汇编语言与逆向技术

第9章 C语言程序逆向分析

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2021-2022学年

本章知识点

1. 识别函数
2. 识别变量、数组、结构体
3. 识别IF分支结构
4. 识别Switch结构
5. 识别循环结构

1. 识别函数

启动函数

- 在编写Win32应用程序时，在源码里都有一个WinMain函数。
- Windows程序的执行并不是从WinMain函数开始的，而是先执行启动函数
  - 首先执行启动函数的代码，启动函数是编译器生成的
  - 启动函数初始化进程完成后，才会执行WinMain函数

启动函数

- C/C++程序运行时，启动函数的作用基本相同
  - 检索指向新进程的命令行指针
  - 检索指向新进程的环境变量指针
  - 全局变量初始化
  - 内存栈初始化

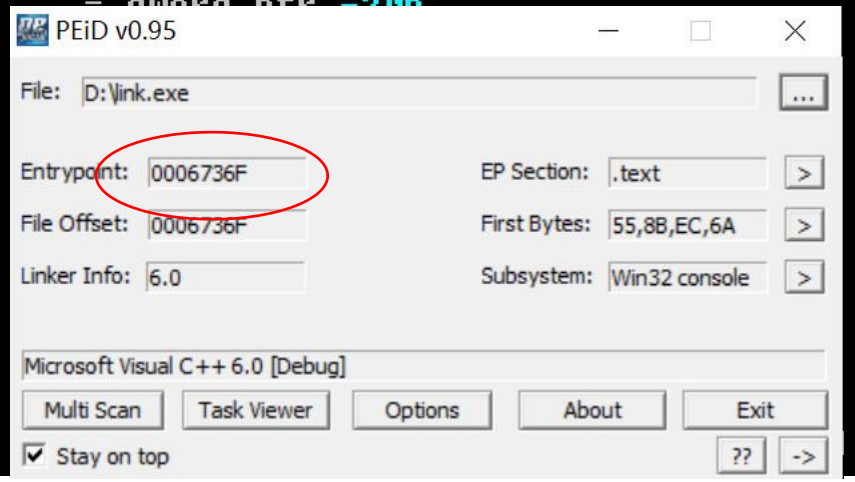
启动函数

- 当所有的初始化操作完成后，启动函数就会调用应用程序的进入点函数(main和WinMain)。
- 调用WinMain函数的示例

```
GetStartupInfo (&StartupInfo);
Int nMainRetVal = WinMain(GetModuleHandle(NULL),NULL,pszCommandLineAnsi, \
                  (StartupInfo.dwFlags&STARTF_USESHOWWINDOW)?StartupInfo.\
                   wShowWindow:SW__SHOWDEFAULT);
```

启动函数

```
.text:0046736F                         ; Attributes: library function noreturn bp-based frame
.text:0046736F
.text:0046736F                                         public start
.text:0046736F                         start           proc near
.text:0046736F
.text:0046736F                         Code            = dword ptr -30h
.text:0046736F                         argv            = dword ptr -2Ch
.text:0046736F                         var_28          = dword ptr -28h
.text:0046736F                         var_24          = dword ptr -24h
.text:0046736F                         envp
.text:0046736F                         argc
.text:0046736F                         ms_exc
.text:0046736F
.text:0046736F 55
.text:00467370 8B EC
.text:00467372 6A FF
.text:00467374 68 C0 90 40 00
.text:00467379 68 80 72 46 00
.text:0046737E 64 A1 00 00 00 00
.text:00467384 50
```



启动函数

```
.text:00467423 50                                      push    eax
.text:00467424 FF 15 AC 11 40 00                       call    ds:__getmainargs
.text:0046742A 68 18 90 46 00                          push    offset unk_469018
.text:0046742F 68 00 90 46 00                          push    offset unk_469000
.text:00467434 E8 53 00 00 00                          call    _initterm
.text:00467439 FF 15 A8 11 40 00                       call    ds:__p___initenv
.text:0046743F 8B 4D E0                                mov     ecx, [ebp+envp]
.text:00467442 89 08                                   mov     [eax], ecx
.text:00467444 FF 75 E0                                push    [ebp+envp]      ; envp
.text:00467447 FF 75 D4                                push    [ebp+argv]      ; argv
.text:0046744A FF 75 E4                                push    [ebp+argc]      ; argc
.text:0046744D E8 AE 66 FA FF                          call    _main
.text:00467452 83 C4 30                                add     esp, 30h
.text:00467455 89 45 DC                                mov     [ebp+var_24], eax
.text:00467458 50                                      push    eax             ; Code
.text:00467459 FF 15 44 11 40 00                       call    ds:exit
.text:0046745F                         ; ---------------------------------------------------------
```

启动函数

```
.text:0040DB00
.text:0040DB00                          ; =============== S U B R O U T I N E ===================================
.text:0040DB00                                                  |
.text:0040DB00                          ; Attributes: bp-based frame
.text:0040DB00
.text:0040DB00                          ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:0040DB00                          _main           proc near               ; CODE XREF: start+DE↓p
.text:0040DB00
.text:0040DB00                          var_248         = dword ptr -248h
.text:0040DB00                          var_244         = dword ptr -244h
.text:0040DB00                          var_240         = dword ptr -240h
.text:0040DB00                          Filename        = byte ptr -23Ch
.text:0040DB00                          var_13C         = dword ptr -13Ch
.text:0040DB00                          var_138         = dword ptr -138h
.text:0040DB00                          var_134         = dword ptr -134h
```

函数

- 程序通过CALL指令来调用函数，在函数执行结束后，通过RET指令返回调用程序继续执行

```
.text:0040DEBA 64 89 0D 00 00 00 00                    mov     large fs:0, ecx
.text:0040DEC1 5F                                      pop     edi
.text:0040DEC2 5E                                      pop     esi
.text:0040DEC3 5B                                      pop     ebx
.text:0040DEC4 8B E5                                   mov     esp, ebp
.text:0040DEC6 5D                                      pop     ebp
.text:0040DEC7 C3                                      retn
.text:0040DEC7                          _main          endp
.text:0040DEC7
```

函数

- C++ 函数定义
- return_type function_name(parameter list) {

    body of the function

  }

函数

- 函数的参数如何传递、局部变量如何定义、函数如何返回?
- CALL指令的操作数就是所调用函数的地址或者相对地址(MASM32的link.exe程序)

```
.text:00467442 89 08                  mov     [eax], ecx
.text:00467444 FF 75 E0               push    [ebp+envp]      ; envp
.text:00467447 FF 75 D4               push    [ebp+argv]      ; argv
.text:0046744A FF 75 E4               push    [ebp+argc]      ; argc
.text:0046744D E8 AE 66 FA FF         call    _main
.text:00467452 83 C4 30               add     esp, 30h
```

填空题 10分

设置

如下图所示，指令“call _main”的二进制编码是E8 AE 66 FA FF，该指令所在的内存地址是0046744Dh，则_main函数的入口地址是 [填空1].

注意：内存地址的书写格式，例如0046744Dh，可以写成0046744D、0046744dh、0046744d，都是正确的。32位的内存地址要保证8个有效字符，字符之间不加空格。

```
.text:00467442 89 08                  mov     [eax], ecx
.text:00467444 FF 75 E0               push    [ebp+envp]      ; envp
.text:00467447 FF 75 D4               push    [ebp+argv]      ; argv
.text:0046744A FF 75 E4               push    [ebp+argc]      ; argc
.text:0046744D E8 AE 66 FA FF         call    _main
.text:00467452 83 C4 30               add     esp, 30h
```

