**深 圳 大 学 实 验 报 告**

|  |
| --- |
| **课程名称 计算机系统1**  **项目名称 Nim游戏**  **学 院 计算机与软件学院**  **专 业 软件工程（腾班）**  **指导教师 陈飞**  **报 告 人 叶茂林 学号 2021155015**  **实验时间 2022.5.27**  **提交时间 2022.5.29** |

**教务处制**

# 一、实验目的与要求

Nim是一个简单的双人游戏，可能起源于中国。游戏中使用的计数器类型有很多种类，如石头、火柴、苹果等。游戏界面被划分为很多行，每行中有数量不等的计数器，如图1所示：

|  |  |
| --- | --- |
| 行号 | 计数器数量 |
| 1 | ○○○ |
| 2 | ○○○○○○ |
| …… | …… |
| n | ○○○○○○○○○○ |

图1 游戏界面

本次实验对Nim游戏做了一些小的改变，具体如下：游戏界面由三行组成，计数器类型为石头，其中A行包含3个石头，B行包含5个石头，C行包含8个石头。

规则如下：

⑴ 每个玩家轮流从某一行中移除一个或多个石头。

⑵ 一个玩家不能在一个回合中从多个行中移除石头。

⑶ 当某个玩家从游戏界面上移除最后剩余的石头时，此时游戏结束，该玩家获胜。

⑴ 在游戏开始时，你应该显示游戏界面的初始化状态。具体包括：在每行石头的前面，你应该先输出行的名称，例如“ROW A”。你应该使用ASCII字符小写字母“o”（ASCII码 x006F）来表示石头。游戏界面的初始化状态应该如下：

*ROW A: ooo*

*ROW B: ooooo*

*ROW C: oooooooo*

⑵ 游戏总是从玩家1先开始，之后玩家1和玩家2轮流进行。在每一个回合开始时，你应该输出轮到哪一个玩家开始，并提示玩家进行操作。例如，对于玩家1，应该有如下显示：

*Player 1,choose a row and number of rocks:*

⑶ 为了指定要移除哪一行中的多少石头，玩家应该输入一个字母后跟一个数字（输入结束后不需要按Enter键），其中字母（A,B或C）指定行，数字（从1到所选行中石头的数量）指定要移除的石头的数量。你的程序必须要确保玩家从有效的行中移除有效数量的石头，如果玩家输入无效，你应该输出错误提示信息并提示该玩家再次进行输入。例如，如果轮到玩家1：

*Player 1, choose a row and number of rocks: D4*

*Invalid move. Try again.*

*Player 1, choose a row and number of rocks: A9*

*Invalid move. Try again.*

*Player 1, choose a row and number of rocks: A\**

*Invalid move. Try again.*

*Player 1, choose a row and number of rocks: &4*

*Invalid move. Try again.*

*Player 1, choose a row and number of rocks:*

你的程序应保持提示玩家，直到玩家选择有效的输入为止。确保你的程序能够回显玩家的输入到屏幕上，当回显玩家的输入后，此时应该输出一个换行符（ASCII码x000A）使光标指向下一行。

⑷ 玩家选择有效的输入后，你应该检查获胜者。如果有一个玩家获胜，你应该显示相应的输出来表明该玩家获胜。如果没有胜利者，你的程序应该更新游戏界面中每行石头的数量，重新显示更新的游戏界面，并轮到下一个玩家继续。

⑸ 当某个玩家从游戏界面上移除最后的石头时，游戏结束。此时，你的程序应该显示获胜者然后停止。例如，如果玩家2移除了最后的石头，你的程序应该输出一下内容：

*Player 2 Wins.*

# 二、实验内容与方法

⑴ 记住，程序中所有的输入输出使用ASCII字符，你应该负责进行必要的转换。

⑵ 从键盘中输入字符你应该使用TRAP x20（GETC）指令，同时为了回显输入的字符到屏幕上，你应该使用TRAP x21（OUT）指令，该指令紧跟在TRAP x20指令之后。

⑶ 你应该在适当的时候使用子程序。

⑷ 在你编写的每个子程序中，应该保存并还原所使用的任何寄存器。这将避免你在调试过程中遇到问题。

⑸ 在一个回合中，玩家的输入必须包含指定为A,B或C（即大写字母）的行，后面紧跟不大于该行仍然存在的石头数量的数字。

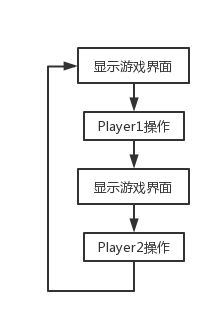
① 你应该设置程序的开始地址在x3000（如,程序的第一行指令应该为 .ORIG x3000）

