九连环问题

郑悟强 PB22051082

2023.12.9

1 实验目的

具体而言，我们有从 1 到 n 编号的 n 个环。一个操作包括选择一个环，然后将其放到板上或从板上取下。可以放置或取下的环需要满足以下条件之一：
(1) 它是第 1 个环。
(2) 它是第 i 个环，并且第 (i-1) 个环在板上，但第 (i-2) 个环（即编号为 1 到 i-2 的环，如果存在的话）不在板上。
令 R(i) 是从板上取下前 i 个环的过程，P(i) 是将前 i 个环放到板上的过程。不难发现解决问题的过程是递归的，而且 R(i) 和 P(i) 是彼此的逆过程。
例如，编码您的操作：我们使用 n 位二进制表示 n 个环的状态，最低（最右）位表示第 1 个环。值为 0 的位表示这些环放在板上。否则，这些环已从板上取下。将超过 n 位的部分设置为 0。

你的任务：
n 的值将手动设置在 x3100 中，n 是一个不超过 12 的正整数。
你应该将第 1 个操作后的所有环的状态存储在 x3101，将第 2 个操作后的状态存储在 x3102，依此类推，直到完成所有操作。

2 程序设计

2.1 总体思路

step1. 实现 R(n) 函数和 P(n) 函数 (需要递归调用实现)。
step2. 主函数为读取 x3100 的 n 后，调用 R(n) 解决这个问题。

2.2 REMOVE 函数和 PUT 函数的实现

2.2.1 递推关系式

由助教给出的关系式，R(n) 满足

$$R(n)=\begin{cases} nothing \quad to \quad do & ,n=0 \\ remove \quad the \quad 1st \quad ring & ,n=1 \\ R(n-2)+remove \quad the \quad nth \quad ring+P(n-2)+R(n-1) & ,n\geq 2 \end{cases}$$

再推导得到 P(n) 的递推关系式为

$$P(n)=\begin{cases} nothing \quad to \quad do & ,n=0 \\ put \quad the \quad 1st \quad ring & ,n=1 \\ P(n-1)+R(n-2)+put \quad the \quad nth \quad ring+P(n-2) & ,n\geq 2 \end{cases}$$

2.2.2 递归中栈的使用

由于递归过程中，需要操作的只有 put/remove ring，所以存入栈中的只有每次递归调用时 R7 的地址 (保证 RET 回到正确的对应位置)，而对于 R(n) 和 P(n) 函数中输入输出值的 R1 寄存器，只需要保证结束函数操作后将 R1 重新变为输入的 n 即可，未发生改变就不用存入栈中。

2.2.3 流程图

以 REMOVE 函数为例，输入为 R1，R0 为当前状态，R3 为当前对应的存储地址。

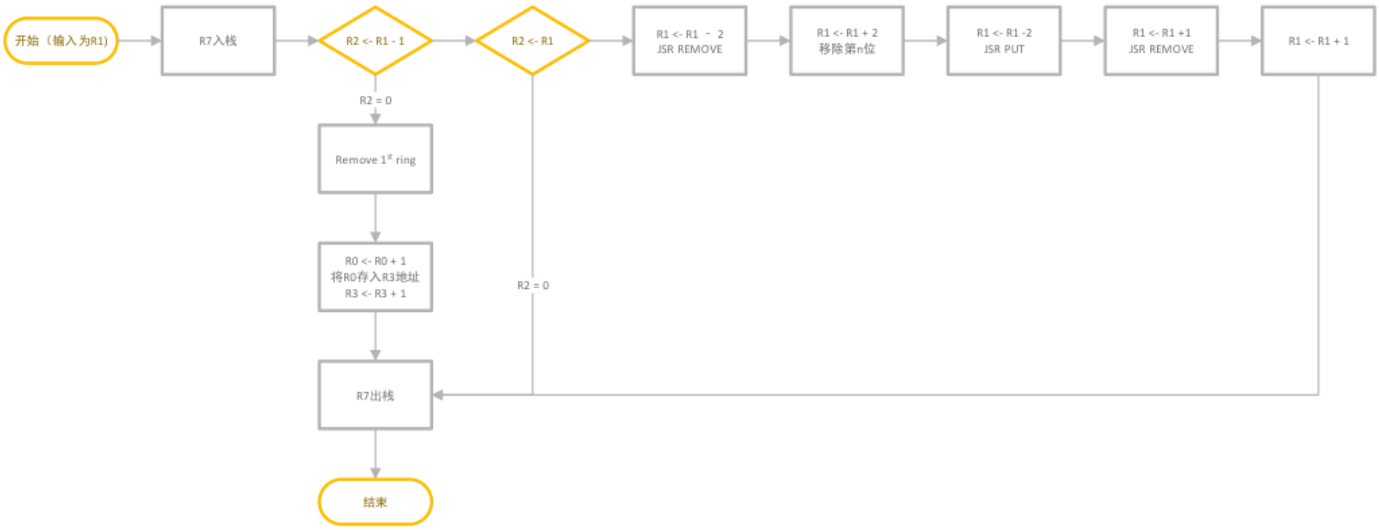

图 1: REMOVE 函数流程图

PUT 函数同理，只是改变顺序，这里不具体阐述了。

2.3 主函数

整体过程为：

1. 读入 x3100 的 n，存入 R1。
2. 将 R0(当前环整体的状态) 置为 0。
3. 将 R3 置为 x3101(存储状态的第一个地址)。
4. 调用 REMOVE 开始执行。
5.HALT x25 结束主函数。

