



**《计算机系统概论》**

**实 验 报 告**

课程名称 计算机系统概论

学生学院 材料与能源学院

专业班级 材料成型及控制工程1班

学 号 3117005505

学生姓名 蔡昂儒

指导教师 张伯泉

2020 年 6 月 12日

**实验一 循环控制程序**

一、实验目的

通过循环控制程序的编写、调试、运行，了解程序执行过程和实现机制，掌握其实现方法，提高程序编写技巧、调试技巧，提升对计算机系统的认识。

二、实验代码

.ORIG x3000

AND R0,R0,#0

LOOP ADD R0,R0,R1

ADD R1,R1,-1

BRp LOOP

TRAP x25

.END

三、运行过程



程序无错误

.ORIG x3000 设初地址为x3000

AND R0,R0,#0 寄存器R0被清零

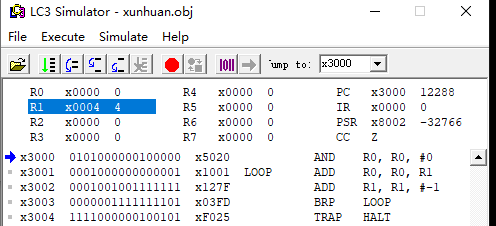
LOOP: ADD R0,R0,R1 循环开始，这里是使R0的值为R0+R1

ADD R1,R1,-1 这里是使R1自减1

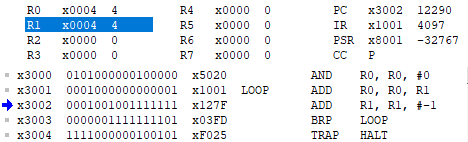
BRp LOOP 循环判断，当R1不为正数时循环结束

TRAP x25 运行至TRAP

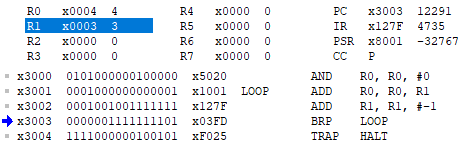
.END 运行结束



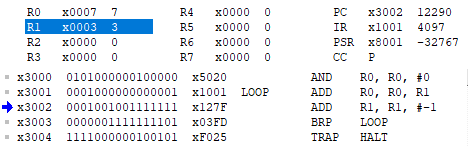
运行前，将R1的初值设为4



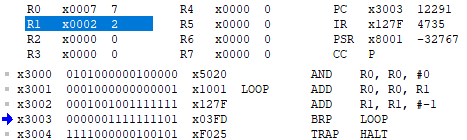
第一次循环，R0=R0+R1=4，R1=4



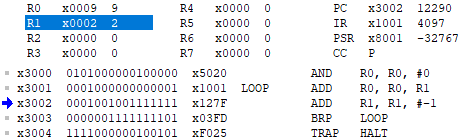
R1=R1-1=3，第一次循环经判断后PC指向LOOP，循环继续



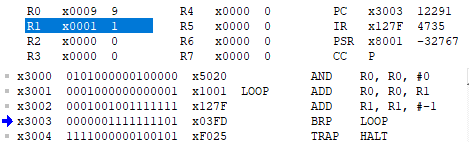
第二次循环，R0=R0+R1=7，R1=3



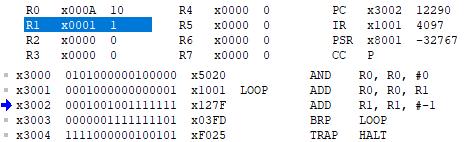
R1=R1-1=2，第二次循环经判断后PC指向LOOP，循环继续



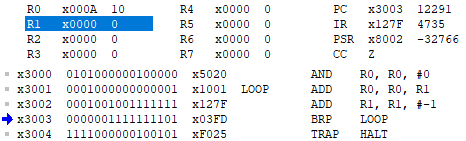
第三次循环，R0=R0+R1=9，R1=2



R1=R1-1=1，第三次循环经判断后PC指向LOOP，循环继续



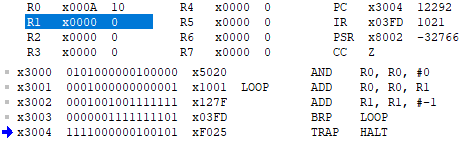
第四次循环，R0=R0+R1=10，R1=1



R1=R1-1=0，第四次循环经判断后PC指向TRAP，循环结束

循环结束后运行至TRAP，运行结束，此时R0=10，R1=0

四、运行结果



R0=10，R1=0

五、总结

这是我第一次使用LC3，因为对使用说明读得不够透彻而吃了很多苦头，但是最终还是摸索出了正确的使用方法，如下：

1. 打开LC3Edit.exe并编写代码
2. 编写代码时发现，LC3对代码的格式要求并不是很严格的，比如

AND R0 R0 0和 AND R0,R0,0都是正确的，每一个语句的后面都可以加;也可以不加。在开头必须加.ORIG x3000以表示起始地址，在结尾必须加TRAP x25和.END以让程序能够正常结束。