② 源文件命名为nim.asm

# 三、实验步骤与过程

（依照实验内容，逐条撰写实验过程与实验所得结果：包括程序总体设计，核心数据结构及算法流程，调试过程。请附上核心代码，及注意格式排版的美观。实验提交时，以上为评分依据，请不删除本行）

**程序总体设计**



**核心数据结构**

1. 显示游戏界面

如图1所示，首先将寄存器的值存进内存，待子函数完成任务后再将该内存的值存进寄存器，用伪操作.stringz开辟内存用来存储字符串，将Row A、Row B和Row C的石头数目也存储在内存中。

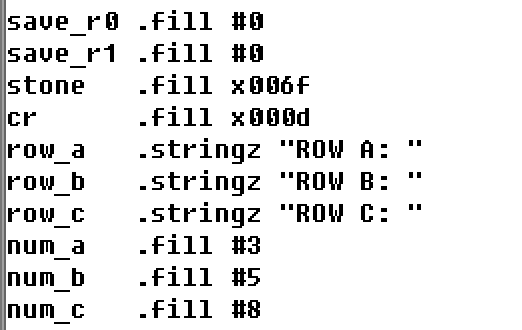


图1

先用LEA指令将字符串的首地址存进R0，然后通过PUTS输出，用LD指令将字符o的ascll码存进R0，然后用LD指令将石头的数目存进R1，R1作为计数器，用OUT循环输出字符o，最后用LD指令将换行符的ascll码存进R0，用OUT输出。

2、用户操作

(1)输出提示

用伪操作.stringz将提示字符串存进内存中，先将用到的寄存器R0和R7的值存进内存保存起来，然后用LEA指令将字符串的首地址存进R0，用PUTS输出提示，然后将R0和R7的值恢复。

(2)用户输入

用GETC读取输入的第一个数据，然后用OUT回显，ADD指令将R0的数据转入R2，然后用NOT将R2取反，ADD将R2加一，即将R2取负，再用GETC读取输入的第二个数据，OUT回显，ADD指令将R0的数据转入R3。

(3)判断数据是否有效

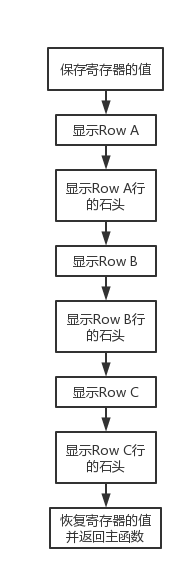
用伪操作.fill将字符A、B和C的ascll码存进内存，用LD指令将相应字符的ascll码存进R0，然后ADD指令将R0和R2相加的结果存在R0，通过判断R0是否为0来判断是A、B、C或无效输入。

(4)取石头

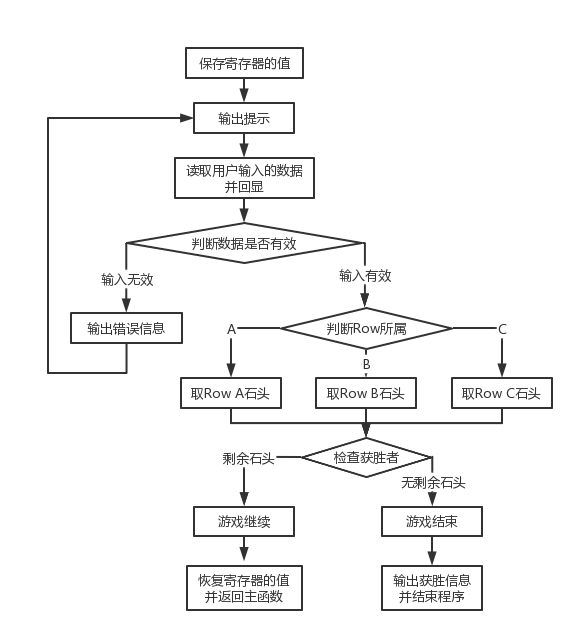
用LD指令将字符0的ascll码存进R0，然后将R0取负，与R3相加的结果存放到R0中，然后用LD指令将石头的数目存进R3，将R0取负，与R3相加的结果存进R3，最后将R3的值存进内存。

