第6篇：C/C++程序栈-call指令和ret指令

铁甲万能狗
自由开发者，专攻C++/Python后端开发(简书平台同号)

+ 关注

承接上文，抛出的几个问题？

- 被调用者必须知道从哪里获取**参数**？
- 被调用者必须知道从哪里获取**返回地址**？
- 调用者必须知道从哪里获取**返回值**？

回到上面的问题，其实如果你理解一点基本的汇编代码的语法话，解析程序栈底层的操作问题,一切都迎刃而解，但我们没有打算用汇编代码来作为一个引例。考虑并不是所有人都有汇编代码的基础，所以本篇会用一个具体的例子分解函数调用过程中每个步骤。

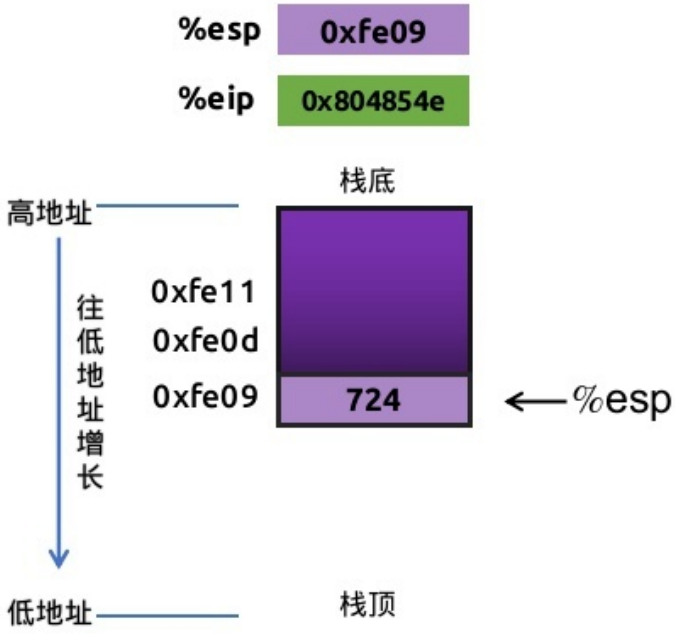