1. 关于LOOP循环，在循环的开始处加LOOP：以表示循环的开始，在循环

结束处用BR指令，使用标识符LOOP

1. 编写代码完成后，点Translate，选择Assemble，保存文件，它通常会生成多个格式的文件，如.bin , .hex , .lst , .obj , .sym等。
2. 打开Simulate.exe，点Load Program，选择刚刚生成的.obj文件，可以看到从地址x3000开始依次录入了刚才所写的代码，但是这里不建议点Run Program，会导致程序直接运行至结束，无法看到过程，这里选择Step Info，每点一次，运行一条指令，这样有利于观察每次程序都运行到了哪个指令，以及每个寄存器数值的变化，利于深入理解这段代码的运行过程。在运行前可以双击寄存器如R0，R1等，在Value一栏修改它的初值，如以上程序给寄存器R1赋初值4，就把Value改成x0004。
3. 在打开Simulate.exe之后会弹出两个窗口，另外一个时LC3 Console，这个窗口是用来给出提醒的，比如当这个程序已经运行完之后再点击运行，它就会提醒这段程序已经结束。

这段代码的功能是使R0完成从R1值到1的逐个递增，如上给出的R1为

4，则R0是完成从4到1的逐个递增，即最后R0的值为4+3+2+1=10，这段代码所实现的这个功能是由循环控制完成的

**实验二 选择控制程序**

一、实验目的

通过选择控制程序的编写、调试、运行，了解程序执行过程和实现机制，掌握其实现方法，提高程序编写技巧、调试技巧，提升对计算机系统的认识。

二、实验代码

.ORIG x3000

ADD R0,R0,0

LOOP BRz BREAK

BRp MINUS

ADD R0,R0,1

BRnzp LOOP

MINUS ADD R0,R0,-1

BRnzp LOOP

BREAK TRAP x25

.END

三、运行过程



程序无错误

.ORIG x3000 设初地址为x3000

AND R0,R0,#0 寄存器R0被清零

LOOP BRz BREAK 循环开始，判断R0是否为0，如果为0，跳转至指令BREAK

BRp MINUS 判断R0是否为正数，如果为正数，跳转至指令MINUS

ADD R0,R0,1 R0自增1

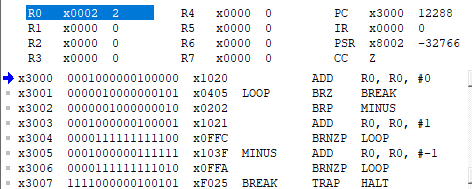
BRnzp LOOP 循环继续，返回循环开头LOOP继续执行指令

MINUS ADD R0,R0,-1 MINUS指令，使R0自增1

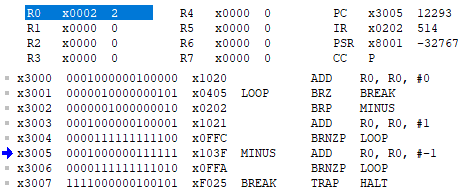
BRnzp LOOP 循环继续，返回循环开头LOOP继续执行指令

BREAK TRAP x25 BREAK指令，运行至TRAP

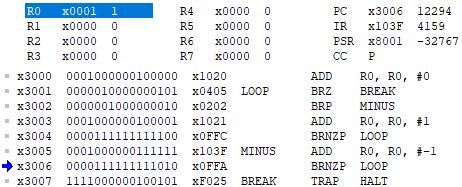
.END 运行结束



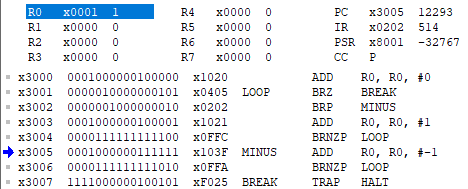
运行前，将R0的初值设为2



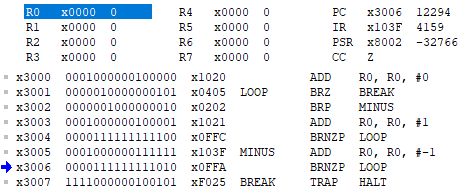
第一次循环，R0为正数2，BRP跳转至MINUS指令



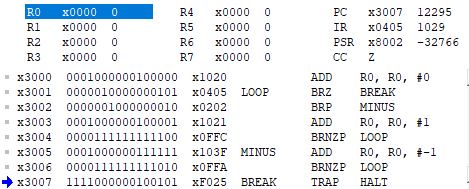
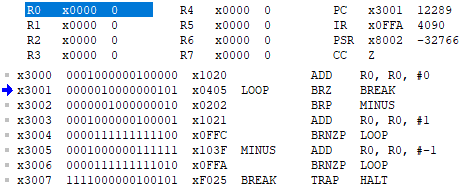
执行MINUS指令后，R0自减1，R0=1



