中国科学技术大学计算机学院
《计算机系统概论》实验报告

LAB 02: Greatest Common Divisor

姓名:王晨 学号:PE20060014

完成日期:2020 年 12 月 15 日

一、实验要求：

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

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.

Example:

	Before Execution	After Execution
R0	x0009	x0003
R1	x000C	Any value you changed R1 to

二、实验环境：

Mac OS Big Sur 11.01

LC-3 Simulator Mac Version

三、算法思路：

求最大公因数，主要有几种方法：辗转相除法、更相减损法、枚举法。然而由于在 LC-3 中，没有除法指令，且枚举法的时间复杂度较高，也需要用到除法。因此通过LC-3指令集求GCD，采用更相减损法比较合适。使用更相减损法，只需要做减法运算即可，在此不证明更相减损法求GCD算法的正确性。R0, R1 为待求GCD的两个数，求出的GCD保存在R0中。以下是具体算法阐述和流程图：

第一步：若R1=R0，那么R0就是GCD，直接退出。

第二步：否则判断R1和R0的大小（不妨设R1>R0），然后用大的数减去小的数，将大数寄存器R1保存减去后的结果R1-R0，返回到第一步，该步运行结束后保证了大数寄存器保存了相减的结果。

以260，104为例，它们的GCD为52，运算过程如下：

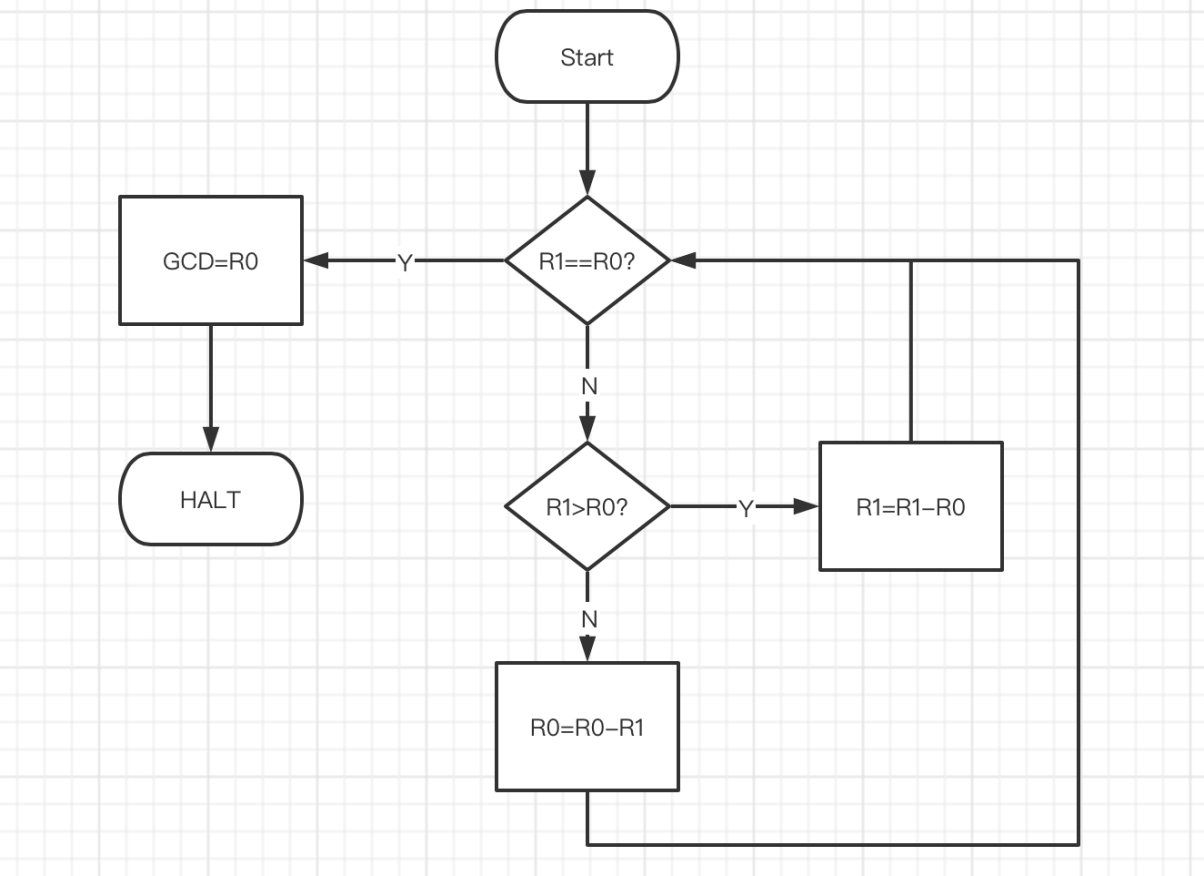
260-104=156

156-104=52

104-52=52

52-52=0

因此GCD=52



四、程序代码及注释：

gcd.asm

```
1             .ORIG x3000
2             LD R0 NUMBER1
3             BRnz ERROR
4             LD R1 NUMBER2
5             BRnz ERROR      ;输入非正整数，输出错误
6   CHECKEQL  NOT R2,R1
7             ADD R2,R2,#1
8             ADD R3,R0,R2    ;R3=R0-R1
9             BRz DONE        ;若R3为0，R0=R1则退出
10            BRp LTR1        ;否则当R0-R1>0时，跳转到LTR1
