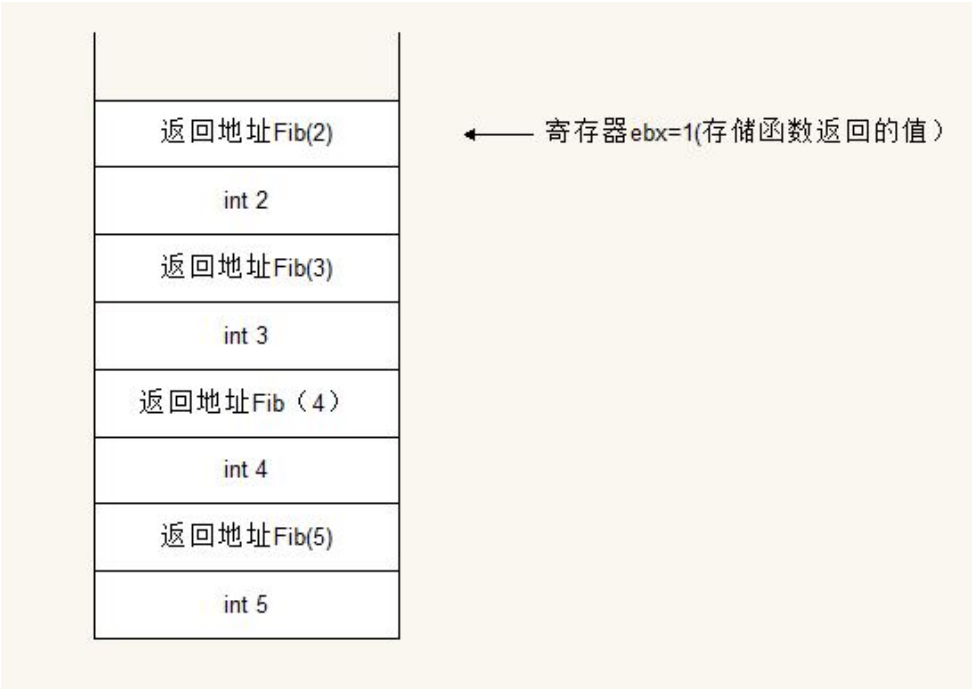
# 函数调用时参数的入栈和出栈顺序

先看看递归的实现和栈的关系，这里引入著名的尾递归–斐波那契数列的实现。既然涉及到底层，自然就该用C语言实现。

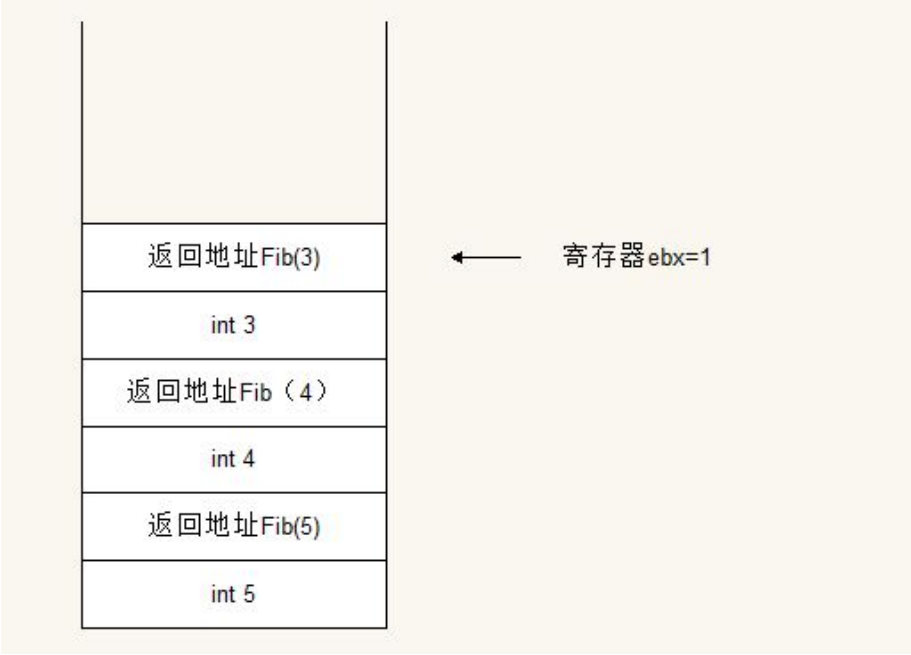
```
int Fib(int n)
{
    if(i==1||i==2)
    return 1;
    return Fib(i–1)+Fib(i–2);
}
```