BRNZP返回至循环开头，第二次循环，经判断，R0为正数1，BRP跳转至MINUS指令

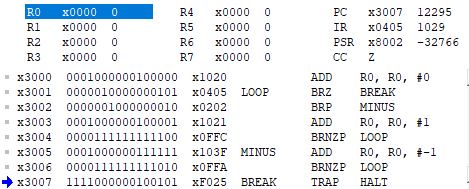


执行MINUS指令后，R0自减1，R0=0



第三次循环，R0=0，BRZ跳转至BREAK指令，运行至TRAP，程序结束

四、运行结果



R0=0

五、总结

这段代码所实现的功能是使寄存器的一个整数数值经过多次条件跳转指令的判断后多次自增自减，并最后使这个数值为0。循环中有条件跳转指令BRz和BRp，如果R0为0，则直接跳出循环并且运行结束，如果R0为正数，则会跳至MINUS指令进行自减1，并且跳回至循环开头继续循环，直到R0自减至0，如果R0为负数，则会自增1，并且跳回至循环开头继续循环，直到R0自增至0。这样一个选择性的跳转包含多种情况，如果满足跳转判断，则会直接跳至这个跳转所值的指令，如果不满足跳转判断，则会按照顺序执行下一条指令，BRnzp在此处的作用是让循环无论如何都回到循环的开头再进行一次循环。

**实验三 中断程序**

一、实验目的

通过中断程序的编写、调试、运行，了解程序执行过程和实现机制，掌握其实现方法，提高程序编写技巧、调试技巧，提升对计算机系统的认识。

二、实验代码

.ORIG x3000

LD R1,ASCII

LEA R0,STR1

TRAP x22

TRAP x23

ADD R2,R0,R1

LEA R0,STR2

TRAP x22

ADD R0,R2,x0

TRAP x21

TRAP x25

HALT

ASCII .FILL xFFE0

STR1 .STRINGZ "Input a word between a and z:"

STR2 .STRINGZ "The capital letter of this letter is:"

.END

三、运行过程



程序无错误

.ORIG x3000 设初地址为x3000

LD R1,ASCII PC相对寻址，ASCII码中大写字母值等于小写字母值-32

LEA R0,STR1 立即数寻址，这里是向屏幕写字符串STR1

TRAP x22 向屏幕写一个字符串，其起始地址由R0决定

TRAP x23 先打印提示在屏幕上，然后等待键盘输入一个字符(此处为小写字母)，这个字母的数值存入R0

ADD R2,R0,R1 这里R1=-32，R2=R0-32，这一步是把小写字母的数值转为大写字母

LEA R0,STR2 立即数寻址，这里是向屏幕写字符串STR2

TRAP x22 向屏幕写一个字符串，其起始地址由R0决定

ADD R0,R2,x0 将R2寄存器所保存的大写字母数值给到R0

TRAP x21 将R0的字符输出在屏幕上显示

TRAP x25 停止执行，并在屏幕上输出信息

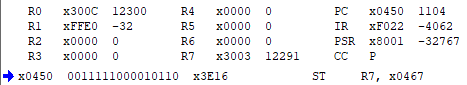
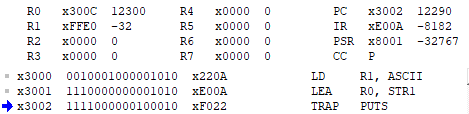
HALT

ASCII .FILL xFFE0

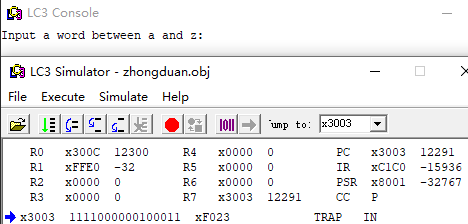
STR1 .STRINGZ "Input a word between a and z:"

STR2 .STRINGZ "The capital letter of this letter is:"

