x64汇编

原创

1390811049 于 2020-09-15 00:27:15 发布 2406 收藏 8

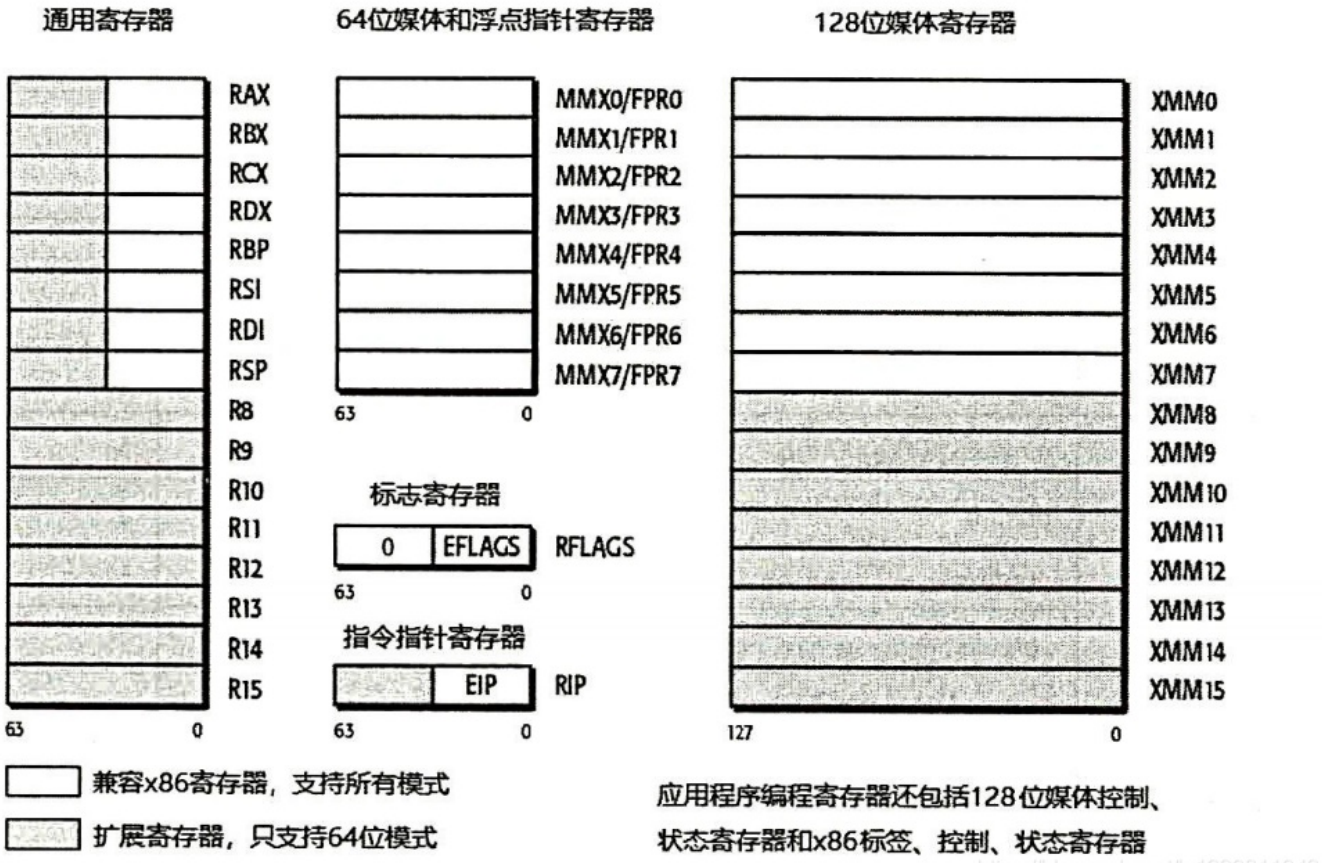
分类专栏： 逆向工程

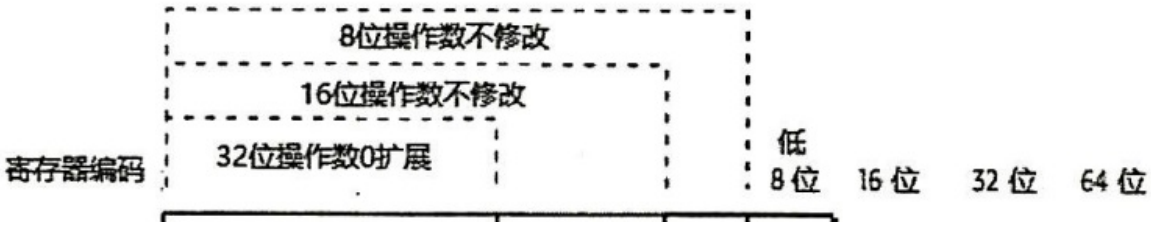
