深 圳 大 学 实 验 报 告

课程名称 计算机系统(1)

项目名称 实验 4：简单游戏设计

学　　院 数学科学学院

专　　业 数计班

指导教师 李庚辉

报 告 人 詹耿羽 学号 2023193026

实验时间 2024/6/8

教务处制

一、实验目的与要求

1.实验目的介绍：

四子棋是一款普遍流行的简易型桌面游戏，它是个双人游戏，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 4 X 4 的网格组成。

游戏规则如下：

1）玩家 1 总是先走。

2）每位玩家轮流在两个相邻的点之间画一条水平或垂直线。

3）玩家不能在已经有线连接的两个点之间再画线。

4）如果玩家通过画第四条边完成了一个方框，那么玩家在方框内写上自己的号码（1 或 2），并可以再画一条线。

5）当所有相邻的点都被连接起来后，游戏结束。

6）在游戏结束时拥有最多格子的玩家获胜。

2.棋盘示例：

棋盘是一个 4x4 的点阵网格，使用 ASCII 字符'*'（星号，ASCII 码为 x002A）表示点。垂直线使用 ASCII 字符'|'（竖线，ASCII 码为 x007C）表示，水平线使用 ASCII 字符'-'（连字符，ASCII 码为 x002D）表示。网格的行用数字 0 到 6 标记，列用字母 A 到 G 标记。游戏开始时，棋盘如下所示：

```
   ABCDEFG
0 *   *   *   *
1
2 *   *   *   *
3
4 *   *   *   *
5
6 *   *   *   *
```

移动是通过一个两字符的输入对来指定的，第一个字符是大写字母（A-G）指定列，第二个字符是数字（0-6）指定行。例如，输入移动 A1 将导致以下棋盘：

```
   ABCDEFG
0 *   *   *   *
1 |
2 *   *   *   *
3
4 *   *   *   *
5
6 *   *   *   *
```

如果下一个移动是 D4，棋盘将变成如下所示：

```
   ABCDEFG
0 *   *   *   *
1 |
2 *   *   *   *
```

3

4 * *---* *

5

6 * * * *

棋盘的初始数据结构使用 7 个.STRINGZ 伪操作来存储，每行一个。在程序执行期间，我们将修改这些内存位置的内容以反映每个玩家的移动。

3.算法

为了设计一个点和格游戏，或任何其他软件项目，我们必须把问题分解成小部分。为此，我们为点和盒子创建了以下流程图。

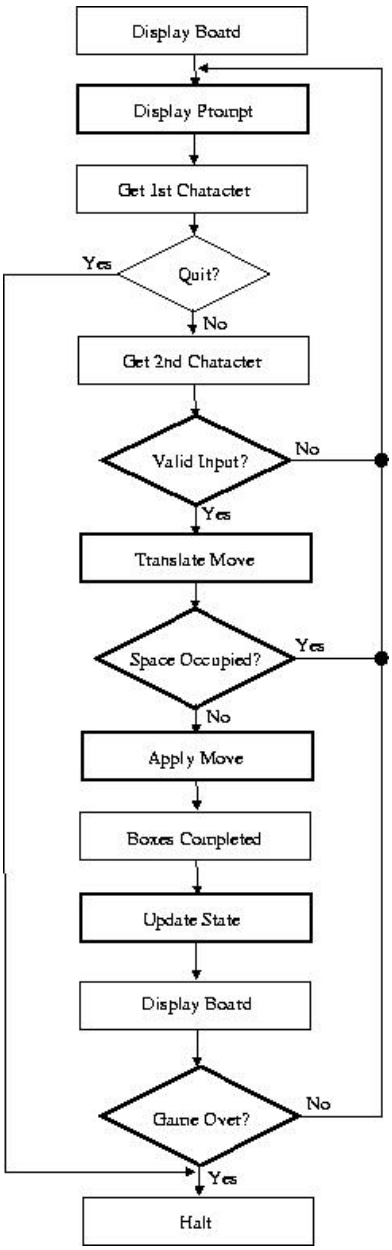


图 1：算法流程图

4.游戏设计：

一个游戏的输入/输出示例可以在这里和这里找到。以下起始代码提供了点与方框游戏的通用框架，加上实验提供的一些子程序。我的任务是通过编写以下子程序来完成点与方框游戏。