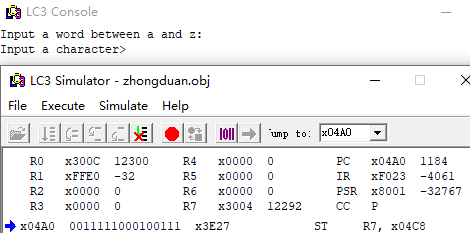
.END



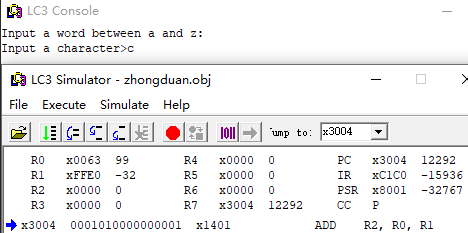
这里的x3002的下一步会跳到x0450，这是因为从x0450开始的这一段代码是用于在LC3 Console上输出字符串的，这里如果一直点Step Info将会看到字符串的每一个字符逐个输出，这里建议点Step Out，可以直接让其输出一整段字符串



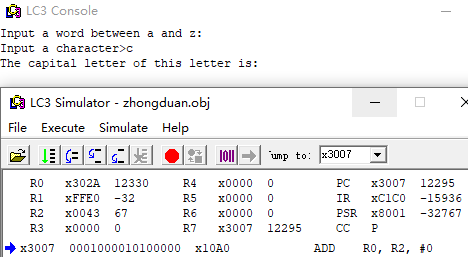
第一段字符串输出完成



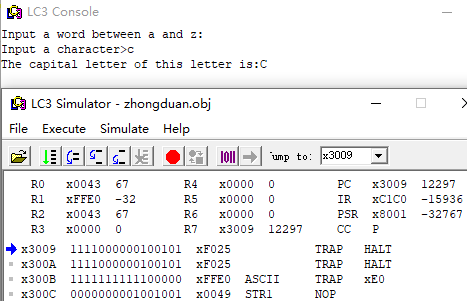
第二段字符串输出完成，此处需要在LC3 Console输入一个小写字母，这个小写字母的数值将会被存入寄存器R0



输入小写字母c之后，运行回到x3004



第三段字符串输出完成



输入小写字母对应的大写字母输入，程序运行结束

四、运行结果



五、总结

这段代码是用HALT来体现中断程序的功能的(课本P150)。在这段代码中，运用到了TRAP x21、x22、x23、x25等(课本P350).x21的功能是将寄存器R0的字符输出在屏幕上显示；x22的功能是向屏幕写出一个字符串，起始地址由R0指定，结束判断由当前内存单元是否为x0000确定；x23的功能是先将字符显示在屏幕上，然后等待键盘输入一个字符，这段代码实现的功能是将小写字母转换为大写字母，故这里输入的字符为想要转换的小写字母；x25的功能是停止执行，并在屏幕上输出信息。

这段代码首次用到了TRAP的打印功能，也就是运行途中跳到的由x0450开始的一大段代码，这一段代码的如果用Step Info，能够看到每一个LC3 Console每一个字符输出的全过程与所有代码，这有利于让我们更好地理解字符被输出到屏幕上的原理。当需要用户在屏幕上输入字符的时候，Stop Execution将会亮起，其他键将会变为不可用。

