Bahria University,

Karachi Campus

COURSE: CEN-221 COMPUTER ARCHITECTURE AND ORGANIZATION

TERM: FALL 2020, CLASS: BSE- 3(B)

******

**A Project Report on**

**Tic Tac Toe Game (In MIPS Assembly Language)**

**Project Documentation**

**Project Title: TIC TAC TOE**

**Course Name: COMPUTER ARCHITECTURE & ORGANIZATION**

**Course Code: (CEN-221)**

**Lab Instructor: Engr. MUHAMMAD REHAN BAIG**

**Group Leader: MUHAMMAD USMAN KHAN**

**Class: BSE-3(B)**

**Group Members: 4**

**GROUP MEMBERS**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Student Name** | **Enrollment** |
| **1** | Muhammad Usman Khan | 02-131192-061 |
| **2** | Muhammad Hassam | 02-131192-019 |
| **3** | Muhammad Aakif Aijaz | 02-131192-007 |
| **4** | Mustafa Hussain | 02-131192-004 |

**TABLE OF CONTENT:**

* Abstract
* Introduction
* Description
* Solution
* Tools

**Project: TIC TAC TOE**

**ABSTRACT:**

We choose tic-tac-toe, since virtually everyone knows the game and the rules are simple enough that we don't need an elaborate analysis on game configuration. Despite being simple game it can be used in more complicated games like checker, connect 4 and even chess.

We have made two mode game First mode (is automatic) where computer play with palyer1 and in second mode (is manual) palyer1 compete with player2. It is a mixture of both manual and automatic game play. In order to solve Tic Tac Toe, we need to go deeper than just to think about it as a game where two players place X’s and O’s on the board. Formally speaking, Tic Tac Toe is a zero-sum and perfect information game. It means that each participant’s gain is equal to the other participants’ losses and we know everything about the current game state. In a two-player (A vs B) game, if player A scores x points (utility units), player B loses x points. Total gains/losses always sum up to 0.

**INTRODUCTION:**

Tic-Tac-Toe is a very simple two player game. So only two players can play at a time. This game is also known as Naught and Crosses or X and O game. One player plays with X and the other player plays with O. In this game we have a board consisting of a 3X3 grid. The number of grids may be increased.

**BACKGROUND:**

Tic-tac-toe is a game that is traditionally played by being drawn on paper, and it can be played on a computer or on a variety of media. Other games, such as Connect 4, are based on this classic.

An early variation of the game was played in the Roman Empire, around the 1st century B.C. It was called "Terni lapilli," which means "three pebbles at a time." The game's grid markings have been found chalked all over Roman ruins. Evidence of the game was also found in ancient Egyptian ruins. The first print reference to "naught and crosses," the British name for the game, appeared in 1864. The first print reference to a game called "tick-tack-toe" occurred in 1884 but referred to a children's game played on a slate.

A relatively simple game usually played on a grid of 3-by-3 squares, tic-tac-toe is mainly enjoyed by children. Tic-tac-toe can be made significantly more complex by increasing the size of the board to 4-by-4, 5-by-5, or even up to a 20-by-20 grid.



**PROJECT DESCRIPTION:**

This tic-tac-toe game is built on MIPS assembly language. MIPS assembly language simply refers to the assembly language of the MIPS processor. The term MIPS is an acronym for Microprocessor without Interlocked Pipeline Stages. It is a reduced-instruction set architecture developed by an organization called MIPS Technologies. The MIPS assembly language is a very useful language to learn because many embedded systems run on the MIPS processor. Knowing how to code in this language brings a deeper understanding of how these systems operate on a lower level.

A tic-tac-toe game has two modes, first one between humans i.e. player1 vs player2, where the player1 compete with player2, the second mode is between computer vs human, where the player always begins the match.

