GCD算法

1、实验要求

Your Mission: Write a program in LC-3 assembly language that is used to calculate the greatest common divisor (GCD) of two positive numbers. Details:

1. Two positive 16-bit signed integers will be given in R0 and R1 register. (You can fill the values in the code or modify registers manually in LC3 simulator.)
2. The output value should be put in R0.
3. Your program should start at memory location x3000 and end with HALT.

2、实验思路

首先讨论几种常见的GCD算法：

1、辗转相除法

已知a,b,c为正整数，若a除以b余c，则GCD（a,b）=GCD (b,c)。

2、更相减损术：

任意给定两个正整数，若是偶数，则用2约简。

以较大的数减较小的数，接着把所得的差与较小的数比较，并以大数减小数。

继续这个操作，直到所得的减数和差相等为止。

3、除穷举法：

将小数依次除N（N为从1开始的自然数，结果不为整数则跳过），对得到的数判断其是否可被大数整除。

4、减穷举法：

将小数依次减1，对得到的数判断其是否可被两数整除。

根据资料查询以及Python程序的测试得到结果是：辗转相除法、更相减损术的效率远高于穷举法，且这两个方法的计算规模几乎不受计算对象量级的影响。辗转相除法的效率最高，更相减损术次之。除穷举法的效率高于减穷举法。所以在这里采用辗转相除法

这个算法的关键在于如何用LC-3汇编实现a mod b的过程，考虑实现思路如下：

法一：a一次减b，用R0来储存前一个相减的值，用R2来储存相减之后的值，如果相减之后的值小于0，则回退，并且比较ab大小，如果a<b，则交换ab重复此操作，如果等于0则结束并且判定b为最大公约数。

代码分析：

【part1】

```
.ORIG X3000
AND R2,R2,#0
AND R3,R3,#0
AND R4,R4,#0
AND R5,R5,#0;COUNTER1
AND R6,R6,#0;COUNTER2
ADD R2,R0,#0
```

初始化一些需要用到的寄存器，其中R5,R6用于计算经过了多少循环，便于分析性能与用到的代码量。需要计算gcd的两个输入值存入R0和R1中。

【part2】

```
JUDGE1:        ADD R0,R0,#0
        BRp JUDGERs
         BRz ZERO
         BRn REVER0
JUDGERs ADD R1,R1,#0
         BRp GCD
         BRz ZERO
         BRn REVER1
ZERO AND R0,R0,#0
     HALT
REVER0:NOT R0,R0
        ADD R0,R0,#1
        ADD R2,R0,#0
        BRnzp JUDGERs
REVER1:NOT R1,R1
       ADD R1,R1,#1
```

特殊情况处理，对于输入为0或者是负数的情况进行讨论，如果输入为负数，则将其取相反数后进行gcd运算，如果输入为0，则直接输出0.

```
GCD ADD R5,R5,#1
    ADD R0,R2,#0
    NOT R3,R1
    ADD R3,R3,#1
    ADD R2,R0,R3

    BRz FINAL
    BRp GCD
```

将R0的值减去R1的值，将所得到的的值存入R2中，如果得到的值大于0，则将R2的值给R0并且继续这样的操作，否则后退，并且交换R0和R1中的值。如果为0，则结束循环，将R1的值赋给R0作为输出。

```
    ADD R2,R0,#0
    ;swap R0,R1
    ADD R4,R0,#0
    ADD R0,R1,#0
    ADD R1,R4,#0
    ADD R2,R0,#0
    BRnzp GCD
```

交换R0和R1的值。

```
FINAL ADD R0,R2,#0
ADD R0,R1,#0
HALT
```

将所得到的值赋值给R0，结束

【测试用例1】R0<R1且GCD=1

R0=156,R1=283

测试结果：

1（与实际相符）

【测试用例2】R0<R1且GCD≠1

R0=14,R1=56

测试结果：14（与实际相符）

1（与实际相符）

【测试用例3】R0>R1且GCD≠1

R0=256,R1=88

测试结果：

8（与实际相符）

【测试用例4】R0>R1且GCD=1

R0=256,R1=83

测试结果：

1（与实际相符）

【测试用例5】输入为0情况分析

R0=45,R1=0

测试结果：

0（与实际相符）

【测试用例6】输入为负数情况分析

R0=-187,R1=44

测试结果：

1（符合要求）

这里为了测试算法的性能添加了R5和R6两个counter来计算一共使用了多少条语句，其中R5记录了一共经过了多少次GCD块，R6记录经过多少次swap块。

测试用例：

R0=2451,R1=3434

测试结果：1

R5=55,R6=6

所以所用的指令数约为：6+5*55+6*7+4=327条

PS:特殊情况的考虑

如果在这里输入中有0，那么输出的gcd直接置为0，如多输入的其中某个数为负数,则取这个数的相反数进行下面的运算，确保运行gcd块时r0和r1中都是正数.

