深圳大学实验报告

课程名称 : 计算机系统(1)

实验名称 : 简单游戏设计

学院 : 数学与统计学院

指导老师 : 俞航

报告人 : 王曦　　组号 : 无

学号 : 2021192010　　实验地点 : LC-3模拟器

实验时间 : 2022年05月26日

提交时间 : 2022年05月26日

1. 实验目的

四子棋是一款普遍流行的简易型桌面游戏，据说，虎克船长曾因专注于此游戏而长期隐身在住所，当船员们发现船长的这一专长之后，他们称这个游戏为“船长的情妇”。

四子棋是个双人游戏，两人轮流下棋，棋盘由行和列组成的网格，每个选手每次下一个子直到两人中有一人的棋子连成一条水平线、垂直线或者是对角线。

本实验需要在LC-3中实现简易版四子棋的游戏，两位选手通过键盘和输出窗口轮流交互操作，棋盘由6 X 6的网格组成。

游戏规则如下：

两位选手依次轮流落子；

选手不能悔棋；

有子的地方不能继续落子；

直到有一方的四个棋子能够连成一条水平线、垂直线或者是对角线；

如果棋盘已满，无人获胜，则平局。

游戏最初时应该打印空的棋盘，可以用ASCII码"-" （即ASCII 码 x002D）来表示该处为空，"O"(ASCII 码 x004F)表示第一位选手的棋子，"X" (ASCII 码 x0058)来表示第二位选手的棋子，为了让棋盘更易于观察，在各列间加一个空格，第6列之后不要添加，初始棋盘应该如下：

```
- - - - - -

- - - - - -

- - - - - -

- - - - - -

- - - - - -

- - - - - -
```

选手一始终先下第一步棋，然后两者轮流落子，在每次落子之后，应该打印该选手的信息，提示他落子，以选手一为例，应该打印信息如下：

```
Player 1, choose a column:
```

为了明确选手的落子的位置，该选手应该输入数字1-6，然后回车，数字1-6指示在落子所在的列，从左到右，无需输入行号，程序应默认从行号6到行号1递减的顺序填入该棋子，若前后输入的列号相同，则行号减一。例如，如果选手第一次在左起第二列落子，应该输入2，然后回车，则该棋子落在行6列2处，当后面输入的列号再次为2时，则将棋子落子行5列2处，以此类推，详情见后续示例输出。程序应该确保选手输入的数字对应正确的列的范围，如果输入不合理，应该输出一条错误信息，提示该选手继续输入，例如，如果对于选手一：

```
Player 1, choose a column: D

Invalid move. Try again.

Player 1, choose a column: 7

Invalid move. Try again.

Player 1, choose a column:
```

程序应该一直提示该选手，知道输入正确的数字，当用户输入完成，程序应通过显示回馈给选手，然后通过换行符(ASCII 码 x000A)换行。

当选手输入成功后，程序应打印更新后的棋盘，并检查是否有人获胜，如果没人获胜，则轮到下一位输入。

当其中一位获胜或平局时，游戏结束，程序显示最后的棋盘情况并终止（Halt）。例如，如果选手二有四子相连，应该输出：

Player 2 Wins.

如果平局，程序应该输出：

Tie Game.

---

2. 实验内容

2.1 整体思路

开一个36个字的连续空间表示棋盘,用0表示无棋子,1表示选手一的棋子,-1表示选手二的棋子,这样可直接通过正负来判断棋子的类型.

开一个计数器$TIMES$,初始为1,用于判断轮到哪位选手下棋以及是否平局.偶数时轮到玩家一下棋,奇数时轮到玩家二下棋.$TIMES > 36$时平局.

子程序DISPLAY用于打印棋盘.

子程序INPUT从键盘输入一个字符并判断其是否介于$1 \sim 6$间,若是则R_0返回对应位置,否则R_0返回0并输出提示信息.

子程序PLAY判断R_0对应的列是否还有位置,若有位置则根据棋子是哪一方的更新键盘.

子程序JUDGE_W检查当前是否有玩家获胜,若有则R_0返回1,否则R_0返回0.

子程序ANSWER输出获胜信息,根据$TIMES$的奇偶判断哪一方获胜.

2.2 Labels

```
;Labels
TIE .STRINGZ "Tie Game."
INVALID .STRINGZ "Invalid move. Try again. "
PROMPT1 .STRINGZ "Player 1,  choose a column: "
PROMPT2 .STRINGZ "Player 2,  choose a column: "
WIN1  .STRINGZ "Player 1 Wins."
WIN2  .STRINGZ "Player 2 Wins."

CHESS_O .FILL X004F
CHESS_X .FILL X0058
CHESS_N .FILL X002D

TIMES .FILL #1
BOARD .BLKW 36 #0

SAVE0 .FILL   0
SAVE1 .FILL   0
SAVE2 .FILL   0
SAVE3 .FILL   0
SAVE4 .FILL   0
SAVE5 .FILL   0
SAVE6 .FILL   0
SAVE7 .FILL   0
```

```
SAVE77  .FILL 0

NUM_ROW .FILL #6
NUM_END .FILL #-36
NUM_CH  .FILL   #-48

SPACE .FILL X0020
ENDL  .FILL X000A
```

2.2 子程序DISPLAY

先备份需要用到的寄存器原来的值(下略),再将棋盘首地址BORAD load到$R6$中.以R_1、R_2分别为行、列指针,R_3用于接收地址R_6中存放的值,遍历棋盘,打印空位或对应的棋子,每打印一个棋子后R_6++,当R_6非正时停止输出空格并输出换行,否则输出空格.棋盘打印结束后,将寄存器备份的值放回原寄存器(下略).

```
;print the board
DISPLAY ST R1,  SAVE1  ;backup
  ST R2,  SAVE2
  ST R3,  SAVE3
  ST R6,  SAVE6

  LEA R6,  BOARD  ;calculate the offset
  LD R2,  NUM_ROW
COL_L LD R1,  NUM_ROW
ROW_L LDR R3,  R6,  #0  ;traverse the board
  BRnp TYPE_L1  ;get the type of the chess
  LD R0, CHESS_N  ;no chess
  BRnzp TYPR_LD

TYPE_L1 BRp TYPE_L2
  LD R0, CHESS_X  ;X chess
  BRnzp TYPR_LD
TYPE_L2 LD R0, CHESS_O  ;O chess
TYPR_LD ST R7, SAVE7  ;backup
  TRAP X21  ;print the chess
  LD R7, SAVE7
  ADD R6, R6, #1  ;board address++
  ADD R1, R1, #-1
  BRNZ SPACE_L
  LD R0, SPACE
  ST R7, SAVE7
  TRAP X21  ;print the space
  LD R7, SAVE7
  ADD R1, R1, #0
SPACE_L BRp ROW_L
  LD R0, ENDL
  ST R7, SAVE7
  TRAP X21  ;print endl
  LD R7, SAVE7
  ADD R2, R2, #-1
  BRp COL_L
  LD R1, SAVE1
  LD R2, SAVE2
  LD R3, SAVE3
```

```
  LD R6, SAVE6


  RET
```

