

中国科学技术大学计算机学院  
《计算机系统概论》报告



实验题目: Lab 04

学生姓名: 高楚晴

学生学号: PB18111688

完成日期: 2019.12.28

# 实验目的

使用汇编语言将给定的c语言程序人工编译成LC3汇编语言，再汇编生成LC-3可执行文件。

其c程序如下：

```
typedef int i16;
typedef unsigned int u16;
i16 func(i16n,i16a,i16b,i16c,i16d,i16e,i16f){//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);
}
_Noreturnvoid__start(){
    u16 __R0 = main();//The return value of function main() should be
moved to R0.
    HALT();
}
```

在转换为lc-3的过程中不能更改程序的执行结构，同时需要符合c标准。

## 设计思路&调用说明

该程序为递归结构，因此在用lc-3实现时需要借助栈结构来完成，由于该函数涉及多个局部变量，再对比将多个数据放入同一个递归栈的形式后，分析该程序，初始n值范围为0~9，最多递归8层，因此栈所需规模较小，单个局部变量需要栈空间小于16个。为提高可读性，使用了多个栈分别存放n,t, func(n), R7值。

由于四个栈空间无法同时将传递值和栈指针存放在寄存器中，引入一个地址空间放置t的栈指针，其余寄存器分配如下：

内容	栈顶指针	操作时数据临时存放位置	栈顶初始值
当前n值	R4	R1	xC00F
FUNC()当前值	R5	R2	xC01F
R7值（用于RET返回）	R6	R7	xC02F
当前递归层t值	m[TADDR]	R1	xC03F

由于未知函数调用之前初始的寄存器及内存值，开始时进行初始化。

对于参数 $a, b, c, d, e, f$ 开辟6个地址空间进行存放，由于在程序中始终与‘0’操作同时存在，因此在内存上与其对应ASCII值存放地址邻近，无需额外定义标签，形式如下。

```
NEGBASE .FILL    xFFD0      ;neg of "ascii of '0' "  
        .BLKW    6
```

在本例中给出的参数 $a, b, c, d, e, f$ 值均为零，因此在开始均清零初始化。

```
        AND R0,R0,#0  
        LEA R1,NEGBASE      ;初始abcdef所在内存清空  
        LD  R2,SIX  
LOOP    ADD R1,R1,#1        ;从NEGBASE的下一个地址空间开始  
        STR R0,R1,#0  
        ADD R2,R2,#-1  
        BRp LOOP
```

由于c程序中没有给出参数 $a, b, c, d, e, f$ 的具体赋值方式，因此本汇编中不添加额外的键盘读入操作，如果使用者希望修改参数 $a, b, c, d, e, f$ 值，仅需要修改该部分内存初始化过程。

## 代码讲解

### main

该部分对应c中的main部分。

```
i16 main(void){  
    i16 n = GETC()-'0';  
    return func(n, 0, 0, 0, 0, 0, 0);  
}
```

尽管实验要求中说明起始位置并不确定，可存在于x3000-xC000中任意位置，但为方便运行，此处以x3000作起始值为例，该部分主要用于初始化各内存以及为跳转FUNC模块提供出入口，并将其与结尾HALT部分链接。

```
.ORIG x3000  
LD    R4,NSTACK  
LD    R5,FSTACK  
LD    R6,PSTACK  
TRAP   x23  
LD    R1,NEGMAX      ;检查输入上溢  
ADD    R1,R1,R0  
BRp    ERROR  
LD    R1,NEGBASE      ;检查输入下溢  
ADD    R0,R0,R1  
BRn    ERROR  
ADD    R4,R4,#-1      ;push n  
STR    R0,R4,#0  
AND    R0,R0,#0  
LEA    R1,NEGBASE      ;初始abcdef所在内存清空  
LD    R2,SIX
```

```

LOOP    ADD R1,R1,#1
        STR R0,R1,#0
        ADD R2,R2,#-1
        BRp LOOP
        JSR FUNC
        ADD R0,R2,#0    ;将返回值存入R0
        BR  END

