[原创]常见函数调用约定(x86、x64、arm、arm64) 优

有影

大侠

2018-2-8 22:42 18156

> 我学习逆向，整理的一些常见的函数调用约定反汇编笔记。由于我是新手，肯定有一些疏漏不完善的，我遇到了会实时更新的。

更新时间：2018年3月7日

X86 函数调用约定

> X86 有三种常用调用约定，cdecl(C规范)/stdcall(WinAPI默认)/fastcall 函数调用约定。

cdecl 函数调用约定

> 参数从右往左依次入栈，调用者实现栈平衡，返回值存放在 EAX 中。

```
20:       int cdecl_sum = cdecl_add(1, 2, 3, 4, 5, 6, 7);
00401138   push        7
0040113A   push        6
0040113C   push        5
0040113E   push        4
00401140   push        3
00401142   push        2
00401144   push        1
00401146   call        @ILT+5(_cdecl_add) (0040100a)
0040114B   add         esp,1Ch  # 栈平衡
0040114E   mov         dword ptr [ebp-4],eax    # 返回值

3:    int __cdecl cdecl_add(int a, int b, int c, int d, int e, int f, int g)
4:    {
00401030   push        ebp
00401031   mov         ebp,esp
00401033   sub         esp,44h
00401036   push        ebx
00401037   push        esi
00401038   push        edi
00401039   lea         edi,[ebp-44h]
0040103C   mov         ecx,11h
00401041   mov         eax,0CCCCCCCCh
00401046   rep stos    dword ptr [edi]
5:        int sum = a+b+c+d+e+f+g;
00401048   mov         eax,dword ptr [ebp+8]
0040104B   add         eax,dword ptr [ebp+0Ch]
0040104E   add         eax,dword ptr [ebp+10h]
00401051   add         eax,dword ptr [ebp+14h]
00401054   add         eax,dword ptr [ebp+18h]
00401057   add         eax,dword ptr [ebp+1Ch]
0040105A   add         eax,dword ptr [ebp+20h]
0040105D   mov         dword ptr [ebp-4],eax
6:        return sum;
00401060   mov         eax,dword ptr [ebp-4]    # 存放返回值
7:    }
00401063   pop         edi
00401064   pop         esi
00401065   pop         ebx
00401066   mov         esp,ebp
00401068   pop         ebp
00401069   ret
```

stdcall 函数调用约定

> 参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 EAX 中。

```
21:       int stdcall_sum = stdcall_add(1, 2, 3, 4, 5, 6, 7);
00401151   push        7
00401153   push        6
00401155   push        5
00401157   push        4
00401159   push        3
0040115B   push        2
0040115D   push        1
0040115F   call        @ILT+15(_stdcall_add@28) (00401014)
00401164   mov         dword ptr [ebp-8],eax    # 返回值

9:    int __stdcall stdcall_add(int a, int b, int c, int d, int e, int f, int g)
10:   {
00401080   push        ebp
00401081   mov         ebp,esp
00401083   sub         esp,44h
00401086   push        ebx
00401087   push        esi
00401088   push        edi
00401089   lea         edi,[ebp-44h]
0040108C   mov         ecx,11h
00401091   mov         eax,0CCCCCCCCh
00401096   rep stos    dword ptr [edi]
11:       int sum = a+b+c+d+e+f+g;
00401098   mov         eax,dword ptr [ebp+8]
0040109B   add         eax,dword ptr [ebp+0Ch]
0040109E   add         eax,dword ptr [ebp+10h]
004010A1   add         eax,dword ptr [ebp+14h]
004010A4   add         eax,dword ptr [ebp+18h]
004010A7   add         eax,dword ptr [ebp+1Ch]
004010AA   add         eax,dword ptr [ebp+20h]
004010AD   mov         dword ptr [ebp-4],eax
12:       return sum;
004010B0   mov         eax,dword ptr [ebp-4]    # 存放返回值
13:   }
004010B3   pop         edi
004010B4   pop         esi
004010B5   pop         ebx
004010B6   mov         esp,ebp
004010B8   pop         ebp
004010B9   ret         1Ch  # 栈平衡（等价于先 add esp, 1Ch 再 ret）
```

