Tic.Tac.Toe

MICROPROCESSOR & ASSEMBLY LANGUAGE

Under supervision:

DR. Fatma Sakr

Eng. Noha Teaima

Team Members:
Mahmoud Yasser Mahmoud (G6)
Hassan Mahmoud Abdelfattah (G6)
Micheal Amgad Bakom (G6)
Marwan ashraf hassan Megahed G6

Tic.Tac.Toe Game

Project Overview

The uploaded file implements a Tic-Tac-Toe game in Assembly Language, designed for two players. The program runs in a text-based environment and uses direct memory access and interrupts to interact with the user and display the game board. The project demonstrates the use of low-level programming techniques for creating an interactive application.

Explanation of the Project Idea

The project aims to replicate the classic two-player Tic-Tac-Toe game, providing an engaging way to showcase assembly programming's capabilities. Key features include:

1. **Player Interaction**:
    - Players take turns to enter their moves.
    - The game alternates between Player 1 ('X') and Player 2 ('O').
2. **Game Board Display**:
    - A graphical representation of the board updates dynamically.
    - Cell numbers indicate empty spaces, while occupied cells display 'X' or 'O'.
3. **Game Mechanics**:
    - The program checks for winning conditions after each move.
    - If all cells are filled without a winner, the game declares a draw.
    - It supports replay functionality, allowing users to restart or exit after a game session.
4. **Input Validation**:
    - The program validates user input to ensure moves are placed in unoccupied cells.
    - Invalid inputs prompt appropriate error messages.

CODE

1)

```
.MODEL SMALL
.STACK 500H

.DATA
; CELL MARK FOR PLAYERS
PC1 DB ' (X)$'
PC2 DB ' (O)$'

; BOARD LINES -------
L1 DB '     !     !     $'
L2 DB '-----------$'
N1 DB ' ! $'
; -------------------

; CELL NUMBERS ------
C1 DB '1$'
C2 DB '2$'
C3 DB '3$'
C4 DB '4$'
C5 DB '5$'
C6 DB '6$'
C7 DB '7$'
C8 DB '8$'
C9 DB '9$'
; -------------------

; PLAYER NO. , MOVES AND CHECK FLAGS FOR IF THE GAME IS WON OR DRAWN
PLAYER DB 50, '$'
MOVES DB 0
DONE DB 0
DR DB 0

; INPUT SECTION PROMTS --------------------------
INP DB 32, ':: Enter cell no. : $'
TKN DB 'This cell is taken! Press any key...$'

; CURRENT MARK (X/O) ----------------------------
CUR DB 88

; FINAL MESSAGES --------------------------------
W1 DB 'Player $'
W2 DB ' won the game!$'
DRW DB 'The game is draw!$'

; TRY AGAIN PROMPT MESSAGES ---------------------------
TRA DB 'Want to play again? (y/n): $'
WI DB  32, 32, 32, 'Wrong input! Press any key...   $'

; THIS LINE IS USED TO OVERWIRTE A LINE TO CLEAN THE AREA
EMP DB '                                                $'

;-----------------------------------------------------
```

- **Cell Marks** (PC1, PC2): Used to display **X** or **O** on the board.
- **Board Lines** (L1, L2, N1): Used to construct the visual layout of the game board.
- **Cell Numbers** (C1 to C9): Initially display the numbered positions, which will be updated as the game progresses.
- **Game State Flags** (MOVES, DONE, DR): Keep track of the current game state, including the number of moves and whether the game is over or drawn.
- **Prompts and Messages** (INP, TKN, W1, W2, etc.): Provide the user with necessary instructions and feedback.
- **Screen Cleaning** (EMP): Used to overwrite and clear areas of the screen during updates.