**实验四 子程序调用程序**

一、实验目的

通过子程序调用程序的编写、调试、运行，了解程序执行过程和实现机制，掌握其实现方法，提高程序编写技巧、调试技巧，提升对计算机系统的认识。

二、实验代码

.ORIG x3000

AND R1,R1,x0

AND R2,R2,x0

JSR PLUS

BR EXIT

PLUS AND R0,R0,x0

ADD R0,R1,R2

trap x21

MINUS AND R4,R4,x0

AND R5,R5,x0

ADD R4,R1,x0

ADD R5,R5,x0

ADD R4,R4,x-1

ADD R5,R5,x-1

BRp MINUS

AND R0,R0,x0

ADD R0,R4,x0

trap x21

EXIT HALT

.END

三、运行过程



程序无错误

.ORIG x3000 设初地址为x3000

AND R1,R1,#0 寄存器R0被清零

AND R2,R2,#0 寄存器R0被清零

JSR PLUS 子程序调用，跳转至PLUS指令

BR EXIT 调至EXIT结束程序

PLUS AND R0,R0,#0 寄存器R0被清零

ADD R0,R1,R2 计算R1与R2的和

trap x21 将R0的字符输出在屏幕上显示

MINUS AND R4,R4,#0 寄存器R4被清零

AND R5,R5,#0 寄存器R5被清零

ADD R4,R1,#0 寄存器R4存放R1的数值

ADD R4,R4,#-1 寄存器R4自减1

ADD R5,R5,#-1 寄存器R5自减1

BRp MINUS 当R5减至0时，减法运算结束

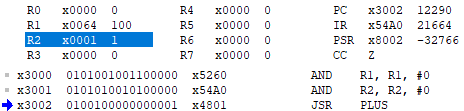
AND R0,R0,#0 寄存器R0被清零

ADD R0,R4,#0 寄存器R0存放R4的数值

trap x21 将R0的字符输出在屏幕上显示

EXIT HALT 程序结束

.END



寄存器R1和R2归零后，手动对R1和R2进行数值输入

随后指令PLUS执行加法运算，指令MINUS执行减法运算，这个过程可以点Run Program直接调至程序结束得出结果



四、运行结果

当R1=100，R2=1时



当R1=68，R2=1时

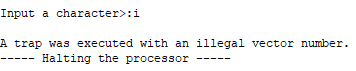


当R1=71，R2=5时



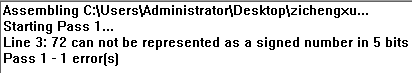
五、总结

关于子程序的调用，我最初尝试了课本P155图9-8的代码，随后出现了以下结果：



程序结果报错了，大意是执行的时候有非法的矢量数字，这说明书上的这个例子是无法正确运行得出结果的。

随后，我自己编写了简单的加减法运算，其实原本是有写乘除法运算的，但是基于LC-3汇编语言较为基础，在执行的时候容易出现大量问题，比如：



这里是给寄存器赋初值时数字过大。

除此之外，还有许多问题，输出的时候是以ASCII码的形式输出的，比如运算结果是101，输出的时候它就会输出101对应的ASCII码e；运算结果是99，输出的时候它就会输出99对应的ASCII码c等。这就会出现很多不便，ASCII码一共只有100多个，当运算结果超出这个范围时输出就会出错，即使在这个范围内，输出的结果有时候也是不便于观察的，因为ASCII码里面有大量的字符不是简单的一个字母或一个数字，有的是空字符，有的是换行键，当运行结果对应的字符是这些时，我们是无法观察到的。这仅仅是测试乘法运算时遇到的问题，除法还需要考虑更多的问题，比如运算结果不为整数，就会导致输出也是有问题的，另一方面，它也无法实现小数的除法运算，所以经过慎重的考虑后，为了保证代码的稳定性和可行性，我最终把这两种运算的代码去掉了。通过这个实验，我认识到LC-3语言是有很多不足的，它会造成代码功能实现上的很多困难和不便。