正常使用填空题需3.0以上版本雨课堂

作答

栈

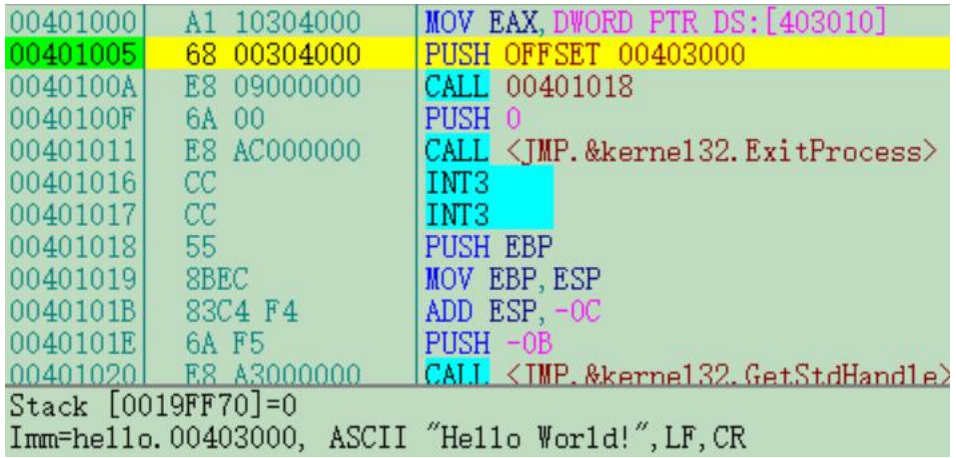
- 栈是一种后入先出的数据存储结构
- 函数的参数、局部变量、返回地址等被存储在栈中
- ESP (Extended Stack Pointer) 存储栈顶的内存地址
- EBP (Extended Base Pointer) 存储栈底的内存地址
- PUSH指令将数据压入栈顶
- POP指令从栈顶取出数据

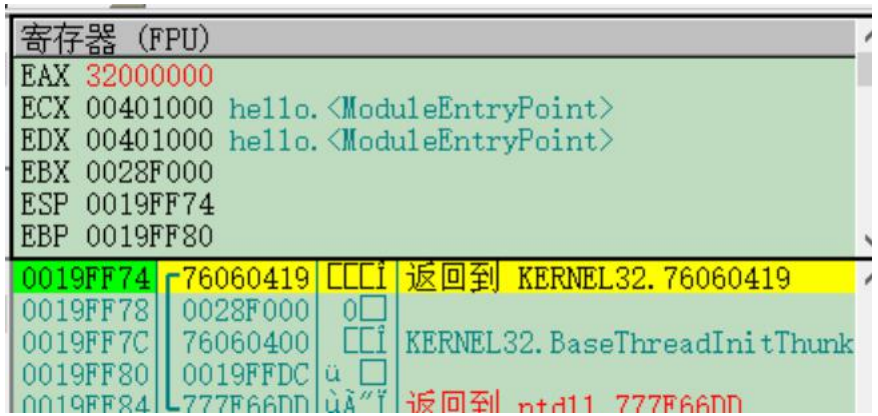
填空题 10分

设置

如下图所示，指令“PUSH OFFSET 00403000”执行之前，ESP的值是0019FF74。PUSH指令执行之后，ESP的值是[填空1]。

注意：32位的内存地址要保证8个有效字符，字符之间不加空格

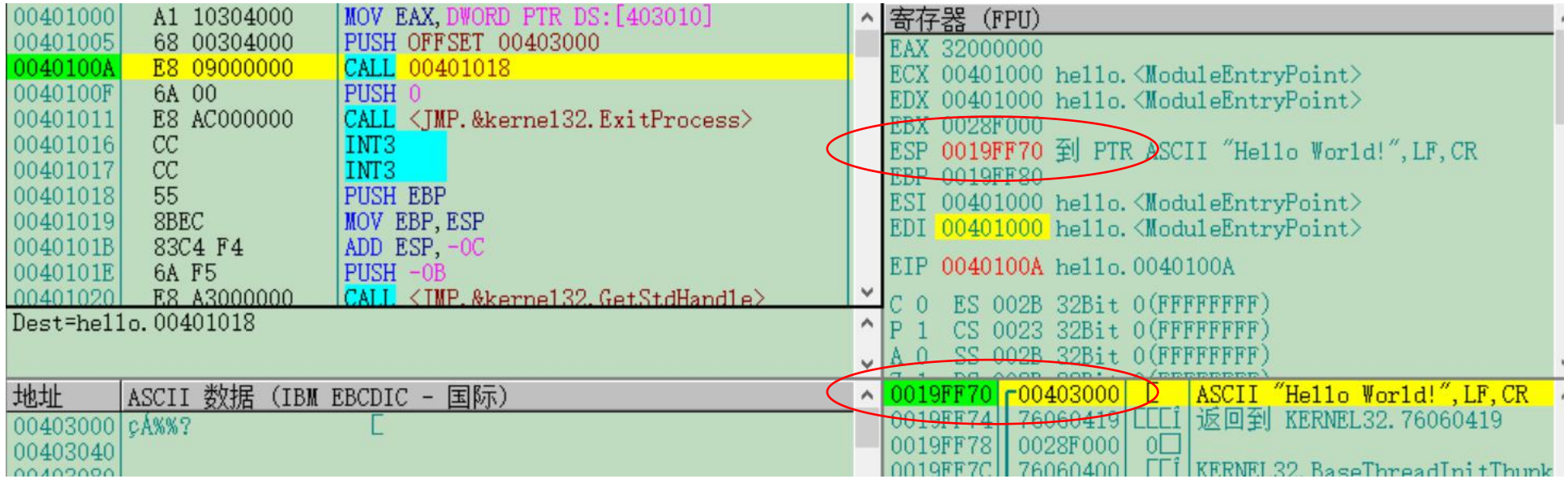




正常使用填空题需3.0以上版本雨课堂

作答

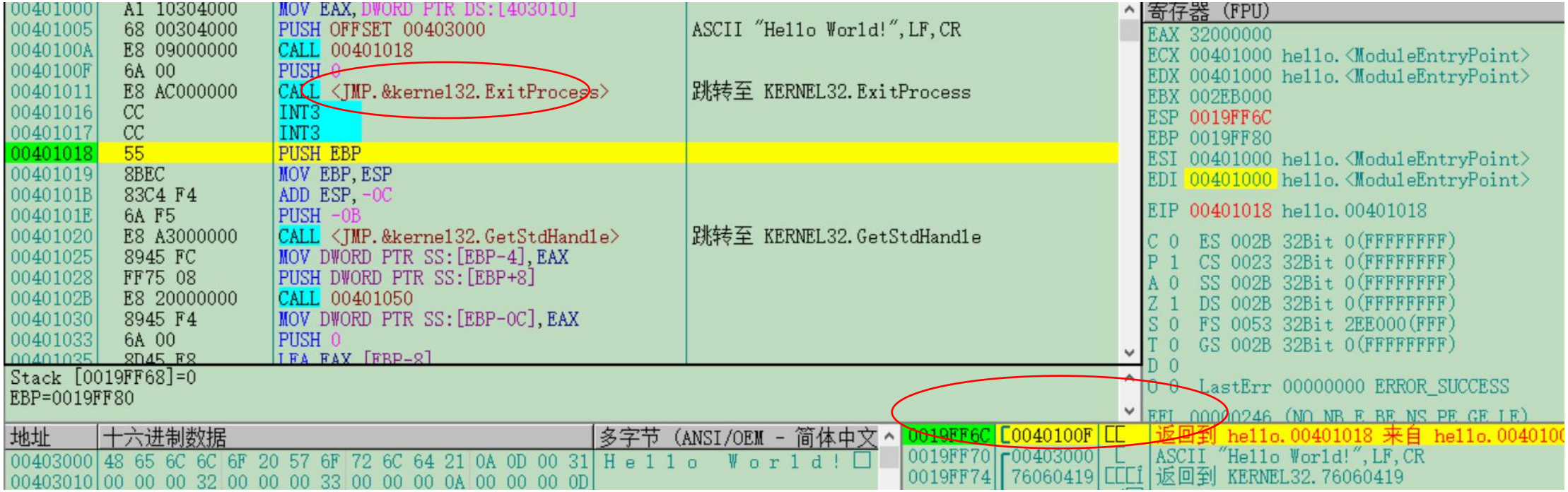
南開大學 Nankai University


00401000 A1 10304000 MOV EAX,DWORD PTR DS:[403010]
00401005 68 00304000 PUSH OFFSET 00403000
0040100A E8 09000000 CALL 00401018
0040100F 6A 00 PUSH 0
00401011 E8 AC000000 CALL <JMP.&kernel32.ExitProcess>
00401016 CC INT3
00401017 CC INT3
00401018 55 PUSH EBP
00401019 8BEC MOV EBP,ESP
0040101B 83C4 F4 ADD ESP,-0C
0040101E 6A F5 PUSH -0B
Dest=hello.00401018
寄存器 (FPU)
EAX 32000000
ECX 00401000 hello.<ModuleEntryPoint>
EDX 00401000 hello.<ModuleEntryPoint>
EBX 0028F000
ESP 0019FF70 到 PTR ASCII "Hello World!",LF,CR
EBP 0019FF80
ESI 00401000 hello.<ModuleEntryPoint>
EDI 00401000 hello.<ModuleEntryPoint>
EIP 0040100A hello.0040100A
C 0 ES 002B 32Bit 0(FFFFFFFF)
P 1 CS 0023 32Bit 0(FFFFFFFF)
A 0 SS 002B 32Bit 0(FFFFFFFF)
地址 ASCII 数据 (IBM EBCDIC - 国际)
00403000
00403040
0019FF70 00403000 ASCII "Hello World!",LF,CR
0019FF74 返回到 KERNEL32.76060419
0019FF78 0028F000
KERNEL32.BaseThreadInitThunk

