# 函数的调用过程

Push param\_2

Push param\_1

// 红色字体部分就是汇编接口 call 会做的事情，除此之外，还会将 eip 指向函数地址

Push return address

Push ebp

Mov esp, ebp

Push local variable

….

// 这里可能疑问，为什么没有pop local variable，因为我们只需要修改ebp和esp就能

//确保当前处于哪个stack frame，包括 param\_1等等，都不需要管

Mov ebp, esp

Pop ebp

// 现在esp的地址就是之前ebp的地址，所以return address就在 新的esp的上面

Ret

// 把param\_1和param\_2回收

Add 4, esp

Add 4, esp

下面是函数调用的时候，栈的内存

A white background with black text

AI-generated content may be incorrect.

具体我建议是看书本（ProgrammingGroundUp-1-0-booksize.pdf）的59页，其实真正需要注意的无非就是call和ret这两个指令，别的结合c++语言来看是挺简单的。

为什么函数的参数要从左往右入栈？

主要解决可变参数函数，当然还有别的原因，我们看看是如何解决的。

假设 printf( a, b )的参数是按照从左往右

Push a

Push b

Call printf

Printf:

push ebp

mov esp, ebp

// 从这一步就会愣住了，因为在printf函数中，他不知道你的第一个参数相比esp是多少// 偏移，但是如果你参数是按照从右往左，那么他就知道第一个参数偏移就是4，其他依// 次递推

mov ?(esp), eax

# IO

.bss 和 .data 两个段唯一的区别就是，是否进行了初始化。

.bss只是预留了空间，但是没给具体的值

.section .bss

.lcomm my\_buffer, 500