

# ICS\_Lab2\_Report

Xiaoma

2022 年 11 月 29 日

## 实验目的

使用 LC-3 汇编命令实现

$$F(0) = F(1) = 1$$

$$F(N) = F(N-2)\%p + F(N-1)\%q \quad (2 \leq N \leq 1024)$$

$$p = 2^k \quad (2 \leq k \leq 10), 10 \leq q \leq 1024$$

- p 存储在内存位置 **x3100**
- q 存储在内存位置 **x3101**
- N 存储在内存位置 **x3102**
- F(N) 存储在内存位置 **x3103**

## 实验原理

对于求斐波那契数列的变式问题，通常情况下采用滑动窗口的方法。

**注意：**本次实验计算公式与一般的斐波那契数列不同， $F(N-1), F(N-2)$

只有在求和的时候取余，而存储时不需要取余。

按照该思想可以得到伪代码：

---

**Algorithm 1:** myFib

---

**Input:**  $p, q, N$ ;**Output:**  $F(N)$  ;

num1 = 1;

num2 = 1;

 $N = N - 1$ ;**while**  $N > 0$  **do**

temp1 = num1;

temp2 = num2;

**while**  $temp1 \geq 0$  **do**

temp1 = temp1 - p;

temp1 = temp1 + p;

**while**  $temp2 \geq 0$  **do**

temp2 = temp2 - q;

temp2 = temp2 + q;

 $f = temp1 + temp2$ ;

num1 = num2;

num2 = f;

 $N = N - 1$ ;**return**  $f$ ;

## 实验步骤

### LC-3 指令集的限制

若要实现上述伪代码，除了需要已知的 8 个变量，还需要 2 个变量来存储  $p, q$  的相反数，而 LC-3 只有 8 个寄存器，所以需要对变量的数量进行压缩。

- $F(N-2)$  在完成本次计算以后将被移出窗口，即原存储  $F(N-2)$  的寄存器将存储  $F(N-1)$ 。
- $F(N)$  为两数取余之和

因此可以采用

- $F(N-2)$  与  $F(N-2)\%p$  共用一个变量
- 存储  $F(N)$  的变量首先存储  $F(N-1)\%q$
- $F(N) = F(N) + F(N-2)\%p$

将所需寄存器的数量压缩至 8 个。

### 取余操作

若需要用 LC-3 汇编命令实现取余操作，可将被除数不断减去除数，直至结果为负数，此时该负数与被除数相加，得到的结果即为余数。

### 减法操作

对于取余时需要的减法操作，采用与计算二进制数的相反数相同的方法，将原二进制数取反再加 1。

### 计算斐波那契数列

已知对于一个普通的斐波那契数列  $F(N) = F(N-2) + F(N-1)$ ，假设有一个长度为 3 的窗口，每次窗口在数列上右移一位，直至得到结果，即相当于每完成一次计算，后一个窗口存储前一个窗口的结果。

## 代码讲解

### 初始化变量

从内存中读取  $p, q, N$  3 个变量，即

$$R0 \leftarrow p, R1 \leftarrow q, R2 \leftarrow N$$

1	LD R0, x0FF
2	LD R1, x0FF
3	LD R2, x0FF

### 减法预处理

已知取余操作要进行减法运算，故首先得到  $p, q$  的相反数，即

$$R3 \leftarrow -p, R4 \leftarrow -q$$

4	NOT R3, R0
5	ADD R3, R3, x1
6	NOT R4, R1
7	ADD R4, R4, x1

### 取余操作

分别求  $F(N-2), F(N-1)$  的余数，使用的变量考虑了 LC-3 寄存器的数量，即

$$R5 \leftarrow F(N-2)\%p, R7 \leftarrow F(N-1)\%q$$

8	ADD R5, R5, R3
9	BRzp <del>#-2</del>
10	ADD R5, R5, R0
11	ADD R7, R6, <del>#0</del>
12	ADD R7, R7, R4
13	BRzp <del>#-2</del>
14	ADD R7, R7, R1

