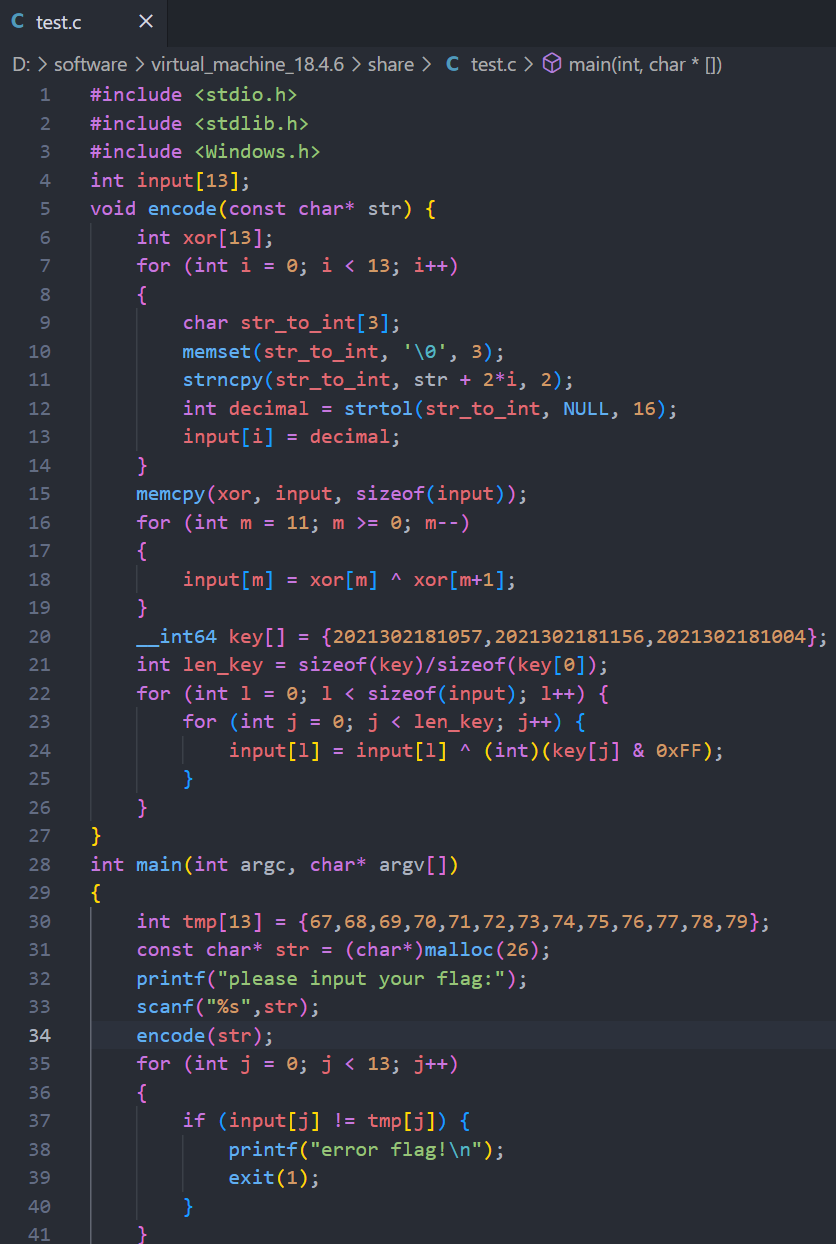
1. 将下面的实验代码进行修改，并使用编译软件进行编译：
   1. 将代码中的key数组修改为学号
   2. 数组长度与组员数量相同
2. 对编译后的程序使用UPX Tool进行加壳
3. 对加壳后的程序使用ESP定律法进行脱壳
4. 对脱壳后的程序使用IDA进行分析，并对IDA反编译后的伪C代码进行注释、变量重命名，截图，需要操作的函数为：main函数、加密函数
5. 编写python脚本，求解出flag的内容