11                            ;注意此时CC尚未改变，因此可以再次BR
12            NOT R2,R0
13            ADD R2,R2,#1
14            ADD R1,R1,R2    ;R1=R1-R0
15            BRnzp CHECKEQL
16  LTR1      ADD R0,R0,R2    ;R0=R0-R1
17            BRnzp CHECKEQL
18  ERROR     LD R0 ERRNUM
19  DONE      HALT
20  NUMBER1   .FILL x8104
21  NUMBER2   .FILL x0001
22  ERRNUM    .FILL xFFFF     ;错误代码
23            .END
```

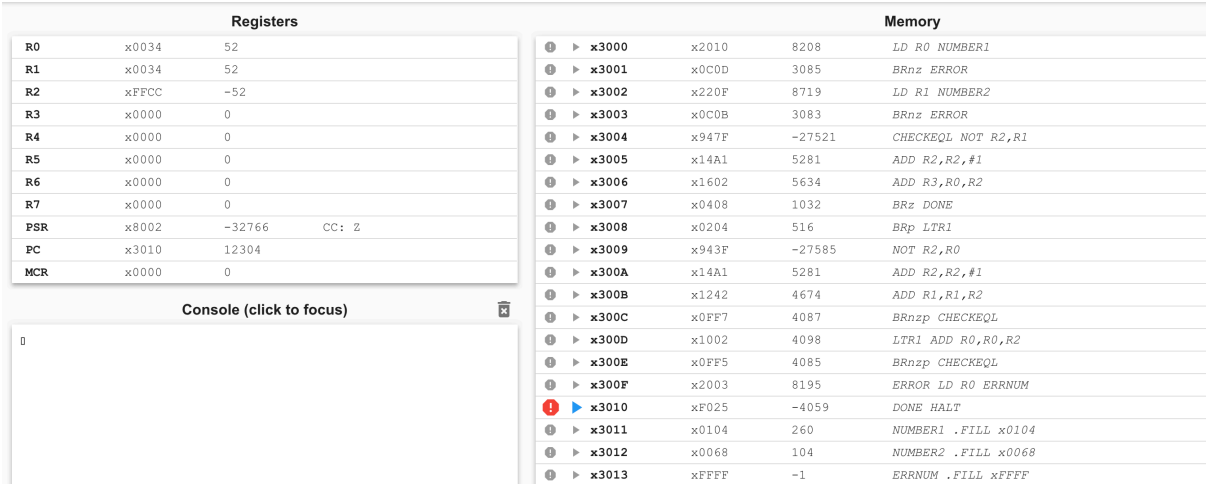
```
            .ORIG x3000
            LD R0 NUMBER1
            BRnz ERROR
            LD R1 NUMBER2
            BRnz ERROR              ;输入非正整数，输出错误
CHECKEQL    NOT R2,R1
            ADD R2,R2,#1
            ADD R3,R0,R2            ;R3=R0-R1
            BRz DONE                ;若R3为0，R0=R1则退出
            BRp LTR1                ;否则当R0-R1>0时，跳转到LTR1
                                    ;注意此时CC尚未改变，因此可以再次BR
            NOT R2,R0
            ADD R2,R2,#1
            ADD R1,R1,R2            ;R1=R1-R0
            BRnzp CHECKEQL
LTR1        ADD R0,R0,R2            ;R0=R0-R1
            BRnzp CHECKEQL
ERROR       LD R0 ERRNUM
DONE        HALT
NUMBER1     .FILL       x0104
NUMBER2     .FILL       x0068
ERRNUM      .FILL       xFFFF       ;错误代码
            .END
```