```

其中几个初始赋值时用到的常量如下。

```

SEVEN   .FILL    #7
SIX     .FILL    #6
TADDR   .FILL    xC00F
PSTACK  .FILL    xC01F
NSTACK  .FILL    xC02F
FSTACK  .FILL    xC03F
NEGMAX  .FILL    xFFC7    ;neg of "ascii of '9' "
NEGBASE .FILL    xFFD0    ;neg of "ascii of '0' "
        .BLKW    6
ERR     .FILL    x000A
        .STRINGZ "ERROR:input is out of range"

```

ERROR模块用于输入字符不合法时跳转，并终止程序运行，实现如下。

```

ERROR   LEA R0,ERR
        TRAP    x22
END     TRAP    x25

```

## FUNC

由于FUNC函数篇幅较长，将其拆分介绍。

```

FUNC    ADD R6,R6,#-1    ;保存R7值，便于函数返回
        STR R7,R6,#0
        TRAP    x23
        LD  R2,SIX
        LD  R1,NEGMAX    ;上溢检查
        ADD R1,R1,R0
        BRp ERROR
        LEA R1,NEGBASE    ;ASCII -> 对应十进制数
        LDR R3,R1,#0
        ADD R0,R3,R0
        BRn ERROR
CIRCLE  ADD R1,R1,#1    ; +a+b+c+d+e+f
        LDR R3,R1,#0
        ADD R0,R3,R0
        ADD R2,R2,#-1
        BRp CIRCLE

```

上述部分还可以进一步优化，由于在递归时，CIRCLE模块需多次调用，而a,b,c,d,e,f始终作为整体出现，故可以引入地址空间SUM，只需在第一次调用时计算 $sum = a + b + c + d + e + f$ ，此后每次简化为加sum值即可，具体实现如下。

```

LD R2,SIX
AND R0,R0,#0
CIRCLE ADD R1,R1,#1      ; R0 = a+b+c+d+e+f
LDR R3,R1,#0
ADD R0,R3,R0
ADD R2,R2,#-1
BRp CIRCLE
ST R0,SUM                ;初次调用时

...

TRAP x23                ;此后的每次调用
LD R3,SUM
ADD R0,R3,R0

...

SUM .BLKW 1

```

下面是递归的主体部分，先进行if条件判断，为简化程序执行，不对第一次调用特殊化，在进入FUNC之前预先push n的值，此后每次调用时先pop得到当前n的值。

```

LD R1,TADDR      ;push t
ADD R1,R1,#-1
STR R0,R1,#0
ST R1,TADDR
LDR R1,R4,#0      ;pop n
ADD R4,R4,#1
ADD R2,R1,#-1      ;判断if(n>1)
BRnz ELSE

```

下面是进入if结构的部分，通过栈递归得到func(n-1)和func(n-2)值。

```

ADD R4,R4,#-1      ;push n
STR R1,R4,#0
ADD R1,R1,#-1
ADD R4,R4,#-1
STR R1,R4,#0      ;push n-1
JSR FUNC
ADD R5,R5,#-1
STR R2,R5,#0      ;push FUNC(N-1)
LDR R1,R4,#0
ADD R4,R4,#1
ADD R1,R1,#-2
ADD R4,R4,#-1      ;push n-2
STR R1,R4,#0

```

```

JSR FUNC      ;func(n-2)
LDR R3,R5,#0  ;pop func(n-1)
ADD R5,R5,#1
ADD R2,R2,R3   ;x+y
LD  R1,TADDR  ;pop t
LDR R0,R1,#0
ADD R1,R1,#1
ST  R1,TADDR
ADD R2,R0,R2   ;X+Y+T
ADD R2,R2,#-1  ;x+y+t-1
BR  RESTORE    ;跳转至恢复R7值的部分

```

下面是else部分，由于讲解时完全按照程序顺序切分，该部分在最后，即紧接着恢复R7值便于返回。

```

ELSE  ADD R2,R0,#0  ;R0未改变
      LD  R1,TADDR
      ADD R1,R1,#1   ;pop t
      ST  R1,TADDR
RESTORE LDR R7,R6,#0 ;恢复R7
      ADD R6,R6,#1
      RET

```

RET将pc值返回到R7值，即main函数中FUNC下一句，此时返回值存在R2中，在主函数中赋给R0即可。

## 调试分析

汇编通过，模拟时发现结果与理论值不符，经单步执行分析后发现有两处错误，一为一处ADD操作应为 `ADD R1,R1,R0` 误写成 `ADD R1,R0,#0`，另一处为执行POP操作时，栈指针移动方向写错。

