中国科学技术大学计算机学院
《计算机系统概论》实验报告

实验题目：HUMANOID COMPILER

学生姓名：王章瀚

学生学号：PB18111697

完成日期：2019 年 12 月 28 日

计算机实验教学中心制

2019 年 09 月

1 实验要求

1.1 总述

手动完成给定 C 语言代码到 LC-3 汇编码的编译，并将它编译为 LC-3 的 object 文件。

1.2 C 代码源码

```c
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();
}
```

1.3 细节要求

- 应按照 C 标准，并完全遵照原代码进行翻译。不要对代码做任何优化，甚至改变其控制流结构。
- 不得使用任何第三方 C 编译器。
- ___start() 不是一个函数。它标志着程序由此开始，并调用 main 函数。由于最开始寄存器和内存空间都是随机的，你需要自己设计并完成初始化过程。这个过程应该要是通用的，这意味着它和 main 函数或其他函数的内容无关。
- GETC() 和 HALT() 是 TRAP。因此请做出一个 TRAP 应该遵循的标准，使得你的程序不会被打乱。请注意你程序中所有的调用也应该遵循

这个标准。

- 如果有程序想调用你的程序中的 func()，它应该遵循某些规则，请设计这些规则。请注意你的程序里的调用也应该遵循这个规则。
- 你的程序会被随机地载入到 x3000-xC000。因此 object 文件的.ORIG 和前两个字节将被忽略。
- 实验报告应该包含：1. 初始化过程；2. 调用约定；3. 其他你设计的标准；4. 错误处理。

2 设计思路

2.1 总体思路

观察可知，C 源码中，除了 main 函数，只有一个 func 函数，而该函数会被递归调用。因此需要考虑函数栈。其他的由于是翻译，并没有太多需要注意的，只需要照着翻译即可。而为了方便，不再按照书上使用 FramePointer 的方法，而是直接通过函数栈指针（这里用 R6）的偏移来访问每个栈元素的各个子元素。

2.2 初始化过程

此处似乎并不需要做任何初始化。但为了好看，将所有寄存器初始化为 0，并读取栈指针位置到 R6。

```
1 AND R0, R0, #0
2 AND R1, R1, #0
3 AND R2, R2, #0
4 AND R3, R3, #0
5 AND R4, R4, #0
6 AND R5, R5, #0
7 AND R6, R6, #0
8 AND R7, R7, #0
9 LD  R6, func_stack  ;读取栈初始地址
```

2.3 调用约定

2.3.1 TRAP 标准

- 程序中所有有调用了 TRAP 的地方，都应该**提前保存 R7**，并在需要用到之前将 R7 重新载回。（例如程序中使用的 GETC，应保存 R7，否则

将无法正常返回。）

2.3.2 func 函数标准

- 要想使用 func 函数，必须有一个寄存器或一个内存空间来**保存 func 函数栈指针**。本例中始终用 R6 保存。
- 本函数**不对寄存器进行任何保存处理**，如有需要，请在调用前自行保存
- 函数栈空间说明：

栈空间起始位置是由汇编代码：**func_stack .FILL xD000** 决定，起始位置即该处值。如此时，则从 **xD000** 开始。

栈空间大小由汇编代码**:stack_size .FILL x1C2C** 决定。大小为该处值-xC。如此时，大小为 x1C2C-xC=**x1C20**。

栈空间中，每个元素为一块，**每块 12 个内存空间**，其含义如下表：

每块中地址	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB
含义	返回值	返回地址	n	a	b	c	d	e	f	t	x	y

- 在调用 func 函数之前，应使 R6 向后偏移 **12** 个内存地址空间，并将**所有参数通过 R6 对应入栈**。例如：将参数 a 的值存入偏移后 R6+3 处。
- 函数的返回：函数返回时，R6 **不会**回退到调用处的栈位置，这是为了读取返回值方便，**如调用 func，请手动进行回退**；如需要读取返回值，可以通过 **R6+x0** 获取。

2.4 错误处理

设计中，会出现的错误在于栈空间溢出，即超出了前述 stack_size-xC 的大小。对此进行的处理是，每次调用时检测 **R6-栈起始位置** 是否达到了栈空间大小，如果达到，则输出”Stack OverFlow!”。

