Ho Chi Minh City University of Technology

Faculty of Computer Science and Engineering

Computer Architecture-CO2007

Assignment report

Topic: Text-based 5x5 board Tic-Tac-Toe game for two players.

Student name: Le Nhat Dang

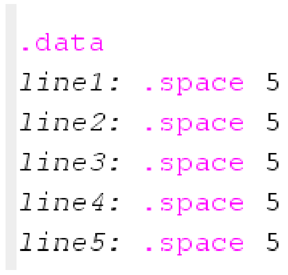
Student ID: 2052950

Class: CC02

I. Outline

- In this tic-tac-toe program, we can separate the program into two parts which is the .data section(declaring the global variables) and .text section(contains the code to run this program).

- In the .data section, some strings are used to print out the instructions, proclaim the winner, and request input data. This program also have five character arrays to create the board and store the data. Since this is a 5x5 board tic-tac-toe, the size of each array is five bytes and the name of these arrays are line1, line2, line3, line4, line5.



```
.data
line1: .space 5
line2: .space 5
line3: .space 5
line4: .space 5
line5: .space 5
```

Declaring five character arrays

- In the .text section, there are several vital function to run the program such as the play1_input and play_2input function for inputing and storing data, the win_condition fuction to check if anyone won this game and the start_the_game function to run the game.

II. How does the program work

In the main program:

- At first, the register $s6 and $s6 are assigned to 2 with purpose(discuss in undo function).

- Next, the game_instruction function is called to print out the rules and notes for players before playing the game.

- After that, init_game and print_tic_tac_toe functions are used to create the board and print out the board.

- Finally, the start_the_game function is used to implement the game. Inside this function, there are some functions that are used to input data, check the winner, check invalid input, and undo a move.

III. How game_instruction, init_game, print_tic_tac_toe functions work

- Game_instruction: print out the rules, requirements of the game and some notes for player with the strings ared declared in .data section

- Init_game:

+ Load address of the arrays into registers. $s0-$s4 store the address of arrays line1-line5.

+ Assign register $t0 to 45 with represent for character '-'.

+ Make a loop by using beq(conditional branch instruction) for storing value of $t0 into each element of the arrays by using sb(store instruction).

- Print_tic_tac_toe:

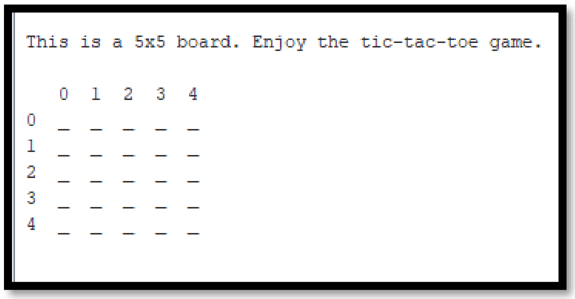
+ Reloading the address of arrays line1-line5 to registers $s0-$s4

+ For the first line, print out the index of the colomn, starting with 0 and end with 4.

+ For the second line, print out the row index which is 0. Then using beq(conditional brach instrucion) inorder to print out each element of $s0, which is the first row of the board.

+ For third lind to last line, same as second line but the row index is 1-4 and print out each element of $s1-$s4.

+ Inside each conditional bracnh instruction, load instruction are used to load value of each element of an array to $t0 register. After that, move content of $t0 register to $a0 register and display the value.



```
This is a 5x5 board. Enjoy the tic-tac-toe game.

   0  1  2  3  4
0  _  _  _  _  _
1  _  _  _  _  _
2  _  _  _  _  _
3  _  _  _  _  _
4  _  _  _  _  _
```

Result after executed init_game and print_tic_tac_toe functions

IV. Start_the_game function

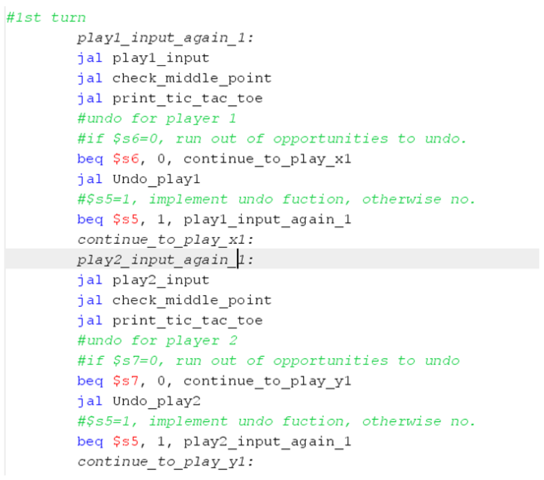
- This function can be separated into three parts.

- First part: The player is required to input data for the first turn provided that the player cannot mark in the middle point of the board by using play1_input, play2_input and check_middle_point functions. After inputting data, the new board will be printed out. Finally, the player can use their undo chances if they want. Players won't be able to undo if they ultilize all of their undo opportunities

- Second part: Making a loop by using beq(conditional branch instruction) for inputting data from second turn to twelfth turn. In each turn, after entering data by using play1_input and play2_input functions, the new board will be printed out and the win_condition function is called to check which player has 3 consecutive points in row, column, diagonal first(winning condition). If the player meets about conditions, that player wins the game. Otherwise, the game continues. Remember that the player can undo as long as they still have chance to undo. This process is stopped after finishing the twelfth turn.

