

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：

```
ABCDEF
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：

```
ABCDEF
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：

```
ABCDEF
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。

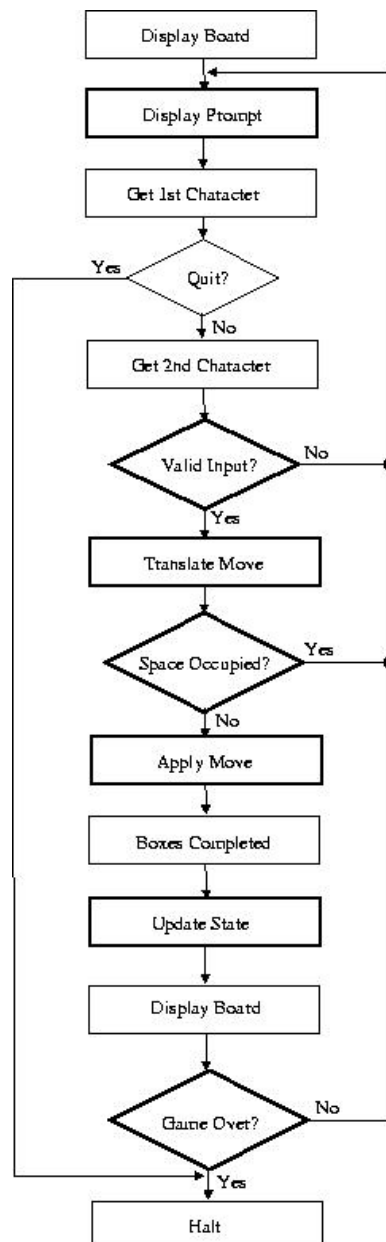


图 1：算法流程图

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。

```
1.  DISPLAY_PROMPT
2.      LDR R0, CURRENT_PLAYER    ; 加载当前玩家到 R0
3.      ADD R0, R0, R0             ; 计算消息偏移量
4.      LDR R1, =PROMPTS          ; 加载消息数组的地址
5.      ADD R1, R1, R0             ; 计算当前玩家的消息地址
6.      LDR R1, [R1]              ; 加载消息
7.      PUTS                      ; 打印消息
8.      RET
9.  PROMPTS
10.     .STRINGZ "Player 1 input a move (or 'Q' to quit):"
11.     .STRINGZ "Player 2 input a move (or 'Q' to quit):"
12.  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。

```
1.  IS_INPUT_VALID
2.      LDR R2, =48               ; '0' 的 ASCII 码
3.      LDR R3, =55               ; '7' 的 ASCII 码
4.      LDR R4, =65               ; 'A' 的 ASCII 码
5.      LDR R5, =71               ; 'G' 的 ASCII 码
6.      ; 检查行是否在 '0' 到 '6' 之间
7.      CMP R0, R2
8.      BRn INVALID_INPUT
9.      CMP R0, R3
10.     BRp INVALID_INPUT
11.     ; 检查列是否在 'A' 到 'G' 之间
12.     CMP R1, R4
13.     BRn INVALID_INPUT
14.     CMP R1, R5
15.     BRp INVALID_INPUT
16.     ; 检查位置是否是 '*'
17.     JSR TRANSLATE_MOVE
18.     LDR R6, =BOARD
19.     ADD R6, R6, R0
20.     ADD R6, R6, R1
21.     LDR R7, [R6]
22.     CMP R7, '*'
```

```

23.      BRz INVALID_INPUT
24.      ; 输入有效
25.      AND R3, R3, #0      ; R3 = 0
26.      RET
27.  INVALID_INPUT
28.      NOT R3, R3
29.      RET
30.  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。

```

1.  TRANSLATE_MOVE
2.      ADD R0, R0, -48 ; 将'0'-'6'转换为 0-6
3.      ADD R1, R1, -65 ; 将'A'-'G'转换为 0-6
4.      RET

```

4) IS_OCCUPIED 子程序

该子程序检查指定位置是否已被占用。

```

1.  IS_OCCUPIED
2.      LDR R6, =BOARD
3.      JSR GET_ADDRESS
4.      LDR R7, [R3]
5.      CMP R7, ' '
6.      BRz NOT_OCCUPIED
7.      ; 位置被占用
8.      NOT R3, R3
9.      RET
10. NOT_OCCUPIED
11.      AND R3, R3, #0
12.      RET

```

5) GET_ADDRESS 子程序

该子程序确定内存中指定位置的地址。