好，回到前文的例子，我们这里需要对前面的过程调用做一个完整的流程说明。

程序栈的执行流

和return指令。以下是一段反编译后的代码段

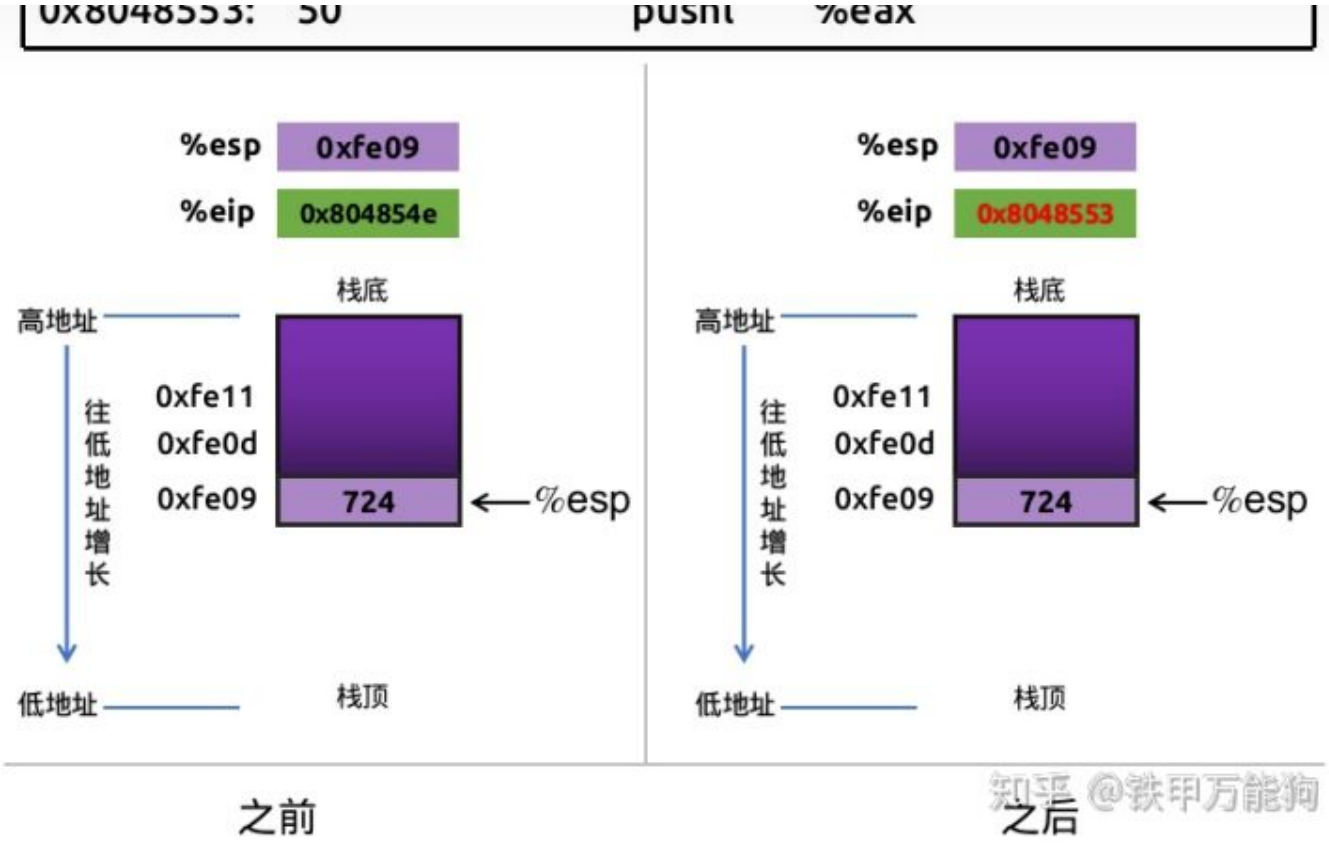
备注:*例子中的代码片段引用紫Washinton University 计算机科学的网上公开课section 5的示例，本文做了一些修改和扩充。*

```
0x804854e:   e8 3d 06 00 00     call    0x8048b90   <main>
0x8048553:   50                 pushl   %eax
```



