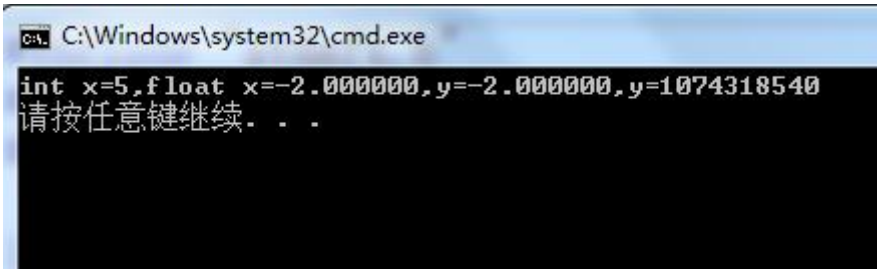
```
//限定x86平台
#include "stdafx.h"
#include <windows.h>
#include <string.h>

int _tmain(int argc, _TCHAR* argv[])
{
    int x=5;
    float y=3.1f;

    printf("int x=%d,float x=%f,y=%f,y=%d\n",x,x,y,y);

    return 0;
}
```

输出：



为什么？

printf 函数按照 cdecl 调用约定传参：

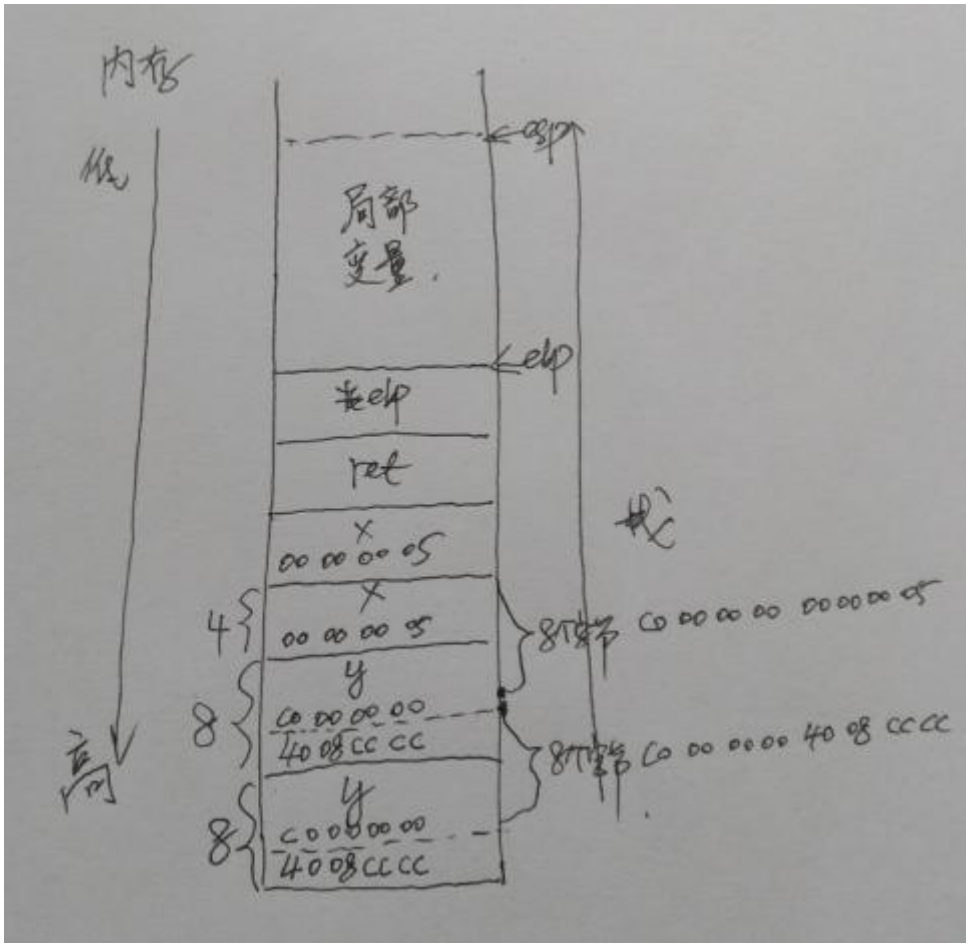
y(3.1f)作为 float 入栈，会提升为 double，因此在栈上，2 次入栈，各占 8 个字节。