本文链接：https://blog.csdn.net/ly1390811049/article/details/108590489

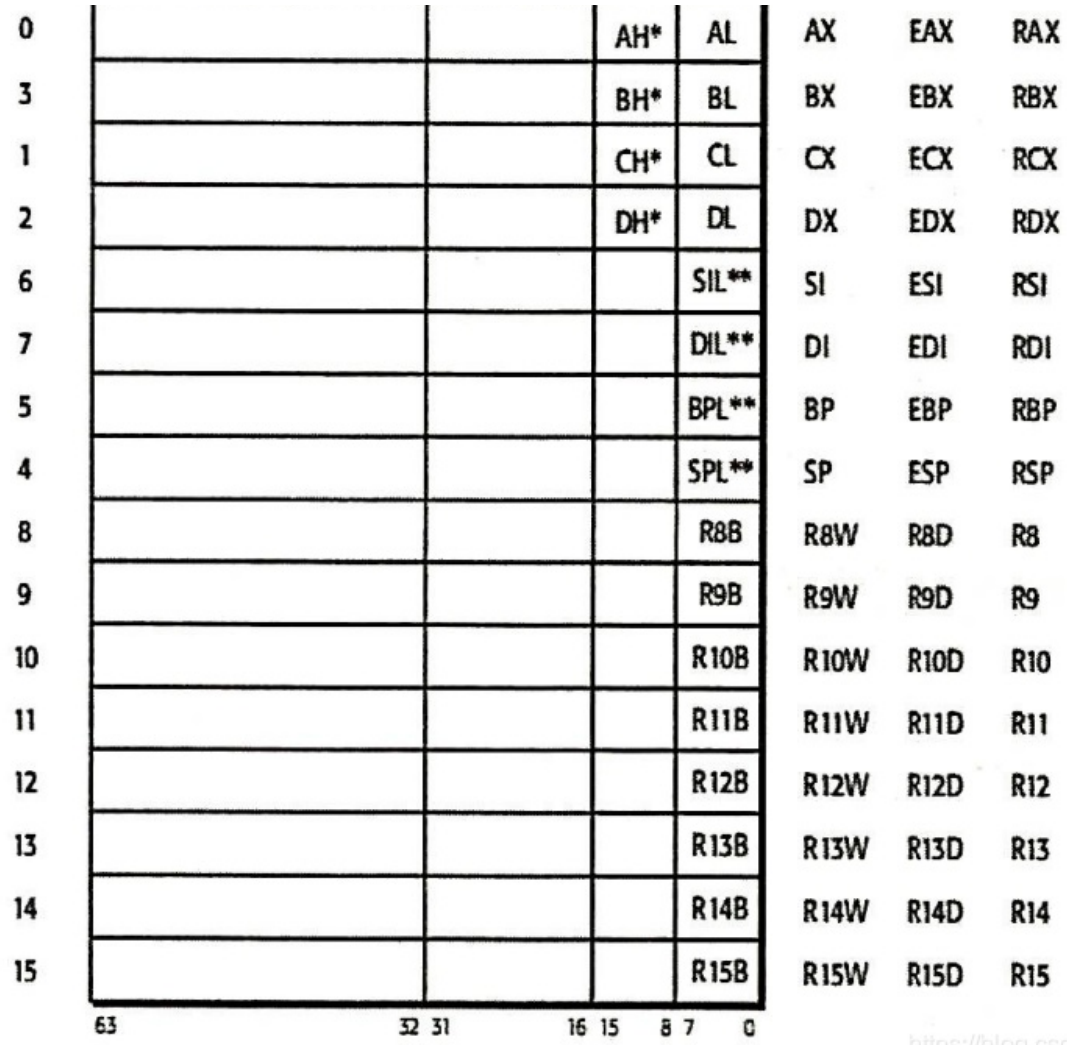
x64是AMD64与Intel64的合称，是指与现有x86兼容的64位CPU。在64位系统中，内存地址为64位。x64为环境下寄存器有较大的变化。

