HO CHI MINH UNIVERSITY OF TECHNOLOGY

COMPUTER ARCHITECTURE

REPORT FOR ASSIGNMENT

Topic: Design and write MIPS assembly language for implementing a text-based 5x5 board Tic-Tac-Toe game for two players

Full name: Hoang Do Phuong Nguyen
Student ID: 1952360
Class: 1952360

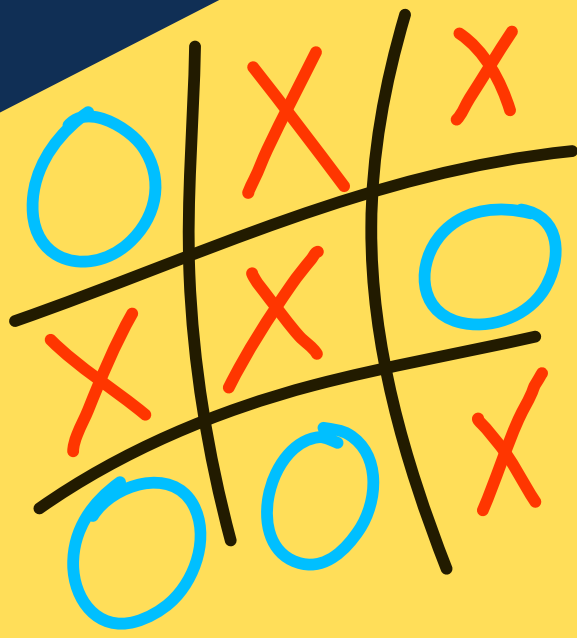

Table of contents

Click here to enter text.

A. Idea

The idea is that an array named “board” containing 25 elements is created. Then, value 0 is set into each element of the “board” array. There is a convention that when player 1 input move into a position, value 1 will be set in that position and when player 2 input move into a position, value 2 will be set in that position, positions are numbered from 1 to 25 so players have to input the positions in the 1-25 interval.

B. Rules to play

1. Player 1 will take x and go first, player 2 will take o and go later.
2. During the first turn of both players, they are not allowed to choose the central point (row 3 & column 3).
3. Any player who has 3 points in a row, column or diagonal will be the winner.
4. Players can undo 1 move before the opponent plays.
5. Each player chooses their desired position based on number of each cell numbered as the board shown in part C.

C. Playing process

At first, rules and a sample board which is numbered in each cell is shown to the screen, which helps players understand the rules clearly and know the position well. These will be shown as below:

```
Welcome to Tic Tac Toe.
Please read the rules below:
1. Player 1 will take x and go first, player 2 will take o and go later.
2. During the first turn of both players, they are not allowed to choose the central point (row 3 & column 3).
3. Any player who has 3 points in a row, column or diagonal will be the winner.
4. Players can undo 1 move before the opponent plays.
5. Each player chooses their desired position based on number of each cell numbered as the board below.

-----+-----+-----+-----+-----
  1  |  2  |  3  |  4  |  5
-----+-----+-----+-----+-----
  6  |  7  |  8  |  9  |  10
-----+-----+-----+-----+-----
  11 |  12 |  13 |  14 |  15
-----+-----+-----+-----+-----
  16 |  17 |  18 |  19 |  20
-----+-----+-----+-----+-----
  21 |  22 |  23 |  24 |  25
-----+-----+-----+-----+-----
```

After that, one more board is shown for players to know the space in which their moves will be stored:

```
 --| --| --| --| --|
--| --| --| --| --|
--| --| --| --| --|
--| --| --| --| --|
--| --| --| --| --|
```

Then, each player will respectively input his/her move into the board, after inputting each move, the player is asked if they want to change the move which has just been entered like this:

```
Player 1, please choose the desired space (1-25): 1
Would you  like to change move? Yes = 1, No = 0, please choose: 1
Choose again (1-25): 8
 --| --| --| --| --|
 --| --| x| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
Player 2, please choose the desired space (1-25): 2
Would you  like to change move? Yes = 1, No = 0, please choose: 0
 --| o| --| --| --|
 --| --| x| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
```