我们不妨把函数Fib和return语句中调用的函数看作是不同的函数（只是具有了相同的名称），那么就涉及到了函数调用的知识，我们知道，在函数调用的过程中（比如A函数中调用了B函数），编译器就会把A函数的参数，局部变量及返回地址压入栈中存储，再进行B函数的调用。这里用汇编的思想解释会比较生动，如下图所示，假设传入参数为5。

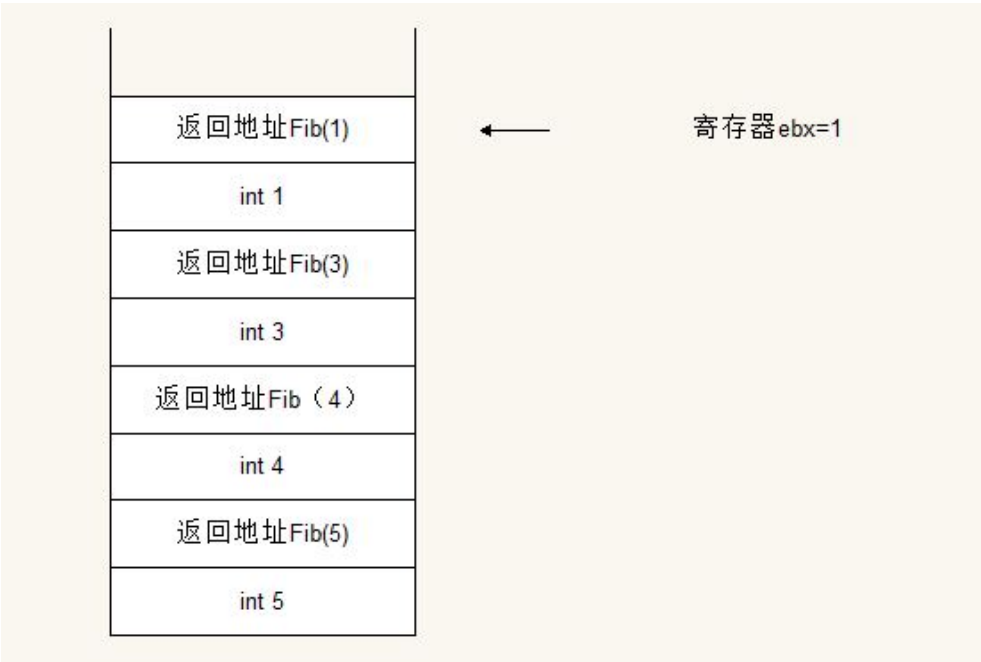
为方便理解，绘图会比较生动，如下图所示，假设传入参数为5




此时返回值已有确定值，开始按顺序出栈，运行到有返回地址字样时执行命令call XXXX（跳入该函数体内执行该函数），如下图