3 关键代码讲解

因为是翻译，没有什么比较特别的地方需要特别说明，因此以下只将代码贴出，并在后面按行说明。

3.1 func 函数

先放出代码：

```
1  func            STR R7, R6, #1  ; 存好R7所保存的返回地址
2                  GETC            ; 计算t开始
3                  LD  R1, c_n30
4                  ADD R0, R0, R1  ; t-='0'
5                  LDR R1, R6, #3  ; 从内存读取a
6                  ADD R0, R0, R1  ; 加上a
7                  LDR R1, R6, #4  ; 从内存读取b
8                  ADD R0, R0, R1  ; 加上b
9                  LDR R1, R6, #5  ; 从内存读取c
10                 ADD R0, R0, R1  ; 加上c
11                 LDR R1, R6, #6  ; 从内存读取d
12                 ADD R0, R0, R1  ; 加上d
13                 LDR R1, R6, #7  ; 从内存读取e
14                 ADD R0, R0, R1  ; 加上e
15                 LDR R1, R6, #8  ; 从内存读取f
16                 ADD R0, R0, R1  ; 加上f，计算t结束
17                 STR R0, R6, #9  ; 存放t
18
19                 LDR R1, R6, #2  ; 读取n
20                 ADD R2, R1, #-1 ; 计算n-1
21                 BRnz    ELSE    ; 若n-1<=0,跳转到ELSE
22
23                 ADD R6, R6, #12 ; 将R6指向下一个栈空间，为下一次递归调用做准备
24                 LD  R3, stack_size  ; 读取栈空间总大小
25                 LD  R4, func_stack  ; 读取栈空间起始位置
26                 ADD R3, R3, R4  ; 计算栈空间末尾位置
27                 NOT R3, R3
28                 ADD R3, R3, #1  ; 计算栈空间末位置的相反数
29                 ADD R3, R3, R6  ; R6-栈空间末位置
30                 BRz stackoverflow   ; 栈溢出
31                 LDR R1, R6, #-10    ; 从内存读取n
32                 ADD R2, R1, #-1 ; 求n-1
33                 STR R2, R6, #2  ; 存到R6+#2上
34                 LDR R1, R6, #-9 ; 从内存读取a
35                 STR R1, R6, #3  ; 存到R6+#3上
36                 LDR R1, R6, #-8 ; 从内存读取b
37                 STR R1, R6, #4  ; 存到R6+#4上
38                 LDR R1, R6, #-7 ; 从内存读取c
39                 STR R1, R6, #5  ; 存到R6+#5上
40                 LDR R1, R6, #-6 ; 从内存读取d
41                 STR R1, R6, #6  ; 存到R6+#6上
42                 LDR R1, R6, #-5 ; 从内存读取e
43                 STR R1, R6, #7  ; 存到R6+#7上
44                 LDR R1, R6, #-4 ; 从内存读取f
45                 STR R1, R6, #8  ; 存到R6+#8上
46                 JSR func        ; 递归调用以求x
47                 LDR R0, R6, #0  ; 读取递归调用求x得到的返回值
48                 STR R0, R6, #-2 ; 存在当前栈的x空间位置上
49                 LDR R1, R6, #-10    ; 从内存读取n
50                 ADD R2, R1, #-2 ; 求n-2
```

```
51                  STR R2, R6, #2  ; 存到 R6+#2 上
52                  ; 其余R1到R6上一次均已经存好，且未改变
53                  JSR func        ; 递归调用以求y
54                  LDR R0, R6, #0  ; 读取递归调用求y得到的返回值
55                  STR R0, R6, #-1 ; 存在当前栈的y空间位置上
56
57                  ADD R6, R6, #-12    ; 栈指针指回当前调用函数对应栈
58
59                  LDR R0, R6, #9  ; 读取t到R0
60                  LDR R1, R6, #10 ; 读取x到R1
61                  ADD R0, R0, R1  ; 求t+x
62                  LDR R1, R6, #11 ; 读取y到R1
63                  ADD R0, R0, R1  ; 求t+x+y
64                  ADD R0, R0, #-1 ; 求t+x+y-1
65                  STR R0, R6, #0  ; 返回值存在R6+#0
66
67                  LDR R7, R6, #1  ; 读取R7所保存的返回地址
68                  RET             ; 以上完成了n>1的情况
69                  ;以下是n<=1的情况
70  ELSE            LDR R0, R6, #9  ; 返回t,把t的值读入R0
71                  STR R0, R6, #0  ; 返回值存在R6+#0
72                  LDR R7, R6, #1  ; 读取R7所保存的返回地址
73                  RET             ; n<=1，直接返回
74  stackoverflow   LEA R0, sof_message
75                  PUTS
76                  HALT
77
78  sof_message     .STRINGZ "Stack OverFlow!"
79  c_n30           .FILL   x-30        ; '0'的ASCII的相反数
80  stack_size      .FILL   x1C2C       ; 栈的大小+#12, x1C2C+#12=#7212
81  func_stack      .FILL   xD000       ; 为func函数的栈保留空间
```