2.3 子程序PROMPT

用于输出等待用户输入时的、输入非法时的提示信息.根据计数器$TIMES$奇偶判断当前轮到哪一方输入.

```
;print prompt info
PROMPT ST R1, SAVE1  ;backup
  ST R2, SAVE2
  ST R0, SAVE0

  ADD R0, R0, #0
  BRp PRO_L3
  LEA R0 INVALID  ;invalid input
  ST R7 SAVE7
  TRAP X22
  LD R0 ENDL
  TRAP X21
  LD R7 SAVE7

PRO_L3  ST R0, SAVE0
  LD R0, TIMES
  ST R7, SAVE7
  JSR JUDGE_N
  ADD R0, R0, #0
  BRZ PRO_L1
  LEA R0, PROMPT1
  BRnzp PRO_L2
PRO_L1  LEA R0, PROMPT2
PRO_L2  LD R7, SAVE7
  ST R7, SAVE7
  TRAP X22
  LD R7, SAVE7
  LD R1, SAVE1
  LD R2, SAVE2
  RET
```

其中判断奇偶的子程序JUDGE_N用数与1相与得到的结果来判断正负:

```
;judge the number is whether odd or even
JUDGE_N ST R1, SAVE1
  LD R1, TIMES
  AND R0, R0, #0
  ADD R0, R0, #1
  AND R0, R0, R1
  LD R1, SAVE1
  RET
```

2.4 子程序INPUT

从键盘输入一个字符存放到R_0,若$R_0 \leq 0$或$R_0 > 6$则为非法输入,返回$R_0 = 0$,否则返回R_0.

```
INPUT   ST R1 SAVE1  ;backup
  ST R7 SAVE7
  TRAP X23
  LD R7 SAVE7
  LD R1 NUM_CH  ;R1=-48
  ADD R0, R0, R1  ;char->int
  BRNZ INP_L1  ;check whether R0 > 0
  AND R1, R1, #0
  ADD R1, R1, #-6
  ADD R1, R0, R1
  BRp INP_L1  ;check whether R0 < 7
  BRnzp INP_L2
INP_L1  AND R0, R0, #0
INP_L2  LD R1 SAVE1
  RET
```

2.5 子程序PLAY

检查棋盘中用于输入的列有无空位,若有则R_0返回1,根据$TIMES$的奇偶判断是哪一方的棋子并更新棋盘;否则R_0返回0.

```
;start the game
PLAY ST R1, SAVE1  ;backup
  ST R2, SAVE2
  ST R3, SAVE3
  AND R3, R3, #0
  ADD R3, R3, #6  ;R3=6
  LEA R1, BOARD  ;calculate the offset
  ADD R1, R1, #+15  ;R1 points at the begin of 6th line
  ADD R1, R1, #+14
  ADD R1, R1, R0
  LD R0 TIMES
PLA_L4  LDR R2, R1, #0
  BRZ PLA_L3
  ADD R3, R3, #-1
  BRNZ PLA_LD
  ADD R1, R1, #-6
  BRnzp PLA_L4
PLA_L3  ST R7 SAVE7
  ST R0 SAVE0
  JSR JUDGE_N
  LD R7, SAVE7
  ADD R0, R0, #0
  BRZ PLA_L2
  AND R2, R2, #0
  ADD R2, R2, #1
  STR R2, R1, #0
  BRnzp PLA_L1
PLA_L2  AND R2, R2, #0
  ADD R2, R2, #-1
  STR R2, R1, #0
PLA_L1  LD R1, TIMES
  ADD R1, R1, #1
```

```
  LEA R2, TIMES
  STR R1, R2, #0
  LD R0 SAVE0
  BRnzp PLA_L5
PLA_LD  LEA R0 INVALID
  AND R0, R0, #0
PLA_L5  LD R1, SAVE1
  LD R2, SAVE2
  LD R3, SAVE3
  RET
```

2.6 子程序SUM_N

根据地址参数R_6和方向参数R_0求连成线的4个值之和,若和为± 4,则有玩家胜利,返回$R_0 = 1$,否则返回$R_0 = 0$.

用R_3记录连成线的四个值之和,初始为0.读取地址到R_6中,R_1接收地址R_6的值,R_1加到R_3上后,R_6跳到下一位置,即R_6加上方向参数R_0.

方向参数R_0取值:①$R_0 = 1$时,求横向的和;②$R_0 = 5$,求斜向左下的和;③$R_0 = 6$,求纵向的和;④$R_0 = 7$,求斜向右下的和.

```
;get sum
SUM_N ST R1 SAVE1  ;backup
  ST R2 SAVE2
  ST R3 SAVE3
  ST R6 SAVE6
  AND R3, R3, #0  ;sum
  AND R2, R2, #0
  ADD R2, R2, #4
SUM_L1  LDR R1, R6, #0  ;R1=M[R6]
  ADD R3, R1, R3
  ADD R6, R6, R0  ;next row(R0=6) or next column(R0=1)
  ADD R2, R2, #-1
  BRp SUM_L1
  AND R1, R1, #0
  ADD R1, R1, #4
  AND R0, R0, #0
  ADD R1, R1, R3
  BRnp SUM_L2
  ADD R0, R0, #1
  BRnzp SUM_LD
SUM_L2  AND R1, R1, #0
  ADD R1, R1, #-4
  ADD R1, R1, R3
  BRnp SUM_LD
  ADD R0, R0, #1
  BRnzp SUM_LD  ;if the total is 4 or -4 , return positive number(R0)
SUM_LD  LD R1 SAVE1 ;else return zero(R0)
  LD R2 SAVE2
  LD R3 SAVE3
  LD R6 SAVE6
  RET
```