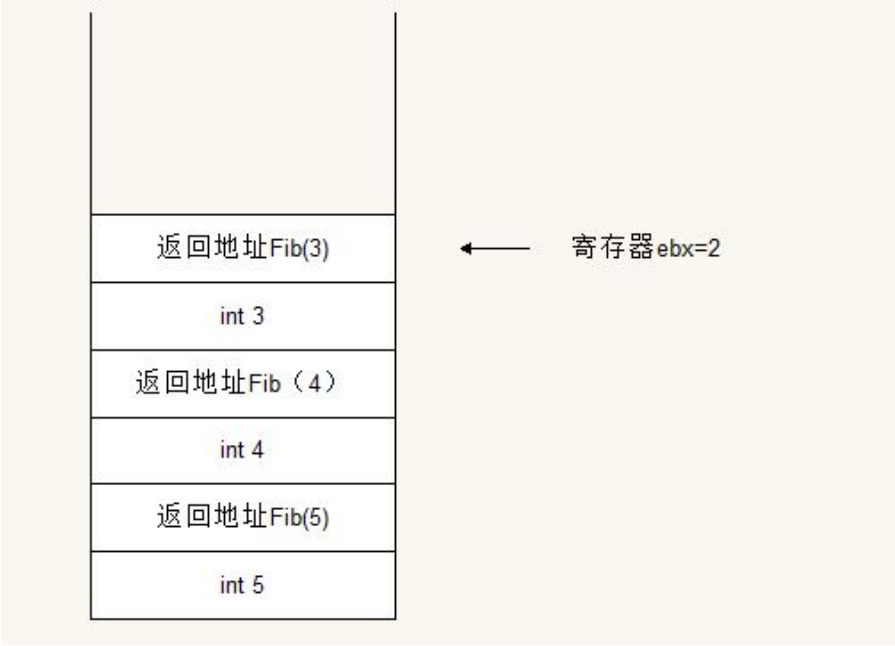

运行到这里跳入Fib(3)函数体内，我们不妨进入函数体内看看，



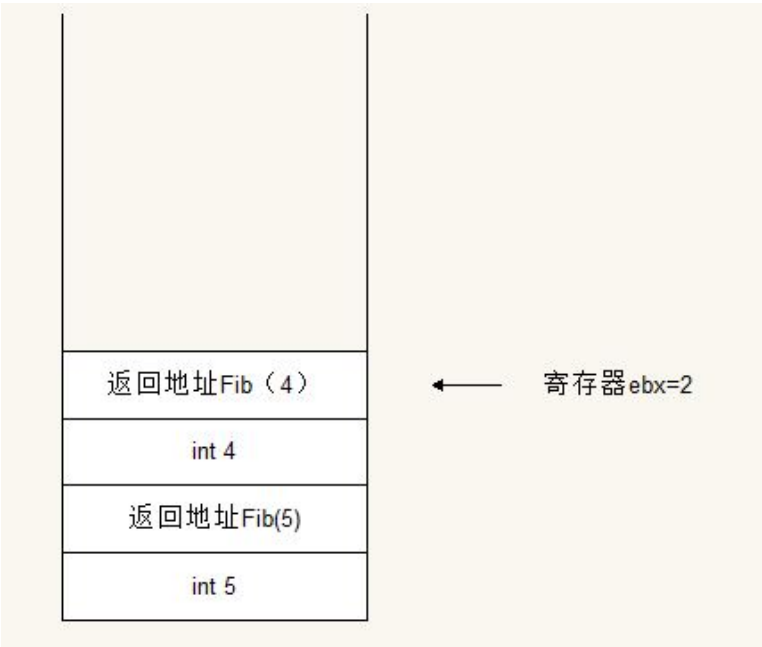

```
int Fib(int n)          <---n=3
{
    if(i==1||i==2)          <---跳过
    return 1;
    return Fib(i-1)+Fib(i-2);    //运行到此处，由于Fib(2)已经由上一步得出，即此时语句等同于return 1+Fib(1);
}
```

