Project Title

Tic Tac Toe Game in MIPS Assembly language

Introduction

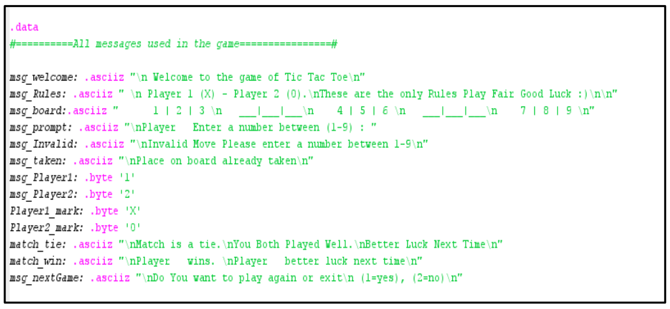
The project was to create a famous game, tic-tac-toe. It is a multiplayer game, where 2 players can play against each other. The player 1 is assigned with the mark (X) and player 2 is assigned with the mark (O). The rules of the game are simple, if any of the player place their mark in a single row, single column, or a diagonal then that player wins the game.

The source code of the game is written in MIPS assembly language using the Mars MIPS simulator.

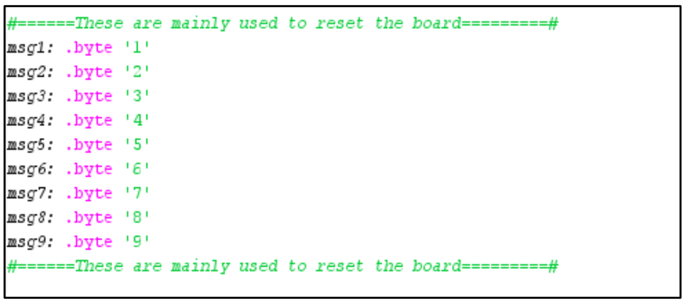
Game Functionalities

.data Segment

The .data segment contains all the messages like rules and prompts that will ask the user to place their mark etc. Also, it contains the marks for the players that they will use in the game. It also contains the board numbers from 1-9 that helps in resetting the board again.



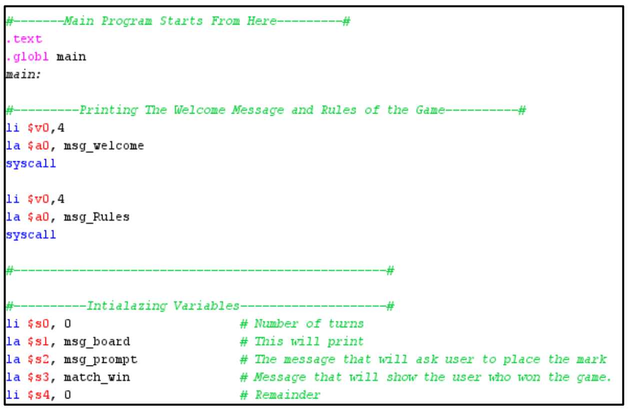
```
.data
#==========All messages used in the game================#

msg_welcome: .asciiz "\n Welcome to the game of Tic Tac Toe\n"
msg_Rules: .asciiz " \n Player 1 (X) - Player 2 (O).\nThese are the only Rules Play Fair Good Luck :)\n\n"
msg_board:.asciiz "      1 | 2 | 3 \n   ___|___|___\n    4 | 5 | 6 \n   ___|___|___\n    7 | 8 | 9 \n"
msg_prompt: .asciiz "\nPlayer   Enter a number between (1-9) : "
msg_Invalid: .asciiz "\nInvalid Move Please enter a number between 1-9\n"
msg_taken: .asciiz "\nPlace on board already taken\n"
msg_Player1: .byte '1'
msg_Player2: .byte '2'
Player1_mark: .byte 'X'
Player2_mark: .byte 'O'
match_tie: .asciiz "\nMatch is a tie.\nYou Both Played Well.\nBetter Luck Next Time\n"
match_win: .asciiz "\nPlayer   wins. \nPlayer   better luck next time\n"
msg_nextGame: .asciiz "\nDo You want to play again or exit\n (1=yes), (2=no)\n"
```



