LAB4

一． 实验要求：

将给出的 C 代码按要求编译成 LC3 汇编语言，然后汇编成.obj 文件

C 代码如下：

```
typedef int i16;
typedef unsigned int u16;
i16 func(i16 n, i16 a, i16 b, i16 c, i16 d, i16 e, i16 f){ //Lots of arguments
    i16 t = GETC() - '0' + a + b + c + d + e + f;
    if(n > 1){
        i16 x = func(n - 1, a, b, c, d, e, f);
        i16 y = func(n - 2, a ,b, c, d, e, f);
        return x + y + t - 1;
    }
    else{
        return t;
    }
}
i16 main(void){
    i16 n = GETC() - '0';
    return func(n, 0, 0, 0, 0, 0, 0);
}
_Noreturn void __start(){
/*
Here is where this program actually starts executing.
Complete this function to do some initialization in your compiled assembly. TODO: Set up C runtime.
*/
    u16 __R0 = main(); //The return value of function main() should be moved to R0.
    HALT();
```

```
}
```

二． 初始化过程：

将栈指针 R6,以及装载参数部分地址的 R2 初始化即可：

```
LD R2,Argument

LD R6,BASE
```

R6 是指向工作栈的指针，R2 是指向参数地址的指针。它们的具体介绍见下文。

三． 调用约定：

由于实验要求程序会被随机加载到 x3000—xC000,故利用剩余部分来传递参数；

现规定如下：

利用空间 **xC001 到 xC007** 来传递参数。

```
Argument    .FILL xC001
```

7 个空间依次存放 n、a、b、c、d、e、f 这 7 个参数；**每次调用 FUNC 之前，都要先将这七个参数拷贝到指定位置**；

代码如下（使用 R2 寄存器记录参数位置的起始地址）

```
LDR R1,R6,#0
ADD R1,R1,#-1
STR R1,R2,#0
LDR R1,R6,#-1
STR R1,R2,#1
LDR R1,R6,#-2
STR R1,R2,#2
LDR R1,R6,#-3
STR R1,R2,#3
LDR R1,R6,#-4
STR R1,R2,#4
LDR R1,R6,#-5
STR R1,R2,#5
LDR R1,R6,#-6
STR R1,R2,#6
```

以上是递归计算 func(n-1,a,b,c,d,e,f)前拷贝参数的过程。
JSR 跳转之后，子函数会默认对应位置存放了对应参数，
直接从这个位置来读取从上一层函数传递来的参数并将其拷贝到栈上
之后，子程序便可以使用这些参数了 ~

四． 其它标准：

（1）每层函数占用的空间，使用规则如下：
按顺序依次存放 n、a、b、c、d、e、f、t、x(如果需要)、y(如果需要),以及 R7 中的内容，需要 11 个内存空间，故栈底与栈顶相差 10 即可
（虽然题干中所给代码中，a ~ f 这 6 个参数的值都为 0，但是我们依然为每层函数的各个参数单独设置了存放空间，
所以编译之后的子函数进行的第一步是为本层函数分配空间，即先将栈指针加上 11（向高地址寻址），之后在将本层函数中使用的变量拷贝到这段内存空间上

之后，在执行 “RET” 返回前，栈指针会相应的减去 11，返回之后上层函数读取的变量即可来自于正确的位置：

```
ADD R6,R6,#-11
RET
```

（2）关于调用 TRAP routines 的调用标准：
由于 TRAP 会改变寄存器 R7 的内容以作为其返回时的链接，
故我们需要再其结束后更改寄存器 R7 的值。而由上述标准，
每层函数的 linkage 已经入栈保存，所以我们调用 TRAP 之后只需从当前栈空间中读取本层函数的 R7 中的原始内容即可：

```
TRAP x20
LDR R7,R6,#-10
```

（3）关于最后 int 转 unsigned int
在将 int 型变量赋值给 unsigned int 时，内存中 16bit 的具体内容实际上是一致的，只是计算机在处理 unsigned 时会把最高