The functions to implement the 4th rule above are:

```
changeMove1:
li $v0, 4
la $a0, change
syscall
li $v0,5
syscall
addi $t1, $t1, -1
beq $v0, 0, out_turn1
li $v0, 4
la $a0, promptChange
syscall
j Input1
```

```
changeMove2:
li $v0, 4
la $a0, change
syscall
li $v0, 5
syscall
subi $t1, $t1, 1
beq $v0, 0, out_turn2
li $v0, 4
la $a0, promptChange
syscall
j Input2
```

Moreover, to check the 2nd rule above, these functions are used:

```
check_first_move:
beq $t0, 13, remove_first_move
beq $t2, 1, after_check_first_move1
j after_check_first_move2

remove_first_move:
beq $t2, 1, invalidMove1
beq $t2, 2, invalidMove2
```

In the function **`check_first_move`**, register `$t0` which stores the first move of each player will be compared with the value 13 (position at row 3 and column 3), if they are equal means that first move is invalid, it will jump to **`remove_first_move`** to conduct remove the first invalid move. If the first move is valid, it will jump to 2 other functions to continue inputting moves.

In the function **`remove_first_move`**, register `$t2` stores the value representing the mark X and O (1 for X, 2 for O). As mentioned before, player 1 will be X and player 2 will be O, so when `$t2` equals to 1, it jumps to the functions that prompt player 1 choose another position. Player 2 will do the same.

Here are 2 functions that will announce to each player that his/her move is invalid and prompt his/her to choose another valid move.

```
invalidMove1:
li $v0, 4
la $a0, invalid
syscall
j Input1

invalidMove2:
li $v0,4
la $a0, invalid
syscall
j Input2
```

```
Player 1, please choose the desired space (1-25): 13
Invalid move, choose another: 2
Would you  like to change move? Yes = 1, No = 0, please choose: 0
 --| x| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
Player 2, please choose the desired space (1-25): 13
Invalid move, choose another: 3
Would you  like to change move? Yes = 1, No = 0, please choose: 0
 --| x| o| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
 --| --| --| --| --|
```

Beside that, invalid move means it is less than 1 and greater than 25 (because the board 5x5 is numbered from 1 to 25). Hence, these functions will check for this offence.

```
check_valid:
beq $t2, 1, check_move
blt $t0, 1, invalidMove2
bgt $t0, 25, invalidMove2
j after_check_move2

check_move:
blt $t0, 1, invalidMove1
bgt $t0, 25, invalidMove1
j after_check_move1
```

In **check_valid**, register `$t2` still represents for player and if it equals to 1, it will jumps to **check_move** to check the validity of the player 1's moves. Register `$t0` in these 2 functions stores the position of each player's move. When it is less than 1 or greater than 25, it will jump to 2 functions **invalidMove1** and **invalidMove2** respect to player 1 and player 2.

```
Player 1, please choose the desired space (1-25): 26
Invalid move, choose another: 1
Would you  like to change move? Yes = 1, No = 0, please choose: 0
 x| - | - | - | - |
 - | - | - | - | - |
 - | - | - | - | - |
 - | - | - | - | - |
 - | - | - | - | - |
Player 2, please choose the desired space (1-25): 102
Invalid move, choose another: 5
Would you  like to change move? Yes = 1, No = 0, please choose: 0
 x| - | - | - | o|
 - | - | - | - | - |
 - | - | - | - | - |
 - | - | - | - | - |
 - | - | - | - | - |
```

After these examinations, a move will be stored and based on the 3rd rule, when player 1 input 3 moves, we will check whether the game has the winner or not.