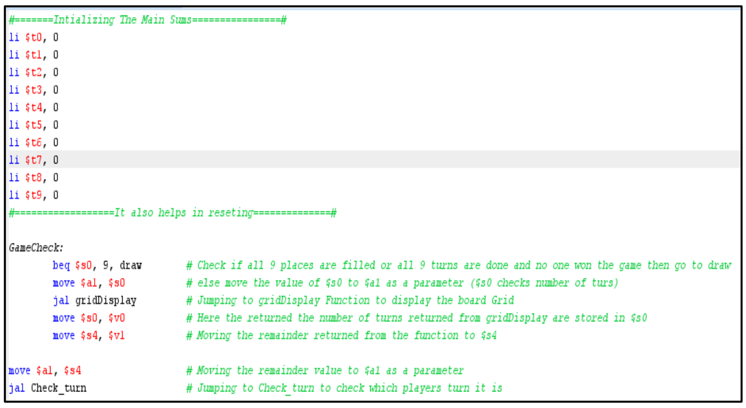
```
#======These are mainly used to reset the board=========#
msg1: .byte '1'
msg2: .byte '2'
msg3: .byte '3'
msg4: .byte '4'
msg5: .byte '5'
msg6: .byte '6'
msg7: .byte '7'
msg8: .byte '8'
msg9: .byte '9'
#======These are mainly used to reset the board=========#
```

Main

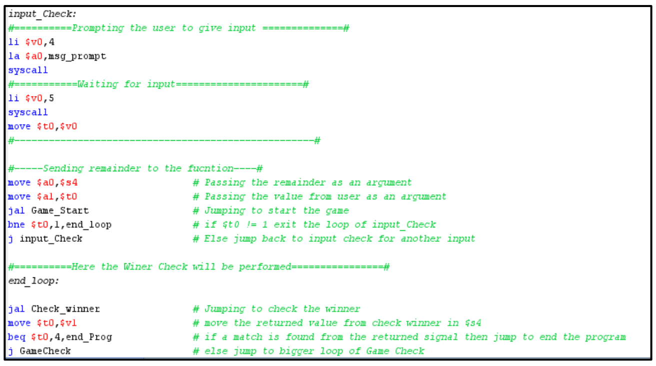
The main is the segment from where the code starts executing the game and all the other segments or functionalities present in the game is called from the main segment. In this segment, we first initialize all the main variables of the game. In $s registers, we load all the things like messages which are needed to remain in permeant use in the game; while $t registers are used to do the sum to calculate the winner of the game.



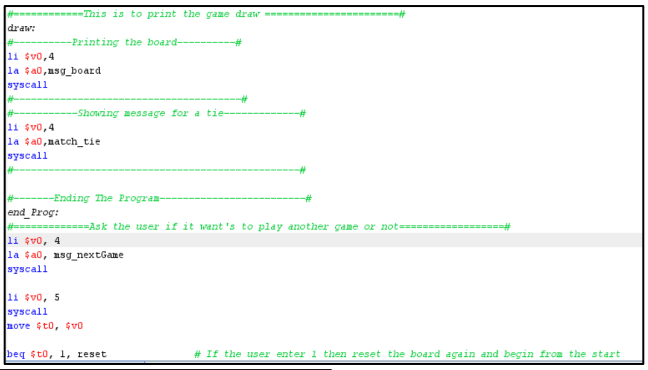
```
#-------Main Program Starts From Here---------#
.text
.globl main
main:

#---------Printing The Welcome Message and Rules of the Game----------#
li $v0,4
la $a0, msg_welcome
syscall

li $v0,4
la $a0, msg_Rules
syscall

#----------------------------------------------------#

#----------Intialazing Variables--------------------#
li $s0, 0                       # Number of turns
la $s1, msg_board               # This will print
la $s2, msg_prompt              # The message that will ask user to place the mark
la $s3, match_win               # Message that will show the user who won the game.
li $s4, 0                       # Remainder
```