函数的调用过程

- 使用push指令将参数压入栈中。
- call memory_location
  - call的返回地址压入栈中
  - EIP的值被设为memory_location
- push ebp，mov ebp，esp，add esp
  - 在栈中分配局部变量的空间

保存返回地址


00401000 A1 10304000 MOV EAX,DWORD PTR DS:[403010]
00401005 68 00304000 PUSH OFFSET 00403000 ASCII "Hello World!",LF,CR
0040100A E8 09000000 CALL 00401018
0040100F 6A 00 PUSH 0
00401011 E8 AC000000 CALL <JMP.&kernel32.ExitProcess> 跳转至 KERNEL32.ExitProcess
00401016 CC INT3
00401017 CC INT3
00401018 55 PUSH EBP
00401019 8BEC MOV EBP,ESP
0040101B 83C4 F4 ADD ESP,-0C
0040101E 6A F5 PUSH -0B
00401020 E8 A3000000 CALL <JMP.&kernel32.GetStdHandle> 跳转至 KERNEL32.GetStdHandle
00401025 8945 FC MOV DWORD PTR SS:[EBP-4],EAX
00401028 FF75 08 PUSH DWORD PTR SS:[EBP+8]
0040102B E8 20000000 CALL 00401050
00401030 8945 F4 MOV DWORD PTR SS:[EBP-0C],EAX
00401033 6A 00 PUSH 0
Stack [0019FF68]=0
EBP=0019FF80
寄存器 (FPU)
EAX 32000000
ECX 00401000 hello.<ModuleEntryPoint>
EDX 00401000 hello.<ModuleEntryPoint>
EBX 002EB000
ESP 0019FF6C
EBP 0019FF80
ESI 00401000 hello.<ModuleEntryPoint>
EDI 00401000 hello.<ModuleEntryPoint>
EIP 00401018 hello.00401018
C 0 ES 002B 32Bit 0(FFFFFFFF)
P 1 CS 0023 32Bit 0(FFFFFFFF)
A 0 SS 002B 32Bit 0(FFFFFFFF)
Z 1 DS 002B 32Bit 0(FFFFFFFF)
S 0 FS 0053 32Bit 2EE000(FFF)
T 0 GS 002B 32Bit 0(FFFFFFFF)
D 0
O 0 LastErr 00000000 ERROR_SUCCESS
地址 十六进制数据 多字节 (ANSI/OEM - 简体中文
00403000 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 0D 00 31 Hello World!
00403010 00 00 00 32 00 00 00 33 00 00 00 0A 00 00 00 0D
0019FF6C 0040100F 返回到 hello.00401018 来自 hello.0040100
0019FF70 00403000 ASCII "Hello World!",LF,CR
0019FF74 76060419 返回到 KERNEL32.76060419

局部变量的初始化

```
; int __stdcall sub_401018(LPCVOID lpBuffer)
sub_401018      proc near               ; CODE XREF: start+A↑p
                                        |
nNumberOfBytesToWrite= dword ptr -0Ch
NumberOfBytesWritten= dword ptr -8
hFile           = dword ptr -4
lpBuffer        = dword ptr  8

                push    ebp
                mov     ebp, esp
                add     esp, 0FFFFFFF4h
                push    0FFFFFFF5h      ; nStdHandle
                call    GetStdHandle
```

```
0019FF5C  00000000
0019FF60  00000000
0019FF64  00000000
0019FF68  0019FF80  0 □
0019FF6C  0040100F  □□  返回到 hello.00401018 来自 hello.0040100F
0019FF70  00403000  □   ASCII "Hello World!",LF,CR
```