以下按行说明代码：

•Line1-2　存 R7, GETC 读数

•Line3-17　计算 t，并存入函数栈空间

•Line19-21　分支语句，判断 n

•Line23-30　n>1，准备递归调用，判断是否栈溢出

•Line31-45　传参过程

•Line46-48　调用函数，保存返回值到 x

•Line49-51　传参过程

•Line53-55　调用函数，保存返回值到 y

•Line57　func 栈指针回退

•Line59-65　计算 t+x+y-1，并存在返回值位置上

•Line67-68　n>1 时的返回

•Line70-73　n<=1 时的返回

•Line74-76　栈溢出处理

以上就是对 func 函数的编写进行的阐述，代码注释也比较清晰。

3.2 main 函数

先放出代码：

```
main        ST   R7, main_R7 ;存好R7所保存的返回地址
            GETC
            LD   R1, c_n30
            ADD R1, R0, R1   ;计算初始n
            LD   R6, func_stack   ;读取栈初始地址
            STR R1, R6, #2   ;把n存到R6+#1上
            AND R1, R1, #0   ;初始化R1为0
            STR R1, R6, #3   ;a处存为0
            STR R1, R6, #4   ;b处存为0
            STR R1, R6, #5   ;c处存为0
            STR R1, R6, #6   ;d处存为0
            STR R1, R6, #7   ;e处存为0
            STR R1, R6, #8   ;f处存为0
            JSR func         ;调用func
            LDR R0, R6, #0   ;读取返回值
            ADD R6, R6, #-12     ;R6回退
            ST   R0, main_ret    ;返回值写到main函数返回值处
            LD   R7, main_R7
            RET
main_ret    .BLKW   1
main_R7     .BLKW   1
```

这段代码对 func 函数进行了一次调用，中间一大部分都是在进行传参操作。

其他则是一些初始化，返回等操作，已注释清楚。

4 调试分析

4.1 正常数据测试

设当 main 函数中，输入为字符 c 时，后面还需要输入 n(c) 个字符。

通过计算，可以知道

$$\begin{cases} n(0) = 1 \\ n(1) = 1 \\ n(c+2) = n(c+1) + n(c) + 1 \end{cases}$$

这是后续数据选取的依据。

以下随机选取了两组测试数据及取了一组 00 的测试数据：

4.1.1 测试数据 1

测试时输入：54896518461sagg6

可以看到最终 R0 得到了：277.

这与实际结果相同：

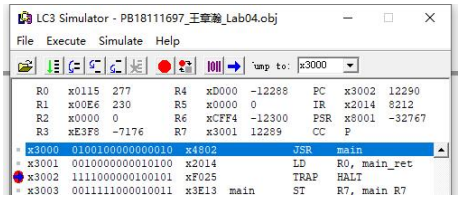

图 1: 测试结果

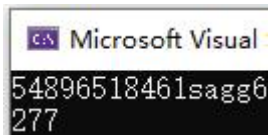

图 2: 实际结果

4.1.2 测试数据 2

测试时输入：6f14ge687941aeo6u52f13v267

可以看到最终 R0 得到了：580.

这与实际结果相同：

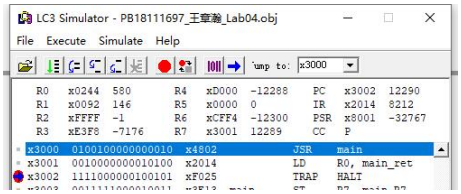

图 3: 测试结果

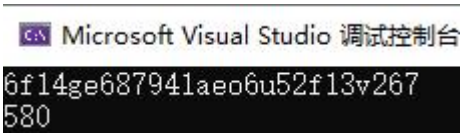

图 4: 实际结果

4.1.3 测试数据 3

测试时输入：00

可以看到最终 R0 得到了：0.