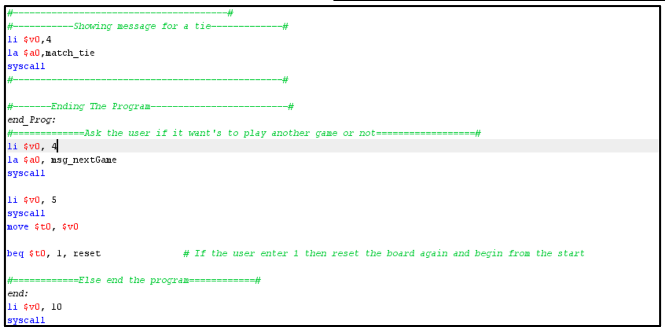
```
#=======Intializing The Main Sums================#
li $t0, 0
li $t1, 0
li $t2, 0
li $t3, 0
li $t4, 0
li $t5, 0
li $t6, 0
li $t7, 0
li $t8, 0
li $t9, 0
#==================It also helps in reseting=============#

GameCheck:
        beq $s0, 9, draw        # Check if all 9 places are filled or all 9 turns are done and no one won the game then go to draw
        move $a1, $s0           # else move the value of $s0 to $a1 as a parameter ($s0 checks number of turs)
        jal gridDisplay         # Jumping to gridDisplay Function to display the board Grid
        move $s0, $v0           # Here the returned the number of turns returned from gridDisplay are stored in $s0
        move $s4, $v1           # Moving the remainder returned from the function to $s4

move $a1, $s4                   # Moving the remainder value to $a1 as a parameter
jal Check_turn                  # Jumping to Check_turn to check which players turn it is
```



```
input_Check:
#==========Prompting the user to give input ==============#
li $v0,4
la $a0,msg_prompt
syscall
#===========Waiting for input======================#
li $v0,5
syscall
move $t0,$v0
#-----------------------------------------------------#

#-----Sending remainder to the fucntion----#
move $a0,$s4                # Passing the remainder as an argument
move $a1,$t0                # Passing the value from user as an argument
jal Game_Start              # Jumping to start the game
bne $t0,1,end_loop          # if $t0 != 1 exit the loop of input_Check
j input_Check               # Else jump back to input check for another input

#==========Here the Winer Check will be performed===============#
end_loop:

jal Check_winner            # Jumping to check the winner
move $t0,$v1                # move the returned value from check winner in $s4
beq $t0,4,end_Prog          # if a match is found from the returned signal then jump to end the program
j GameCheck                 # else jump to bigger loop of Game Check
```



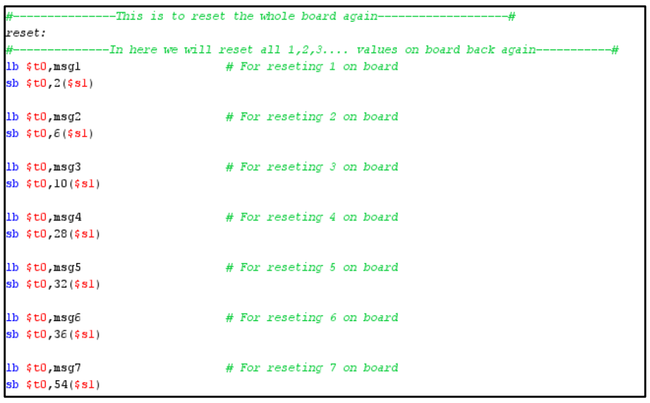
```
#============This is to print the game draw ======================#
draw:
#----------Printing the board----------#
li $v0,4
la $a0,msg_board
syscall
#--------------------------------------#
#-----------Showing message for a tie-------------#
li $v0,4
la $a0,match_tie
syscall
#-----------------------------------------------#

#-------Ending The Program------------------------#
end_Prog:
#=============Ask the user if it want's to play another game or not==================#
li $v0, 4
la $a0, msg_nextGame
syscall

li $v0, 5
syscall
move $t0, $v0

beq $t0, 1, reset           # If the user enter 1 then reset the board again and begin from the start
```