按照第二节的算法思路，对 R0>R1, R0==R1, R0<R1 三种情况进行分类处理，相应注释已写上，代码和思路都比较简单，故不再赘述其内容。

注意由于GCD一般是对于正整数而言的，因此不考虑R0和R1为负数和0的情况。当R0和R1输入非正整数时，R0会输出错误代码0xFFFF。当然也可以在都为负数的情况下，转换符号，作GCD的计算。

五、调试和测试：

1.R0=260,R1=104,从输出结果可以看到GCD=R0=52，R1=52，R3=0，结果正确。



Registers

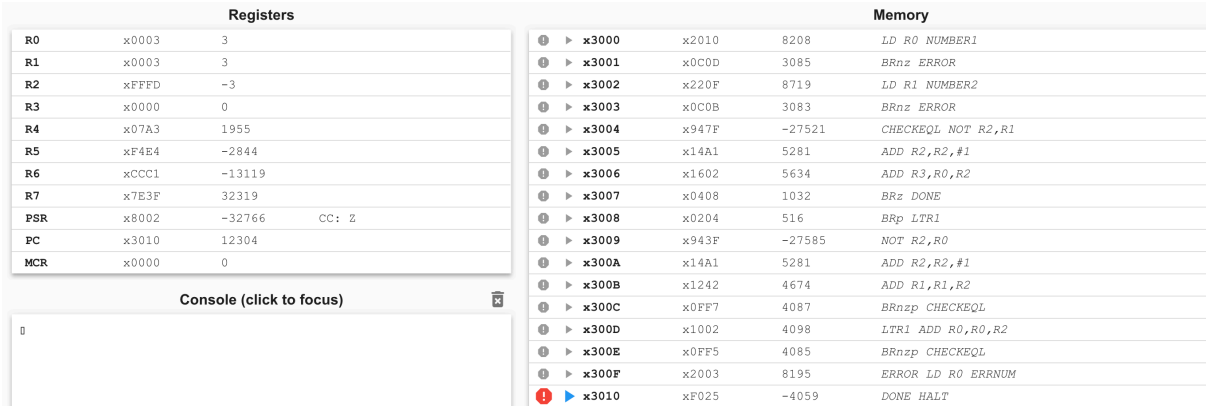
R0	x0034	52	
R1	x0034	52	
R2	xFFCC	-52	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8002	-32766	CC: Z
PC	x3010	12304	
MCR	x0000	0	

Console (click to focus)

Memory

x3000	x2010	8208	LD R0 NUMBER1
x3001	x0C0D	3085	BRnz ERROR
x3002	x220F	8719	LD R1 NUMBER2
x3003	x0C0B	3083	BRnz ERROR
x3004	x947F	-27521	CHECKEQL NOT R2,R1
x3005	x14A1	5281	ADD R2,R2,#1
x3006	x1602	5634	ADD R3,R0,R2
x3007	x0408	1032	BRz DONE
x3008	x0204	516	BRp LTR1
x3009	x943F	-27585	NOT R2,R0
x300A	x14A1	5281	ADD R2,R2,#1
x300B	x1242	4674	ADD R1,R1,R2
x300C	x0FF7	4087	BRnzp CHECKEQL
x300D	x1002	4098	LTR1 ADD R0,R0,R2
x300E	x0FF5	4085	BRnzp CHECKEQL
x300F	x2003	8195	ERROR LD R0 ERRNUM
x3010	xF025	-4059	DONE HALT
x3011	x0104	260	NUMBER1 .FILL x0104
x3012	x0068	104	NUMBER2 .FILL x0068
x3013	xFFFF	-1	ERRNUM .FILL xFFFF