**算法流程**

1. 显示游戏界面



1. Player操作



**调试过程**

打完代码后，第一次运行，发现没有输出第一个用户的提示信息，如图2所示：

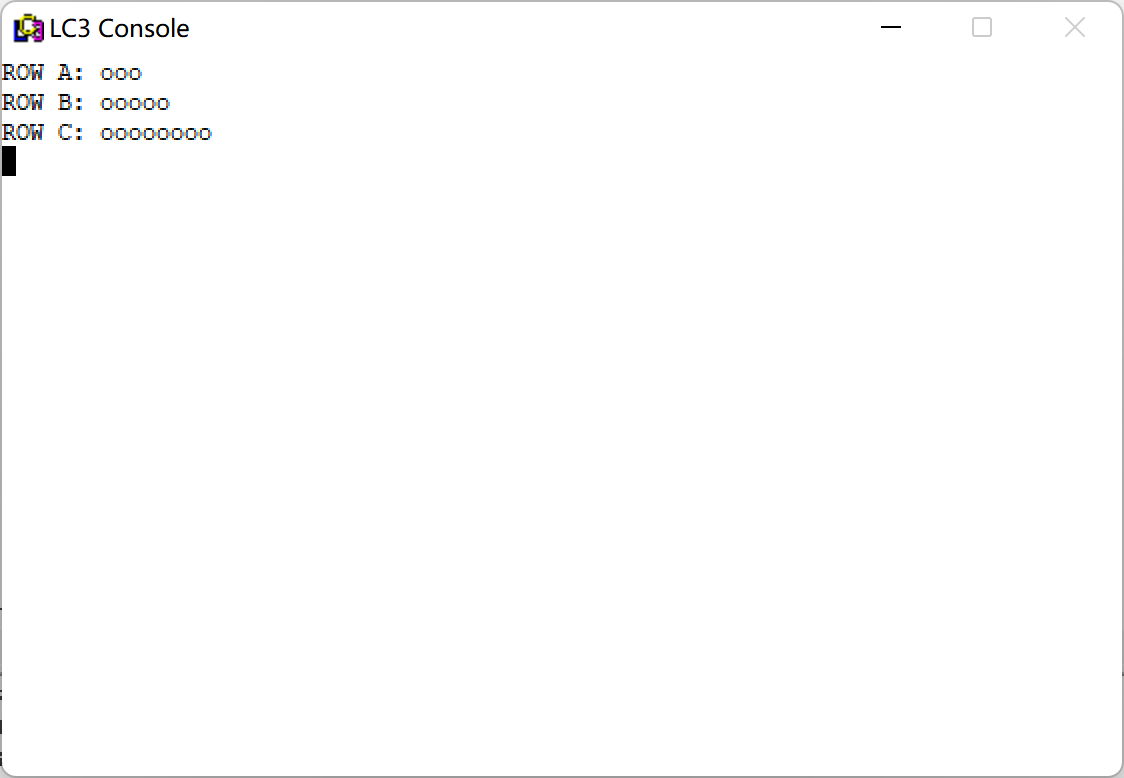


图2

然后回到程序调试，发现R7的值保存的是OUT指令的下一条指令地址，考虑到调用trap服务程序会改变R7的值，使得原本保存返回调用代码的链接地址的R7被修改，即子程序无法跳回调用程序，因此，需要保存R7的值。

