Lab3 Better Angels

实验内容

- 通过获得代码的最后几行中填充的值，推导出对方的学号。
- 对对方的代码的运行效率进行改进

学号推导

我收到的对方学号的数列值为：

```
FA .FILL x03A2
FB .FILL x0001
FC  .FILL x0002
FD .FILL x0236
```

直接手动推算出对应的n显然是不现实的。由于学号被分为4个部分，每个部分对应十进制数值不超过99，因此我们可以通过C语言程序进行查找，代码如下：

```
#include <iostream>
int main()
{
    int r0 = 99;
    int r1 = 1;
    int r2 = 1;
    int r3 = 2;
    int r4 = 1023;
    int num;
    std::cin >> num;
    int r5 = r0;
    int r7 = 0;
    r0 -= 2;
    if(r0 <= 0) r7 = r5;
    else
    {
        do
        {
            r1 += r1;
            r7 = r1 + r3;
            r7 = r7 & r4;
            if(r7 == num) std:: cout << 100 - r0 << std::endl;
            r1 = r2;
            r2 = r3;
            r3 = r7;
            r0 -= 1;
            if(r0 <= 0) break;
        } while (r0 > 0);
    }
    return 0;
}
```

这里不得不注意的是，F(0)和F(1)的值都是1，但由于我班学号3、4位为01的同学末尾4位并不匹配，因此我推断这两位必为00。

通过对我在lab2中算法的小修改（这里需要在每一步都mod1024），我们就可以找到对方的学号。对方的学号为：

```
20000287
```

程序运行效率改进

原代码

```
.ORIG  x3000
ADD R7 ,R7 ,#1
ADD R6 ,R6 ,#1
LD R1, NUM
ADD R5 ,R5 ,#1

ADD R0 ,R0 ,#-1; check 1
BRnz P

ADD R7 ,R7,#1; check 2
ADD R0 ,R0,#-1
BRz Q

ADD R0,R0,#-1
M
ADD R0 ,R0 ,#-1
ADD R4 ,R5 ,#0; R4 = R5
ADD R2 ,R4 ,R4; R2 += R4

NOT R2,R2
ADD R2,R2 #1; R2 = -R2

ADD R3,R5,R5; R3 = 2 * R5
ADD R7,R7,R3; R7 += R3
ADD R5,R6,#0; R5 = R6
ADD R6,R7,R2; R6 = R7 + R2
N
ADD R7,R7,R1; R7 += R1(mod 1024)
BRzp    N; if(r7 >= 0) back

NOT R1,R1
ADD R1,R1,#1; R1 = 1024

ADD R7,R7,R1; minus, should add to postive
NOT R1,R1
ADD R1,R1,#1; R1 = -1024

ADD R0 ,R0 ,#0
BRzp     M
Q
P   HALT
NUM  .FILL xFC00
```

```
FA .FILL x03A2
FB .FILL x0001
FC  .FILL x0002
FD .FILL x0236
.END
```

程序分析

此段代码的思路依然是经典的“移位寄存器”的思路，但略有不同的是，这里使用了R7（F(n)）反推R6（F(n - 1)），具体方法为使用R4记录R5(F(n - 3))，再用R7将R4的相反数的2倍减去即可。我认为该程序可改进之处主要有以下几点：

- 循环的指令显然是时间的主要开销，因此必须减少赘余的循环指令，例如：

  ```
  ADD R4 ,R5 ,#0; R4 = R5
  ADD R2 ,R4 ,R4; R2 += R4
  ```

  可以缩为1个语句

  ```
  ADD R2 ,R5 ,R5
  ```

  同时，对于R0的判断也是如此，可以将R0自减的语句放到判断语句之前

  另外，这里对1024取模的方法是在循环里不断减1024，直至减为负数，然后再加上1024：

  ```
  LD R1, NUM

  N
  ADD R7,R7,R1; R7 += R1(mod 1024)
  BRzp    N; if(r7 >= 0) back

  NOT R1,R1
  ADD R1,R1,#1; R1 = 1024

  ADD R7,R7,R1; minus, should add to postive
  NOT R1,R1
  ADD R1,R1,#1; R1 = -1024

  NUM .FILL xFC00
  ```

  事实上可以采取对1023取按位与运算来实现，而不必要使用循环减来实现：

  ```
  LD R1 NUM
  AND R7, R7, R1
  NUM .FILL x03FF
  ```

  由于16位溢出后自动对65536取模，由于65536是1024的64倍，因此无需在循环内取模，而是在最后输出结果之前取模。

  进行上述改动后，程序可以改写为：

  ```
  .ORIG  x3000
  ADD R7 ,R7 ,#1
  ADD R6 ,R6 ,#1
  LD R1, NUM
  ```

```
ADD R5 ,R5 ,#1

ADD R0 ,R0 ,#-1; check 1
BRnz P

ADD R7 ,R7,#1; check 2
ADD R0 ,R0,#-1
BRz Q

ADD R0,R0,#-1
M
ADD R2 ,R5 ,R5

NOT R2,R2
ADD R2,R2 #1; R2 = -R2

ADD R3,R5,R5; R3 = 2 * R5
ADD R7,R7,R3; R7 += R3
ADD R5,R6,#0; R5 = R6
ADD R6,R7,R2; R6 = R7 + R2

ADD R0 ,R0 ,#-1
BRzp     M
AND R7, R7, R1
Q
P   HALT
NUM .FILL x03FF
FA .FILL x03A2
FB .FILL x0001
FC  .FILL x0002
FD .FILL x0236
.END
```