The computer algorithm is really simple. First it looks for any two of its marks in a row, so it can place the third one to win the game. If there are none, it looks for any two of opponent’s marks in a row, so it can block them. Finally, if there are none, it marks a randomly chosen empty square on the board.

User specifies the amount of matches to be played. After all of the matches are finished, the final scores are displayed to the user.

**GAME FUNCTION:**

Tic-tac-toe is a two player game (automatic & manual). First we select the mode of the game. First mode is automatic in which we play with computer and the second mode is manual in which two platers compete with each other. Here we say for two player game. The two players take turns by putting marks on a 3x3 board. The player who first gets 3 of his/her marks in a row (vertically, horizontally, or diagonally) wins the game, and the other loses the game, and if no player wins the game is draw(tie).

**IMPLEMENTATION:**

In our program, the moves taken by the computer and the human are chosen randomly.

Our project consists of following four major parts:

1. Computer function
2. Player function
3. Loop function
4. Check win function
5. Board function
6. Main function

Basically there are two modes for playing the game. First is the Computer function in which we play with computer. Second one is Player Function I which palyer1 compete with player2. Loop function loops the game operation (how many games you want to play). Board function consists of characters like X, O and is used for placing the X, O on the board. Check win function is used for detecting the winner, looser or the draw result. Main function is used for printing/displaying out playing board. Lastly, in main function, the logical and arithmetical steps occur.

**RULES OF THE GAME:**

The game is to be played between two people (in this program between HUMAN and COMPUTER). One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells. The game starts with one of the players and the game ends when one of the players has one whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’). If no one wins, then the game is said to be draw. The program is not played optimally by both sides because the moves are chosen randomly. The program can be easily modified so that both players play optimally.

**SOLUTION:**

**CODE:**

# CAO GROUP PORJECT

# MEMBER LIST:

# MUHAMMAD USMAN KHAN 02-131192-061

# MUHAMMAD HASSAM 02-131192-019

# MUSTAFA HUSSAIN 02-131192-004

# MUHAMMAD AAKIF AIJAZ 02-131192-007

######COMPUTER ARCHITECTURE AND ORGANIZATION LAB PROJECT###########

###################################################################

.data

des: .asciiz "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n"

TICTACTOE:.asciiz "\t\tTIC TAC TOE\n"

name: .asciiz "ENTER YOUR NAME: "

p1name:.asciiz "PLAYER 1, ENTER YOUR NAME: "

p2name:.asciiz "PLAYER 2, ENTER YOUR NAME: "

INPUT: .space 20

INPUT2: .space 20

input: .space 20

intro: .asciiz "ENTER THE NUMBER OF GAMES YOU WANT TO PLAY:\n "

invintro: .asciiz "YOU HAVE ENTERED WRONG NUMBER OF MATCHES:\n "

again: .asciiz"DO YOU WANT TO PLAY MORE MATCHES:\n\tPRESS 1 FOR YES\n\tPRESS 0 TO QUIT\n"

choice: .asciiz "DO YOU WANT TO PLAY WITH PLAYER OR COMPUTER? \n\1)PRESS 0 FOR PLAYER\n2)PRESS 1 FOR COMPUTER\n"

GAMENUMBER: .asciiz "\tGAME # "

nl: .asciiz "\n"

col: .asciiz ":"

line: .asciiz ""

design: .asciiz "\t|\_|"

player: .asciiz "\t|X|"

computer: .asciiz "\t|+|"

p1turn: .asciiz "PLAYER 1 ENTER VALUE FROM ( 1-9):\n "

p2turn: .asciiz "PLAYER 2 ENTER VALUE FROM ( 1-9):\n "

turn: .asciiz "ENTER VALUE FROM ( 1-9):\n "

invturn: .asciiz "\tINVALID VALUE IS ENTERED!\n"

fill: .asciiz "\tTHE CELL IS ALREADY FILLED!\n"

tie: .asciiz "\t\tTIE!\n"

X\_WINS: .asciiz "\tWINNER IS "