2.7 子程序JUDGE_W

判断是否有玩家获胜.根据方向参数,依次判断横向、纵向、斜向左下、斜向右下有无四个棋子连成线.

```
;judge win
JUDGE_W ST R1 SAVE1  ;backup
  ST R2 SAVE2
  ST R3 SAVE3
  ST R4 SAVE4
  ST R5 SAVE5
  ST R6 SAVE6

  ;judge line
  LD R1 NUM_ROW  ;first layer loop
  AND R2, R2, #0
  ADD R2, R2, #3  ;second layer loop
  LEA R6 BOARD  ;board address
WIN_L1  AND R0, R0, #0
  ADD R0, R0, #1
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #1
  ADD R2, R2, #-1
  BRp WIN_L1
  ADD R6, R6, #-3
  ADD R6, R6, #6
  ADD R2, R2, #3
  ADD R1, R1, #-1
  BRp WIN_L1

  ;judge column
  LD R1 NUM_ROW  ;first layer loop
  AND R2, R2, #0
  ADD R2, R2, #3  ;second layer loop
  LEA R6 BOARD  ;board address
WIN_L2  AND R0, R0, #0
  ADD R0, R0, #6
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #6
  ADD R2, R2, #-1
  BRp WIN_L2
  ADD R6, R6, #-9
  ADD R6, R6, #-9
  ADD R6, R6, #1
  ADD R2, R2, #3
  ADD R1, R1, #-1
  BRp WIN_L2

  ;judge down
  AND R1, R1, #0  ;first layer loop
  ADD R1, R1, #3
  LEA R6 BOARD  ;board address
```

```
WIN_L4  AND R2, R2, #0
  ADD R2, R2, R1  ;second layer loop
WIN_L3  AND R0, R0, #0
  ADD R0, R0, #7
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #1
  ADD R2, R2, #-1
  BRp WIN_L3
  AND R3, R3, #0
  ADD R3, R3, R1
  NOT R3, R3
  ADD R3, R3, #1
  ADD R6, R6, R3
  ADD R6, R6, #6
  ADD R1, R1, #-1
  BRp WIN_L4


  ;judge up
  AND R1, R1, #0  ;first layer loop
  ADD R1, R1, #3
  LEA R6 BOARD
  ADD R6, R6, #5  ;board address
WIN_L6  AND R2, R2, #0
  ADD R2, R2, R1  ;second layer loop
WIN_L5  AND R0, R0, #0
  ADD R0, R0, #5
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #-1
  ADD R2, R2, #-1
  BRp WIN_L5
  ADD R6, R6, R1
  ADD R6, R6, #6
  ADD R1, R1, #-1
  BRp WIN_L6
  AND R0, R0, #0
WIN_LW  LD R1, TIMES
  LD R2, NUM_END  ;R2=-36
  ADD R2, R1, R2
  BRNZ WIN_LD
  ADD R0, R0, #1
WIN_LD  LD R1 SAVE1
  LD R2 SAVE2
  LD R3 SAVE3
  LD R4 SAVE4
  LD R5 SAVE5
  LD R6 SAVE6
  RET
```

2.8 总代码

```
  .ORIG x3000
LOOP1 LD R0 TIMES  ;R0=1
  JSR DISPLAY  ;print the board
REINPUT JSR PROMPT  ;output prompt info
  JSR INPUT
  ADD R0,  R0,  #0
  BRZ REINPUT  ;invalid input
  JSR PLAY  ;begin the game
  ADD R0,  R0,  #0
  BRZ REINPUT
  JSR JUDGE_W  ;check whether someone wins
  ADD R0,  R0,  #0
  BRp OVER  ;game over
  BRnzp LOOP1  ;game continues
OVER JSR DISPLAY  ;print the board
  JSR ANSWER  ;show the result
  HALT  ;OHHHHHHH

;print the board
DISPLAY ST R1,  SAVE1  ;backup
  ST R2,  SAVE2
  ST R3,  SAVE3
  ST R6,  SAVE6

  LEA R6,  BOARD  ;calculate the offset
  LD R2,  NUM_ROW
COL_L LD R1,  NUM_ROW
ROW_L LDR R3,  R6,  #0  ;traverse the board
  BRnp TYPE_L1  ;get the type of the chess
  LD R0, CHESS_N  ;no chess
  BRnzp TYPR_LD

TYPE_L1 BRp TYPE_L2
  LD R0, CHESS_X  ;X chess
  BRnzp TYPR_LD
TYPE_L2 LD R0, CHESS_O  ;O chess
TYPR_LD ST R7, SAVE7  ;backup
  TRAP X21  ;print the chess
  LD R7, SAVE7
  ADD R6, R6, #1  ;board address++
  ADD R1, R1, #-1
  BRNZ SPACE_L
  LD R0, SPACE
  ST R7, SAVE7
  TRAP X21  ;print the space
  LD R7, SAVE7
  ADD R1, R1, #0
SPACE_L BRp ROW_L
  LD R0, ENDL
  ST R7, SAVE7
  TRAP X21  ;print endl
  LD R7, SAVE7
  ADD R2, R2, #-1
  BRp COL_L
  LD R1, SAVE1
```

```
  LD R2, SAVE2
  LD R3, SAVE3
  LD R6, SAVE6



  RET

;print answer
ANSWER ST R1 SAVE1  ;backup
  ST R2 SAVE2
  LD R1, TIMES
  LD R2, NUM_END  ;R2=-36
  ADD R2, R1, R2
  BRNZ ANS_L1
  LEA R0 TIE  ;tie game
  BRnzp ANS_LD
ANS_L1  LD R0, TIMES  ;backup
  ST R7 SAVE77
  JSR JUDGE_N
  ADD R0, R0, #0
  BRp ANS_L2
  LEA R0, WIN1
  BRnzp ANS_LD
ANS_L2  LEA R0, WIN2
ANS_LD  LD R7, SAVE77  ;backup
  ST R7, SAVE7
  TRAP X22
  LD R7, SAVE7
  LD R1 SAVE1
  LD R2 SAVE2
  RET

;print prompt info
PROMPT ST R1, SAVE1  ;backup
  ST R2, SAVE2
  ST R0, SAVE0

  ADD R0, R0, #0
  BRp PRO_L3
  LEA R0 INVALID  ;invalid input
  ST R7 SAVE7
  TRAP X22
  LD R0 ENDL
  TRAP X21
  LD R7 SAVE7

PRO_L3  ST R0, SAVE0
  LD R0, TIMES
  ST R7, SAVE7
  JSR JUDGE_N
  ADD R0, R0, #0
  BRZ PRO_L1
  LEA R0, PROMPT1
  BRnzp PRO_L2
PRO_L1  LEA R0, PROMPT2
PRO_L2  LD R7, SAVE7
  ST R7, SAVE7
  TRAP X22
```

```
  LD R7, SAVE7
  LD R1, SAVE1
  LD R2, SAVE2
  RET

;Labels
TIE .STRINGZ "Tie Game."
INVALID .STRINGZ "Invalid move. Try again. "
PROMPT1 .STRINGZ "Player 1,  choose a column: "
PROMPT2 .STRINGZ "Player 2,  choose a column: "
WIN1  .STRINGZ "Player 1 Wins."
WIN2  .STRINGZ "Player 2 Wins."

CHESS_O .FILL X004F
CHESS_X .FILL X0058
CHESS_N .FILL X002D

TIMES .FILL #1
BOARD .BLKW 36 #0

SAVE0 .FILL   0
SAVE1 .FILL   0
SAVE2 .FILL   0
SAVE3 .FILL   0
SAVE4 .FILL   0
SAVE5 .FILL   0
SAVE6 .FILL   0
SAVE7 .FILL   0
SAVE77  .FILL 0

NUM_ROW .FILL #6
NUM_END .FILL #-36
NUM_CH  .FILL   #-48

SPACE .FILL X0020
ENDL  .FILL X000A

INPUT   ST R1 SAVE1  ;backup
  ST R7 SAVE7
  TRAP X23
  LD R7 SAVE7
  LD R1 NUM_CH  ;R1=-48
  ADD R0, R0, R1  ;char->int
  BRNZ INP_L1  ;check whether R0 > 0
  AND R1, R1, #0
  ADD R1, R1, #-6
  ADD R1, R0, R1
  BRp INP_L1  ;check whether R0 < 7
  BRnzp INP_L2
INP_L1  AND R0, R0, #0
INP_L2  LD R1 SAVE1
  RET

;start the game
PLAY ST R1, SAVE1  ;backup
  ST R2, SAVE2
  ST R3, SAVE3
  AND R3, R3, #0
```

```
  ADD R3, R3, #6  ;R3=6
  LEA R1, BOARD  ;calculate the offset
  ADD R1, R1, #+15  ;R1 points at the begin of 6th line
  ADD R1, R1, #+14
  ADD R1, R1, R0
  LD R0 TIMES
PLA_L4  LDR R2, R1, #0
  BRZ PLA_L3
  ADD R3, R3, #-1
  BRNZ PLA_LD
  ADD R1, R1, #-6
  BRnzp PLA_L4
PLA_L3  ST R7 SAVE7
  ST R0 SAVE0
  JSR JUDGE_N
  LD R7, SAVE7
  ADD R0, R0, #0
  BRZ PLA_L2
  AND R2, R2, #0
  ADD R2, R2, #1
  STR R2, R1, #0
  BRnzp PLA_L1
PLA_L2  AND R2, R2, #0
  ADD R2, R2, #-1
  STR R2, R1, #0
PLA_L1  LD R1, TIMES
  ADD R1, R1, #1
  LEA R2, TIMES
  STR R1, R2, #0
  LD R0 SAVE0
  BRnzp PLA_L5
PLA_LD  LEA R0 INVALID
  AND R0, R0, #0
PLA_L5  LD R1, SAVE1
  LD R2, SAVE2
  LD R3, SAVE3
  RET

;judge the number is whether odd or even
JUDGE_N ST R1, SAVE1
  LD R1, TIMES
  AND R0, R0, #0
  ADD R0, R0, #1
  AND R0, R0, R1
  LD R1, SAVE1
  RET

;judge win
JUDGE_W ST R1 SAVE1  ;backup
  ST R2 SAVE2
  ST R3 SAVE3
  ST R4 SAVE4
  ST R5 SAVE5
  ST R6 SAVE6

  ;judge line
  LD R1 NUM_ROW  ;first layer loop
  AND R2, R2, #0
```