2) INITIALIZE

The given code snippet is written in assembly language, and its purpose is to initialize various variables for a game, likely a simple player-based game like Tic-Tac-Toe or a similar grid-based game.

```
; ---------- INITIALIZE ---------------------

      INIT:

            MOV PLAYER, 50      ; INITIALIZING ALL VARIABLES
            MOV MOVES, 0
            MOV DONE, 0
            MOV DR, 0

            MOV C1, 49
            MOV C2, 50
            MOV C3, 51
            MOV C4, 52
            MOV C5, 53
            MOV C6, 54
            MOV C7, 55
            MOV C8, 56
            MOV C9, 57

            JMP PLRCHANGE

; ---------- INITIALIZATION ENDS -------------
```

This code initializes variables necessary for the game's logic:

- **PLAYER** is set to 50, representing the player number.
- **MOVES**, **DONE**, and **DR** are set to 0 to track the number of moves, the completion of the game, and whether the game was a draw.
- Variables **C1 to C9** are initialized with ASCII codes for the digits 1 through 9, possibly representing the cells on a game board.
- Finally, the program jumps to **PLRCHANGE**, which likely handles the player-changing logic or further game processes.

3) VICTORY

This code snippet is responsible for displaying a message indicating the victory of a player and handling some input operations in DOS-based assembly programming.

```
; ------------ VICTORY ------------------------
    VICTORY:

            LEA DX, W1
            MOV AH, 9   ; invokes the DOS interrupt for displaying a string.
            INT 21H     ;displaying the string at the address pointed.

            LEA DX, PLAYER
            MOV AH, 9   ;This displays the number of the winning player.
            INT 21H

            LEA DX, W2
            MOV AH, 9
            INT 21H

                ; SET CURSOR
                MOV AH, 2
                MOV DH, 17
                MOV DL, 28
                INT 10H


            MOV AH, 7     ; INPUT WITHOUT ECHO
            INT 21H

            JMP TRYAGAIN
```

This code handles the victory display for a player:

- Displays a victory message (e.g., "Player X wins!").
- Moves the cursor to a specific position on the screen (row 17, column 28).
- Waits for input from the user without echoing the pressed key.
- After receiving input, the program jumps to **TRYAGAIN**, presumably to give the option to restart or try again.

4)DRAW

This part of the code handles the scenario where the game ends in a draw:

```
DRAW:
        LEA DX, DRW
        MOV AH, 9
        INT 21H


            ; SET CURSOR
        MOV AH, 2
        MOV DH, 17
        MOV DL, 28
        INT 10H


                MOV AH, 7    ; INPUT WITHOUT ECHO
        INT 21H
        JMP TRYAGAIN
```

- It displays a message (likely "It's a Draw!").
- Moves the cursor to a specific screen position (row 17, column 28).
- Waits for the player to press a key (without displaying it on the screen).
- Jumps to the **TRYAGAIN** label, which presumably handles restarting or prompting the player to play again.

5)Check

The CHECK routine is responsible for determining if a player has won the game by checking all possible winning combinations. Additionally, it verifies if the game ends in a draw when all positions are filled and no winner is found.

```
CHECK:    ; THERE ARE 8 POSSIBLE WINNING COMBINATIONS

    CHECK1:   ; CHECKING 1, 2, 3
        MOV AL, C1
        MOV BL, C2
        MOV CL, C3

        CMP AL, BL
        JNZ CHECK2

        CMP BL, CL
        JNZ CHECK2

        MOV DONE, 1
        JMP BOARD
```

- This code is responsible for checking all possible winning combinations in a Tic-Tac-Toe game.
- There are 8 possible ways to win:
- Three horizontal lines (1-2-3, 4-5-6, 7-8-9)
- Three vertical lines (1-4-7, 2-5-8, 3-6-9)
- Two diagonals (1-5-9, 3-5-7)
- For each combination, the code compares the values in three cells. If they are equal, it sets DONE to 1 and jumps to BOARD, indicating a win.
- If no combination results in a win, it checks if the game is a draw by calling DRAWCHECK.

