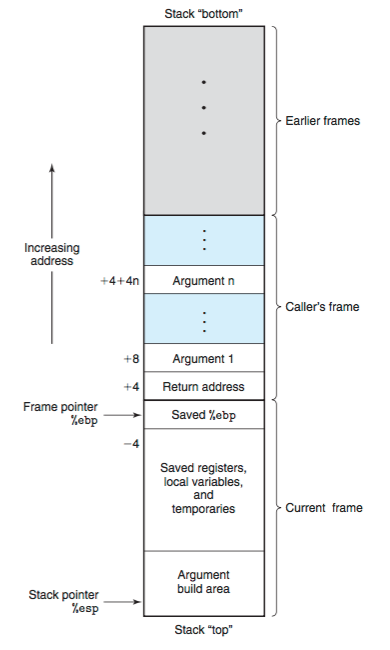
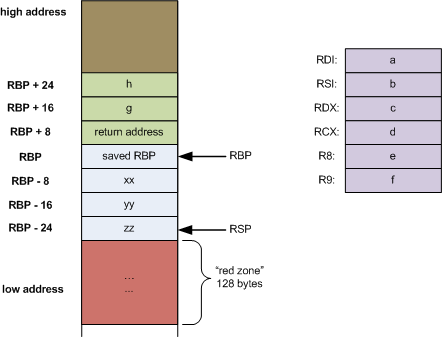
Red Zone 是x86-64执行AMD64标准比起x86新增的栈帧布局。它的大小和位置都是相对栈顶%rsp是固定的，是%rsp下面的128字节。这个区域是保留的，不被信号或中断处理句柄改写。（具体可看下图）



x86-64 x86

作用：

函数在Red Zone中存储无需跨越函数调用的临时数据。特别的，叶子函数可以将这个区域用作它们整个栈帧，从而省去调整%rsp、%rbp分配和释放栈空间的操作。

简单地说，Red Zone是一个优化。代码可以用%rsp以下128个字节存储临时数据，而无需显式地移动栈指针压栈或弹栈，也就节省了编码。

限制：

Red Zone将被函数调用破坏，因此它通常在叶子函数中最有用。

简单使用：

long utilfunc(long a, long b, long c)

{

long xx = a + 2;

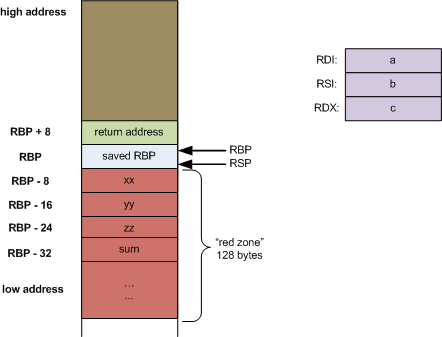
long yy = b + 3;

long zz = c + 4;

long sum = xx + yy + zz;

return xx \* yy \* zz + sum;

}

utilfunc:

保存上一个栈底

pushq %rbp

movq %rsp, %rbp

新栈底就是现在栈顶

movq %rdi, -40(%rbp)

不再移动栈指针，直接在下方存储临时变量

movq %rsi, -48(%rbp)

movq %rdx, -56(%rbp)

movq -40(%rbp), %rax

addq $2, %rax

movq %rax, -32(%rbp)

movq -48(%rbp), %rax

addq $3, %rax

movq %rax, -24(%rbp)

movq -56(%rbp), %rax

addq $4, %rax

movq %rax, -16(%rbp)

movq -32(%rbp), %rdx

movq -24(%rbp), %rax

addq %rax, %rdx

movq -16(%rbp), %rax

addq %rdx, %rax

movq %rax, -8(%rbp)

movq -32(%rbp), %rax

imulq -24(%rbp), %rax

imulq -16(%rbp), %rax

movq %rax, %rdx

movq -8(%rbp), %rax

addq %rdx, %rax

弹出上一个顶

popq %rbp

ret

在Windows的ABI中不存在Red Zone

关于Red Zone的内容可以参见：

<https://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64/>

详细信息看AMD64 ABI