2. 随机尝试两个较大的数，例如R0=2793，R1=4527, GCD=3，结果正确。



Registers

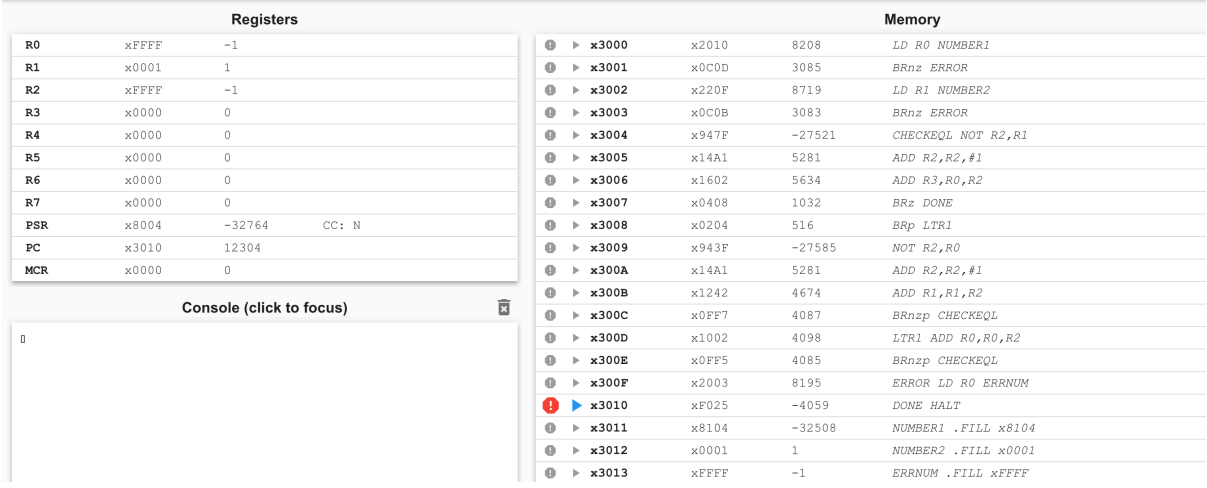
R0	x0003	3	
R1	x0003	3	
R2	xFFFD	-3	
R3	x0000	0	
R4	x07A3	1955	
R5	xF4E4	-2844	
R6	xCCC1	-13119	
R7	x7E3F	32319	
PSR	x8002	-32766	CC: Z
PC	x3010	12304	
MCR	x0000	0	

Console (click to focus)

Memory

x3000	x2010	8208	LD R0 NUMBER1
x3001	x0C0D	3085	BRnz ERROR
x3002	x220F	8719	LD R1 NUMBER2
x3003	x0C0B	3083	BRnz ERROR
x3004	x947F	-27521	CHECKEQL NOT R2,R1
x3005	x14A1	5281	ADD R2,R2,#1
x3006	x1602	5634	ADD R3,R0,R2
x3007	x0408	1032	BRz DONE
x3008	x0204	516	BRp LTR1
x3009	x943F	-27585	NOT R2,R0
x300A	x14A1	5281	ADD R2,R2,#1
x300B	x1242	4674	ADD R1,R1,R2
x300C	x0FF7	4087	BRnzp CHECKEQL
x300D	x1002	4098	LTR1 ADD R0,R0,R2
x300E	x0FF5	4085	BRnzp CHECKEQL
x300F	x2003	8195	ERROR LD R0 ERRNUM
x3010	xF025	-4059	DONE HALT