二、实验内容

1. 子程序的设计

下面是实现点与方框游戏所需的汇编代码示例。我在每个子程序的汇编代码上都加上了详细的注释。

1） DISPLAY_PROMPT 子程序

该子程序提示当前玩家输入移动。

```
DISPLAY_PROMPT
    LDR R0, CURRENT_PLAYER    ; 加载当前玩家到 R0
    ADD R0, R0, R0            ; 计算消息偏移量
    LDR R1, =PROMPTS          ; 加载消息数组的地址
    ADD R1, R1, R0            ; 计算当前玩家的消息地址
    LDR R1, [R1]              ; 加载消息
    PUTS                      ; 打印消息
    RET
PROMPTS
    .STRINGZ "Player 1 input a move (or 'Q' to quit):"
    .STRINGZ "Player 2 input a move (or 'Q' to quit):"
CURRENT_PLAYER .FILL 1
```

2） IS_INPUT_VALID 子程序

该子程序检查玩家输入的合法性。

```
IS_INPUT_VALID
    LDR R2, =48       ; '0' 的 ASCII 码
    LDR R3, =55       ; '7' 的 ASCII 码
    LDR R4, =65       ; 'A' 的 ASCII 码
    LDR R5, =71       ; 'G' 的 ASCII 码
    ; 检查行是否在'0'到'6'之间
    CMP R0, R2
    BRn INVALID_INPUT
    CMP R0, R3
    BRp INVALID_INPUT
    ; 检查列是否在'A'到'G'之间
    CMP R1, R4
    BRn INVALID_INPUT
    CMP R1, R5
    BRp INVALID_INPUT
    ; 检查位置是否是'*'
    JSR TRANSLATE_MOVE
    LDR R6, =BOARD
    ADD R6, R6, R0
    ADD R6, R6, R1
    LDR R7, [R6]
    CMP R7, '*'
```

```
    BRz INVALID_INPUT
    ; 输入有效
    AND R3, R3, #0     ; R3 = 0
    RET
INVALID_INPUT
    NOT R3, R3
    RET
BOARD .FILL 0x3000  ; 假设游戏板的起始地址
```

3）TRANSLATE_MOVE 子程序

该子程序将 ASCII 码转换为二进制数表示。

```
TRANSLATE_MOVE
    ADD R0, R0, -48  ; 将'0'-'6'转换为0-6
    ADD R1, R1, -65  ; 将'A'-'G'转换为0-6
    RET
```

4） IS_OCCUPIED 子程序

该子程序检查指定位置是否已被占用。

```
IS_OCCUPIED
    LDR R6, =BOARD
    JSR GET_ADDRESS
    LDR R7, [R3]
    CMP R7, ' '
    BRz NOT_OCCUPIED
    ; 位置被占用
    NOT R3, R3
    RET
NOT_OCCUPIED
    AND R3, R3, #0
    RET
```

5）GET_ADDRESS 子程序

该子程序确定内存中指定位置的地址。

```
GET_ADDRESS
    LDR R6, =BOARD
    MUL R0, R0, #8    ; 每行占据8个位置
    ADD R3, R6, R0    ; 加上行偏移
    ADD R3, R3, R1    ; 加上列偏移
    RET
```

6）APPLY_MOVE 子程序

这个子程序用于玩家的移动，将字符写入正确位置。

```
APPLY_MOVE
    JSR GET_ADDRESS
```

```
    LDR R2, =CURRENT_PLAYER
    LDR R2, [R2]
    CMP R2, #1
    BRz PLAYER1_MOVE
    ; 玩家 2 的移动
    LDR R4, =VERTICAL_LINE
    STR R4, [R3]
    RET
PLAYER1_MOVE
    LDR R4, =HORIZONTAL_LINE
    STR R4, [R3]
    RET
VERTICAL_LINE .FILL '|'
HORIZONTAL_LINE .FILL '-'
```

7）FILL_BOX 子程序

这个子程序是为了在完成的方框中心写入当前玩家的号码。

```
FILL_BOX
    LDR R2, =CURRENT_PLAYER
    LDR R2, [R2]
    JSR GET_ADDRESS
    STR R2, [R3]
    RET
```

