

电 子 科 技 大 学

软件开发环境实验二：函数调用栈帧布局

学生姓名：任振华 学号：2017060801023

指导教师：李林 实验时间：2019/12/22

1 实验目的

本实验总体目的是，通过使用 Visual Studio 2017 查看函数调用时参数、局部变量等在栈上的分布情况，以达到掌握函数调用时栈帧布局的目的。

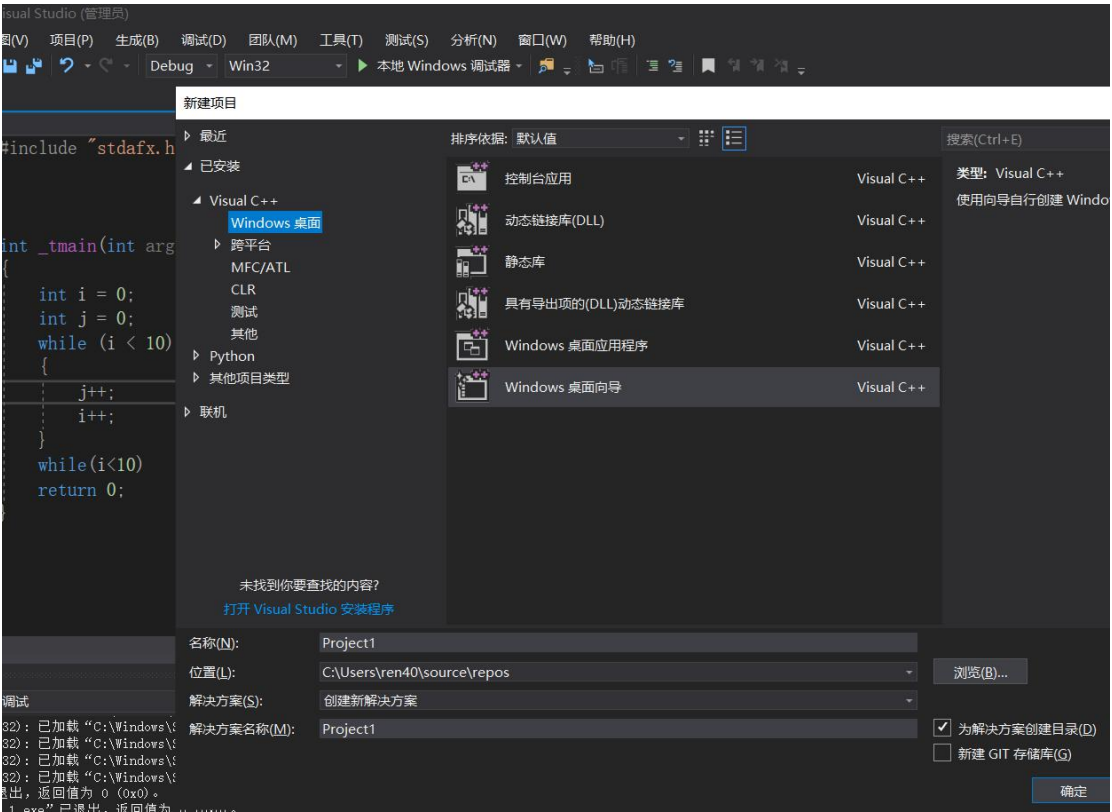
2 实验原理

编译器对函数调用的支持，通常情况下都会使用栈。例如使用栈传递传参，保存函数返回地址。另外，局部变量也通常位于栈上。Visual Studio 2017 为了防止栈上局部数组溢出，又采取了特殊的保护措施。本实验就需要通过观察栈帧布局，来了解这些保护措施。

本实验的环境是 Visual Studio 2017。

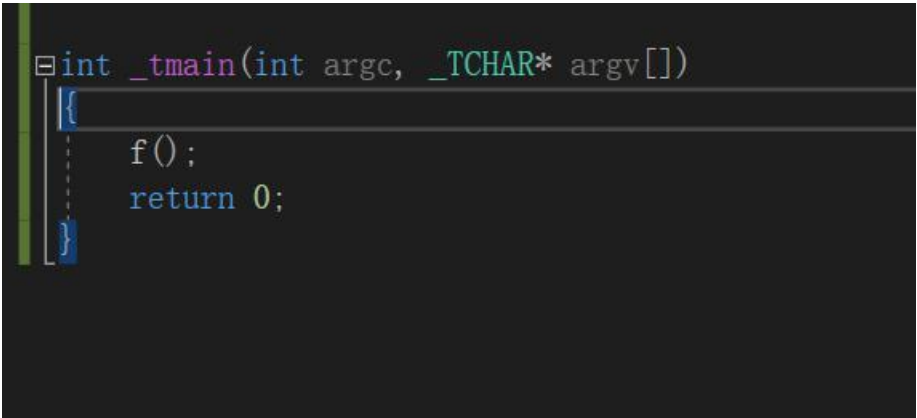
3 实验步骤及要求

3.1 工程的创建



3.2.1 函数中无任何局部变量的情况

代码清单 1



```

void f()
{
00401470  push      ebp      已用时间 <= 1ms
00401471  mov       ebp, esp
00401473  sub       esp, 0C0h
00401479  push      ebx
0040147A  push      esi
0040147B  push      edi
0040147C  lea       edi, [ebp-0C0h]
00401482  mov       ecx, 30h
00401487  mov       eax, 0CCCCCCCCh
0040148C  rep stos   dword ptr es:[edi]

}
0040148E  pop       edi
0040148F  pop       esi
00401490  pop       ebx
00401491  mov       esp, ebp
00401493  pop       ebp
00401494  ret

```

```

void f()
{
00401470  push      ebp
将ebp寄存器的值存入栈中
00401471  mov       ebp, esp
将esp寄存器的值存入ebp寄存器
00401473  sub       esp, 0C0h
sub指令是减法指令，该指令是将esp-12，再把结果送到esp
00401479  push      ebx
将ebx出栈
0040147A  push      esi
；将esi压栈
0040147B  push      edi
；将edi压栈
0040147C  lea       edi, [ebp-0C0h]
；lea是取地址指令，该指令使edi = [ebp-0C0h]
00401482  mov       ecx, 30h
；使寄存器ecx的值为30h
00401487  mov       eax, 0CCCCCCCCh
；使寄存器eax的值为0CCCCCCCCh
0040148C  rep stos   dword ptr es:[edi]
；rep是重复前缀，stos指令是将eax的值放入[edi]（es:edi所指向的地址）中，
之后在这里将edi+4。rep stos 是循环指向stos，直到ecx为0，每次循环ecx都会
减1

```

```

}
0040148E pop      edi
; 将edi寄存器出栈
0040148F pop      esi
; 将esi寄存器出栈
00401490 pop      ebx
; 将ebp寄存器出栈
00401491 mov      esp, ebp
; 把ebp寄存器的值送到esp中
00401493 pop      ebp
将ebp寄存器出栈
00401494 ret
; 结束 f() 函数，返回到调用源