fastcall 函数调用约定

> 参数1、参数2分别保存在 ECX、EDX ，剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 EAX 中。

```
25:       int fastcall_sum = fastcall_add(1, 2, 3, 4, 5, 6, 7);
00401167   push        7
00401169   push        6
0040116B   push        5
0040116D   push        4
0040116F   push        3
00401171   mov         edx,2
00401176   mov         ecx,1
0040117B   call        @ILT+0(@fastcall_add@28) (00401005)
00401180   mov         dword ptr [ebp-0Ch],eax  # 返回值

15:   int __fastcall fastcall_add(int a, int b, int c, int d, int e, int f, int g)
16:   {
004010D0   push        ebp
004010D1   mov         ebp,esp
004010D3   sub         esp,4Ch
004010D6   push        ebx
004010D7   push        esi
004010D8   push        edi
004010D9   push        ecx
004010DA   lea         edi,[ebp-4Ch]
004010DD   mov         ecx,13h
004010E2   mov         eax,0CCCCCCCCh
004010E7   rep stos    dword ptr [edi]
004010E9   pop         ecx
004010EA   mov         dword ptr [ebp-8],edx
004010ED   mov         dword ptr [ebp-4],ecx
17:       int sum = a+b+c+d+e+f+g;
004010F0   mov         eax,dword ptr [ebp-4]
004010F3   add         eax,dword ptr [ebp-8]
004010F6   add         eax,dword ptr [ebp+8]
004010F9   add         eax,dword ptr [ebp+0Ch]
004010FC   add         eax,dword ptr [ebp+10h]
004010FF   add         eax,dword ptr [ebp+14h]
00401102   add         eax,dword ptr [ebp+18h]
00401105   mov         dword ptr [ebp-0Ch],eax
18:       return sum;
00401108   mov         eax,dword ptr [ebp-0Ch]  # 存放返回值
19:   }
0040110B   pop         edi
0040110C   pop         esi
0040110D   pop         ebx
0040110E   mov         esp,ebp
00401110   pop         ebp
00401111   ret         14h  # 栈平衡（等价于先 add esp, 14h 再 ret）
```

X64 函数调用约定

> X64只有一种 fastcall 函数调用约定

fastcall 函数调用约定

> 参数1、参数2、参数3、参数4分别保存在 RCX、RDX、R8D、R9D ，剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 RAX 中。

```
# 该代码是 msvc 2017 x64 生成的汇编代码
    int fastcall_sum = fastcall_add(1, 2, 3, 4, 5, 6, 7);
00007FF6577A366E  mov         dword ptr [rsp+30h],7
00007FF6577A3676  mov         dword ptr [rsp+28h],6
00007FF6577A367E  mov         dword ptr [rsp+20h],5
00007FF6577A3686  mov         r9d,4
00007FF6577A368C  mov         r8d,3
00007FF6577A3692  mov         edx,2
00007FF6577A3697  mov         ecx,1
00007FF6577A369C  call        fastcall_add (07FF6577A11C2h)
00007FF6577A36A1  mov         dword ptr [fastcall_sum],eax  # 返回值

int __fastcall fastcall_add(int a, int b, int c, int d, int e, int f, int g)
{
00007FF6D22D1790  mov         dword ptr [rsp+20h],r9d
00007FF6D22D1795  mov         dword ptr [rsp+18h],r8d
00007FF6D22D179A  mov         dword ptr [rsp+10h],edx
00007FF6D22D179E  mov         dword ptr [rsp+8],ecx
00007FF6D22D17A2  push        rbp
00007FF6D22D17A3  push        rdi
00007FF6D22D17A4  sub         rsp,0E8h
00007FF6D22D17AB  mov         rbp,rsp
00007FF6D22D17AE  mov         rdi,rsp
00007FF6D22D17B1  mov         ecx,3Ah
00007FF6D22D17B6  mov         eax,0CCCCCCCCh
00007FF6D22D17BB  rep stos    dword ptr [rdi]
00007FF6D22D17BD  mov         ecx,dword ptr [rsp+108h]
    int sum = a + b + c + d + e + f + g;
00007FF6D22D17C4  mov         eax,dword ptr [b]
00007FF6D22D17CA  mov         ecx,dword ptr [a]
00007FF6D22D17D0  add         ecx,eax
00007FF6D22D17D2  mov         eax,ecx
00007FF6D22D17D4  add         eax,dword ptr [c]
00007FF6D22D17DA  add         eax,dword ptr [d]
00007FF6D22D17E0  add         eax,dword ptr [e]
00007FF6D22D17E6  add         eax,dword ptr [f]
00007FF6D22D17EC  add         eax,dword ptr [g]
00007FF6D22D17F2  mov         dword ptr [sum],eax
    return sum;
00007FF6D22D17F5  mov         eax,dword ptr [sum]       # 存放返回值
}
00007FF6D22D17F8  lea         rsp,[rbp+0E8h]
00007FF6D22D17FF  pop         rdi
00007FF6D22D1800  pop         rbp
00007FF6D22D1801  ret                                   # 没做栈平衡
```