栈帧（Stack Frame）

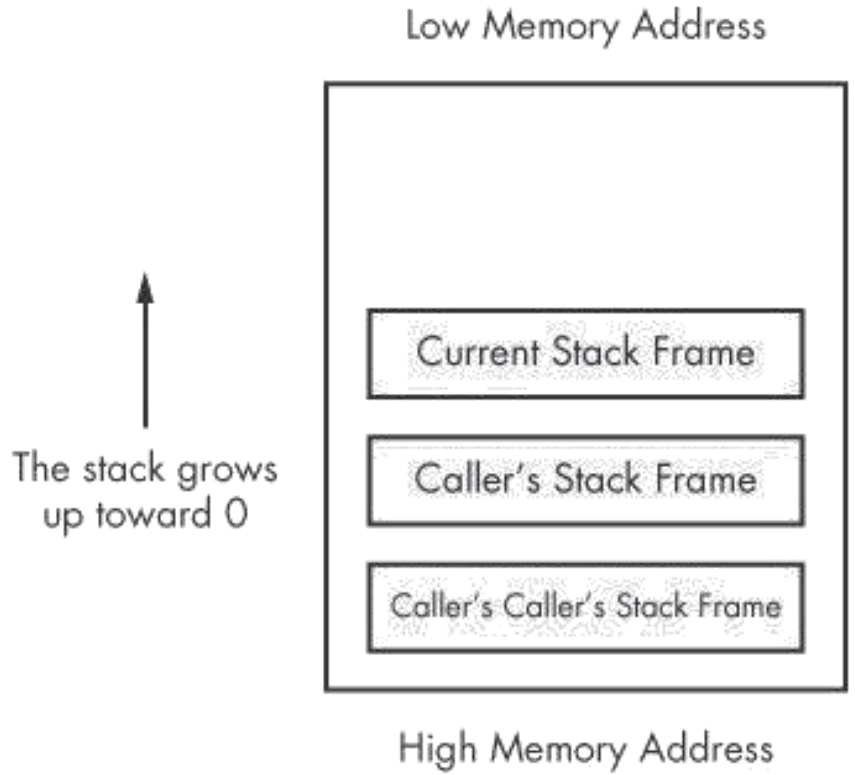


Figure 5-7. x86 stack layout

栈帧（Stack Frame）

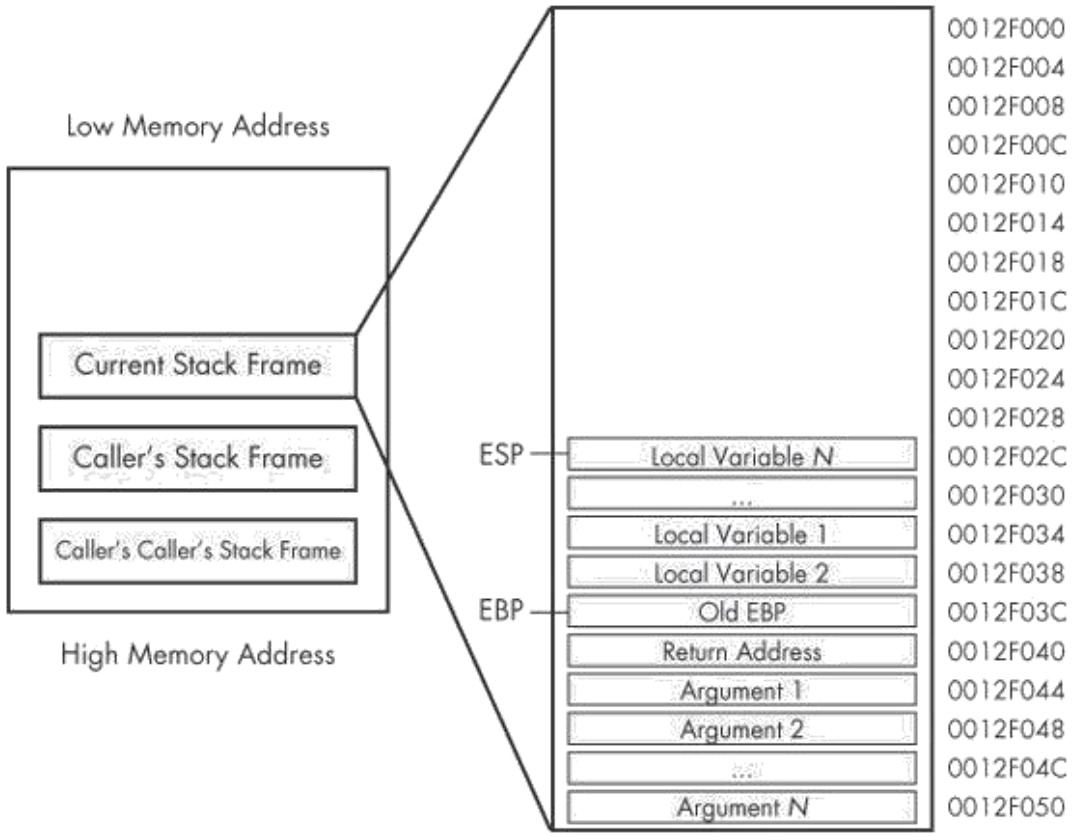
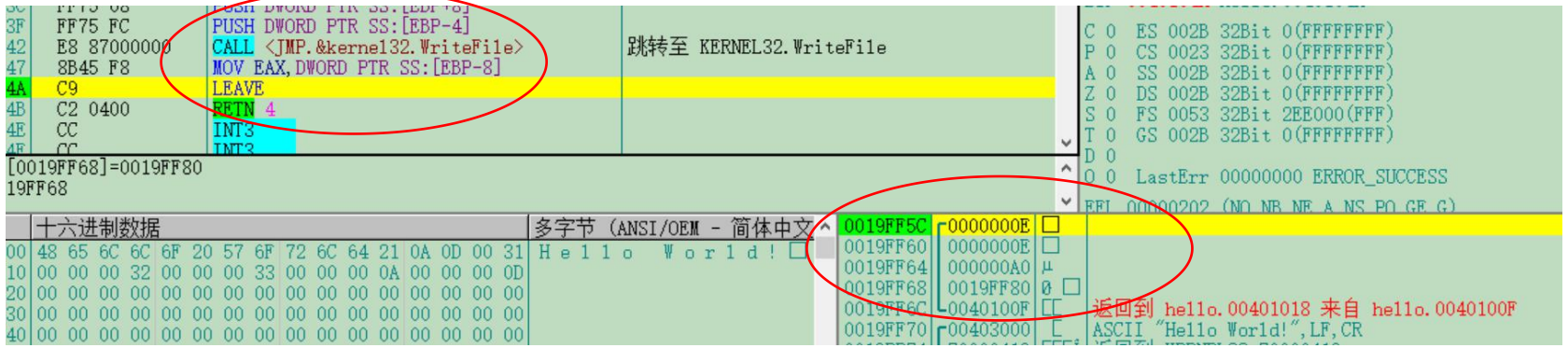


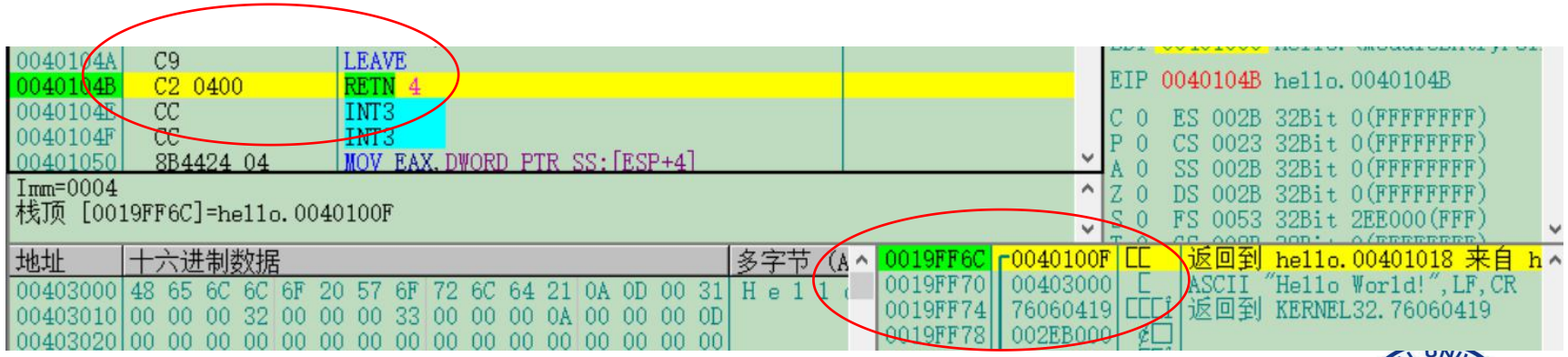
Figure 5-8. Individual stack frame

函数调用过程

- 执行函数
- 恢复局部变量的栈空间
- ret指令从栈中读取返回地址，设置EIP
- 恢复参数占用的栈空间

函数的调用过程


3F FF75 FC PUSH DWORD PTR SS:[EBP-4]
42 E8 87000000 CALL <JMP.&kernel32.WriteFile>
跳转至 KERNEL32.WriteFile
47 8B45 F8 MOV EAX,DWORD PTR SS:[EBP-8]
4A C9 LEAVE
4B C2 0400 RETN 4
4E CC INT3
[0019FF68]=0019FF80
19FF68
十六进制数据
多字节 (ANSI/OEM - 简体中文
00 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 0D 00 31 Hello World!
10 00 00 00 32 00 00 00 33 00 00 00 0A 00 00 00 0D
20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0019FF5C 0000000E
0019FF60 0000000E
0019FF64 000000A0
0019FF68 0019FF80
0019FF6C 0040100F
0019FF70 00403000
C 0 ES 002B 32Bit 0(FFFFFFFF)
P 0 CS 0023 32Bit 0(FFFFFFFF)
A 0 SS 002B 32Bit 0(FFFFFFFF)
Z 0 DS 002B 32Bit 0(FFFFFFFF)
S 0 FS 0053 32Bit 2EE000(FFF)
T 0 GS 002B 32Bit 0(FFFFFFFF)
D 0
O 0 LastErr 00000000 ERROR_SUCCESS
返回到 hello.00401018 来自 hello.0040100F
ASCII "Hello World!",LF,CR


0040104A C9 LEAVE
0040104B C2 0400 RETN 4
0040104E CC INT3
0040104F CC INT3
00401050 8B4424 04 MOV EAX,DWORD PTR SS:[ESP+4]
Imm=0004
栈顶 [0019FF6C]=hello.0040100F
地址
十六进制数据
00403000 48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 0D 00 31
00403010 00 00 00 32 00 00 00 33 00 00 00 0A 00 00 00 0D
00403020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0019FF6C 0040100F
0019FF70 00403000
0019FF74 76060419
0019FF78 002EB000
EIP 0040104B hello.0040104B
C 0 ES 002B 32Bit 0(FFFFFFFF)
P 0 CS 0023 32Bit 0(FFFFFFFF)
A 0 SS 002B 32Bit 0(FFFFFFFF)
Z 0 DS 002B 32Bit 0(FFFFFFFF)
S 0 FS 0053 32Bit 2EE000(FFF)
返回到 hello.00401018 来自
ASCII "Hello World!",LF,CR
返回到 KERNEL32.76060419

调用约定（Calling Convention）

```
; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
_main           proc near               ; CODE XREF: start+AF↓p

var_4           = dword ptr -4
argc            = dword ptr  8
argv            = dword ptr  0Ch
envp            = dword ptr  10h

                push    ebp
                mov     ebp, esp
```

Calling Convention

- 在x86平台，函数所有参数的宽度都是32bits
- 函数的返回值（Return values）的宽度是 32bits， 存储在EAX 寄存器中

Calling Convention

- 被调函数callee和主函数caller如何传递参数和返回值的约定
- VC 编译器支持以下两种调用约定
  - __cdecl
  - __stdcall

Calling Convention

- **__cdecl** 是 C and C++ 程序的标准函数调用

Element	Implementation
Argument-passing order	Right to left.
Stack-maintenance responsibility	Calling function pops the arguments from the stack.