修改后程序运行正常，分析其复杂度，时间复杂度中，递归主体部分复杂度最高，为 $O(n^2)$ ，其余操作复杂度均在 $O(n)$ 之内，因此总时间复杂度为 $O(n^2)$ 。空间复杂度上，其中栈部分空间复杂度为 $O(n)$ ，其余空间复杂度均在常数级，总的空间复杂度为 $O(n)$ 。

## 异常处理

### 输入异常

通过阅读c语言，我们发现输入值只能在'0'~'9'之间，不然与理论值不符，因此通过检查上下溢出并在错误时输出提示。具体实现在“代码讲解”部分已给出，此处给出测试展示。

```

Input a character>4
Input a character>+
ERROR:input is out of range

```

下溢出

```
Input a character>F

ERROR:input is out of range
----- Halting the processor -----
```

上溢出

## 栈空间异常

由于本实验中，为节省执行PUSH,POP函数所产生的RET保存R7步骤，且实验中用到了多个栈，设置函数也较为繁琐，并没有单独设置PUSH与POP模块，只伴随在函数顺序执行语句中且出现较为频繁，为了避免大量检测栈上下溢出语句降低可读性，同时由于没有单独提供用户单独使用PUSH与POP操作的权限，因此在程序正常执行时用户不存在触发栈溢出的情况，并没有提供栈溢出检测。

给出一个最大栈空间占用情况，即n=9时执行后栈结果，全部输入均为5.

» xC020	0000000000000000	x0000	NOP
» xC021	0000000000000000	x0000	NOP
» xC022	0000000000000000	x0000	NOP
» xC023	0000000000000000	x0000	NOP
» xC024	0000000000000000	x0000	NOP
» xC025	0000000000000000	x0000	NOP
» xC026	0000000000000001	x0001	NOP
» xC027	0000000000000000	x0000	NOP
» xC028	0000000000000001	x0001	NOP
» xC029	0000000000000000	x0000	NOP
» xC02A	0000000000000001	x0001	NOP
» xC02B	0000000000000000	x0000	NOP
» xC02C	0000000000000001	x0001	NOP
» xC02D	0000000000000000	x0000	NOP
» xC02E	0000000000000001	x0001	NOP
» xC02F	0000000000000000	x0000	NOP
» xC030	0000000000000000	x0000	NOP
» xC031	0000000000000000	x0000	NOP
» xC032	0000000000000000	x0000	NOP
» xC033	0000000000000000	x0000	NOP
» xC034	0000000000000000	x0000	NOP
» xC035	0000000000000000	x0000	NOP
» xC036	0000000000000000	x0000	NOP
» xC037	0000000000000000	x0000	NOP
» xC038	0000000000000000	x0000	NOP
» xC039	0000000000000000	x0000	NOP
» xC03A	0000000000000000	x0000	NOP
» xC03B	0000000000001110	x000E	NOP
» xC03C	0000000000101001	x0029	NOP
» xC03D	0000000001110001	x0071	NOP
» xC03E	0000000100101110	x012E	NOP
» xC03F	0000000000000000	x0000	NOP

发现并没有出现溢出现象，运行正常。

## 代码测试

测试一组数据，结果如下，与理论值一致。

```
Input a character>4
Input a character>3
Input a character>2
Input a character>1
Input a character>2
Input a character>1
Input a character>3
Input a character>2
Input a character>1
Input a character>2
```

Register	Value (Hex)	Value (Dec)
R0	x000D	13
R1	xC7FF	-14337
R2	x000D	13
R3	x0007	7

  

Address	Value (Hex)	Value (Bin)
x3000	0010100001001101	
x3001	0010101001001101	
x3002	0010110001001010	
x3003	1111000000100011	
x3004	0010001001001011	
x3005	0001000000000001	
x3006	0000100000000000	
x3007	0001100100110011	
x3008	0111000100000000	
x3009	0101000000010010	
x300A	1110001001001000	
x300B	0010010000110010	
x300C	0001001001101100	

实验总结

通过本次实验对lc3实现c语言中递归的方法有了更好地理解。

如果扩展本实验可以接受多位字符输入或者接受a-f输入。

附录

PB18111688\_高楚晴\_Lab04.asm

PB18111688\_Lab04.obj