LAB2：Lending Your Name

2021.12 赖永凡 PB20061210

实验内容

定义一个数列 F(n) = F(n-1) + 2 * F(n-3) (1 <= n <= 16384)，其中F(0) = 1, F(1) = 1, F(2) = 2。本实验希望设计一个程序计算此数列，R0寄存器保存n的值，计算出来的F(n)保存在R7中。

L版本的代码旨在用尽可能少的代码行数来实现本实验。

L版本

测试样例

本人学号为PB20061210，测试的F(20), F(06), F(12), F(10)已在代码中给出

设计思路

本次实验我一共完成了三个版本的代码，其中第二代是在第一代版本的基础上进行了少量优化，所以放在一起反而更加直观，叙述起来也更加简练。

VERSION 1&2

```
;verl(v.2)
;This program calculates a Fibonacci-like sequence.
;F(n) = (F(n-1) + 2 * F(n-3)) mod 1024 (1 <= n <= 16384)
;F(0) = 1
;F(1) = 1
;F(2) = 2
;
;Input in R0
;Output in R7
        .ORIG x3000
        AND R1,R1,#0
        AND R2,R2,#0
        AND R7,R7,#0    ;Reset register, which can be omitted.

        LD R6,SADD
        LD R4,MOD
        JSR FIB
                        ;In version 1, here is a meaningless line "AND R7,R7,#0"
        ADD R7,R1,#0
        HALT

;The subroutine to calculate the sequence,
;Recieve input in R0, yeild in R1.

FIB     ADD R6,R6,#3    ;ADD R6,R6,#1
        STR R7,R6,#-2   ;STR R7,R6,#0
        STR R0,R6,#-1   ;ADD R6,R6,#1
        STR R2,R6,#0    ;STR R0,R6,#0
```

```
                ;ADD R6,R6,#1
                ;STR R2,R6,#0
                ;This part does the preparation of entering a Subroutine
                ;Code in the comment is version 1

        ADD R3,R0,#-2   ;check for base case
        BRp MAIN

        ADD R1,R0,#0
        BRnp DONE       ;if R0 = 1,2, DONE
        ADD R1,R0,#1    ;if R0 = 0, make it 1
        BRnzp DONE

MAIN    ADD R0,R0,#-1
        JSR FIB
        ADD R2,R1,#0    ;F(n-1) stores in R2
        ADD R0,R0,#-2
        JSR FIB         ;F(n-3) stores in R1
        ADD R1,R1,R1
        ADD R1,R1,R2    ;F(n-1) + 2 * F(n-3)
        AND R1,R1,R4    ;mod 1024

DONE    LDR R2,R6,#0    ;LDR R2,R6,#0
        LDR R0,R6,#-1   ;ADD R6,R6,#-1
        LDR R7,R6,#-2   ;LDR R0,R6,#0
        ADD R6,R6,#-3   ;ADD R6,R6,#-1
                        ;LDR R7,R6,#0
                        ;ADD R6,R6,#-1
                        ;Ends calling, pop the argument of the caller routine
                        ;As above, comment is the code of version 1
        RET

SADD    .FILL x4000     ;base address of stack
MOD     .FILL x03FF     ;0000 0011 1111 1111 = 1024(10)

F20     .FILL #930
F06     .FILL #18
F12     .FILL #418
F10     .FILL #146
        .END
```