**实验五 递归程序**

一、实验目的

通过递归程序的编写、调试、运行，了解程序执行过程和实现机制，掌握其实现方法，提高程序编写技巧、调试技巧，提升对计算机系统的认识。

二、实验代码

.ORIG x3000

AND R0,R0,#0

AND R1,R1,#0

AND R2,R2,#0

AND R3,R3,#0

MOVE AND R4,R4,#0

ADD R4,R1,#0

AND R5,R2,#0

ADD R5,R2,#0

AND R6,R3,#0

ADD R6,R3,#0

JSR MULT1

MULT1 ADD R0,R0,R1

ADD R2,R2,#-1

BRp MULT1

BRz MULT2

MULT2 ADD R3,R3,#-1

BRz DIVISION1

ADD R0,R0,R0

BRp MULT2

DIVISION1 AND R1,R1,#0

AND R7,R7,#0

ADD R7,R0,#0

NOT R4,R4

ADD R4,R4,#1

LOOP1 ADD R1,R1,#1

ADD R7,R7,R4

BRp LOOP1

BRz DIVISION2

DIVISION2 AND R7,R7,#0

ADD R7,R0,#0

NOT R5,R5

ADD R5,R5,#1

LOOP2 ADD R2,R2,#1

ADD R7,R7,R5

BRp LOOP2

BRz DIVISION3

DIVISION3 AND R7,R7,#0

ADD R7,R0,#0

NOT R6,R6

ADD R6,R6,#1

LOOP3 ADD R3,R3,#1

ADD R7,R7,R6

BRp LOOP3

BRz EXIT

EXIT HALT

.END

三、运行过程



程序无错误

.ORIG x3000 设初地址为x3000

AND R0,R0,#0 寄存器R0被清零

AND R1,R1,#0 寄存器R1被清零

AND R2,R2,#0 寄存器R2被清零

AND R3,R3,#0 寄存器R3被清零

MOVE AND R4,R4,#0 这里的MOVE指令并不需要跳转，而是用于提醒用户在这一步即将运行时输入R1，R2，R3的数据

ADD R4,R1,#0 将寄存器R1的数据存入到寄存器R4中

AND R5,R2,#0 寄存器R5被清零

ADD R5,R2,#0 将寄存器R2的数据存入到寄存器R5中

AND R6,R3,#0 寄存器R6被清零

ADD R6,R3,#0 将寄存器R3的数据存入到寄存器R6中

JSR MULT1 执行MULT1指令

MULT1 ADD R0,R0,R1 MULT1指令的作用是将R1和R2相乘，结果存入R0

ADD R2,R2,#-1 寄存器R0每加一次R1，R2就自减1，这是乘法的原理

BRp MULT1 如果R2为正数，说明乘法运算未完成，继续执行MULT1

BRz MULT2 如果R2为0，说明乘法运算完成，执行MULT2

MULT2 ADD R3,R3,#-1 MULT2指令的作用是将R0和R3相乘，结果存入R0

BRz DIVISION1 如果R3为0，说明乘法运算完成，执行DIVISION1

ADD R0,R0,R0 R3不为0的情况下，说明乘法运算未完成

BRp MULT2 继续执行MULT2

DIVISION1 AND R1,R1,#0 DIVISION1指令的作用是将三个数的乘积R0除以R4，结果存入R1

AND R7,R7,#0 寄存器R7被清零

ADD R7,R0,#0 寄存器R7存入R0的数据，并进行除法运算

NOT R4,R4 寄存器R4取反

ADD R4,R4,#1 寄存器R4自增1，这样R4的值就是原来的值的取负了

LOOP1 ADD R1,R1,#1 LOOP1的作用是除法运算循环

ADD R7,R7,R4 寄存器R7的值减去R4

BRp LOOP1 如果R7为正数，说明除法运算未完成，继续执行LOOP1

BRz DIVISION2 如果R7为0，说明除法运算已完成，执行DIVISION2

DIVISION2 AND R7,R7,#0 DIVISION2指令的作用是将三个数的乘积R0除以R5，结果存入R2

ADD R7,R0,#0 以下运算过程类似DIVISION1

NOT R5,R5

ADD R5,R5,#1

LOOP2 ADD R2,R2,#1

ADD R7,R7,R5

BRp LOOP2

BRz DIVISION3

DIVISION3 AND R7,R7,#0 DIVISION3指令的作用是将三个数的乘积R0除以R6，结果存入R3

ADD R7,R0,#0 以下运算过程类似DIVISION1

NOT R6,R6

ADD R6,R6,#1

LOOP3 ADD R3,R3,#1

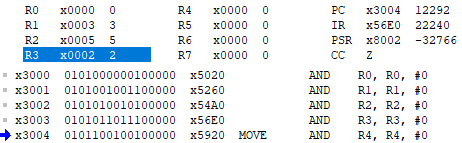
ADD R7,R7,R6

BRp LOOP3

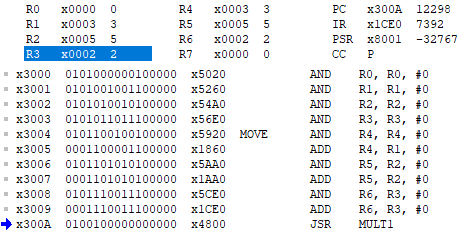
BRz EXIT 当R7为0时，程序的所有运算完成，结束程序

EXIT HALT

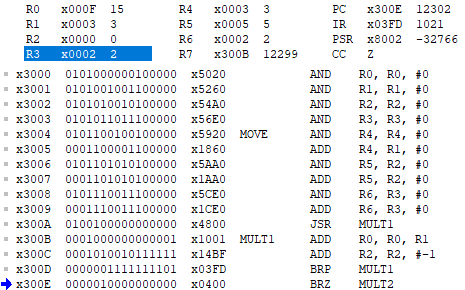
.END



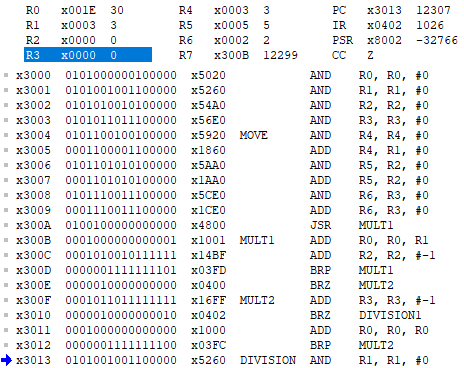
在MOVE进行前这一步输入R1，R2，R3的值，这里取R1=3,R2=5,R3=2



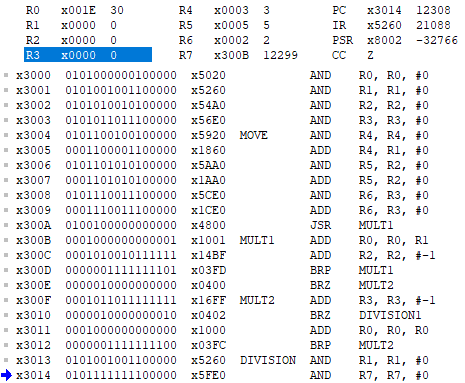
MOVE指令的作用是将R1、R2、R3的值分别存入R4、R5、R6，后面需要用到这三个寄存器