__cdecl

```
lea     ecx, [eax+7]
mov     dl, [eax+6]
push    ecx
push    edx
lea     eax, [esp+70h+var_64]
push    offset aS_0     ; "%*s"
push    eax             ; char *
call    _sprintf
add     esp, 10h
lea     ecx, [esp+68h+var_64]
push    0               ; int
```

Calling Convention

- **__stdcall** 是Win32 API 函数的调用约定

Element	Implementation
Argument-passing order	Right to left.
Stack-maintenance responsibility	Called function pops its own arguments from the stack.

__stdcall

```
loc_438E61:                             ; CODE XREF: __lseek+36↑j
                push    [esp+0Ch+dwMoveMethod] ; dwMoveMethod
                push    0               ; lpDistanceToMoveHigh
                push    [esp+14h+lDistanceToMove] ; lDistanceToMove
                push    eax             ; hFile
                call    ds:SetFilePointer
                mov     ebx, eax
                cmp     ebx, 0FFFFFFFFh
                jnz     short loc_438E81
                call    ds:GetLastError
                jmp     short loc_438E83
```

单选题 20分

设置

函数URLDownloadToFileA的调用约定是?

```
push    0                       ; LPBINDSTATUSCALLBACK
push    0                       ; DWORD
push    offset aCEmpdownload_e ; "c:\tempdownload.exe"
mov     eax, [ebp+var_4]
mov     ecx, [eax]
push    ecx                     ; LPCSTR
push    0                       ; LPUNKNOWN
call    URLDownloadToFileA
mov     esp, ebp
pop     ebp
retn
endp
```

A __cdecl

B __stdcall

提交

南開大學 Nankai University

Calling Convention

```
push    0                       ; LPBINDSTATUSCALLBACK
push    0                       ; DWORD
push    offset aCEmpdownload_e ; "c:\tempdownload.exe"
mov     eax, [ebp+var_4]
mov     ecx, [eax]
push    ecx                     ; LPCSTR
push    0                       ; LPUNKNOWN
call    URLDownloadToFileA
mov     esp, ebp
pop     ebp
retn
endp
```

识别变量、数组、结构体

局部变量和全局变量

- 全局变量
  - 可以任意函数访问和修改的变量
- 局部变量
  - 只能在定义该变量的函数内部，访问和修改

全局变量和局部变量

```
int x = 1;
int y = 2;

void main() {
  x = x+y;
  printf("Total = %d\n", x);
}
```

```
void main() {
  int x = 1;
  int y = 2;
  x = x+y;
  printf("Total = %d\n", x);
}
```

全局变量

```
00401003        mov     eax, dword_40CF60
00401008        add     eax, dword_40C000
0040100E        mov     dword_40CF60, eax ❶
00401013        mov     ecx, dword_40CF60
00401019        push    ecx
0040101A        push    offset aTotalD  ;"total = %d\n"
0040101F        call    printf
```

局部变量

```
00401006        mov     dword ptr [ebp-4], 0
0040100D        mov     dword ptr [ebp-8], 1
00401014        mov     eax, [ebp-4]
00401017        add     eax, [ebp-8]
0040101A        mov     [ebp-4], eax
0040101D        mov     ecx, [ebp-4]
00401020        push    ecx
00401021        push    offset aTotalD  ; "total = %d\n"
00401026        call    printf
```

单选题 10分

设置

mov eax, [ebp+var_4]”
[ebp+var_4] 一个全局变量还是局部变量？

A 局部变量

B 全局变量

提交

数组

- 数组是相同数据类型的元素的集合，它们在内存中按顺序连续存放在一起。
- 在汇编状态下访问数组一般是通过基址加变址寻址实现的

数组

- int ary[4] = {1, 2, 3, 4}
  - 每个整数占用4个字节，数组占用了16个字节，假设数组的首地址是0x1000
  - ary[0] 的位置是0x1000
  - ary[1]的位置是0x1004
  - ary[2]的位置是0x1008
  - ary[3]的位置是0x100C
- 数组元素的地址=数组首地址+ sizeof(元素类型)*索引值

数组

- 数组a是局部
- 变量，数组b是全局变量

```
int b[5] = {123,87,487,7,978};
void main()
{
   int i;
   int a[5];

   for(i = 0; i<5; i++)
   {
      a[i] = i;
      b[i] = i;
   }
}
```

数组

```
00401006          mov     [ebp+var_18], 0
0040100D          jmp     short loc_401018
0040100F loc_40100F:
0040100F          mov     eax, [ebp+var_18]
00401012          add     eax, 1
00401015          mov     [ebp+var_18], eax
00401018 loc_401018:
00401018          cmp     [ebp+var_18], 5
0040101C          jge     short loc_401037
0040101E          mov     ecx, [ebp+var_18]
00401021          mov     edx, [ebp+var_18]
00401024          mov     [ebp+ecx*4+var_14], edx ❶
00401028          mov     eax, [ebp+var_18]
0040102B          mov     ecx, [ebp+var_18]
0040102E          mov     dword_40A000[ecx*4], eax ❷
00401035          jmp     short loc_40100F
```

结构体

- 在c语言中，结构体(struct)是一种数据结构，可以将不同类型的数据结构组合到一个复合的数据类型中

结构体

```
struct my_structure { ❶
      int x[5];
      char y;
      double z;
};

struct my_structure *gms; ❷

void test(struct my_structure *q)
{
      int i;
      q->y = 'a';
      q->z = 15.6;
      for(i = 0; i<5; i++){
            q->x[i] = i;
      }
}

void main()
{
      gms = (struct my_structure *) malloc(
      sizeof(struct my_structure));
      test(gms);
}
```

```
00401000          push    ebp
00401001          mov     ebp, esp
00401003          push    ecx
00401004          mov     eax,[ebp+arg_0]
00401007          mov     byte ptr [eax+14h], 61h
0040100B          mov     ecx, [ebp+arg_0]
0040100E          fld     ds:dbl_40B120 ❶
00401014          fstp    qword ptr [ecx+18h]
00401017          mov     [ebp+var_4], 0
0040101E          jmp     short loc_401029
00401020 loc_401020:
00401020          mov     edx,[ebp+var_4]
00401023          add     edx, 1
00401026          mov     [ebp+var_4], edx
00401029 loc_401029:
00401029          cmp     [ebp+var_4], 5
0040102D          jge     short loc_40103D
0040102F          mov     eax,[ebp+var_4]
00401032          mov     ecx,[ebp+arg_0]
00401035          mov     edx,[ebp+var_4]
00401038          mov     [ecx+eax*4],edx ❷
0040103B          jmp     short loc_401020
0040103D loc_40103D:
0040103D          mov     esp, ebp
0040103F          pop     ebp
00401040          retn
```

结构体

```
00000000 ; (Class Informer)
00000000 type_info       struc ; (sizeof=0x8, variable size) ; XREF: sub_4175C0↓r
00000000 vftable         dd ?                    ; offset (00000000)
00000004 _m_data         dd ?
00000008 _m_d_name       db 0 dup(?)             ; string(C)
00000008 type_info       ends
00000008
00000000 ; ---------------------------------------------------------------------
00000000
00000000 ; (Class Informer)
00000000 PMD             struc ; (sizeof=0xC)    ; XREF: RTTIBaseClassDescriptor↓r
00000000 mdisp           dd ?
00000004 pdisp           dd ?
00000008 vdisp           dd ?
0000000C PMD             ends
```

3. 识别IF分支结构

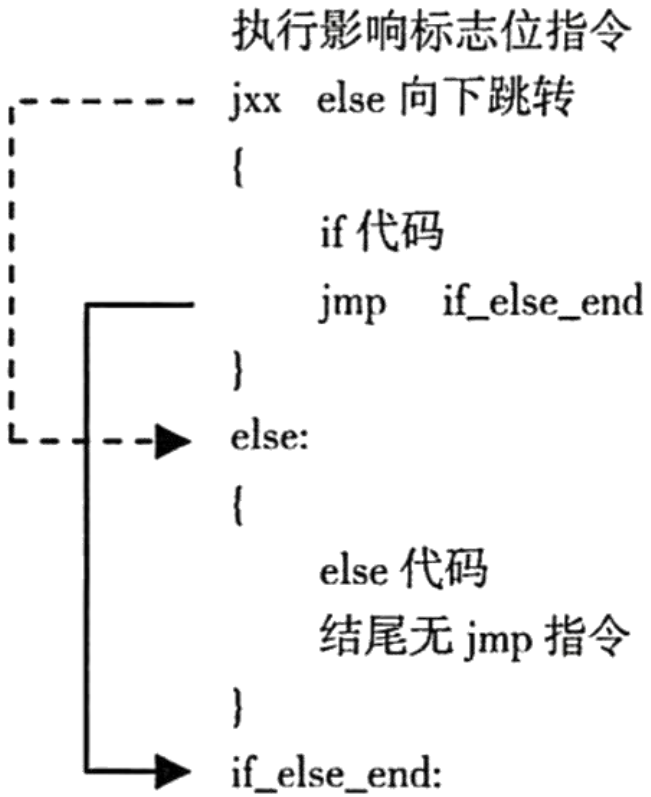
识别IF分支结构

```
int x = 1;
int y = 2;

if(x == y){
      printf("x equals y.\n");
}else{
      printf("x is not equal to y.\n");
}
```