```

根据代码清单 1 的反汇编代码，以及内存映像（调式->窗口->内存）可画出栈帧的布局，即图 2。

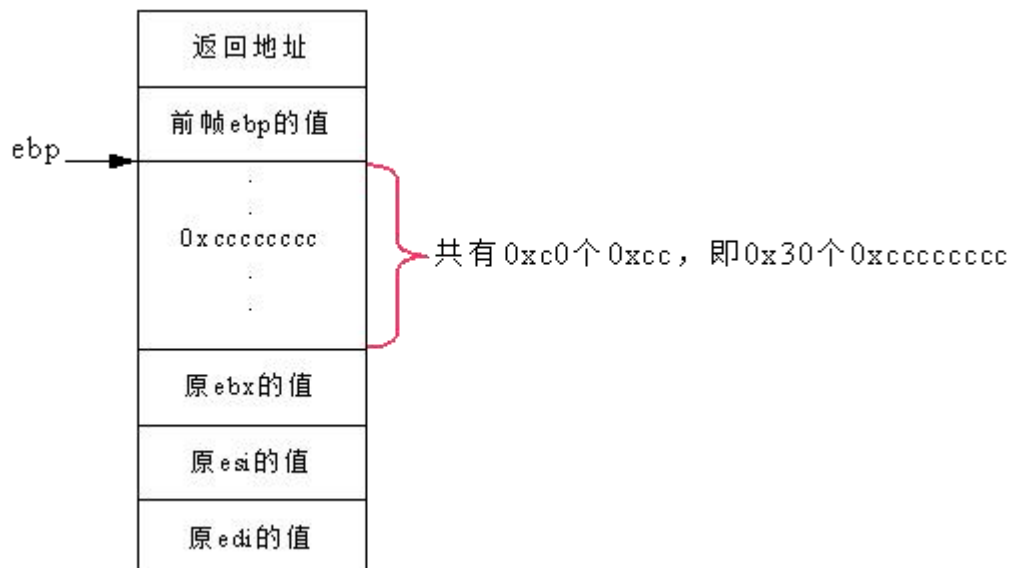


图 2 无局部变量时的布局

从图 2 可以知道，即使没有任何局部变量，栈上仍然有 0xc0 个字节的 0xcc 存在。这个是为了检测是否有溢出而写的。

3.2.2 函数中只有一个局部变量的情况

代码清单 3

```
void f()
{
    int i = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

```
void f()
{
00401470  push      ebp
00401471  mov       ebp, esp
00401473  sub       esp, 0CCh
00401479  push      ebx
0040147A  push      esi
0040147B  push      edi
0040147C  lea       edi, [ebp-0CCh]
00401482  mov       ecx, 33h
00401487  mov       eax, 0CCCCCCCCh
0040148C  rep stos  dword ptr es:[edi]
    int i = 0;
0040148E  mov       dword ptr [i], 0
}
00401495  pop       edi
00401496  pop       esi
00401497  pop       ebx
00401498  mov       esp, ebp
0040149A  pop       ebp
0040149B  ret
-- 无源文件 --
```

```
void f()
{
00401470  push      ebp
;将ebp寄存器压栈
```

```

00401471  mov          ebp, esp
;将esp寄存器的数据送到ebp
00401473  sub          esp, 0CCh
;sub是减法指令, esp = esp - 12
00401479  push         ebx
;将ebx寄存器压栈
0040147A  push         esi
;将esi寄存器压栈
0040147B  push         edi
;将edi寄存器压栈
0040147C  lea          edi, [ebp-0CCh]
;lea是取地址指令, edi = ebp - 0CCh
00401482  mov          ecx, 33h
;使exc = 33h
00401487  mov          eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos     dword ptr es:[edi]
;rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为, 每次循环ecx都会
减1
    int i = 0;
0040148E  mov          dword ptr [i], 0
; 使 i = 0
}
00401495  pop          edi
; 将edi寄存器出栈
00401496  pop          esi
; 将esi寄存器出栈
00401497  pop          ebx
; 将ebx寄存器出栈
00401498  mov          esp, ebp
; 使esp = ebp
0040149A  pop          ebp
; 将ebp寄存器出栈
0040149B  ret
; 结束 f() 函数, 返回到调用源

```

根据代码清单 4, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局, 即图 4。

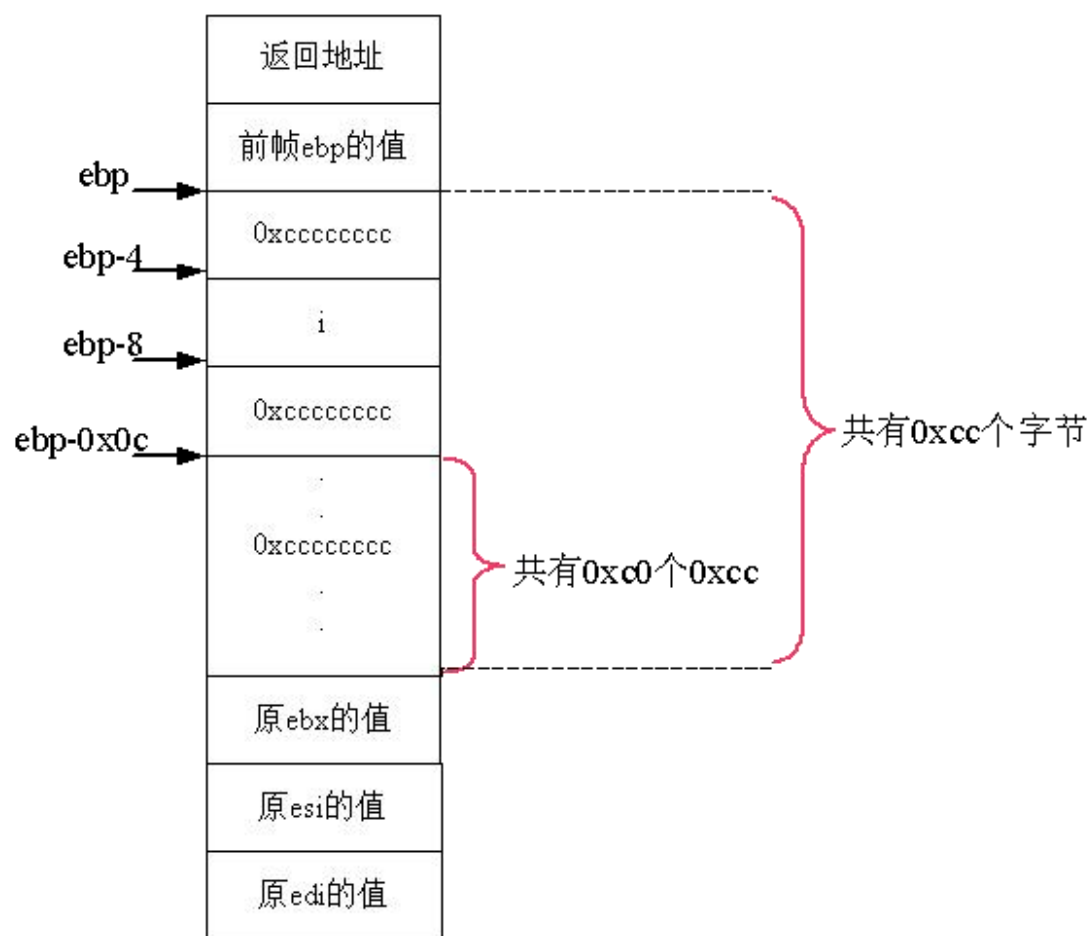


图 4 只有一个整型局部变量时的布局

代码清单 5

```

void f()
{
    char i = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}

```

```

void f()
{
00401470  push      ebp
00401471  mov       ebp, esp
00401473  sub       esp, 0CCh
00401479  push      ebx
0040147A  push      esi
0040147B  push      edi
0040147C  lea       edi, [ebp+FFFFFF34h]
00401482  mov       ecx, 33h
00401487  mov       eax, 0CCCCCCCCh
0040148C  rep stos  dword ptr es:[edi]
    char i = 0;
0040148E  mov       byte ptr [ebp-5], 0
}
00401492  pop       edi
00401493  pop       esi
00401494  pop       ebx
00401495  mov       esp, ebp
00401497  pop       ebp
00401498  ret

```

```

void f()
{
00401470  push      ebp
;将ebp寄存器压栈
00401471  mov       ebp, esp
;使ebp = esp
00401473  sub       esp, 0CCh
;sub是减法指令，使esp = esp - 0CCh
00401479  push      ebx
;将ebx寄存器压栈
0040147A  push      esi
;将esi寄存器压栈
0040147B  push      edi
;将edi寄存器压栈
0040147C  lea       edi, [ebp+FFFFFF34h]
;lea是取地址指令，这条指令使edi = ebp + FFFFFFF34h
00401482  mov       ecx, 33h
;使ecx = 33h
00401487  mov       eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos  dword ptr es:[edi]
;rep是重复前缀，stos指令是将eax的值放入[edi]（es:edi所指向的地址）中，
之后在这里将edi+4。rep stos 是循环指向stos，直到ecx为0，每次循环ecx都会

```