ARM/ARM64 函数调用约定

ARM和ARM64使用的是ATPCS(ARM-Thumb Procedure Call Standard/ARM-Thumb过程调用标准)的函数调用约定。

ATPCS 函数调用约定

ARM

参数1~参数4 分别保存到 R0~R3 寄存器中， 剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 R0 中。

```
; 该代码是 arm-linux-androideabi-gcc + IDA PRO 生成的反汇编代码
.text:00008438                 MOV             R3, #5
.text:0000843C                 STR             R3, [SP]
.text:00008440                 MOV             R3, #6
.text:00008444                 STR             R3, [SP,#4]
.text:00008448                 MOV             R3, #7
.text:0000844C                 STR             R3, [SP,#8]
.text:00008450                 MOV             R3, #8
.text:00008454                 STR             R3, [SP,#12]
.text:00008458                 MOV             R3, #9
.text:0000845C                 STR             R3, [SP,#16]
.text:00008460                 MOV             R3, #10
.text:00008464                 STR             R3, [SP,#20]
.text:00008468                 MOV             R0, #1
.text:0000846C                 MOV             R1, #2
.text:00008470                 MOV             R2, #3
.text:00008474                 MOV             R3, #4
.text:00008478                 BL              add
.text:0000847C                 STR             R0, [R11,#-8]

.text:000083C4                 EXPORT add
.text:000083C4
.text:000083C4                 STR             R11, [SP,#-4]!
.text:000083C8                 ADD             R11, SP, #0
.text:000083CC                 SUB             SP, SP, #0x1C
.text:000083D0                 STR             R0, [R11,#-16]
.text:000083D4                 STR             R1, [R11,#-20]
.text:000083D8                 STR             R2, [R11,#-24]
.text:000083DC                 STR             R3, [R11,#-28]
.text:000083E0                 LDR             R2, [R11,#-16]
.text:000083E4                 LDR             R3, [R11,#-20]
.text:000083E8                 ADD             R2, R2, R3
.text:000083EC                 LDR             R3, [R11,#-24]
.text:000083F0                 ADD             R2, R2, R3
.text:000083F4                 LDR             R3, [R11,#-28]
.text:000083F8                 ADD             R2, R2, R3
.text:000083FC                 LDR             R3, [R11,#4]
.text:00008400                 ADD             R2, R2, R3
.text:00008404                 LDR             R3, [R11,#8]
.text:00008408                 ADD             R2, R2, R3
.text:0000840C                 LDR             R3, [R11,#12]
.text:00008410                 ADD             R3, R2, R3
.text:00008414                 STR             R3, [R11,#-8]
.text:00008418                 LDR             R3, [R11,#-8]
.text:0000841C                 MOV             R0, R3                  # 返回值
.text:00008420                 SUB             SP, R11, #0
.text:00008424                 LDR             R11, [SP],#4
.text:00008428                 BX              LR
```

