中国科学技术大学计算机学院
《计算机系统概论》实验报告

实验题目：两正数的最大公因数

学生姓名：王章瀚

学生学号：PB18111697

完成日期：2019 年 12 月 5 日

计算机实验教学中心制

2019 年 09 月

1 实验要求

1.1 总述

用 LC-3 汇编语言写一个程序，并编译成 obj 文件。这个程序用以计算两个正数的最大公因数。

1.2 细节要求

- 两个正数以 16-bit 有符号数给出，存在 R0 和 R1 两个寄存器. 输出应置于 R0.
- 不要使用除了 x3000-xFDFF 的内存空间.
- 程序结尾应是 HALT.
- R7 应该保持不变.

2 设计思路

2.1 总述

要求最大公因数，必然能想到的是辗转相除法或更相减损术。然而在 LC-3 中，并没有除法可供直接使用，因为其没有直接的硬件上实现。因此即便想用辗转相除法的思路，也将在 LC-3 中退化为更相减损术。

诚然，存在其他一些方法，比如逐一检验法（从最小的一个数开始逐个到 1 地检验是不是公因子）或取素因数分解指数最小值。这些方法可能在高级语言中看上去会比较优美，但在 LC-3 这样一个 RISC 指令集中，似乎并不那么适用。

综上所述，本次实验，将使用更相减损术的思路来完成。下面具体阐述算法。

2.2 初步思路

既然使用更相减损术，则相互做减法运算即可。

以下是算法流程图。R0,R1 为相减过程中的数，R2 用以计算-R1。

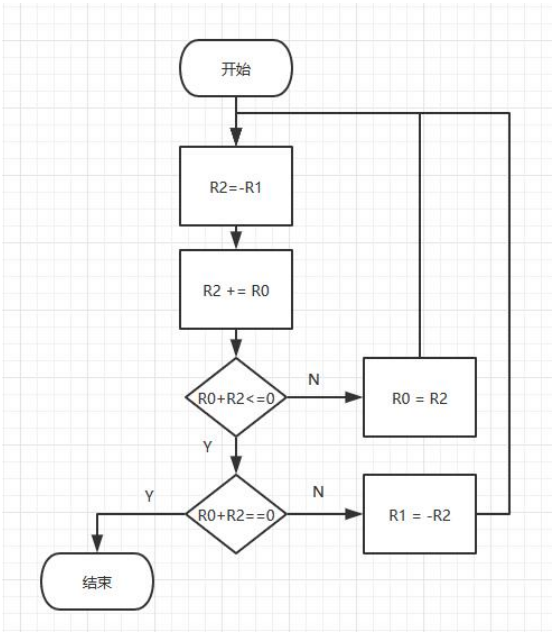

图 1: 初步思路的流程图

至于其具体代码，下面将会讲解。

2.3 改进思路

上面的初步思路有个很大的缺点，就是当一个数远大于另一个数的时候，需要做多次减法，但每一次都重新计算了同一个数的相反数，这使用的指令数是比较多的。

以下是一个优化的办法：不妨设 R0 远大于 R1，则令 R3=R0-R1，然后循环这条指令”R3=R3-R1”，直到 R3<0，这时只需要将 R0 赋值为 R3+R1 即可。另一种情况同理。这样子就免去了多次计算的麻烦。