```

char i = 0;
0040148E  mov             byte ptr [ebp-5], 0
;使*(ebp - 5) = 0, byte是字节, ptr是属性修饰符
}
00401492  pop            edi
;将edi寄存器出栈
00401493  pop            esi
;将esi寄存器出栈
00401494  pop            ebx
;将ebx寄存器出栈
00401495  mov            esp, ebp
; 使esp = ebp
00401497  pop            ebp
; 将ebp寄存器出栈
00401498  ret
; 结束 f() 函数, 返回到调用源

```

Diagram illustrating the stack frame structure and memory layout:

- Return Address (返回地址)
- Previous frame's `ebp` value (前帧`ebp`的值)
- Memory area containing `0xcccccccc`, `i`, and several `0xcc` bytes.
- Memory area containing `0xcccccccc`, `...`, `0xcccccccc`, and `...`.
- Original `ebp` value (原`ebp`的值)
- Original `esi` value (原`esi`的值)
- Original `edi` value (原`edi`的值)

Annotations and Calculations:

- `ebp` points to the start of the previous frame's `ebp` value.
- `ebp-4` points to the variable `i`.
- `ebp-8` points to the first `0xcc` byte.
- `ebp-0xc` points to the `0xcccccccc` byte.
- The first group of bytes (from `0xcccccccc` down to the first `0xcccccccc`) contains `0xcc` bytes.
- The second group of bytes (from the first `0xcccccccc` down to the second `0xcccccccc`) contains `0xc0` `0xcc` bytes.

图 5 只有一个 char 型局部变量时的布局

代码清单 6

```
#include "stdafx.h"

void f()
{
    short i = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

```
void f()
{
00401470  push     ebp
00401471  mov      ebp, esp
00401473  sub      esp, 0CCh
00401479  push     ebx
0040147A  push     esi
0040147B  push     edi
0040147C  lea      edi, [ebp+FFFFFF34h]
00401482  mov      ecx, 33h
00401487  mov      eax, 0CCCCCCCCh
0040148C  rep stos dword ptr es:[edi]
    short i = 0;
0040148E  xor      eax, eax
00401490  mov      word ptr [ebp-8], ax
}
00401494  pop      edi
00401495  pop      esi
00401496  pop      ebx
00401497  mov      esp, ebp
00401499  pop      ebp
0040149A  ret
```

```
void f()
{
00401470  push     ebp
```

```

;将ebp寄存器压栈
00401471  mov          ebp, esp
;使ebp = esp
00401473  sub          esp, 0CCh
; sub是减法指令, 使esp = esp - 0CCh
00401479  push         ebx
;将ebx寄存器压栈
0040147A  push         esi
;将esi寄存器压栈
0040147B  push         edi
;将edi寄存器压栈
0040147C  lea          edi, [ebp+FFFFFF34h]
; lea是取地址指令, 这条指令使edi = ebp + FFFFFFF34h
00401482  mov          ecx, 33h
;使ecx = 33h
00401487  mov          eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos     dword ptr es:[edi]
; rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
; 之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为0, 每次循环ecx都会
; 减1
        short i = 0;
0040148E  xor          eax, eax
;使eax寄存器清0
00401490  mov          word ptr [ebp-8], ax
;使*(ebp-8) = ax , 而eax寄存器已经被清0
}
00401494  pop          edi
;将edi寄存器出栈
00401495  pop          esi
;将esi寄存器出栈
00401496  pop          ebx
;将ebx寄存器出栈
00401497  mov          esp, ebp
;使esp = ebp
00401499  pop          ebp
;将ebp寄存器出栈
0040149A  ret
; 结束 f() 函数, 返回到调用源

```

根据代码清单 6, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局, 即图 6

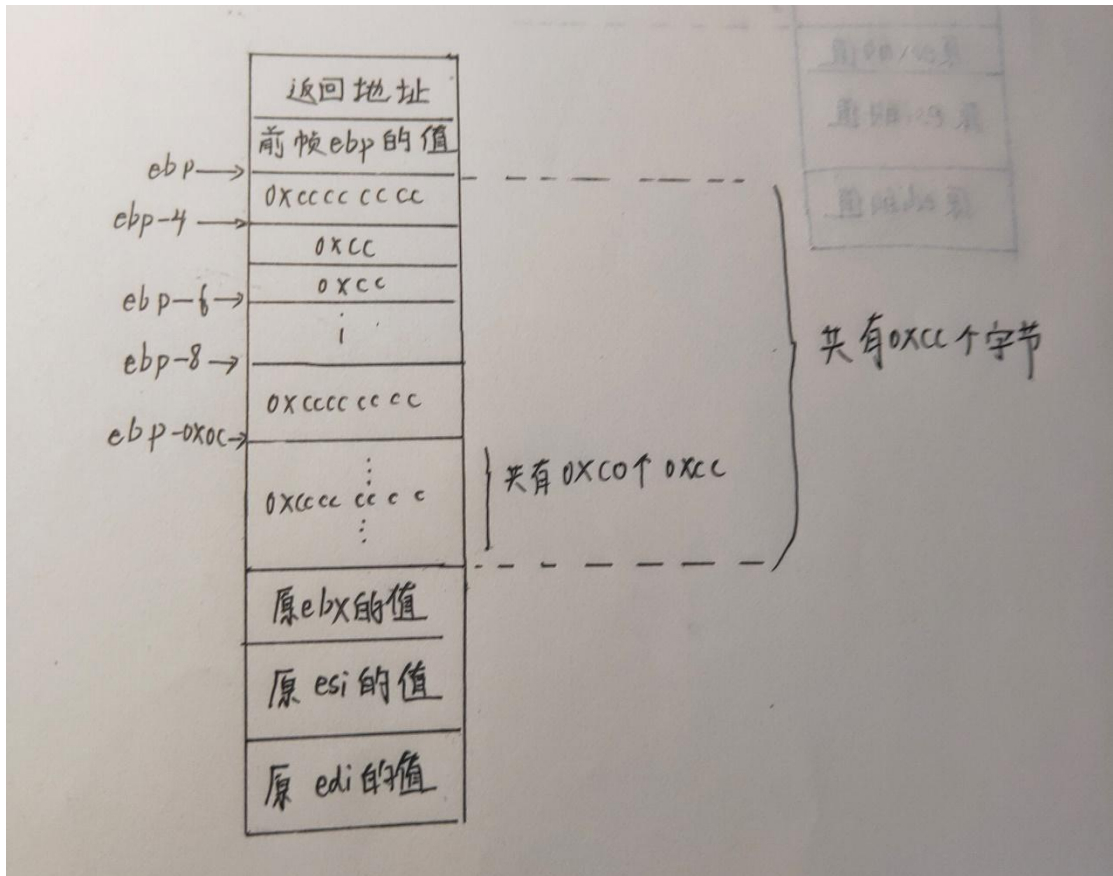


图 6 只有一个 short 型局部变量时的布局

代码清单 7

```
#include "stdafx.h"

void f()
{
    double i = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

```
▼ 查看选项
void f()
{
00401470 push      ebp      已用时间 <= 1ms
00401471 mov       ebp, esp
00401473 sub       esp, 0D0h
00401479 push      ebx
0040147A push      esi
0040147B push      edi
0040147C lea       edi, [ebp+FFFFFF30h]
00401482 mov       ecx, 34h
00401487 mov       eax, 0CCCCCCCCh
0040148C rep stos   dword ptr es:[edi]
    double i = 0;
0040148E xorps     xmm0, xmm0
00401491 movsd    mmword ptr [ebp-0Ch], xmm0
}
00401496 pop       edi
00401497 pop       esi
00401498 pop       ebx
00401499 mov       esp, ebp
0040149B pop       ebp
0040149C ret
```

