Lab 4 Report

崔士强 PB22151743

December 17, 2023

1 Purpose

1.1 Problem description

The purpose of the program is to compute the number of steps required to unlock Baguenaudier, a series of rings fitted on a board. A ring can be put on or removed if one of the two conditions below is met:

1. It is the 1st ring.
2. It is the i-th ring, and the $(i-1)$-th ring is on the board, but the first $(i-2)$ rings are not on the board.

The state of rings are represented by a n-bit binary number. 1 means the corresponding ring is removed from the board.

1.2 Anticipated outcomes

The state after each operation will be stored in memory locations starting from `x3101`.

2 Principles

2.1 Overview

The problem can be recursively solved. Let operations of removing n rings be denoted as $R(n)$, and the operations of putting on n rings as $P(n)$. Thus we have:

$$P(i) = \begin{cases} Nothing\ to\ do, & n = 0 \\ Put\ the\ 1^{st}\ ring, & n = 1 \\ P(i-1) + R(i-2) + Put\ the\ i^{th}\ ring + P(i-2), & n \geq 2 \end{cases} \tag{1}$$

$$R(i) = \begin{cases} Nothing\ to\ do, & n = 0 \\ Remove\ the\ 1^{st}\ ring, & n = 1 \\ R(i-2) + Remove\ the\ i^{th}\ ring + P(i-2) + R(i-1), & n \geq 2 \end{cases} \tag{2}$$

So the general idea is to use two subroutines to implement $P(n)$ and $R(n)$, and call the subroutines within subroutines.

2.2 Stack

Due to the fact that the subroutines are repeatedly called, the data in relevant registers may be lost(e.g. R7). Solution is to use a stack. When a subroutine is called, data in registers are pushed onto the stack first. Before returning, the data is poped back. In this program, R6 is the stack pointer which is initialized with `x4000`.

Example of operations on the stack:

```
            ADD    R6, R6, #-1
            STR    R0, R6, #0    ; Push R0 onto stack
            LDR    R0, R6, #0    ; Pop R0 back
            ADD    R6, R6, #1
```

2.3 Operations on the i-th ring

Suppose we have R5 stored a 16-bit binary number with only the i-th bit set as 1, performing bit-wise OR with the number representing current state gives the result after removing the i-th ring.

According to DeMorgan Rule:

$$A\ OR\ B = \overline{\overline{A}\ AND\ \overline{B}}$$

OR can be implemented by `AND` and `NOT` instructions:

```
            NOT    R1, R1
            NOT    R5, R5
            AND    R1, R1, R5
            NOT    R1, R1
            ADD    R3, R3, #1
            STR    R1, R3, #0    ; Remove the i-th ring
```

And performing bit-wise NAND will give the result after putting the i-th ring:

```
            NOT    R5, R5
            AND    R1, R1, R5
            ADD    R3, R3, #1
            STR    R1, R3, #0    ; Put the i-th ring
```

In order to get the right number in R5, we use a subroutine to repeatedly move the number 1 left:

```
SINGLEBIT   ADD    R6, R6, #-1
            STR    R0, R6, #0    ; Push R0 onto stack

            ADD    R0, R0, #-1
            AND     R5, R5, #0    ; Clear R5
            ADD     R5, R5, #1    ; Set R5 as 1

MOVE_LEFT   ADD     R5, R5, R5    ; Multiply R5 by 2
            ADD     R0, R0, #-1
            BRp     MOVE_LEFT

            LDR     R0, R6, #0    ; Pop R0 back
            ADD     R6, R6, #1
  RET
```

3 Procedure

3.1 Bugs encountered

1. When testing the program, an endless loop is detected. The reason is that in REMOVE subroutine, I used AND to remove the i-th bit, while the right way is to use OR.
2. Another bug comes from misusing of the stack. When pushing a new item onto the stack, stack pointer substracts 1, which is `ADD R6, R6, #-1`. But I used `ADD R6, R6, #1` which caused a bug.