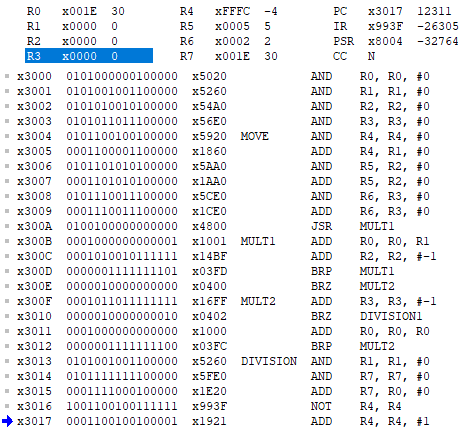
MULT1运算完成，MULT1指令的作用是让R1和R2相乘，结果存入R0



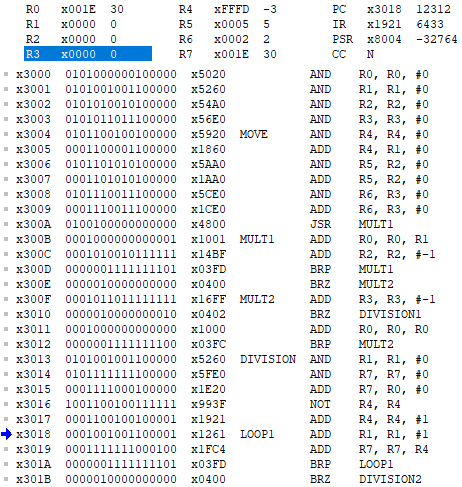
MULT2运算完成，MULT2指令的作用是让R1与R2相乘所得结果R0与R3相乘，结果存入R0，现在的R0是原R1、R2、R3三个寄存器数值的乘积。前面的乘法运算中R1是没有清零的，所以DIVISION1的第一步就是让R1清零



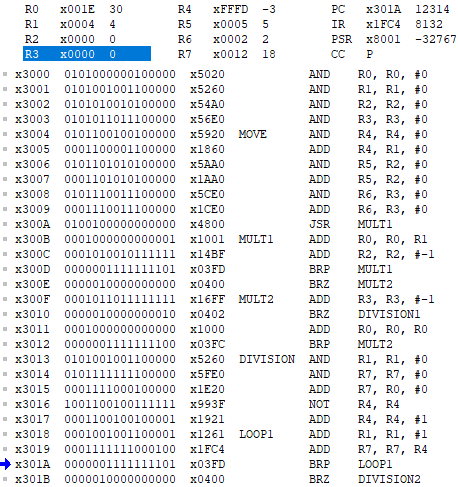
这里出现了一个现象，虽然前面的运算中没有用到R7，但是在这里，R7的数值是不为0的，所以DIVISION1的第二步是让R7清零，这一步至关重要，如果缺少这一步将导致后面的所有除法运算全部失败。



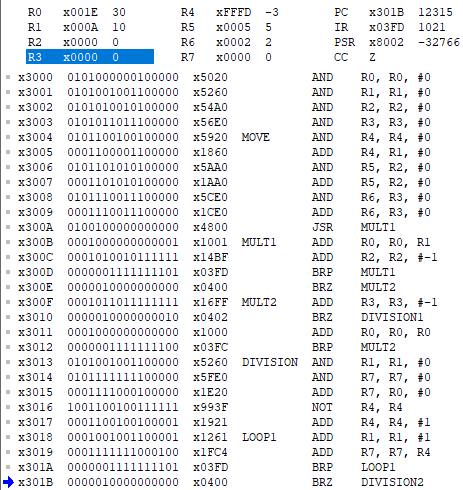
这里可以发现，NOT指令不是直接让数值取负，而是它对应负值-1，这符合十六进制运算



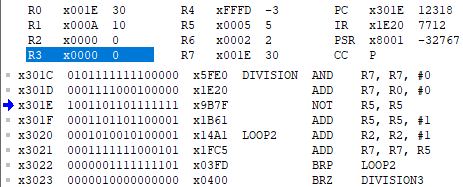
让R4自增1之后，R4才是它原来数值的取负，随后LOOP1指令的循环是让R7对原R1（3）做除法运算



