdecl的指令执行是 一致的。
 - Ostdcall方式的优点在于,被调用者函数内部存在清理参数代码,与调用函数后再执行add esp, xxx相比,代码尺寸要小,是Win 32 API库使用的函数调用约定。

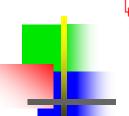




□fastcall方式

- 〇与stdcall方式基本类似,参数的清理也由被调用函数来负责
- ○但该方式通常会使用寄存器(而非栈内存)去传递参数。
 - ❖若函数需要传递多个参数,会优先使用ECX和EDX来传递 后两个参数,其余的参数再从右向左由栈传入。





```
.text:00401150
                                          ebp
.text:00401150
                                 push
                                          ebp, esp
.text:00401151
                                 mov
                                          esp, 40h
.text:00401153
                                 sub
.text:00401156
                                 push
                                          ebx
                                 push
                                          esi
.text:00401157
.text:00401158
                                 push
                                          edi
                                          edi, [ebp+var 40]
.text:00401159
                                 lea.
                                          ecx, 10h
.text:0040115C
                                 mov
.text:00401161
                                 mov
                                          eax, OCCCCCCCCh
.text:00401166
                                 rep stosd
                                          3
.text:00401168
                                 push
                                          2
.text:0040116A
                                 push
.text:0040116C
                                 push
.text:0040116E
                                 call
                                          sub_401005
.text:00401173
                                 add
                                          esp, OCh
.text:00401176
                                 push
                                          3
                                          2
.text:00401178
                                 push
                                          1
.text:0040117A
                                 push
.text:0040117C
                                 call
                                          sub 40100F
                                          4
.text:00401181
                                 push
                                          3
.text:00401183
                                 push
                                          edx, 2
.text:00401185
                                 mov
                                          ecx, 1
.text:0040118A
                                 mov
.text:0040118F
                                          sub 401019
                                 call
.text:00401194
                                 xor
                                          eax, eax
                                          edi
.text:00401196
                                 pop
                                          esi
.text:00401197
                                 pop
.text:00401198
                                 pop
                                          ebx
.text:00401199
                                 add
                                          esp, 40h
.text:0040119C
                                 CMP
                                          ebp, esp
.text:0040119E
                                 call
                                          sub 401240
                                          esp, ebp
.text:004011A3
                                 mov
.text:004011A5
                                 pop
                                          ebp
                                 retn
.text:004011A6
.text:004011A6 sub 401150
                                 endp
.text:004011A6
```

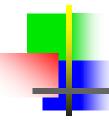




□fastcall方式是很快的

- ○由于CPU对寄存器的访问要快于对栈所在内存的访问速度,因此从函数调用本身来看,fastcall方式是很快的
- ○但有时需要额外的系统开销来管理寄存器,如在调用函数前ECX、EDX中存有重要数据,那么需要先进行备份
- 此外,如果函数本身需要使用这两个寄存器,也需要将参数迁移到其他位置进行使用。





Q & A

