COMPUTER ARCHITECTURE - CO2007

Assignment

Tic Tac Toe 5x5 in MIPS

Instructors: Pham Quoc Cuong
Students: Tran Quang Kiet - 2052563

HO CHI MINH CITY, April, 2022

Contents

1 Introduction

1.1 Tic Tac Toe

Definition: Tic Tac Toe is a paper and pencil game for two players who take turns making the spaces in three-by-three, five-by-five, or full of paper grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. In Vietnam, It's familiar with the name: "Caro". The rules are as follow:

- 2 player will choose X or O to play.
- Each player take turn by turn to placing their X or O into one of the squares in the board.
- A player cannot place their X or O in a square already occupied by a symbol
- The game ends when a player creates a three same symbol in the row, or in the column or in diagonal row.
- In neither player creates a winning combination when all nine squares are occupied, the game is a draw game

1.2 Assignment

In this assignment I will implement this game in 5x5 version using MIPS assembly code with all of these following requirement:

- User interface so that player can play easily without any confusion.
- Run without any errors.

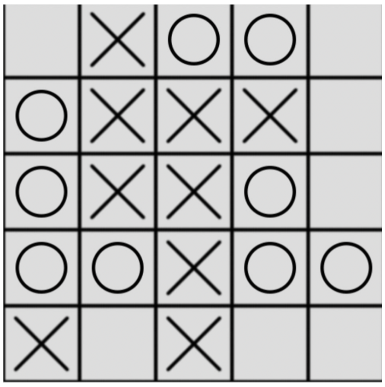


Figure 1: 5x5 Board

2 Implementation

2.1 Basic Idea

Main idea for implementation of Tic Tac Toe in MIPS:
In MIPS, the matrix 5x5 will be stored as a ID array of 25 elements, with value from 0 to 24 in ASCII. Each player will choose a number for row index and column index corresponding to the square they wish to play. The address of the square is calculated by this formula:

$$addr = baseAdd + (rowIndex * colsize + colindex) * datasize$$

After a player finished their turn, the main function will call the board function to print the board with the player's input and asking if they want to undo your move one time. Then main function will run checkwin function to check for winner and draw. If there is a winner or draw, main function will end the game and print the corresponding message. If there is no winner nor draw, the game will continue.

1. Set up the new game
2. Print Title and rule description
3. If 25 squares occupied, move to step 14
4. Change player
5. Require move form player
6. Player input move
7. If move is invalid, show message and move to step 5
8. If selected square is occupied, show message and move to step 5
9. Print board
10. Check for winner
11. Check undo, move to step 5
12. If there is no winner, move to step 3
13. Print victory announcement
14. Print draw announcement
15. End game

2.2 Flowchart

Here is the flowchart of my implementation:

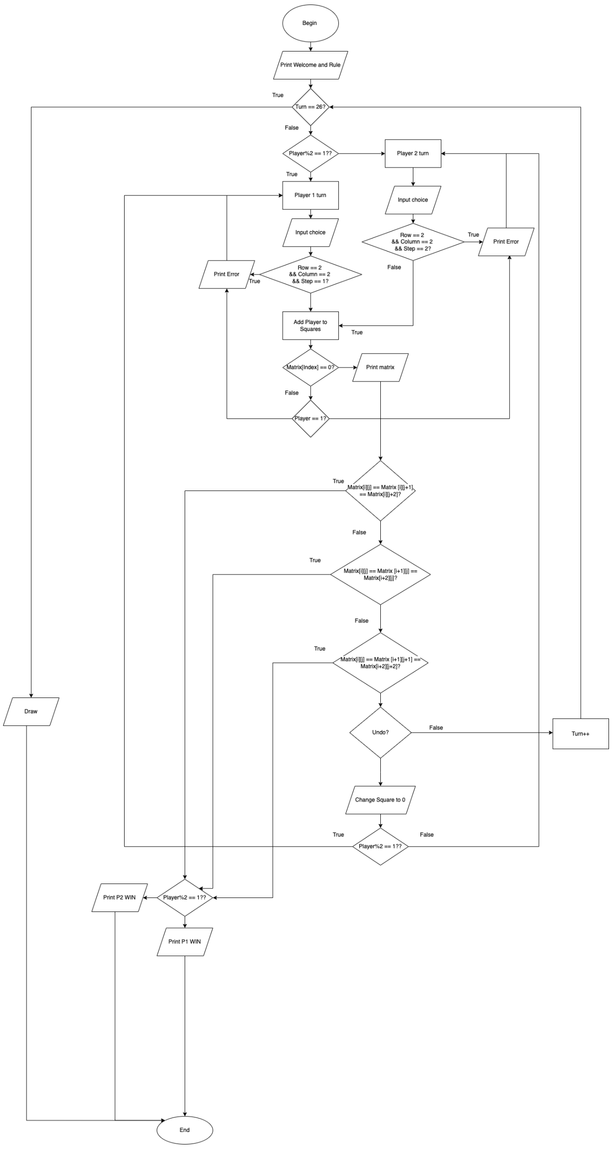
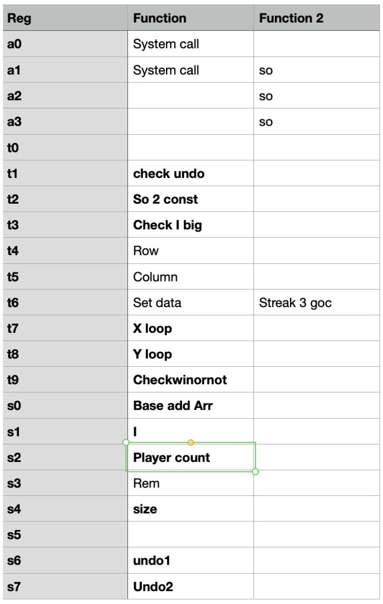


Figure 2: Flowchart

2.3 Explanation

So now, I will explain my idea for this problem. First I will so you all of my definition for all of variable in mips.



Reg	Function	Function 2
a0	System call	
a1	System call	so
a2		so
a3		so
t0		
t1	check undo	
t2	So 2 const	
t3	Check I big	
t4	Row	
t5	Column	
t6	Set data	Streak 3 goc
t7	X loop	
t8	Y loop	
t9	Checkwinornot	
s0	Base add Arr	
s1	I	
s2	Player count	
s3	Rem	
s4	size	
s5		
s6	undo1	
s7	Undo2	

Figure 3: Variable List

2.3.1 Main function pseudo-code

```
Main Function:
 Print Request
 if all squares filled then
 |  Print draw and done
else
 |  if Player == 1 then
 |  |  Call to p1 turn function
 |  else
 |  |  Call to p2 turn function
 |  end
end
```

2.3.2 P1 function

```
P1 turn Function:
 Input Row index and Column index
 if First step rule then
 |  Print invalid move
 |   Call to P1 turn
else
 |  Call to Moving function
end
```

2.3.3 P2 function

```
P1 turn Function:
 Input Row index and Column index
 if First step rule then
     Print invalid move
     Call to P2 turn
 else
     Call to Moving function
 end
```

2.3.4 Moving function

```
Moving Function:
 Calculate real position
 if matrix[pos] != 0 then
     Print invalid move
     if player == 1 then
         Call to P1 turn
     else
         Call to P2 turn
     end
 else
     Change value of pos
     Call to Print Function
 end
```

2.3.5 Print function

```
Print function:
 for $i = 0, i < 24$ do
     Print matrix[i]
     i++
 end
 Turn to check win function
```