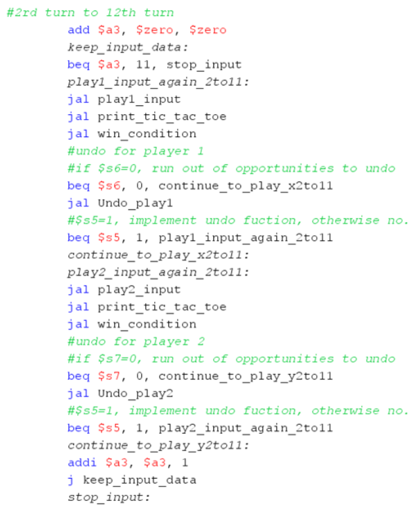
- Third part: After twelfth turn, 24 points on the board have been marked. Therefore, only player1 can input to the last point since player1 go first. Then, the win condition function is used to determine whether or not player1 can win the game. If this is the case, the game will terminate with player 1 as the winner. If not, the game will end in a draw.

- In this function, because it does not need to return to the caller, no need to use stack.



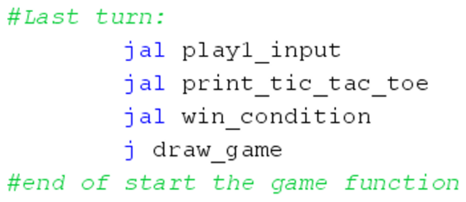
```
#1st turn
        play1_input_again_1:
        jal play1_input
        jal check_middle_point
        jal print_tic_tac_toe
        #undo for player 1
        #if $s6=0, run out of opportunities to undo.
        beq $s6, 0, continue_to_play_x1
        jal Undo_play1
        #$s5=1, implement undo fuction, otherwise no.
        beq $s5, 1, play1_input_again_1
        continue_to_play_x1:
        play2_input_again_1:
        jal play2_input
        jal check_middle_point
        jal print_tic_tac_toe
        #undo for player 2
        #if $s7=0, run out of opportunities to undo
        beq $s7, 0, continue_to_play_y1
        jal Undo_play2
        #$s5=1, implement undo fuction, otherwise no.
        beq $s5, 1, play2_input_again_1
        continue_to_play_y1:
```

First part of start_the_game function



```
#2rd turn to 12th turn
        add $a3, $zero, $zero
        keep_input_data:
        beq $a3, 11, stop_input
        play1_input_again_2to11:
        jal play1_input
        jal print_tic_tac_toe
        jal win_condition
        #undo for player 1
        #if $s6=0, run out of opportunities to undo
        beq $s6, 0, continue_to_play_x2to11
        jal Undo_play1
        #$s5=1, implement undo fuction, otherwise no.
        beq $s5, 1, play1_input_again_2to11
        continue_to_play_x2to11:
        play2_input_again_2to11:
        jal play2_input
        jal print_tic_tac_toe
        jal win_condition
        #undo for player 2
        #if $s7=0, run out of opportunities to undo
        beq $s7, 0, continue_to_play_y2to11
        jal Undo_play2
        #$s5=1, implement undo fuction, otherwise no.
        beq $s5, 1, play2_input_again_2to11
        continue_to_play_y2to11:
        addi $a3, $a3, 1
        j keep_input_data
        stop_input:
```

Second part of start_the_game function



```
#Last turn:
        jal play1_input
        jal print_tic_tac_toe
        jal win_condition
        j draw_game
#end of start the game function
```

Last part of start_the_game function

V. Two input functions and check_middle_point function

- There are two input funcions in this program which is the play1_input and play2_input.

- Play1_input: player 1 marks 'X' into the board. This function in separated into two parts which is input the data and store the data.

Input data:

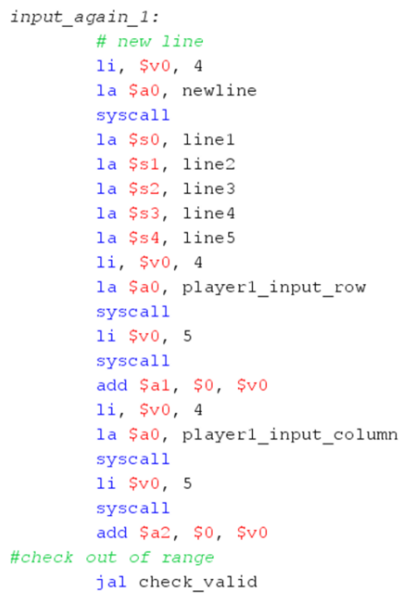
- At first, as this function call some other functions so storing the address where play1_input function is called into stack is needed.

- Then, Reloading the address of arrays line1-line5 to registers $s0-$s4.