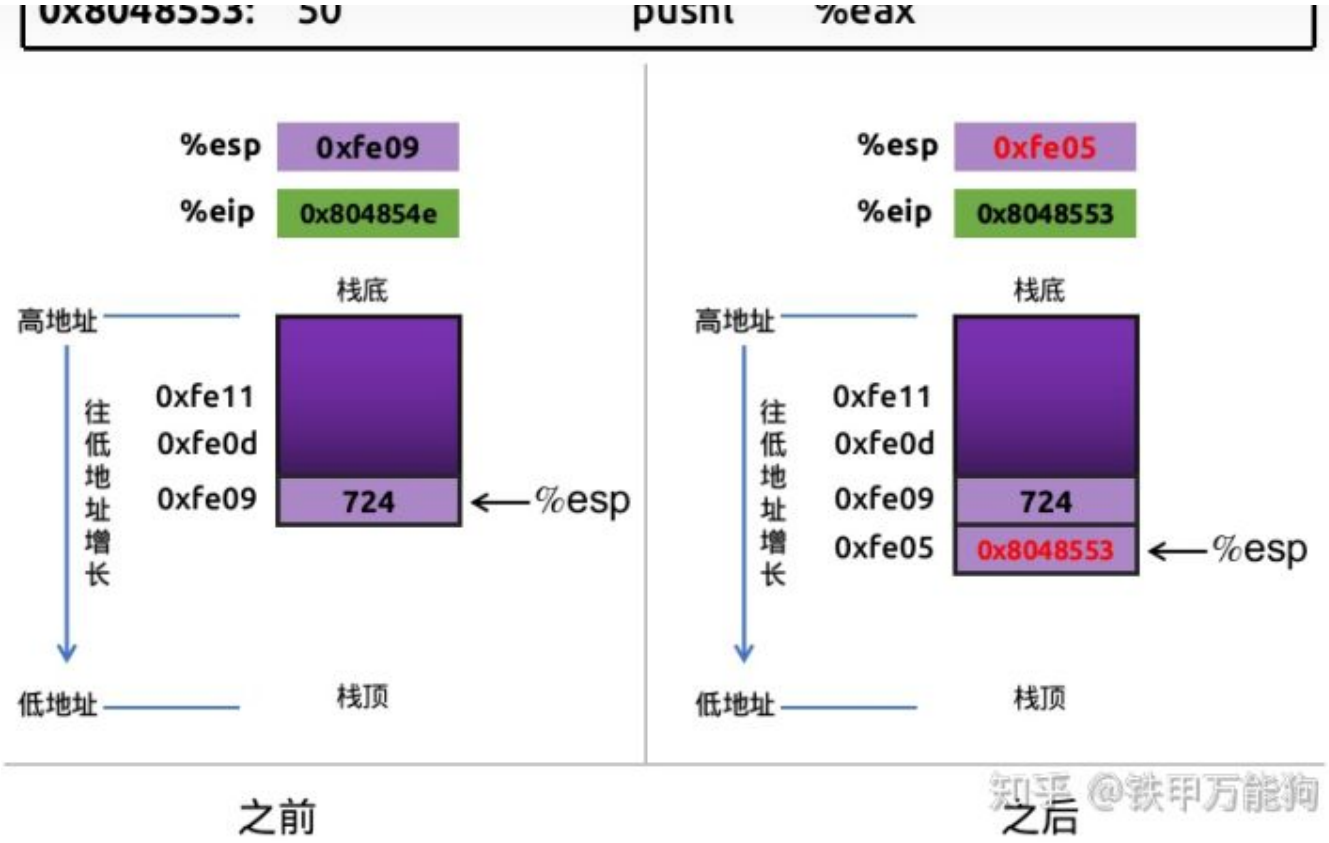
一个参数，该示例描述的是目前将要执行call指令(尚未执行)的程序状态

当执行call 0x8048b90这条语句,被调用者函数位于内存地址0x8048b90的位置，接下来会发生什么呢?
由于此时已经读取了call 0x8048b90这条语句，但我们还没完成对call 0x8048b90的调用，此时%eip指针已经向前指向call指令语句的下一条指令，即会如图变化所示：**eip指针存储的内存地址更新为0x8048553**



接下来发生的事情，是将call 0x8048b90语句的下一条指令的地址，并将该地址值压入栈，此时栈的变化如下图所示：这里伴随着push指令的发生两个变化

- 将**返回地址**(即紧接着call指令的下一条指令所在行的地址)压入栈。
- 栈指针的向低地址递减4个字节，即此时%esp指针只想0xfe05.



接下来发生的事情，会跳转到8048b90这个地址即被调用者函数的所在行的起始地址。你必须要知道为什么是8048b90这个地址，而不是其他内存地址？生成这个地址的方式如下图,这种称为**相对寻址法**

%eip 0x8048553 + 0x000063d = 0x8048b90

代码片段里显示为 5d00,但我们逻辑上的表示是0x5d,编译器会将该**内存地址常数**和eip指针的当前值作加法生成新的内存地址值，在这里即被调用者函数的内存地址。

可能有人会问,为什么不让编译器自行决定任意一个可用的内存地址呢?

对不起！我们不是编译器的设计者,作为C/C++程序员，只要理解到编译器使用的寻址原理，并在知道在生成汇编代码时，编译器已经在底层做了这些工作，我们没必要“打烂沙盘，问到底”。