循环中的语句从17条减少为8条指令，因此运行效率基本变为原来的2倍。

- 在循环过程中，我们可以一次直接求出F(n + 1)，F(n + 2)，F(n + 3)，从而循环数减少为原来的三分之一，相应的，其中的指令数也可能会增加，但我们不妨做出尝试：

```
.ORIG x3000

;INITIAL
ADD R1,R1,#1
ADD R2,R2,#1
ADD R7,R7,#2
LD R5,MOD

ADD R0,R0,#-3
BRn PRINT

CAL
ADD R1,R1,R1; R1 = 2 * R1
ADD R1,R1,R7; R1 += R7, F(n + 1) = F(n) + 2 * F(n - 2)
ADD R2,R2,R2; R2 = 2 * R2
ADD R2,R2,R1; R2 += R1, F(n + 2) = F(n + 1) + 2 * F(n - 1)
ADD R7,R7,R7; R7 = 2 * R7
ADD R7,R7,R2; R7 += R2, F(n + 3) = F(n + 2) + 2 * F(n)
ADD R0,R0,#-3
```

```
BRzp CAL;

PRINT
ADD R0,R0,#2;
BRz NUM2;
BRp OVER;

NUM1
AND R7,R1,R5
BRnzp OVER

NUM2
AND R7,R2,R5

OVER
AND R7,R7,R5
HALT
MOD .FILL #1023
FA .FILL x03A2
FB .FILL x0001
FC .FILL x0002
FD .FILL x0236
.End
```

在这段改进代码中，循环部分也为8指令，但循环次数减少为三分之一。当数据很大的时候，多出来的输出语句就可以被忽视，因此这种改进方法在数据规模很大的时候效果尤为显著。而在[1, 16384]的数据规模下，毫无疑问，平均指令数一定会减少。

- 在对给出的测试数据进行测试后，初步修改版本执行命令平均为29968.69条，最终修改版本执行命令平均为8890.46条，而原代码执行命令平均为66512.69条，达到了优化的效果。
- 此后，在对实验结果进行整理总结后，我发现数列从第20项开始出现了长度为128的循环节。因此，我使用查表的方法，将指令执行条数大幅缩减。代码如下：

```
.ORIG x3000
LEA R1, LIST
LD  R6, MOD
ADD R2, R0,#-10
ADD R2, R2,#-10
BRn #1

AND R2,R2,R6
ADD R1,R1,R2; Find the result
LDR R7,R1,#0
HALT
MOD .FILL #127

.FILL #1
.FILL #2
.FILL #4
.FILL #6
.FILL #10
.FILL #18
.FILL #30
.FILL #50
.FILL #86
.FILL #146
.FILL #246
```

```
.FILL #418
.FILL #710
.FILL #178
.FILL #1014
.FILL #386
.FILL #742
.FILL #722
.FILL #470
LIST
.FILL #930
.FILL #326
.FILL #242
.FILL #54
.FILL #706
.FILL #166
.FILL #274
.FILL #662
.FILL #994
.FILL #518
.FILL #818
.FILL #758
.FILL #770
.FILL #358
.FILL #850
.FILL #342
.FILL #34
.FILL #710
.FILL #370
.FILL #438
.FILL #834
.FILL #550
.FILL #402
.FILL #22
.FILL #98
.FILL #902
.FILL #946
.FILL #118
.FILL #898
.FILL #742
.FILL #978
.FILL #726
.FILL #162
.FILL #70
.FILL #498
.FILL #822
.FILL #962
.FILL #934
.FILL #530
.FILL #406
.FILL #226
.FILL #262
.FILL #50
.FILL #502
.FILL #2
.FILL #102
.FILL #82
.FILL #86
.FILL #290
```

```
.FILL #454
.FILL #626
.FILL #182
.FILL #66
.FILL #294
.FILL #658
.FILL #790
.FILL #354
.FILL #646
.FILL #178
.FILL #886
.FILL #130
.FILL #486
.FILL #210
.FILL #470
.FILL #418
.FILL #838
.FILL #754
.FILL #566
.FILL #194
.FILL #678
.FILL #786
.FILL #150
.FILL #482
.FILL #6
.FILL #306
.FILL #246
.FILL #258
.FILL #870
.FILL #338
.FILL #854
.FILL #546
.FILL #198
.FILL #882
.FILL #950
.FILL #322
.FILL #38
.FILL #914
.FILL #534
.FILL #610
.FILL #390
.FILL #434
.FILL #630
.FILL #386
.FILL #230
.FILL #466
.FILL #214
.FILL #674
.FILL #582
.FILL #1010
.FILL #310
.FILL #450
.FILL #422
.FILL #18
.FILL #918
.FILL #738
.FILL #774
.FILL #562
```

```
.FILL #1014
.FILL #514
.FILL #614
.FILL #594
.FILL #598
.FILL #802
.FILL #966
.FILL #114
.FILL #694
.FILL #578
.FILL #806
.FILL #146
.FILL #278
.FILL #866
.FILL #134
.FILL #690
.FILL #374
.FILL #642
.FILL #998
.FILL #722
.FILL #982
.END
```

实验总结

得到代码后，我通过分析，发现其中存在许多冗余的指令，因此我首先对指令进行了梳理，去除了多余的指令。但其中仍然在循环中存在求负数这种操作，而这无疑增大了时间开销，当我在设法修改时，我注意到此处移位寄存器的移位过程是存在隐形的冗余的，因此我结合这个数列的性质，直接使用递推求得F(n + 1)，F(n + 2)，F(n + 3)，并进行选择性的输出，这样就可以极大减少循环的次数，得到一个执行效率较高的代码。

但是，使用寻常思路的优化始终无法取得实质性的突破，通过对结果的研究我们往往可以得到意想不到的规律结果，因此在本次本次实验中，最终我选择了查表的方法进行优化，得到了极其惊人的优化效果。