x64系统通用寄存器的名称，第1个字母从"E"改为"R"(例如"RAX"), 大小扩展到64位，数量增加了8个(R8~R15), 扩充了8个128位XMM寄存器(在64位程序中，XMM寄存器经常被用来优化代码)。64位寄存器与x86下的32位寄存器兼容，例如RAX(64位)、EAX(32位)、AX(低16位)、AL(低8位)、AH(8-15位)。

x64新扩展的寄存器高低位访问，使用WORD,BYTE,DWORD后缀，例如R8(64位)、R8D(低32位)、R8W(低16位)、R8B(低8位)



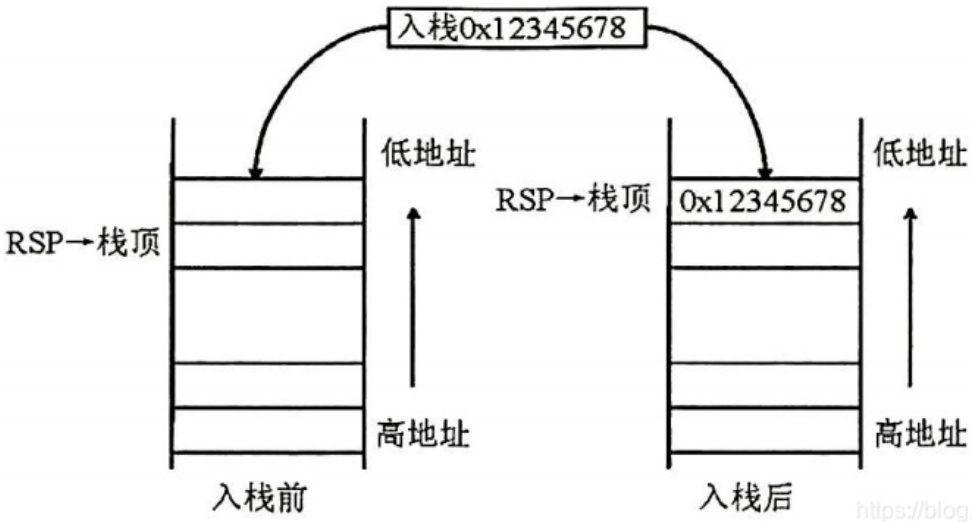




0			AH*	AL	AX	EAX	RAX
3			BH*	BL	BX	EBX	RBX
1			CH*	CL	CX	ECX	RCX
2			DH*	DL	DX	EDX	RDX
6				SIL**	SI	ESI	RSI
7				DIL**	DI	EDI	RDI
5				BPL**	BP	EBP	RBP
4				SPL**	SP	ESP	RSP
8				R8B	R8W	R8D	R8
9				R9B	R9W	R9D	R9
10				R10B	R10W	R10D	R10
11				R11B	R11W	R11D	R11
12				R12B	R12W	R12D	R12
13				R13B	R13W	R13D	R13
14				R14B	R14W	R14D	R14
15				R15B	R15W	R15D	R15
	63	32	31 16	15 8	7 0		

栈平衡

栈是程序在内存中的一块特殊区域,它的存储特点是先进后出,即先存储进去的数据最后被释放。RSP用来保存当前的栈顶指针,每8字节的栈空间用来保存一个数据。在汇编指令中,通常使用push和pop来入栈和出栈。



栈中存储的数据主要包括局部变量、函数参数、函数返回地址等。每当调用一个函数时，就会与函数调用前的位置一致。这个释放栈空间的过程称为栈平衡。
根据函数的需要申请相应的栈空间。当函数调用完成时，就需要释放刚才申请的栈空间，保证栈顶
为什么需要栈平衡?在程序运行过程中，栈内存空间会被各函数重复利用，如果函数调用只申请栈空间而不释放它，那么随着函数调用次数的增加，栈内存很快就会耗光，程序会因此无法正常运行。平衡栈的操作，目的是保证函数调用后的栈顶位置和函数调用前的位置一致，这样就可以重复利用栈的内存空间了。过多或者过少地释放栈空间都会影响其他函数对栈空间数据的操作，进而造成程序错误或者崩溃。需要注意的是，在x64环境下，某些汇编指令对栈顶的对齐值有要求，因此，Visual Studio编译器在申请栈空间时，会尽量保证栈顶地址的对齐值为16（可以被16整除）。如果在逆向过程中发现申请了栈空间却不使用的情况，可能就是为了实现对齐。