这与实际结果相同：

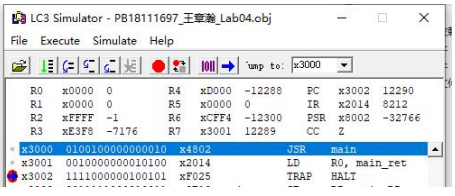

图 5: 测试结果

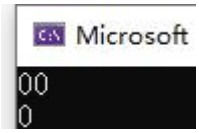

图 6: 实际结果

4.2 异常数据测试

由于输入不便，测试栈溢出的时候，缩小栈空间来测试。设置栈空间为 #36-xB=#24，令输入为 5555，则会有如下提示：

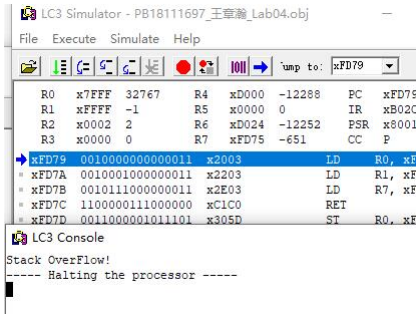

图 7: 函数栈溢出

5 Great Idea

本程序中，将函数中所有的参数，变量都当作一个栈元素的整体压入栈中，这将对程序日后需要的拓展提供了很大方便。且这样的思路，是完全可以作为 C to LC-3 汇编码的编译器中函数的处理思路。因为它所有的变量读取与赋值，不是依赖于寄存器的使用，而主要是直接靠内存读写。

6 实验总结

本次实验中，实现了函数的递归调用操作，这其中运用的栈思想是十分重要的。通过栈的使用，可以避免参数丢失，又能满足函数调用的 FILO 的规律。相信它在日后的计算机的学习中，将有很大作用。