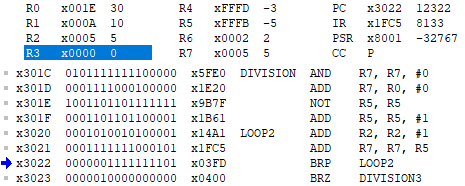
这是LOOP1执行的过程，当R7减去4次R4之后，R1的数值为4



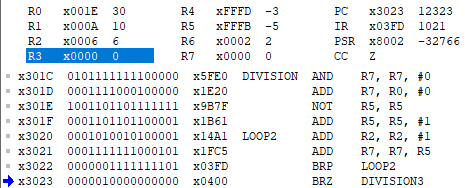
当LOOP1完成对应的除法运算时，R7为0，而R1就是30/3，也就是10



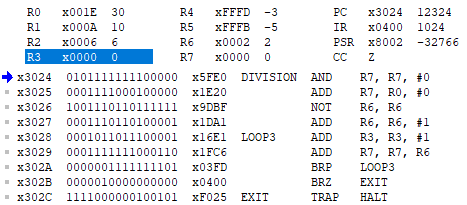
LOOP2执行除法运算前，DIVISION2也需要执行类似DIVISION1的步骤，即给R7赋值，以及需要的取负



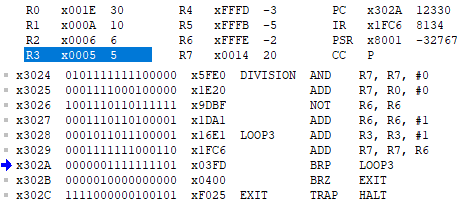
这是LOOP2执行的过程，当R7减去5次R5之后，R1的数值为5



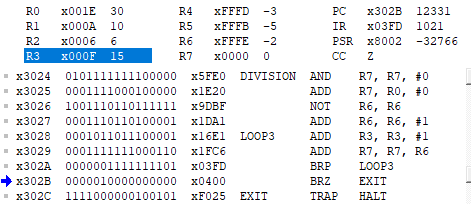
当LOOP2完成对应的除法运算时，R7为0，而R2就是30/5，也就是6



LOOP3执行除法运算前，DIVISION3也需要执行类似DIVISION1的步骤，即给R7赋值，以及需要的取负

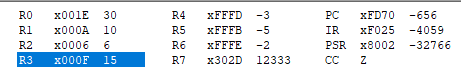


这是LOOP3执行的过程，当R7减去5次R5之后，R1的数值为20



当LOOP2完成对应的除法运算时，R7为0，而R3就是30/2，也就是15

四、运行结果



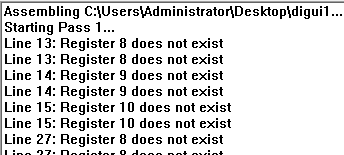
R0=30，R1=10，R2=6，R3=15

五、总结

在实验四中提到，四则运算中的乘法和除法的实现并没有那么简单，在我优化了先前的代码以及简化了思路之后，做出了实验五这样一个程序。这个程序的功能是求出3个数字的乘积，并且求出这个乘积分别除以这3个数字所得到的商。因为我在这个实验中省去了输出的步骤，直接通过修改寄存器的数值和观察各寄存器的计算，就不需要考虑寄存器结果超过ASCII数值的情况。另一方面，对于除法可能出现无法整除的情况，这段代码能够在实现除法运算的同时不让无法整除而导致程序错误的情况出现，因为3个数的乘积是必定能够被其中任意一个数整除的。

递归与循环类似，但是又有所不同。这个程序中有许许多多的指令名，通过递归多次调用自身能够让不同的数值和不同的运算有条不紊地执行，每一次递归的数值都是不同的。递归的难点在于，栈的先进后出这一原则必须得到保证，必须弄清楚程序中不同代码的顺序，才不会导致栈里数据的混乱。

在我完成这段代码前，我的思路还不够简洁，用到了太多的寄存器，后来报错时我才想起来，LC-3是只有R0到R7一共8个寄存器的，如下图：



当我们需要更复杂的运算、对大量的数据进行统计计算时，可能会出现8个寄存器不够用的情况，这是LC-3的其中一个缺点。当我思路更简洁并且简化了代码之后，3个数字的乘法与除法的实现可以控制在8个寄存器内完成。

在代码编辑界面，指令名是DIVISION1、DIVISION2、DIVISION3，但是在运行界面统一显示成了DIVISION，这是因为运行界面指令名的显示只有8个字符，但是这并不影响程序的正常运行。

在DIVISION1开始运行时，出现了这样一个现象，R7在未被指令执行的情况下数值变为了非0，通过向老师的请教得知，JSR指令在执行时会用到R7保存PC的值，而后面我把他归零之后再拿R7来进行运算，这个是没有问题的，但是如果同时有许多JSR指令又要用寄存器R7进行各种数值运算的话，就容易出现问题。