然后紧接着是 2 个 x 入栈，分别占 4 个字节。

单精度浮点数 3.1f 的二进制内存为：40466666

3.1f 提升为双精度 8 个字节浮点数后，二进制内存为：4008ccccc0000000

如下图所示：



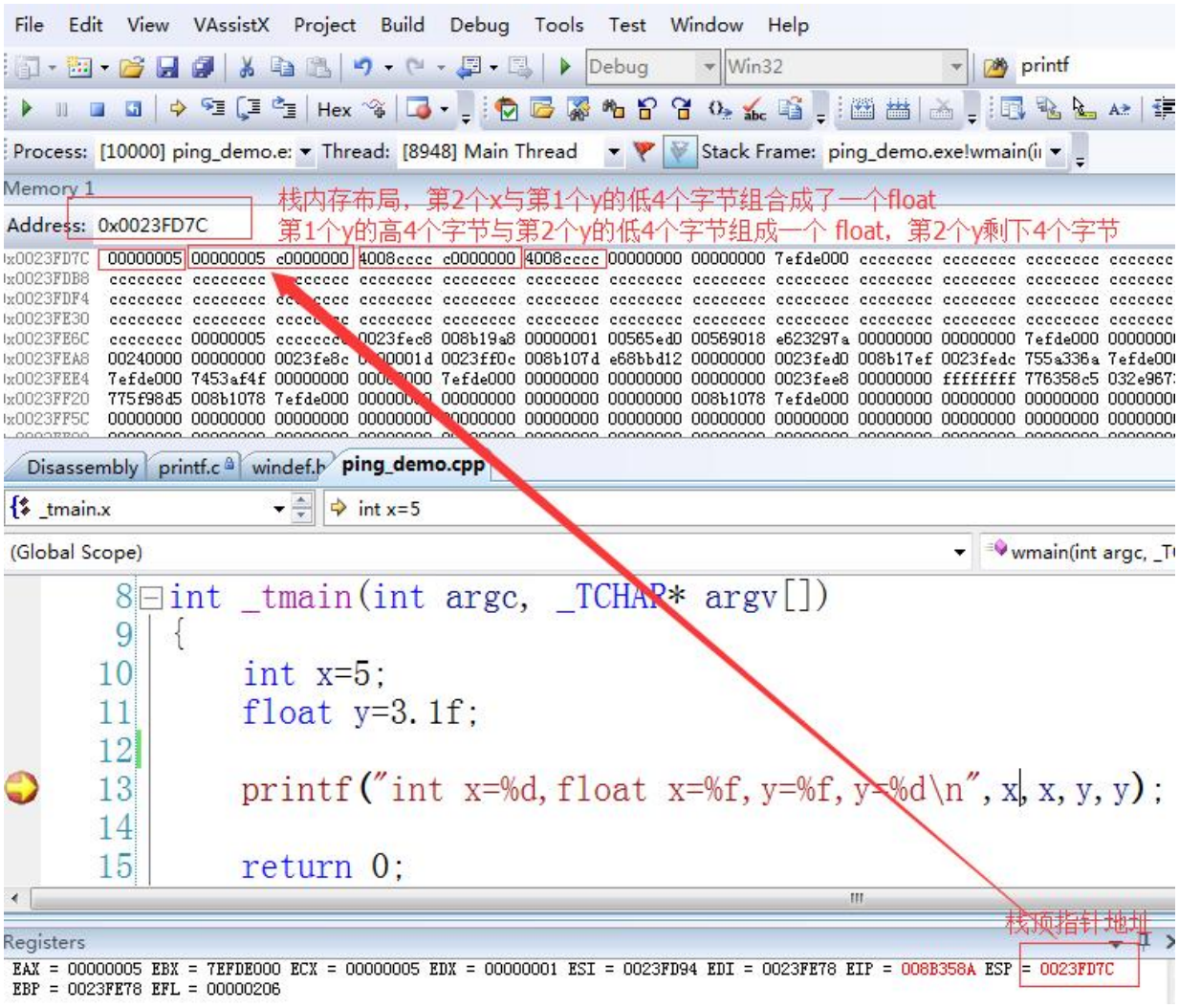
在打印的时候，第一个 x 被当做整数打印，所以从栈上取 4 个字节，打印出来是 5 没有问题。当时打印第二个 x 的时候，被当做 float 类型，会用第二个 x 的 4 个字节，和 y 的低 4 个字节组成一个 8 个字节的 float，打印出来是-2.0
打印 y 的时候，由于 y 的低 4 个字节被 x 占用了，它会继续占用第 2 个 y 的低 4 个字节，组成一个新的 y，打印出来的是-2.0
这就是你看到的现象：x 和 y