1.编译源程序

首先将key数组的值换成组员学号如下所示

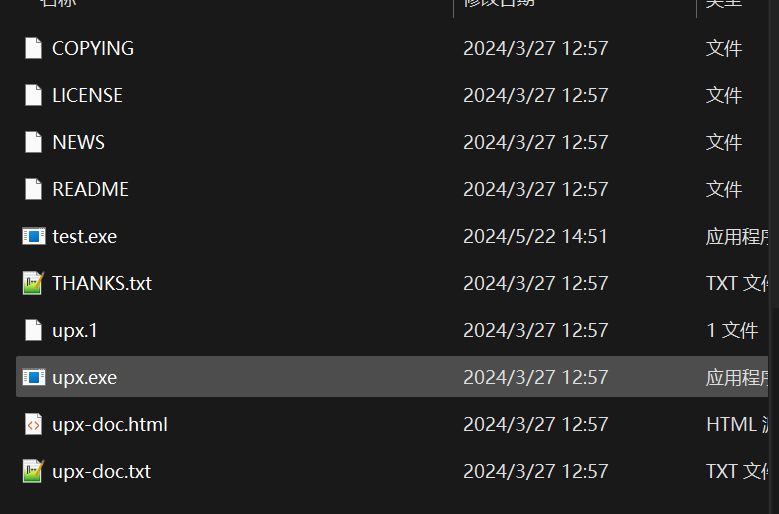


运行程序后获取到程序的可执行程序test.exe

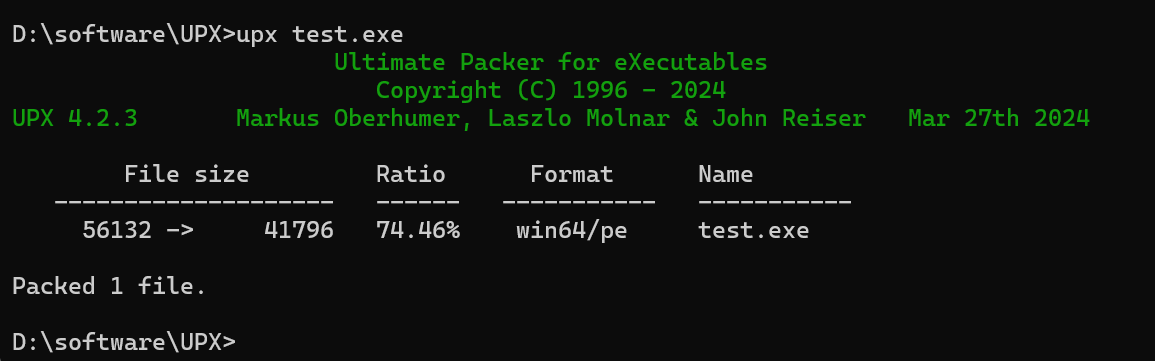


2.对编译后的程序使用UPX Tool进行加壳

将test.exe放到upx根目录下

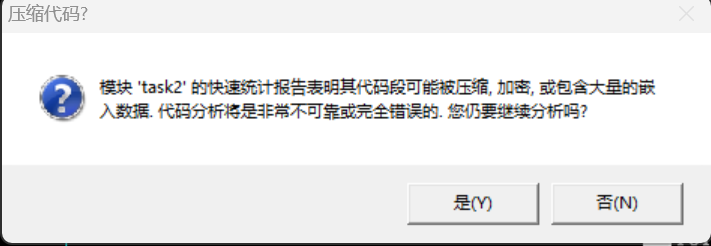


upx test.exe进行加壳

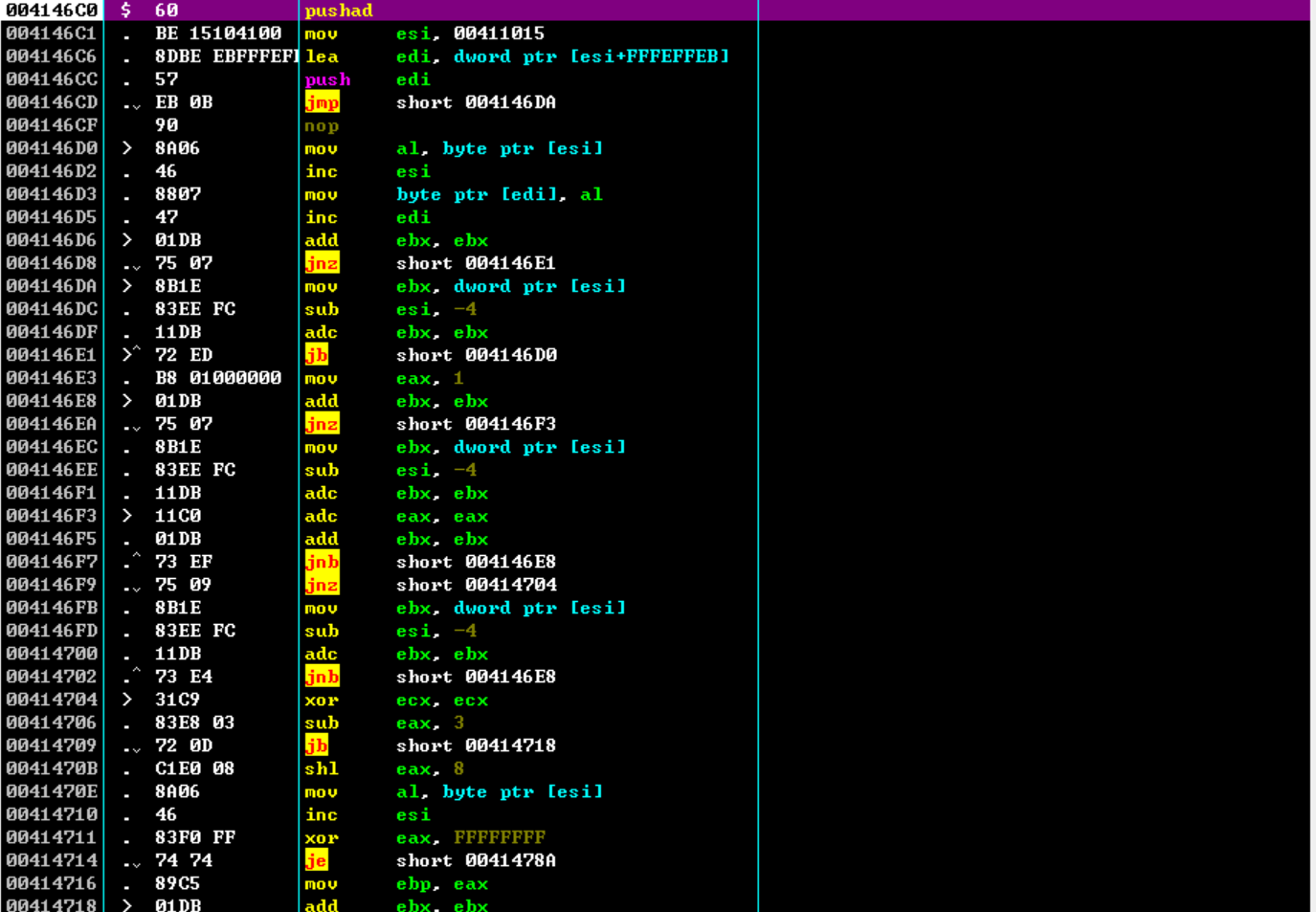


