

Bahria University,

Karachi Campus



COURSE CODE: CEN-221
COURSE: COMPUTER ARCHITECTURE AND ORGANIZATION
TERM: FALL 2020
CLASS: BSE- 3 (B)

Semester Project's REPORT

Submitted By:

S no.	Name	Enrollment
03.	LAIBA IQBAL	02-131192-054
04.	NISHA AMIN	02-131192-039

Submitted to:

Dr. Samar Yazdani
Engr. Muhammad Rehan Baig

Remarks:_____ Date:_____ Signature:_____

Tic-Tac-Toe Game

Project Report

PREFACE

This report is an introduction to the Tic-tac-toe game in MIPS. Anybody, who doesn't know even the basics of Tic-tac-toe in Assembly language, will be certainly able to understand and gain the great knowledge from this report. The report main focuses on the development of Tic-tac-toe game in the **MIPS**.

Table Of Contents

PREFACE	03
INTRODUCTION.....	Error! Bookmark not defined.05
CODE.....	Error! Bookmark not defined.5
WORK FLOW	14
OUTPUT INTERFACES.....	15
CONCLUSION	17

INTRODUCTION

Tic-tac-toe is not a very challenging game for human beings. Tic-tac-toe (also known as **noughts and crosses** or **Xs and Os**) is for two players, **X** and **O**, who take turns marking the spaces in a 3×3 grid. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row **wins** the game.

CODE

.data

```
board: .ascii " 1 2 3\n| | |\n-----\n2 | | \n-----\n3 | | \n"
askForMove: .ascii "Player insert your play (column|row): "
invalidMove: .ascii "*****Invalid Move*****"
spaceOccupied: .ascii "*****Space already occupied*****\n"
x: .ascii "X"
o: .ascii "O"
won: .ascii "\nPlayer Won!!!!\n"
tie: .ascii "\nTie!!!"
gameMenu: .ascii "\n\nChoose an option:\n[1] New Game\t[99] Quit\nOption: "
clean: .byte ''
```

.text

.globl main

main:

```
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

li \$s0, 0

li \$s5, 0

la \$s1, board

la \$s2, askForMove

la \$s3, won

lb \$a1, clean

sb \$a1, 14(\$s1)

sb \$a1, 18(\$s1)

sb \$a1, 22(\$s1)

sb \$a1, 40(\$s1)

sb \$a1, 44(\$s1)

sb \$a1, 48(\$s1)

sb \$a1, 66(\$s1)

sb \$a1, 70(\$s1)

sb \$a1, 74(\$s1)

PrintBoard:

li \$v0, 4

la \$a0, board

syscall

```

    beq $s5, 9, Tie
    add $s5, $s5, 1
    rem $t0, $s0, 2      #$t0 stores the remainder when $s0 is divided by 2
    add $s0, $s0, 1
    bnez $t0, Player0

```

PlayerX:

```

    lb $a1, x
    sb $a1, 7($s2)
    sb $a1, 8($s3)
    j Play

```

Player0:

```

    lb $a1, o
    sb $a1, 7($s2)
    sb $a1, 8($s3)

```

Play:

```

    li $v0, 4
    la $a0, askForMove
    syscall

```

```

    li $v0, 5      #user input
    syscall
    move $s6, $v0

```

```

    beq $s6, 11, CR11
    beq $s6, 21, CR21
    beq $s6, 31, CR31
    beq $s6, 12, CR12
    beq $s6, 22, CR22
    beq $s6, 32, CR32

```

```
beq $s6, 13, CR13
beq $s6, 23, CR23
beq $s6, 33, CR33
```

```
li $v0, 4      #Prints the message for invalid move
la $a0, invalidMove
syscall
j Play
```

CR11:

```
bnez $t1, Occupied #Branch to the label Occupied if $t1 is not equal to zero
bnez $t0, O11
```

X11:

```
li $t1, 1
sb $a1, 14($s1)
j CheckVictory #jumps to the label CheckVictory to check for the victory
```

O11:

```
li $t1, 2
sb $a1, 14($s1)
j CheckVictory
```

CR21:

```
bnez $t2, Occupied #Branch to the label Occupied if $t2 is not equal to zero
bnez $t0, O21
```

X21:

```
li $t2, 1
sb $a1, 18($s1)
j CheckVictory
```


O21:

li \$t2, 2
sb \$a1, 18(\$s1)
j CheckVictory

CR31:

bnez \$t3, Occupied #Branch to the label Occupied if \$t3 is not equal to zero
bnez \$t0, O31

X31:

li \$t3, 1
sb \$a1, 22(\$s1)
j CheckVictory

O31:

li \$t3, 2
sb \$a1, 22(\$s1)
j CheckVictory

CR12:

bnez \$t4, Occupied #Branch to the label Occupied if \$t4 is not equal to zero
bnez \$t0, O12

X12:

li \$t4, 1
sb \$a1, 40(\$s1)
j CheckVictory

O12:

li \$t4, 2
sb \$a1, 40(\$s1)

j CheckVictory

CR22:

bnez \$t5, Occupied **#Branch to the label Occupied if \$t5 is not equal to zero**
bnez \$t0, O22

X22:

li \$t5, 1
sb \$a1, 44(\$s1)
j CheckVictory

O22:

li \$t5, 2
sb \$a1, 44(\$s1)
j CheckVictory

CR32:

bnez \$t6, Occupied **#Branch to the label Occupied if \$t6 is not equal to zero**
bnez \$t0, O32

X32:

li \$t6, 1
sb \$a1, 48(\$s1)
j CheckVictory

O32:

li \$t6, 2
sb \$a1, 48(\$s1)
j CheckVictory

CR13:

bnez \$t7, Occupied **#Branch to the label Occupied if \$t7 is not equal to zero**
bnez \$t0, O13

X13:

li \$t7, 1
sb \$a1, 66(\$s1)
j CheckVictory

O13:

li \$t7, 2
sb \$a1, 66(\$s1)
j CheckVictory

CR23:

bnez \$t8, Occupied **#Branch to the label Occupied if \$t8 is not equal to zero**
bnez \$t0, O23

X23:

li \$t8, 1
sb \$a1, 70(\$s1)
j CheckVictory

O23:

li \$t8, 2
sb \$a1, 70(\$s1)
j CheckVictory

CR33:

bnez \$t9, Occupied **#Branch to the label Occupied if \$t9 is not equal to zero**
bnez \$t0, O33

X33:

```
li $t9, 1
sb $a1, 74($s1)
j CheckVictory
```

O33:

```
li $t9, 2
sb $a1, 74($s1)
j CheckVictory
```

Occupied:

```
li $v0, 4
la $a0, spaceOccupied      #Prints the spaceOccupied message
syscall
j Play
```

CheckVictory:

```
and $s7, $t1, $t2      # AND instruction, $s7 = $t1 AND $t2
and $s7, $s7, $t3      # AND instruction, $s7 = $s7 AND $t3
bnez $s7, Victory      #Branch to the label victory if $s7 is not equal to zero

and $s7, $t4, $t5      # AND instruction, $s7 = $t4 AND $t5
and $s7, $s7, $t6      # AND instruction, $s7 = $s7 AND $t6
bnez $s7, Victory

and $s7, $t7, $t8      # AND instruction, $s7 = $t7 AND $t8
and $s7, $s7, $t9      # AND instruction, $s7 = $s7 AND $t9
bnez $s7, Victory

and $s7, $t1, $t4      # AND instruction, $s7 = $t1 AND $t4
```

```

and $s7, $s7, $t7      # AND instruction, $s7 = $s7 AND $t7
bnez $s7, Victory

and $s7, $t2, $t5      # AND instruction, $s7 = $t2 AND $t5
and $s7, $s7, $t8      # AND instruction, $s7 = $s7 AND $t8
bnez $s7, Victory