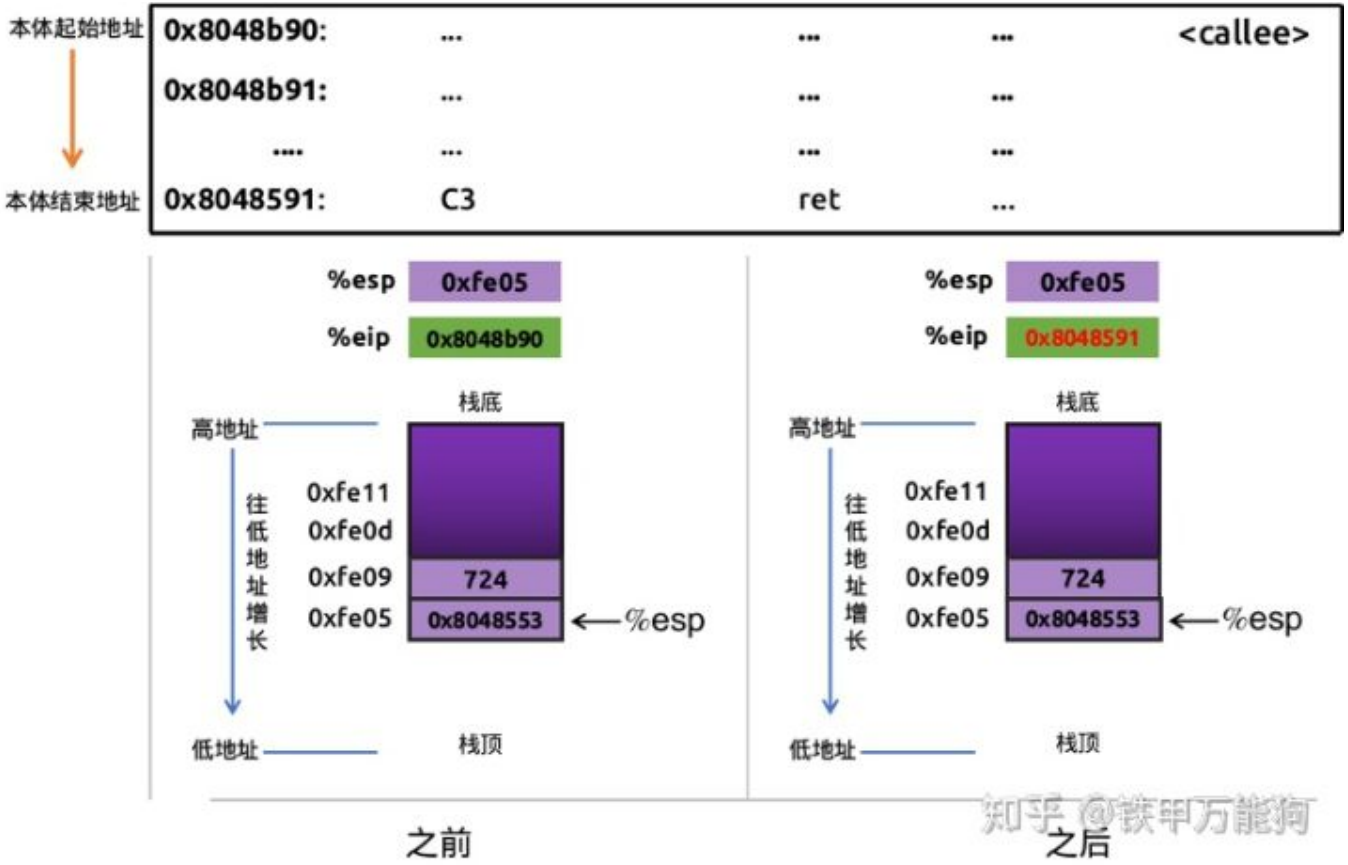
此时，我们真正在意的是,eip指针已经被替换为0x8048b90这个地址,换句话说,call指令告诉编译器可以执行jmp指令跳转到该地址即被调用者函数本体中的第一条指令，

如下图所示(左边的图例）:此时的程序状态包含了如下特征

- **对CPU/寄存器的控制权已经从调用者函数本体转移到被调用者函数的本体。**
- **当前eip指针指向的是被调用者函数的本体中的第一条指令的地址**，即0x8048b90。