3.对加壳后的程序使用ESP定律法进行脱壳

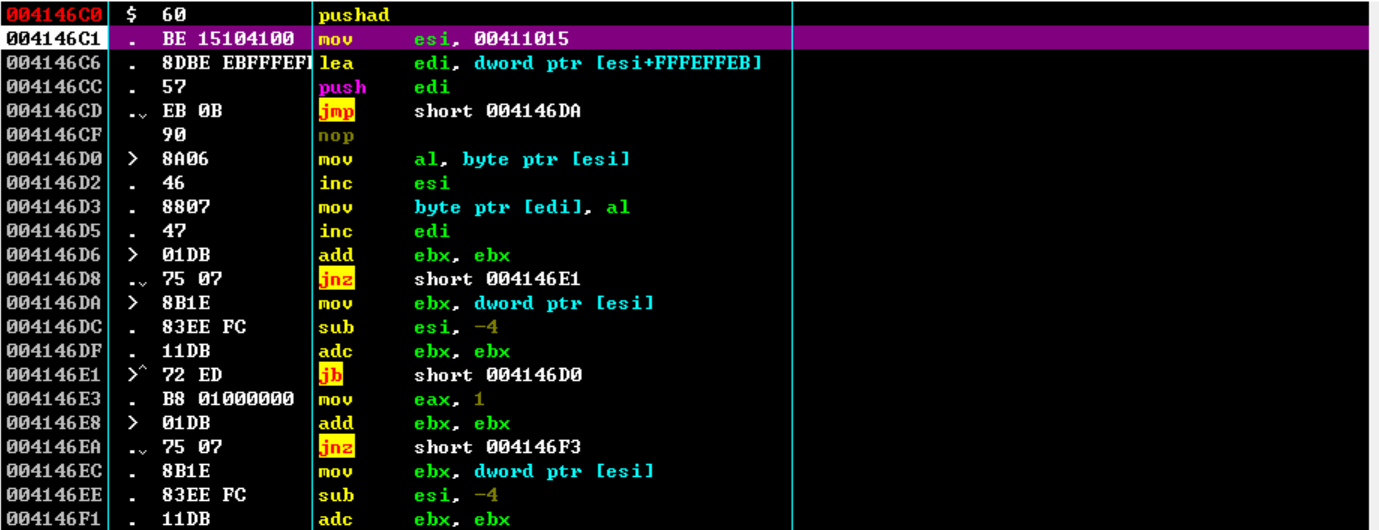
使用ollydgb将加壳后的可执行程序打开会弹出如下框



打开后的界面如下所示，起始指令为pushad可以确定为已经加壳



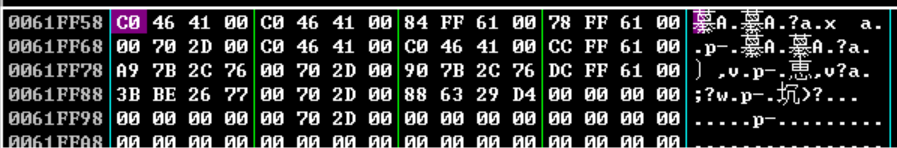
单步步过使程序运行到mov指令处



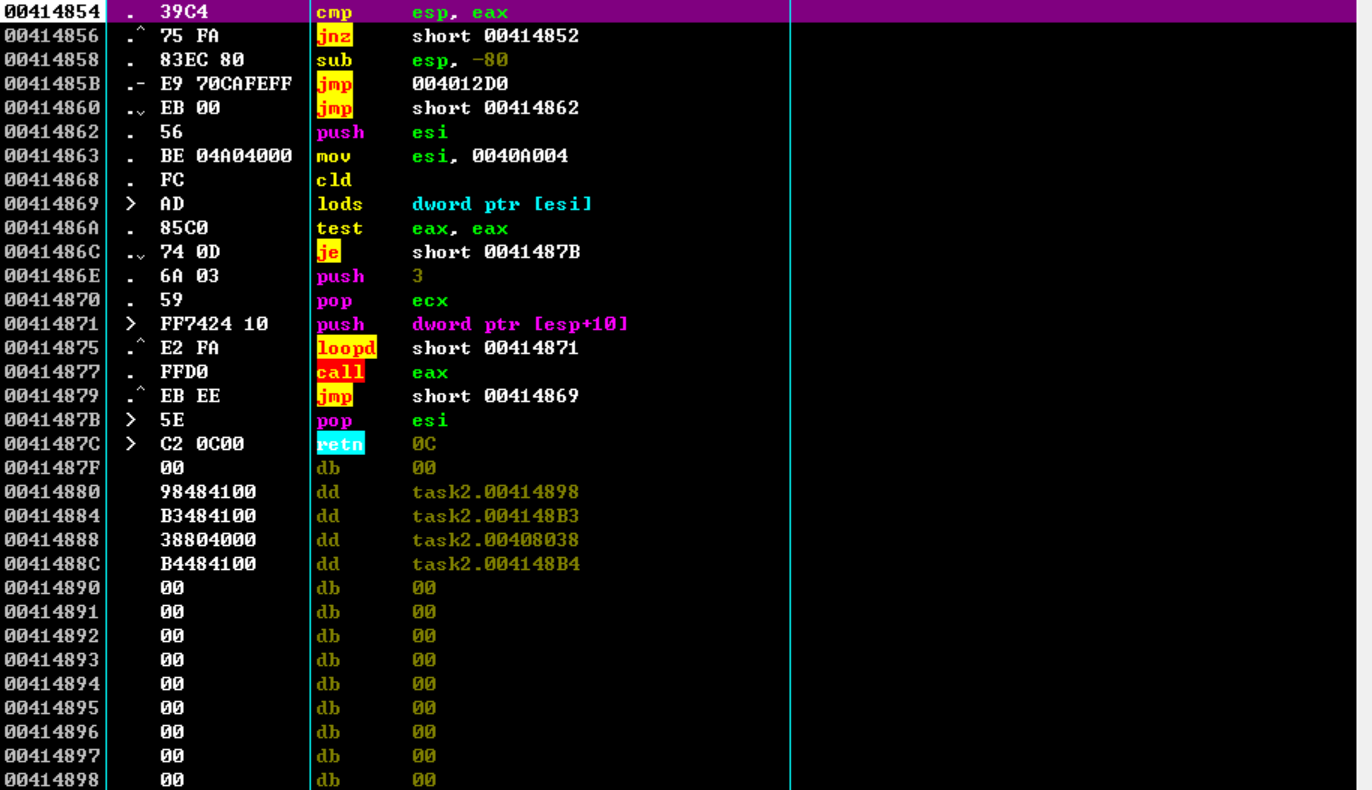
删除原有的断点，在ESP寄存器处右键点击“在数据窗口中跟随”来对esp下硬件断点



