- 1. 实验题目: GREATEST COMMON DIVISOR
- 2. 实验要求

#### Your job:

Write a program in LC-3 assembly language and assemble it into LC-3 object file using an assembler.

The program is used to calculate the greatest common divisor of two positive numbers.

#### Details:

ullet Two positive16-bit signed integers will be given in RO and R1 register. And the output value should

be put in RO.

- DO NOT access any part of memory other than x3000-xFDFF.
- The program should be ended with HALT.
- R7 register should remain the same after the execution

#### 3. 实验思路&&代码讲解

#### 三种写法的探讨:

法一考虑了储存和从高到低比较,和我写的第二版本来说,还是耗费了更多指令,但我觉得这种储存思维对于要减的位数更多的测试情况会更好,思路也比较特别,特此记录(内容很长因为是我第一版本的实验报告( $^{\wedge}$ <sub>2</sub>))。

法二是我选择的"傻瓜"算法,减数左移到最高位,从被减数减去后,重新循环,左移到最高位,再减去。根据我粗略地平均随机测试集指令数,指令数与法三"查表法"相差不太多(由于LC3位数少,查表法的优化有限),但非常节约空间,也不需要做太多预先处理,我个人认为这个算法更好。

法三是qq群里有人提到的"查表法",我没有具体写在报告中,我认为内存空间同样需要节省。(图来自罗丽薇同学)

Performing the GCD algorithm recursively.

- 1. Both A and B are even, GCD(A,B)=2\*GCD(A/2,B/2);
- 2. An odd and an even number, assuming that A is odd and B is even, GCD(A,B) = GCD(A,B/2);
- 3. Both A and B are odd, GCD(A,B) = GCD(abs(A-B)/2, min(A,B)). When A-B=0, GCD=A;

To perform /2 efficiently, create an array Div2 before calculation. Furthermore, Div2[x]=x>>1.

#### 法一:

#### 关键思想:

利用欧几里得辗转相除法: 计算公式gcd(a,b) = gcd(b,a mod b)

LC3并没有除法,那就考虑减法。

考虑二进制形式,存储减数的所有2<sup>n</sup>次倍,并从高到低逐一比较,小于被减数就从被减数中减去相应的值,减到没得减,就交换被减数与减数。判断结束的条件是被减数为0。

#### 具体思路:

```
初判断,利用R2: R1>R0跳转函数3; R1=R0,HALT; <则顺序执行 SPACE申请16个连续地址存减数,R5LEA存SPACE的第一个地址 函数0: 初始化计数器R3=0,R1取反加一的结果放进R4,顺序执行 函数1: R4存进地址,R5地址++,计数器R3++,R4左移,是n就跳转函数1,存完往下执行 函数2: 计数器--,BRn 函数3,地址--,R2取出地址的内容,R2加R0 z:跳转函数4 n:跳回函数2 p:R0<-R2,跳回函数2 函数3: 交换R0和R1,利用R2,结束后跳回函数0 函数4: R0<-R1,HALT
```

### ——对应代码讲解:

。 最初比较R1和R0的大小,根据三种情况选择接下来的执行情况:

```
;first, specify R1 and R0

NOT R4,R1

ADD R4,R4,#1

ADD R2,R4,R0

BRZ THEEND

BRN F3

THEEND HALT
```

。 关于申请的16个内存空间

```
;save the address of the SPACE in R5
LEA R5,SPACE

SPACE .BLKW 16
```

○ 函数0: 初始化计数器R3, R4=-R1

```
FO AND R3,R3,#0
;R4 = -R1 in order to save the minus
NOT R4,R1
ADD R4,R4,#1
```

○ 函数1: R4存入内存, R5指针++, 计数器++, R4左移, 若还是负数, 跳转函数1

```
F1 STR R4,R5,#0
ADD R5,R5,#1
ADD R3,R3,#1
ADD R4,R4,R4
BRn F1
```

○ 函数2:

计数器--,是负数跳转函数3,否则地址--,R2取出地址的内容,R2+=R0

z:跳转函数4

n:跳回函数2

# p:R0<-R2,跳回函数2

```
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
```

。 函数3:交换R0和R1,结束后跳回函数0

```
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
```