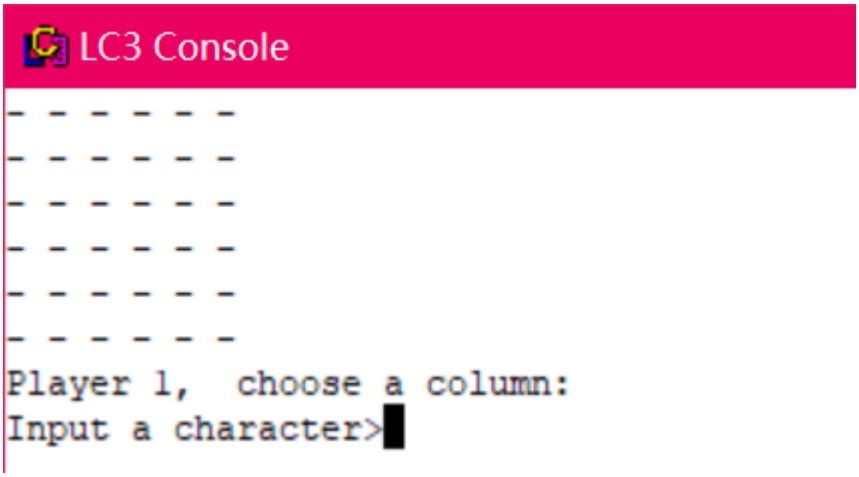
```
  ADD R2, R2, #3  ;second layer loop
  LEA R6 BOARD  ;board address
WIN_L1  AND R0, R0, #0
  ADD R0, R0, #1
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #1
  ADD R2, R2, #-1
  BRp WIN_L1
  ADD R6, R6, #-3
  ADD R6, R6, #6
  ADD R2, R2, #3
  ADD R1, R1, #-1
  BRp WIN_L1

  ;judge column
  LD R1 NUM_ROW  ;first layer loop
  AND R2, R2, #0
  ADD R2, R2, #3  ;second layer loop
  LEA R6 BOARD  ;board address
WIN_L2  AND R0, R0, #0
  ADD R0, R0, #6
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #6
  ADD R2, R2, #-1
  BRp WIN_L2
  ADD R6, R6, #-9
  ADD R6, R6, #-9
  ADD R6, R6, #1
  ADD R2, R2, #3
  ADD R1, R1, #-1
  BRp WIN_L2

  ;judge down
  AND R1, R1, #0  ;first layer loop
  ADD R1, R1, #3
  LEA R6 BOARD  ;board address
WIN_L4  AND R2, R2, #0
  ADD R2, R2, R1  ;second layer loop
WIN_L3  AND R0, R0, #0
  ADD R0, R0, #7
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #1
  ADD R2, R2, #-1
  BRp WIN_L3
  AND R3, R3, #0
  ADD R3, R3, R1
```

```
  NOT R3, R3
  ADD R3, R3, #1
  ADD R6, R6, R3
  ADD R6, R6, #6
  ADD R1, R1, #-1
  BRp WIN_L4


  ;judge up
  AND R1, R1, #0  ;first layer loop
  ADD R1, R1, #3
  LEA R6 BOARD
  ADD R6, R6, #5  ;board address
WIN_L6  AND R2, R2, #0
  ADD R2, R2, R1  ;second layer loop
WIN_L5  AND R0, R0, #0
  ADD R0, R0, #5
  ST R7 SAVE7
  JSR SUM_N
  LD R7 SAVE7
  ADD R0, R0, #0
  BRp WIN_LW
  ADD R6, R6, #-1
  ADD R2, R2, #-1
  BRp WIN_L5
  ADD R6, R6, R1
  ADD R6, R6, #6
  ADD R1, R1, #-1
  BRp WIN_L6
  AND R0, R0, #0
WIN_LW  LD R1, TIMES
  LD R2, NUM_END  ;R2=-36
  ADD R2, R1, R2
  BRNZ WIN_LD
  ADD R0, R0, #1
WIN_LD  LD R1 SAVE1
  LD R2 SAVE2
  LD R3 SAVE3
  LD R4 SAVE4
  LD R5 SAVE5
  LD R6 SAVE6
  RET

;get sum
SUM_N ST R1 SAVE1  ;backup
  ST R2 SAVE2
  ST R3 SAVE3
  ST R6 SAVE6
  AND R3, R3, #0  ;sum
  AND R2, R2, #0
  ADD R2, R2, #4
SUM_L1  LDR R1, R6, #0  ;R1=M[R6]
  ADD R3, R1, R3
  ADD R6, R6, R0  ;next row(R0=6) or next column(R0=1)
  ADD R2, R2, #-1
  BRp SUM_L1
  AND R1, R1, #0
  ADD R1, R1, #4
```