3. R0=任意值，R1=1，输出结果R0=1，结果正确。



Registers

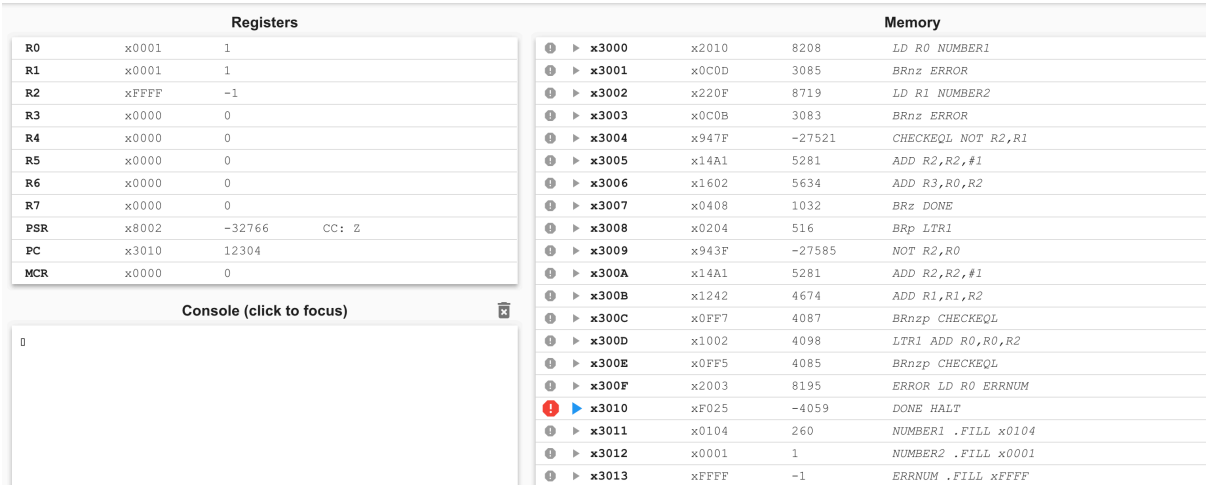
R0	xFFFF	-1	
R1	x0001	1	
R2	xFFFF	-1	
R3	x0000	0	
R4	x0000	0	
R5	x0000	0	
R6	x0000	0	
R7	x0000	0	
PSR	x8004	-32764	CC: N
PC	x3010	12304	
MCR	x0000	0	

Console (click to focus)