7 附录：完整的 LC-3 汇编代码

```
1                .ORIG    x3000
2               AND R0, R0, #0
3               AND R1, R1, #0
4               AND R2, R2, #0
5               AND R3, R3, #0
6               AND R4, R4, #0
7               AND R5, R5, #0
8               AND R6, R6, #0
```

```
9                   AND R7, R7, #0
10                  JSR main
11                  LD  R0, main_ret      ;读取返回值
12                  HALT
13
14  main            ST  R7, main_R7 ;存好R7所保存的返回地址
15                  GETC
16                  LD  R1, c_n30
17                  ADD R1, R0, R1  ;计算初始n
18                  LD  R6, func_stack  ;读取栈初始地址
19                  STR R1, R6, #2  ;把n存到R6+#1上
20                  AND R1, R1, #0  ;初始化R1为0
21                  STR R1, R6, #3  ;a处存为0
22                  STR R1, R6, #4  ;b处存为0
23                  STR R1, R6, #5  ;c处存为0
24                  STR R1, R6, #6  ;d处存为0
25                  STR R1, R6, #7  ;e处存为0
26                  STR R1, R6, #8  ;f处存为0
27                  JSR func            ;调用func
28                  LDR R0, R6, #0  ;读取返回值
29                  ADD R6, R6, #-12      ;R6回退
30                  ST  R0, main_ret      ;返回值写到main函数返回值处
31                  LD  R7, main_R7
32                  RET
33  main_ret        .BLKW   1
34  main_R7         .BLKW   1
35
36  func            STR R7, R6, #1  ;存好R7所保存的返回地址
37                  GETC                ;计算t开始
38                  LD  R1, c_n30
39                  ADD R0, R0, R1  ;t-='0'
40                  LDR R1, R6, #3  ;从内存读取a
41                  ADD R0, R0, R1  ;加上a
42                  LDR R1, R6, #4  ;从内存读取b
43                  ADD R0, R0, R1  ;加上b
44                  LDR R1, R6, #5  ;从内存读取c
45                  ADD R0, R0, R1  ;加上c
46                  LDR R1, R6, #6  ;从内存读取d
47                  ADD R0, R0, R1  ;加上d
48                  LDR R1, R6, #7  ;从内存读取e
49                  ADD R0, R0, R1  ;加上e
50                  LDR R1, R6, #8  ;从内存读取f
51                  ADD R0, R0, R1  ;加上f，计算t结束
52                  STR R0, R6, #9  ;存放t
53
54                  LDR R1, R6, #2  ;读取n
55                  ADD R2, R1, #-1 ;计算n-1
56                  BRnz    ELSE            ;若n-1<=0,跳转到ELSE
57
58                  ADD R6, R6, #12 ;将R6指向下一个栈空间，为下一次递归调用做准备
59                  LD  R3, stack_size  ;读取栈空间总大小
60                  LD  R4, func_stack  ;读取栈空间起始位置
61                  ADD R3, R3, R4  ;计算栈空间末尾位置
62                  NOT R3, R3
63                  ADD R3, R3, #1  ;计算栈空间末位置的相反数
64                  ADD R3, R3, R6  ;R6-栈空间末位置
65                  BRz stackoverflow     ;栈溢出
66                  LDR R1, R6, #-10      ;从内存读取n
67                  ADD R2, R1, #-1 ;求n-1
68                  STR R2, R6, #2  ;存到R6+#2上
69                  LDR R1, R6, #-9 ;从内存读取a
70                  STR R1, R6, #3  ;存到R6+#3上
71                  LDR R1, R6, #-8 ;从内存读取b
```

```
72                  STR R1, R6, #4   ;存到R6+#4上
73                  LDR R1, R6, #-7  ;从内存读取c
74                  STR R1, R6, #5   ;存到R6+#5上
75                  LDR R1, R6, #-6  ;从内存读取d
76                  STR R1, R6, #6   ;存到R6+#6上
77                  LDR R1, R6, #-5  ;从内存读取e
78                  STR R1, R6, #7   ;存到R6+#7上
79                  LDR R1, R6, #-4  ;从内存读取f
80                  STR R1, R6, #8   ;存到R6+#8上
81                  JSR func         ;递归调用以求x
82                  LDR R0, R6, #0   ;读取递归调用求x得到的返回值
83                  STR R0, R6, #-2  ;存在当前栈的x空间位置上
84                  LDR R1, R6, #-10     ;从内存读取n
85                  ADD R2, R1, #-2  ;求n-2
86                  STR R2, R6, #2   ;存到R6+#2上
87                  ;其余R1到R6上一次均已经存好，且未改变
88                  JSR func         ;递归调用以求y
89                  LDR R0, R6, #0   ;读取递归调用求y得到的返回值
90                  STR R0, R6, #-1  ;存在当前栈的y空间位置上
91
92                  ADD R6, R6, #-12     ;栈指针指回当前调用函数对应栈
93
94                  LDR R0, R6, #9   ;读取t到R0
95                  LDR R1, R6, #10  ;读取x到R1
96                  ADD R0, R0, R1   ;求t+x
97                  LDR R1, R6, #11  ;读取y到R1
98                  ADD R0, R0, R1   ;求t+x+y
99                  ADD R0, R0, #-1  ;求t+x+y-1
100                 STR R0, R6, #0   ;返回值存在R6+#0
101
102                 LDR R7, R6, #1   ;读取R7所保存的返回地址
103                 RET              ;以上完成了n>1的情况
104                                  ;以下是n<=1的情况
105 ELSE            LDR R0, R6, #9   ;返回t,把t的值读入R0
106                 STR R0, R6, #0   ;返回值存在R6+#0
107                 LDR R7, R6, #1   ;读取R7所保存的返回地址
108                 RET          ;n<=1，直接返回
109                 stackoverflow    LEA R0, sof_message
110                 PUTS
111                 HALT
112
113
114 sof_message     .STRINGZ "Stack OverFlow!"
115 c_n30           .FILL    x-30        ;'0'的ASCII的相反数
116 ;stack_size     .FILL    #36
117 stack_size      .FILL    x1C2C       ;栈的大小+#12, x1C2C+#12=#7212
118 func_stack      .FILL    xD000       ;为func函数的栈保留空间
119                                       ;func函数栈空间存储按照下述格式:
120                                       ;1. 每12个内存地址为一块
121                                       ;2. 每块顺序存放:
122                                       ;    x0   返回值
123                                       ;    x1   RET应返回到的R7
124                                       ;    x2   n
125                                       ;    x3   a
126                                       ;    x4   b
127                                       ;    x5   c
128                                       ;    x6   d
129                                       ;    x7   e
130                                       ;    x8   f
131                                       ;    x9   t
132                                       ;    xA   x
133                                       ;    xB   y
134
```

```
135                                   .END
```