```
void f()
{
00401470 push      ebp
;ebp寄存器压栈
00401471 mov       ebp, esp
;使ebp = esp
00401473 sub       esp, 0D0h
; sub是减法指令，使esp = esp - 0D0h
00401479 push      ebx
;将ebx寄存器压栈
0040147A push      esi
;将esi寄存器压栈
0040147B push      edi
;将edi寄存器压栈
0040147C lea       edi, [ebp+FFFFFF30h]
; lea是取地址指令，这条指令使edi = ebp + FFFFFFF30h
00401482 mov       ecx, 34h
;使ecx = 34h
00401487 mov       eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C rep stos   dword ptr es:[edi]
; rep是重复前缀，stos指令是将eax的值放入[edi]（es:edi所指向的地址）中，
之后在这里将edi+4。rep stos 是循环指向stos，直到ecx为0，每次循环ecx都会
减1
    double i = 0;
0040148E xorps     xmm0, xmm0
```

;xorps—压缩单精度浮点值的按位逻辑异或, xmm0 是 SSE 的 128 位寄存器, 该指令使 xmm0 寄存器清 0

```
00401491 movsd      mmword ptr [ebp-0Ch], xmm0
```

;movsd是数据传送指令, 传送一个双字

;mmword用于具有MMX和SSE(XMM) 命令的64位多媒体操作数

}

```
00401496 pop      edi
```

;将edi寄存器出栈

```
00401497 pop      esi
```

;将esi寄存器出栈

```
00401498 pop      ebx
```

;将ebx寄存器出栈

```
00401499 mov      esp, ebp
```

;使esp = ebp

```
0040149B pop      ebp
```

将ebp寄存器出栈

```
0040149C ret
```

; 结束 f() 函数, 返回到调用源

根据代码清单 7, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局, 即图 7

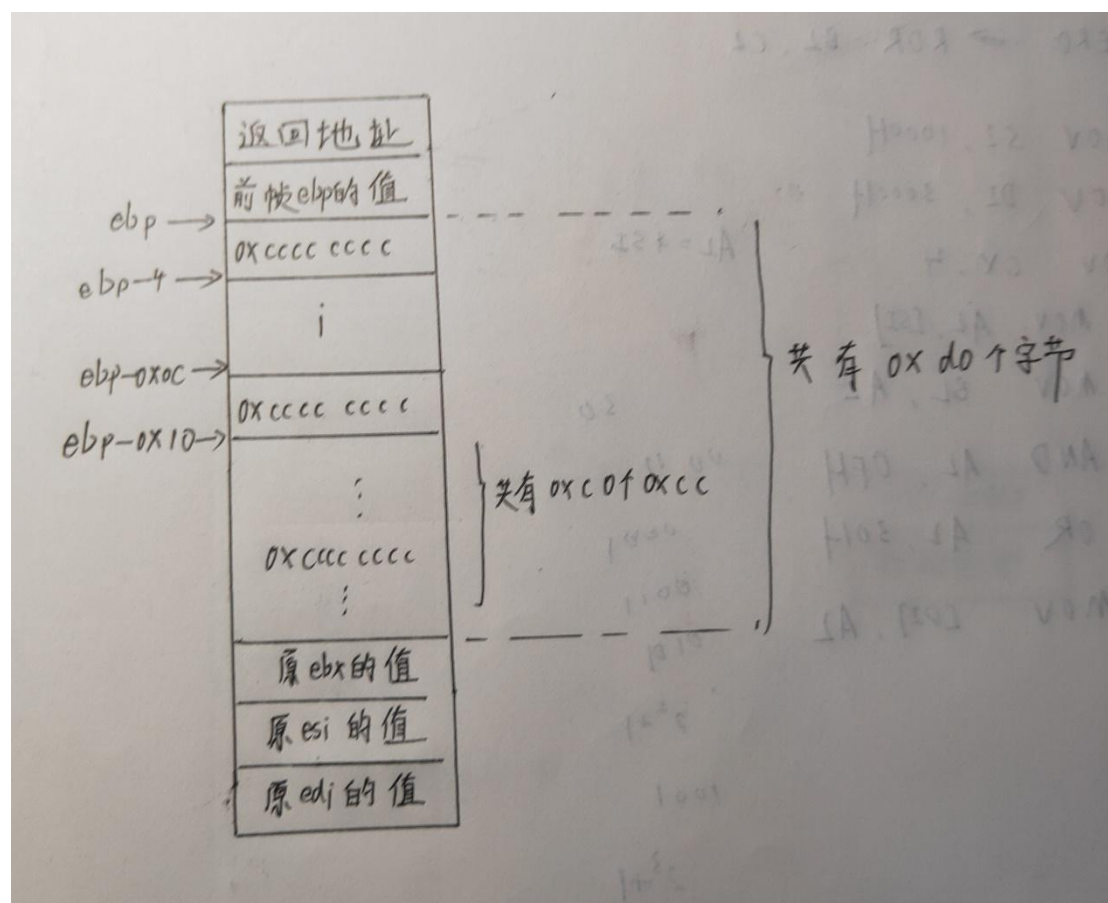


图 7 只有一个 double 型局部变量时的布局

3.2.3 函数中有多个局部变量的情况

代码清单 8

```
3
4     #include "stdafx.h"
5
6     void f()
7     {
8         char i = 0;
9         int j = 0;
10    }
11
12    int _tmain(int argc, _TCHAR* argv[])
13    {
14        f();
15        return 0;
16    }
```

查看选项

```
void f()
{
00401473  sub     esp, 0D8h
00401479  push    ebx
0040147A  push    esi
0040147B  push    edi
0040147C  lea     edi, [ebp+FFFFFF28h]
00401482  mov     ecx, 36h
00401487  mov     eax, 0CCCCCCCCh
0040148C  rep stos dword ptr es:[edi]
    char i = 0;
0040148E  mov     byte ptr [ebp-5], 0
    int j = 0;
00401492  mov     dword ptr [ebp-14h], 0
}
00401499  pop     edi
0040149A  pop     esi
0040149B  pop     ebx
0040149C  mov     esp, ebp
0040149E  pop     ebp
0040149F  ret
```