启动函数

程序在运行时，先执行初始化函数代码，再调用main函数执行用户编写的代码。

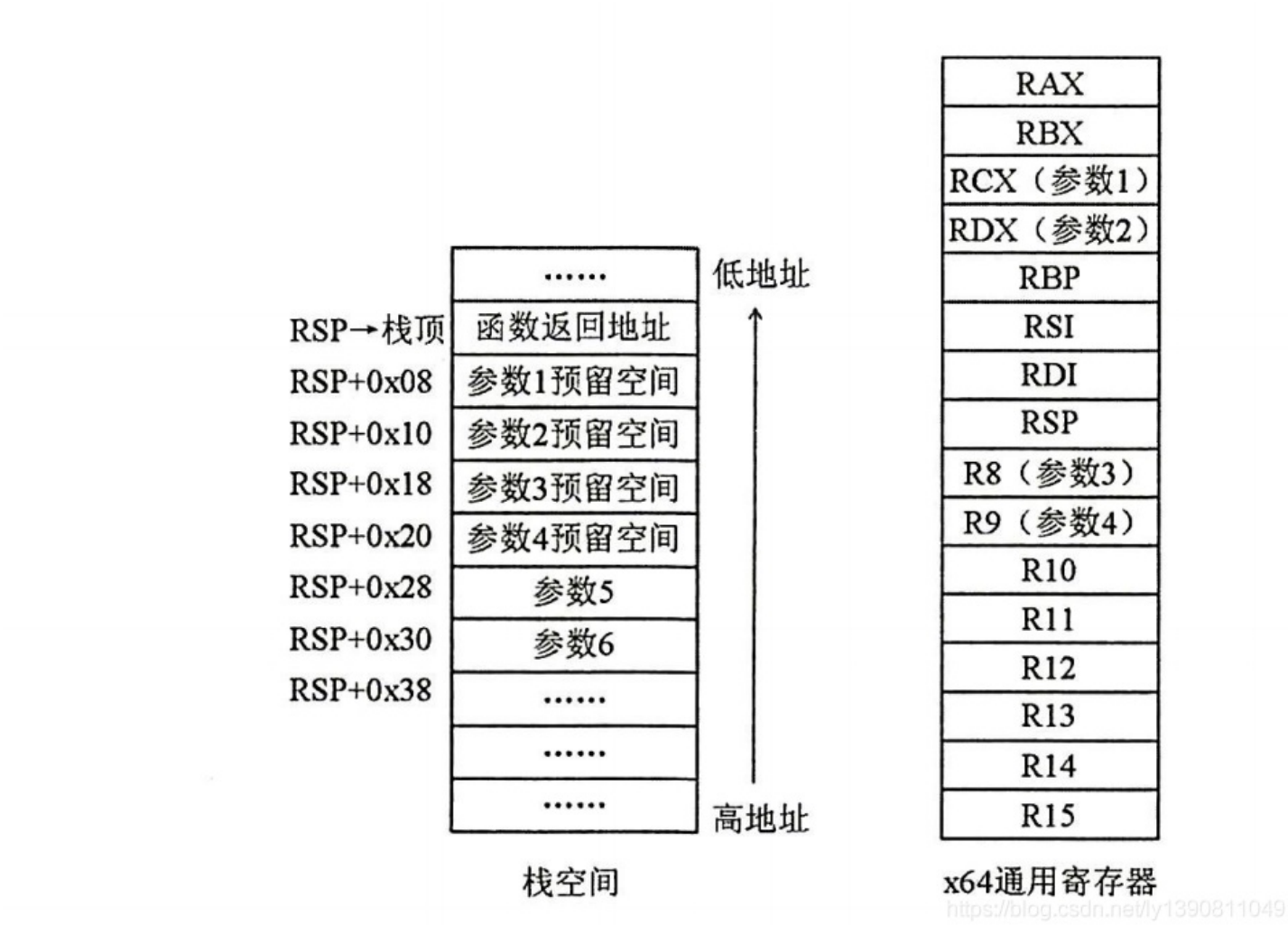
调用约定

x86应用程序的函数调用有stdcall、_cdecl、Fastcall等方式，但x64应用程序只有1种寄存器快速调用约定。前4个参数使用寄存器传递，如果参数超过4个，多余的参数就放在栈里，人栈顺序为从右到左，由函数调用方平衡栈空间。前4个参数存放的寄存器是固定的，分别是第1个参数RCX、第2个参数RDX、第3个参数R8、第4个参数R9，其他参数从右往左依次人栈。任何大于8字节或者不是1字节、2字节、4字节、8字节的参数必须由引用来传递（地址传递)。所有浮点参数的传递都是使用XMM 寄存器完成的，它们在XMMO、XMM1、XMM2和XMM3 中传递

参　数	类　型	浮点类型
第 1 个参数	RCX	XMM0
第 2 个参数	RDX	XMM1
第 3 个参数	R8	XMM2
第 4 个参数	R9	XMM3

注意:如果参数既有浮点类型，又有整数类型，例如“void fun(float,int，float,int)”，那么参数传递顺序为第1个参数（XMMO）、第2个参数（RDX)、第3个参数（XMM2)、第4个参数（R9)。

函数的前4个参数虽然使用寄存器来传递，但是栈仍然为这4个参数预留了空间(32字节)，为方便描述，这里称之为预留栈空间。在x64环境里，前4个参数使用寄存器传递，因此在函数内部这4个寄存器就不能使用了，相当于函数少了4个可用的通用寄存器。当函数功能比较复杂时，这可能导致寄存器不够用。为了避免这个问题，可以使用预留栈空间，方法是函数调用者多申请32字节的栈空间，当函数寄存器不够用时，可以把寄存器的值保存到刚才申请的栈空间中。预留栈空间由函数调用者提前申请。由函数调用者负责平衡栈空间。



2个参数的传递

```
#include <iostream>

int Add(int nNum1, int nNum2)
{
 return nNum1 + nNum2;
}

int main()
{
 std::cout << Add(1, 2);
 return 0;
}
```

vs2017的反汇编代码

```
main:
00007FF76BAF1840 40 55                push        rbp
00007FF76BAF1842 57                   push        rdi
00007FF76BAF1843 48 81 EC E8 00 00 00 sub         rsp,0E8h                        ;开辟栈空间，包括预留的32字节
00007FF76BAF184A 48 8D 6C 24 20       lea         rbp,[rsp+20h]         ;最后的32字节空间预留函数调用传参，rbp从除去预留空间的地方开始定位