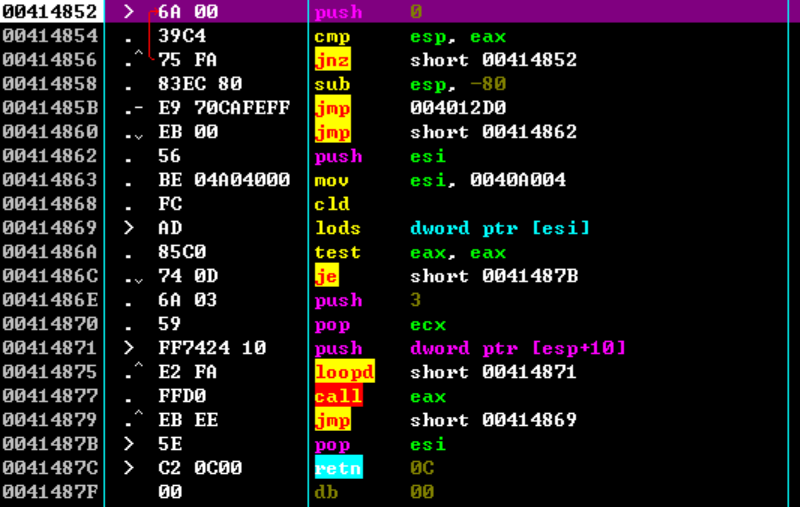
选中起始数据右键-断点-硬件访问-WORD设置断点



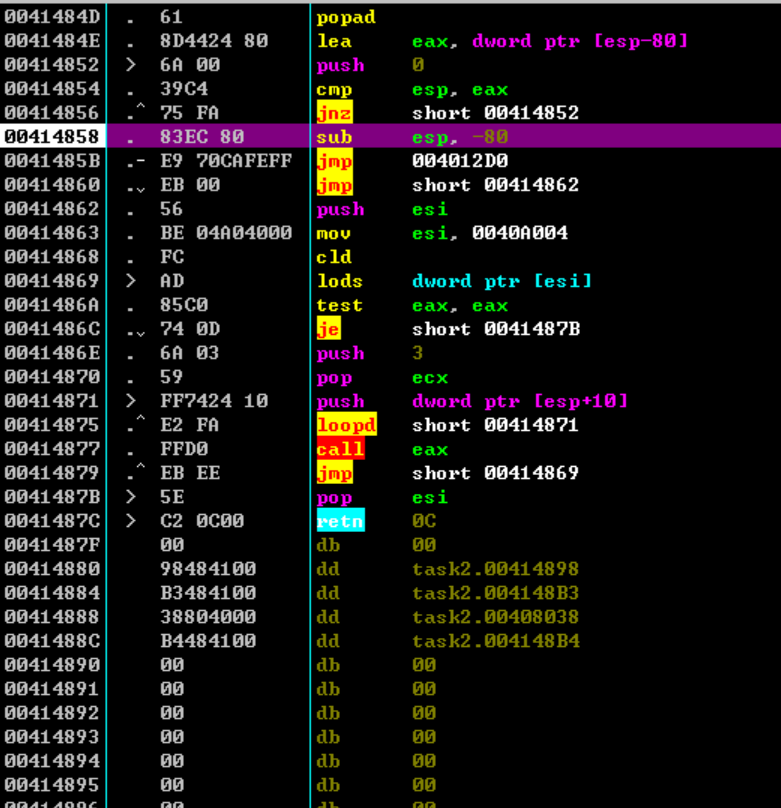
按下F9将程序运行到断点位置



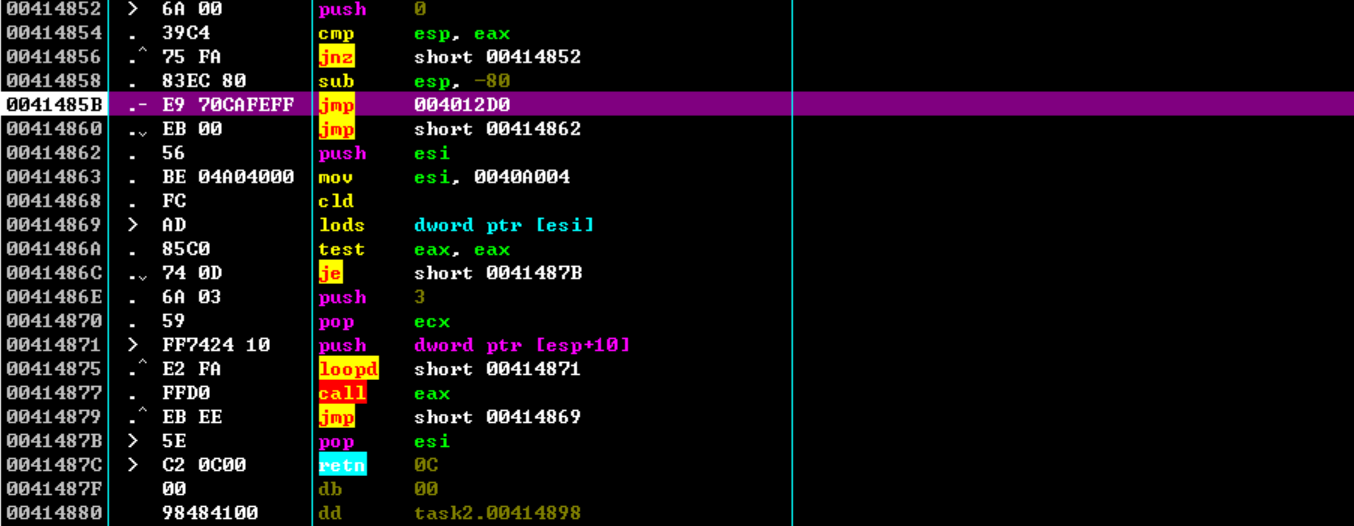
F8运行该程序之后发现程序在循环执行该语句



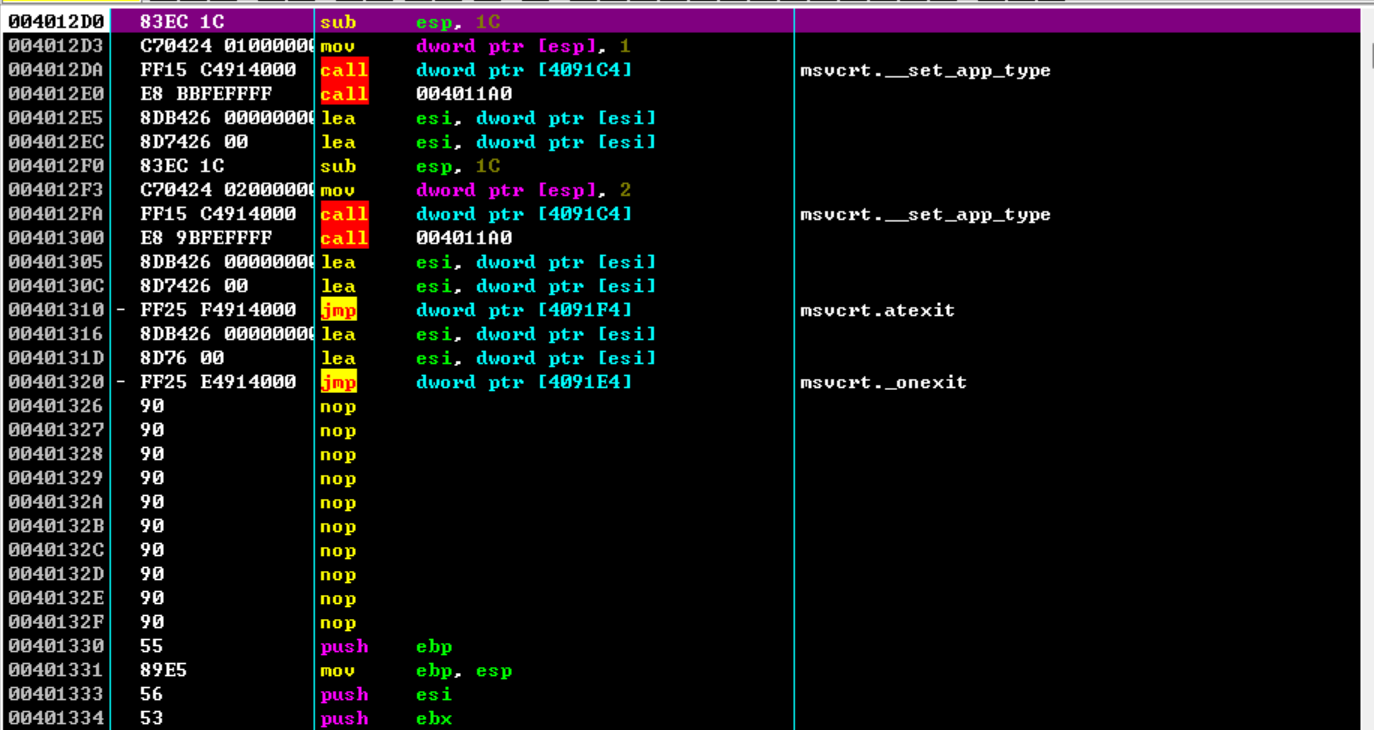
将鼠标指向sub esp,-80语句按下F4之后将程序跳出循环，执行至当前语句