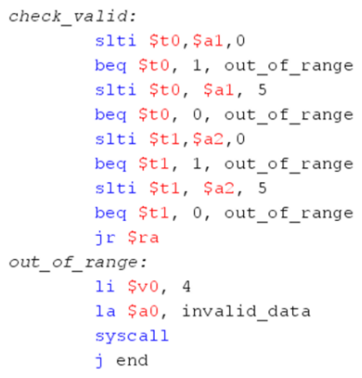
- After that, the strings ‘Player1, please enter the row:’ and ‘Player1, please enter the column:’ will be printed out. Player must enter the row index and column index for marking a position in the board. Remember that the index of row 1 is 0, row 2 is 3 and so on. The column index also follow that order. It means that player must enter an index which is in the range of 0 to 4. For example, inorder to mark in the top-left position, the row index is 0 and it also the same with the column index. Since the top-left position in column 1 and row 1, which represent for row index 0 and column index 0. The bottom right position has row index is 4 and column index is 4.

- Next, the value of row index is stored in $a1 register and $a2 register stores the value of column index.

- Then, the check_valid function is called. Since $a1 stores the value of row index and $a2 stored the value of column index, if $a1or $a2 is less than 0 and greater than 4 and the program will print out the string ‘Player 1 or player 2 enter an invalid index for row or column’ and end the program. Otherwise, the program continues with the next instruction. That if statement can be maked by using two slti instruction.



```
input_again_1:
        # new line
        li, $v0, 4
        la $a0, newline
        syscall
        la $s0, line1
        la $s1, line2
        la $s2, line3
        la $s3, line4
        la $s4, line5
        li, $v0, 4
        la $a0, player1_input_row
        syscall
        li $v0, 5
        syscall
        add $a1, $0, $v0
        li, $v0, 4
        la $a0, player1_input_column
        syscall
        li $v0, 5
        syscall
        add $a2, $0, $v0
#check out of range
        jal check_valid
```



```
check_valid:
        slti $t0,$a1,0
        beq $t0, 1, out_of_range
        slti $t0, $a1, 5
        beq $t0, 0, out_of_range
        slti $t1,$a2,0
        beq $t1, 1, out_of_range
        slti $t1, $a2, 5
        beq $t1, 0, out_of_range
        jr $ra
out_of_range:
        li $v0, 4
        la $a0, invalid_data
        syscall
        j end
```

The first part of play1_input and check_valid function.

Store data:

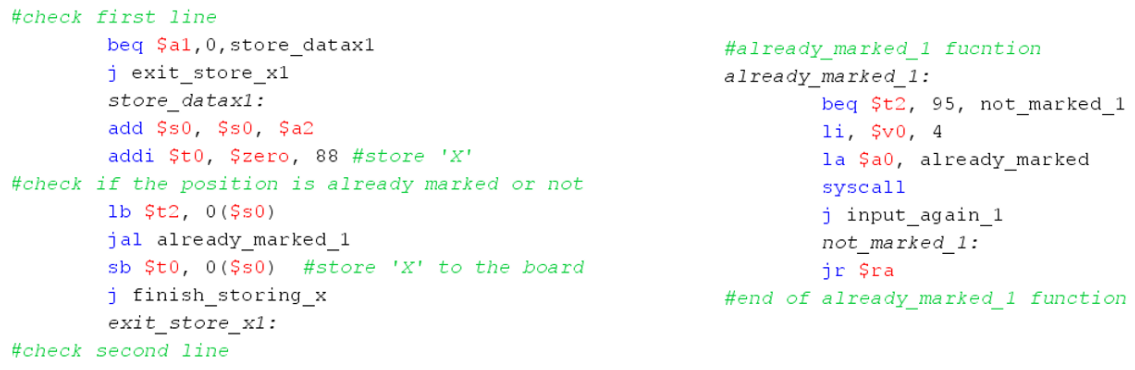
- In the next step, because the board is created by five character arrays so line1 arrays use to storing input of the first row and so on.

- For storing data into the first row, by using beq(conditional branch instruction), if $a1 equal to 0, it means that we need to store 'X' in somewhere in row 1. Since $s0 hold the address of array line1, a position to store data can be determined easily by adding $s0 with $a2. Because $a2 store the value of column index, after excecution the add instruction, $s0 holds the address of the position that player1 want to mark. Then, storing instuction is used to mark 'X' into board.

- Before storing 'X' into the board, it is vital to check that the storing point is marked or not. The already_marked_1 function is used to do that. Before calling this function, loading data in the position that play1 want to mark to register $t2 is required by using lb $t2, 0($s0). Then inside the function, beq is used to compare the value of $t2 to 45(ascii of '-'). If they not are equal, jump to the input stage to enter the row and column index again.

Otherwise, return to address where this function was called. Then assigin value of $t0 to 88(ascii of 'X') and using sb $t0, 0($s0) to store 'X' into the board.