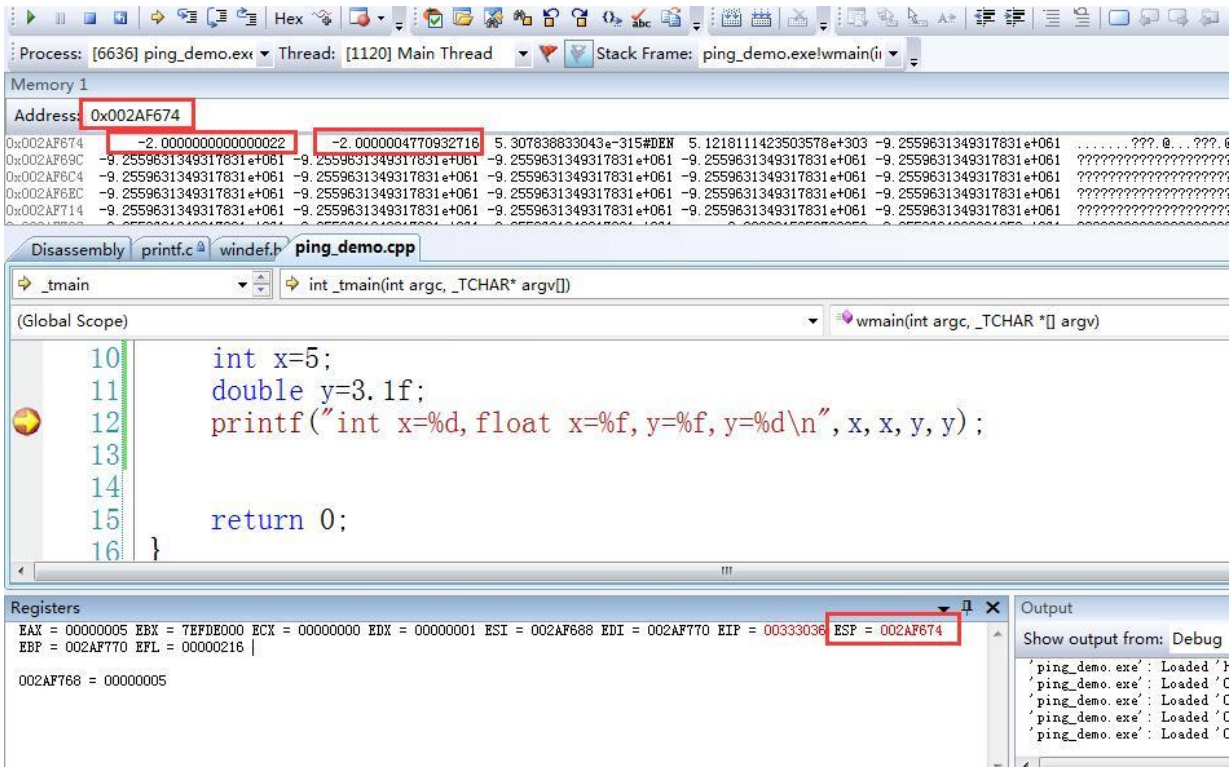
```
c000000000000005 c00000004008cccc
```

这就是第 2 个 x 的 4 个字节和第一个 y 的低 4 字节组成的数被当做 x 的 float 来打印（实际上是 x 被当做 float 了之后，占用了它旁边的 y 的低 4 字节）：

`c000000000000005`, 这个数，约等于-2.0。

第一个 y 的高 4 字节和第二个 y 的低 4 字节（第一个 y 的低 4 字节被第 2 个 x 占用了，因此它会继续占用第二个 y 的低字节）组成的数：`c00000004008cccc`, 这个数，也约等于-2.0。如下图所示：红框中的就是新的 x 和 y 的 float 值：


栈内存布局，第2个x与第1个y的低4个字节组合成了一个float
第1个y的高4个字节与第2个y的低4个字节组成一个 float，第2个y剩下4个字节
栈顶指针地址
int _tmain(int argc, _TCHAR* argv[])
{
int x=5;
float y=3.1f;
printf("int x=%d,float x=%f,y=%f,y=%d\n",x,x,y,y);
return 0;
}



因此，第 2 个 y 只剩 4 个字节了，被当做了 4 个字节的整数打印