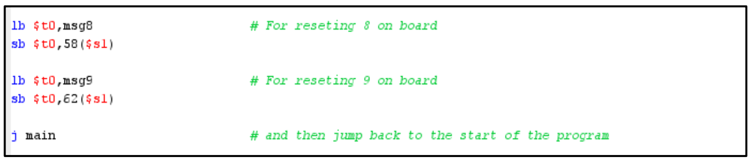
```
#--------------------------------------#
#-----------Showing message for a tie-------------#
li $v0,4
la $a0,match_tie
syscall
#-----------------------------------------------#

#-------Ending The Program------------------------#
end_Prog:
#=============Ask the user if it want's to play another game or not==================#
li $v0, 4
la $a0, msg_nextGame
syscall

li $v0, 5
syscall
move $t0, $v0

beq $t0, 1, reset           # If the user enter 1 then reset the board again and begin from the start

#============Else end the program============#
end:
li $v0, 10
syscall
```

Reset

The reset function is used to reset the whole board. When either of the players wins the game then the users are prompt either to play again or terminate the program. If they press (1) then the whole board is reset using reset function and the program jumps to the main segment. Else the program is terminated.



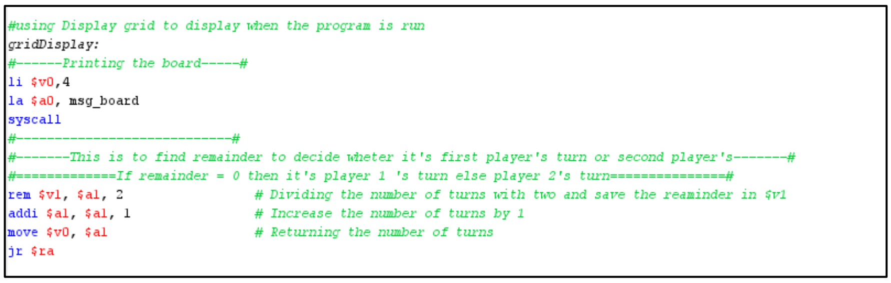
```
#---------------This is to reset the whole board again-------------------#
reset:
#---------------In here we will reset all 1,2,3.... values on board back again-----------#
lb $t0,msg1                 # For reseting 1 on board
sb $t0,2($s1)

lb $t0,msg2                 # For reseting 2 on board
sb $t0,6($s1)

lb $t0,msg3                 # For reseting 3 on board
sb $t0,10($s1)

lb $t0,msg4                 # For reseting 4 on board
sb $t0,28($s1)

lb $t0,msg5                 # For reseting 5 on board
sb $t0,32($s1)

lb $t0,msg6                 # For reseting 6 on board
sb $t0,36($s1)

lb $t0,msg7                 # For reseting 7 on board
sb $t0,54($s1)
```



```
lb $t0,msg8                 # For reseting 8 on board
sb $t0,58($s1)

lb $t0,msg9                 # For reseting 9 on board
sb $t0,62($s1)

j main                      # and then jump back to the start of the program
```