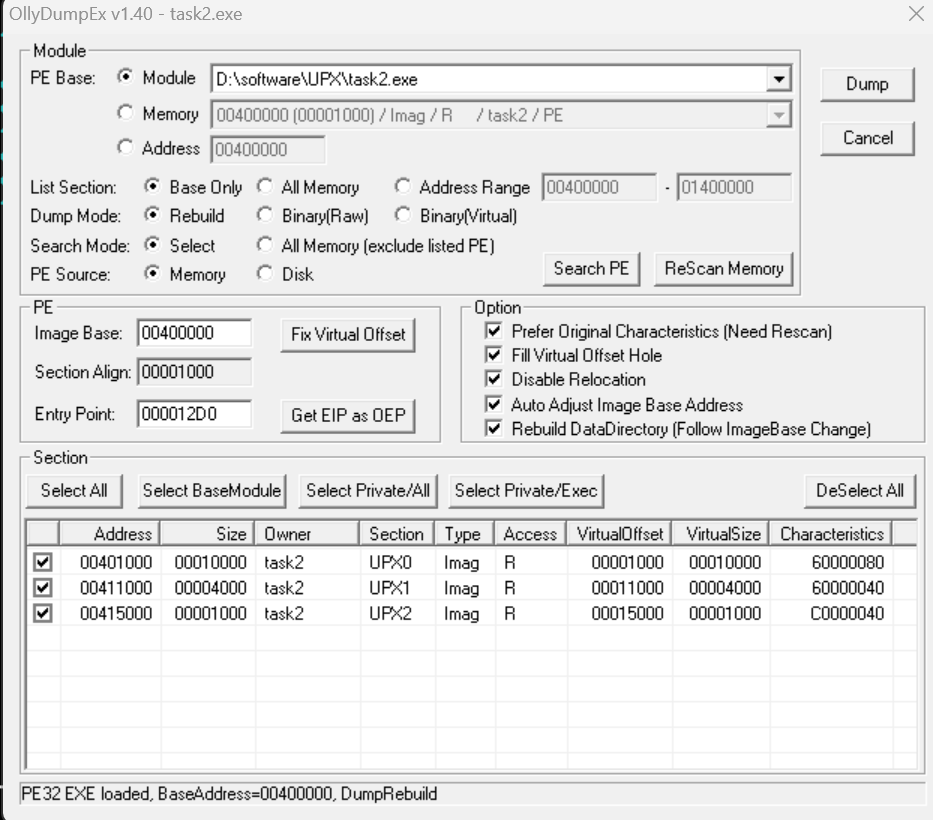
继续执行到第一个jmp语句按F7步入



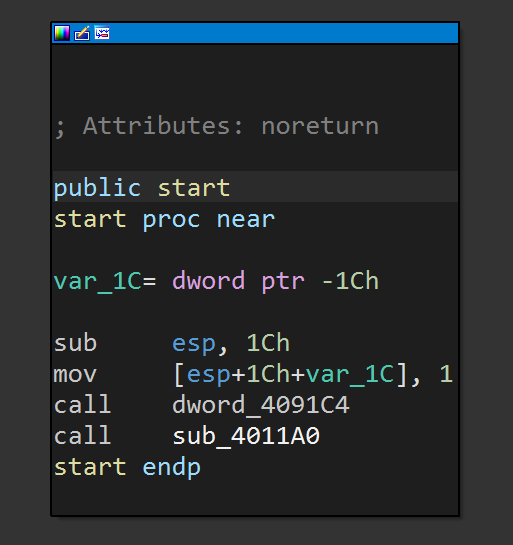
步入后的第一条指令即为我们的OEP，在该程序位于0x004012D0位置



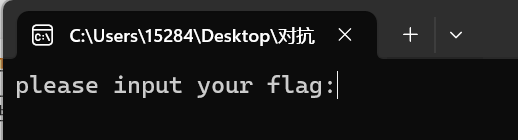
在地址（0x004012d0）处右键运行OllyDumpEx插件，点击Get EIP as OEP修复OEP之后点击Dump按钮进行脱壳