以下是这种思路的流程图，为了简洁起见，减法运算直接表示出来，而不是用“取反加一”来表示。

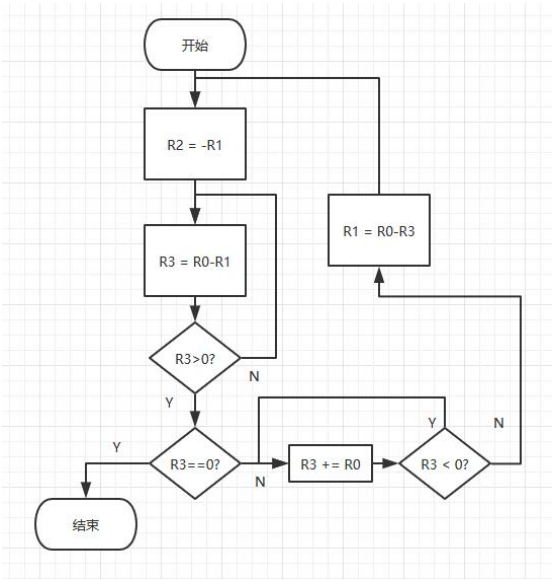

图 2: 改进思路的流程图

至于其具体代码，下面将会讲解。

2.4 另一个思路：结合查找表

除了上述的改进办法以外，还有一些其他的想法。例如，可以用查找表来实现右移，以此能够较好地避免两数相差多大需要多减很多次。

这一步改进的主要思路可以由下面流程图体现：

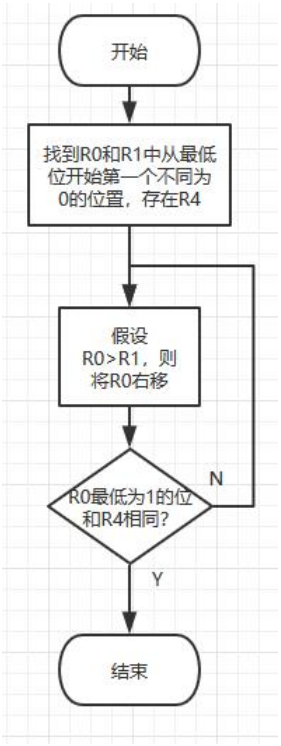

图 3: 结合查找表右移的方法

而查找表中，将第 14 位单独判断，后面几位利用 x8000 以后的 x8000 个内存空间来直接查找获得。表的部分如下：

x4000	x0000
x4001	x0000
x4002	x0001
x4003	x0001
x4004	x0002
...	...

该方法的一个示例如下：

步骤	R0	R1
0	1100	0010
1.1	0110	0010
1.2	0100	0010
2.1	0010	0010
2.2	0000	0010

3 时空复杂度分析

3.1 初步思路

对于初步思路，使用更相减损术。

空间复杂度上，使用了三个寄存器，为 o(3) o(1)。

时间复杂度上，最好的情况就是两数相等，此时时间复杂度为 o(1)；而最坏的情况中，有一种是一个数为 1，另一个数为 a 时，复杂度为 o(a)。可想而知，若两个数为 a,b。则相减次数必然不超过 max{a,b} 就可以减到 1。综合这两点可以看出来，其时间复杂度最坏为 o(max{a, b})

3.2 改进思路

对于改进了的思路，多使用了一个寄存器。

空间复杂度应为 o(4) o(1)。

时间复杂度上，由于改进处在于减少了取相反数的次数。因此就复杂度而言仍然是 o(max{a, b}) 的。只不过每一个循环主体少执行了一些语句。

3.3 结合查找表

由于加了查找表，多用了 x8000 个地址，故其空间复杂度为 o(x8000)，而时间复杂度上，假设为 a, b 两数。且 a 末尾 0 比 b 多 n 个，则时间复杂度至多为 $o(max\{\frac{a}{2^n}, b\})$。

4 关键代码讲解

4.1 初步思路

初步思路的代码实现如下：

```
1                .ORIG    x3000
2
3     LOOP       NOT R2, R1
4                ADD R2, R2, #1
5                ADD R2, R0, R2
6                BRnz     NZ          ;R0>R1
7                ADD R0, R2, #0
8                BRnzp    LOOP
9     NZ         BRn N                ;R0<R1
10               BRnzp    OK          ;R0==R1
11    N          ADD R2, R2, #-1
12               NOT R2, R2
13               ADD R1, R2, #0
14               BRnzp    LOOP
15
16    OK         HALT
17
18               .END
19
```

初步思路的代码

按照上述思路，对 R0>R1, R0==R1, R0<R1 三种情况进行分类处理，代码十分简洁，相应注释已写上，故不再赘述其内容。

4.2 改进思路

改进思路的代码实现如下：

```
1                .ORIG    x3000
2
3     LOOP       NOT R2, R1
4                ADD R2, R2, #1
5                ADD R3, R0, R2
6                BRnz     NZ          ;R0 > R1
7     P          ADD R3, R3, R2       ;减减看
8                BRp P                ;是正数则减到非正
9                ADD R0, R1, R3       ;更新R0为R1+R3
10               BRnzp    LOOP
11    NZ         BRn N
12               BRnzp    OK          ;R0==R1，结束
13    N          ADD R3, R3, R0       ;否则R0<R1，加加看
14               BRn N                ;是负数则加到非负
15               NOT R3, R3
16               ADD R3, R3, #1
17               ADD R1, R0, R3       ;更新R1为R0-R3
18               BRnzp    LOOP
19
20    OK         HALT
21
22               .END
```

```
23
```