Grid Display

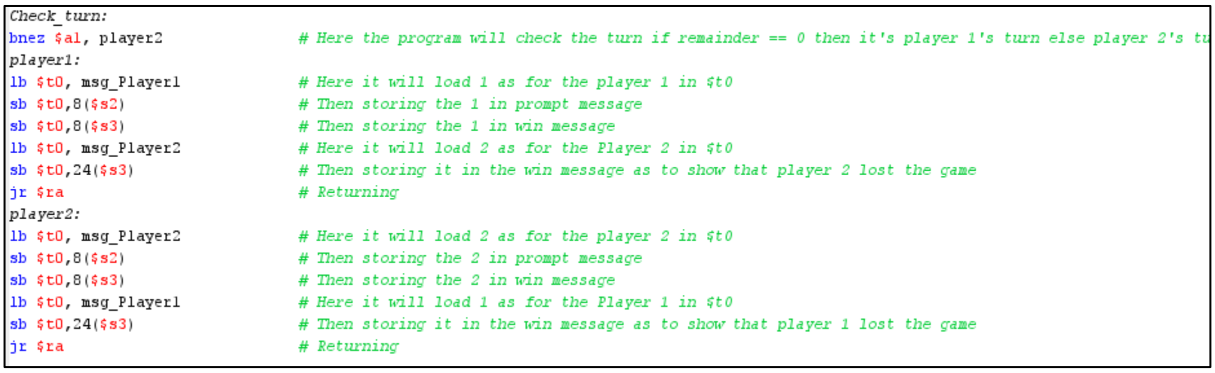
Grid display function is used to show the board in the game which displays the rows and columns for the tic tac toe. In grid display remainder is also calculated to check whether it’s 1st player’s turn or 2nd player’s; also, in the grid display, we increase the number of turns by 1 to calculate the maximum number of turns.



```
#using Display grid to display when the program is run
gridDisplay:
#------Printing the board-----#
li $v0,4
la $a0, msg_board
syscall
#-----------------------------#
#-------This is to find remainder to decide wheter it's first player's turn or second player's-------#
#=============If remainder = 0 then it's player 1 's turn else player 2's turn===============#
rem $v1, $a1, 2             # Dividing the number of turns with two and save the reaminder in $v1
addi $a1, $a1, 1            # Increase the number of turns by 1
move $v0, $a1               # Returning the number of turns
jr $ra
```

Check Turn

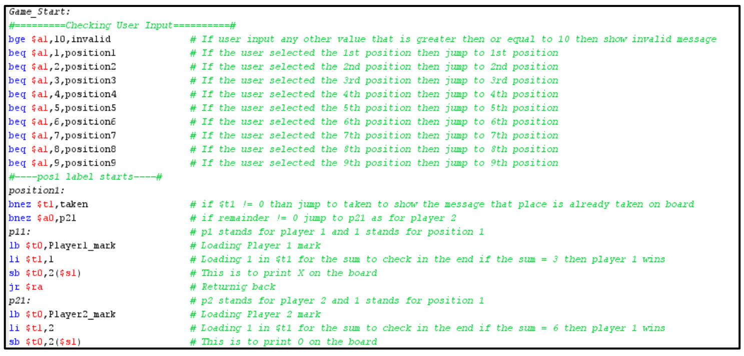
Check turn is used to check which player's turn it is; and then prompt it to the user whether 1st player will place the mark or the second player will place the mark. Just like this, the winner is also prompt whether 1st player won the game or the second player, while the opposite player lost the game.



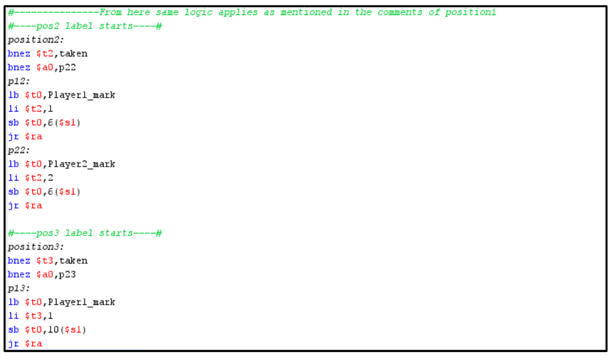
```
Check_turn:
bnez $a1, player2                # Here the program will check the turn if remainder == 0 then it's player 1's turn else player 2's tu
player1:
lb $t0, msg_Player1              # Here it will load 1 as for the player 1 in $t0
sb $t0,8($s2)                    # Then storing the 1 in prompt message
sb $t0,8($s3)                    # Then storing the 1 in win message
lb $t0, msg_Player2              # Here it will load 2 as for the Player 2 in $t0
sb $t0,24($s3)                   # Then storing it in the win message as to show that player 2 lost the game
jr $ra                           # Returning
player2:
lb $t0, msg_Player2              # Here it will load 2 as for the player 2 in $t0
sb $t0,8($s2)                    # Then storing the 2 in prompt message
sb $t0,8($s3)                    # Then storing the 2 in win message
lb $t0, msg_Player1              # Here it will load 1 as for the Player 1 in $t0
sb $t0,24($s3)                   # Then storing it in the win message as to show that player 1 lost the game
jr $ra                           # Returning
```

