程序内存布局进阶

一、什么是寄存器?

寄存器是 CPU 内部用来存放数据的一些小型存储区域，用来暂时存放参与运算的数据和运算结果。一个 CPU 内部会有多个寄存器，其中一部分寄存器有特定的功能，例如我们在课堂上涉及的 EIP、ESP 和 EBP。这三个寄存器的功能如下:

EIP：指令寄存器，存放 CPU 将要执行的下一条指令的地址。

ESP：存放当前栈顶的地址。

EBP：存放当前栈帧的栈底的地址。

二、发生函数调用时栈如何变化?

以下面这个程序为例:

```
1   int add_B(int B1,int B2 )
2   {
3       int B3;
4       B3=B1+B2;
5       return B3;
6   }
7
8   int add_A(int A1,int A2)
9   {
10      int A3;
11      A3=add_B(A1,A2);
12      return A3;
13  }
14
15  int add_C(int C1,int C2)
16  {
17      int C3;
18      C3=C1+C2;
19      return C3;
20  }
21
22  void main()
23  {
24      int X,Y;
25      X=add_A(2,1);
26      Y=add_C(4,3);
27      return ;
28  }
```

图 1

1.当执行完第 24 行代码时，该程序的栈布局如图 2 所示。

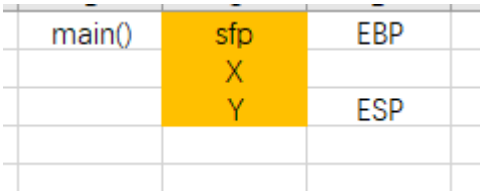

图 2

2.执行 25 行的代码，发生针对 add_A 函数的调用，于是先将传入参数压入栈中，并保存返回地址。执行完第 10 行代码后，程序的栈布局如图 3 所示。此时 sfp0 存储的值为 sfp 的地址。

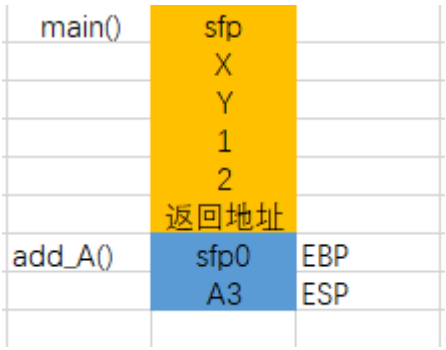

图 3

3.执行第 11 行代码，发生对 add_B 函数的调用，过程同上。执行完第 4 行代码时，程序的栈布局如图 4 所示。此时 sfp1 中的值为 sfp0 的地址。

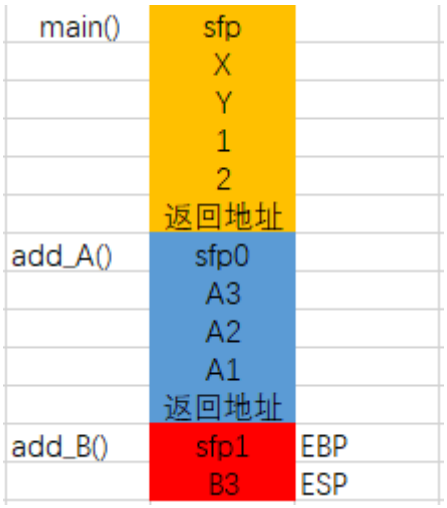

图 4

4.第 5 行代码执行完后，add_B 函数出栈；第 12 行代码执行完后，add_A 函数出栈。第 25 行代码执行完后，程序的栈布局如图 5 所示。

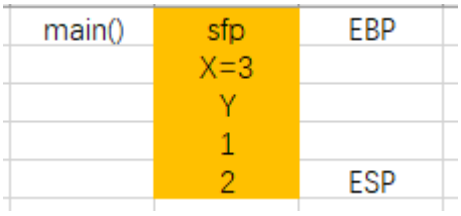

图 5

5.第 26 行代码发生对 add_C 函数的调用，执行完第 18 行代码后，程序的栈布局如图 6 所示。sfp2 的值为 sfp 的地址

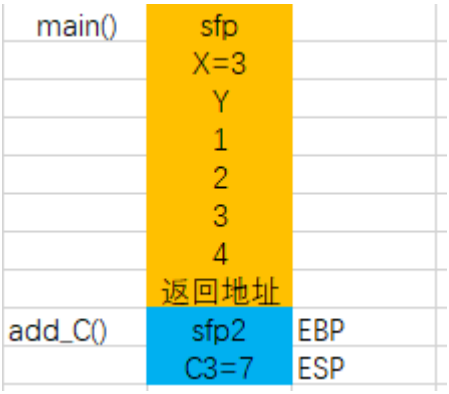

图 6

看完以上的内容，大家应该已经对程序运行过程中发生函数调用时栈的变化有了足够的了解。不过如果你足够细心，也许会有这样的疑问“既然函数调用完成后它的栈帧会出栈，那么第 5、12 和 19 行 return 后面紧跟的值是如何传递出来的呢？”这其实用到了 CPU 内的其它寄存器，比较常用的是 EAX 寄存器。当函数的栈帧要出栈时，会将 return 后面紧跟的值存入 EAX，然后由 EAX 传递给上一级函数。