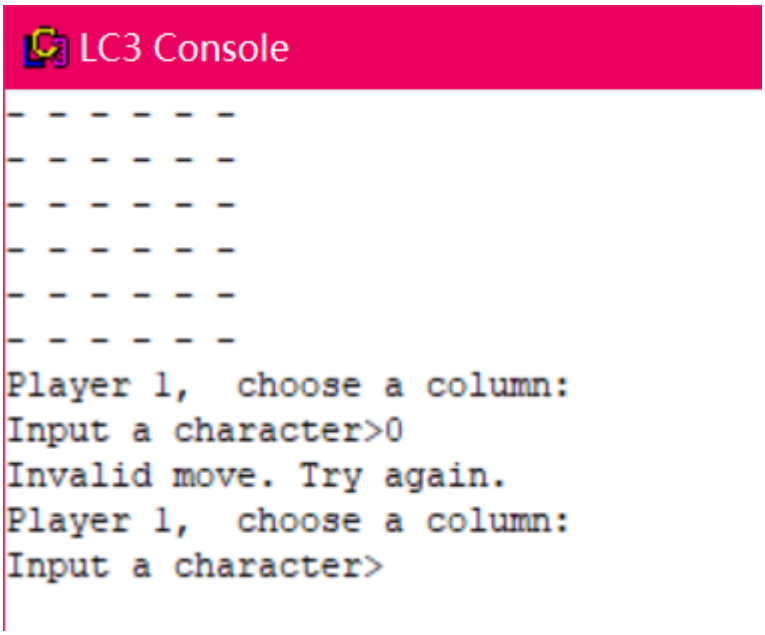
```
  AND R0, R0, #0
  ADD R1, R1, R3
  BRnp SUM_L2
  ADD R0, R0, #1
  BRnzp SUM_LD
SUM_L2  AND R1, R1, #0
  ADD R1, R1, #-4
  ADD R1, R1, R3
  BRnp SUM_LD
  ADD R0, R0, #1
  BRnzp SUM_LD  ;if the total is 4 or -4 , return positive number(R0)
SUM_LD  LD R1 SAVE1 ;else return zero(R0)
  LD R2 SAVE2
  LD R3 SAVE3
  LD R6 SAVE6
  RET

  .END
```