ARM64

> 参数1~参数8 分别保存到 X0~X7 寄存器中， 剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 X0 中。

```
; 该代码是 aarch64-linux-android-gcc + IDA PRO 生成的反汇编代码
.text:000000000040065C                 MOV             W0, #9
.text:0000000000400660                 STR             W0, [SP]
.text:0000000000400664                 MOV             W0, #10
.text:0000000000400668                 STR             W0, [SP,#8]
.text:000000000040066C                 MOV             W0, #1
.text:0000000000400670                 MOV             W1, #2
.text:0000000000400674                 MOV             W2, #3
.text:0000000000400678                 MOV             W3, #4
.text:000000000040067C                 MOV             W4, #5
.text:0000000000400680                 MOV             W5, #6
.text:0000000000400684                 MOV             W6, #7
.text:0000000000400688                 MOV             W7, #8
.text:000000000040068C                 BL              add
.text:0000000000400690                 STR             W0, [X29,#28]

.text:00000000004005E8                 EXPORT add
.text:00000000004005E8
.text:00000000004005E8                 SUB             SP, SP, #0x30
.text:00000000004005EC                 STR             W0, [SP,#28]
.text:00000000004005F0                 STR             W1, [SP,#24]
.text:00000000004005F4                 STR             W2, [SP,#20]
.text:00000000004005F8                 STR             W3, [SP,#16]
.text:00000000004005FC                 STR             W4, [SP,#12]
.text:0000000000400600                 STR             W5, [SP,#8]
.text:0000000000400604                 STR             W6, [SP,#4]
.text:0000000000400608                 STR             W7, [SP]
.text:000000000040060C                 LDR             W1, [SP,#28]
.text:0000000000400610                 LDR             W0, [SP,#24]
.text:0000000000400614                 ADD             W1, W1, W0
.text:0000000000400618                 LDR             W0, [SP,#20]
.text:000000000040061C                 ADD             W1, W1, W0
.text:0000000000400620                 LDR             W0, [SP,#16]
.text:0000000000400624                 ADD             W1, W1, W0
.text:0000000000400628                 LDR             W0, [SP,#12]
.text:000000000040062C                 ADD             W1, W1, W0
.text:0000000000400630                 LDR             W0, [SP,#8]
.text:0000000000400634                 ADD             W1, W1, W0
.text:0000000000400638                 LDR             W0, [SP,#4]
.text:000000000040063C                 ADD             W0, W1, W0
.text:0000000000400640                 STR             W0, [SP,#44]
.text:0000000000400644                 LDR             W0, [SP,#44]         # 返回值
.text:0000000000400648                 ADD             SP, SP, #0x30
.text:000000000040064C                 RET
```

C++ 函数调用约定

> thiscall用于C++中类成员函数（方法）的调用

thiscall 函数调用约定

x86

> 参数从右往左依次入栈，this指针存放ECX中，被调用者实现栈平衡，返回值存放在 EAX 中。

```
16:       int sum = calc.thiscall_add(1, 2, 3, 4, 5, 6, 7);
00401098   push        7
0040109A   push        6
0040109C   push        5
0040109E   push        4
004010A0   push        3
004010A2   push        2
004010A4   push        1
004010A6   lea         ecx,[ebp-4]                # this指针
004010A9   call        @ILT+0(Calc::thiscall_add) (00401005)
004010AE   mov         dword ptr [ebp-8],eax    # 返回值

7:    int Calc::thiscall_add(int a, int b, int c, int d, int e, int f, int g)
8:    {
00401020   push        ebp
00401021   mov         ebp,esp
00401023   sub         esp,48h
00401026   push        ebx
00401027   push        esi
00401028   push        edi
00401029   push        ecx
0040102A   lea         edi,[ebp-48h]
0040102D   mov         ecx,12h
00401032   mov         eax,0CCCCCCCCh
00401037   rep stos    dword ptr [edi]
00401039   pop         ecx
0040103A   mov         dword ptr [ebp-4],ecx
9:        int sum = a + b + c + d + e + f + g;
0040103D   mov         eax,dword ptr [ebp+8]
00401040   add         eax,dword ptr [ebp+0Ch]
00401043   add         eax,dword ptr [ebp+10h]
00401046   add         eax,dword ptr [ebp+14h]
00401049   add         eax,dword ptr [ebp+18h]
0040104C   add         eax,dword ptr [ebp+1Ch]
0040104F   add         eax,dword ptr [ebp+20h]
00401052   mov         dword ptr [ebp-8],eax
10:       return sum;
00401055   mov         eax,dword ptr [ebp-8]    # 存放返回值
11:   }
00401058   pop         edi
00401059   pop         esi
0040105A   pop         ebx
0040105B   mov         esp,ebp
0040105D   pop         ebp
0040105E   ret         1Ch        # 栈平衡（等价于先 add esp, 1Ch 再 ret）
```