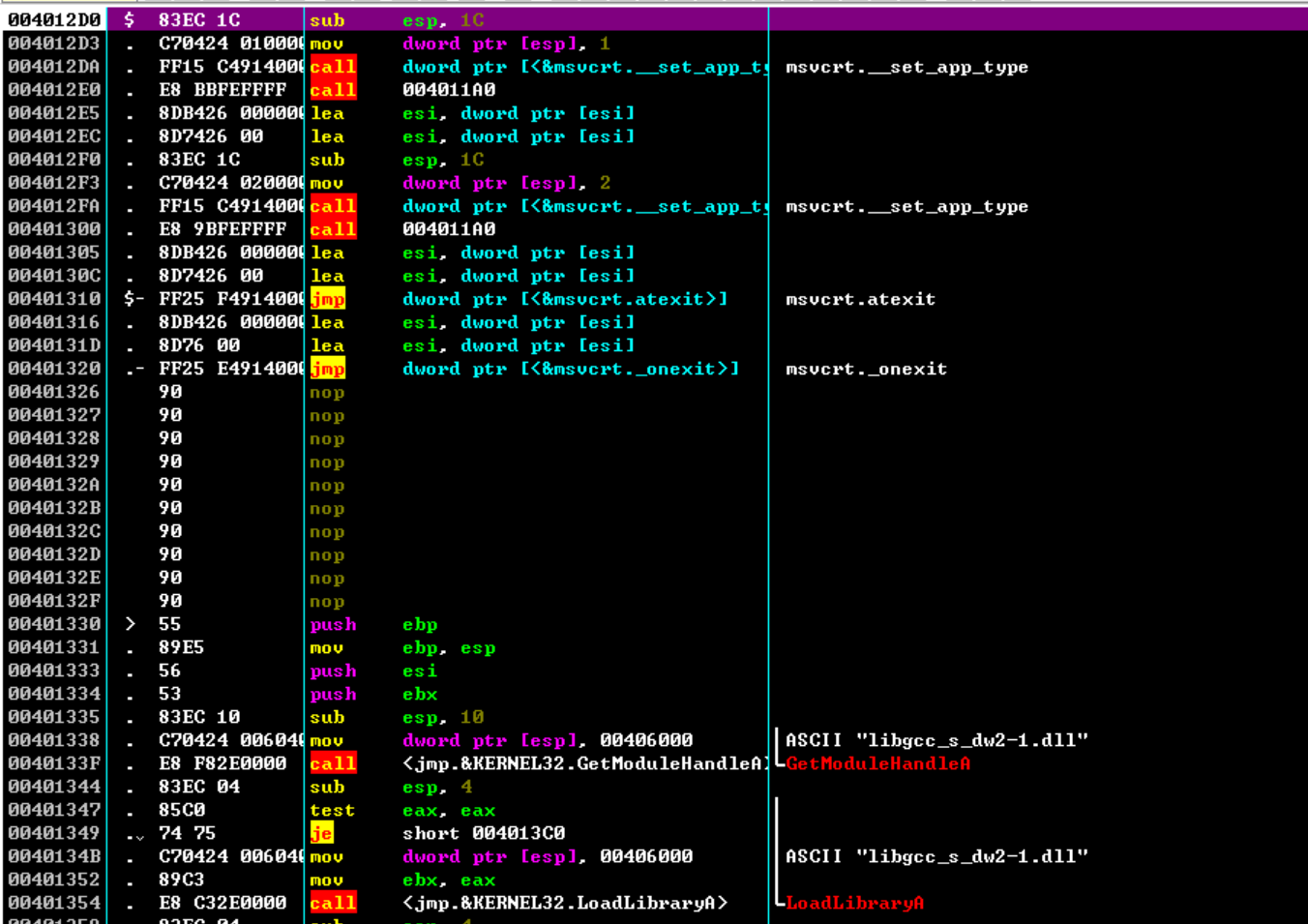
将导出后的文件放到IDApro中打开结果如下，可以看到没有报错



运行脱壳后的程序如下，发现可以成功运行，这里如果打开后不能够成功运行的话可以查看UPX版本，将UPX版本使用更低版本进行加壳之后再重新进行上面的步骤进行脱壳之后就可以成功运行。