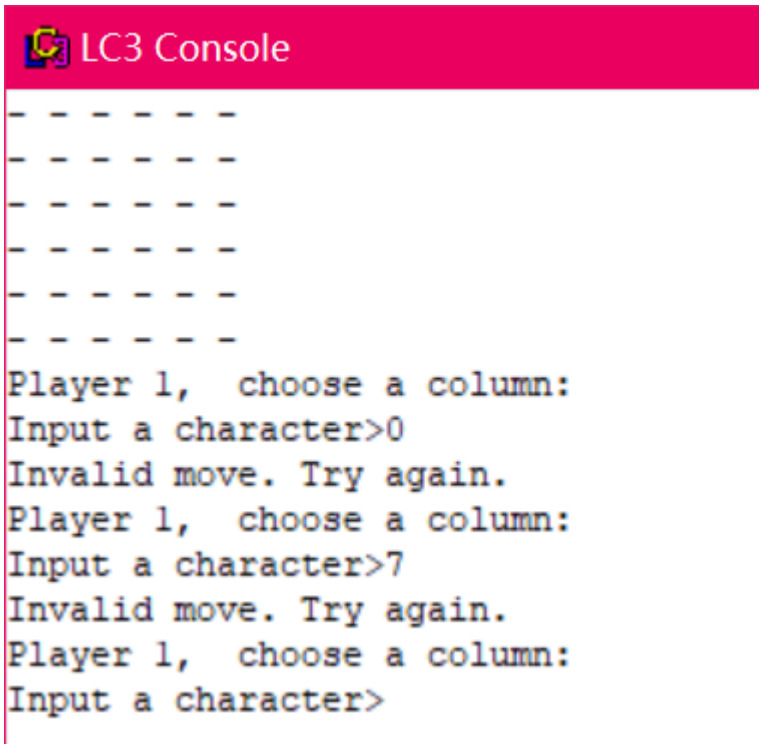
3. 实验结果

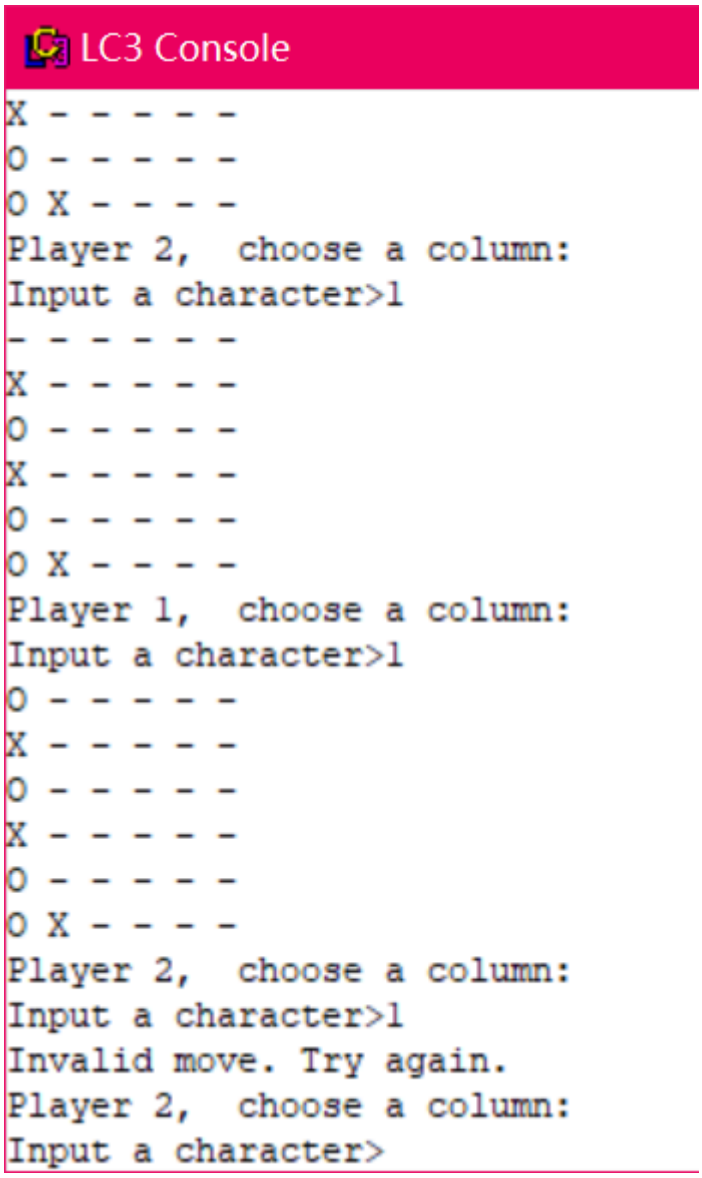
3.1 等待用户输入



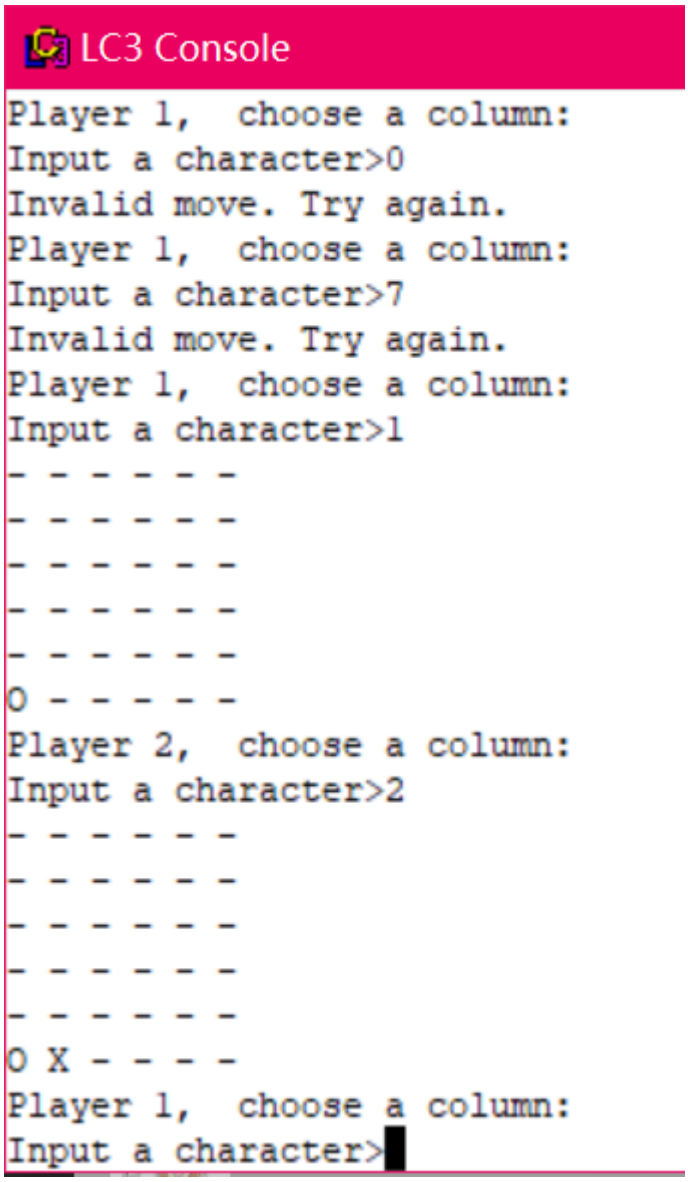
3.2 非法输入


LC3 Console
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
Player 1, choose a column:
Input a character>0
Invalid move. Try again.
Player 1, choose a column:
Input a character>

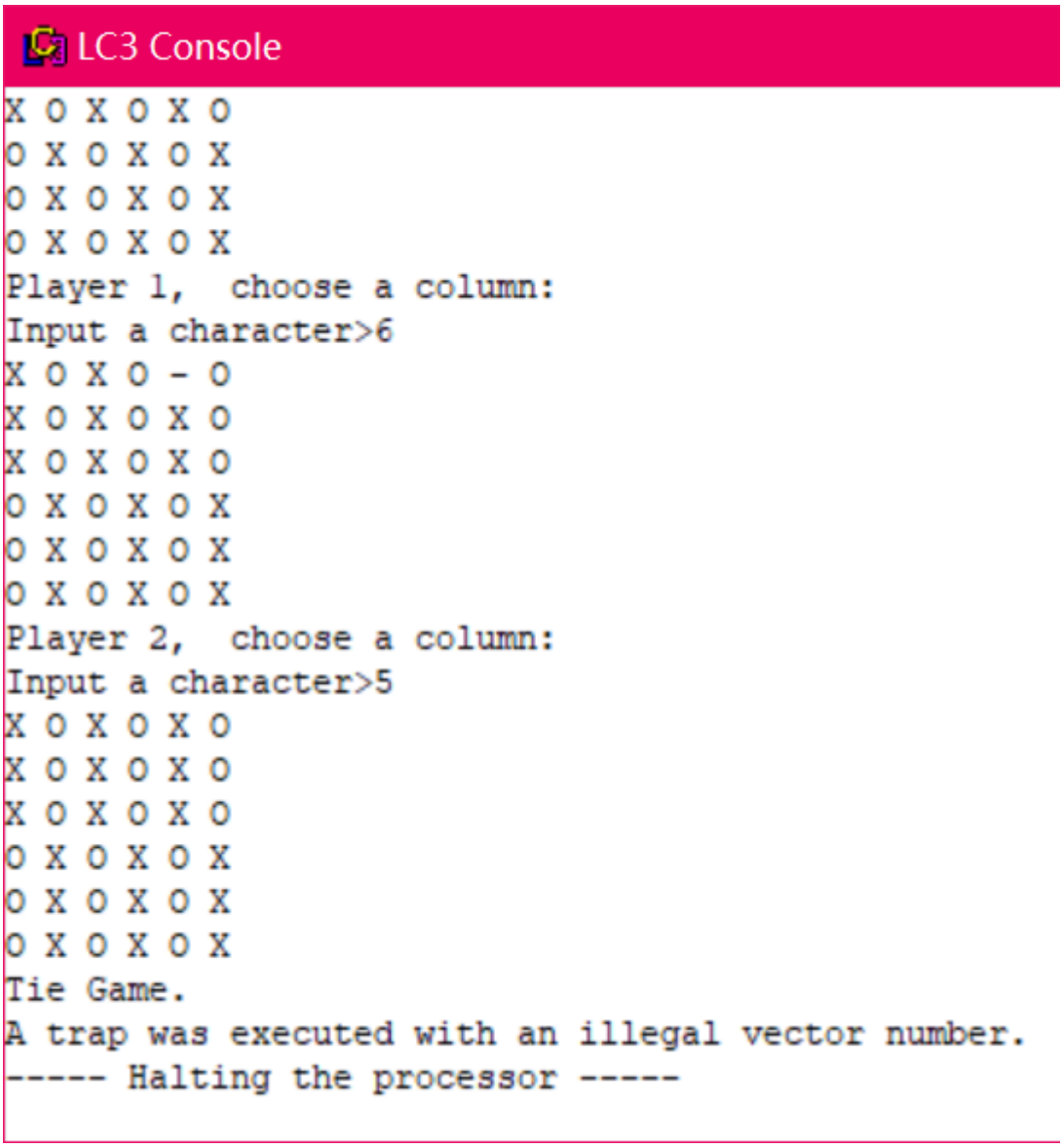

LC3 Console
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
Player 1, choose a column:
Input a character>0
Invalid move. Try again.
Player 1, choose a column:
Input a character>7
Invalid move. Try again.
Player 1, choose a column:
Input a character>


LC3 Console
X - - - - -
O - - - - -
O X - - - -
Player 2, choose a column:
Input a character>1
- - - - - -
X - - - - -
O - - - - -
X - - - - -
O - - - - -
O X - - - -
Player 1, choose a column:
Input a character>1
O - - - - -
X - - - - -
O - - - - -
X - - - - -
O - - - - -
O X - - - -
Player 2, choose a column:
Input a character>1
Invalid move. Try again.
Player 2, choose a column:
Input a character>

