LB4 实验报告

Purpose

这个程序通过使用汇编语言中的递归，实现了输出中国传统益智玩具孔明锁解决步骤的功能。其中，递归功能的使用很好的考查了老师课上讲的有关于栈以及子程序功能的使用。值得注意的是，此处子程序的递归调用需要时刻对 PC 值进行维护，这里使用栈来维护。

Principles

把孔明锁的解决翻译成数学递推关系式可以得到：

R(0) = nothing to do

R(1)=remove the 1st ring

R(i)=R(i-2)+remove the i th ring+P(i-2)+R(i-1) ,i>=2

其中，对于 P，我们也有类似的递推关系式

P(0) = nothing to do

P(1) = put the 1st ring on the board

P(i) = P(i-1)+R(i-2)+put the i th ring onboard+P(i-2) ,i>=2

于是我们可以知道，递归函数的终止条件是 i=0 或 1（注意 i=1 的时候记得顺带进行改变孔明锁的状态而 i=0 的时候我们将不会进行任何操作）

其他时候我们只需要关注 PC 值的维护以及递归函数的实现即可;

Procedures

1. 关于 PC 值的维护：
   我们在每一次进入函数的递归调用的时候，都必须进行一次 PC 值的维护
   在一切开始之前有：
   LD R6, STACKLOC ;将栈的基地址 X4000 存入 R6
   在每一次调用函数的时候我们有：

```
STR R1, R6, #0     ;R1->MEM[R6]， R1 = i
ADD R6, R6, #1    ; R6++,栈生长
STR R7, R6, #0    ;R7 - >MEM[R6];
ADD R6, R6, #1    ; 栈再次生长     //建立递归栈
```

把 R7 暂时保存的 PC 值存入栈中，把这次递归时的 i 存入栈中

在每一次我们的函数遇到终止条件的时候，我们在进行子程序的终止之前也进行一次维护：把这个维护包装成一个小函数 FINISH

```
FINISH    ADD R6, R6, #-1    ;//辅助出栈函数
          LDR R7, R6, #0
          ADD R6, R6, #-1
          LDR R1, R6, #0
          RET
```

将栈中最顶上的，应该是 PC 值的数存入 R7，

将栈中次顶的，应该是上一次调用函数时的 i 值存入 R1；

这样我们每一次调用函数时都不在会出现 i 值被不小心篡改或者 R7 中存值不正确的情形了。

2. 关于函数终止条件：

我们的 i 存在 R1 中，只需要：

```
ADD R0, R1, #0       ; //R0 = i
BRz FINISH    ;NOTHING TO DO; //判断 i 是否为零,IF (i ==0)直接完成这次递归，出栈；
```

以及：

```
ADD R0, R1, #-1    ;R0 = i-1
  BRnp    REMOVE1    ;IF R1!=1 GOTO REMOVE1 //判断 i 是否为 1，是 1 的话改变状
```

态然后可以出栈；不是 1 的话进入下一次递归

3. 关于孔明锁状态的维护：

```
ADD R5, R0, #0     ;R5 用来辅助输出计数
```

一开始给 R5 以 x3100 的值，在每一次将要改变孔明锁状态的时候，我们都会让 R5++，以保证一步一步有序且地址正确的输出

我们将孔明锁的状态暂时存在 R2 中，一开始先归为 X0000;

在每一次需要改变孔明锁的状态时：

i==1 时：直接对 R2+-1 即可；

i 是其他值的时候，这里用了一个特别巧妙的手法：

考虑到 n 的取值有限且不是特别大（不大于 12），我们可以预先存入一个数组，每一个数值都是 16bit 位上的一位，例如 1，2，4，8，16 等，在每一次改变孔明锁状态时候，我们只需要根据 i 的值，来挑选一个对应数组中的值加在 R2 上面即可。

Result

将程序在助教给出的测评网站上测评时，都给出了通过的提示：