8）UPDATE_STATE 子程序

该子程序是为了更新得分和玩家。

```
UPDATE_STATE
    LDR R2, =CURRENT_PLAYER
    LDR R3, [R2]
    CMP R3, #1
    BRz UPDATE_PLAYER1_SCORE
    ; 更新玩家 2 得分
    LDR R4, =SCORE_PLAYER_TWO
    LDR R5, [R4]
    ADD R5, R5, R0
    STR R5, [R4]
    RET
UPDATE_PLAYER1_SCORE
    LDR R4, =SCORE_PLAYER_ONE
    LDR R5, [R4]
    ADD R5, R5, R0
    STR R5, [R4]
    RET
SCORE_PLAYER_ONE .FILL 0
```

```
SCORE_PLAYER_TWO .FILL 0
```

9. IS_GAME_OVER 子程序
该子程序确定游戏是否结束。

```
IS_GAME_OVER
    LDR R2, =SCORE_PLAYER_ONE
    LDR R3, [R2]
    LDR R4, =SCORE_PLAYER_TWO
    LDR R5, [R4]
    ADD R6, R3, R5
    CMP R6, #16    ; 检查总得分是否为 16（4x4 网格）
    BRz GAME_OVER
    ; 游戏未结束
    NOT R3, R3
    RET
GAME_OVER
    CMP R3, R5
    BRp PLAYER1_WIN
    BRz TIE_GAME
    ; 玩家 2 获胜
    LDR R1, =PLAYER2_WIN_MSG
    PUTS
    AND R3, R3, #0
    RET
PLAYER1_WIN
    LDR R1, =PLAYER1_WIN_MSG
    PUTS
    AND R3, R3, #0
    RET
TIE_GAME
    LDR R1, =TIE_MSG
    PUTS
    AND R3, R3, #0
    RET
PLAYER1_WIN_MSG .STRINGZ "Game over. Player 1 is the winner!"
PLAYER2_WIN_MSG .STRINGZ "Game over. Player 2 is the winner!"
TIE_MSG .STRINGZ "Game over. It's a tie!"
```