00007FF76BAF184F 48 8B FC             mov         rdi,rsp
00007FF76BAF1852 B9 3A 00 00 00       mov         ecx,3Ah
00007FF76BAF1857 B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00007FF76BAF185C F3 AB                rep stos    dword ptr [rdi]
00007FF76BAF185E 48 8D 0D A3 F7 00 00 lea         rcx,[__04B3BE74_consoleapplication3@cpp (07FF76BB01008h)]
00007FF76BAF1865 E8 1D F8 FF FF       call        __CheckForDebuggerJustMyCode (07FF76BAF1087h)
00007FF76BAF186A BA 02 00 00 00       mov         edx,2               ;参数2
00007FF76BAF186F B9 01 00 00 00       mov         ecx,1      ; 参数1
00007FF76BAF1874 E8 4E F9 FF FF       call        Add (07FF76BAF11C7h)     ;调用Add函数
00007FF76BAF1879 8B D0                mov         edx,eax
00007FF76BAF187B 48 8B 0D CE E8 00 00 mov         rcx,qword ptr [__imp_std::cout (07FF76BB00150h)]
00007FF76BAF1882 FF 15 D0 E8 00 00    call        qword ptr [__imp_std::basic_ostream<char,std::char_traits<char> >::operator<< (07FF76BB00158h)]
00007FF76BAF1888 33 C0                xor         eax,eax
00007FF76BAF188A 48 8D A5 C8 00 00 00 lea         rsp,[rbp+0C8h]
00007FF76BAF1891 5F                   pop         rdi
00007FF76BAF1892 5D                   pop         rbp
00007FF76BAF1893 C3                   ret
```