位理解为数值而非符号位。
故在本次编译得到的汇编语言中，没有显式地出现转换相关的语句。

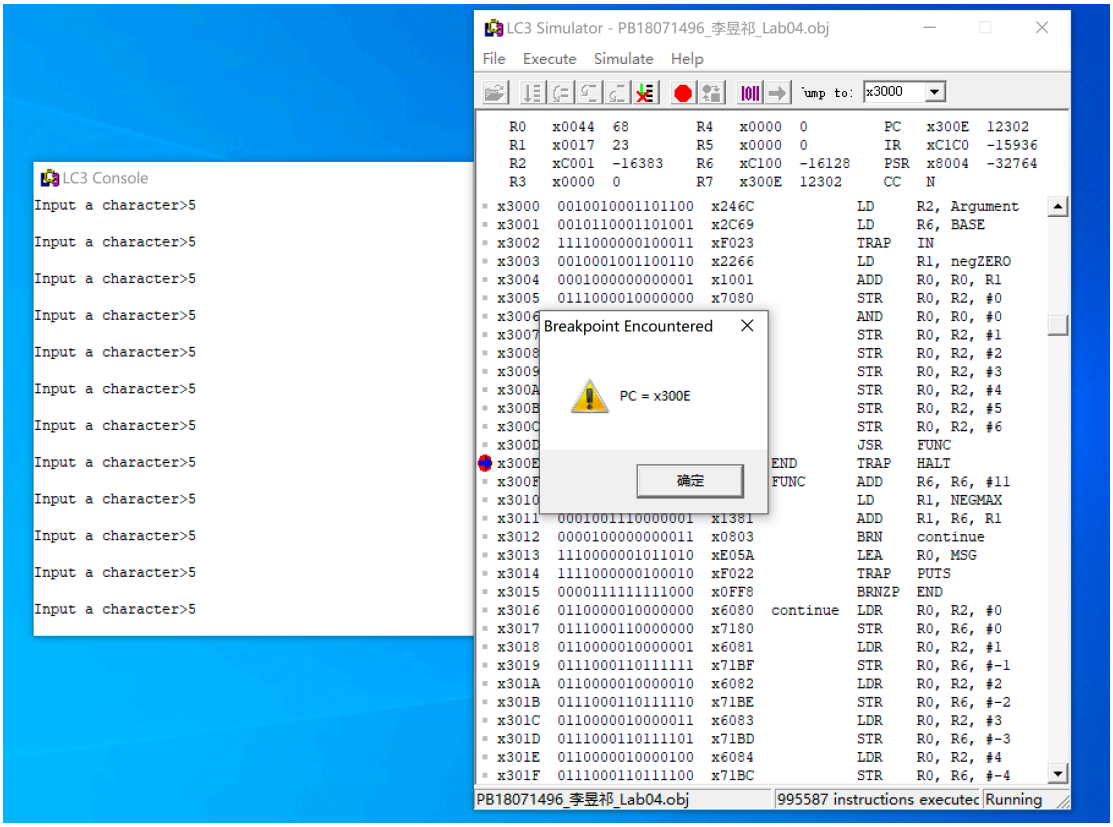
五． 错误处理：

本实验产生异常的情况为栈上溢，即函数递归次数过多导致栈溢出；
检查方法为：每次调用 FUNC 并修改栈指针后，先检查栈指针是否越界：
若未越界，再进行后续的拷贝操作；
若已经越界，则输出提示错误信息 “Stackoverflow!”，之后结束程序。

```
ADD R6,R6,#11
LD R1,NEGMAX
ADD R1,R6,R1
BRn    continue
LEA R0,MSG
PUTS
BR     END
MSG     .STRINGZ "Stackoverflow!"
NEGMAX  .FILL x1001
```