```
void f()
{
00401473  sub     esp, 0D8h
; sub是减法指令，使esp = esp - 0D8h
```

```

00401479  push        ebx
;将ebx寄存器入栈
0040147A  push        esi
;将esi寄存器压栈
0040147B  push        edi
;将edi寄存器压栈
0040147C  lea         edi, [ebp+FFFFFF28h]
; lea是取地址指令，这条指令使edi = ebp + FFFFFFF28h
00401482  mov         ecx, 36h
;使ecx = 36h
00401487  mov         eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos    dword ptr es:[edi]
; rep是重复前缀，stos指令是将eax的值放入[edi]（es:edi所指向的地址）中，
之后在这里将edi+4。rep stos 是循环指向stos，直到ecx为0，每次循环ecx都会
减1
    char i = 0;
0040148E  mov         byte ptr [ebp-5], 0
;这条指令使 *(ebp-5) = 0, byte是字节
    int j = 0;
00401492  mov         dword ptr [ebp-14h], 0
;这条指令使*(ebp-14h) = 0, dword是双字，即四字节
}
00401499  pop         edi
;将edi寄存器出栈
0040149A  pop         esi
;将esi寄存器出栈
0040149B  pop         ebx
;将ebx寄存器出栈
0040149C  mov         esp, ebp
;使esp = ebp
0040149E  pop         ebp
;将ebp寄存器出栈
0040149F  ret
; 结束 f() 函数，返回到调用源

```

根据代码清单 8，以及内存映像（调式->窗口->内存）可画出栈帧的布局，即图 8。

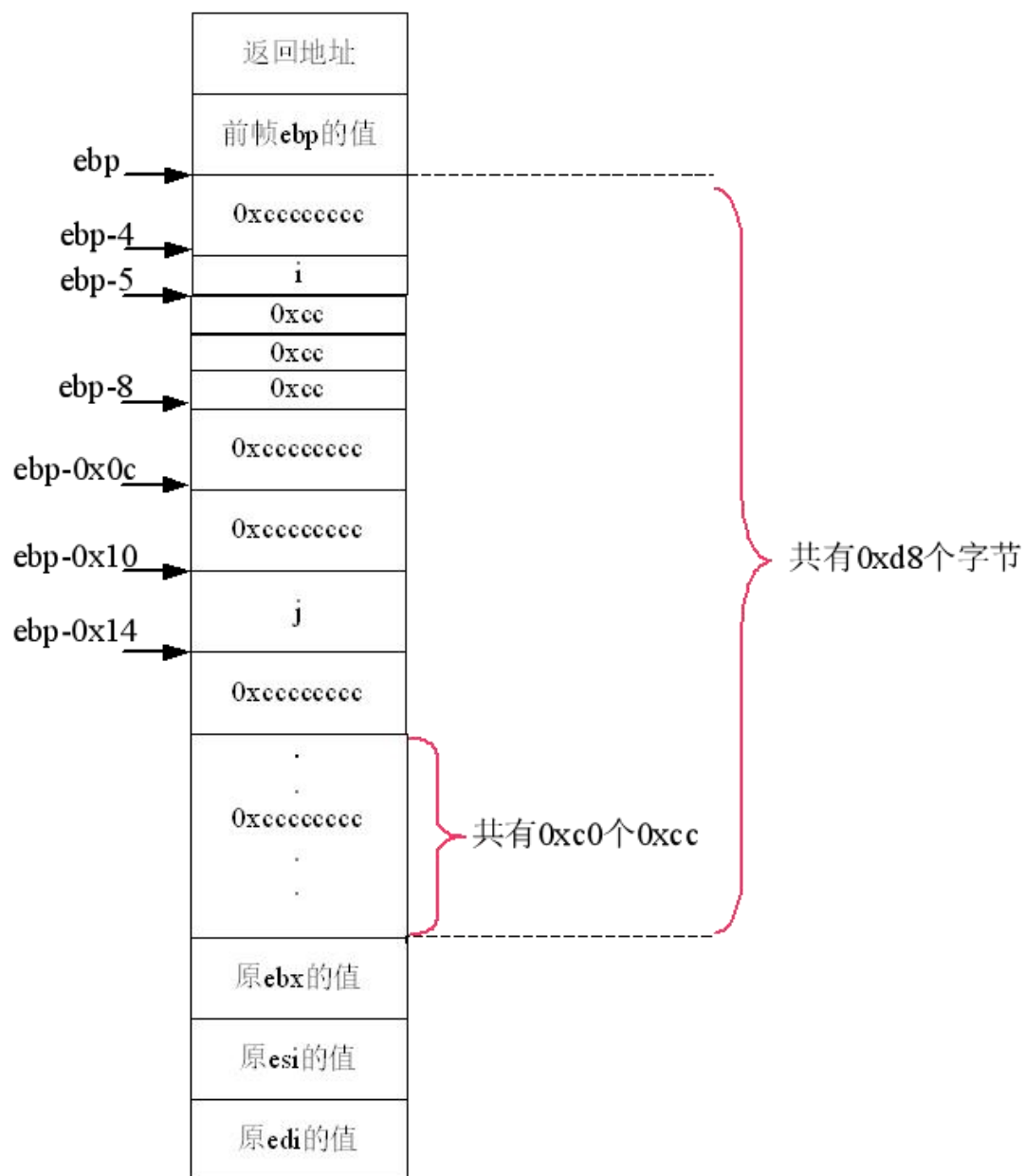


图 8 代码清单 8 对应的栈帧布局

代码清单 10

```
5
6 void f()
7 {
8     char i = 0;
9     short j = 0;
10 }
11
12 int _tmain(int argc, _TCHAR* argv[])
13 {
14     f();
15     return 0;
16 }
```

```
void f()
{
00401470 push     ebp    已用时间 <= 1ms
00401471 mov     ebp, esp
00401473 sub     esp, 0D8h
00401479 push     ebx
0040147A push     esi
0040147B push     edi
0040147C lea     edi, [ebp+FFFFFF28h]
00401482 mov     ecx, 36h
00401487 mov     eax, 0CCCCCCCCh
0040148C rep stos dword ptr es:[edi]
    char i = 0;
0040148E mov     byte ptr [ebp-5], 0
    short j = 0;
00401492 xor     eax, eax
00401494 mov     word ptr [ebp-14h], ax
}
00401498 pop     edi
00401499 pop     esi
0040149A pop     ebx
0040149B mov     esp, ebp
0040149D pop     ebp
0040149E ret
```

```
void f()
{
00401470 push     ebp
;将ebp寄存器压栈
```

```

00401471  mov          ebp, esp
;使ebp = esp
00401473  sub          esp, 0D8h
; sub是减法指令, 使esp = esp - 0D8h
00401479  push         ebx
;将ebx寄存器压栈
0040147A  push         esi
;将esi寄存器压栈
0040147B  push         edi
;将edi寄存器压栈
0040147C  lea          edi, [ebp+FFFFFF28h]
; lea是取地址指令, 这条指令使edi = ebp + FFFFFFF28h
00401482  mov          ecx, 36h
;使ecx = 36h
00401487  mov          eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos     dword ptr es:[edi]
; rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为0, 每次循环ecx都会
减1
    char i = 0;
0040148E  mov          byte ptr [ebp-5], 0
;使*(ebp-5) = 0, byte是字节
    short j = 0;
00401492  xor          eax, eax
;xor是异或指令, 使eax为0
00401494  mov          word ptr [ebp-14h], ax
;使*(ebp-14h) = ax, 而ax为0
}
00401498  pop          edi
;将edi寄存器出栈
00401499  pop          esi
;将esi寄存器出栈
0040149A  pop          ebx
;将ebx寄存器出栈
0040149B  mov          esp, ebp
;使esp = ebp
0040149D  pop          ebp
;将ebp寄存器出栈
0040149E  ret
; 结束 f() 函数, 返回到调用源

```

根据代码清单 10, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局, 即图 10。

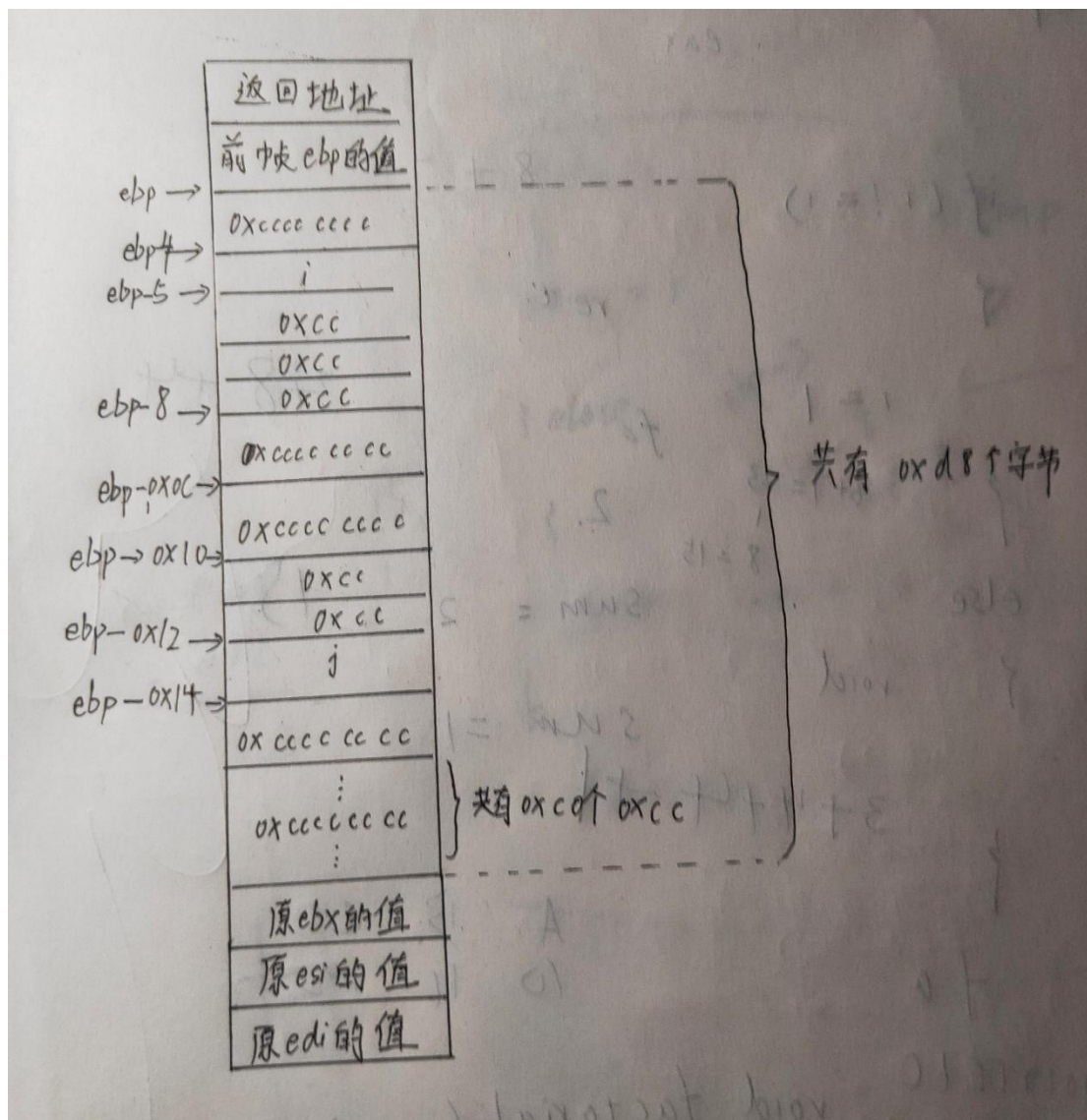


图 10 代码清单 10 对应的栈帧布局

代码清单 11