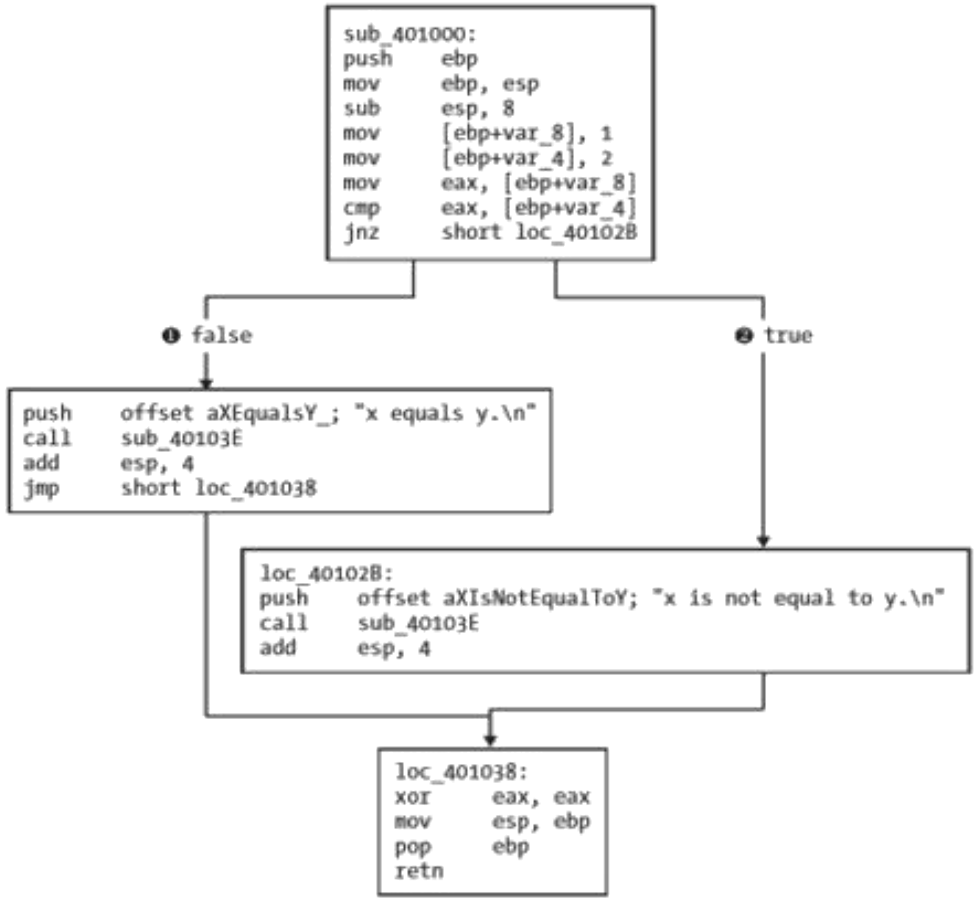
```
00401006          mov     [ebp+var_8], 1
0040100D          mov     [ebp+var_4], 2
00401014          mov     eax, [ebp+var_8]
00401017          cmp     eax, [ebp+var_4] ❶
0040101A          jnz     short loc_40102B ❷
0040101C          push    offset aXEqualsY_ ; "x equals y.\n"
00401021          call    printf
00401026          add     esp, 4
00401029          jmp     short loc_401038 ❸
0040102B loc_40102B:
0040102B          push    offset aXIsNotEqualToY ; "x is not equal to y.\n"
00401030          call    printf
```

识别IF分支结构

- IF语句的识别特征，jxx的跳转和一个无条件jmp指令



识别IF分支结构

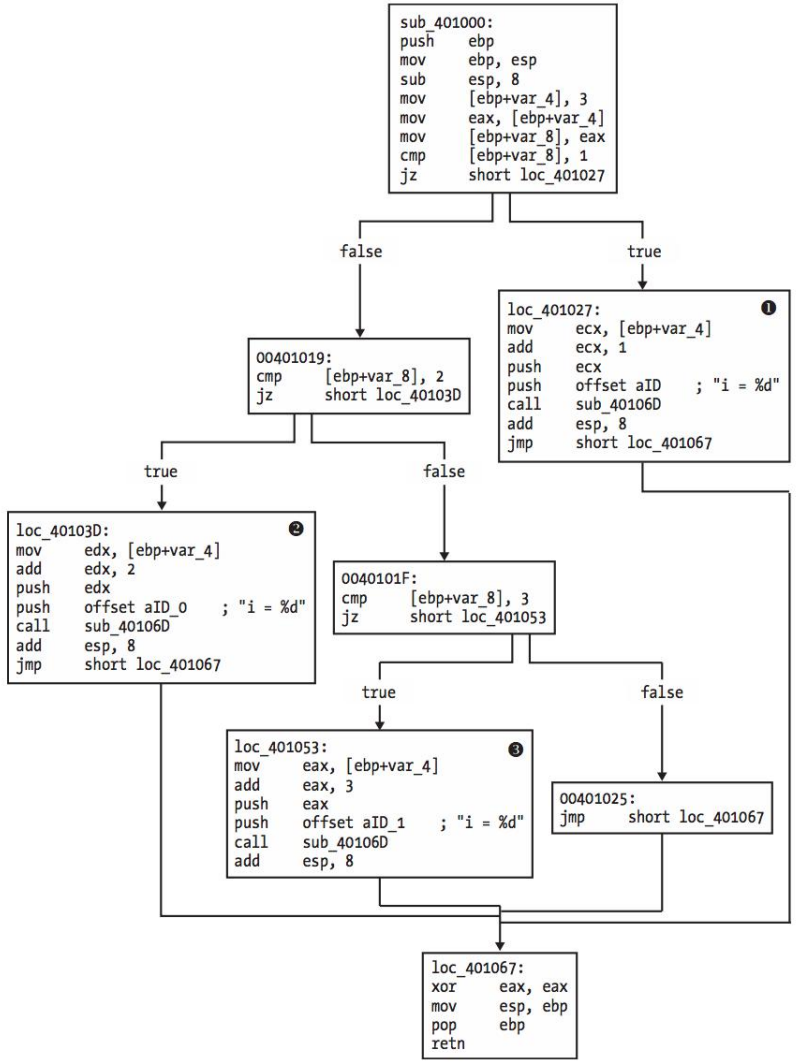

sub_401000:
push ebp
mov ebp, esp
sub esp, 8
mov [ebp+var_8], 1
mov [ebp+var_4], 2
mov eax, [ebp+var_8]
cmp eax, [ebp+var_4]
jnz short loc_40102B
❶ false
❷ true
push offset aXEqualsY_; "x equals y.\n"
call sub_40103E
add esp, 4
jmp short loc_401038
loc_40102B:
push offset aXIsNotEqualToY; "x is not equal to y.\n"
call sub_40103E
add esp, 4
loc_401038:
xor eax, eax
mov esp, ebp
pop ebp
retn

