链滴

汇编分析

作者：moddemod

原文链接：https://ld246.com/article/1567823037819

来源网站：链滴

main.s

```
g:
    pushq    %rbp
    movq    %rsp, %rbp
    movl    %edi, -4(%rbp)
    movl    -4(%rbp), %eax
    addl    $3, %eax
    popq    %rbp
    ret
f:
    pushq    %rbp
    movq    %rsp, %rbp
    subq    $8, %rsp
    movl    %edi, -4(%rbp)
    movl    -4(%rbp), %eax
    movl    %eax, %edi
    call    g
    leave
    ret
main:
    pushq    %rbp
    movq    %rsp, %rbp
    movl    $8, %edi
    call    f
    addl    $1, %eax
    popq    %rbp
    ret
```

enter指令和leave指令可以理解为宏指令。其中leave指令用来撤销函数堆栈。

等价于下面两条指令

```
movl %rbp, %rsp
popl %rbp
```

enter指令就是再建立起一个空栈。

通常寄存器的名称为分为16位和32位置以及64位置，E开头的就是32位寄存器，R开头为64位寄存，都是向下兼容的。

EIP寄存器是指向代码段中的一条条指令，即main.s中的汇编指令，从main：开始，它会自动加一调用call指令时它会修改EIP寄存器。EBP寄存器和ESP寄存器也特别重要，这两个寄存器总是指向一堆栈，EBP指向栈底，而ESP指向栈顶。

每个函数都有自己的函数堆栈和基地址。另外，EAX寄存器用于暂存一些数值，函数的返回值默认用EAX寄存器存储并返回给上一级调用函数。

首先假定堆栈为空栈的情况下EBP和ESP寄存器都指向栈底，X86体系结构栈地址是向下生长的（地减小）。

首先说几个概念，对于call指令，它在执行的时候的实际动作是先将EIP压栈，然后再把call指令所要用的函数地址给EIP，对于EIP而言，由于每次执行的时候自动加1，也就是指向下一条指令，所以当行call指令的时候，EIP已经指向了call指令的下一条指令了，所以压栈的数据也即是call指令的下一条令地址。

为什么要这样？因为在高级语言中，我们每次调用一个函数的时候，调用的函数执行完，都要回到用之前的函数中，代码也就是接着该函数的下一调指令继续执行。这里也是一样的道理。

□□与call指令对应的就是ret指令了，ret指令就是函数的返回，该指令的实际动作也就是将我们call指执行时候压入的地址