Add函数汇编代码

```
Add:
00007FF76BAF1710 89 54 24 10          mov         dword ptr [rsp+10h],edx    ;将参数2保留到预留栈空间中
00007FF76BAF1714 89 4C 24 08          mov         dword ptr [rsp+8],ecx    ;将参数1保存到预留栈空间中
00007FF76BAF1718 55                   push        rbp             ;保存环境
00007FF76BAF1719 57                   push        rdi              ;保存环境
00007FF76BAF171A 48 81 EC E8 00 00 00 sub         rsp,0E8h        ; 开辟栈空间，包括预留的32字节
00007FF76BAF1721 48 8D 6C 24 20       lea         rbp,[rsp+20h]                          ;最后的32字节空间预留函数调用传参，rbp从除去预留空间的地方开始定位
00007FF76BAF1726 48 8B FC             mov         rdi,rsp
00007FF76BAF1729 B9 3A 00 00 00       mov         ecx,3Ah
00007FF76BAF172E B8 CC CC CC CC       mov         eax,0CCCCCCCCh
00007FF76BAF1733 F3 AB                rep stos    dword ptr [rdi]
00007FF76BAF1735 8B 8C 24 08 01 00 00 mov         ecx,dword ptr [rsp+108h]
00007FF76BAF173C 48 8D 0D C5 F8 00 00 lea         rcx,[__04B3BE74_consoleapplication3@cpp (07FF76BB01008h)]
00007FF76BAF1743 E8 3F F9 FF FF       call        __CheckForDebuggerJustMyCode (07FF76BAF1087h)
00007FF76BAF1748 8B 85 E8 00 00 00    mov         eax,dword ptr [nNum2]       ; [rbp+00000000000000E8h]从预留栈中取出参数    本函数开辟的栈空间为E8其中预留20H用于函数调用，rbp从E8 - 20H开始定位  + 保存环境10H + 返回地址8H + 参数1 8H  = rbp + E8
00007FF76BAF174E 8B 8D E0 00 00 00    mov         ecx,dword ptr [nNum1]      ;[rbp+00000000000000E0h]从预留栈中取出参数
00007FF76BAF1754 03 C8                add         ecx,eax
00007FF76BAF1756 8B C1                mov         eax,ecx
00007FF76BAF1758 48 8D A5 C8 00 00 00 lea         rsp,[rbp+0C8h]
00007FF76BAF175F 5F                   pop         rdi
00007FF76BAF1760 5D                   pop         rbp
00007FF76BAF1761 C3                   ret
```

传递4个以上参数

测试代码

```cpp
#include <iostream>

int Add(int nNum1, int nNum2, int nNum3, int nNum4, int nNum5, int nNum6, int nNum7)
{
 return nNum1 + nNum2 + nNum3 + nNum4 + nNum5 + nNum6 + nNum7;
}

int main()
{
 std::cout << Add(1, 2, 3, 4, 5, 6, 7);
 return 0;
}
```