4. 识别Switch结构

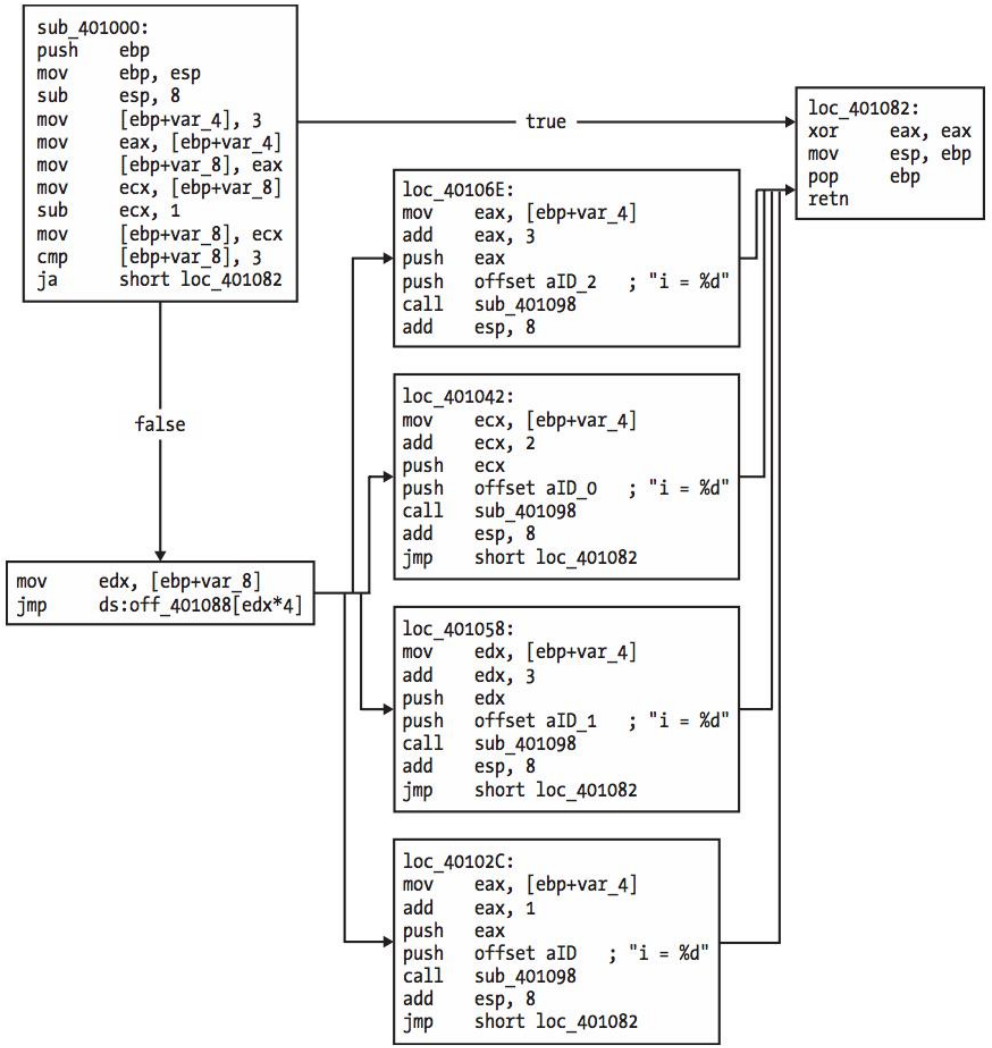
识别Switch结构

- Switch结构用来实现基于字符或者整数的决策。
- Switch结构通常以两种方式被编译
  - 使用IF方式
  - 使用跳转表

识别Switch结构


sub_401000:
push ebp
mov ebp, esp
sub esp, 8
mov [ebp+var_4], 3
mov eax, [ebp+var_4]
mov [ebp+var_8], eax
cmp [ebp+var_8], 1
jz short loc_401027
false
true
loc_401027:
mov ecx, [ebp+var_4]
add ecx, 1
push ecx
push offset aID ; "i = %d"
call sub_40106D
add esp, 8
jmp short loc_401067
00401019:
cmp [ebp+var_8], 2
jz short loc_40103D
true
false
loc_40103D:
mov edx, [ebp+var_4]
add edx, 2
push edx
push offset aID_0 ; "i = %d"
call sub_40106D
add esp, 8
jmp short loc_401067
0040101F:
cmp [ebp+var_8], 3
jz short loc_401053
true
false
loc_401053:
mov eax, [ebp+var_4]
add eax, 3
push eax
push offset aID_1 ; "i = %d"
call sub_40106D
add esp, 8
00401025:
jmp short loc_401067
loc_401067:
xor eax, eax
mov esp, ebp
pop ebp
retn

跳转表


sub_401000:
push ebp
mov ebp, esp
sub esp, 8
mov [ebp+var_4], 3
mov eax, [ebp+var_4]
mov [ebp+var_8], eax
mov ecx, [ebp+var_8]
sub ecx, 1
mov [ebp+var_8], ecx
cmp [ebp+var_8], 3
ja short loc_401082
true
false
mov edx, [ebp+var_8]
jmp ds:off_401088[edx*4]
loc_401082:
xor eax, eax
mov esp, ebp
pop ebp
retn
loc_40106E:
mov eax, [ebp+var_4]
add eax, 3
push eax
push offset aID_2 ; "i = %d"
call sub_401098
add esp, 8
loc_401042:
mov ecx, [ebp+var_4]
add ecx, 2
push ecx
push offset aID_0 ; "i = %d"
call sub_401098
add esp, 8
jmp short loc_401082
loc_401058:
mov edx, [ebp+var_4]
add edx, 3
push edx
push offset aID_1 ; "i = %d"
call sub_401098
add esp, 8
jmp short loc_401082
loc_40102C:
mov eax, [ebp+var_4]
add eax, 1
push eax
push offset aID ; "i = %d"
call sub_401098
add esp, 8
jmp short loc_401082