D. Examine the result

1. *Check row:*

```
check_row:
la $a1, board
addi $t1, $zero, 1
addi $t0, $zero, 1
result_row:
lw $t5, 0($a1)
lw $t6, 4($a1)
lw $t7, 8($a1)
bne $t5, $t6, check_row_again
bne $t6, $t7, check_row_again
bne $t6, 0, winner
check_row_again:
beq $t0, 15, check_column # t0: counter to count cases can win according to row
beq $t1, 3, next_row # 1 hang toi da 3 truong hop co the thang ( 123,234,345)
addi $t0, $t0, 1
addi $a1, $a1, 4
addi $t1, $t1, 1 # t1: counter to count winning cases
j result_row
next_row: # change to next row
addi $t1, $zero, 1
addi $t0, $t0, 1
addi $a1, $a1, 12
j result_row
```

The function **`check_row`** is to check the result according to each row, it will check according to the positions as the below board:

0	4	8	12	16
20	24	28	32	36
40	44	48	52	56
60	64	68	72	76
80	84	88	92	96

Value of each cell is loaded into 3 register `$t5`, `$t6`, `$t7` to check if they are the same or not. Because its origin is an array, so we have to load its position which are spaced 4 bits apart. However, if 2 adjacent positions are different, it jumps to **`check_row_again`** to re-check the result.

In **`check_row_again`**, register `$t0` counts all cases that can win in a row and when it reaches the all winning cases *(cases can win in a row is 0,4,8 / 4,8,12 / 8,12,16 / 20,24,28 / 24,28,32 / 28,32,36 / 40,44,48 / 44,48,52 / 48,52,56 / 60,64,66 / 64,68,72 / 68,72,76 / 80,84,88 / 84,88,92 / 88,92,96*), it jumps to check column's result. Register `$t1` stores the number of winning cases in each row, each row has at most 3 winning cases so when `$t1` is equal to 3, it jumps to **`next_row`** to check the next row. The procedure repeats until `$t0` reaches value 15 and there is no winning case in row, the column will be checked next.

2. Check column:

```
check_column:
la $a1, board
addi $t1, $zero, 1
addi $t0, $zero, 1
result_col:
lw $t5, 0($a1)
lw $t6, 20($a1)
lw $t7, 40($a1)
bne $t5, $t6, check_col_again
bne $t6, $t7, check_col_again
bne $t6, 0, winner
check_col_again:
beq $t0, 15, check_left_diagonal
beq $t1, 5, next_col
addi $t0, $t0, 1
addi $t1, $t1, 1
addi $a1, $a1, 4
j result_col
next_col:
addi $t1, $zero, 1
addi $a1, $a1, 4
j result_col
```

The procedure to check column is almost the same as checking row. The difference is the positions in the array. Column also has at most 15 winning cases: *0,20,40 / 20,40,60 / 40,60,80 / 4,24,44 / 24,44,64 / 44,64,84 / 8,28,48 / 28,48,68 / 48,68,88 / 12,32,52 / 32,52,72 / 52,72,92 / 16,36,56 / 36,56,76 / 56,76,96.*

0	4	8	12	16
20	24	28	32	36
40	44	48	52	56
60	64	68	72	76
80	84	88	92	96

When counter `$t0` reaches the maximum number of winning cases and there is no winning case in column, it jumps to check diagonal from left to right.

3. Check diagonal from left to right:

```
check_left_diagonal: #check diagonal from left to right
la $a1, board
addi $t1, $zero, 1
addi $t0, $zero, 1
result_Ldiagonal:
lw $t5, 0($a1)
lw $t6, 24($a1)
lw $t7, 48($a1)
bne $t5, $t6, check_Ldiagonal_again
bne $t6, $t7, check_Ldiagonal_again
bne $t6, 0, winner
check_Ldiagonal_again:
beq $t0, 9, check_right_diagonal
beq $t1, 3, next_Ldiagonal
addi $t0, $t0, 1
addi $t1, $t1, 1
addi $a1, $a1, 4
j result_Ldiagonal
next_Ldiagonal:
addi $t0, $t0, 1
addi $t1, $zero, 1
addi $a1, $a1, 12
j result_Ldiagonal
```