下图被调用者函数的中间指令集不是本文讨论的内容，因此其中间的指令集我用“...”忽略了,当被调用函数**将**到达该本体的结尾之时，即ei

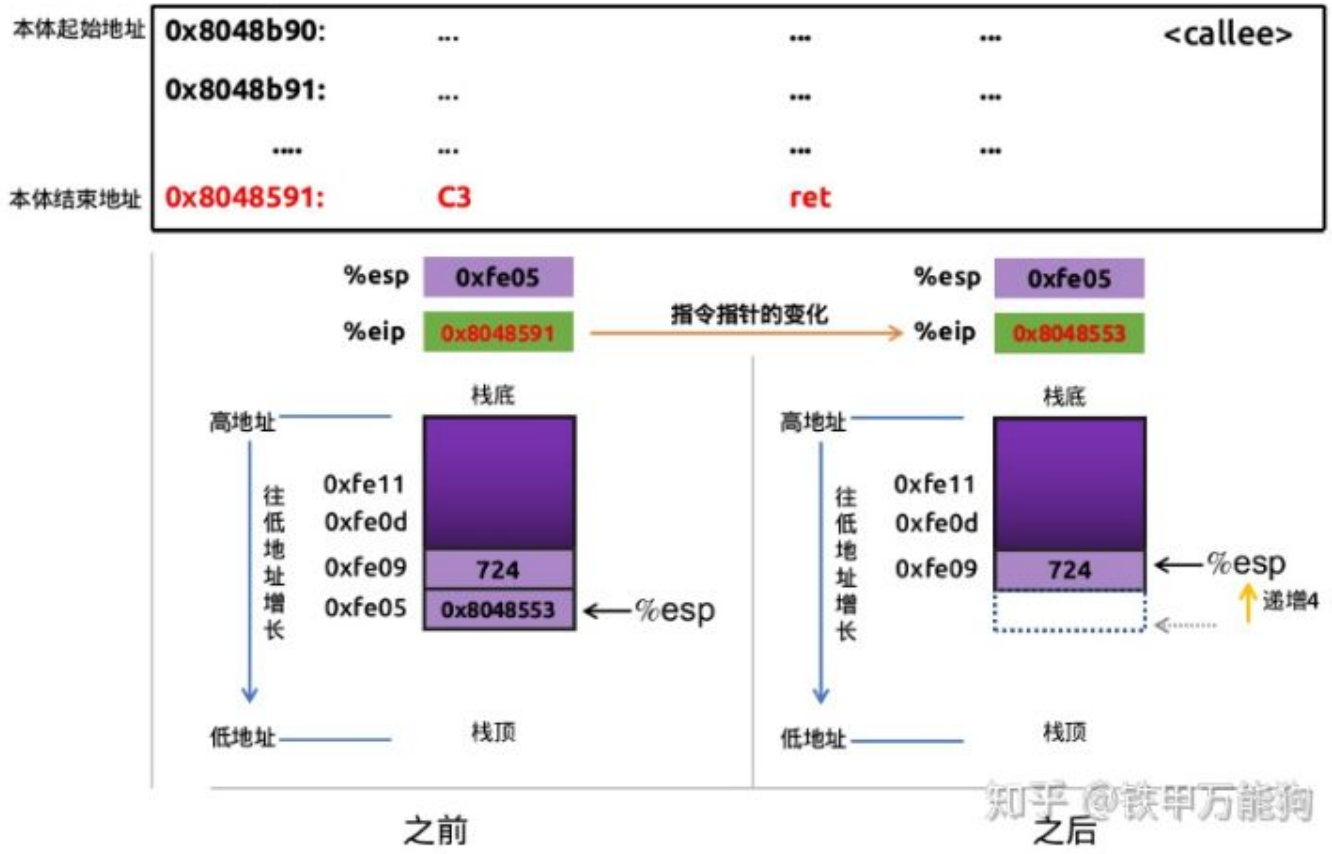
- %esp指针指向栈中的**返回地址**，但此时还没执行出栈操作。



跟接着,就弹出栈顶的**返回地址**(即pop操作),**返回地址**出栈是为了取得该地址,并跳转到该地址指向原来**调用者函数本体**中**紧接call 指令所在行**的**下一条指令。**此时程序的状态变化如下

- %esp指针会向高地址移动4个字节，即esp递增4,即指向0x

如下图变化所示



返回值的处理

从上面的示例中，我们都没有谈及到返回值是如何从被调用者[illegible]数的。是为了简化上面的示例分析，按照惯例，被调用者函数

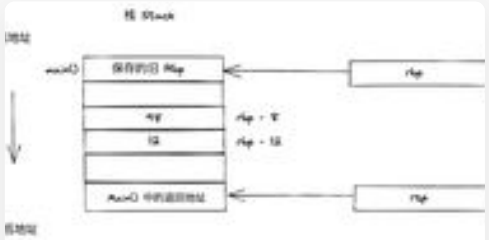
- 调用者函数必须保证在调用可能返回一个值的被调用者函数之前,保存(eax)寄存器中的信息，因为被调用者函数和调用者函数共用同一个寄存器，被调用函数执行后会覆盖(eax)寄存器中的信息，这是寄存器保存操作中的约定。
- 被调用者函数在执行ret指令时,会将(计算过)的适合4个字节的任意类型的返回值保存到(通常是%eax)寄存器,也可能是其他寄存器, x86环境中的eax寄存器只有4个字节。
- 如果要返回大于4个字节的数据类型，最好的方式是返回一个自定义类型的对象的指针，而不是对象本身。
- 返回时，调用者函数在%eax寄存器(也可能是其他寄存器)中找到返回值。

发布于 2020-08-15 02:56

汇编语言 C / C++ 栈（数据结构）

C/C++内存管理
侧重C/C++内存模型的分析

推荐阅读



从汇编的视角看函数调用

陈元

函数调用栈

C语言一些问题几乎是所有的
都会遇到，而且也常因为缺乏
些基本的知识而无从下手。函
调用栈的内容就是其中之一。
是花点时间把以前写的内容整
出来。如果能很好地理解函数

唐风无影

还没有评论

写下你的评论...