```
void f()
{
    char i = 0;
    char j = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

地址(A): f(void)

查看选项

```
void f()
{
00401470  push     ebp
00401471  mov      ebp, esp
00401473  sub      esp, 0D8h
00401479  push     ebx
0040147A  push     esi
0040147B  push     edi
0040147C  lea      edi, [ebp+FFFFFF28h]
00401482  mov      ecx, 36h
00401487  mov      eax, 0CCCCCCCCh
0040148C  rep stos dword ptr es:[edi]
    char i = 0;
0040148E  mov      byte ptr [ebp-5], 0
    char j = 0;
00401492  mov      byte ptr [ebp-11h], 0
}
➔ 00401496  pop      edi    已用时间 <= 1ms
00401497  pop      esi
00401498  pop      ebx
00401499  mov      esp, ebp
0040149B  pop      ebp
0040149C  ret
```

```
void f()
{
00401470  push     ebp
;将ebp寄存器入栈
00401471  mov      ebp, esp
;使ebp = esp
00401473  sub      esp, 0D8h
```

```

; sub是减法指令, 使esp = esp - 0D8h
00401479  push      ebx
;将ebx寄存器压栈
0040147A  push      esi
;将esi寄存器压栈
0040147B  push      edi
;将edi寄存器压栈
0040147C  lea        edi, [ebp+FFFFFF28h]
; lea是取地址指令, 这条指令使edi = ebp + FFFFFFF28h
00401482  mov        ecx, 36h
;使ecx = 36h
00401487  mov        eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos   dword ptr es:[edi]
; rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为0, 每次循环ecx都会
减1
    char i = 0;
0040148E  mov        byte ptr [ebp-5], 0
;使*(ebp-5) = 0, byte是字节
    char j = 0;
00401492  mov        byte ptr [ebp-11h], 0
;使*(ebp-11h) = 0, byte是字节
}
00401496  pop        edi
;将edi寄存器出栈
00401497  pop        esi
;将esi寄存器出栈
00401498  pop        ebx
;将ebx寄存器出栈
00401499  mov        esp, ebp
;使esp = ebp
0040149B  pop        ebp
;将ebp寄存器出栈
0040149C  ret
; 结束 f() 函数, 返回到调用源