```

1.  GET_ADDRESS
2.      LDR R6, =BOARD
3.      MUL R0, R0, #8 ; 每行占据 8 个位置
4.      ADD R3, R6, R0 ; 加上行偏移
5.      ADD R3, R3, R1 ; 加上列偏移
6.      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。

```

1.  APPLY_MOVE
2.      JSR GET_ADDRESS

```

```

3.      LDR R2, =CURRENT_PLAYER
4.      LDR R2, [R2]
5.      CMP R2, #1
6.      BRz PLAYER1_MOVE
7.      ; 玩家 2 的移动
8.      LDR R4, =VERTICAL_LINE
9.      STR R4, [R3]
10.     RET
11.  PLAYER1_MOVE
12.     LDR R4, =HORIZONTAL_LINE
13.     STR R4, [R3]
14.     RET
15.  VERTICAL_LINE .FILL '|'
16.  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。

```

1.      FILL_BOX
2.      LDR R2, =CURRENT_PLAYER
3.      LDR R2, [R2]
4.      JSR GET_ADDRESS
5.      STR R2, [R3]
6.      RET

```

8) UPDATE_STATE 子程序

该子程序是为了更新得分和玩家。

```

1.      UPDATE_STATE
2.      LDR R2, =CURRENT_PLAYER
3.      LDR R3, [R2]
4.      CMP R3, #1
5.      BRz UPDATE_PLAYER1_SCORE
6.      ; 更新玩家 2 得分
7.      LDR R4, =SCORE_PLAYER_TWO
8.      LDR R5, [R4]
9.      ADD R5, R5, R0
10.     STR R5, [R4]
11.     RET
12.  UPDATE_PLAYER1_SCORE
13.     LDR R4, =SCORE_PLAYER_ONE
14.     LDR R5, [R4]
15.     ADD R5, R5, R0
16.     STR R5, [R4]
17.     RET
18.  SCORE_PLAYER_ONE .FILL 0

```

```
19. SCORE_PLAYER_TWO .FILL 0
```

9. IS_GAME_OVER 子程序

该子程序确定游戏是否结束。

```
1. IS_GAME_OVER
2.     LDR R2, =SCORE_PLAYER_ONE
3.     LDR R3, [R2]
4.     LDR R4, =SCORE_PLAYER_TWO
5.     LDR R5, [R4]
6.     ADD R6, R3, R5
7.     CMP R6, #16      ; 检查总得分是否为 16 (4x4 网格)
8.     BRZ GAME_OVER
9.     ; 游戏未结束
10.    NOT R3, R3
11.    RET
12. GAME_OVER
13.    CMP R3, R5
14.    BRp PLAYER1_WIN
15.    BRz TIE_GAME
16.    ; 玩家 2 获胜
17.    LDR R1, =PLAYER2_WIN_MSG
18.    PUTS
19.    AND R3, R3, #0
20.    RET
21. PLAYER1_WIN
22.    LDR R1, =PLAYER1_WIN_MSG
23.    PUTS
24.    AND R3, R3, #0
25.    RET
26. TIE_GAME
27.    LDR R1, =TIE_MSG
28.    PUTS
29.    AND R3, R3, #0
30.    RET
31. PLAYER1_WIN_MSG .STRINGZ "Game over. Player 1 is the winner!"
32. PLAYER2_WIN_MSG .STRINGZ "Game over. Player 2 is the winner!"
33. 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：

```
1. .ORIG x3000
```

```

2.      ; 初始化内存位置和常量
3.      BOARD_ADDR      .FILL  x4000      ; 游戏板起始地址
4.      CURRENT_PLAYER  .FILL  1          ; 当前玩家（1 或 2）
5.      SCORE_PLAYER_ONE .FILL  0          ; 玩家 1 的得分
6.      SCORE_PLAYER_TWO .FILL  0          ; 玩家 2 的得分
7.      ; 游戏板初始状态
8.      BOARD
9.      .STRINGZ "*" * * * * "
10.     .STRINGZ "          "
11.     .STRINGZ "*" * * * * "
12.     .STRINGZ "          "
13.     .STRINGZ "*" * * * * "
14.     .STRINGZ "          "
15.     .STRINGZ "*" * * * * "
16.     ; 主程序入口
17.     MAIN
18.     JSR DISPLAY_BOARD      ; 显示初始游戏板
19.     GAME_LOOP
20.     JSR DISPLAY_PROMPT     ; 提示当前玩家输入
21.     JSR GET_INPUT          ; 获取玩家输入
22.     JSR IS_INPUT_VALID     ; 检查输入是否合法
23.     BRnz INVALID_INPUT     ; 如果输入无效，重新提示
24.     JSR TRANSLATE_MOVE     ; 转换输入为二进制位置
25.     JSR IS_OCCUPIED        ; 检查位置是否被占用
26.     BRnz INVALID_INPUT     ; 如果位置被占用，重新提示
27.     JSR APPLY_MOVE         ; 应用玩家的移动
28.     JSR DISPLAY_BOARD      ; 显示更新后的游戏板
29.     JSR BOXES_COMPLETED    ; 检查是否完成了方框
30.     JSR UPDATE_STATE       ; 更新游戏状态
31.     JSR IS_GAME_OVER       ; 检查游戏是否结束
32.     BRz GAME_OVER          ; 如果游戏结束，跳转到游戏结束处理
33.     BRnz GAME_LOOP         ; 否则继续游戏循环
34.     INVALID_INPUT
35.     JSR DISPLAY_PROMPT     ; 输入无效，重新提示当前玩家输入
36.     BRnz GAME_LOOP         ; 继续游戏循环
37.     GAME_OVER
38.     HALT                   ; 游戏结束，停止程序
39.     ; 子程序代码
40.     DISPLAY_PROMPT
41.     LDR R0, CURRENT_PLAYER  ; 加载当前玩家到 R0
42.     ADD R0, R0, R0          ; 计算消息偏移量
43.     LDR R1, =PROMPTS        ; 加载消息数组的地址
44.     ADD R1, R1, R0          ; 计算当前玩家的消息地址
45.     LDR R1, [R1]            ; 加载消息