3 过程中遇到的错误

3.1 递归调用中栈的使用

由于在汇编语言中，函数中的参数和各种有关变量都是在寄存器中存储，不能像高级语言一样直接简单的重复调用以实现递归操作，这样会覆盖掉一些重要信息，导致函数错误。

而本次实验中，需要考虑什么寄存器信息会被覆盖，从而需要栈来动态存储。首先 R7 存储函数地址会被覆盖，需要存入栈中。其次，本实验中函数 REMOVE 和 PUT 的传入参数都是 R1，但我们不需要对 R1 和有关寄存器进行计算，函数也无返回值，这样我们完全可以将 R1 在函数操作之后手动加减变回原来的值，这样就可以不用传入栈中。

综上，我的代码中，递归调用时只需入栈出栈 R7 地址即可。

3.2 PUT 函数递推式

当 n=1 或 n=0 时，PUT 函数很容易得到。当 n≥2 时，我们就需要结合实际考虑。

我们要将第 n 个环放上，由已知条件可得，就需要将第 n-1 个环放上，1 n-2 个环放下，所以 PUT 的顺序为：

1. 放下前 n-1 个
2. 放上前 n-2 个
3. 放上第 n 个
4. 放上前 n-2 个

写成表达式即为

$$P(n-1)+R(n-2)+put\quad the\quad nth\quad ring+P(n-2)$$

4 调试结果

使用 lc3web 评测结果：

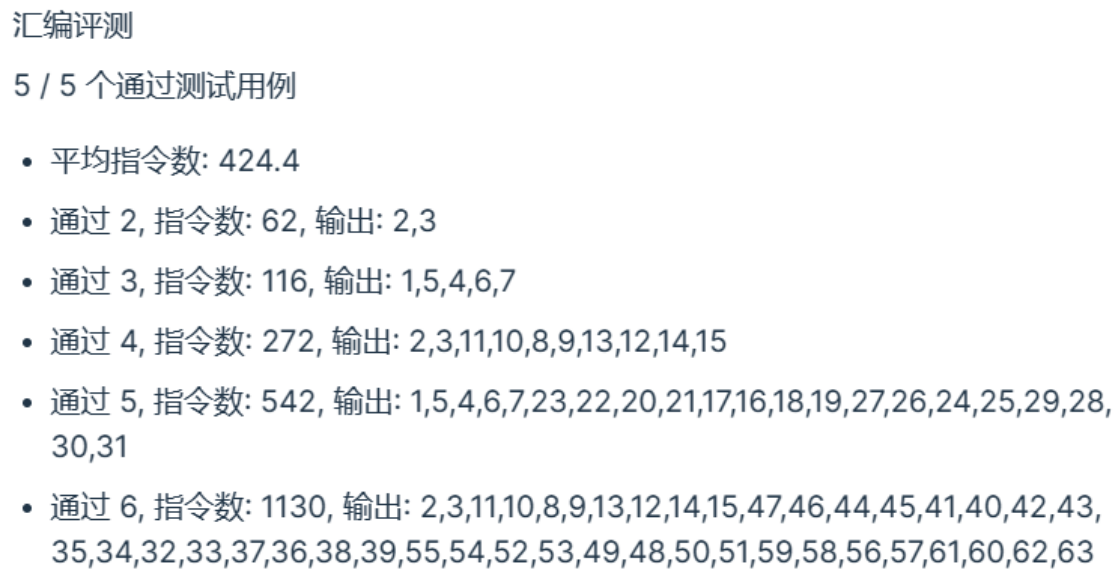

图 1: 评测结果

5 代码效率

由代码执行指令可以推断时间复杂度为 O(2^n)，再通过数学归纳法可以证明代码的时间复杂度确实如此。

具体证明：从右往左数，第 n 个位置的环会被放下 1 次，第 n-1 个位置的环会放下 2 次放上 1 次，第 n-2 个位置的环会被放下 3 次放上 2 次，以此类推，第 k 个位置的环会被放下放上共 $2^{n-k}+1$ 次，再通过累加

$$2^0+1+2^1+1+2^2+1+...+2^{n-1}+1=2^n-1+n$$

所以得到时间复杂度为 O(2^n)。

6 实验体会与收获

通过本次实验，我学会了用栈来辅助实现 lc3 汇编语言的递归写法，也明白了栈在递归中的实际作用，熟悉了栈的有关操作。

另一方面，我也熟练了使用递归思想，动态解决一个复杂问题，将一个抽象的复杂的过程，通过递推式，层层递归的方法来解决。