X64

参数1、参数2、参数3分别保存在RDX、R8D、R9D中，this指针存放RCX中，剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 RAX 中。

```
# 该代码是 msvc 2017 x64 生成的汇编代码
    int sum = calc.thiscall_add(1, 2, 3, 4, 5, 6, 7);
00007FF602E6190F  mov         dword ptr [rsp+38h],7
00007FF602E61917  mov         dword ptr [rsp+30h],6
00007FF602E6191F  mov         dword ptr [rsp+28h],5
00007FF602E61927  mov         dword ptr [rsp+20h],4
00007FF602E6192F  mov         r9d,3
00007FF602E61935  mov         r8d,2
00007FF602E6193B  mov         edx,1
00007FF602E61940  lea         rcx,[calc]              # this指针
00007FF602E61944  call        Calc::thiscall_add (07FF602E610A0h)
00007FF602E61949  mov         dword ptr [sum],eax    # 返回值

int Calc::thiscall_add(int a, int b, int c, int d, int e, int f, int g)
{
00007FF602E61770  mov         dword ptr [rsp+20h],r9d
00007FF602E61775  mov         dword ptr [rsp+18h],r8d
00007FF602E6177A  mov         dword ptr [rsp+10h],edx
00007FF602E6177E  mov         qword ptr [rsp+8],rcx
00007FF602E61783  push        rbp
00007FF602E61784  push        rdi
00007FF602E61785  sub         rsp,0E8h
00007FF602E6178C  mov         rbp,rsp
00007FF602E6178F  mov         rdi,rsp
00007FF602E61792  mov         ecx,3Ah
00007FF602E61797  mov         eax,0CCCCCCCCh
00007FF602E6179C  rep stos    dword ptr [rdi]
00007FF602E6179E  mov         rcx,qword ptr [rsp+108h]
    int sum = a + b + c + d + e + f + g;
00007FF602E617A6  mov         eax,dword ptr [b]
00007FF602E617AC  mov         ecx,dword ptr [a]
00007FF602E617B2  add         ecx,eax
00007FF602E617B4  mov         eax,ecx
00007FF602E617B6  add         eax,dword ptr [c]
00007FF602E617BC  add         eax,dword ptr [d]
00007FF602E617C2  add         eax,dword ptr [e]
00007FF602E617C8  add         eax,dword ptr [f]
00007FF602E617CE  add         eax,dword ptr [g]
00007FF602E617D4  mov         dword ptr [sum],eax
    return sum;
00007FF602E617D7  mov         eax,dword ptr [sum]  # 存放返回值
}
00007FF602E617DA  lea         rsp,[rbp+0E8h]
00007FF602E617E1  pop         rdi
00007FF602E617E2  pop         rbp
00007FF602E617E3  ret                               # 没做栈平衡
```

ARM

> 参数1、参数2、参数3分别保存在R1、R2、R3中，this指针存放R0中，剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 R0 中。

```
; 该代码是 arm-linux-androideabi-gcc + IDA PRO 生成的反汇编代码
.text:000085BC                 MOV             R3, #4
.text:000085C0                 STR             R3, [SP] ; int
.text:000085C4                 MOV             R3, #5
.text:000085C8                 STR             R3, [SP,#4] ; int
.text:000085CC                 MOV             R3, #6
.text:000085D0                 STR             R3, [SP,#8] ; int
.text:000085D4                 MOV             R3, #7
.text:000085D8                 STR             R3, [SP,#12] ; int
.text:000085DC                 MOV             R3, #8
.text:000085E0                 STR             R3, [SP,#16] ; int
.text:000085E4                 MOV             R3, #9
.text:000085E8                 STR             R3, [SP,#20] ; int
.text:000085EC                 MOV             R3, #10
.text:000085F0                 STR             R3, [SP,#24] ; int
.text:000085F4                 MOV             R0, R2  ; this
.text:000085F8                 MOV             R1, #1  ; int
.text:000085FC                 MOV             R2, #2  ; int
.text:00008600                 MOV             R3, #3  ; int
.text:00008604                 BL              _ZN4Calc12thiscall_addEiiiiiiiiii ; Calc::thiscall_
.text:00008608                 MOV             R3, R0

.text:00008544                 EXPORT _ZN4Calc12thiscall_addEiiiiiiiiii
.text:00008544
.text:00008544                 STR             R11, [SP,#-4]!
.text:00008548                 ADD             R11, SP, #0
.text:0000854C                 SUB             SP, SP, #0x1C
.text:00008550                 STR             R0, [R11,#-16]
.text:00008554                 STR             R1, [R11,#-20]
.text:00008558                 STR             R2, [R11,#-24]
.text:0000855C                 STR             R3, [R11,#-28]
.text:00008560                 LDR             R2, [R11,#-20]
.text:00008564                 LDR             R3, [R11,#-24]
.text:00008568                 ADD             R2, R2, R3
.text:0000856C                 LDR             R3, [R11,#-28]
.text:00008570                 ADD             R2, R2, R3
.text:00008574                 LDR             R3, [R11,#4]
.text:00008578                 ADD             R2, R2, R3
.text:0000857C                 LDR             R3, [R11,#8]
.text:00008580                 ADD             R2, R2, R3
.text:00008584                 LDR             R3, [R11,#12]
.text:00008588                 ADD             R2, R2, R3
.text:0000858C                 LDR             R3, [R11,#16]
.text:00008590                 ADD             R3, R2, R3
.text:00008594                 STR             R3, [R11,#-8]
.text:00008598                 LDR             R3, [R11,#-8]
.text:0000859C                 MOV             R0, R3                 # 返回值
.text:000085A0                 SUB             SP, R11, #0
.text:000085A4                 LDR             R11, [SP],#4
.text:000085A8                 BX              LR
```