```


46. PUTS ; 打印消息

47. RET

48.

49. GET_INPUT

50. GETC

51. OUT

52. STR R0, R1, #0

53. RET

54.

55. IS_INPUT_VALID

56. LDR R2, =48 ; '0' 的 ASCII 码

57. LDR R3, =55 ; '7' 的 ASCII 码

58. LDR R4, =65 ; 'A' 的 ASCII 码

59. LDR R5, =71 ; 'G' 的 ASCII 码

60. ; 检查行是否在 '0' 到 '6' 之间

61. CMP R0, R2

62. BRn INVALID_INPUT

63. CMP R0, R3

64. BRp INVALID_INPUT

65. ; 检查列是否在 'A' 到 'G' 之间

66. CMP R1, R4

67. BRn INVALID_INPUT

68. CMP R1, R5

69. BRp INVALID_INPUT

70. ; 检查位置是否是 '*'

71. JSR TRANSLATE_MOVE

72. LDR R6, =BOARD

73. ADD R6, R6, R0

74. ADD R6, R6, R1

75. LDR R7, [R6]

76. CMP R7, '*'

77. BRz INVALID_INPUT

78. ; 输入有效

79. AND R3, R3, #0 ; R3 = 0

80. RET

81. INVALID_INPUT

82. NOT R3, R3

83. RET

84.

85. APPLY_MOVE

86. JSR GET_ADDRESS

87. LDR R2, =CURRENT_PLAYER

88. LDR R2, [R2]

89. CMP R2, #1

```

90. BRz PLAYER1_MOVE
91. ; 玩家 2 的移动
92. LDR R4, =VERTICAL_LINE
93. STR R4, [R3]
94. RET
95. PLAYER1_MOVE
96. LDR R4, =HORIZONTAL_LINE
97. STR R4, [R3]
98. RET
99.
100. DISPLAY_BOARD
101. LDR R0, =BOARD
102. PUTS
103. RET
104.
105. TRANSLATE_MOVE
106. ADD R0, R0, -48 ; 将'0'-'6'转换为0-6
107. ADD R1, R1, -65 ; 将'A'-'G'转换为0-6
108. RET
109.
110. IS_OCCUPIED
111. LDR R6, =BOARD
112. JSR GET_ADDRESS
113. LDR R7, [R3]
114. CMP R7, ' '
115. BRz NOT_OCCUPIED
116. ; 位置被占用
117. NOT R3, R3
118. RET
119. NOT_OCCUPIED
120. AND R3, R3, #0
121. RET
122.
123. GET_ADDRESS
124. LDR R6, =BOARD
125. MUL R0, R0, #8 ; 每行占据 8 个位置
126. ADD R3, R6, R0 ; 加上行偏移
127. ADD R3, R3, R1 ; 加上列偏移
128. RET
129.
130. BOXES_COMPLETED
131. LDR R0, =BOARD_ADDR
132. LDR R1, =HORIZONTAL_LINE
133. JSR CHECK_HORIZONTAL_LINES