○ 函数4: R0=R1, HALT

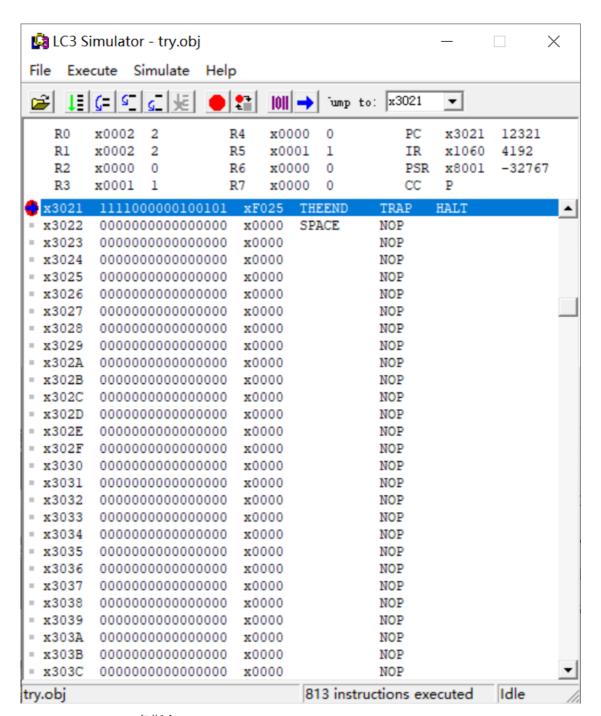
```
F4 AND R0,R0,#1
ADD R0,R1,#0

THEEND HALT
```

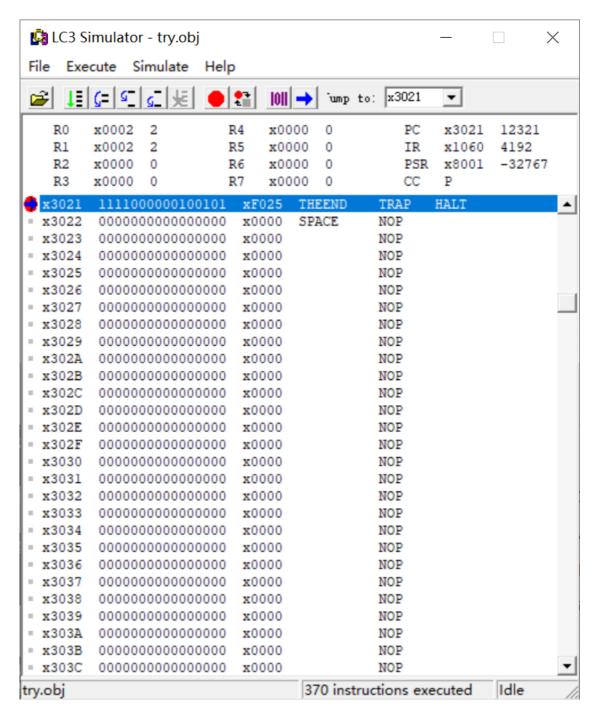
# 4. 运行结果

本次实验指令数与测试数据关系很大, 我随机试了几个:

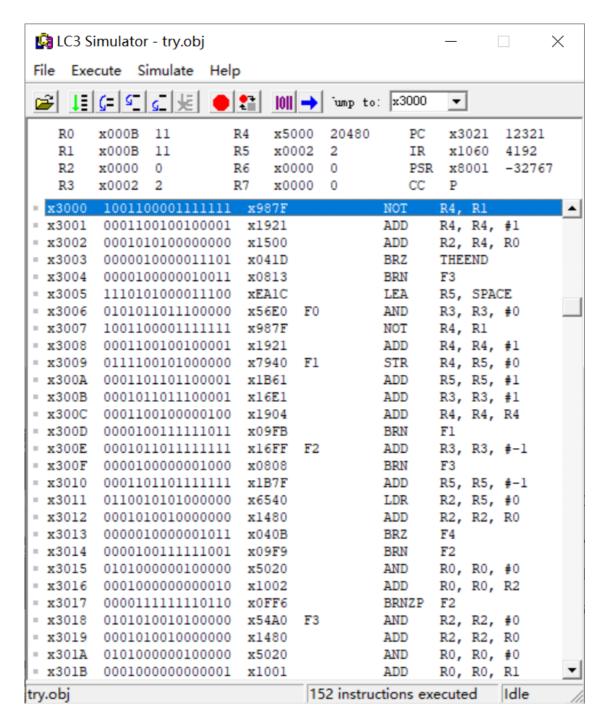
R0=#40, R1=#9102 813条指令



R0=#6, R1=#8 370条指令



R0=#11, R1=#1100 152条指令



## 5. 关于时空复杂度

——"Analyze the space and time complexity of your algorithm in your report."