2.3.6 Check win function

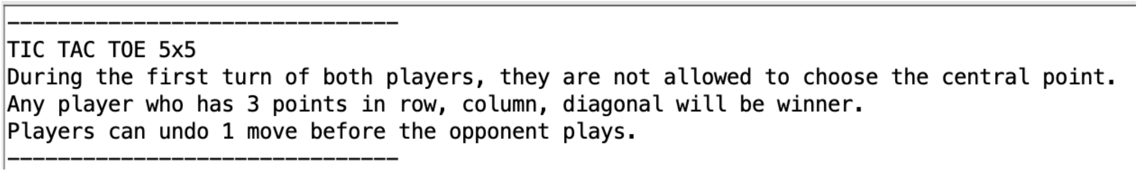
```
Check win function:
 for $i = 0, i < 5$ do
     for $j = 0, j < 5$ do
         if $matrix[i][j] == matrix[i][j+1] == matrix[i][j+2]$ then
             if $Player == 1$ then
                 Print P1 win
             else
                 Print P2 win
             end
         end
     end
end
for $i < 5$ do
     for $j < 5$ do
         if $matrix[i][j] == matrix[i+1][j] == matrix[i+2][j]$ then
             if $Player == 1$ then
                 Print P1 win
             else
                 Print P2 win
             end
         end
     end
end
for $i < 3$ do
     for $j < 3$ do
         if $matrix[i][j] == matrix[i+1][j+1] == matrix[i+2][j+2]$ then
             if $Player == 1$ then
                 Print P1 win
             else
                 Print P2 win
             end
         end
     end
end
for $i < 3$ do
     for $j < 3$ do
         if $matrix[i][j] == matrix[i-1][j-1] == matrix[i-2][j2]$ then
             if $Player == 1$ then
                 Print P1 win
             else
                 print P2 win
             end
         end
     end
end
Call to undo function
```

2.3.7 Undo function

```
Undo function: if $Player == 1$ then
    if $Undo1 == 1$ then
        if $Player\ undo$ then
            matrix[index] = 0
             call to p1 turn
        end
    else
        turn++
         Call to main function
    end
else
    if $Undo2 == 1$ then
        if $Player\ undo$ then
            matrix[index] = 0
             call to p2 turn
        end
    end
    turn++
    Call to main function
end
```

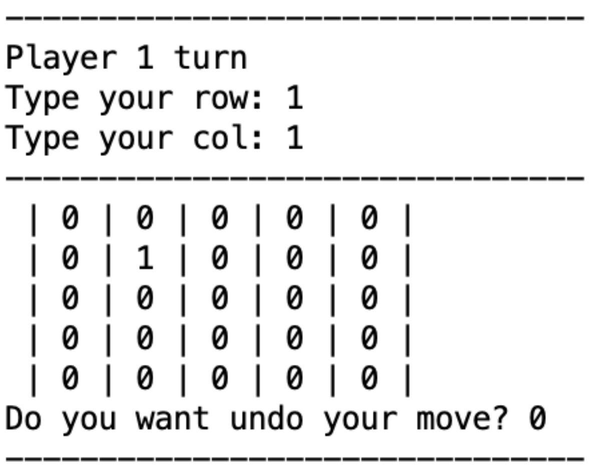
2.4 Demonstration on MARS Simulator

The code will be demonstrate on MARS Simulator, Player 1 will play first then Player 2



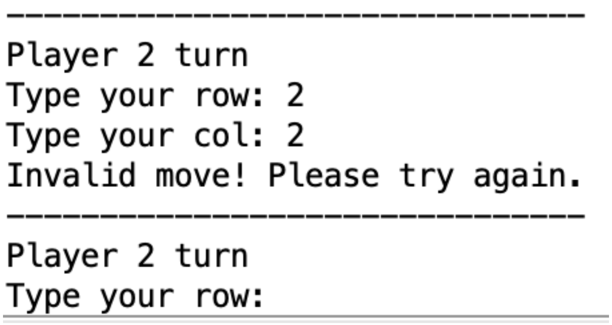
```
-------------------------------
TIC TAC TOE 5x5
During the first turn of both players, they are not allowed to choose the central point.
Any player who has 3 points in row, column, diagonal will be winner.
Players can undo 1 move before the opponent plays.
-------------------------------
```