Game Start

In the game start function, the mark is placed on the board and also checks that whether the given input to place the mark on board is valid or not. If the place input is invalid then the program jumps to the invalid function else jumps on the position label where the mark should be placed.



```
Game_Start:
#=========Checking User Input==========#
bge $a1,10,invalid               # If user input any other value that is greater then or equal to 10 then show invalid message
beq $a1,1,position1              # If the user selected the 1st position then jump to 1st position
beq $a1,2,position2              # If the user selected the 2nd position then jump to 2nd position
beq $a1,3,position3              # If the user selected the 3rd position then jump to 3rd position
beq $a1,4,position4              # If the user selected the 4th position then jump to 4th position
beq $a1,5,position5              # If the user selected the 5th position then jump to 5th position
beq $a1,6,position6              # If the user selected the 6th position then jump to 6th position
beq $a1,7,position7              # If the user selected the 7th position then jump to 7th position
beq $a1,8,position8              # If the user selected the 8th position then jump to 8th position
beq $a1,9,position9              # If the user selected the 9th position then jump to 9th position
#----pos1 label starts----#
position1:
bnez $t1,taken                   # if $t1 != 0 than jump to taken to show the message that place is already taken on board
bnez $a0,p21                     # if remainder != 0 jump to p21 as for player 2
p11:                             # p1 stands for player 1 and 1 stands for position 1
lb $t0,Player1_mark              # Loading Player 1 mark
li $t1,1                         # Loading 1 in $t1 for the sum to check in the end if the sum = 3 then player 1 wins
sb $t0,2($s1)                    # This is to print X on the board
jr $ra                           # Returnig back
p21:                             # p2 stands for player 2 and 1 stands for position 1
lb $t0,Player2_mark              # Loading Player 2 mark
li $t1,2                         # Loading 1 in $t1 for the sum to check in the end if the sum = 6 then player 1 wins
sb $t0,2($s1)                    # This is to print 0 on the board
```



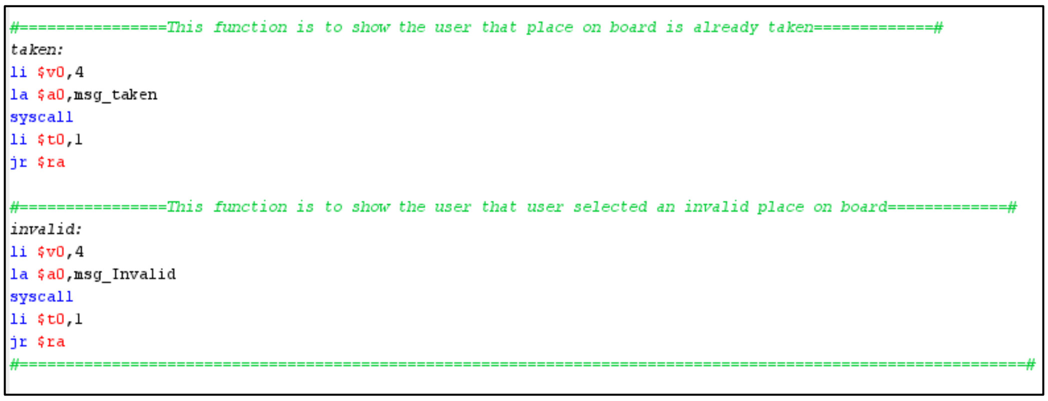
```
#---------------From here same logic applies as mentioned in the comments of position1
#----pos2 label starts----#
position2:
bnez $t2,taken
bnez $a0,p22
p12:
lb $t0,Player1_mark
li $t2,1
sb $t0,6($s1)
jr $ra
p22:
lb $t0,Player2_mark
li $t2,2
sb $t0,6($s1)
jr $ra

#----pos3 label starts----#
position3:
bnez $t3,taken
bnez $a0,p23
p13:
lb $t0,Player1_mark
li $t3,1
sb $t0,10($s1)
jr $ra
```