```

```
134. LDR R1, =VERTICAL_LINE
135. JSR CHECK_VERTICAL_LINES
136. RET
137.
138. CHECK_HORIZONTAL_LINES
139. LDR R2, =0
140. CHECK_H_LOOP
141. ADD R3, R0, R2
142. LDR R4, [R3]
143. CMP R4, #45 ; '-' 的 ASCII 码
144. BRz INCREMENT_SCORE
145. ADD R2, R2, #1
146. CMP R2, #8
147. BRnz CHECK_H_LOOP
148. RET
149.
150. INCREMENT_SCORE
151. LDR R3, =SCORE_PLAYER_ONE
152. LDR R4, [R3]
153. ADD R4, R4, #1
154. STR R4, [R3]
155. RET
156.
157. CHECK_VERTICAL_LINES
158. LDR R2, =0
159. CHECK_V_LOOP
160. ADD R3, R0, R2
161. LDR R4, [R3]
162. CMP R4, #124 ; '|' 的 ASCII 码
163. BRz INCREMENT_SCORE
164. ADD R2, R2, #8
165. CMP R2, #48
166. BRnz CHECK_V_LOOP
167. RET
168.
169. INVALID_INPUT
170. AND R3, R3, #0 ; R3 = 0
171. RET
172.
173. UPDATE_STATE
174. LDR R2, =CURRENT_PLAYER
175. LDR R3, [R2]
176. CMP R3, #1
177. BRz UPDATE_PLAYER1_SCORE
```

```

178. ; 更新玩家 2 得分
179. LDR R4, =SCORE_PLAYER_TWO
180. LDR R5, [R4]
181. ADD R5, R5, R0
182. STR R5, [R4]
183. RET
184. UPDATE_PLAYER1_SCORE
185. LDR R4, =SCORE_PLAYER_ONE
186. LDR R5, [R4]
187. ADD R5, R5, R0
188. STR R5, [R4]
189. RET
190.
191. IS_GAME_OVER
192. LDR R2, =SCORE_PLAYER_ONE
193. LDR R3, [R2]
194. LDR R4, =SCORE_PLAYER_TWO
195. LDR R5, [R4]
196. ADD R6, R3, R5
197. CMP R6, #16 ; 检查总得分是否为 16 (4x4 网格)
198. BRz GAME_OVER
199. ; 游戏未结束
200. NOT R3, R3
201. RET
202. GAME_OVER
203. CMP R3, R5
204. BRp PLAYER1_WIN
205. BRz TIE_GAME
206. ; 玩家 2 获胜
207. LDR R1, =PLAYER2_WIN_MSG
208. PUTS
209. AND R3, R3, #0
210. RET
211. PLAYER1_WIN
212. LDR R1, =PLAYER1_WIN_MSG
213. PUTS
214. AND R3, R3, #0
215. RET
216. TIE_GAME
217. LDR R1, =TIE_MSG
218. PUTS
219. AND R3, R3, #0
220. RET
221.

```

```

222. VERTICAL_LINE .FILL '|'
223. HORIZONTAL_LINE .FILL '-'
224.
225. PROMPTS
226. .STRINGZ "Player 1 input a move (or 'Q' to quit):"
227. .STRINGZ "Player 2 input a move (or 'Q' to quit):"
228.
229. PLAYER1_WIN_MSG .STRINGZ "Game over. Player 1 is the winner!"
230. PLAYER2_WIN_MSG .STRINGZ "Game over. Player 2 is the winner!"
231. TIE_MSG .STRINGZ "Game over. It's a tie!"
232.
233. .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：