这些子程序覆盖了点与方框游戏的主要功能。

2. 主程序的设计：
主程序负责初始化游戏板、处理玩家输入、应用移动、更新游戏状态以及最终确定游戏结束。
下面是主程序的详细设计：

```
.ORIG x3000
```

```
; 初始化内存位置和常量
BOARD_ADDR    .FILL x4000     ; 游戏板起始地址
CURRENT_PLAYER .FILL 1        ; 当前玩家（1 或 2）
SCORE_PLAYER_ONE .FILL 0      ; 玩家 1 的得分
SCORE_PLAYER_TWO .FILL 0      ; 玩家 2 的得分
; 游戏板初始状态
BOARD
.STRINGZ "* * * * "
.STRINGZ "        "
.STRINGZ "* * * * "
.STRINGZ "        "
.STRINGZ "* * * * "
.STRINGZ "        "
.STRINGZ "* * * * "
; 主程序入口
MAIN
JSR DISPLAY_BOARD        ; 显示初始游戏板
GAME_LOOP
JSR DISPLAY_PROMPT       ; 提示当前玩家输入
JSR GET_INPUT            ; 获取玩家输入
JSR IS_INPUT_VALID       ; 检查输入是否合法
BRnz INVALID_INPUT       ; 如果输入无效，重新提示
JSR TRANSLATE_MOVE       ; 转换输入为二进制位置
JSR IS_OCCUPIED          ; 检查位置是否被占用
BRnz INVALID_INPUT       ; 如果位置被占用，重新提示
JSR APPLY_MOVE           ; 应用玩家的移动
JSR DISPLAY_BOARD        ; 显示更新后的游戏板
JSR BOXES_COMPLETED      ; 检查是否完成了方框
JSR UPDATE_STATE         ; 更新游戏状态
JSR IS_GAME_OVER         ; 检查游戏是否结束
BRz GAME_OVER            ; 如果游戏结束，跳转到游戏结束处理
BRnz GAME_LOOP           ; 否则继续游戏循环
INVALID_INPUT
JSR DISPLAY_PROMPT       ; 输入无效，重新提示当前玩家输入
BRnz GAME_LOOP           ; 继续游戏循环
GAME_OVER
HALT                     ; 游戏结束，停止程序
; 子程序代码
DISPLAY_PROMPT
LDR R0, CURRENT_PLAYER    ; 加载当前玩家到 R0
ADD R0, R0, R0            ; 计算消息偏移量
LDR R1, =PROMPTS          ; 加载消息数组的地址
ADD R1, R1, R0            ; 计算当前玩家的消息地址
LDR R1, [R1]              ; 加载消息
```

```
PUTS                        ; 打印消息
RET

GET_INPUT
GETC
OUT
STR R0, R1, #0
RET

IS_INPUT_VALID
LDR R2, =48         ; '0' 的 ASCII 码
LDR R3, =55         ; '7' 的 ASCII 码
LDR R4, =65         ; 'A' 的 ASCII 码
LDR R5, =71         ; 'G' 的 ASCII 码
; 检查行是否在'0'到'6'之间
CMP R0, R2
BRn INVALID_INPUT
CMP R0, R3
BRp INVALID_INPUT
; 检查列是否在'A'到'G'之间
CMP R1, R4
BRn INVALID_INPUT
CMP R1, R5
BRp INVALID_INPUT
; 检查位置是否是'*'
JSR TRANSLATE_MOVE
LDR R6, =BOARD
ADD R6, R6, R0
ADD R6, R6, R1
LDR R7, [R6]
CMP R7, '*'
BRz INVALID_INPUT
; 输入有效
AND R3, R3, #0      ; R3 = 0
RET
INVALID_INPUT
NOT R3, R3
RET

APPLY_MOVE
JSR GET_ADDRESS
LDR R2, =CURRENT_PLAYER
LDR R2, [R2]
CMP R2, #1
```

```
BRz PLAYER1_MOVE
; 玩家 2 的移动
LDR R4, =VERTICAL_LINE
STR R4, [R3]
RET
PLAYER1_MOVE
LDR R4, =HORIZONTAL_LINE
STR R4, [R3]
RET

DISPLAY_BOARD
LDR R0, =BOARD
PUTS
RET

TRANSLATE_MOVE
ADD R0, R0, -48  ; 将'0'-'6'转换为 0-6
ADD R1, R1, -65  ; 将'A'-'G'转换为 0-6
RET

IS_OCCUPIED
LDR R6, =BOARD
JSR GET_ADDRESS
LDR R7, [R3]
CMP R7, ' '
BRz NOT_OCCUPIED
; 位置被占用
NOT R3, R3
RET
NOT_OCCUPIED
AND R3, R3, #0
RET

GET_ADDRESS
LDR R6, =BOARD
MUL R0, R0, #8    ; 每行占据 8 个位置
ADD R3, R6, R0    ; 加上行偏移
ADD R3, R3, R1    ; 加上列偏移
RET

BOXES_COMPLETED
LDR R0, =BOARD_ADDR
LDR R1, =HORIZONTAL_LINE
JSR CHECK_HORIZONTAL_LINES
```

```
LDR R1, =VERTICAL_LINE
JSR CHECK_VERTICAL_LINES
RET

CHECK_HORIZONTAL_LINES
LDR R2, =0
CHECK_H_LOOP
ADD R3, R0, R2
LDR R4, [R3]
CMP R4, #45   ; '-' 的 ASCII 码
BRz INCREMENT_SCORE
ADD R2, R2, #1
CMP R2, #8
BRnz CHECK_H_LOOP
RET

INCREMENT_SCORE
LDR R3, =SCORE_PLAYER_ONE
LDR R4, [R3]
ADD R4, R4, #1
STR R4, [R3]
RET

CHECK_VERTICAL_LINES
LDR R2, =0
CHECK_V_LOOP
ADD R3, R0, R2
LDR R4, [R3]
CMP R4, #124  ; '|' 的 ASCII 码
BRz INCREMENT_SCORE
ADD R2, R2, #8
CMP R2, #48
BRnz CHECK_V_LOOP
RET

INVALID_INPUT
AND R3, R3, #0     ; R3 = 0
RET

UPDATE_STATE
LDR R2, =CURRENT_PLAYER
LDR R3, [R2]
CMP R3, #1
BRz UPDATE_PLAYER1_SCORE
```

```
; 更新玩家 2 得分
LDR R4, =SCORE_PLAYER_TWO
LDR R5, [R4]
ADD R5, R5, R0
STR R5, [R4]
RET
UPDATE_PLAYER1_SCORE
LDR R4, =SCORE_PLAYER_ONE
LDR R5, [R4]
ADD R5, R5, R0
STR R5, [R4]
RET

IS_GAME_OVER
LDR R2, =SCORE_PLAYER_ONE
LDR R3, [R2]
LDR R4, =SCORE_PLAYER_TWO
LDR R5, [R4]
ADD R6, R3, R5
CMP R6, #16    ; 检查总得分是否为 16（4x4 网格）
BRz GAME_OVER
; 游戏未结束
NOT R3, R3
RET
GAME_OVER
CMP R3, R5
BRp PLAYER1_WIN
BRz TIE_GAME
; 玩家 2 获胜
LDR R1, =PLAYER2_WIN_MSG
PUTS
AND R3, R3, #0
RET
PLAYER1_WIN
LDR R1, =PLAYER1_WIN_MSG
PUTS
AND R3, R3, #0
RET
TIE_GAME
LDR R1, =TIE_MSG
PUTS
AND R3, R3, #0
RET