改进思路的代码

在该代码中，标签 P 表示的部分是对 R0>R1 的情况的处理，N 部分表示的是 R0<R1 的情况处理。代码相对清晰，注释也较为完善，可根据注释和上述流程图较好地理解程序。

4.3 另一个思路：结合查找表

```
24              .ORIG    x3000
25
26      ; 测试需要移位到哪
27      RSHIFT      NOT     R2, R0
28                  NOT     R3, R1
29                  LD      R4, DEC1
30                  AND     R5, R2, R3
31      NEXT_TEST   AND     R6, R5, R4
32                  BRz     DO
33                  ADD     R4, R4, R4
34                  BRnzp   NEXT_TEST
35
36      ; 移位开始前的一些准备
37      DO          LD      R2, HEX4000
38                  AND     R5, R0, R4
39                  BRz     R0_RSHIFT
40
41      ; 对R1右移
42      R1_RSHIFT   ADD     R3, R1, R2
43                  LDR     R1, R3, #0
44                  AND     R5, R1, R4   ;确认是否移到
45                  BRz     R1_RSHIFT
46                  BRnzp   MINUS
47
48      ; 对R0右移
49      R0_RSHIFT   ADD     R3, R0, R2
50                  LDR     R0, R3, #0
51                  AND     R5, R0, R4   ;确认是否移到
52                  BRz     R0_RSHIFT
53
54      ; 更相减损术
55      MINUS       NOT     R3, R1
56                  ADD     R3, R3, #1
57                  ADD     R3, R0, R3
58                  BRn     MINUS_TO_N
59                  BRz     OK
60                  ADD     R0, R3, #0
61                  BRnzp   RSHIFT
62      MINUS_TO_N  ADD     R3, R3, #-1
63                  NOT     R3, R3
```

```
64                  ADD     R1, R3, #0
65                  BRnzp   RSHIFT
66
67      OK          HALT
68
69      DATA_LOC    .FILL   xD000
70
71      LUT         .FILL   x4000
72      DEC1        .FILL   #1
73      HEX4000     .FILL   x4000
74      .END
75
```

