Lab 4 Report

--**PB22111599 杨映川**

Purpose

Visualize the process of solving the **Baguenaudier puzzle** with recursion algorithm of the lc3 assembly language.

Principles

1. In order to pretend from losing the corresponding return addresses amongst the recursion operations, a **stack** starting from x4000 is used to load the addresses. R5 is the stack pointer.

```
BAGUE_R
; ***PUSH THE RETURN ADDRESS***
    ADD R5, R5, #1
    STR R7, R5, #0
; ***R(i - 2)***
    ADD R4, R0, #-2
    BRnz SKIPI      ; if(n>2) B_R(n-2)
    ADD R0, R0, #-2
    JSR BAGUE_R
    ADD R0, R0, #2
SKIPI
; ***r(i)***
    JSR REMOVE_ONE
; ***P(i - 2)***
    ADD R4, R0, #-2
    BRnz SKIPII
    ADD R0, R0, #-2
    JSR BAGUE_P
    ADD R0, R0, #2
SKIPII
; ***R(i - 1)***
    ADD R4, R0, #-1
    BRnz SKIPIII
    ADD R0, R0, #-1
    JSR BAGUE_R
    ADD R0, R0, #1
SKIPIII
; ***POP THE RETURN ADDRESS***
    LDR R7, R5, #0
    ADD R5, R5, #-1
    RET
```

2. In order to update the state of the baguenaudier, first locate the bit at the correct digit. If the operation is to remove the ring, then ADD the bit code to the current state code; if the operation is to put the ring, then NOT the bit code then AND it to the current state code.

3. In order to maintain the value of the number of the the rings, at the both sides of the recursion instruction, which is the JSR instruction, subtract 1 or 2, which depends on the recursion function given inside the lab file, then add it back.

```
19  ; ***R(i - 2)***
20      ADD R4, R0, #-2
21      BRnz SKIPI          ; if(n>2) B_R(n-2)
22      ADD R0, R0, #-2
23      JSR BAGUE_R
24      ADD R0, R0, #2
25  SKIPI
```

4. According to the lab file, P(i) is the inverse of R(i). We can figure out that:

> 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 - 2), i >= 2;
>
> ---
>
> P(0) = nothing to do; P(1) = put the 1st ring; **P(i) = P(i - 1) + R(i - 2) + put the i-th ring + P(i - 2)**, i >= 2;

Procedure

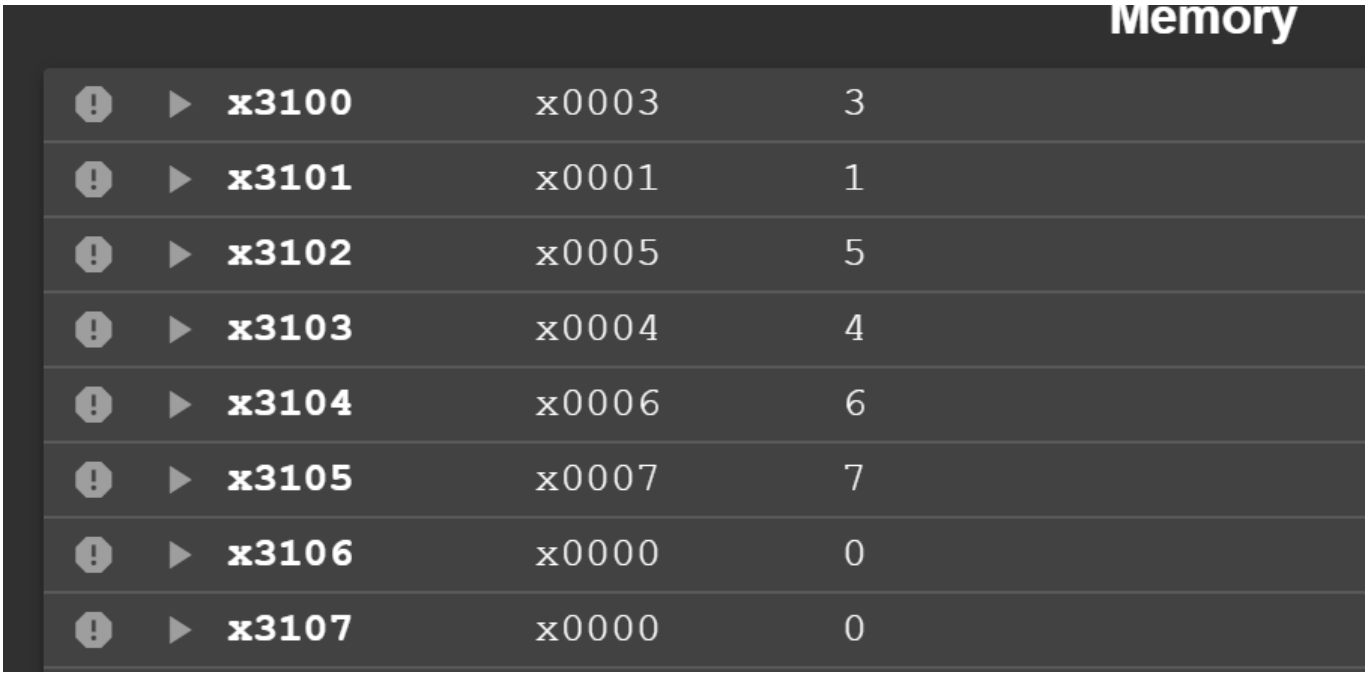
1. Figure out the principle of how the recursion works in lc3 assembly language. Every time a JSR or a JSRR instruction is executed, the return address, i.e the PC incremented, is loaded into R7. The RET instruction at the end of the a sub-function will steer the program back to the correct address.
2. Encode a C language version to help figure out how the program goes.