代码：

```
.ORIG X3000
AND R2,R2,#0
AND R3,R3,#0
AND R4,R4,#0
AND R5,R5,#0;COUNTER1
AND R6,R6,#0;COUNTER2
ADD R2,R0,#0

JUDGE1:        ADD R0,R0,#0
         BRp JUDGERs
          BRz ZERO
          BRn REVER0
JUDGERs ADD R1,R1,#0
          BRp GCD
          BRz ZERO
          BRn REVER1
ZERO AND R0,R0,#0
     HALT
REVER0:NOT R0,R0
         ADD R0,R0,#1
         ADD R2,R0,#0
         BRnzp JUDGERs
REVER1:NOT R1,R1
       ADD R1,R1,#1

GCD ADD R5,R5,#1
    ADD R0,R2,#0
    NOT R3,R1
    ADD R3,R3,#1
```

```
    ADD R2,R0,R3

    BRz FINAL
    BRp GCD
    ADD R6,R6,#1
    ADD R2,R0,#0
    ;swap R0,R1
    ADD R4,R0,#0
    ADD R0,R1,#0
    ADD R1,R4,#0
    ADD R2,R0,#0
    BRnzp GCD


FINAL ADD R0,R2,#0
ADD R0,R1,#0
HALT
.END
```

法二：存储减数的所有2^n倍，并且从高到低依次与被减数比较，如果小于被减数就从被减数中减去相应的值，最后结束后判断是否被减数是0，如果是，结束，如果不是，交换被减数与减数继续上述操作。

分析：这种方法可以减少a与b相减的次数，实现性能上的优化，但是判断过程以及运算得到减数的所有2^n倍的操作可能会花费更多的指令，尤其是在输入的数比较小的情况下，可能效果并不一定比法一更好。而且，这个方法需要调用内存，涉及到的LD指令的时间代价会更高，以及这种方法并不节约内存。

有关这种方法的算法实现如下：（由于仅用于对照分析不同算法的性能，代码非本人编写，来自github）

```
.ORIG x3000
;first, specify R1 and R0
NOT R4,R1
ADD R4,R4,#1
ADD R2,R4,R0
BRz THEEND
BRn F3
;save the address of the SPACE in R5
LEA R5,SPACE
F0 AND R3,R3,#0
    ;R4 = -R1 in order to save the minus
    NOT R4,R1
    ADD R4,R4,#1
F1 STR R4,R5,#0
    ADD R5,R5,#1
    ADD R3,R3,#1
    ADD R4,R4,R4
    BRn F1
F2 ADD R3,R3,#-1
    BRn F3
    ADD R5,R5,#-1
    LDR R2,R5,#0
    ADD R2,R2,R0
    BRz F4
    BRn F2
```

```
    AND R0,R0,#0
    ADD R0,R0,R2
    BR F2
F3 AND R2,R2,#0
    ADD R2,R2,R0 ;R2=R0
    AND R0,R0,#0
    ADD R0,R0,R1 ;R0=R1
    AND R1,R1,#0
    ADD R1,R1,R2 ;R1=R0
BR F0
F4 AND R0,R0,#1
    ADD R0,R1,#0
THEEND HALT
SPACE .BLKW 16
.END
```

法3：查表法

具体算法如下：

递归执行GCD算法：

1、如果A,B都是偶数，那么GCD(A,B)=2*GCD(A/2,B/2)

2、如果A,B一个是奇数一个是偶数，假设A是奇数，B是偶数，那么GCD(A,B)=GCD(A,B/2)

3、如果A和B都是奇数，那么GCD(A,B)=GCD(abs(A-B)/2,min(A,B)).如果A-B=0,GCD=A

```
    addi ra,ra,-50
    andi a2,a2,0
    andi a3,a3,0
    addi a4,a4,0
    andi a5,a5,0
    andi a6,a6,0
   addi a2,a0,0
   GCD:
   addi a5,a5,1
    addi a0,a2,0
    not a3,a1
    addi a3,a3,1
    add a2,a0,a3
    beqz a2,THEEND
    bgtz a2,GCD
    addi a6,a6,1
    addi a2,a0,0
    addi a4,a0,0
    addi a0,a1,0
    addi a1,a4,0
    addi a2,a0,0
    jal GCD

    THEEND:
```

```
    addi a0,a2,0
    addi a0,a1,0
    andi ra,ra,0
    addi ra,ra,-50
```