```

根据代码清单 10, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局, 即图 10。

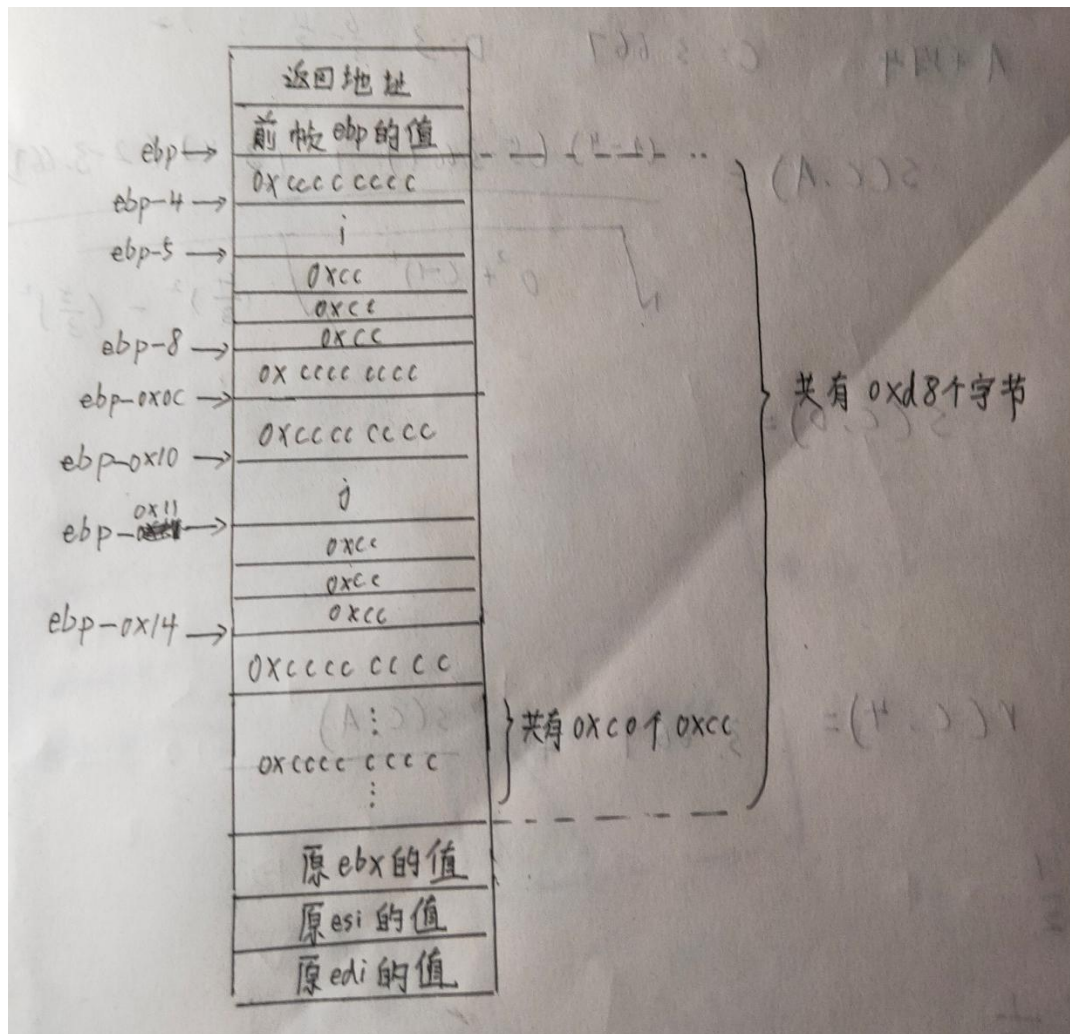


图 11 代码清单 11 对应的栈帧布局

代码清单 12

```
3
4     #include "stdafx.h"
5
6     void f()
7     {
8         char i = 0;
9         int j = 0;
10        short k = 0;
11    }
12
13    int _tmain(int argc, _TCHAR* argv[])
14    {
15        f();
16        return 0;
17    }
```

```
void f()
{
00401470  push     ebp
00401471  mov      ebp, esp
00401473  sub      esp, 0E4h
00401479  push     ebx
0040147A  push     esi
0040147B  push     edi
0040147C  lea      edi, [ebp+FFFFFF1Ch]
00401482  mov      ecx, 39h
00401487  mov      eax, 0CCCCCCCCh
0040148C  rep stos dword ptr es:[edi]
    char i = 0;
0040148E  mov      byte ptr [ebp-5], 0
    int j = 0;
00401492  mov      dword ptr [ebp-14h], 0
    short k = 0;
00401499  xor      eax, eax
0040149B  mov      word ptr [ebp-20h], ax
}
* 0040149F  pop      edi    已用时间 <= 1ms
004014A0  pop      esi
004014A1  pop      ebx
004014A2  mov      esp, ebp
004014A4  pop      ebp
004014A5  ret
```

```
void f()
{
00401470  push     ebp
;将ebp寄存器入栈
00401471  mov      ebp, esp
```



```

;使ebp = esp
00401473  sub          esp, 0E4h
;sub是减法指令, 使esp = esp - 0E4h
00401479  push         ebx
;将ebx寄存器压栈
0040147A  push         esi
;将esi寄存器压栈
0040147B  push         edi
;将edi寄存器压栈
0040147C  lea          edi, [ebp+FFFFFF1Ch]
;lea是取地址指令, 这条指令使edi = ebp + FFFFFFF1Ch
00401482  mov          ecx, 39h
;使ecx = 39h
00401487  mov          eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
0040148C  rep stos     dword ptr es:[edi]
;rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为0, 每次循环ecx都会
减1
    char i = 0;
0040148E  mov          byte ptr [ebp-5], 0
;使*(ebp-5) = 0, byte是字节
    int j = 0;
00401492  mov          dword ptr [ebp-14h], 0
;使*(ebp-14h) = 0
    short k = 0;
00401499  xor          eax, eax
;xor是异或指令, 使eax = 0
0040149B  mov          word ptr [ebp-20h], ax
;使*(ebp-20h) = ax
}
0040149F  pop          edi
;将edi寄存器出栈
004014A0  pop          esi
;将esi寄存器出栈
004014A1  pop          ebx
;将ebx寄存器出栈
004014A2  mov          esp, ebp
;使esp = ebp
004014A4  pop          ebp
;将ebp寄存器出栈
004014A5  ret
; 结束 f() 函数, 返回到调用源

```

根据代码清单 12, 以及内存映像 (调式->窗口->内存) 可画出栈帧的布局,

即图 12。

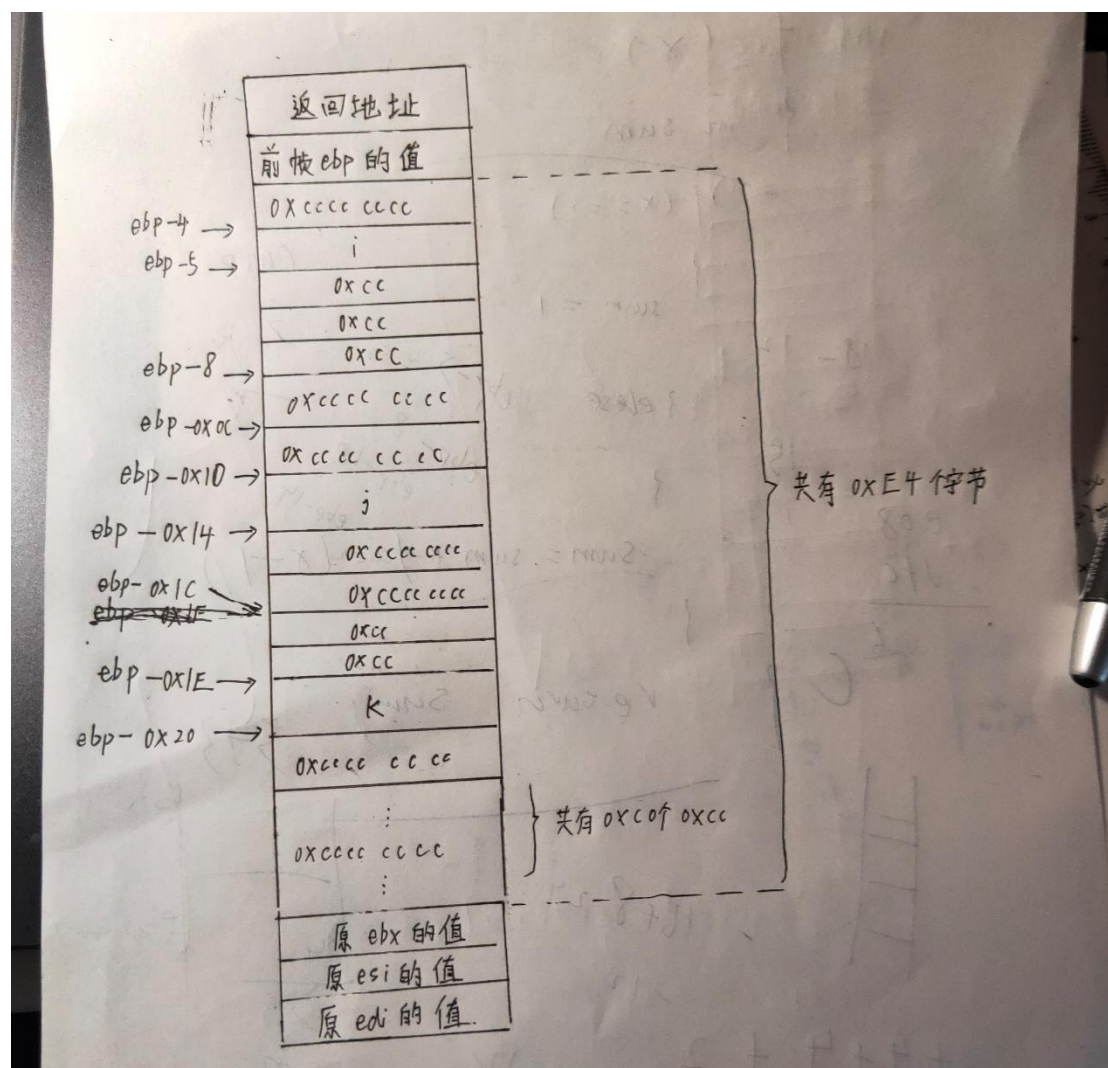


图 12 代码清单 12 对应的栈帧布局

栈帧布局的规律总结

1. 在函数栈帧结构中 `ebp` 是栈底，即高地址，`esp` 是栈顶，即低地址。
2. 任何函数都只有一份 `ebp` 和 `esp`，可以通过内存窗口发现 `ebp` 和 `esp` 永远保存最新当前函数的数值。用函数时，原函数的 `ebp` 和 `esp` 要保存起来，以便返回时恢复。
3. 在栈帧布局中，局部变量两边都有 `0xcccccccc`，以 4 字节来看，`char` 型具备变量占据了 4 字节的最高一个字节的的地方，`short` 型占据了 4 字节低位的两个字节。

3.3 函数中有局部数组的情况

代码清单 13

```
#include "stdafx.h"

void f()
{
    char i = 0;
    char a[20];
    a[0] = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

00403FAF int 3
--- d:\agent_work\3\s\src\vc\tools\crt\vcstartup\src\eh\i386\secchk.c -----
★ 00403FB0 cmp ecx,dword ptr [__security_cookie (0409024h)] 已用时间 <= 1ms
00403FB6 bnd jne failure (0403FBBh)
00403FB9 bnd ret
00403FBB bnd jmp ___report_gsfailure (0401005h)
--- 无源文件 -----
00403FC1 int 3
00403FC2 int 3
00403FC3 int 3
00403FC4 int 3
00403FC5 int 3
00403FC6 int 3

名称	值	类型
ecx	237977300	unsigned int
cookie	237977300	unsigned int

调用堆栈

- 1.9.exe!_security_check
- 1.9.exelf() 行 11
- 1.9.exe!wmain(int argc, ...)
- [外部代码]
- [下面的框架可能不正...

这时候 ecx 的值同__security_cookie 一致

```
#include "stdafx.h"

void f()
{
    char i = 0;
    char a[20];
    a[0] = 0;
    a[20] = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