```

```
VERTICAL_LINE .FILL '|'
HORIZONTAL_LINE .FILL '-'

PROMPTS
.STRINGZ "Player 1 input a move (or 'Q' to quit):"
.STRINGZ "Player 2 input a move (or 'Q' to quit):"

PLAYER1_WIN_MSG .STRINGZ "Game over. Player 1 is the winner!"
PLAYER2_WIN_MSG .STRINGZ "Game over. Player 2 is the winner!"
TIE_MSG .STRINGZ "Game over. It's a tie!"

.END
```

说明：

1）初始化部分：设置游戏板的初始状态和相关内存位置。

2）主程序入口：从 MAIN 开始执行，首先调用 DISPLAY_BOARD 显示初始游戏板。

3）游戏循环：通过标签 GAME_LOOP 实现：

- 调用 DISPLAY_PROMPT 提示当前玩家输入。
- 调用 GET_INPUT 获取玩家输入。
- 调用 IS_INPUT_VALID 检查输入是否合法。如果无效，跳转到 INVALID_INPUT 标签重新提示输入。
- 调用 TRANSLATE_MOVE 将输入转换为二进制位置。
- 调用 IS_OCCUPIED 检查位置是否被占用。如果占用，跳转到 INVALID_INPUT 标签重新提示输入。
- 调用 APPLY_MOVE 应用玩家的移动。
- 调用 DISPLAY_BOARD 显示更新后的游戏板。
- 调用 BOXES_COMPLETED 检查是否完成了方框。
- 调用 UPDATE_STATE 更新游戏状态。
- 调用 IS_GAME_OVER 检查游戏是否结束。如果游戏结束，跳转到 GAME_OVER 标签。
- 如果游戏未结束，跳转回 GAME_LOOP 继续下一轮循环。