5. 识别循环

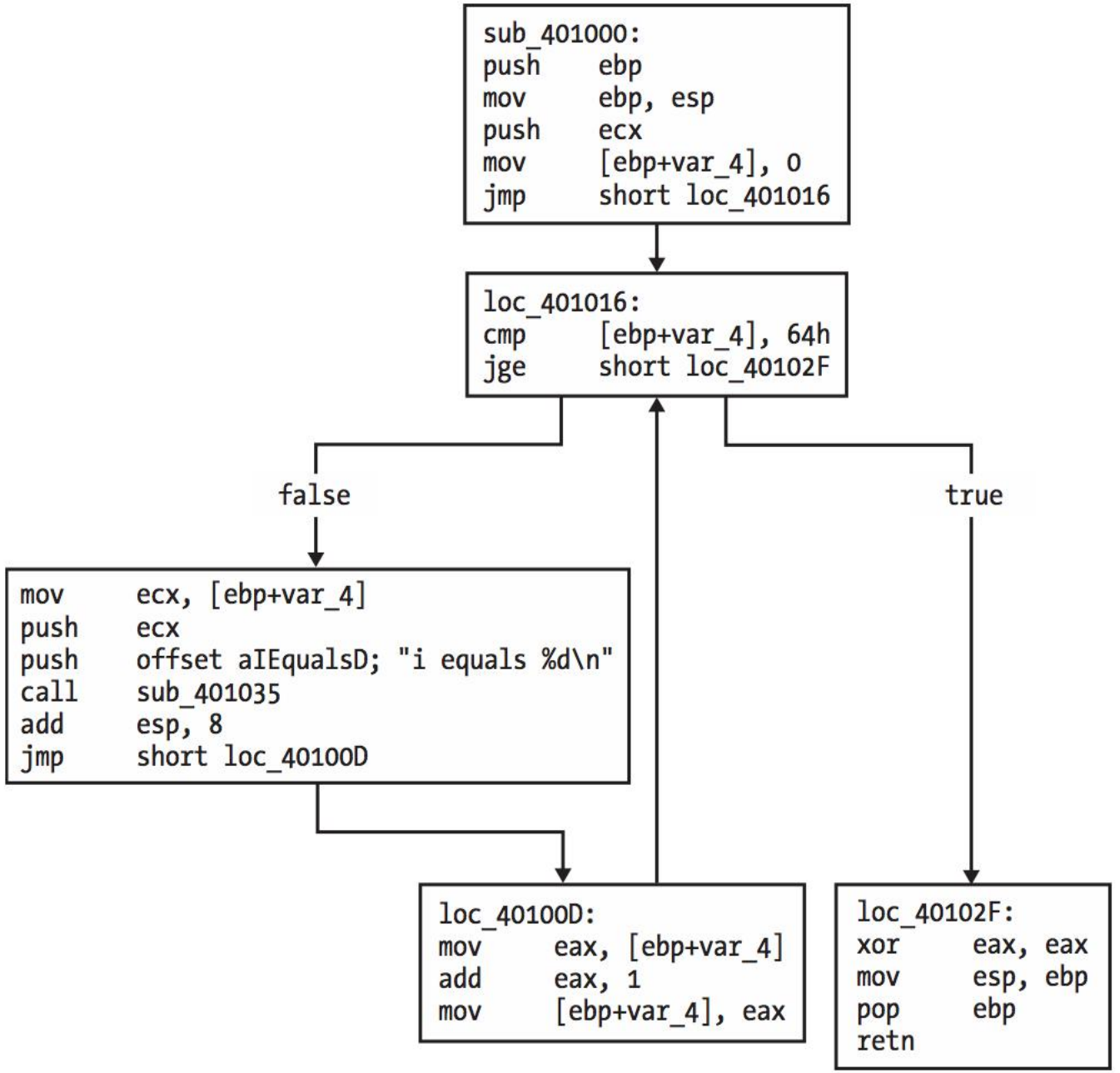
识别循环

- FOR循环是一个C/C++编程使用的基本循环机制。
- FOR循环有4个组件：
  - 初始化
  - 比较
  - 指令执行体
  - 递增或递减

识别循环

```
int i;

for(i=0; i<100; i++)
{
   printf("i equals %d\n", i);
}
```

```
00401004        mov     [ebp+var_4], 0 ❶
0040100B        jmp     short loc_401016 ❷
0040100D loc_40100D:
0040100D        mov     eax, [ebp+var_4] ❸
00401010        add     eax, 1
00401013        mov     [ebp+var_4], eax ❹
00401016 loc_401016:
00401016        cmp     [ebp+var_4], 64h ❺
0040101A        jge     short loc_40102F ❻
0040101C        mov     ecx, [ebp+var_4]
0040101F        push    ecx
00401020        push    offset aID  ; "i equals %d\n"
00401025        call    printf
0040102A        add     esp, 8
0040102D        jmp     short loc_40100D ❼
```

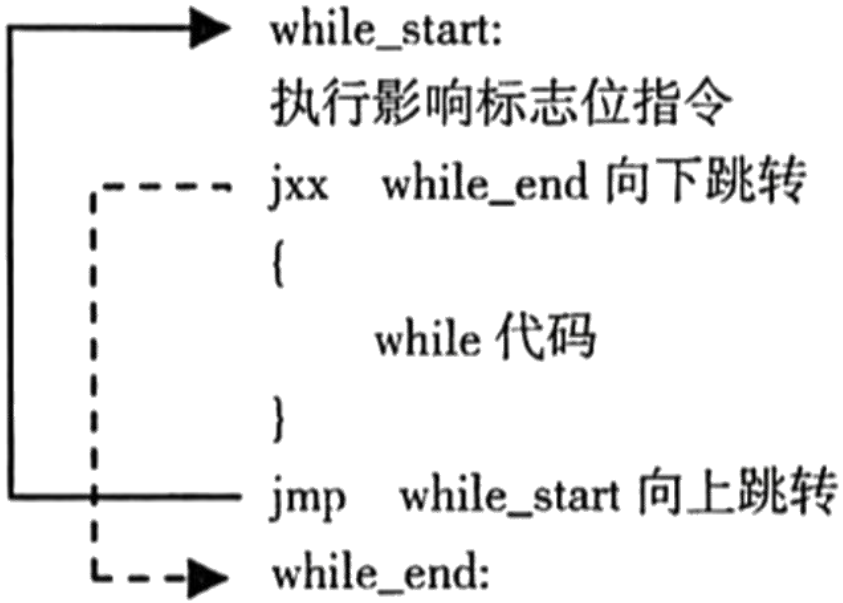


While循环

```
int status=0;
int result = 0;

while(status == 0){
     result = performAction();
     status = checkResult(result);
}
```

```
00401036          mov     [ebp+var_4], 0
0040103D          mov     [ebp+var_8], 0
00401044 loc_401044:
00401044          cmp     [ebp+var_4], 0
00401048          jnz     short loc_401063 ❶
0040104A          call    performAction
0040104F          mov     [ebp+var_8], eax
00401052          mov     eax, [ebp+var_8]
00401055          push    eax
00401056          call    checkResult
0040105B          add     esp, 4
0040105E          mov     [ebp+var_4], eax
00401061          jmp     short loc_401044 ❷
```

While循环的识别特征



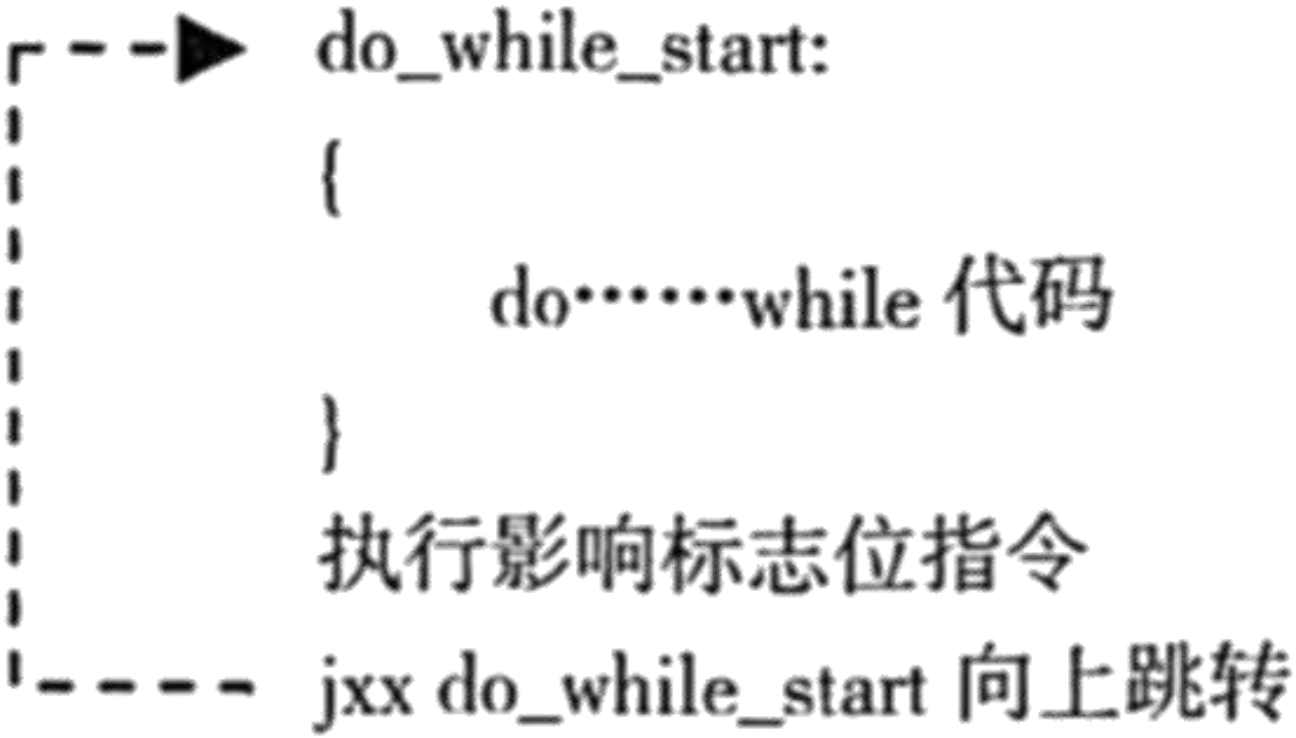
Do循环

```
#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[]) {
    int nCount = 0;
    do
    {
        printf("%d\r\n", nCount);
        nCount++;
    } while (nCount < argc);

    return 0;
}
```

Do循环

```
mov     edx, [rsp+20h]          ;参数 2: edx=nCount
lea     rcx, asc_14000678C      ;参数 1: "%d\r\n"
call    cs:printf               ;调用 printf 函数
mov     eax, [rsp+20h]
inc     eax
mov     [rsp+20h], eax          ;nCount=nCount+1
mov     eax, [rsp+40h]          ;eax=argc
cmp     [rsp+20h], eax
jl      short loc_140001039 ;if(nCount<argc)，跳转到 do 循环开始
```

Do循环的识别特征



本章知识点

1. 识别函数
2. 识别变量、数组、结构体
3. 识别IF分支结构
4. 识别Switch结构
5. 识别循环结构

汇编语言与逆向技术

第9章 C语言程序逆向分析

王志
zwang@nankai.edu.cn

南开大学 网络空间安全学院

2021-2022学年