再次运行，发现程序异常结束，如图3所示：

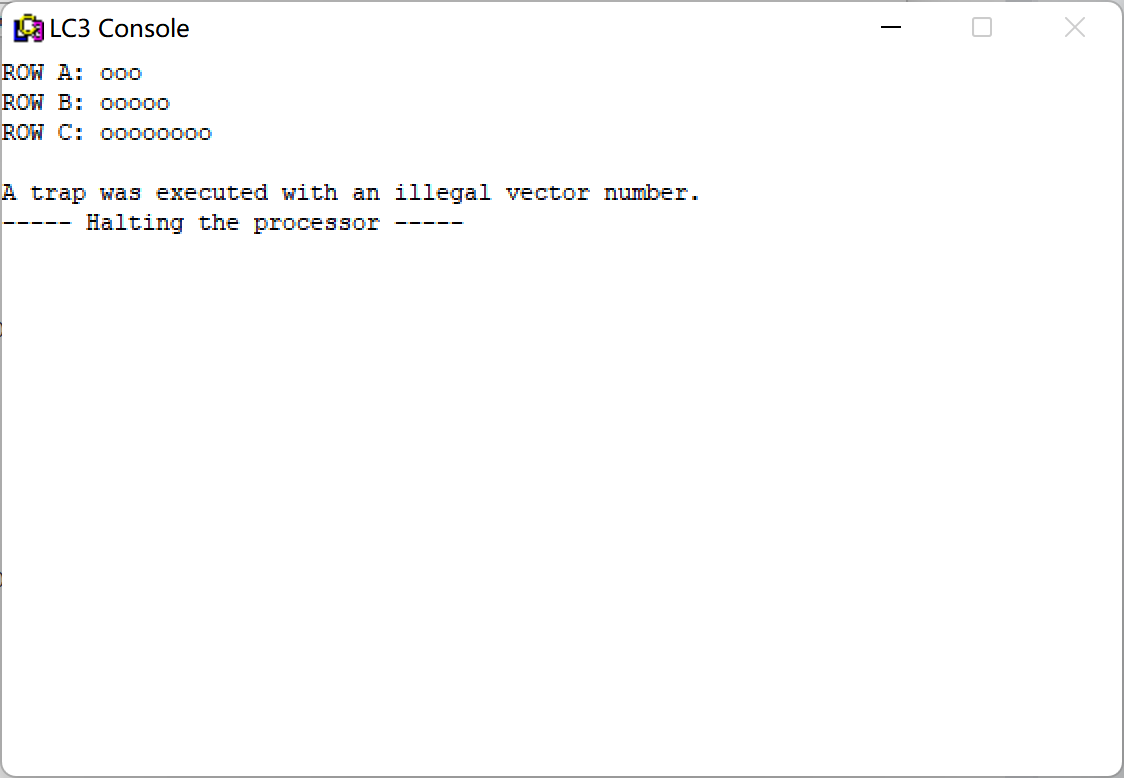


图3

然后重新调试，发现把子程序写在了程序开头，而主函数写在了程序末尾，于是重新把主函数移到程序开头。

再次运行，发现当输入无效数据时，没有输出错误信息，如图4所示：

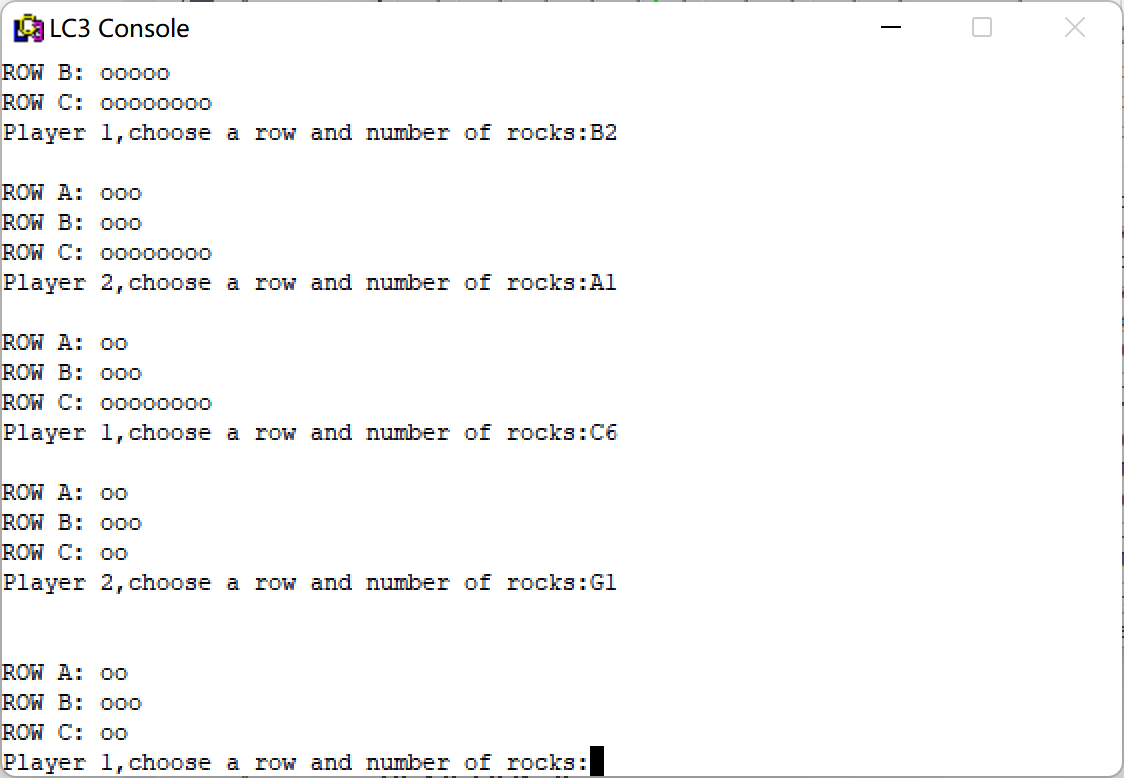


图4

找到相应代码，发现忘记写输出PUTS，如图5所示：

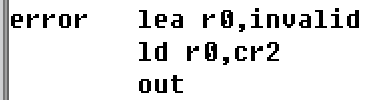


图5

再次运行，发现，输出错误信息时多换了一次行，如图6所示:

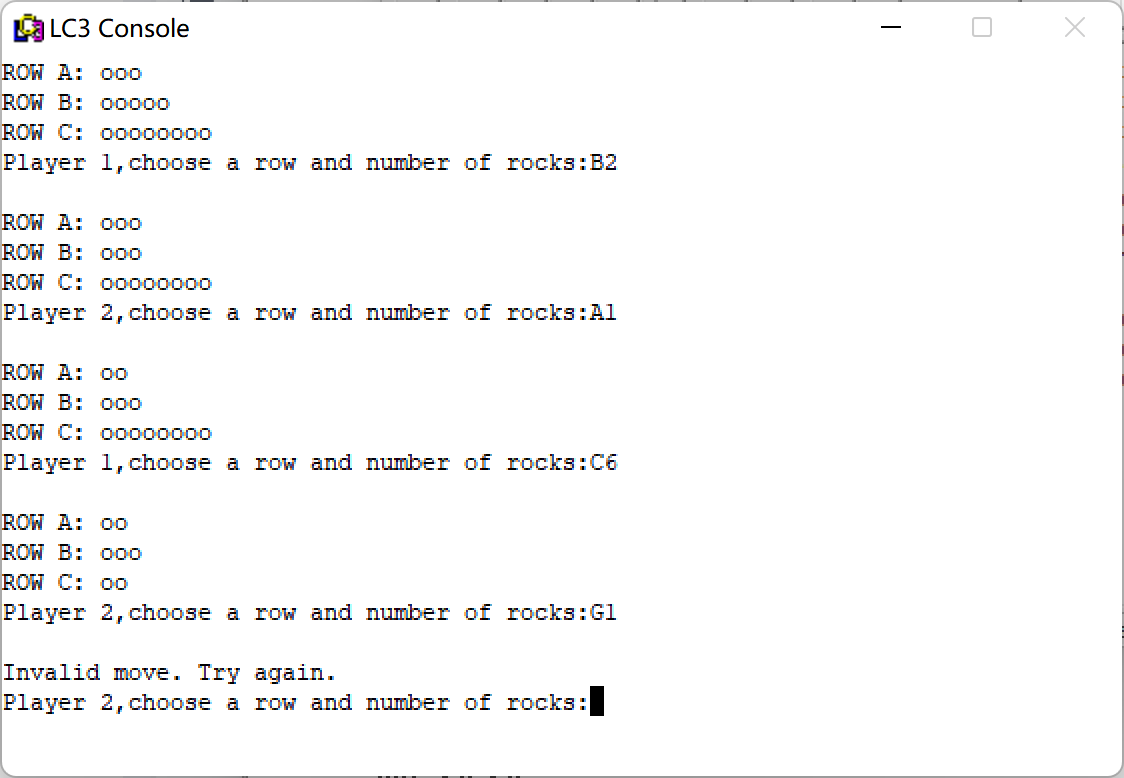


图6

于是把输出换行设置为当转到下一个用户输入时才进行。

再次运行，得出正确结果，如图7所示：

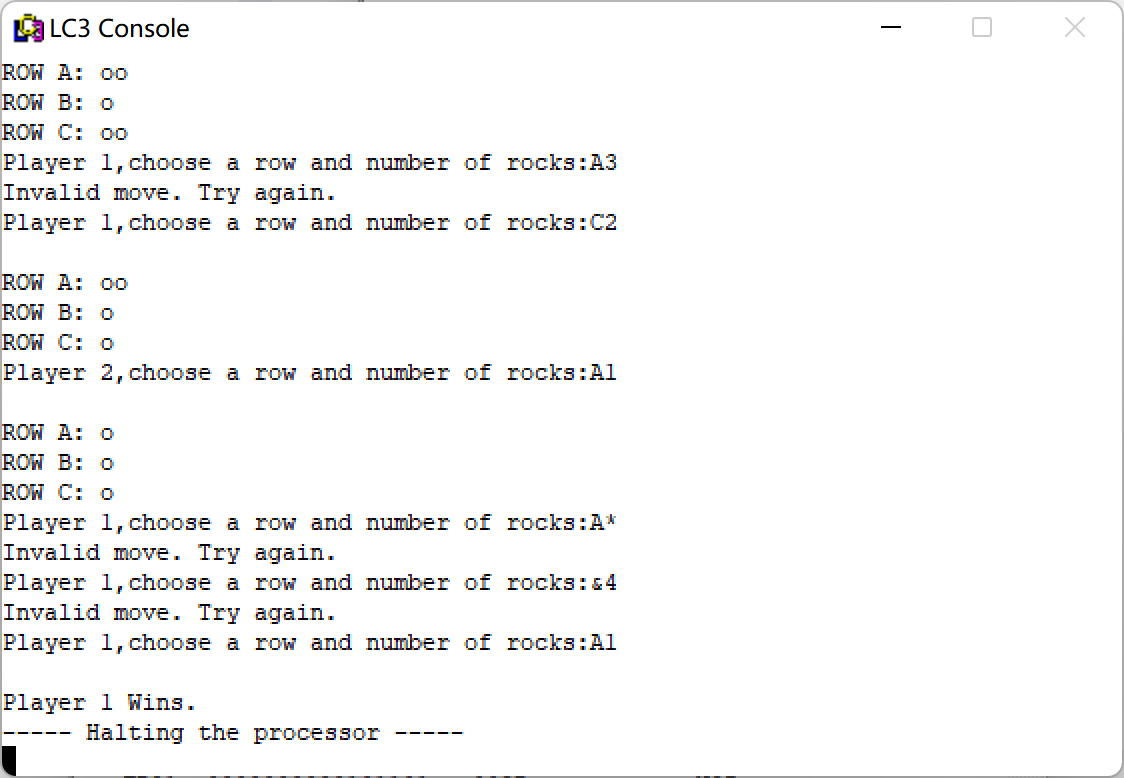


图7

# 四、实验结论或体会

（撰写实验收获及思考）

这次实验是用LC-3汇编语言写一个简化版的Nim游戏，一下子看起来这个游戏实现还是有点困难的，但是我们用模块化的思想去思考的时候，这个问题就会变得简单，我们只需要把这个游戏过程分成几个小部分，然后相对独立地分别去实现每一个小模块就可以了。

这次实验涉及到了子程序的问题，在子程序中，我们通常会先把用到的寄存器的保存起来，这里会出现一个问题，就是我们有时候会不知道自己用到了哪些寄存器，就像我这次实验中的子程序print，我显性地只使用了R0寄存器，但是我还在子程序里面调用了trap服务程序，所以导致R7的值被改变，因此还需要事先保存R7的值。

本次实验还发现一个问题，即我在子程序里面调用自己写的子程序的时候，我们在子程序里面用来保存寄存器的值的内存不能相同，否则会覆盖之前的数据存储。

在这次实验中，我们更多了使用内存来存储数据，而不是像之前一样只使用寄存器，在这里，寄存器更多地像一个暂时的存放地，而这也的确是寄存器的作用，这在大数据和大程序里尤其明显与重要。