Figure 4: Welcome to the game intro



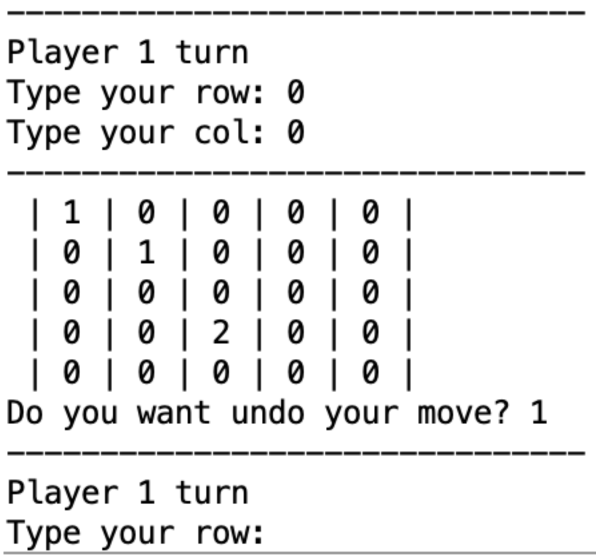
```
-------------------------------
Player 1 turn
Type your row: 1
Type your col: 1
-------------------------------
 | 0 | 0 | 0 | 0 | 0 |
 | 0 | 1 | 0 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
Do you want undo your move? 0
-------------------------------
```

Figure 5: Player turn



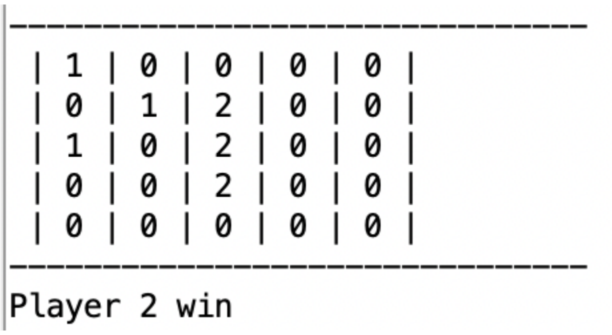
```
-------------------------------
Player 2 turn
Type your row: 2
Type your col: 2
Invalid move! Please try again.
-------------------------------
Player 2 turn
Type your row:
```

Figure 6: Invalid move



```
-------------------------------
Player 1 turn
Type your row: 0
Type your col: 0
-------------------------------
 | 1 | 0 | 0 | 0 | 0 |
 | 0 | 1 | 0 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
 | 0 | 0 | 2 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
Do you want undo your move? 1
-------------------------------
Player 1 turn
Type your row:
```

Figure 7: Undo move



```
-------------------------------
 | 1 | 0 | 0 | 0 | 0 |
 | 0 | 1 | 2 | 0 | 0 |
 | 1 | 0 | 2 | 0 | 0 |
 | 0 | 0 | 2 | 0 | 0 |
 | 0 | 0 | 0 | 0 | 0 |
-------------------------------
Player 2 win
```

Figure 8: Win announcement

3 Conclusion

Through the assignment I have encountered the following difficulties:

- How to design UI for User in MIPS
- How to translate algorithm into assembly language to find alternative way to code objects like pointers and 2D array into MIPS

Beside the aforementioned difficulties, I have also conclude some facts and lessons in building application in MIPS assembly:

- Assembly language helps in providing full control of what tasks a computer is performing.
- Assembly allows the programmer to consider how the underlying hardware operates with each machine instruction they write.

- Assembly show how data is represented in memory.
- It takes a lot of time and effort to write the code compare to high level programming language.