The procedure to check left diagonal is almost the same as checking row and column. However, there are at most 9 winning cases in diagonal: *0,24,48 / 4,28,52 / 8,32,56 / 20,44,68 / 24,48,72 / 28,52,76 / 40,64,88 / 44,68,92 / 48,72,96.*

0	4	8	12	16
20	24	28	32	36
40	44	48	52	56
60	64	68	72	76
80	84	88	92	96

When counter `$t0` reaches the maximum number of winning cases and there is no winning cases in left diagonal, it jumps to check diagonal from right to left.

4. Check diagonal from right to left:

```
check_right_diagonal:  # check diagonal from right to left
addi $t1, $zero, 1
la $a1, board
addi $t0, $zero, 1
result_Rdiagonal:
lw $t5, 8($a1)
lw $t6, 24($a1)
lw $t7, 40($a1)
bne $t5, $t6, check_Rdiagonal_again
bne $t6, $t7, check_Rdiagonal_again
bne $t6, 0, winner
check_Rdiagonal_again:
beq $t0, 15, if_tie
beq $t1, 3, next_Rdiagonal
addi $t0, $t0, 1
addi $t1, $t1, 1
addi $a1, $a1, 4
j result_Rdiagonal
next_Rdiagonal:
addi $t0, $t0, 1
addi $t1, $zero, 1
addi $a1, $a1, 12
j result_Rdiagonal
```

The procedure to check right diagonal is almost the same as checking row, column and left column. However, there are at most 9 winning cases in diagonal: *8,24,40 / 12,28,44 / 16,32,48 / 28,44,60 / 32,48,64 / 48,64,80 / 36,52,68 / 52,68,84 / 56,72,88*

0	4	8	12	16
20	24	28	32	36
40	44	48	52	56
60	64	68	72	76
80	84	88	92	96

When counter `$t0` reaches the maximum number of winning cases and there is no case winning in right diagonal, this means that there is no winner, this is a tie game, so it jumps to **`if_tie`** to check and announce to 2 players that the game is a tie.

E. Determine the winner

As can be seen in functions **`check_row_again`**, **`check_column_again`**, **`check_Ldiagonal_again`**, **`check_Rdiagonal_again`**, there is an instruction `bne $t6, 0, winner`. Because register `$t6` stores the value in the position which each player chooses, at first it is agreed that when player 1 input move, it will change to 1 to store in the board and when player 2 input move, it will change to 2 to store in the board,

so when a position is not chosen, it will be 0. Therefore, if a position is not a zero, we will jump to check for the winner.

```
winner:
li $v0, 4
la $a0, new_line
syscall
beq $a2, 1, player2_win
li $v0, 4
la $a0, win_1
syscall
j exit

player2_win:
li $v0, 4
la $a0, win_2
syscall
j exit
```

Function **`winner`** will determine who is the winner, register `$a0` is a flag to give the signal that if `$a0` is equal to 1 representative for player 1, it jumps to function **`player2_win`** to announce that player 2 wins and on the contrary, it announces that player 1 wins. Otherwise, 2 functions **`if_tie`** and **`check_tie`** are used to examined if it is a tie game.

However, when concluding that a game is tie, it have to be examined whether all the cell in the board are filled or not. If yes, after checking all winning cases and there is no winner, the game is concluded a tie game. To check tie game, 2 functions are used:

```
if_tie:
la $a1, board
addi $t8, $zero, 1
check_tie:
lw $t7, 0($a1)
beq $t7, 0, done
beq $t8, 25, tieGame
addi $t8, $t8, 1
addi $a1, $a1, 4
j check_tie
```

Finally, function **`tieGame`** is to announce to players know that this is a tie game.

```
tieGame:
li $v0, 4
la $a0, new_line
syscall
li $v0, 4
la $a0, tie
syscall
```

-The end-