操作同第一步

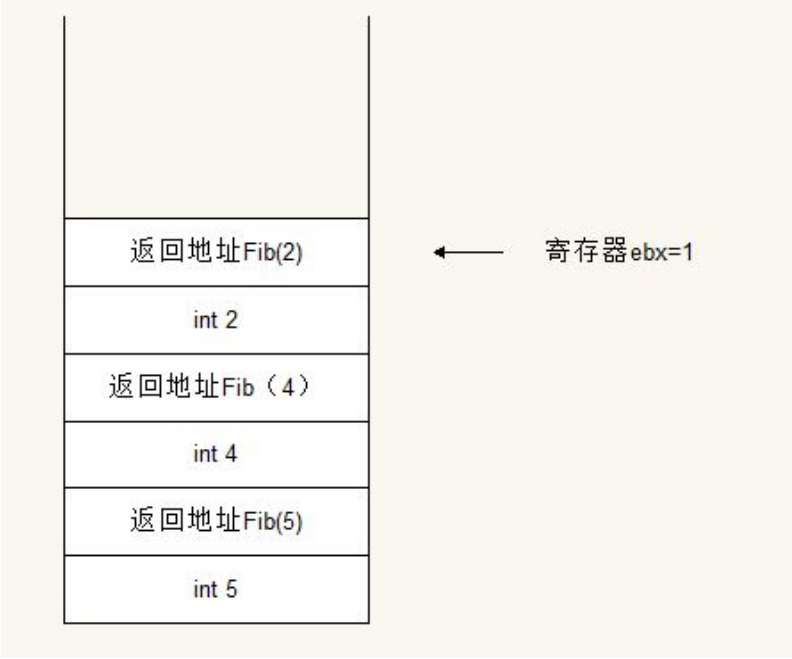
下一步，出栈，由于Fib(3)的值已经明确，继续出栈



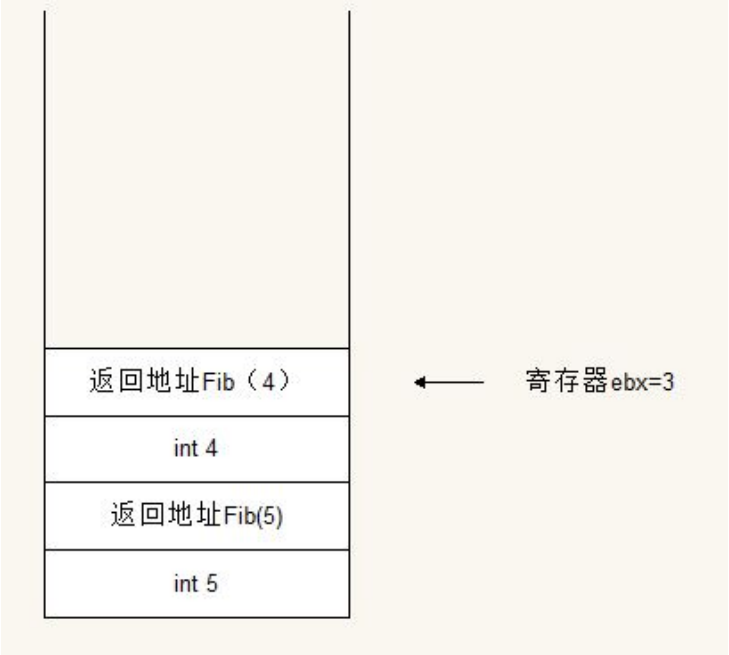

继续出栈



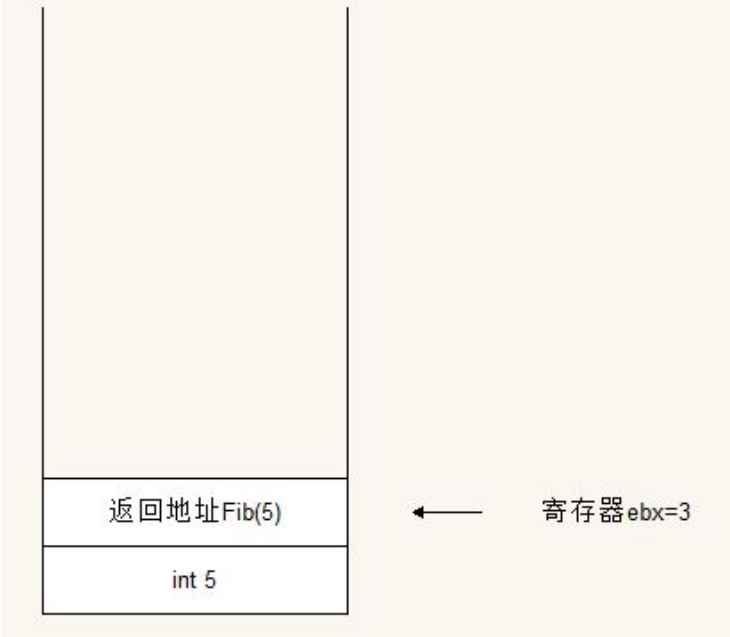

步入Fib(4)函数体内，同理运行到return 2+Fib(2)语句，调用函数Fib(2)，


返回地址Fib(2)
寄存器ebx=1
int 2
返回地址Fib（4）
int 4
返回地址Fib(5)
int 5

出栈.....