从汇编语言到高级语言，确实需要付出很多。

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后10日内。

**附录**

程序源代码

.orig x3000

again jsr print

jsr datain1

jsr print

jsr datain2

br again

print st r0,save\_r0

st r1,save\_r1

st r7,save\_r7

lea r0,row\_a

puts

ld r0,stone

ld r1,num\_a

loop\_a out

add r1,r1,#-1

brp loop\_a

ld r0,cr

out

lea r0,row\_b

puts

ld r0,stone

ld r1,num\_b

loop\_b out

add r1,r1,#-1

brp loop\_b

ld r0,cr

out

lea r0,row\_c

puts

ld r0,stone

ld r1,num\_c

loop\_c out

add r1,r1,#-1

brp loop\_c

ld r0,cr

out

ld r0,save\_r0

ld r1,save\_r1

ld r7,save\_r7

ret

save\_r0 .fill #0

save\_r1 .fill #0

stone .fill x006f

cr .fill x000d

row\_a .stringz "ROW A: "

row\_b .stringz "ROW B: "

row\_c .stringz "ROW C: "

num\_a .fill #3

num\_b .fill #5

num\_c .fill #8

cue1 st r0,save\_r0

st r7,save\_r7

lea r0,play1

puts

ld r0,save\_r0

ld r7,save\_r7

ret

play1 .stringz "Player 1,choose a row and number of rocks:"

cue2 st r0,save\_r0

st r7,save\_r7

lea r0,play2

puts

ld r0,save\_r0

ld r7,save\_r7

ret

play2 .stringz "Player 2,choose a row and number of rocks:"

save\_r7 .fill #0

datain1 st r0,save\_r0

st r2,save\_r2

st r3,save\_r3

st r7,saver7

try1 jsr cue1

getc

out

add r2,r0,#0

not r2,r2

add r2,r2,#1

getc

out

add r3,r0,#0

ld r0,lf

out

test1a ld r0,char\_a

add r0,r2,r0

brnp test1b

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error1

ld r3,num\_a

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error1

st r3,num\_a

ld r3,sum\_abc

add r3,r3,r0

brz win1

st r3,sum\_abc

br save

test1b ld r0,char\_b

add r0,r2,r0

brnp test1c

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error1

ld r3,num\_b

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error1

st r3,num\_b

ld r3,sum\_abc

add r3,r3,r0

brz win1

st r3,sum\_abc

br save

test1c ld r0,char\_c

add r0,r2,r0

brnp error1

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error1

ld r3,num\_c

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error1

st r3,num\_c

ld r3,sum\_abc

add r3,r3,r0

brz win1

st r3,sum\_abc

br save

win1 ld r0,lf

out

lea r0,wins1

puts

halt

error1 lea r0,invalid

puts

ld r0,lf

out

br try1

datain2 st r0,save\_r0

st r2,save\_r2

st r3,save\_r3

st r7,saver7

try2 jsr cue2

getc

out

add r2,r0,#0

not r2,r2

add r2,r2,#1

getc

out

add r3,r0,#0

ld r0,lf

out

test2a ld r0,char\_a

add r0,r2,r0

brnp test2b

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error2

ld r3,num\_a

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error2

st r3,num\_a

ld r3,sum\_abc

add r3,r3,r0

brz win2

st r3,sum\_abc

br save

test2b ld r0,char\_b

add r0,r2,r0

brnp test2c

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error2

ld r3,num\_b

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error2

st r3,num\_b

ld r3,sum\_abc

add r3,r3,r0

brz win2

st r3,sum\_abc

br save

test2c ld r0,char\_c

add r0,r2,r0

brnp error2

ld r0,char\_0

not r0,r0

add r0,r0,#1

add r0,r0,r3

brn error2

ld r3,num\_c

not r0,r0

add r0,r0,#1

add r3,r0,r3

brn error2

st r3,num\_c

ld r3,sum\_abc

add r3,r3,r0

brz win2

st r3,sum\_abc

br save

win2 ld r0,lf

out

lea r0,wins2

puts

halt

error2 lea r0,invalid

puts

ld r0,lf

out

br try2

save ld r0,lf

out

ld r0,saver0

ld r2,save\_r2

ld r3,save\_r3

ld r7,saver7

ret

lf .fill x000a

char\_a .fill x0041

char\_b .fill x0042

char\_c .fill x0043

char\_0 .fill x0030

wins1 .stringz "Player 1 Wins."

wins2 .stringz "Player 2 Wins."

invalid .stringz "Invalid move. Try again."

sum\_abc .fill #16

saver0 .fill #0

save\_r2 .fill #0

save\_r3 .fill #0

saver7 .fill #0

.end