```

LC3Edit - test3.asm
File Edit Translate Help
.ORG 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

```

2. 将生成的程序 obj 放入 LC3 运算器中:

LC3 Simulator - test3.obj

File Execute Simulate Help

Registers:

R0	x0000	0	R4	x0000	0	PC	x3000	12288
R1	x0000	0	R5	x0000	0	IR	x0000	0
R2	x0000	0	R6	x0000	0	PSR	x8002	-32766
R3	x0000	0	R7	x0000	0	CC	Z	

Instruction List:

Address	Hex	Label	Op	Operand
x3000	0010000011100110	LOOP1	LD	R0, TIMES
x3001	0100100000001110		JSR	DISPLAY
x3002	0100100001001010	REINPUT	JSR	PROMPT
x3003	0100100100010110		JSR	INPUT
x3004	0001000000100000		ADD	R0, R0, #0
x3005	0000010111111100		BRZ	REINPUT
x3006	0100100100100010		JSR	PLAY
x3007	0001000000100000		ADD	R0, R0, #0
x3008	0000010111111001		BRZ	REINPUT
x3009	0100100101001111		JSR	JUDGE_W
x300A	0001000000100000		ADD	R0, R0, #0
x300B	0000001000000001		BRP	OVER
x300C	0000111111110011		BRNZP	LOOP1
x300D	0100100000000010	OVER	JSR	DISPLAY
x300E	0100100000100111		JSR	ANSWER
x300F	1111000000100101		TRAP	HALT
x3010	0011001011111100	DISPLAY	ST	R1, SAVE1
x3011	0011010011111100		ST	R2, SAVE2
x3012	0011011011111100		ST	R3, SAVE3
x3013	0011110011111110		ST	R6, SAVE6
x3014	1110110011010011		LEA	R6, BOARD
x3015	0010010011111111		LD	R2, NUM_ROW
x3016	0010001011111110	COL_L	LD	R1, NUM_ROW
x3017	0110011110000000	ROW_L	LDR	R3, R6, #0
x3018	0000101000000010		BRNP	TYPE_L1

3. 等待用户输入

```

R0 x0000 0      R4 x0000 0      PC x3000 12288
R1 x0000 0      R5 x0000 0      IR x0000 0
R2 x0000 0      R6 x0000 0      PSR x8002 -32766
R3 x0000 0      R7 x0000 0      CC Z

* x3000 0010000011100100 x20E4 LOOP1 LD R0, TIMES
* x3001 01001000000001110 x480E JSR DISPLAY
* x3002 0100100001001010 x484A REINPUT JSR PROMPT
* x3003 0100100100000000 x4900 JSR INPUT
* x3004 0001000000010000 x1020 ADD R0, R0, #0
* x3005 0000010111111100 x05FC BRZ REINPUT
* x3006 01001001000001100 x490C JSR PLAY
* x3007 0001000000010000 x1020 ADD R0, R0, #0
* x3008 0000010111111001 x05F9 BRZ REINPUT
* x3009 0100100100011001 x4939 JSR JUDGE_W
* x300A 0001000000010000 x1020 ADD R0, R0, #0
* x300B 0000000100000000 x0201 BRP OVER
* x300C 0000111111110011 x0FF3 BRNZP LOOP1
* x300D 0100100000000010 x4802 OVER JSR DISPLAY
* x300E 0100100000010011 x4827 JSR ANSWER
* x300F 11110000000100101 xF025 TRAP HALT
* x3010 0011001011100110 x32E6 DISPLAY ST R1, SAVE1
* x3011 0011010011100110 x34E6 ST R2, SAVE2
* x3012 0011011011100110 x36E6 ST R3, SAVE3
* x3013 0011110011101000 x3CE8 ST R6, SAVE6
* x3014 11101100111010001 xECD1 LEA R6, BOARD
  
```

4.非法输入示例:

```

Player 1, choose a column:
Input a character>9
Invalid move. Try again.
  
```

5.正常输入示例:

```

Input a character>2
- - - -
- - - -
- - X O
Player 2, choose a column:
Input a character>
  
```

6.正常运行示例:

```

Input a character>1
- - - -
- - O -
- - X -
- - X O
Player 2, choose a column:
  
```

7.连线示例:

1) 纵向:

```

O - - -
O X - -
O X - -
O X - -
Player 1
----- Ha
  
```

2) 横向:

```
- - - -  
- - - -  
X X X -  
O O O O  
Player 1
```

3) 斜向:

```
- - - X  
- - X X  
O X O X  
Player 2  
----- Ha
```

8.结果示例

1) 玩家一赢得比赛:

```
Player 1 Wins.  
----- Halting the processor -----
```

2) 玩家二赢得比赛:

```
Player 2 Wins.  
----- Halting the processor -----
```

3) 平局:

```
Tie Game.  
A trap was executed with an illegal vector number.  
----- Halting the processor -----
```

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。