ARM64

> 参数1~参数7 分别保存到 X1~X7 寄存器中，this指针存放X0中，剩下的参数从右往左依次入栈，被调用者实现栈平衡，返回值存放在 X0 中。

首页 论坛 课程 招聘 发现

```
; 该代码是 aarch64-linux-android-gcc + IDA PRO 生成的反汇编代码
.text:00000000004006A0                 MOV             W0, #8
.text:00000000004006A4                 STR             W0, [SP] ; int
.text:00000000004006A8                 MOV             W0, #9
.text:00000000004006AC                 STR             W0, [SP,#8] ; int
.text:00000000004006B0                 MOV             W0, #10
.text:00000000004006B4                 STR             W0, [SP,#16] ; int
.text:00000000004006B8                 MOV             X0, X1  ; this
.text:00000000004006BC                 MOV             W1, #1  ; int
.text:00000000004006C0                 MOV             W2, #2  ; int
.text:00000000004006C4                 MOV             W3, #3  ; int
.text:00000000004006C8                 MOV             W4, #4  ; int
.text:00000000004006CC                 MOV             W5, #5  ; int
.text:00000000004006D0                 MOV             W6, #6  ; int
.text:00000000004006D4                 MOV             W7, #7  ; int
.text:00000000004006D8                 BL              _ZN4Calc12thiscall_addEiiiiiiiii ; Calc::t
.text:00000000004006DC                 STR             W0, [X29,#0x1C]

.text:0000000000400628                 EXPORT _ZN4Calc12thiscall_addEiiiiiiiii
.text:0000000000400628
.text:0000000000400628                 SUB             SP, SP, #0x40
.text:000000000040062C                 STR             X0, [SP,#40]
.text:0000000000400630                 STR             W1, [SP,#36]
.text:0000000000400634                 STR             W2, [SP,#32]
.text:0000000000400638                 STR             W3, [SP,#28]
.text:000000000040063C                 STR             W4, [SP,#24]
.text:0000000000400640                 STR             W5, [SP,#20]
.text:0000000000400644                 STR             W6, [SP,#16]
.text:0000000000400648                 STR             W7, [SP,#12]
.text:000000000040064C                 LDR             W1, [SP,#36]
.text:0000000000400650                 LDR             W0, [SP,#32]
.text:0000000000400654                 ADD             W1, W1, W0
.text:0000000000400658                 LDR             W0, [SP,#28]
.text:000000000040065C                 ADD             W1, W1, W0
.text:0000000000400660                 LDR             W0, [SP,#24]
.text:0000000000400664                 ADD             W1, W1, W0
.text:0000000000400668                 LDR             W0, [SP,#20]
.text:000000000040066C                 ADD             W1, W1, W0
.text:0000000000400670                 LDR             W0, [SP,#16]
.text:0000000000400674                 ADD             W1, W1, W0
.text:0000000000400678                 LDR             W0, [SP,#12]
.text:000000000040067C                 ADD             W0, W1, W0
.text:0000000000400680                 STR             W0, [SP,#60]
.text:0000000000400684                 LDR             W0, [SP,#60]         # 返回值
.text:0000000000400688                 ADD             SP, SP, #0x40
.text:000000000040068C                 RET
```

最后于 2018-10-26 12:19 被有影编辑 ， 原因:

收藏 · 81　点赞 · 2　打赏　分享

最新回复 (24)

春华q 2018-2-8 23:46　2 楼　0

极客

楼主辛苦了

lzgking 2018-2-9 00:38　3 楼　0

极客

感谢分享 辛苦

littlewisp 2 2018-2-18 07:16　4 楼　0

感谢分享，还差linux和mac，楼主继续

最新回复 (24)

holing 6 15 2018-2-18 08:05 5 楼 0
学者

可以

kanxue 11 8 2018-2-18 08:51 6 楼 0
坛主

小结的不错，有些地方可以在深入些

chixiaojie 6 2018-2-18 12:32 7 楼 0
极客

能收录优秀的一看就知道是出自大神之作。

有影 5 2018-2-18 13:03 8 楼 0
大侠

> kanxue 小结的不错，有些地方可以在深入些

感谢评优。新手自学中，目前还不是很懂哪些地方可以深入些？能不能指教一下？

CkDebug 2 2018-2-18 17:03 9 楼 0
极客

感谢分享　辛苦

maomaolk 2 2018-2-20 10:42 10 楼 0
极客

Mark

聖blue 4 2018-2-20 21:20 11 楼 0
极客

✌

ashiyouwu 2 2018-2-23 09:40 12 楼 0
极客

> 有影 感谢评优。新手自学中，目前还不是很懂哪些地方可以深入些？能不能指教一下？

对 arm 稍微熟悉一点。 arm里面参数超过了四个，四个之外的参数就不是存储到寄存器中，而是压栈了。
类似这些吧。

Yougar 3 2018-4-17 16:46 13 楼 0
极客

x86/x64调用约定最后ret n返回的地方有点问题，ret n的执行流程应该是pop eip 也就是ret 然后再add esp，n 楼主 你看看是不是这样

自由地 2 2018-4-17 19:06 14 楼 0
极客

> kanxue 小结的不错，有些地方可以在深入些

一楼的文章代码着色如何在复制到word文档中的时候能操持呢？
复制进去没有着色了。

首页 论坛 课程 招聘 发现

最新回复 (24)

icycityone 2 2018-4-20 11:17 15 楼 0
极客
感谢分享，学习中

老梨nobody 2 2018-6-11 06:34 16 楼 0
极客
谢谢分享，辛苦了

dapei 2 2018-6-19 13:27 17 楼 0
极客
lz 很用心，支持。函数调用约定调试常用到。

flylinfan 3 2018-6-19 17:57 18 楼 0
极客
总结的不错

andrOday 2 2018-10-12 10:35 19 楼 0
极客
优秀， 指出一点笔误：c++ arm64 "this指针存放R0中"， 应该是X0中吧

有影 5 2018-10-26 12:19 20 楼 0
大侠
> andrOday 优秀， 指出一点笔误：c++ arm64 "this指针存放R0中"， 应该是X0中吧

感谢指出，已修改。

yber 2 2018-10-29 08:59 21 楼 0
大侠
X64调用约定应该是_fastCall的扩展，调用方释放栈

chpeagle 7 2019-1-25 17:24 22 楼 1
专家
X64平台不同编译器传参方式:
vc调用的传参方式。前4个参数使用rcx,rdx,r8,r9，之后的参数使用堆栈。
GCC前6个参数使用rdi、rsi、rdx、rcx、r8、r9，剩下的参数用栈。注意rdx、rcx的顺序和MSVC上不一样。

最后于 2019-1-25 18:02 被chpeagle编辑 ，原因:

Lemonr 2 2019-11-8 00:23 23 楼 0
极客

killpy 6 2 2019-11-8 03:56 24 楼 0
大牛
> chpeagle X64平台不同编译器传参方式:vc调用的传参方式。前4个参数使用rcx,rdx,r8,r9，之后的参数使用堆栈。GCC前6个参数使用rdi、rsi、rdx、rcx、r8、r9，剩下的参数用栈。注意rd
> ...

感谢

首页 论坛 课程 招聘 发现

最新回复 (24)

TopC 2021-3-25 15:52 25 楼 0

极客

111

最后于 2021-3-25 15:57 被TopC编辑，原因：看错了，楼主总结的很对

游客

登录 | 注册 方可回帖

回帖 表情 高级回复

返回

首页 论坛 课程 招聘 发现