- With row 2 to row 5, the same way. Remember that $s1 hold the address of array line2, $2 for line3, $s3 for line4 and $s4 for line5.



```
#check first line
        beq $a1,0,store_datax1
        j exit_store_x1
        store_datax1:
        add $s0, $s0, $a2
        addi $t0, $zero, 88 #store 'X'
#check if the position is already marked or not
        lb $t2, 0($s0)
        jal already_marked_1
        sb $t0, 0($s0)  #store 'X' to the board
        j finish_storing_x
        exit_store_x1:
#check second line
```

```
#already_marked_1 fucntion
already_marked_1:
        beq $t2, 95, not_marked_1
        li, $v0, 4
        la $a0, already_marked
        syscall
        j input_again_1
        not_marked_1:
        jr $ra
#end of already_marked_1 function
```

The second part of play1_input and already_marked_1.

- Play2_input: player 2 marks '0' into the board.

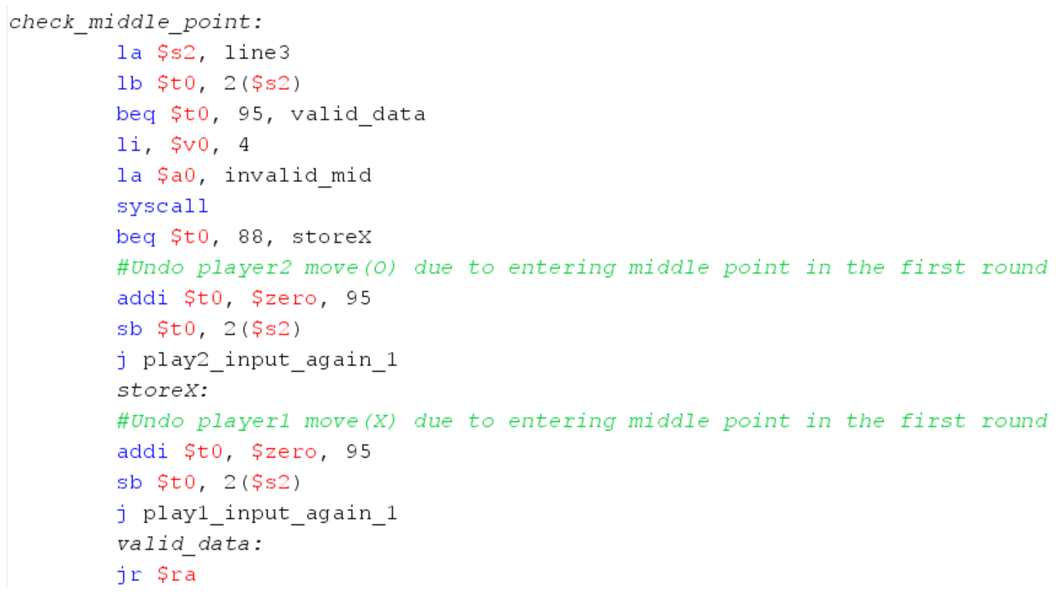
The same logic is apllied for play2_input function. However, this function will store 'O' into the position that player2 want to mark into the board by assign $t0 to 79(ascii of 'O'). In play2_input function, the already_marked_2 function is used instead of already_marked_1 since it jumps to player2 input stage.

- Check_middle_point function:

In this fuction, since the middle point has the row index is to and column index is 2, reloading the address of $s2(row3) is needed. Then loading the value in middle point to register $t0 by using lb $t0, 2($s2).

Using beq to check the value of $t0. If $t0 is equal to 88(ascii of 'X'), print out the string "Cannot choose the MIDDLE position in the first round" and assign $t0 to 45(ascii of '-') then store the value of $t0 to the middle point. After that, jump back to the step for player 1 input data for the first turn.

If $t0 is equal to 79 (ascii of 'O'), do the same process but jump back to the step for player 2 input again. Otherwise, do nothing and continue the program.



```
check_middle_point:
        la $s2, line3
        lb $t0, 2($s2)
        beq $t0, 95, valid_data
        li, $v0, 4
        la $a0, invalid_mid
        syscall
        beq $t0, 88, storeX
        #Undo player2 move(O) due to entering middle point in the first round
        addi $t0, $zero, 95
        sb $t0, 2($s2)
        j play2_input_again_1
        storeX:
        #Undo player1 move(X) due to entering middle point in the first round
        addi $t0, $zero, 95
        sb $t0, 2($s2)
        j play1_input_again_1
        valid_data:
        jr $ra
```

check_middle_point fucntion.

VI. Win-condition function

Inside the win_condtion function, three other funcitons is called to check which player wins the game. Therefore, loading the return address into the stack is needed. The row_check, column_check, diagonal_check functions are called respectively.