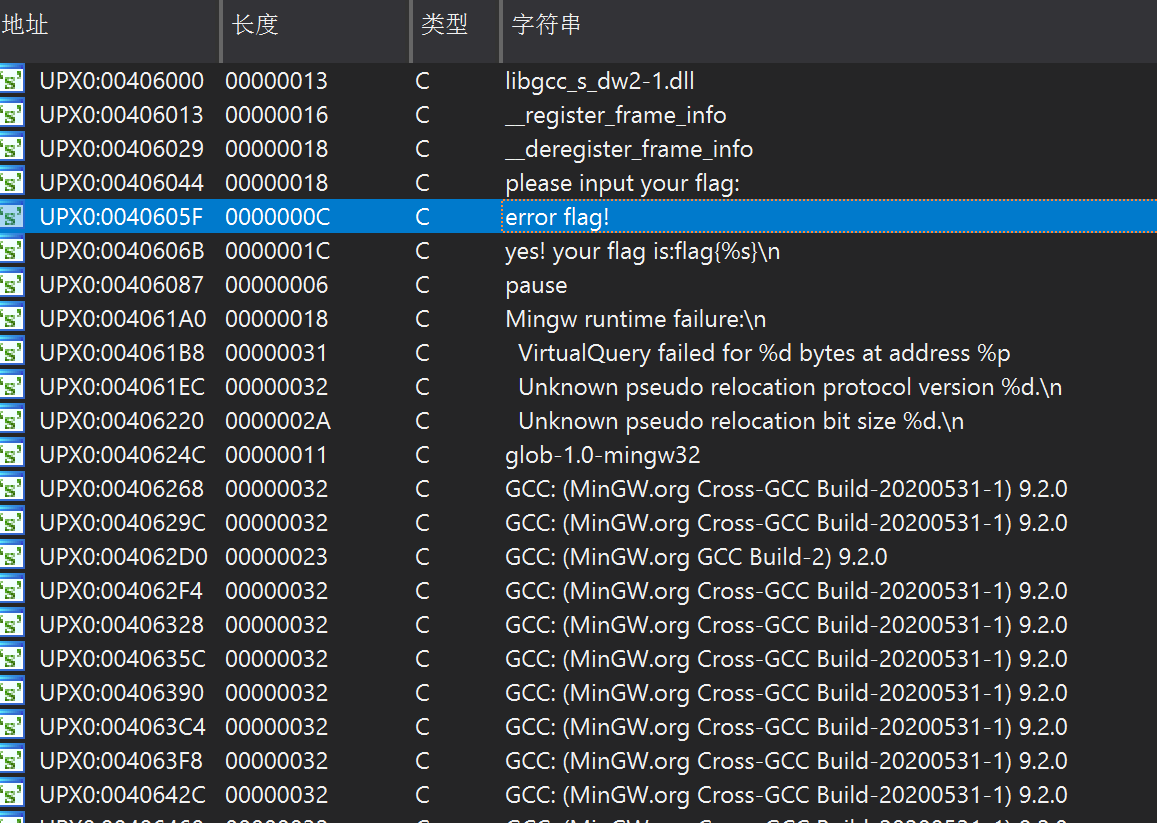


另外的验证方法是可以将没有加壳的源程序在ollydbg中打开如下，可以发现定位到的程序起始位置就是我们找到的地址。

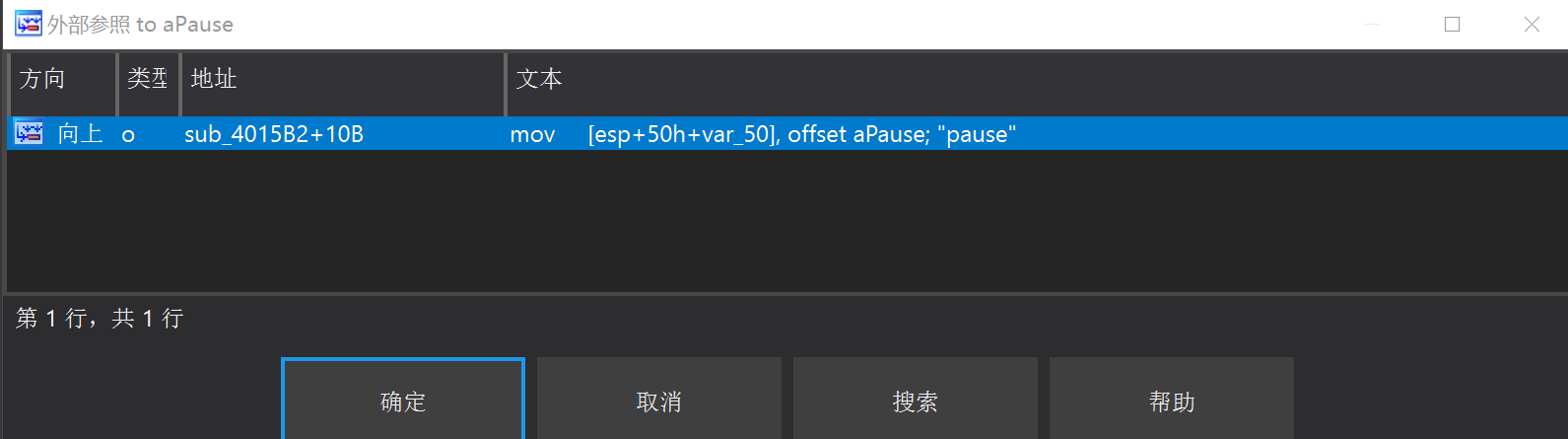


4.对脱壳后的程序使用IDA进行分析，并对IDA反编译后的伪C代码进行注释、变量重命名，截图，需要操作的函数为：main函数、加密函数

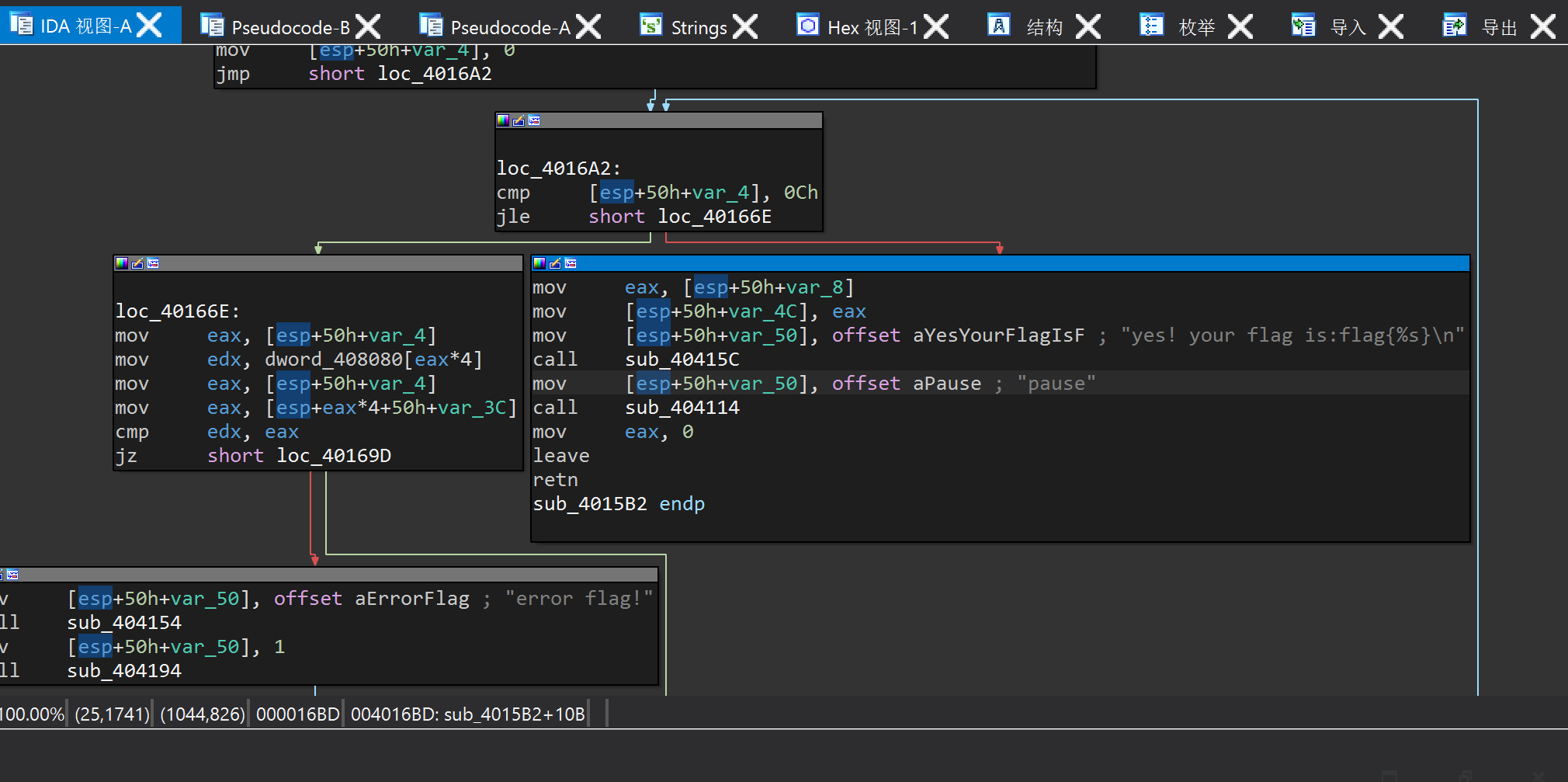
将脱壳后的程序打开之后shift+F12打开字符串表



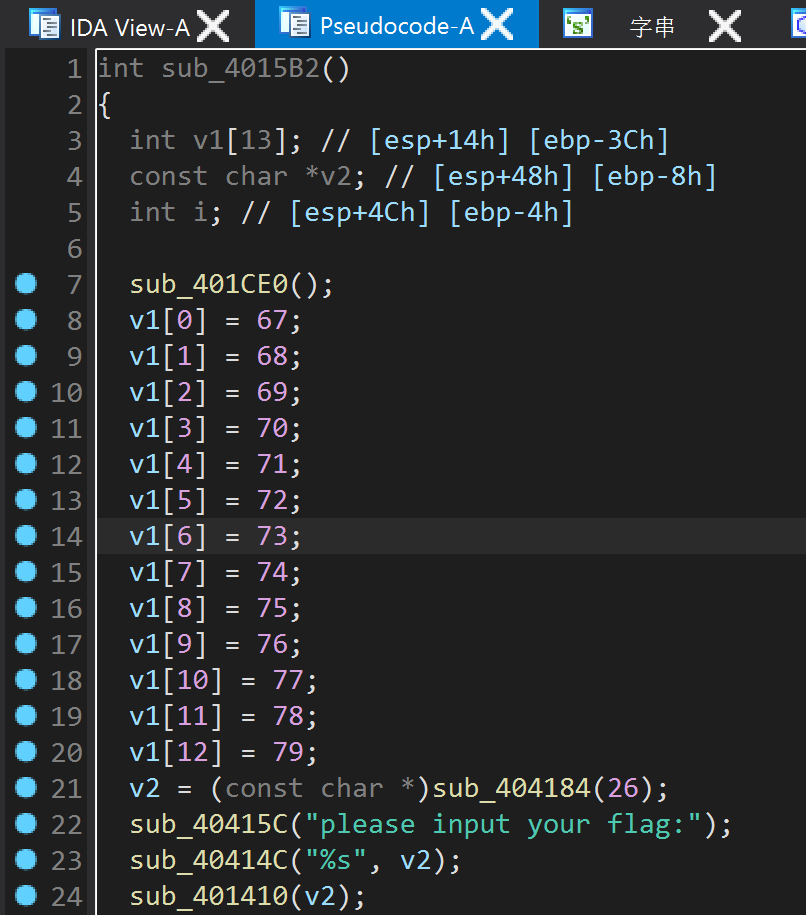
进入到 error flag部分之后找到一个函数按下x

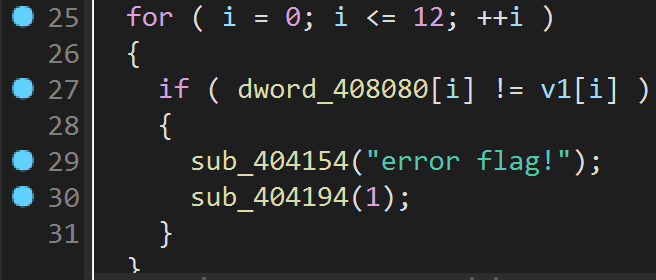


进入到对应的函数关系图

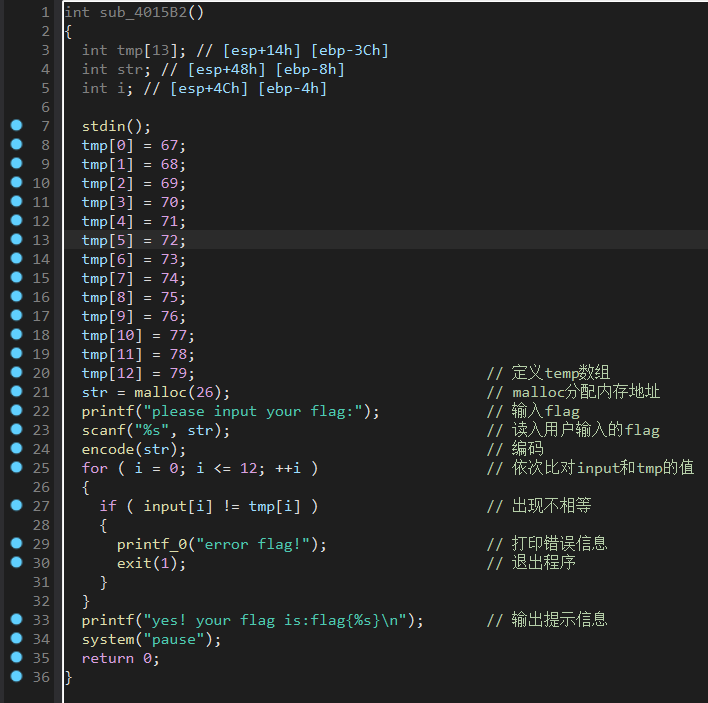


之后按下F5得到反汇编之后的C代码





下面对C代码进行注释,在注释的过程中可能是因为在去壳的过程中可执行文件损坏的原因导致第二个printf不能与其他printf同时命名，因此在这里将其命名为printf\_0



1. 编写python脚本，求解出flag的内容

实验要求：

1. 实验的关键步骤需要截图，并将相应代码粘贴至报告中。

报告中的代码使用等宽字体，对齐使用四个空格，禁止使用tab或双空格

#include <stdio.h>

#include <stdlib.h>

#include <Windows.h>

int input[13];

void encode(const char\* str) {

int xor[13];

for (int i = 0; i < 13; i++)

{

char str\_to\_int[3];

memset(str\_to\_int, '\0', 3);

strncpy(str\_to\_int, str + 2\*i, 2);

int decimal = strtol(str\_to\_int, NULL, 16);

input[i] = decimal;

}

memcpy(xor, input, sizeof(input));

for (int m = 11; m >= 0; m--)

{

input[m] = xor[m] ^ xor[m+1];

}

\_\_int64 key[] = {2025302181001,2025302181002,2025302181003};

int len\_key = sizeof(key)/sizeof(key[0]);

for (int l = 0; l < sizeof(input); l++) {

for (int j = 0; j < len\_key; j++) {

input[l] = input[l] ^ (int)(key[j] & 0xFF);

}

}

}

int main(int argc, char\* argv[])

{

int tmp[13] = {67,68,69,70,71,72,73,74,75,76,77,78,79};

const char\* str = (char\*)malloc(26);

printf("please input your flag:");

scanf("%s",str);

encode(str);

for (int j = 0; j < 13; j++)

{

if (input[j] != tmp[j]) {

printf("error flag!\n");

exit(1);

}

}

printf("yes! your flag is:flag{%s}\n",str);

system("pause");

return 0;

}