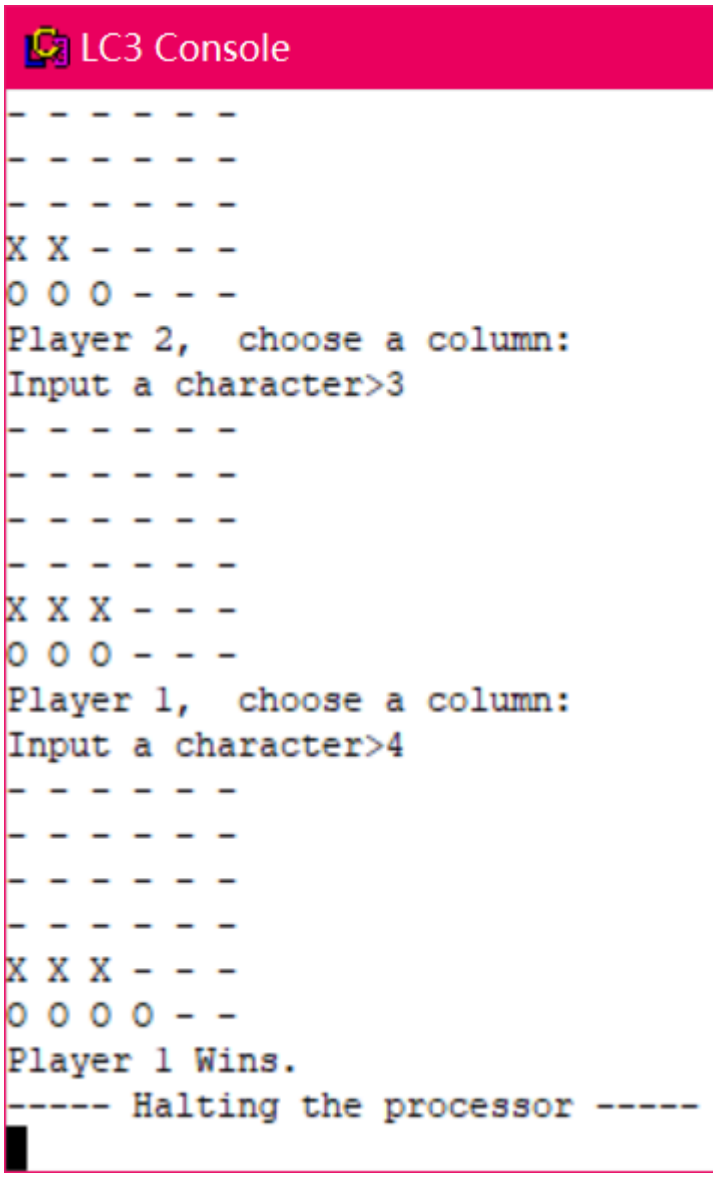
3.3 合法输入


LC3 Console
Player 1, choose a column:
Input a character>0
Invalid move. Try again.
Player 1, choose a column:
Input a character>7
Invalid move. Try again.
Player 1, choose a column:
Input a character>1
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
O - - - - -
Player 2, choose a column:
Input a character>2
- - - - - -
- - - - - -
- - - - - -
- - - - - -
- - - - - -
O X - - - -
Player 1, choose a column:
Input a character>

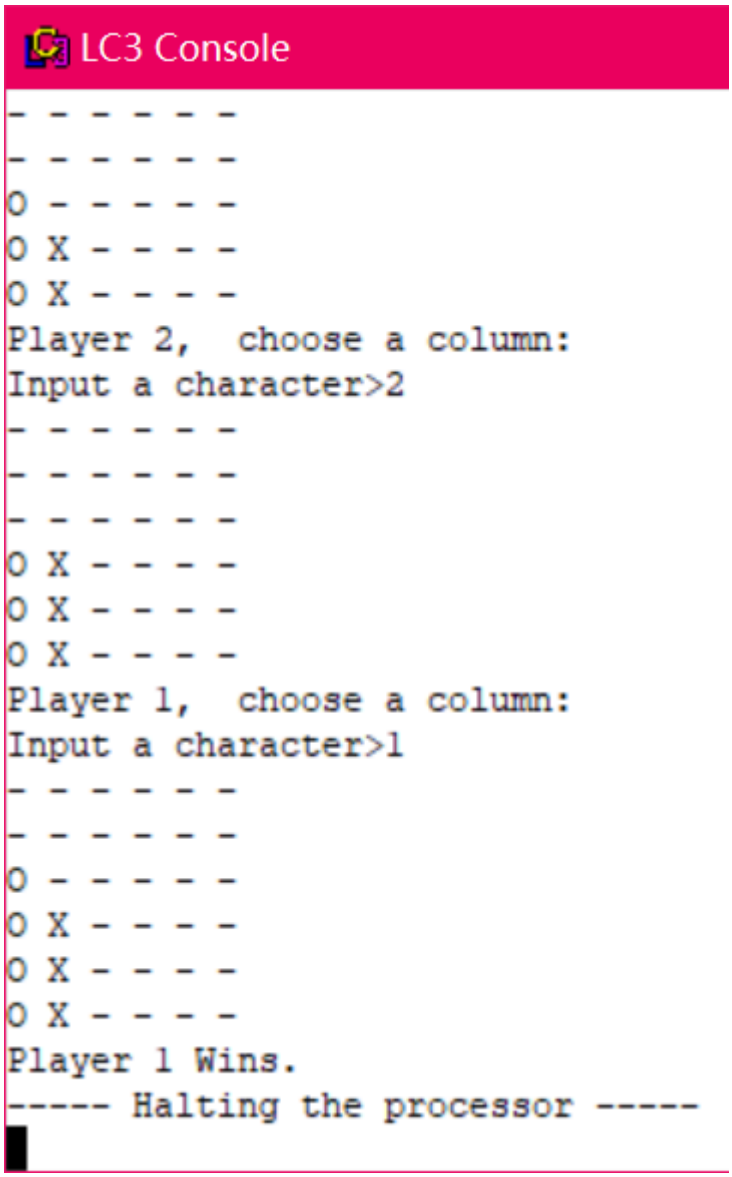
3.4 平局


LC3 Console
X O X O X O
O X O X O X
O X O X O X
O X O X O X
Player 1, choose a column:
Input a character>6
X O X O - O
X O X O X O
X O X O X O
O X O X O X
O X O X O X
O X O X O X
Player 2, choose a column:
Input a character>5
X O X O X O
X O X O X O
X O X O X O
O X O X O X
O X O X O X
O X O X O X
Tie Game.
A trap was executed with an illegal vector number.
----- Halting the processor -----