and $s7, $t3, $t6      # AND instruction, $s7 = $t3 AND $t6
and $s7, $s7, $t9      # AND instruction, $s7 = $s7 AND $t9
bnez $s7, Victory

and $s7, $t1, $t5      # AND instruction, $s7 = $t1 AND $t5
and $s7, $s7, $t9      # AND instruction, $s7 = $s7 AND $t9
bnez $s7, Victory

and $s7, $t7, $t5      # AND instruction, $s7 = $t7 AND $t5
and $s7, $s7, $t3      # AND instruction, $s7 = $s7 AND $t3
bnez $s7, Victory
j PrintBoard

```

Victory:

```

li $v0, 4
la $a0, board
syscall

li $v0, 4
la $a0, won      #Prints the won message
syscall
j NewGameMenu

```

Tie:

```
li $v0, 4
la $a0, tie    #Prints the tie message
syscall
```

NewGameMenu:

```
li $v0,4
la $a0, gameMenu    #Prints gameMenu message
syscall

li $v0,5    #user input
syscall

bne $v0, 99, main    #Move to main, if $v0 is not equal to 99

li $v0, 10    #Exit program
syscall
```

WORKFLOW:

The board is printed in console, there is choice for player to enter the column/row number. After that second player will insert its choice if the choice entered will be same which was entered by first player then it will pop up message that space is already occupied.

If the player enter the wrong move then means than wrong column and row number then show error message. After every insert it will check the victory with

every possibility. If there is no victory of any player till all the boxes are filled then it will show the message that game tie. It will ask the user in the end of game that they want to play a new game or exit.

OUTPUT INTERFACES

1) Wrong move

```
  1   2   3
1   |   |
  ---+---+---
2   |   |
  ---+---+---
3   |   |
Player X insert your play (column|row):12
  1   2   3
1   |   |
  ---+---+---
2 X |   |
  ---+---+---
3   |   |
Player O insert your play (column|row):34
**Invalid Move**Player O insert your play (column|row):45
**Invalid Move**Player O insert your play (column|row):43
**Invalid Move**Player O insert your play (column|row):|
```

2) Space occupied

```
  1   2   3
1  |   |
---+---+---
2  |   |
---+---+---
3  |   |
Player X insert your play (column|row):12
  1   2   3
1  |   |
---+---+---
2 X |   |
---+---+---
3  |   |
Player O insert your play (column|row):32
  1   2   3
1  |   |
---+---+---
2 X |   | O
---+---+---
3  |   |
Player X insert your play (column|row):21
  1   2   3
1  | X |
---+---+---
2 X |   | O
---+---+---
3  |   |
Player O insert your play (column|row):32
**Space already occupied**
Player O insert your play (column|row):|
```

3) Player won

```
~ ~ ~
Player X insert your play (column|row):12
  1   2   3
1 X |   |
---+---+---
2 X |   |
---+---+---
3  | O |
Player O insert your play (column|row):23
**Space already occupied**
Player O insert your play (column|row):32
  1   2   3
1 X |   |
---+---+---
2 X |   | O
---+---+---
3  | O |
Player X insert your play (column|row):13
  1   2   3
1 X |   |
---+---+---
2 X |   | O
---+---+---
3 X | O |

Player X Won!

Choose an option:
[1] New Game      [99] Quit
Option: 99

-- program is finished running --
```


CONCLUSION

The **Tic-tac-toe game** is most familiar among all the age groups. Intelligence can be a property of any purpose-driven programming language. A code of playing Tic tac-toe game has been presented and tested that works in efficient way. Overall the MIPS Program works without any errors.

In the conclusion of this project, we would like to say that MIPS Assembly language is an efficient and easy programming language and while creating a project like this, it has not just been a good experience but it also helped in the development of our **creativity** and **logical thinking**.