Result

example 1

- n = 3

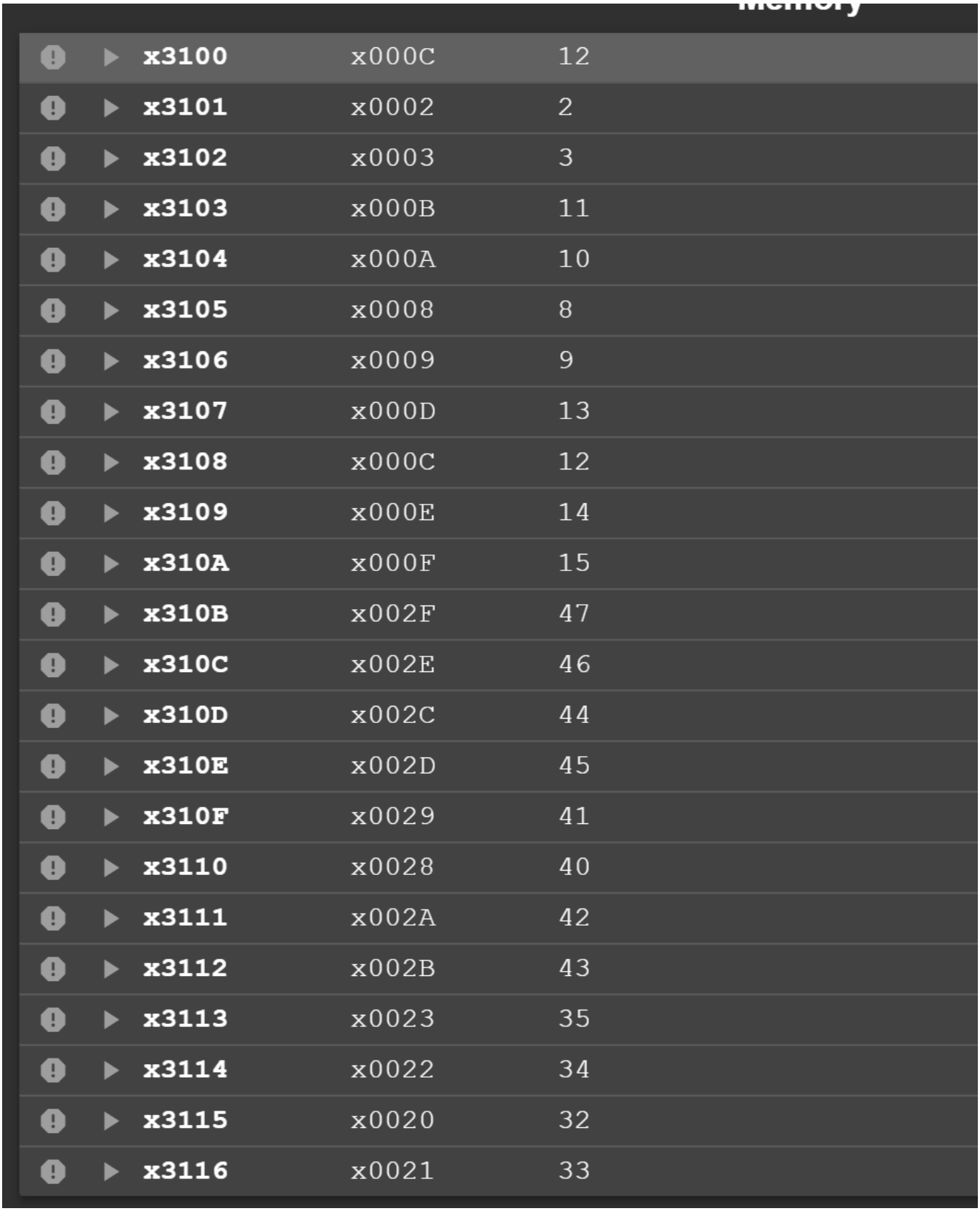
output



example 2

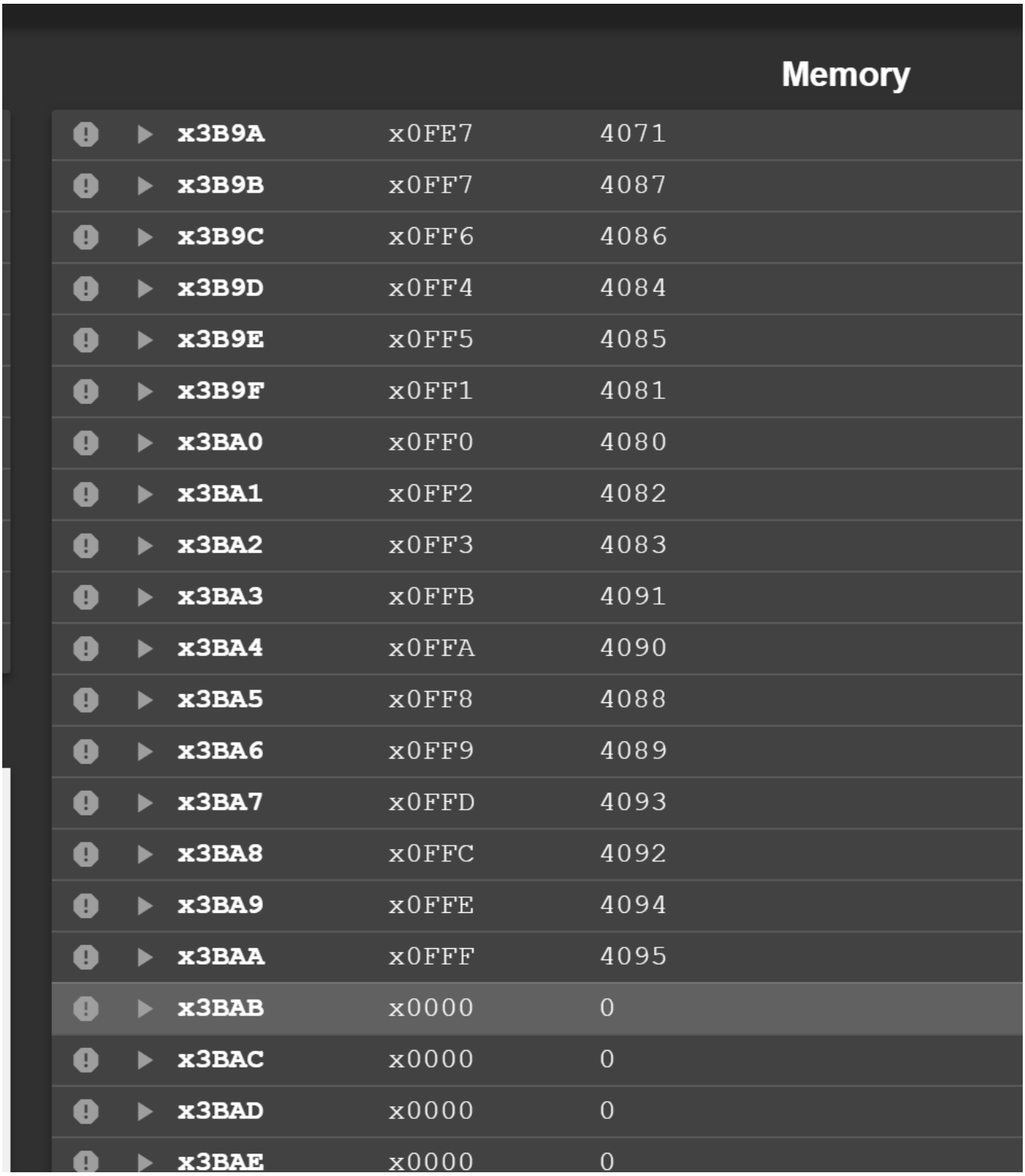
- n = 12

output



x3100	x000C	12
x3101	x0002	2
x3102	x0003	3
x3103	x000B	11
x3104	x000A	10
x3105	x0008	8
x3106	x0009	9
x3107	x000D	13
x3108	x000C	12
x3109	x000E	14
x310A	x000F	15
x310B	x002F	47
x310C	x002E	46
x310D	x002C	44
x310E	x002D	45
x310F	x0029	41
x3110	x0028	40
x3111	x002A	42
x3112	x002B	43
x3113	x0023	35
x3114	x0022	34
x3115	x0020	32
x3116	x0021	33

- over 2000 items **omitted**


Memory
x3B9A x0FE7 4071
x3B9B x0FF7 4087
x3B9C x0FF6 4086
x3B9D x0FF4 4084
x3B9E x0FF5 4085
x3B9F x0FF1 4081
x3BA0 x0FF0 4080
x3BA1 x0FF2 4082
x3BA2 x0FF3 4083
x3BA3 x0FFB 4091
x3BA4 x0FFA 4090
x3BA5 x0FF8 4088
x3BA6 x0FF9 4089
x3BA7 x0FFD 4093
x3BA8 x0FFC 4092
x3BA9 x0FFE 4094
x3BAA x0FFF 4095
x3BAB x0000 0
x3BAC x0000 0
x3BAD x0000 0
x3BAE x0000 0