main函数的反汇编代码

```
00007FF6993E3DE0 40 55                 push        rbp
00007FF6993E3DE2 57                    push        rdi
00007FF6993E3DE3 48 81 EC 08 01 00 00 sub         rsp,108h     ;开辟栈空间
00007FF6993E3DEA 48 8D 6C 24 40        lea         rbp,[rsp+40h]    ; 预留函数调用的参数空间 7 * 8 = 38H 内存对齐 8H
     rbp从这里开始访问局部变量 下面的40H为预留函数调用
00007FF6993E3DEF 48 8B FC              mov         rdi,rsp
00007FF6993E3DF2 B9 42 00 00 00        mov         ecx,42h
00007FF6993E3DF7 B8 CC CC CC CC        mov         eax,0CCCCCCCCh
00007FF6993E3DFC F3 AB                 rep stos    dword ptr [rdi]
00007FF6993E3DFE 48 8D 0D 03 D2 00 00 lea         rcx,[00007FF6993F1008h]
00007FF6993E3E05 E8 7D D2 FF FF        call        00007FF6993E1087
00007FF6993E3E0A C7 44 24 30 07 00 00 00 mov         dword ptr [rsp+30h],7   ; 参数7
00007FF6993E3E12 C7 44 24 28 06 00 00 00 mov         dword ptr [rsp+28h],6   ; 参数6
00007FF6993E3E1A C7 44 24 20 05 00 00 00 mov         dword ptr [rsp+20h],5  ; 参数5   [esp] - [esp + 18]这段空间是
预留给前4个参数的
00007FF6993E3E22 41 B9 04 00 00 00     mov         r9d,4        ; 参数4
00007FF6993E3E28 41 B8 03 00 00 00     mov         r8d,3        ; 参数3
00007FF6993E3E2E BA 02 00 00 00        mov         edx,2          ; 参数2
00007FF6993E3E33 B9 01 00 00 00        mov         ecx,1         ; 参数1
00007FF6993E3E38 E8 51 D5 FF FF        call        00007FF6993E138E    ;调用Add函数
00007FF6993E3E3D 8B D0                 mov         edx,eax
00007FF6993E3E3F 48 8B 0D 0A C3 00 00 mov         rcx,qword ptr [00007FF6993F0150h]
00007FF6993E3E46 FF 15 0C C3 00 00     call        qword ptr [00007FF6993F0158h]
00007FF6993E3E4C 33 C0                 xor         eax,eax
00007FF6993E3E4E 48 8D A5 C8 00 00 00 lea         rsp,[rbp+00000000000000C8h]
00007FF6993E3E55 5F                    pop         rdi
00007FF6993E3E56 5D                    pop         rbp
00007FF6993E3E57 C3                    ret
```

thiscall传递

thiscall是C++类的成员函数调用约定

```
class CAdd
{
public:
 int Add(int nNum1, int nNum2)
 {
  return nNum1 + nNum2;
 }
};

int main()
{
 CAdd object;
 std::cout << object.Add(1, 2);
 return 0;
}
```

main函数的反汇编

```
00007FF7AFB03DE0  push        rbp
00007FF7AFB03DE2  push        rdi
00007FF7AFB03DE3  sub         rsp,108h
00007FF7AFB03DEA  lea         rbp,[rsp+20h]
00007FF7AFB03DEF  mov         rdi,rsp
00007FF7AFB03DF2  mov         ecx,42h
00007FF7AFB03DF7  mov         eax,0CCCCCCCCh
00007FF7AFB03DFC  rep stos    dword ptr [rdi]
00007FF7AFB03DFE  mov         rax,qword ptr [00007FF7AFB0C040h]
00007FF7AFB03E05  xor         rax,rbp
00007FF7AFB03E08  mov         qword ptr [rbp+00000000000000D8h],rax
00007FF7AFB03E0F  lea         rcx,[00007FF7AFB11008h]
00007FF7AFB03E16  call        00007FF7AFB01087
00007FF7AFB03E1B  mov         r8d,2        参数2
00007FF7AFB03E21  mov         edx,1        参数1
00007FF7AFB03E26  lea         rcx,[rbp+4]                  ;rcx 用户传递this指针
00007FF7AFB03E2A  call        00007FF7AFB0139D      ; 调用CAdd::Add
00007FF7AFB03E2F  mov         edx,eax
00007FF7AFB03E31  mov         rcx,qword ptr [00007FF7AFB10150h]
00007FF7AFB03E38  call        qword ptr [00007FF7AFB10158h]
00007FF7AFB03E3E  xor         eax,eax
00007FF7AFB03E40  mov         edi,eax
00007FF7AFB03E42  lea         rcx,[rbp-20h]
00007FF7AFB03E46  lea         rdx,[00007FF7AFB0A790h]
00007FF7AFB03E4D  call        00007FF7AFB013A2
00007FF7AFB03E52  mov         eax,edi
00007FF7AFB03E54  mov         rcx,qword ptr [rbp+00000000000000D8h]
00007FF7AFB03E5B  xor         rcx,rbp
00007FF7AFB03E5E  call        00007FF7AFB010D7
00007FF7AFB03E63  lea         rsp,[rbp+00000000000000E8h]
00007FF7AFB03E6A  pop         rdi
00007FF7AFB03E6B  pop         rbp
00007FF7AFB03E6C  ret
```