改进思路的代码

5 调试分析

5.1 平均指令数

为了检验改进是否有效，用 C 语言生成了 10000 个随机数来测试，结果如下：

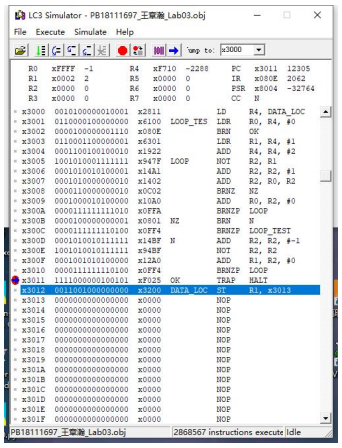

图 4: 10000 个随机数测试——初步思路（平均指令数 2868567/5000=574 条）

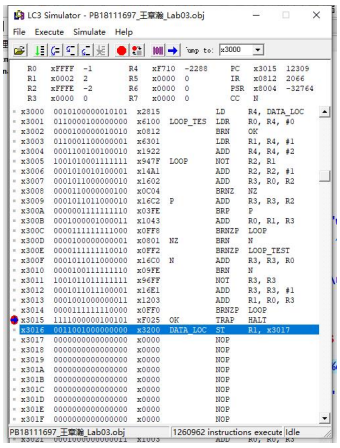

图 5: 10000 个随机数测试——改进思路（平均指令数 1260962/5000=252 条）

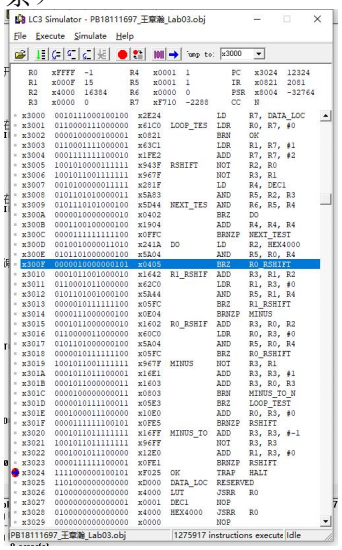

图 6: 10000 个随机数测试——结合查找表（平均指令数 1275917/5000=255 条）

可以看出来，改进后的方式的平均指令数为改进前的 $\frac{2}{5}$ 左右。确实是

很大的进步。而结合查找表和改进思路不相上下。

注：上面截图中 R0 均为-1，是因为设置了-1 作为哨兵来确认是否使用了所有数据。并不是求出了最大公因数为-1。

5.2 特殊测试数据选取

对于此题的测试数据选取，有三个极端值得一看考虑。

5.2.1 测试数据 1

一种应该选取两数之比接近 $\frac{\sqrt{5}+1}{2}$ 的值，如 1618 和 1000。这能将改良算法的优化降低至接近 0 甚至比原来差。这个值的推导过程大致如下表：

R0	x	x-y	x-y	2x-3y	...
R1	y	y	2y-x	2y-x	...
另一寄存器改值的条件	x>y	2y>x	$x > \frac{3}{2}y$	$\frac{5}{3}y > x$	...