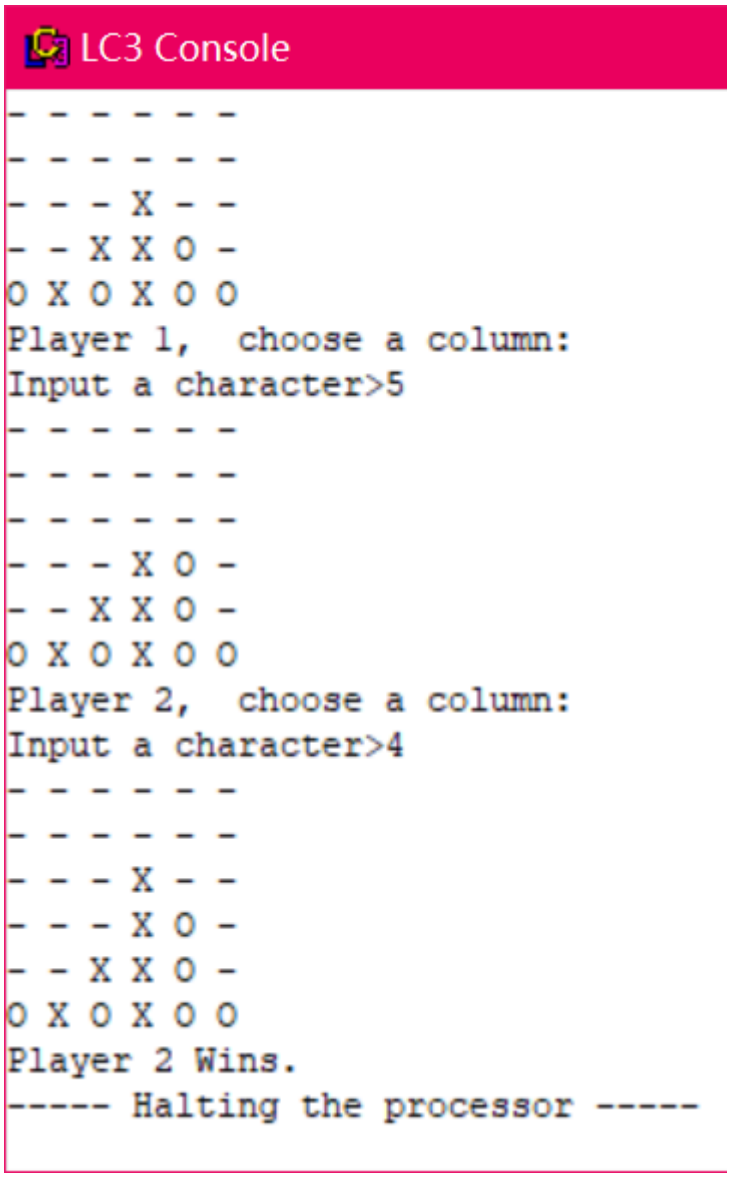
3.5 横向连线


LC3 Console
- - - - - -
- - - - - -
- - - - - -
X X - - - -
O O O - - -
Player 2, choose a column:
Input a character>3
- - - - - -
- - - - - -
- - - - - -
- - - - - -
X X X - - -
O O O - - -
Player 1, choose a column:
Input a character>4
- - - - - -
- - - - - -
- - - - - -
- - - - - -
X X X - - -
O O O O - -
Player 1 Wins.
----- Halting the processor -----

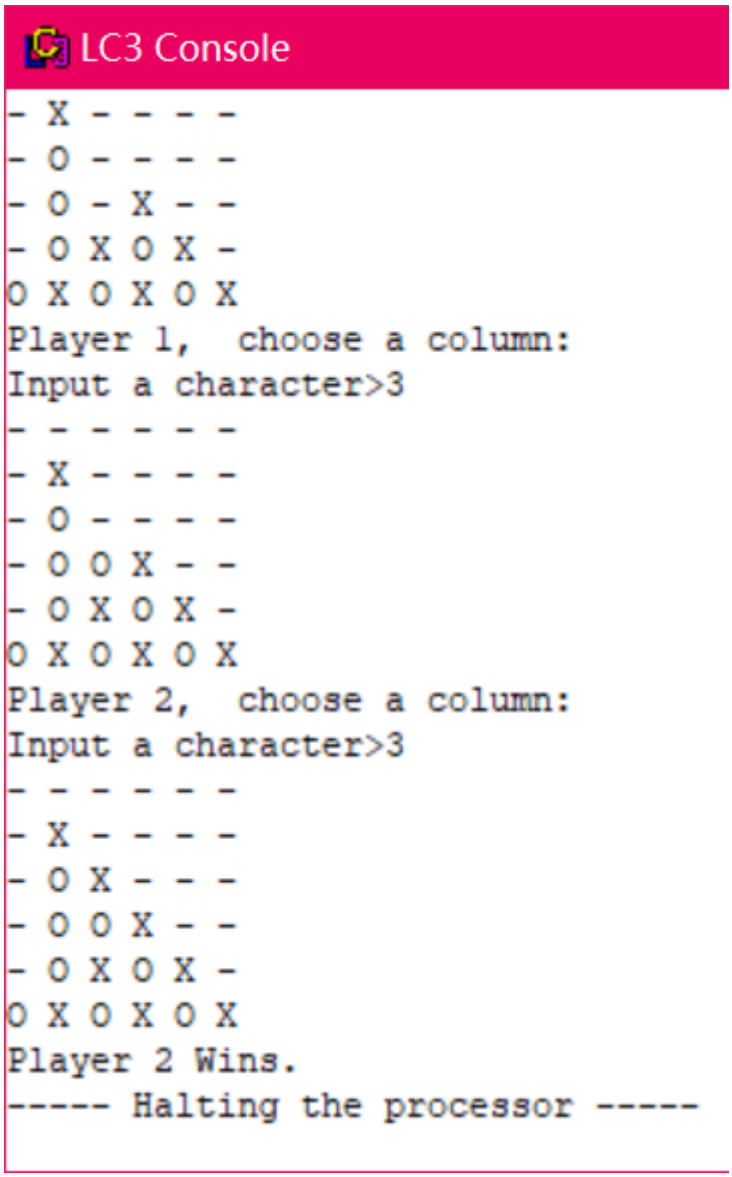
3.6 纵向连线


LC3 Console
- - - - - - -
- - - - - - -
O - - - - - -
O X - - - - -
O X - - - - -
Player 2, choose a column:
Input a character>2
- - - - - - -
- - - - - - -
- - - - - - -
O X - - - - -
O X - - - - -
O X - - - - -
Player 1, choose a column:
Input a character>1
- - - - - - -
- - - - - - -
O - - - - - -
O X - - - - -
O X - - - - -
O X - - - - -
Player 1 Wins.
----- Halting the processor -----

3.7 斜向左下连线


LC3 Console
- - - - - - -
- - - - - - -
- - - X - - -
- - X X O - -
O X O X O O
Player 1, choose a column:
Input a character>5
- - - - - - -
- - - - - - -
- - - - - - -
- - - X O - -
- - X X O - -
O X O X O O
Player 2, choose a column:
Input a character>4
- - - - - - -
- - - - - - -
- - - X - - -
- - - X O - -
- - X X O - -
O X O X O O
Player 2 Wins.
----- Halting the processor -----

3.8 斜向右下连线



4. 实验结论

1. 用子程序可以让代码思路更加清晰.
2. 进入子程序和从子程序返回主程序时要注意寄存器的备份.
3. 可通过一个数的二进制表示的末位来判断其奇偶性.