代码34个空间,数据相关占用16个空间,复杂度为O(1)。

时间最长的情况可以想象是一直在辗转减、减小得很慢,是对数级别的复杂度(log)。

#### 6. 关于输入含负数或0

——"Think about handling negative and zero inputs in your report"

#### 加入预判断:

如果有0,最大公因数为另一个数;

如果有负数,取反加一后开始本程序的运算,但有例外情况,即x8000 (-32768) ,可以任意加上或减去另一个数,因为不影响辗转相除法的算法,之后重新进入预判断即可。

### 7. 源代码

```
.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
FO 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
```

### 法二:

#### 【关键思想】

利用欧几里得辗转相除法: 计算公式gcd(a,b) = gcd(b,a mod b)

LC3并没有除法,那就考虑减法。减数左移到最高位,从被减数减去后,重新循环,左移到最高位,再减去。 (模仿竖式除法)

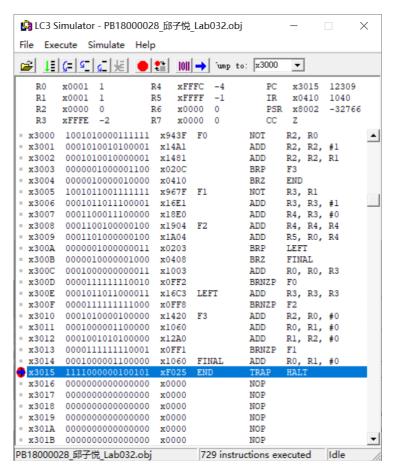
## 【代码】

```
.ORIG x3000
;函数0: 利用R2: R1>R0跳转函数3; R1=R0, HALT; <则顺序执行
F0 NOT R2,R0
ADD R2,R2,#1
ADD R2,R2,R1
BRP F3
BRZ END
```

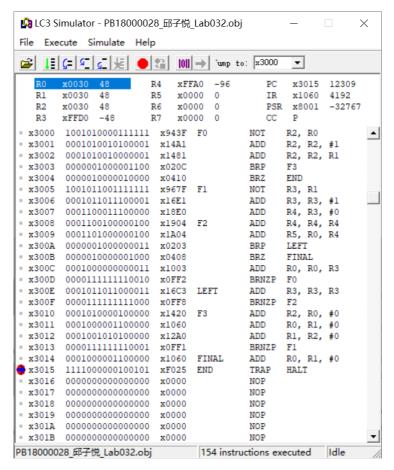
```
;函数1: 初始化R3=-R1,R4=R3(=-R1)
F1 NOT R3,R1
    ADD R3, R3, #1
   ADD R4, R3, #0
;函数2: R4*2, R5=R0-R4,
    p LEFT
   z FINAL
   n R0-=R3,跳回函数0
F2 ADD R4, R4, R4
    ADD R5, R0, R4
    BRp LEFT
    BRZ FINAL
    ADD R0, R0, R3
    BR F0
;LEFT: R3*2, BR 函数2
LEFT ADD R3,R3,R3
    BR F2
;函数3:交换RO和R1,利用R2,结束后跳回函数1
F3 ADD R2, R0, #0
    ADD R0, R1, #0
    ADD R1, R2, #0
    BR F1
;FINAL:R0=R1+#0
;HALT
FINAL ADD RO,R1,#0
END HALT
. END
```

#### 【测试】

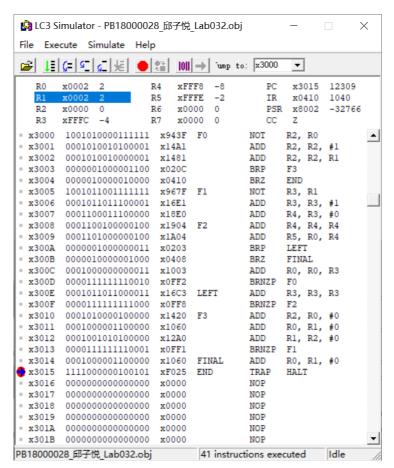
1. 指令数最多的可能性:R1=x7fff, R0=x0001, 729条指令



2. 随机生成: R1=x3930, R0=x2820, 154条指令



3. 指令数比较少的情况: R0=x0008, R1=x0006, 41条指令



#### 【关于时空复杂度】

——"Analyze the space and time complexity of your algorithm in your report." 代码24个空间,复杂度为O(1)。 时间最长的情况可以想象是一直在辗转减、减小得很慢,是对数级别的复杂度(log)。

# 【关于输入含负数或0】

——"Think about handling negative and zero inputs in your report"

# 加入预判断:

如果有0,最大公因数为另一个数;

如果有负数,取反加一后开始本程序的运算,但有例外情况,即x8000 (-32768) ,可以任意加上或减去另一个数,因为不影响辗转相除法的算法,之后重新进入预判断即可。