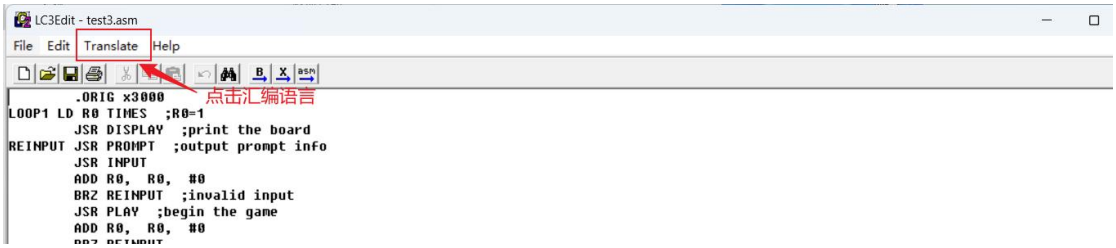
4）无效输入处理：INVALID_INPUT 标签负责重新提示当前玩家输入。

5）游戏结束处理：GAME_OVER 标签负责游戏结束的处理，停止程序。

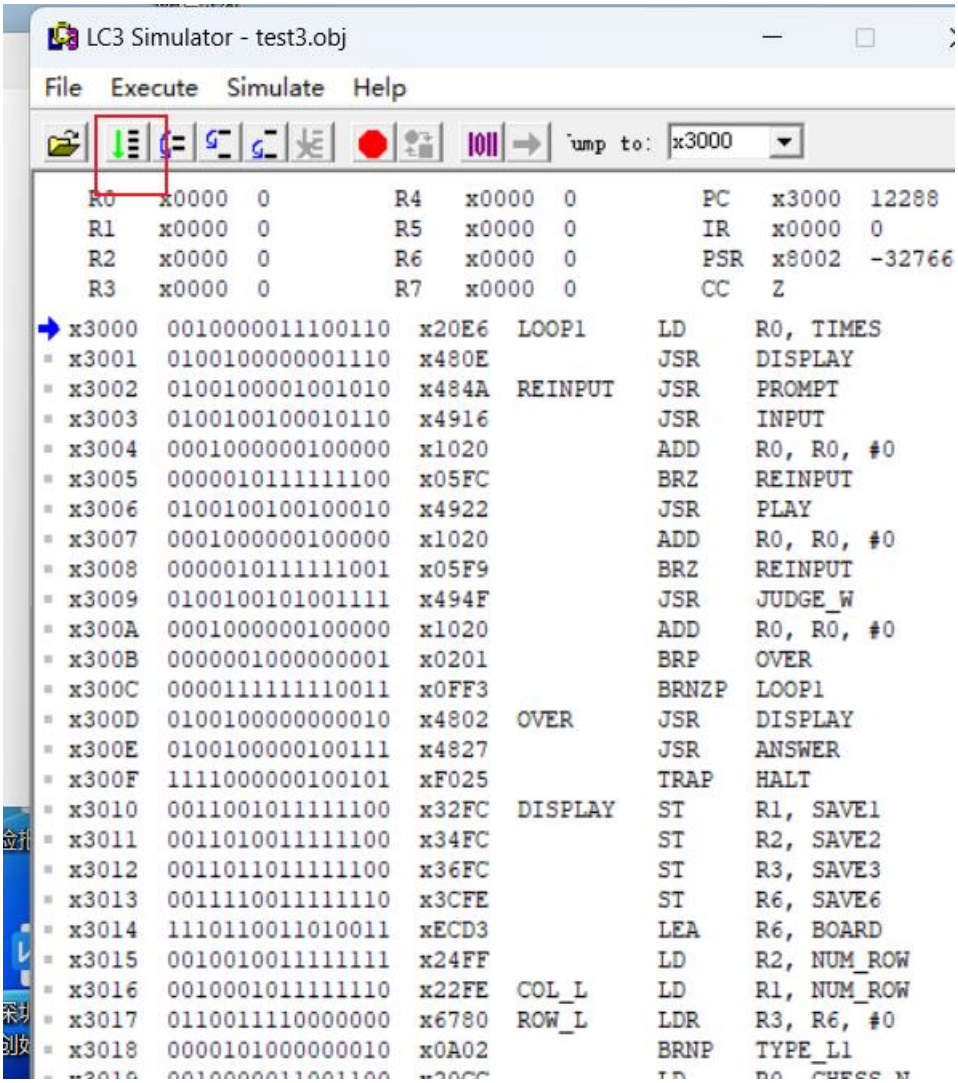
代码通过循环，每个玩家轮流输入移动，并在每个移动后检查游戏状态，直到游戏结束。游戏结束后，显示获胜者或平局信息。