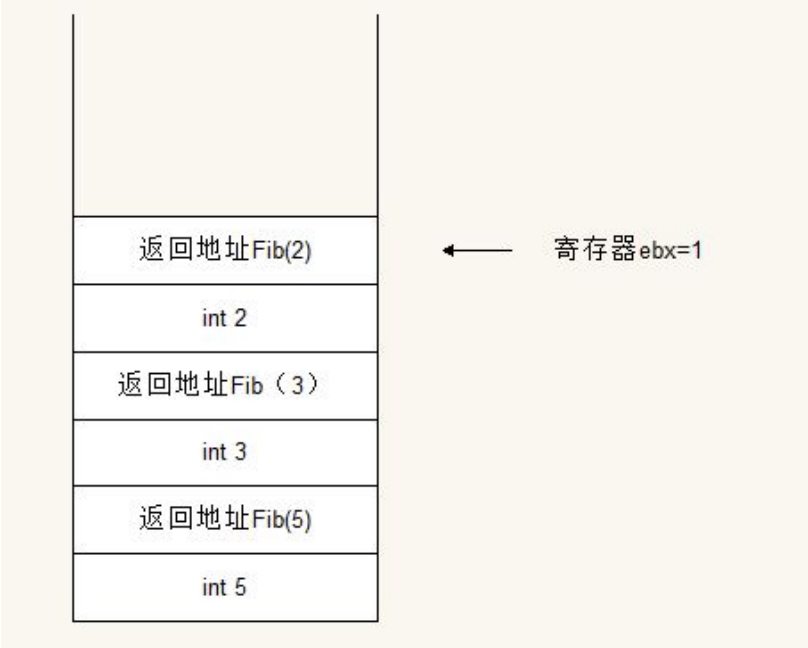

返回地址Fib（4）
寄存器ebx=3
int 4
返回地址Fib(5)
int 5

值已明确，继续出栈

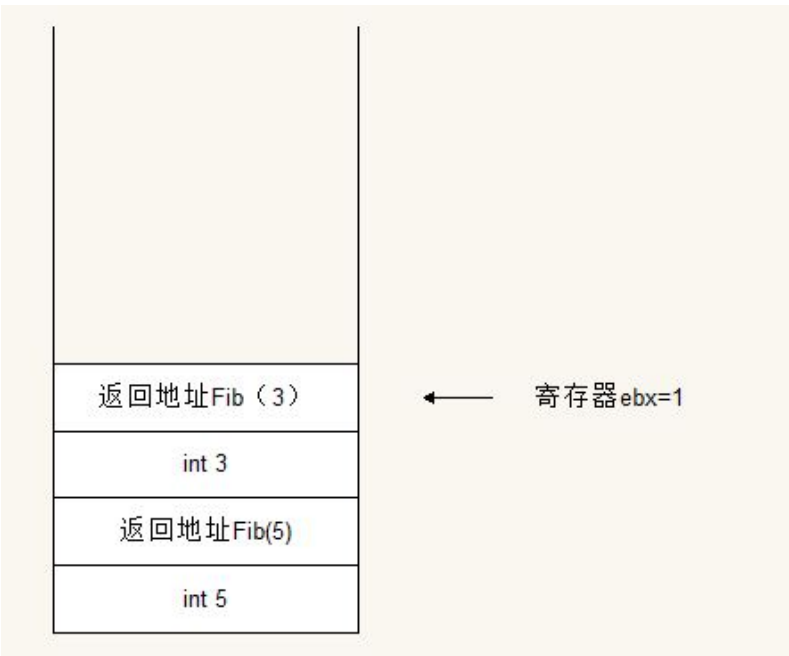



步入函数Fib(5)，运行至return 3+Fib(3)语句处，调用函数Fib(3)