Note: The photos attached are only for the 1st till 3rd position, because all the other positions have the same logic.

Taken & Invalid

These two functions are used to show if the place on board is taken or the user input is invalid. If the place on board is not equaled to 0 then the place on board is taken.

The invalid function is used to show the user the message that the user input place is invalid on board.



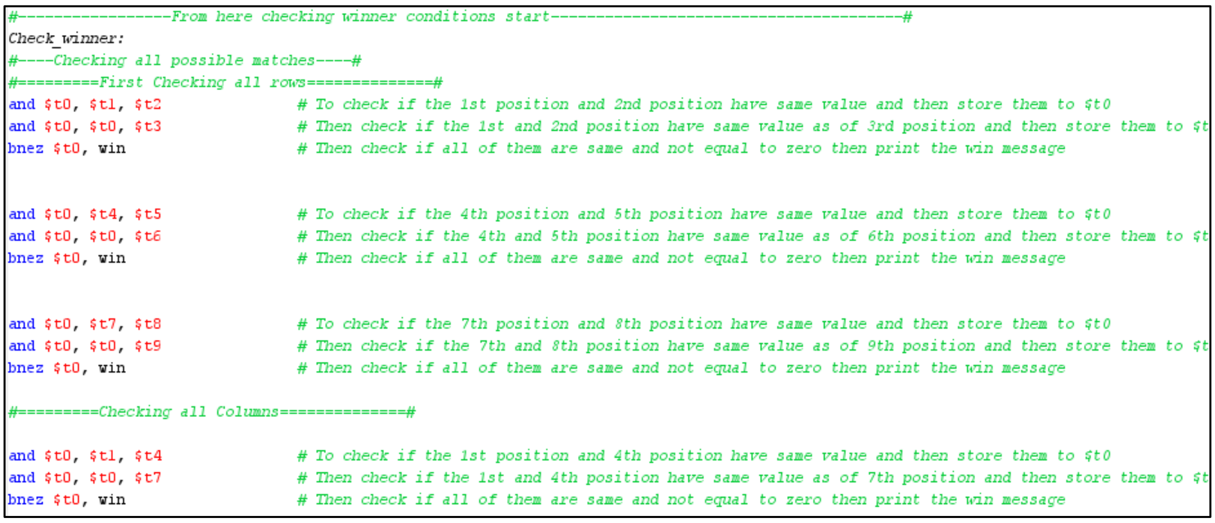
```
#================This function is to show the user that place on board is already taken=============#
taken:
li $v0,4
la $a0,msg_taken
syscall
li $t0,1
jr $ra

#================This function is to show the user that user selected an invalid place on board=============#
invalid:
li $v0,4
la $a0,msg_Invalid
syscall
li $t0,1
jr $ra
#==================================================================================================#
```