- Row_check:

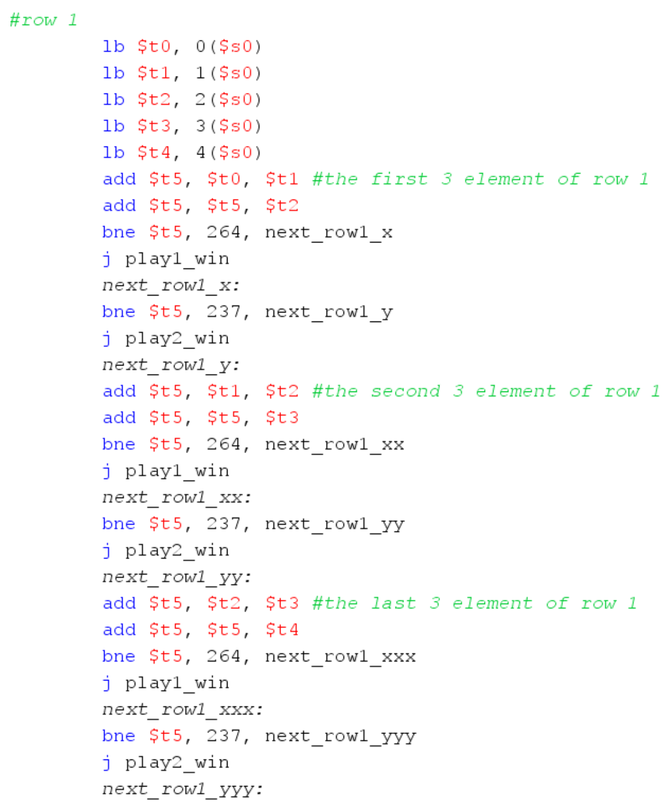
+Reloading the address of arrays line1-line5 to registers $s0-$s4.

+Checking if player 1 or player 2 has 3 consecutive 'X' or 'O' in row or not. Starting from the first row to the last row.

+First row: Since every position in the first row of the board is the element of array line1($s0 hold the address), loading the first element into $t0 register, second into $t1 register by using lb $t0, 0($s0) and lb $t1, 1($s0) and so on. So the value of $t0 to $t4 store the value of each element in $s0.

+Then adding the first three element($t0+$t1+$t2) of the array to $t5. If $t5 is equal to 264(88+88+88), player 1 wins the game since. If $t5 is equal to to 237(79+79+79), player 2 wins the game. Otherwise, check the next three element element. At this time $t5 is equal to ($t1 +$t2 + $3). If no one win the game, check the last three elements. If the winner cannot be determined in the first row, move to the next row.

+The same logic is applied to row2 to row5. Remember that $s1 hold the address of array line2, $s2 for array line3 and so on. After finish checking row5 and no one win the game, return to the address that row_check function was called.



```
#row 1
        lb $t0, 0($s0)
        lb $t1, 1($s0)
        lb $t2, 2($s0)
        lb $t3, 3($s0)
        lb $t4, 4($s0)
        add $t5, $t0, $t1 #the first 3 element of row 1
        add $t5, $t5, $t2
        bne $t5, 264, next_row1_x
        j play1_win
        next_row1_x:
        bne $t5, 237, next_row1_y
        j play2_win
        next_row1_y:
        add $t5, $t1, $t2 #the second 3 element of row 1
        add $t5, $t5, $t3
        bne $t5, 264, next_row1_xx
        j play1_win
        next_row1_xx:
        bne $t5, 237, next_row1_yy
        j play2_win
        next_row1_yy:
        add $t5, $t2, $t3 #the last 3 element of row 1
        add $t5, $t5, $t4
        bne $t5, 264, next_row1_xxx
        j play1_win
        next_row1_xxx:
        bne $t5, 237, next_row1_yyy
        j play2_win
        next_row1_yyy:
```

A part of row_check function.

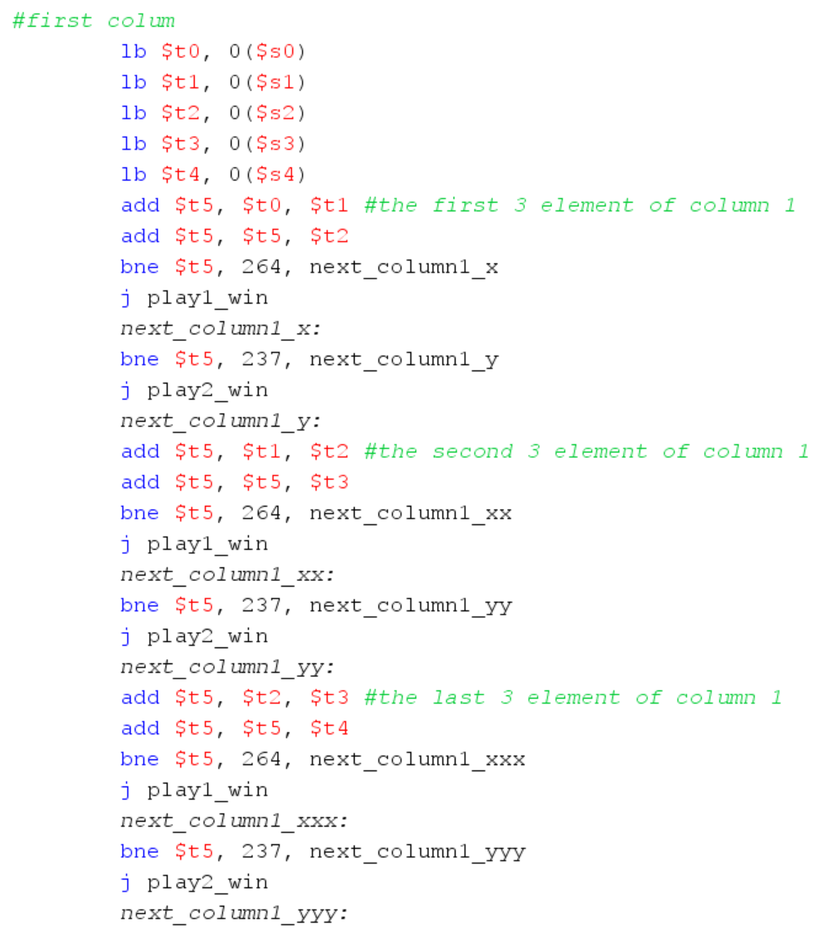
- Column_check:

- Reloading the address of arrays line1-line5 to registers $s0-$s4.

- Checking if player 1 or player 2 has 3 consecutive 'X' or 'O' in column or not. Starting from the first column to the last column.

- First column: Every position in the first column has column index is 0. Therfore, $t0 has the value of the first element in $s0 by using lb $t0, 0($s0), $t1 has the value of the first element in $s1 by using lb $t0, 0($s1) and so on. After loading, by using the method which is utilized in row_check function. Checking the sum of the first three elements($t0+ $t1, $t2), the second three elements($t1+$t2+t3) and the last three elements($t2+$t3+t4). If there is a winner, congrats them and end the program. Otherwise move to the next column

- The same logic is applied to column2 to colummn5. Remember that increment the value of offset by one for accessing data in the second column, two for third column and so on. After finish checking column5 and no one win the game, return to the address that column_check function was called.

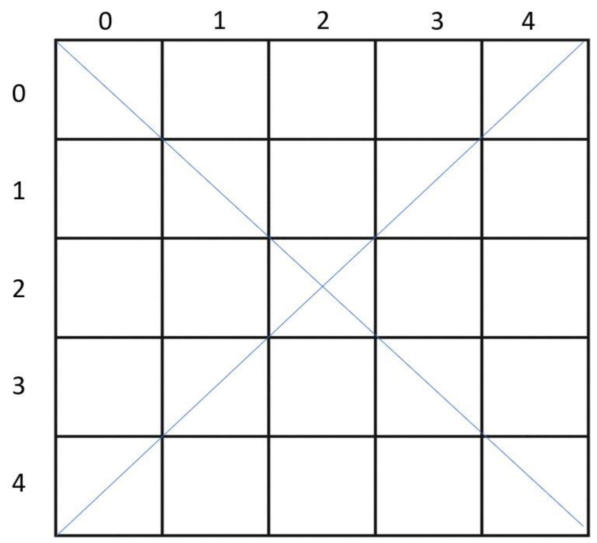


```
#first colum
        lb $t0, 0($s0)
        lb $t1, 0($s1)
        lb $t2, 0($s2)
        lb $t3, 0($s3)
        lb $t4, 0($s4)
        add $t5, $t0, $t1 #the first 3 element of column 1
        add $t5, $t5, $t2
        bne $t5, 264, next_column1_x
        j play1_win
        next_column1_x:
        bne $t5, 237, next_column1_y
        j play2_win
        next_column1_y:
        add $t5, $t1, $t2 #the second 3 element of column 1
        add $t5, $t5, $t3
        bne $t5, 264, next_column1_xx
        j play1_win
        next_column1_xx:
        bne $t5, 237, next_column1_yy
        j play2_win
        next_column1_yy:
        add $t5, $t2, $t3 #the last 3 element of column 1
        add $t5, $t5, $t4
        bne $t5, 264, next_column1_xxx
        j play1_win
        next_column1_xxx:
        bne $t5, 237, next_column1_yyy
        j play2_win
        next_column1_yyy:
```

A part of column_check function.

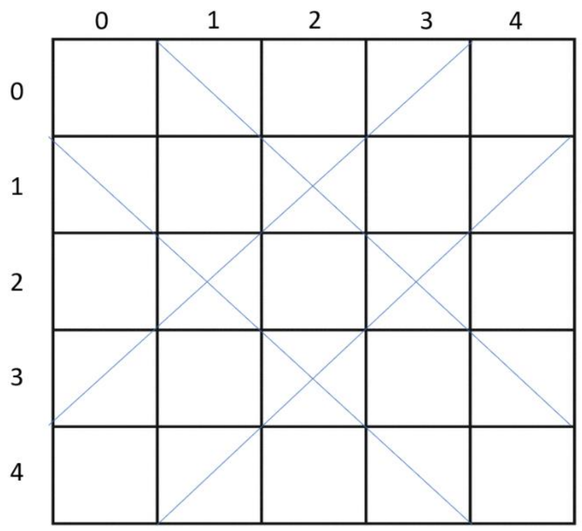
- Diagonal check: There are total 10 diagonal that players can mark and win the game if they have three consecutive 'X' or 'O'.

The first two diagonal with at most five consecutive 'X' or 'O'.



