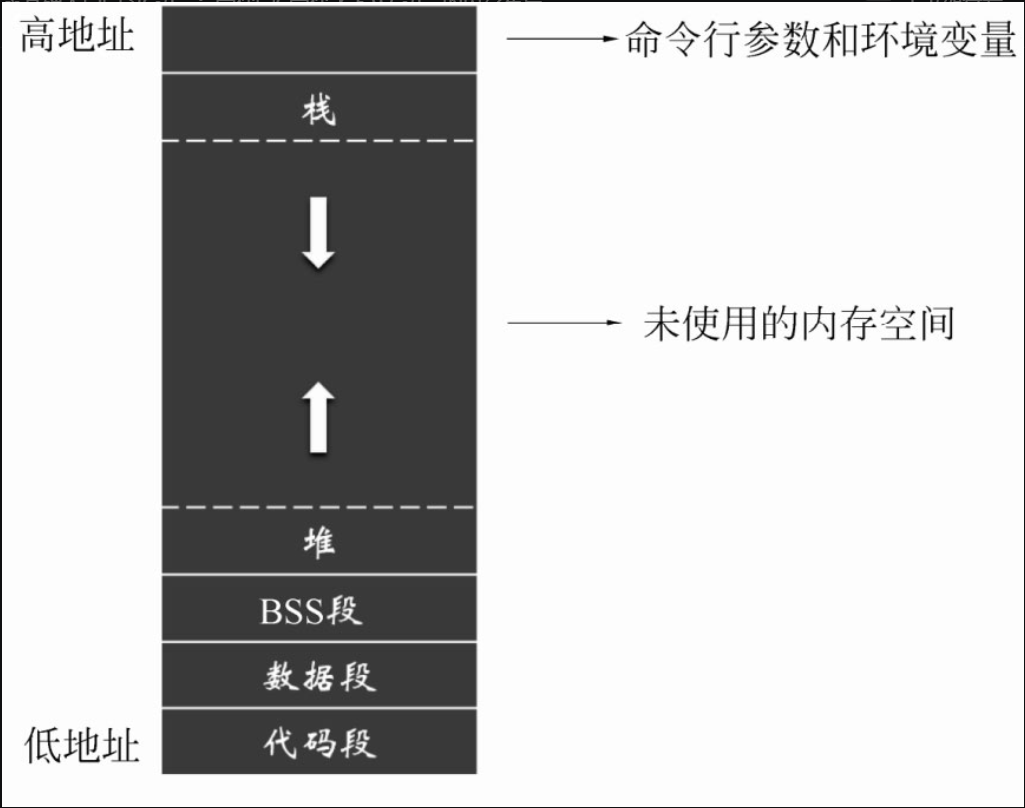
栈

操作系统会为正在运行的程序分配内存空间来存放运行过程中会用到的代码和数据，其中有一块区域就是栈。栈是CPU寄存器的某个指针指针指向的一片内存区域，按照FILO的原则储存数据，**通常用于储存局部变量、传递参数变量、保存函数返回地址等**。

首先，栈是由高地址向低地址延伸的（堆则相反，二者中间是未使用的内存空间），而且是按照FILO的原则单向存放提取数据。



ESP：栈指针寄存器(extended stack pointer)，也叫栈顶指针，其内存放着一个指针，该指针永远指向系统栈最上面一个栈帧的栈顶。  
EBP：基址指针寄存器(extended base pointer)，也叫栈底指针（栈帧指针），其内存放着一个指针，该指针永远指向系统栈最上面一个栈帧的底部。

这两个指针是用来维护函数栈帧的。

最核心的两个操作就是压栈和出栈：  
压栈（push）时，esp指针向低地址移动（32位就减去4,64位就减去8），然后将数据写到该指针指向的内存中。注意：改变的是指针本身的地址。

出栈（pop）时，先从esp指针指向的内存中获取数据，再把esp指针向高地址移动。

所以，esp的上下移动就可以动态地为局部变量创造空间和清除空间，这也是为什么局部变量不能在函数外被调用，因为已经被清除（pop的时候）了

栈帧

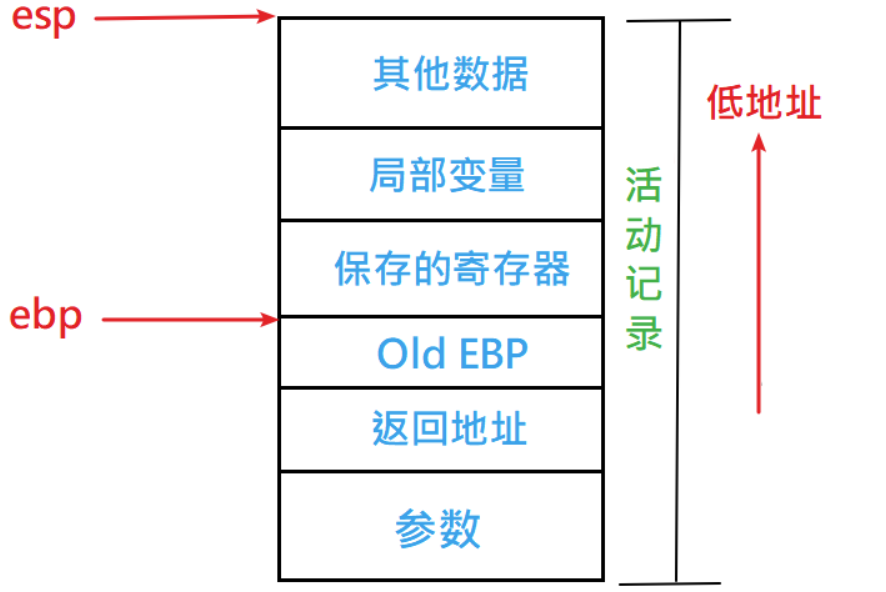
栈帧，也叫过程活动记录，是一种利用esp寄存器访问栈内局部变量、参数、函数返回地址等的手段，它随着ebp的设置（mov）好而生成。

在程序运行过程中，由于参数的传入，栈向低地址的延伸，esp会一直随着栈顶的变化而变化，所以在调用函数时（push压栈之后）会经历这些过程：

1.创造栈帧：先将ebp寄存器里存放的指针指向的地址变为esp寄存器里存放的指针指向的地址（这个地址就是函数的起始地址），这个过程就是mov ebp，esp。然后就可以进行一系列的操作了。

2.销毁栈帧：执行完函数的主要功能（比如加减法、赋值什么的完成）之后，将函数的起始地址返还给esp，这个过程就是mov esp，ebp，这个时候局部变量就失效了。这个时候的栈顶就刚好是创造的这个函数栈帧的栈基址。最后出栈（pop ebp），将栈帧销毁，空间释放，这个时候ebp指针的地址就是一开始压栈之后ebp的地址。返回的参数会在执行函数时保存到eax寄存器。

3.返回原程序：栈帧销毁后，esp指针便会指向返回地址（函数起始地址+1），然后就可以通过这个地址退出函数，回到call指令的下一行指令，继续运行。



这个过程中其实还设计了程序计数器（PC），大致了解了一下，不多赘述。

这样就可以对函数进行维护，同理将寄存器看成函数，也可以通过这种操作来保存和恢复寄存器。

由于栈底的地址不会改变，ebp就不会改变，这样就可以保证esp在函数调用中无论怎么变，在最后清理栈的时候，都可以恢复到起初的状态。

通过函数得到参数也会保存到eax寄存器中，不会随着栈的消失而改变，之后需要调用的时候直接push eax即可。