O\_WINS: .asciiz "\tWINNER IS COMPUTER\n"

RESULT: .asciiz "\tRESULT: "

dash: .asciiz "-"

board: .byte 1,1,0,2,1,2,0,0,1

row\_preferences: .space 8

row\_preference\_state\_lookup\_table: .byte 1, 3, 2, 5, 6, 5, 5, 4, 5, 7, 5, 5

cell\_to\_rows: .byte 0, 3, 6, 8, 0, 4, 8, 8, 0, 5, 7, 8,

1, 3, 8, 8, 1, 4, 6, 7, 1, 5, 8, 8,

2, 3, 7, 8, 2, 4, 8, 8, 2, 5, 6, 8

row\_to\_cells: .byte 0, 1, 2, 3, 4, 5, 6, 7, 8, 0, 3, 6, 1, 4, 7, 2, 5, 8, 0, 4, 8, 2, 4, 6

.text

main:

la $a0, des

li $v0, 4

syscall

la $a0, TICTACTOE

li $v0, 4

syscall

la $a0, des

li $v0, 4

syscall

CHOICE:

li $v0, 4

la $a0, choice

syscall

li $v0,5

syscall

beq $v0,0,PLAYER

beq $v0,1,COMPUTER

COMPUTER:

la $a0, name

li $v0, 4

syscall

li $v0,8

la $a0,input

li $a1,20

syscall

la $a0, des

li $v0, 4

syscall

b computergame

CONTINUE:

la $a0, des

li $v0, 4

syscall

li $v0, 4

la $a0, again

syscall

li $v0,5

syscall

beq $v0,1,CHOICE

beq $v0,0,exit

computergame:

computergame\_enter\_number:

la $a0, intro

li $v0, 4

syscall

li $v0, 5

syscall

blt $v0, 1, computerinput\_invalid\_amount

nop

move $s0, $v0

j computergame\_matches

nop

computerinput\_invalid\_amount:

la $a0, invintro

li $v0, 4

syscall

j computergame\_enter\_number

nop

computergame\_matches:

li $s1, 0

li $s2, 0

li $s3, 0

computergame\_match:

beq $s1, $s0, computergame\_print\_scores

la $a0, GAMENUMBER

li $v0, 4

syscall

add $a0, $s1, 1

li $v0, 1

syscall

la $a0, col

li $v0, 4

syscall

la $a0, nl

li $v0, 4

syscall

addi $sp, $sp, -20

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

sw $s3, 16($sp)

jal computermatch

nop

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

lw $s3, 16($sp)

addi $sp, $sp, 20

addi $s1, $s1, 1

add $s2, $s2, $v0

add $s3, $s3, $v1

or $t1, $v0, $v1

beqz $t1, computergame\_match\_result\_tie

nop

beqz $v0, computergame\_match\_result\_o\_won

nop

computergame\_match\_result\_x\_won:

la $a0, X\_WINS

li $v0, 4

syscall

li $v0, 4

la $a0, input

syscall

j computergame\_match

nop

computergame\_match\_result\_o\_won:

la $a0, O\_WINS

li $v0, 4

syscall

j computergame\_match

nop

computergame\_match\_result\_tie:

la $a0, tie

li $v0, 4

syscall

j computergame\_match

nop

computergame\_print\_scores:

la $a0, des

li $v0, 4

syscall

la $a0, RESULT

li $v0, 4

syscall

move $a0, $s2

li $v0, 1

syscall

la $a0, dash

li $v0, 4

syscall

move $a0, $s3

li $v0, 1

syscall

la $a0, nl

li $v0, 4

syscall

la $a0, des

li $v0, 4

syscall

la $a0, nl

li $v0, 4

syscall

b CONTINUE

computergame\_return:

jr $ra

nop

computermatch:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 0

li $t1, 0

computermatch\_clear\_board:

beq $t0, 9, computermatch\_clear\_preferences

nop

sb $zero, board($t0)

addi $t0, $t0, 1

j computermatch\_clear\_board

nop

computermatch\_clear\_preferences:

beq $t1, 8, computermatch\_preturn

nop

sb $zero, row\_preferences($t1)

addi $t1, $t1, 1

j computermatch\_clear\_preferences

nop

computermatch\_preturn:

li $s0, 0

computermatch\_turn:

addi $sp, $sp, -4

sw $s0, 0($sp)

jal board\_print

nop

jal computerplayer\_move

nop

li $t0, 1

sb $t0, board($v0)

move $a0, $v0

li $a1, 0

jal computerupdate\_row\_preference\_state

nop

beq $v0, 1, computermatch\_turn\_x\_win

nop

lw $s0, 0($sp)

addi $sp, $sp, 4

beq $s0, 8, computermatch\_turn\_tie

nop

addi $s0, $s0, 1

addi $sp, $sp, -4

sw $s0, 0($sp)

jal ai\_move

nop

li $t0, 2

sb $t0, board($v0)

move $a0, $v0

li $a1, 1

jal computerupdate\_row\_preference\_state

nop

beq $v0, 1, computermatch\_turn\_o\_win

nop

lw $s0, 0($sp)

addi $sp, $sp, 4

addi $s0, $s0, 1

j computermatch\_turn

nop

computermatch\_turn\_tie:

jal board\_print

nop

li $v0, 0

li $v1, 0

j computermatch\_return

nop

computermatch\_turn\_x\_win:

addi $sp, $sp, 4

jal board\_print

nop

li $v0, 1

li $v1, 0

j computermatch\_return

nop

computermatch\_turn\_o\_win:

addi $sp, $sp, 4

jal board\_print

nop

li $v0, 0

li $v1, 1

computermatch\_return:

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

nop

computerupdate\_row\_preference\_state:

mul $t0, $a0, 4

li $t1, 0

li $v0, 0

computerupdate\_row\_preference\_state\_get\_row:

beq $t1, 4, computerupdate\_row\_preference\_state\_return

nop

add $t2, $t0, $t1

lb $t3, cell\_to\_rows($t2)

bne $t3, 8, computerupdate\_row\_preference\_state\_update

nop

addi $t1, $t1, 1

j computerupdate\_row\_preference\_state\_get\_row

nop

computerupdate\_row\_preference\_state\_update:

addi $t1, $t1, 1

lb $t4, row\_preferences($t3)

mul $t4, $t4, 2

add $t4, $t4, $a1

lb $t5, row\_preference\_state\_lookup\_table($t4)

sb $t5, row\_preferences($t3)

bge $t5, 6, computerupdate\_row\_preference\_state\_update\_win\_occured

nop

j computerupdate\_row\_preference\_state\_get\_row

nop

computerupdate\_row\_preference\_state\_update\_win\_occured:

li $v0, 1

j computerupdate\_row\_preference\_state\_get\_row

nop

computerupdate\_row\_preference\_state\_return:

jr $ra

nop

computerplayer\_move:

computerplayer\_move\_enter\_number:

la $a0, turn

li $v0, 4

syscall

li $v0, 5

syscall

blt $v0, 1, computerplayer\_move\_invalid\_index

nop

bgt $v0, 9, computerplayer\_move\_invalid\_index

nop

addi $v0, $v0, -1

lb $t0, board($v0)

bnez $t0, computerplayer\_move\_cell\_occupied

nop

j computerplayer\_move\_return

nop

computerplayer\_move\_cell\_occupied:

la $a0,fill

li $v0, 4

syscall

j computerplayer\_move\_enter\_number

nop

computerplayer\_move\_invalid\_index:

la $a0, invturn

li $v0, 4

syscall

j computerplayer\_move\_enter\_number

nop

computerplayer\_move\_return:

jr $ra

nop

ai\_move:

ai\_move\_attack:

li $t0, 0

ai\_move\_attack\_loop:

beq $t0, 8, ai\_move\_defend

nop

lb $t1, row\_preferences($t0)

beq $t1, 4, ai\_move\_lethal\_found

nop

addiu $t0, $t0, 1

j ai\_move\_attack\_loop

nop

ai\_move\_defend:

li $t0, 0

ai\_move\_defend\_loop:

beq $t0, 8, random\_cell

nop

lb $t1, row\_preferences($t0)

beq $t1, 2, ai\_move\_lethal\_found

nop

addiu $t0, $t0, 1

j ai\_move\_defend\_loop

nop

ai\_move\_lethal\_found:

mul $t0, $t0, 3

lb $v0, row\_to\_cells+0($t0)

lb $t2, board($v0)

beqz $t2, ai\_move\_return

nop

lb $v0, row\_to\_cells+1($t0)

lb $t2, board($v0)

beqz $t2, ai\_move\_return

nop

lb $v0, row\_to\_cells+2($t0)

lb $t2, board($v0)

beqz $t2, ai\_move\_return

nop

random\_cell:

li $a0, 0

li $a1, 8

li $v0, 42

syscall

lb $t0, board($a0)

bnez $t0, random\_cell

nop

move $v0, $a0

ai\_move\_return:

jr $ra

nop

board\_print:

li $t0, 0

board\_print\_collumn:

li $t1, 0

beq $t0, 9, board\_print\_end

board\_print\_row:

lbu $t2, board($t0)

beq $t2, 1, board\_print\_x

nop

beq $t2, 2, board\_print\_o

nop

board\_print\_space:

li $v0, 4

la $a0, design

syscall

j board\_print\_skip

nop

board\_print\_x:

li $v0, 4

la $a0, player

syscall

j board\_print\_skip

nop

board\_print\_o:

li $v0, 4

la $a0, computer

syscall

board\_print\_skip:

addiu $t0, $t0, 1

addiu $t1, $t1, 1

blt $t1, 3, board\_print\_newline\_skip

nop

li $v0, 4

la $a0, nl

syscall

j board\_print\_collumn

nop

board\_print\_newline\_skip:

j board\_print\_row

nop

board\_print\_end:

jr $ra

nop

b exit

PLAYER:

la $a0, p1name

li $v0, 4

syscall

li $v0,8

la $a0,INPUT

li $a1,20

syscall

la $a0, p2name

li $v0, 4

syscall

li $v0,8

la $a0,INPUT2

li $a1,20

syscall

la $a0, des

li $v0, 4

syscall

b game

game:

game\_enter\_number:

la $a0, intro

li $v0, 4

syscall

li $v0, 5

syscall

blt $v0, 1, input\_invalid\_amount

nop

move $s0, $v0

j game\_matches

nop

input\_invalid\_amount:

la $a0, invintro

li $v0, 4

syscall

j game\_enter\_number

nop

game\_matches:

li $s1, 0

li $s2, 0

li $s3, 0

game\_match:

beq $s1, $s0, game\_print\_scores

la $a0, GAMENUMBER

li $v0, 4

syscall

add $a0, $s1, 1

li $v0, 1

syscall

la $a0, col

li $v0, 4

syscall

la $a0, nl

li $v0, 4

syscall

addi $sp, $sp, -20

sw $ra, 0($sp)

sw $s0, 4($sp)

sw $s1, 8($sp)

sw $s2, 12($sp)

sw $s3, 16($sp)

jal match

nop

lw $ra, 0($sp)

lw $s0, 4($sp)

lw $s1, 8($sp)

lw $s2, 12($sp)

lw $s3, 16($sp)

addi $sp, $sp, 20

addi $s1, $s1, 1

add $s2, $s2, $v0

add $s3, $s3, $v1

or $t1, $v0, $v1

beqz $t1, game\_match\_result\_tie

nop

beqz $v0, game\_match\_result\_o\_won

nop

game\_match\_result\_x\_won:

la $a0, X\_WINS

li $v0, 4

syscall

li $v0, 4

la $a0, INPUT

syscall

j game\_match

nop

game\_match\_result\_o\_won:

la $a0, X\_WINS

li $v0, 4

syscall

li $v0, 4

la $a0, INPUT2

syscall

j game\_match

nop

game\_match\_result\_tie:

la $a0, tie

li $v0, 4

syscall

j game\_match

nop

game\_print\_scores:

la $a0, des

li $v0, 4

syscall

la $a0, RESULT

li $v0, 4

syscall

move $a0, $s2

li $v0, 1

syscall

la $a0, dash

li $v0, 4

syscall

move $a0, $s3

li $v0, 1

syscall

la $a0, nl

li $v0, 4

syscall

la $a0, des

li $v0, 4

syscall

la $a0, nl

li $v0, 4

syscall

b CONTINUE

game\_return:

jr $ra

nop

match:

addi $sp, $sp, -4

sw $ra, 0($sp)

li $t0, 0

li $t1, 0

match\_clear\_board:

beq $t0, 9, match\_clear\_preferences

nop

sb $zero, board($t0)

addi $t0, $t0, 1

j match\_clear\_board

nop

match\_clear\_preferences:

beq $t1, 8, match\_preturn

nop

sb $zero, row\_preferences($t1)

addi $t1, $t1, 1

j match\_clear\_preferences

nop

match\_preturn:

li $s0, 0

match\_turn:

addi $sp, $sp, -4

sw $s0, 0($sp)

jal board\_print

nop

jal player\_move

nop

li $t0, 1

sb $t0, board($v0)

move $a0, $v0

li $a1, 0

jal update\_row\_preference\_state

nop

beq $v0, 1, match\_turn\_x\_win

nop

lw $s0, 0($sp)

addi $sp, $sp, 4

beq $s0, 8, match\_turn\_tie

nop

addi $s0, $s0, 1

addi $sp, $sp, -4

sw $s0, 0($sp)

jal secondplayer\_move

nop

li $t0, 2

sb $t0, board($v0)

move $a0, $v0

li $a1, 1

jal update\_row\_preference\_state

nop

beq $v0, 1, match\_turn\_o\_win

nop

lw $s0, 0($sp)

addi $sp, $sp, 4

addi $s0, $s0, 1

j match\_turn

nop

match\_turn\_tie:

jal board\_print

nop

li $v0, 0

li $v1, 0

j match\_return

nop

match\_turn\_x\_win:

addi $sp, $sp, 4

jal board\_print

nop

li $v0, 1

li $v1, 0

j match\_return

nop

match\_turn\_o\_win:

addi $sp, $sp, 4

jal board\_print

nop

li $v0, 0

li $v1, 1

match\_return:

lw $ra, 0($sp)

addi $sp, $sp, 4

jr $ra

nop

update\_row\_preference\_state:

mul $t0, $a0, 4

li $t1, 0

li $v0, 0

update\_row\_preference\_state\_get\_row:

beq $t1, 4, update\_row\_preference\_state\_return

nop

add $t2, $t0, $t1

lb $t3, cell\_to\_rows($t2)

bne $t3, 8, update\_row\_preference\_state\_update

nop

addi $t1, $t1, 1

j update\_row\_preference\_state\_get\_row

nop

update\_row\_preference\_state\_update:

addi $t1, $t1, 1

lb $t4, row\_preferences($t3)

mul $t4, $t4, 2

add $t4, $t4, $a1

lb $t5, row\_preference\_state\_lookup\_table($t4)

sb $t5, row\_preferences($t3)

bge $t5, 6, update\_row\_preference\_state\_update\_win\_occured

nop

j update\_row\_preference\_state\_get\_row

nop

update\_row\_preference\_state\_update\_win\_occured:

li $v0, 1

j update\_row\_preference\_state\_get\_row

nop

update\_row\_preference\_state\_return:

jr $ra

nop

player\_move:

player\_move\_enter\_number:

la $a0, p1turn

li $v0, 4

syscall

li $v0, 5

syscall

blt $v0, 1, player\_move\_invalid\_index

nop

bgt $v0, 9, player\_move\_invalid\_index

nop

addi $v0, $v0, -1

lb $t0, board($v0)

bnez $t0, player\_move\_cell\_occupied

nop

j player\_move\_return

nop

player\_move\_cell\_occupied:

la $a0,fill

li $v0, 4

syscall

j player\_move\_enter\_number

nop

player\_move\_invalid\_index:

la $a0, invturn

li $v0, 4

syscall

j player\_move\_enter\_number

nop

player\_move\_return:

jr $ra

nop

secondplayer\_move:

secondplayer\_move\_enter\_number:

la $a0, p2turn

li $v0, 4

syscall

li $v0, 5

syscall

blt $v0, 1, secondplayer\_move\_invalid\_index

nop

bgt $v0, 9, secondplayer\_move\_invalid\_index

nop

addi $v0, $v0, -1

lb $t0, board($v0)

bnez $t0, secondplayer\_move\_cell\_occupied

nop

j secondplayer\_move\_return

nop

secondplayer\_move\_cell\_occupied:

la $a0,fill

li $v0, 4

syscall

j secondplayer\_move\_enter\_number

nop

secondplayer\_move\_invalid\_index:

la $a0, invturn

li $v0, 4

syscall

j secondplayer\_move\_enter\_number

nop

secondplayer\_move\_return:

jr $ra

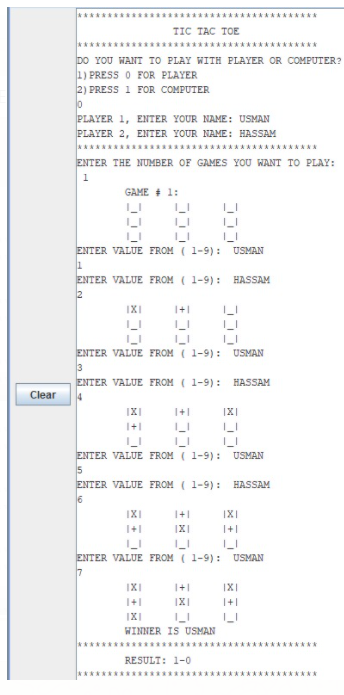
nop

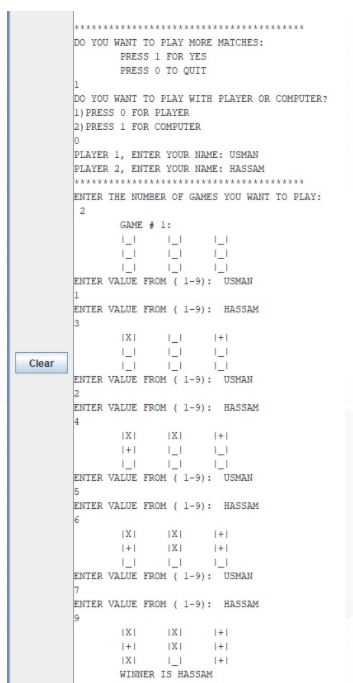
exit:

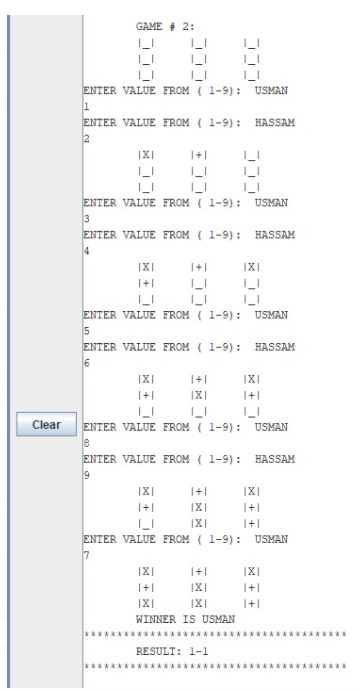
li $v0,10

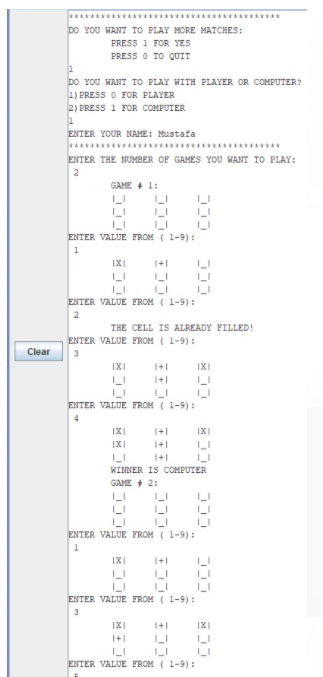
syscall

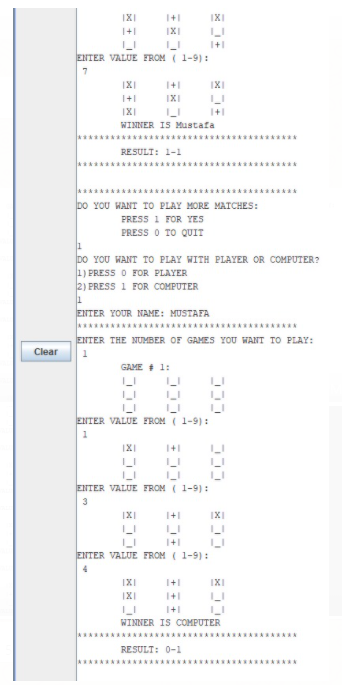
**OUTPUT:**

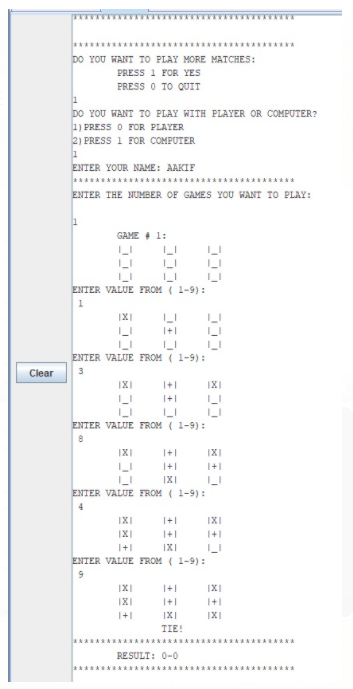


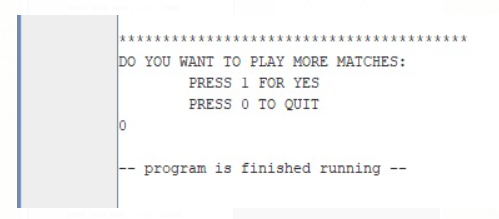












**TOOLS:**

1. Mars(MIPS Simulator).
2. Microsoft Word (Documentation)
3. GitHub (Version Control)

**CONCLUSION:**

In this project we made a game to be played by two persons using looping and branching function to make our game responsive and user friendly on Mars software. Tic-tac-toe can be also played on a 5-by-5 grid with each player trying to get five in a row. The game can also be played on larger grids, such as 10-by-10 or even 20-by-20. For any grid of 6-by-6 or greater, it might be best to make your goal to get five in a row. This turns the basic game of tic-tac-toe into a much more complex game with similarities to the board game Pente, meaning "five" in Greek. Similarly, the goal of Pente is for a player to score five marks in a row.

**Thank You!**