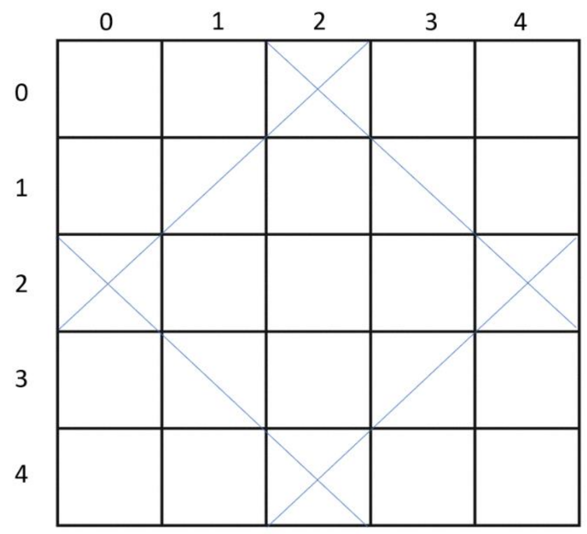
The left to right diagonal pass through the [0][0], [1][1], [2][2], [3][3], [4][4] position. By storing the value of [0][0] to $t0, [1][1] to $t1 and so on. Then, applying the method which is used in row_check and column_check(sum each 3-pair). If no one win the game, do the same thing with right to left diagonal.

The next 4 diagonal with at most four consecutive 'X' or 'O'.



Taking the left to right diagonal starting from row0 to row 3, this diagonal pass through the [0][1], [1][2], [2][3], [3][4] position. By storing the value of [0][1] to $t0, [1][2] to $t1, [2][3] to $t2 and [3][4] to $t3. Then, applying the method which is used in row_check and column_check(sum each 3-pair). If no one win the game, do the same thing with others diagonal.

The last 4 diagonal with exactly three consecutive 'X' or 'O'.



Taking the left to right diagonal starting from row0 to row 2, this diagonal pass through the [0][2], [1][3], [2][4]position. By storing the value of [0][2] to $t0, [1][3] to $t1, [2][4] to $t2. Then, applying the method which is used in row_check and column_check(sum each 3-pair). If no one win the game, do the same thing with others diagonal.

After checking all 10 diagonals, return to the address that call diagonal_check if the winner no one win the game.

VII. How Undo work in this program

In this program, each players can undo 1 move before the opponent plays. Remember that each play can undo twice during the game.

There are two undo functions in this program which is Undo_play1 and Undo_play2.

Register $s6 store the number of times that player 1 can undo and register $s7 store the number of time that player 2 can undo.

The Undo_play1 function is called when $s6 is not equal to zero and Undo_play2 function is called when $s7 is not equal to zero.

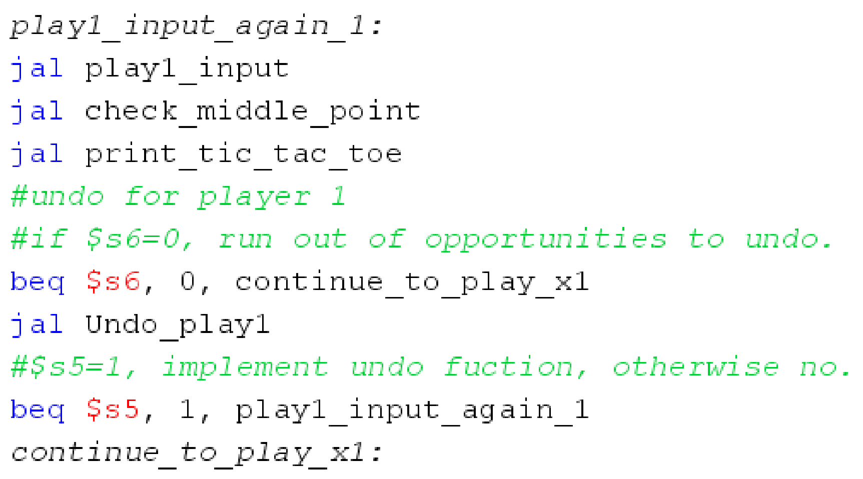
Inside Undo_play1 function:

- At the beginning , the string ‘Player 1, if you want to undo, enter 1 and, else enter any number except 1. Enter number here: ’ is printed out. The value that player 1 inputs is stored into $s5 register. If $s5 not equal to 1, return to the address where Undo_play1 function was called.

- Otherwise, do the undo process. $a1 and $a2 registers still store the data of row index and column index. Therefore, loading '-' into the position corressponding to $a1 and $a2 can be done easily by assign $t0 to 45 and using lb instruction.

- Decrement the value of $s6 to 1 and return to the address where this function was called.

After return to address where Undo_play1 function was called. If $s5 is equal to 1, jump back to the input stage, otherwise, continue the program.