Check winner

Check winner is used check all the winning conditions. First all the rows checked using ‘and’ bitwise operator. For example:

and $t0, $t1, $t2	$t0 = $t1 & $t2
and $t0, $t0, $t3	$t0 = $t0 & $t3

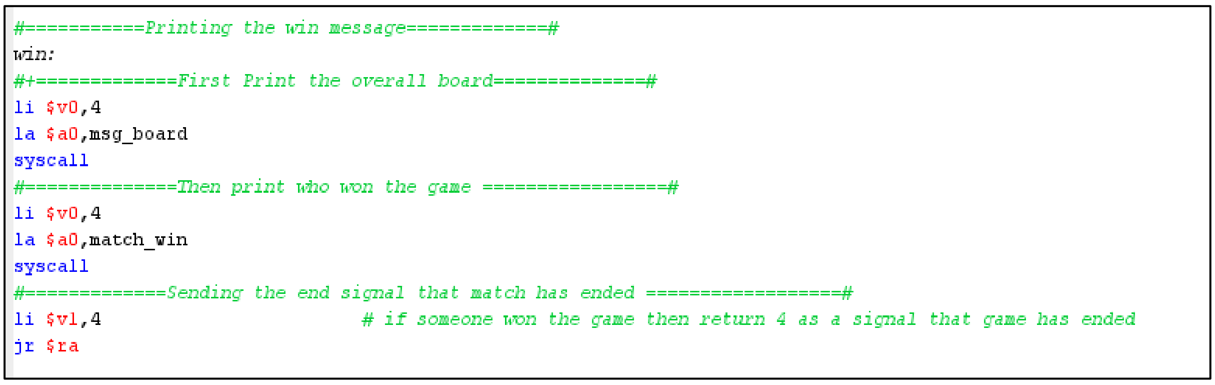
So in the code mentioned above, it checks that whether the 1st place and 2nd place are the same then if they are same the value is placed in $t0 and then it checks whether $t0 and $t3 are same because $t0 contains the check of 1st and 2nd position it checks whether or not it matches with 3rd position also; if so then the player wins the game. Just like this all the rows, columns, and diagonals are checked.



```
#-----------------From here checking winner conditions start--------------------------------------#
Check_winner:
#----Checking all possible matches----#
#=========First Checking all rows==============#
and $t0, $t1, $t2             # To check if the 1st position and 2nd position have same value and then store them to $t0
and $t0, $t0, $t3             # Then check if the 1st and 2nd position have same value as of 3rd position and then store them to $t
bnez $t0, win                 # Then check if all of them are same and not equal to zero then print the win message


and $t0, $t4, $t5             # To check if the 4th position and 5th position have same value and then store them to $t0
and $t0, $t0, $t6             # Then check if the 4th and 5th position have same value as of 6th position and then store them to $t
bnez $t0, win                 # Then check if all of them are same and not equal to zero then print the win message


and $t0, $t7, $t8             # To check if the 7th position and 8th position have same value and then store them to $t0
and $t0, $t0, $t9             # Then check if the 7th and 8th position have same value as of 9th position and then store them to $t
bnez $t0, win                 # Then check if all of them are same and not equal to zero then print the win message

#=========Checking all Columns==============#

and $t0, $t1, $t4             # To check if the 1st position and 4th position have same value and then store them to $t0
and $t0, $t0, $t7             # Then check if the 1st and 4th position have same value as of 7th position and then store them to $t
bnez $t0, win                 # Then check if all of them are same and not equal to zero then print the win message
```

Win

After all the checks are done in the game and if there is a match found then the program jumps to win the label and prompt which player won the game and which player lost the game. After prompting a signal is sent to the main segment telling the program that the game has ended with a winning condition.



```
#===========Printing the win message=============#
win:
#+=============First Print the overall board==============#
li $v0,4
la $a0,msg_board
syscall
#==============Then print who won the game =================#
li $v0,4
la $a0,match_win
syscall
#=============Sending the end signal that match has ended ==================#
li $v1,4                          # if someone won the game then return 4 as a signal that game has ended
jr $ra
```

Demo-Video Link

Part-1

https://youtu.be/GkXWXJkqJfI

Part-2

https://youtu.be/XjmYOuG8S64