### 求和

求和得到  $F(N)$ ，即

$$R7 \leftarrow F(N-2)\% + F(N-1)\%q$$

15	ADD R7, R7, R5
----	----------------

### 滑动窗口

得到计算结果后滑动窗口，准备下一个计算，即

$$R5 \leftarrow R6, R6 \leftarrow R7$$

16	ADD R5, R6, #0
17	ADD R6, R7, #0

### 存储结果

在迭代结束后，将计算结果存储至 x3103，即

$$R7 \rightarrow x3103$$

18	ST R7, x0EC
----	-------------

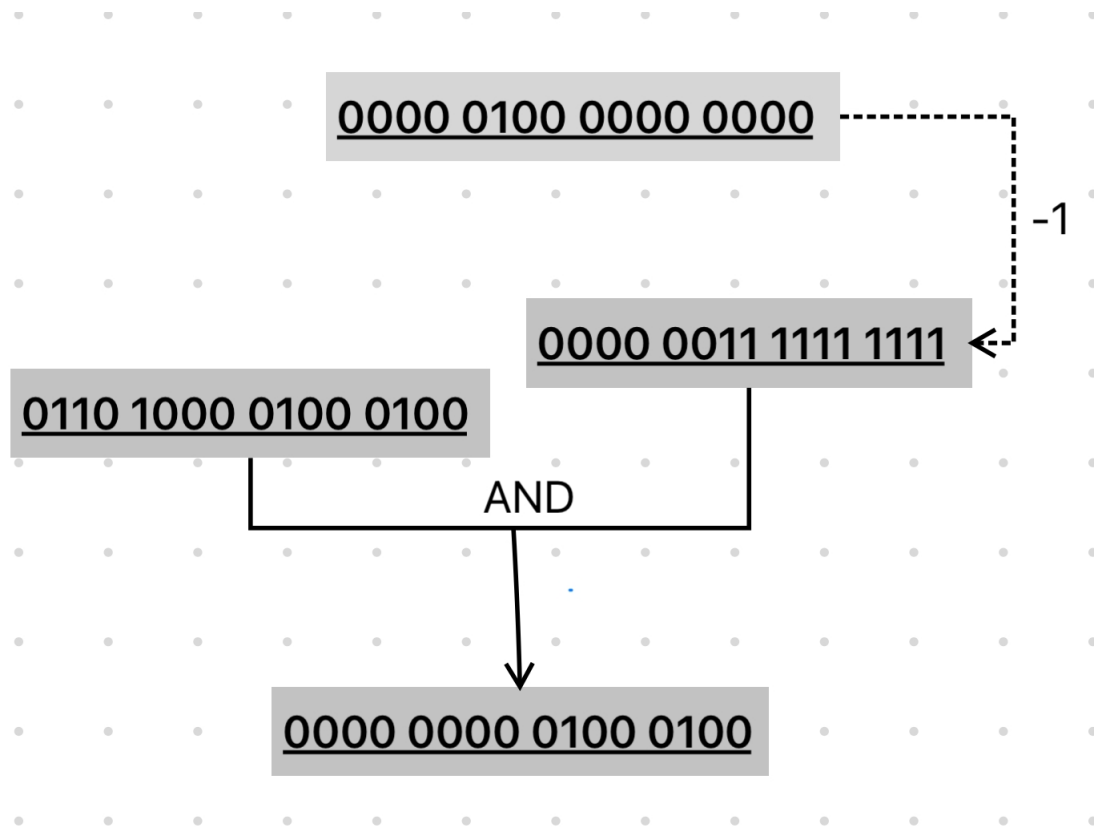
## 如何提高循环效率

观察  $p$  可以发现， $p$  为 2 的整数次方，若用 16 位二进制数表示则只有一位为 1，那么可以采用更简单的操作对  $F(N-2)$  进行取余。

### 更高效的取余操作

若将  $p$  减去 1 以后将结果和  $F(N-2)$  进行与操作，则  $F(N-2)$  只有低位为 1 的部分被保留，即为  $p$  的余数。

```
1  ADD R3, R0, #-1
2  AND R5, R5, R3
```



## 实验结果

### 未改进取余操作

依次对实验文档给出的例子进行测试，结果如下：

#### 汇编评测

3 / 3 个通过测试用例

- 平均指令数: 2983
- 通过 256:123:100, 指令数: 1523, 输出: 146
- 通过 512:456:200, 指令数: 2941, 输出: 818
- 通过 1024:789:300, 指令数: 4485, 输出: 1219

自行编写了部分测试例子，结果如下：

#### 汇编评测

5 / 5 个通过测试用例

- 平均指令数: 50375.4
- 通过 2:10:200, 指令数: 3537, 输出: 4
- 通过 1024:1024:500, 指令数: 7449, 输出: 450
- 通过 128:50:500, 指令数: 7935, 输出: 134
- 通过 64:100:800, 指令数: 12123, 输出: 66
- 通过 2:500:1000, 指令数: 220833, 输出: 167

## 改进取余操作

依次对实验文档给出的例子进行测试，结果如下：

#### 汇编评测

3 / 3 个通过测试用例

- 平均指令数: 2248.6666666666665
- 通过 256:123:100, 指令数: 1182, 输出: 146
- 通过 512:456:200, 指令数: 2196, 输出: 818
- 通过 1024:789:300, 指令数: 3368, 输出: 1219

自行编写了部分测试例子，结果如下：

汇编评测

5 / 5 个通过测试用例

- 平均指令数: 6442.4
- 通过 2:10:200, 指令数: 2026, 输出: 4
- 通过 1024:1024:500, 指令数: 5476, 输出: 450
- 通过 128:50:500, 指令数: 6260, 输出: 134
- 通过 64:100:800, 指令数: 8448, 输出: 66
- 通过 2:500:1000, 指令数: 10002, 输出: 167

可以发现改变取余操作大大提升了计算效率。