最后会得到 x 和 y 的比例应该是 $\frac{\sqrt{5}+1}{2}$ 左右。

测试结果如下：

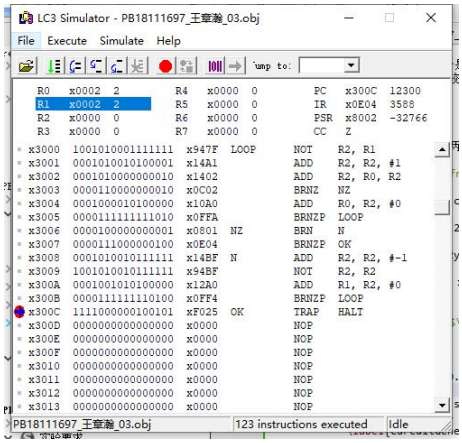

图 7: 测试数据 1——初步思路（123 条指令）

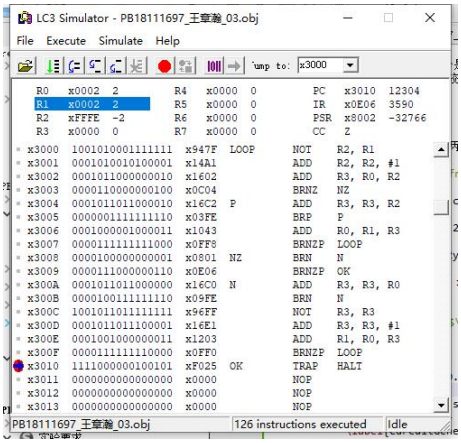

图 8: 测试数据 1——改进思路（126 条指令）

5.2.2 测试数据 2

另一种则是前面提到的一个是 1，另一个是 n。为了表现出其极端性，取 1 和 x0FFF。

测试结果如下：

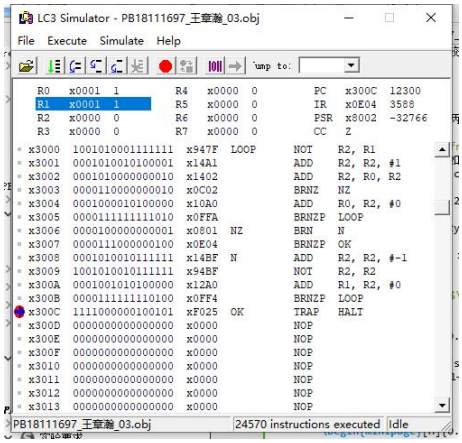

图 9: 测试数据 2——初步思路（24570 条指令）

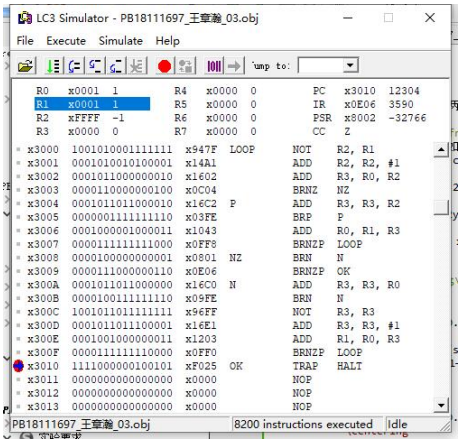

图 10: 测试数据 2——改进思路（8200 条指令）

5.2.3 测试数据 3

另一种则是为了对比查找表方法和改进思路方法。取 x7000 和 x0001。则有如下运行结果：

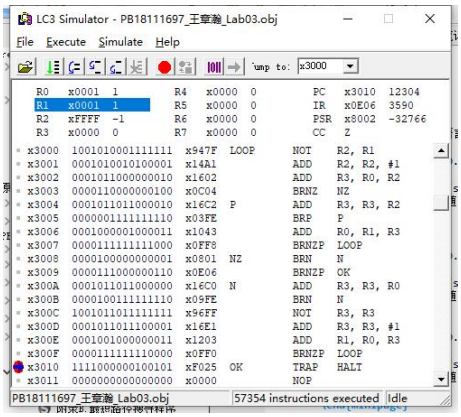

图 11: 测试数据 2——初步思路（57354 条指令）

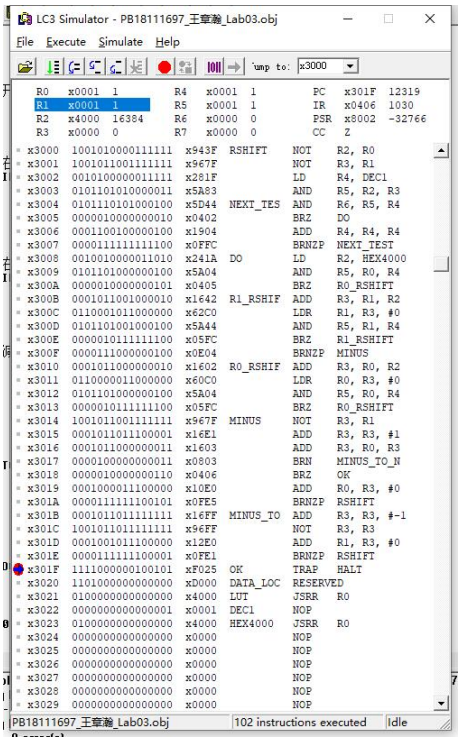

图 12: 测试数据 2——结合查找表思路（102 条指令）

可以明显看出，但两个数的末尾 0 数量差的比较多的时候，借助查找表执行右移将大大提升效率。

5.3 对测试结果的评述

可以看出来，对改进算法最不利的时候，多使用的语句数也并不多很多。然而对初步思路最不利的时候，改进思路的提升就很显然了。

而对于查找表右移的方法，最佳情况是两个数的末尾 0 个数差的比较多，可以快速右移而不是一个一个慢慢地减。

6 对负数和零的处理

如果是负数，可以一开始就检测正负性，将负数改为正数即可。如果是 0，则直接将 R0 置为另一个数的绝对值。这都是十分简单的处理。

7 Great Idea

• **更相减损术中的求相反数**　由于可能要用到多次这个相反数，于是将得到的相反数重复利用直到不可再用。

• **查找表的使用**　由于第三个思路需要用到右移的操作。而从上一次实验可以知道，常规右移需要的指令数是极大的，因此如果能借用查找表，将会使得指令数只需要一到两条。大大减少了右移的损耗。

8 实验总结

本实验在 LC-3 中实现了更相减损术。可以看出来，对于 LC-3 这样的精简指令集体系结构，我们难以使用除法这样的计算；此外，如果能够重复利用的数据一定要重复利用，尽量避免重新计算带来的对执行指令数的大幅增加。

此外，查找表也是一种很好的方法，它能够避免一些难以实现的运算从而以空间换时间来完成想要的功能。

最后说明，由于查找表不容易让助教批改，故提交的是改进思路的代码。