第一和第二代版本都是基于函数的递归调用算法，在一定程度上参考了书上计算斐波那契数列的程序，其中最关键的递归栈的运用也是基于标准范式之下，总的来说可谓是乏善可陈。

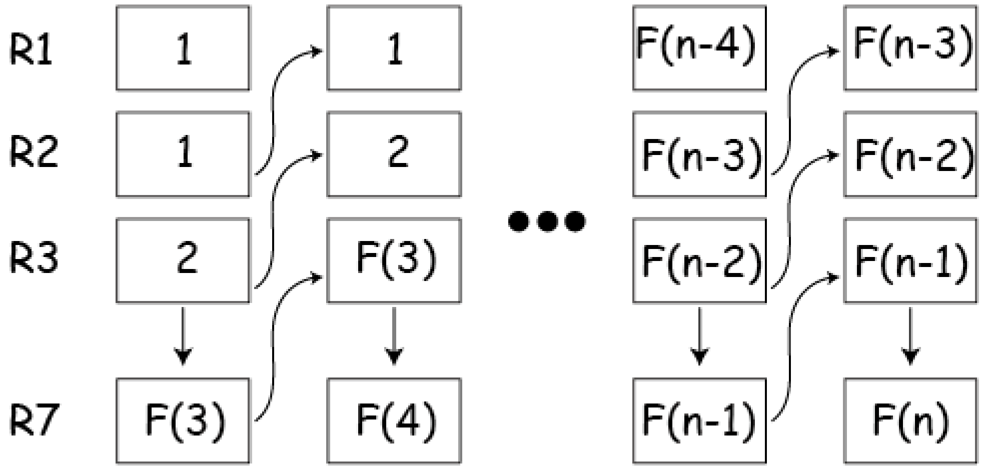
不难发现，尽管递归程序在计算的部分还算简便，但是在调用子程序这一环节上显得十分的繁琐。虽然第二代的代码在出入栈操作上有所优化，还是仅仅能够将代码缩短可怜的4行。（~~不过其实是从38缩短到了34行，可以说是最关键的四行了XD~~）

然而，在测试的过程中，递归版本的代码还是出现了一个**致命的缺陷**——处理不了n值过大的情形。
当n值较大时，首当其冲的就是用来存放递归栈的地址空间。对这个程序而言，调用一次子程序就需要占用三个储存空间，可以估算最大的占用开销大约为3*n数量级，因此如果n过大的话，或许会造成溢出。
此外，当n值在30以上的时候，程序执行起来就**慢得令人发指**。通过分析可以发现，由于缺少动态规划，计算同一F(n)的子程序会被反复调用，最终造成程序执行起来极为缓慢。随着n的增加，执行时间的增加感觉达到了指数级别。
这种天生的缺陷，让我不得不放弃递归这种程序设计方式。

VERSION 3

```
;verl(v.3)
;This program calculates a Fibonacci-like sequence.
;F(n) = (F(n-1) + 2 * F(n-3)) mod 1024 (1 <= n <= 16384)
;F(0) = 1
;F(1) = 1
;F(2) = 2

;Input in R0
;Output in R7
        .ORIG x3000

        AND R1,R1,#0    ;Storing F(n-3)
        AND R2,R2,#0    ;F(n-2)
        AND R3,R3,#0    ;F(n-1)
        AND R4,R4,#0    ;2 * F(n-3)
        AND R5,R5,#0    ;x03FF, the template of moding #1024
        AND R7,R7,#0    ;Output.
                        ;This part clear the registers, which will
                        ;not be presented in the handed-in version.

        ADD R1,R1,#1    ;Set the initial value.
        ADD R2,R2,#1
        ADD R3,R3,#2
        LD  R5,MOD

        ADD R0,R0,#-2   ;Check for base cases.
        BRp LOOP
        BRn BASEI
        ADD R7,R7,#1
BASEI   ADD R7,R7,#1
        BRnzp FIN

LOOP    BRz FIN         ;Main loop, quit if n decays to zero.
        ADD R4,R1,R1
        ADD R7,R4,R3
        AND R7,R7,R5    ;Mod 1024, afterwards R7 is F(n).
        ADD R1,R2,#0
        ADD R2,R3,#0
        ADD R3,R7,#0    ;update the F(n-3), F(n-2) and F(n-1),
                        ;so that we can use it in the next iteration.
        ADD R0,R0,#-1
        BRnzp LOOP

FIN     HALT

MOD     .FILL x03FF

F20     .FILL #930
F06     .FILL #18
F12     .FILL #418
F10     .FILL #146

        .END
```

在第三代程序中，我使用了循环结构的程序，具体的逻辑结构如图所示：



由于在检查完n是否为初始条件的值后，R0所存的值已经减了2，所以循环实际只执行了n-2次。
平心而论，第三代版本的代码相较于前两代更容易理解，注释和逻辑结构图也解释得比较完善了，在此便不再赘述。

循环版本的程序只使用了25行代码，看上去简练得多。针对递归版本无法处理的n过大的情况也得以彻底解决，毕竟执行的时间复杂度仅仅为O(n)。就算是计算当n=16384时，执行时间也仅需要5秒左右。就空间复杂度来说相较于递归版本也大大减小，原因就在于少了递归栈的储存消耗。

至此，也就得到了一个还算令人满意的程序了。此外，经由本次实验，我对书上的这句话也有了更深刻的体会。

> Fibonacci, an Even Worse Example