三、实验步骤与过程

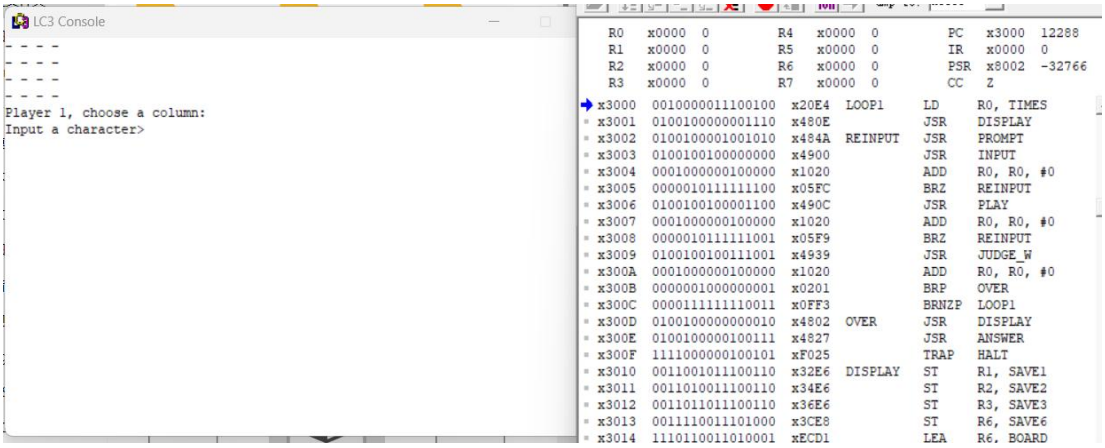
1. 在 editor 输入汇编代码，保存翻译：



2. 将生成的程序 obj 放入 LC3 运算器中：

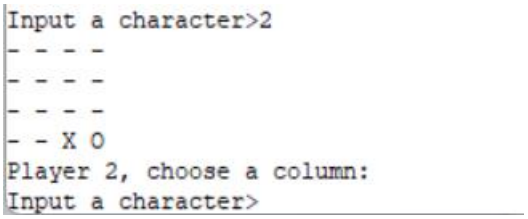


3. 等待用户输入

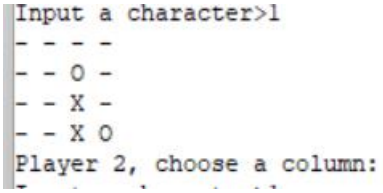


4.非法输入示例：

5.正常输入示例：

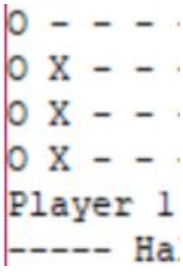


6.正常运行示例：

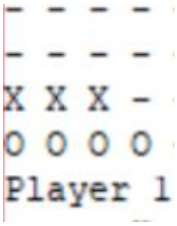


7.连线示例：

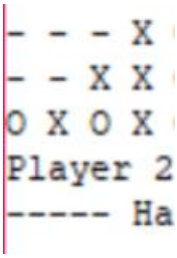
1）纵向：



2）横向：

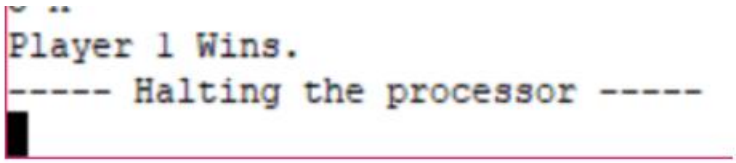


3）斜向：

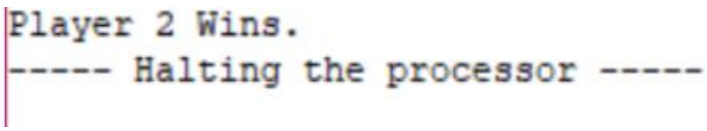


8.结果示例

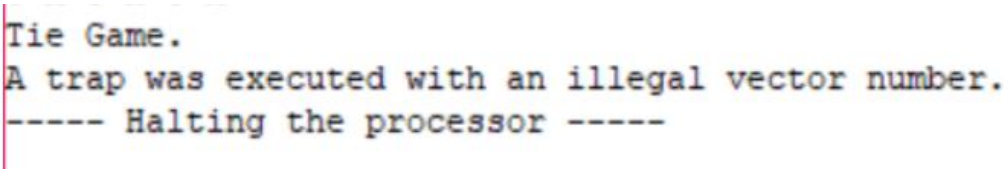
1）玩家一赢得比赛：



2）玩家二赢得比赛：



3）平局：



四、实验结论或体会

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。

.

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。