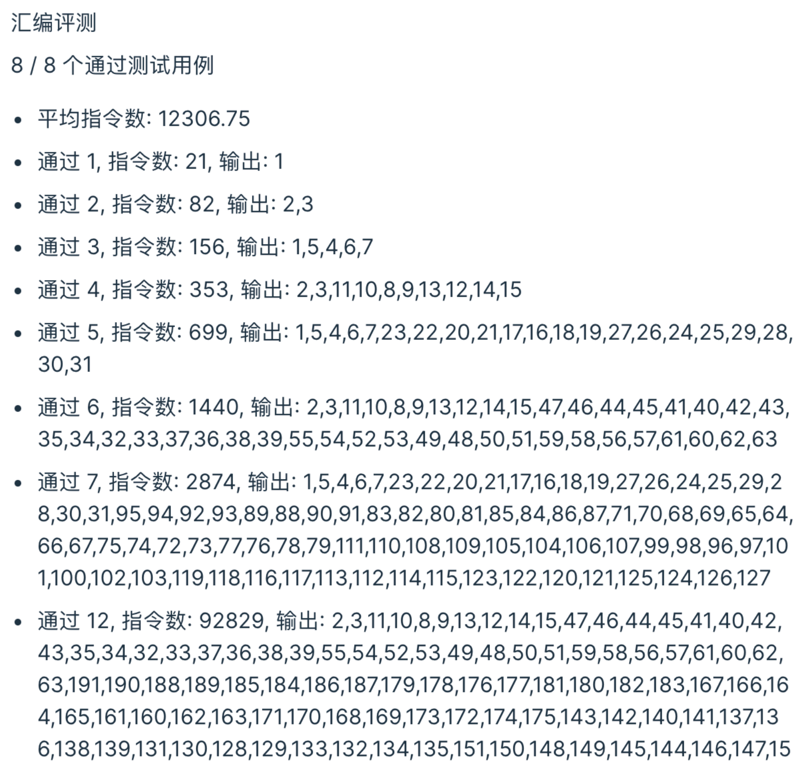
3.2 Chanllenges

A major chanllenge in the task is the use of subroutines and stack. To get things work properly, we need to find out which registers are expected to remain unchanged after calling the subroutine and which are not.

In this case, R0(stores number of rings to be moved) and R7(stores return linkage) are expected to remain unchanged. So we push R6 and R0 at the beginning of the subroutine, and pop into R0 and R6 before returning.

4 Results

Results are shown below:



汇编评测

8 / 8 个通过测试用例

- 平均指令数: 12306.75
- 通过 1, 指令数: 21, 输出: 1
- 通过 2, 指令数: 82, 输出: 2,3
- 通过 3, 指令数: 156, 输出: 1,5,4,6,7
- 通过 4, 指令数: 353, 输出: 2,3,11,10,8,9,13,12,14,15
- 通过 5, 指令数: 699, 输出: 1,5,4,6,7,23,22,20,21,17,16,18,19,27,26,24,25,29,28,30,31
- 通过 6, 指令数: 1440, 输出: 2,3,11,10,8,9,13,12,14,15,47,46,44,45,41,40,42,43,35,34,32,33,37,36,38,39,55,54,52,53,49,48,50,51,59,58,56,57,61,60,62,63
- 通过 7, 指令数: 2874, 输出: 1,5,4,6,7,23,22,20,21,17,16,18,19,27,26,24,25,29,28,30,31,95,94,92,93,89,88,90,91,83,82,80,81,85,84,86,87,71,70,68,69,65,64,66,67,75,74,72,73,77,76,78,79,111,110,108,109,105,104,106,107,99,98,96,97,101,100,102,103,119,118,116,117,113,112,114,115,123,122,120,121,125,124,126,127
- 通过 12, 指令数: 92829, 输出: 2,3,11,10,8,9,13,12,14,15,47,46,44,45,41,40,42,43,35,34,32,33,37,36,38,39,55,54,52,53,49,48,50,51,59,58,56,57,61,60,62,63,191,190,188,189,185,184,186,187,179,178,176,177,181,180,182,183,167,166,164,165,161,160,162,163,171,170,168,169,173,172,174,175,143,142,140,141,137,136,138,139,131,130,128,129,133,132,134,135,151,150,148,149,145,144,146,147,15

Figure 1: Result