Memory

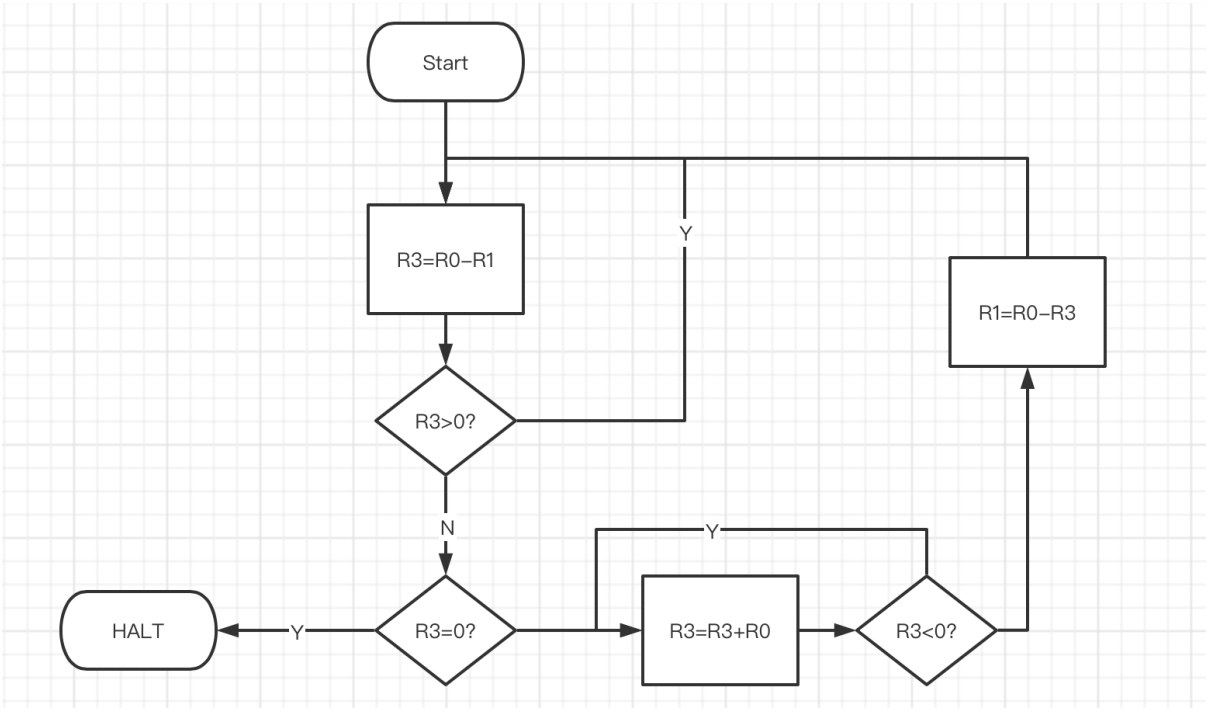
x3000	x2010	8208	LD R0 NUMBER1
x3001	x0C0D	3085	BRnz ERROR
x3002	x220F	8719	LD R1 NUMBER2
x3003	x0C0B	3083	BRnz ERROR
x3004	x947F	-27521	CHECKEQL NOT R2,R1
x3005	x14A1	5281	ADD R2,R2,#1
x3006	x1602	5634	ADD R3,R0,R2
x3007	x0408	1032	BRz DONE
x3008	x0204	516	BRp LTR1
x3009	x943F	-27585	NOT R2,R0
x300A	x14A1	5281	ADD R2,R2,#1
x300B	x1242	4674	ADD R1,R1,R2
x300C	x0FF7	4087	BRnzp CHECKEQL
x300D	x1002	4098	LTR1 ADD R0,R0,R2
x300E	x0FF5	4085	BRnzp CHECKEQL
x300F	x2003	8195	ERROR LD R0 ERRNUM
x3010	xF025	-4059	DONE HALT
x3011	x8104	-32508	NUMBER1 .FILL x8104
x3012	x0001	1	NUMBER2 .FILL x0001
x3013	xFFFF	-1	ERRNUM .FILL xFFFF

4. R0或R1为负数或0，例如R0=x8104，R1=1，输出错误代码R0=0xFFFF，正确。



六、分析和改进：

使用未经优化的更相减损法，在空间复杂度上，总共使用了四个寄存器，R0~R3。在时间复杂度上，最好的情况是两数相等，时间复杂度为 o(1); 而对于最坏的情况，有一种是一个数为 1，另一个数为n时，复杂度为o(n)。若两个数为A、B，则相减的次数最多不超过 max{A, B} 就可以减到 1。 因此，综合这几点来看，其时间复杂度为 o(max{A, B})。从以上可以看出，未经优化的更相减损法的缺点是，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。



上图是第一种优化的办法: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重复的计算。

可以看到，改进后的方法在逻辑上稍复杂些，但可以避免多次计算相同的相反数，减少运行的指令数，在空间复杂度上与之前相同，平均时间复杂度仍然为o(max{A, B})。

第二种方法，是通过右移来实现除2的操作，从而实现减少多次减法以及不必要的计算。结合第一次实验，当R0和R1中有偶数时，我们可以通过对二进制串逻辑左移15次，来实现逻辑右移1位，即实现除2操作，这种方法对于存在某一个或者两个极大偶数的情况，比较有效，可以很快降低数量级，用一次除法达到很多次减法的效果。当两个数最后都为奇数时，用更相减损法计算出结果，最后再将结果乘若干次2。在这里就不再编写程序计算了。

七、实验心得：

本实验通过更相减损法实现了求GCD，熟悉了LC-3汇编的编写，实验成功。由于LC-3没有硬件直接支持除法指令集，因此无法采用基于除法的算法实现GCD的计算。通过更相减损法，可以仅通过加/减法计算GCD，在逻辑和实现上是最简单的，但是处理较大值的时候效果不佳，因此还是要根据实际情况，调整算法，采用上述的两种方法进行优化。