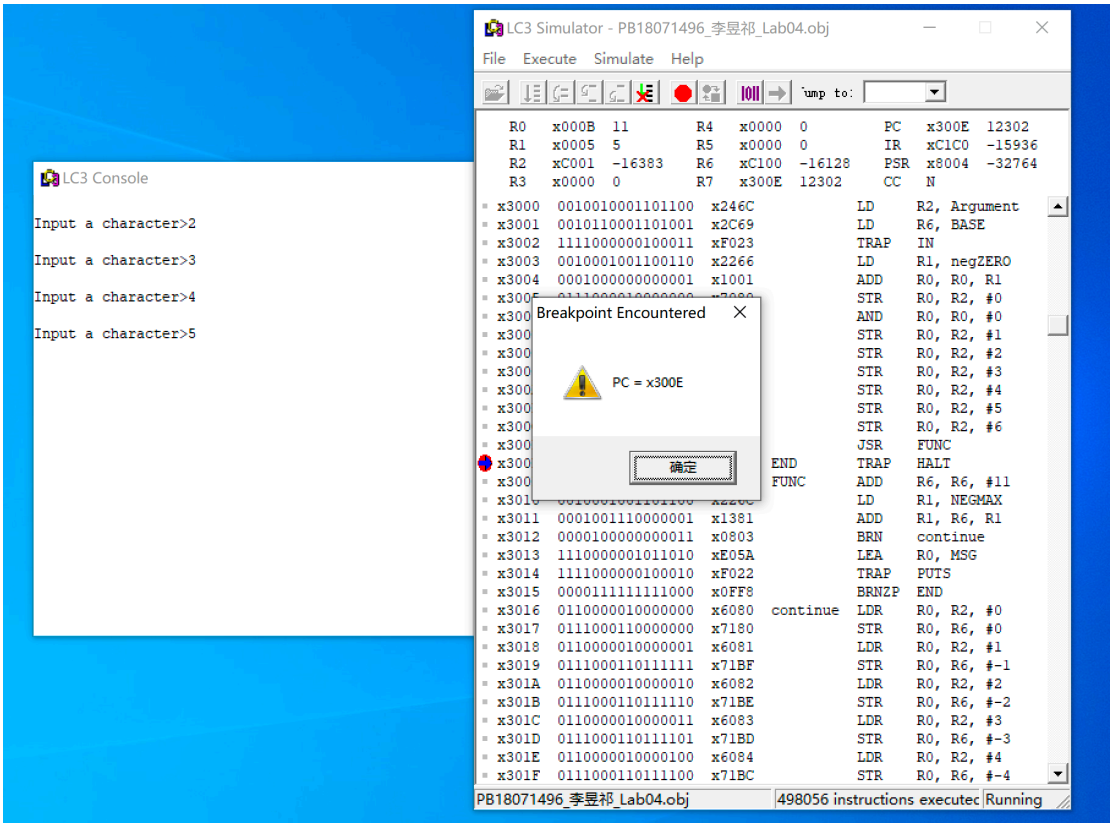
六． 运行测试

为方便显示，测试时输入使用可以回显的 TRAP x23 “IN”
输入 “555555555555……” ：
（程序执行结束后 R0 寄存器中的值为 68）



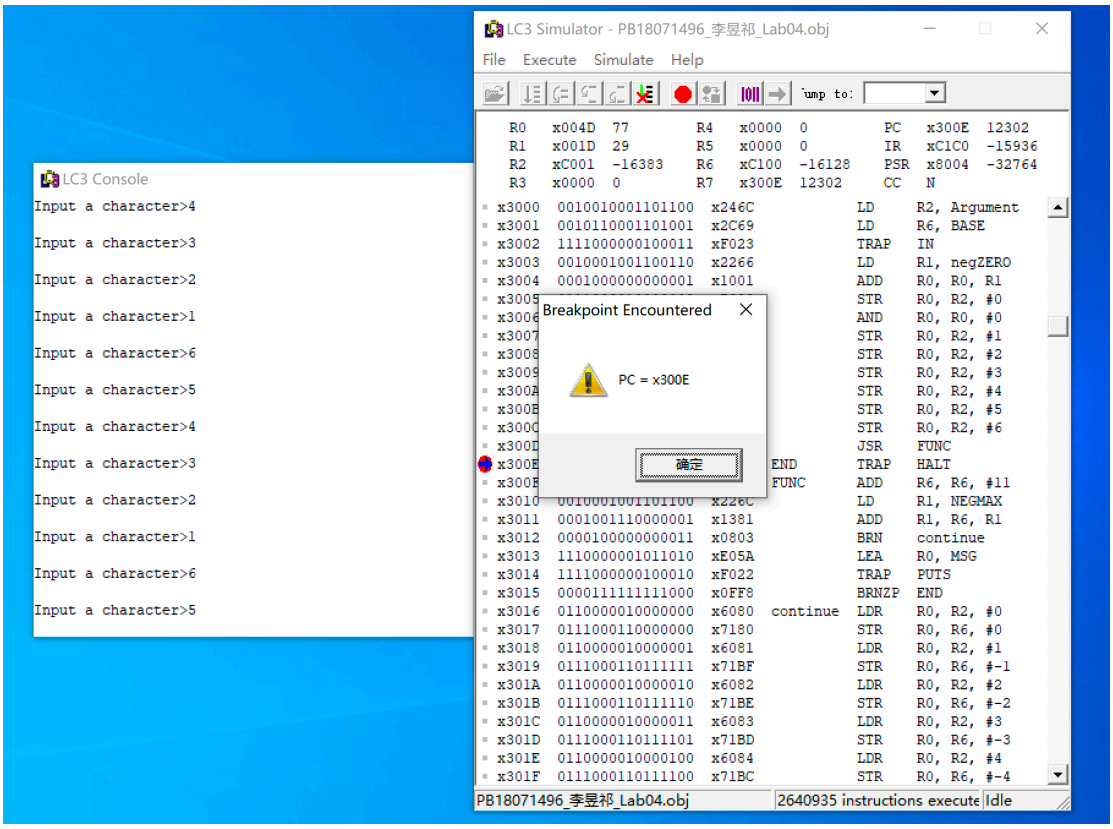
输入 “2345”：

（程序执行结束后 R0 寄存器中的值为 11）



输入“654321654321……”：

（程序执行结束后 R0 寄存器中的值为 77）



以上测试结果均与所给 c 程序编译后运行测试出的结果一致。

七． 实验小结：

（1） 通过本次实验，使用 LC-3 汇编语言，练习了使用栈来实现高级语言的递归功能；

（2） 通过栈来保存信息：在函数的调用、返回过程中，通过栈指针的增、减来寻找相应变量在内存空间中的存储位置；

（3） 熟悉了使用 TRAP 服务程序进行输入、输出操作

八． 附录

“PB18071496_李昱祁_Lab04.asm”

“PB18071496_李昱祁_Lab04.obj”