在函数添加 `a[20] = 0;` `a[20]` 可以造成越界写的效果
因为声明变量 `char a[20];`, 访问下标是从 0 到 19, 不包括 20

地址(A): `__report_securityfailure(unsigned long)`

查看选项

00404591	mov	ebp, esp
00404593	sub	esp, 31Ch
463:	if	(IsProcessorFeaturePresent(PF_FASTFAIL_AVAILABLE))
00404599	push	17h
0040459B	call	__IsProcessorFeaturePresent@4 (0404958b)
004045A0	test	eax, eax
004045A2	je	__report_securityfailure+19h (0404558b)
464:	{	
465:	__fastfail(failure_code);	
004045A4	mov	ecx, dword ptr [failure_code]
004045A7	int	29h
466:	}	
467:		
468:		// Set up a fake exception, and report it
469:		// We can't raise a true exception because
470:		// exception handling) can't be trusted.
471:		// if it originated after the call to __report_securityfailure, so it
472:		// is attributed to the function where the violation occurred.
473:		//
474:		// We assume that the immediate caller of __report_securityfailure is
475:		// the function where the security violation occurred. Note that the

未经处理的异常

0x004045A7 处有未经处理的异常(在 1.9.exe 中): RangeChecks 检测代码检测到超出范围的数组访问。

复制详细信息

异常设置

代码清单 16

```
#include "stdafx.h"

void f()
{
    char a[1];
    a[0] = 0;
}

int _tmain(int argc, _TCHAR* argv[])
{
    f();
    return 0;
}
```

```
2: //
3:
4: #include "stdafx.h"
5:
6: void f()
7: {
    00403ED0 push     ebp      已用时间 <= 1ms
    00403ED1 mov     ebp, esp
    00403ED3 sub     esp, 0CCh
    00403ED9 push     ebx
    00403EDA push     esi
    00403EDB push     edi
    00403EDC lea     edi, [ebp-0CCh]
    00403EE2 mov     ecx, 33h
    00403EE7 mov     eax, 0CCCCCCCCh
    00403EEC rep stos  dword ptr es:[edi]
    8:     char a[1];
    9:     a[0] = 0;
    00403EEE mov     eax, 1
    00403EF3 imul    ecx, eax, 0
    00403EF6 mov     byte ptr a[ecx], 0
    10: }
    00403EFB push     edx
    00403EFC mov     ecx, ebp
    00403EFE push     eax
    00403EFF lea     edx, ds:[403F14h]
    00403F05 call    @_RTC_CheckStackVars@8 (0401177h)
    00403F0A pop     eax
    00403F0B pop     edx
    00403F0C pop     edi
    00403F0D pop     esi
    00403F0E pop     ebx
    00403F0F mov     esp, ebp
    00403F11 pop     ebp
    00403F12 ret
```

6: void f()

```

    7: {
00403ED0  push        ebp
;将ebp寄存器入栈
00403ED1  mov         ebp, esp
;使ebp = esp
00403ED3  sub         esp, 0CCh
; sub是减法指令, 使esp = esp - 0CCh
00403ED9  push        ebx
;将ebx寄存器入栈
00403EDA  push        esi
;将esi寄存器压栈
00403EDB  push        edi
;将edi寄存器入栈
00403EDC  lea         edi, [ebp-0CCh]
;lea是取地址指令, 该指令使edi = ebp-0CCh
00403EE2  mov         ecx, 33h
;使ecx = 33h
00403EE7  mov         eax, 0CCCCCCCCh
;使eax = 0CCCCCCCCh
00403EEC  rep stos    dword ptr es:[edi]
; rep是重复前缀, stos指令是将eax的值放入[edi] (es:edi所指向的地址) 中,
; 之后在这里将edi+4。rep stos 是循环指向stos, 直到ecx为0, 每次循环ecx都会减1
    8:      char a[1];
    9:      a[0] = 0;
00403EEE  mov         eax, 1
;使eax = 1
00403EF3  imul        ecx, eax, 0
;imul是有符号乘法指令, 将eax与0相乘放入ecx
00403EF6  mov         byte ptr a[ecx], 0
;使a[0] = 0
    10: }
00403EFB  push        edx
;将edx寄存器压栈
00403EFC  mov         ecx, ebp
;使ecx = ebp
00403EFE  push        eax
;将eax寄存器压栈
00403EFF  lea         edx, ds:[403F14h]
;lea是取地址指令, 使edx = 403F14h, ds是数据段寄存器
00403F05  call        @_RTC_CheckStackVars@8 (0401177h)
; 调用了_RTC_CheckStackVars 函数。该函数的主要任务是检查局部数组是否存在越界写的问题。
00403F0A  pop         eax

```

```

;将eax寄存器出栈
00403F0B pop      edx
;将edx寄存器出栈
00403F0C pop      edi
;将edi寄存器出栈
00403F0D pop      esi
;将esi寄存器出栈
00403F0E pop      ebx
;将ebx寄存器出栈
00403F0F mov      esp, ebp
;使esp = ebp
00403F11 pop      ebp
;将ebp寄存器出栈
00403F12 ret
; 结束 f() 函数，返回到调用源

```

根据代码清单 16，以及内存映像（调式->窗口->内存）可画出栈帧的布局，即图 16。

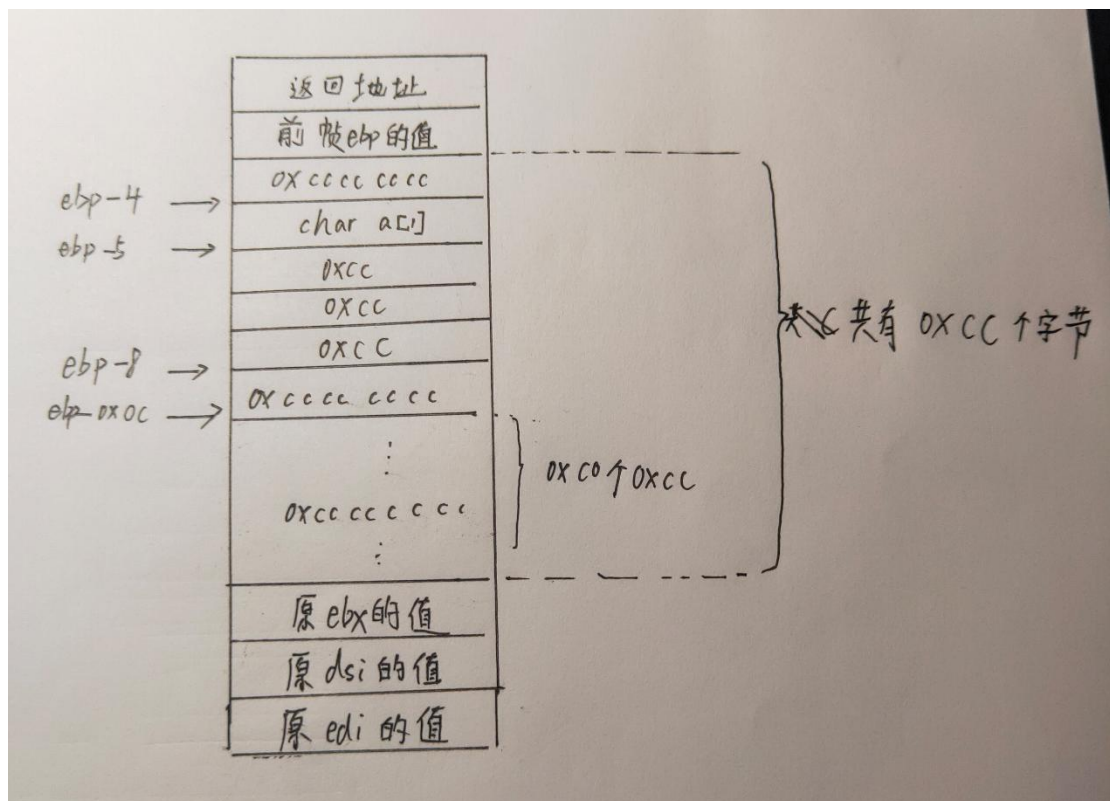


图 16 代码清单 16 对应的栈帧布局