Lab2 递推数列的计算

实验算法设计

> 在本次实验中，数列的递推公式为：
>
> $$F(n) = (F(n-1) + 2 * F(n-3)) \bmod 1024 \ (1 <= n <= 16384)$$
>
> 由于L程序以**代码行数最短**为宗旨，故结合上述公式可以设计出如下算法：
>
> - 初始化：使用R1存储F(k-3)（初始为1），R2存储F(k-2)（初始为1）、R7存储F(k-1)（初始为2）、R4存储1023（用于取模）；
> - 对于F(n)，我们只需要循环n - 2次即可求得，但需要特判1,2的情况，因此先使用R5保存R0的值，再令R0自减2，若R0中值小于等于0，则进入特判模块（SEC）并结合**该数列第1,2项值与项数相同**的特点，使用R5完成对R7的赋值；
> - 进入主循环，先进行寄存器移位，即使用R4保存2 * R1，再令R1 <- R2， R2 <- R7，最后令R7 = R7 + R4，并令R0自减；
> - 判断R0是否为正值，若为正值则返回第三步，否则说明R0已经递减为0，算法结束。
>
>   最终代码实现为**22**行。

实验代码

```
.ORIG x3000

;initial
ADD R1, R1, #1; R1 = 1 = F(0);                F(n - 3)
ADD R2, R2, #1; R2 = 1 = F(1);                F(n - 2)
ADD R7, R7, #2; R3 = 2 = F(2);                F(n - 1)
LD R4 MOD1024;
ADD R5, R0, #0

ADD R0, R0, #-2;
BRnz SEC; if(n == 1 || n == 2)

LOOP
;Shifting Registers
ADD R3, R1, R1 ; R3 = R1 * 2;
ADD R1, R2, #0; R1 = R2
ADD R2, R7, #0; R2 = R7

;Calculate
ADD R7, R7, R3 ; F(n) = F(n - 1) + 2 * F(n - 3)
ADD R0, R0, #-1; R0 -= 1;
BRp LOOP
BRnz OVER

SEC ADD R7, R5, #0

OVER
AND R7, R7, R4 ; mod(1024)
HALT
```

```
MOD1024 .FILL x03FF
SEG1 .FILL x03A2; F(20)
SEG2 .FILL x00F6; F(11)
SEG3 .FILL x0182; F(16)
SEG4 .FILL x0036; F(23)
.END
```

代码分析

- **正确性分析**

  这里先给出两组测试，首先是n = 7的情况：

Registers

R0	x0007	7	
R1	x0000	0	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

  运行后的结果为：

Registers

R0	x0000	0	
R1	x000A	10	
R2	x0012	18	
R3	x001E	30	
R4	x03FF	1023	
R5	x0007	7	
R6	x0000	0	
R7	x001E	30	
PSR	x8002	-32766	CC: Z
PC	x3011	12305	
MCR	x0000	0	

  下面是n = 16834的情况：

Registers

R0	x4000	16384	
R1	x0000	0	
R2	x0000	0	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3000	12288	
MCR	x0000	0	

运行结果为：

Registers

R0	x0000	0	
R1	x0232	562	
R2	x03F6	1014	
R3	x0202	514	
R4	x03FF	1023	
R5	x4000	16384	
R6	x0000	0	
R7	x0202	514	
PSR	x8002	-32766	CC: Z
PC	x3011	12305	
MCR	x0000	0	

> 事实上，运用这种寄存器移位的方法，可以很直观的记录连续三个递推数的信息，从而确保了结果的正确性。当n ≤ 2的时候，我们只需进行特判，再结合**该数列第1,2项值与项数相同**的特点，就可以正确输出地推数列初始值。
>
> 我也使用了十分直观且容易从逻辑上验证正确性的递归编程进行了结果比对，但递归开销太大，只能验证较小数的正确性，其代码如下：

```
#include <iostream>
int fun(int n)
{
    if(n == 0 || n == 1) return 1;
    else if(n == 2) return 2;
    return fun(n - 1) + 2 * fun(n - 3);
}
int main()
{
    int n;
    std::cin >> n;
    int result = fun(n) % 1024;
    std::cout << result;
    return 0;
```

```
}
```

- **效率分析**

> 该算法依然是利用循环对最终结果进行递推，因此运行时间基本与n的大小成正比。当n达到10000左右时，程序已经会出现明显的卡顿。
>
> 事实上，如果能够计算出该数列的通项公式，那么必然可以在O(1)的时间内获得解，但是参照斐波那契数列的通项公式来看，这个数列的通项公式可能极为复杂，如果含有平方根等非整数数值，那么LC3可能比较难以计算出精确的解

- **代码开发过程**

> 最早版本的代码十分冗余，我对n是否为1,2进行了过多的判断，导致代码长达30行。之后我通过对判断进行精简，并利用了最简单的**do……while**循环，一次性解决了对1和2的判断。
>
> 早期版本的代码使用R1，R2，R3这三个寄存器来保存F(n - 3)，F(n - 2)，F(n - 1)，之后运算后对它们进行移位，这实际上是冗余的，因为本身R7中值在上一个循环中为F（n），这个循环中自然变为F（n - 1），因此可以直接使用。
>
> 除此以外，之前的代码为在每次循环中都取模，但这完全没有必要，因为一旦溢出，实际上对无符号整数来说是对65536取模，因此只需要最后再对1024取模即可。

实验总结

在本次实验中，我对LC3的汇编语言应用有了充分的理解，同时，通过实现这个数列递推，我也对算法在LC3的低层汇编语言的实现方法有了基本的理解。本次的L程序依然存在很多可以优化之处，希望下次实验中可以进行改进。