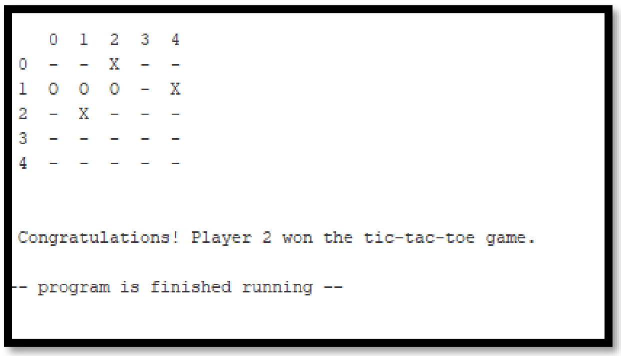
```
play1_input_again_1:
jal play1_input
jal check_middle_point
jal print_tic_tac_toe
#undo for player 1
#if $s6=0, run out of opportunities to undo.
beq $s6, 0, continue_to_play_x1
jal Undo_play1
#$s5=1, implement undo fuction, otherwise no.
beq $s5, 1, play1_input_again_1
continue_to_play_x1:
```

How undo work for player 1.

The Undo process for player 2 is the same as the process for player 1. But the register $s7 and Undo_play2 function are used instead of register $s6 and Undo_play1 function.

The Undo_play2 fucntion accomplishes the same with Undo_play1 function, but it has been modified to work with player 2.

VIII. Possible resuts

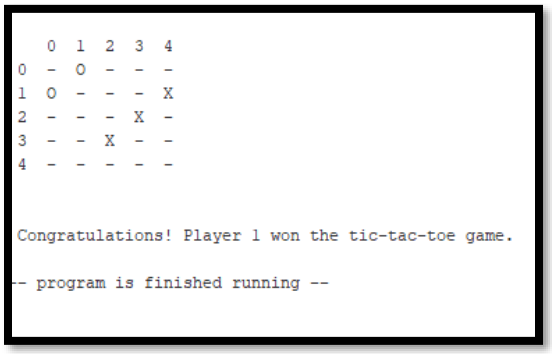


```
   0  1  2  3  4
0  -  -  X  -  -
1  O  O  O  -  X
2  -  X  -  -  -
3  -  -  -  -  -
4  -  -  -  -  -


Congratulations! Player 2 won the tic-tac-toe game.

-- program is finished running --
```

The program end and player 2 won the

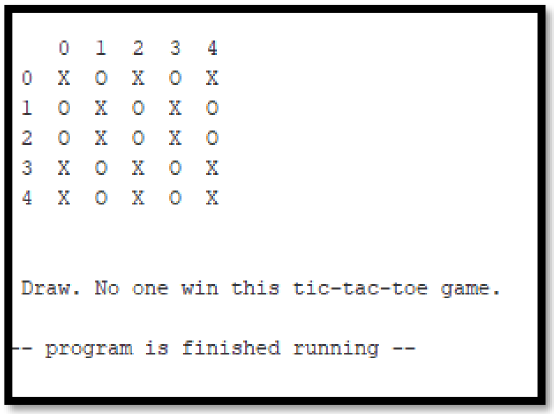


```
   0  1  2  3  4
0  -  O  -  -  -
1  O  -  -  -  X
2  -  -  -  X  -
3  -  -  X  -  -
4  -  -  -  -  -


Congratulations! Player 1 won the tic-tac-toe game.

-- program is finished running --
```

The program end and player 1 won the

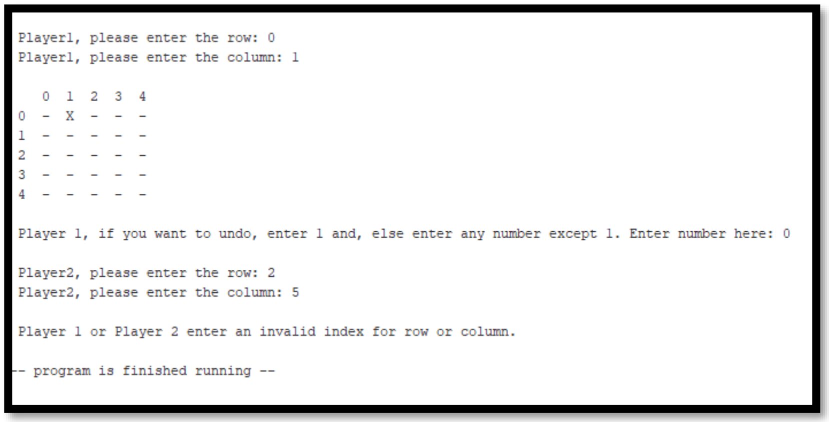


```
   0  1  2  3  4
0  X  O  X  O  X
1  O  X  O  X  O
2  O  X  O  X  O
3  X  O  X  O  X
4  X  O  X  O  X


Draw. No one win this tic-tac-toe game.

-- program is finished running --
```

The program end with draw result.



```
Player1, please enter the row: 0
Player1, please enter the column: 1

   0  1  2  3  4
0  -  X  -  -  -
1  -  -  -  -  -
2  -  -  -  -  -
3  -  -  -  -  -
4  -  -  -  -  -

Player 1, if you want to undo, enter 1 and, else enter any number except 1. Enter number here: 0

Player2, please enter the row: 2
Player2, please enter the column: 5

Player 1 or Player 2 enter an invalid index for row or column.

-- program is finished running --
```

The program end due to invalid input