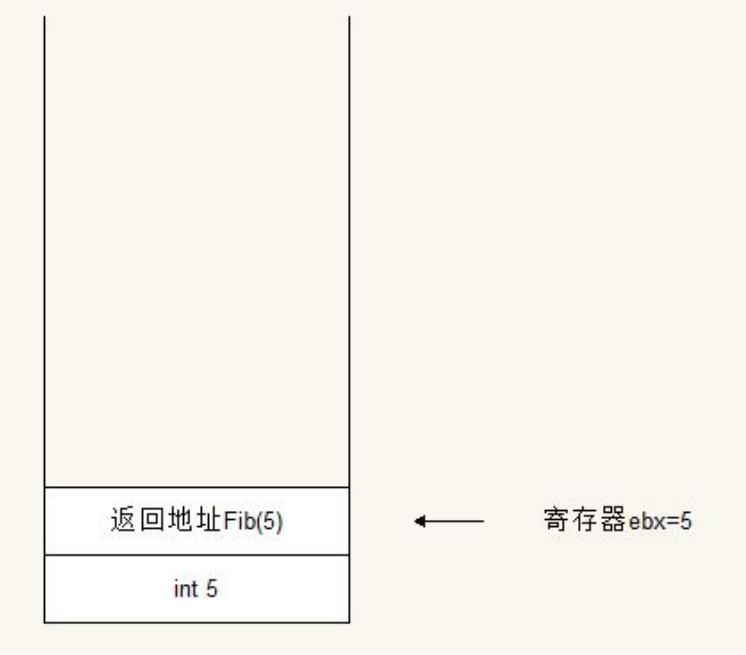

同理步入Fib(3)函数，运行至return 1+Fib(1)语句处，调用函数Fib(1),进而出栈，ebx更新为2，继续出栈



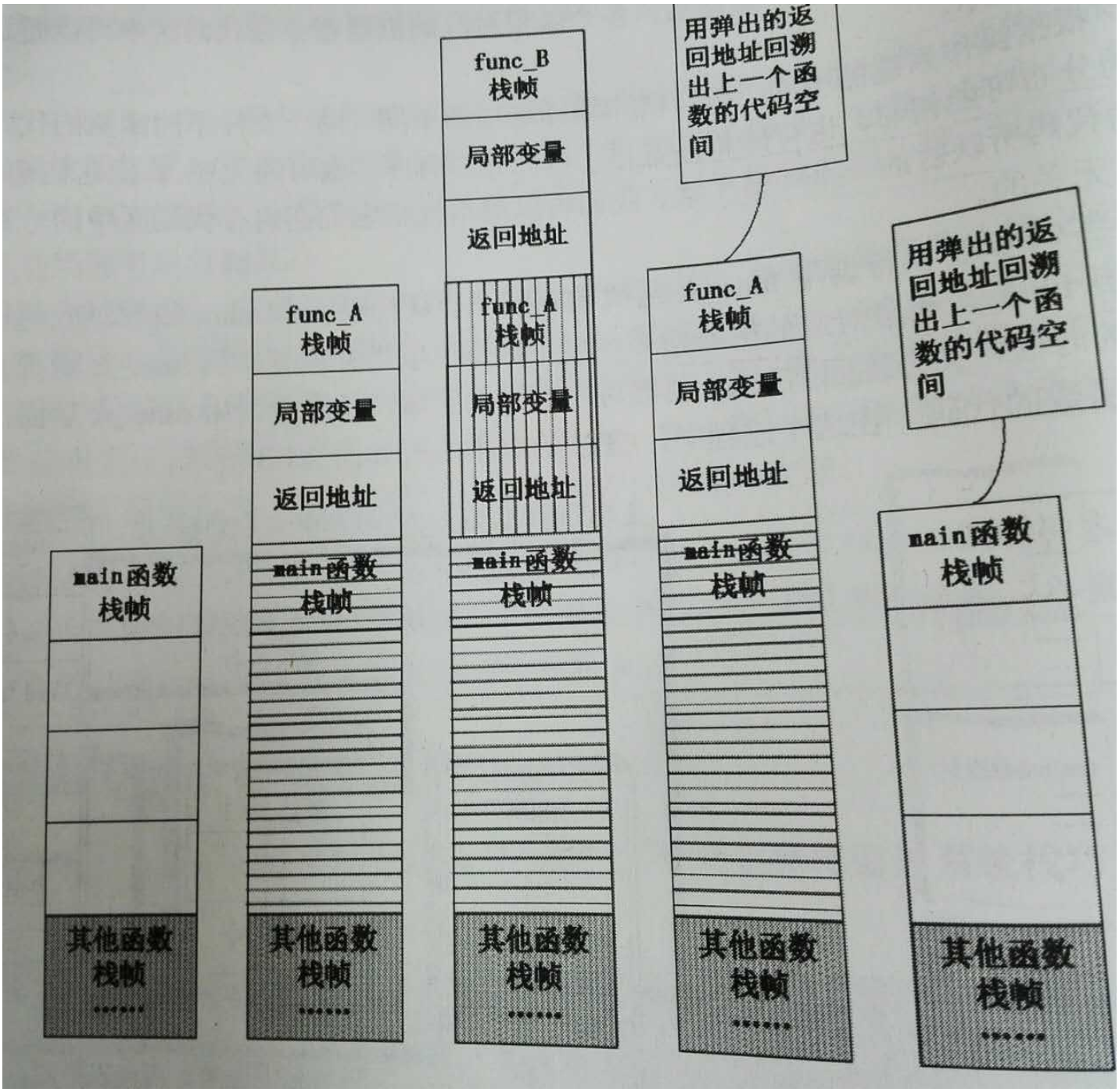

Fib(5)已有确定值，出栈，此时栈空，即Fib（5）等于5.




到这里我们就可以比较直观看出递归及函数调用过程中与栈的关系了，




软件漏洞与技术一书上也描述的很详细，在这里贴出来。


main函数
栈帧
其他函数
栈帧
func_A
栈帧
局部变量
返回地址
main函数
栈帧
其他函数
栈帧
func_B
栈帧
局部变量
返回地址
func_A
栈帧
局部变量
返回地址
main函数
栈帧
其他函数
栈帧
用弹出的返回地址回溯出上一个函数的代码空间
func_A
栈帧
局部变量
返回地址
main函数
栈帧
其他函数
栈帧
用弹出的返回地址回溯出上一个函数的代码空间
main函数
栈帧
其他函数
栈帧

分类: 基本功