6) Check for a Draw

If no winning combination is found, the program checks for a draw

```
DRAWCHECK:
      MOV AL, MOVES
      CMP AL, 9
      JB PLRCHANGE

      MOV DR, 1
      JMP BOARD

      JMP EXIT
```

This code checks the number of moves made in the game:

- If the moves are less than 9, it jumps to **PLRCHANGE** to continue the game.
- If the moves have reached 9 (meaning all cells are filled and no winner has been declared), it sets **DR** to 1 to mark a draw and jumps to **BOARD** to handle the draw condition.
- If neither of these conditions is met, it jumps to **EXIT** to either end or exit the process.

7) PLRCHANGE

changing the player turn during the game. In games like Tic-Tac-Toe, players take turns, and this logic alternates between Player 1 and Player 2.

```
PLRCHANGE:

    CMP PLAYER, 49
    JZ P2
    CMP PLAYER, 50
    JZ P1

    P1:
        MOV PLAYER, 49
        MOV CUR, 88


        JMP BOARD

    P2:
        MOV PLAYER, 50
        MOV CUR, 79
        JMP BOARD
```

- **CMP PLAYER, 49** checks if it's Player 1's turn. If true, it jumps to **P2**.
- If Player 1 is playing, the code in **P1** sets **PLAYER** to 49 and sets **CUR** to 88 (representing the 'X' symbol), then jumps to **BOARD**.
- **CMP PLAYER, 50** checks if it's Player 2's turn. If true, it jumps to **P1**.
- If Player 2 is playing, the code in **P2** sets **PLAYER** to 50 and sets **CUR** to 79 (representing the 'O' symbol), then jumps to **BOARD**.

This alternates the turns between Player 1 ('X') and Player 2 ('O'), ensuring that both players get a chance to play their moves.

8) Input

This part of the code handles the player's input for the game. It interacts with the user, validates the input, and updates the necessary variables.

```
INPUT:
 LEA DX, W1
 MOV AH, 9
 INT 21H

 MOV AH, 2
 MOV DL, PLAYER
 INT 21H

 CMP PLAYER, 49
 JZ PL1

      LEA DX, PC2
      MOV AH, 9
      INT 21H
      JMP TAKEINPUT

      PL1:
           LEA DX, PC1
           MOV AH, 9
           INT 21H

 TAKEINPUT:
 LEA DX, INP
 MOV AH, 9
 INT 21H

 MOV AH, 1
 INT 21H

 INC MOVES ; INCREMENTING MOVES COUNTER BY 1

 MOV BL, AL
 SUB BL, 48

 MOV CL, CUR

 ; CHECKING IF INPUT IS BETWEEN 1-9
 CMP BL, 1
 JZ  C1U

 CMP BL, 2
 JZ  C2U

 CMP BL, 3
 JZ  C3U

 CMP BL, 4
 JZ  C4U

 CMP BL, 5
 JZ  C5U
```

- **Display Message:** A message is displayed indicating the player's current turn.
- **Player identification:** The code checks the current player (1 or 2) and displays the appropriate message.
- **Input Request:** A request to enter the number from the player is displayed.
- **Receive input:** Read the number entered by the player.
- **Input Validation:** Ensure that the entered number is between 1 and 9.
- **Increase the move counter:** The move counter is updated once the player enters his number.

This piece of code ensures that the game receives correct input and prepares it for the next steps.

9) IF INPUT IS INVALID

code manages player inputs, ensures the validity of the input, updates the game board, and checks for any winning conditions or errors.

```
TRYAGAIN:
        ; CLEAR SCREEN
        MOV AX,0600H
        MOV BH,07H
        MOV CX,0000H
        MOV DX,184FH
        INT 10H

        ; SET CURSOR
        MOV AH, 2
        MOV BH, 0
        MOV DH, 10
        MOV DL, 24
        INT 10

    LEA DX, TRA    ; Address of the "TRY AGAIN" prompt text.
    MOV AH, 9      ; DOS function to display a string.
    INT 21H        ; Displays the string at the address DX points to.

    MOV AH, 1
    INT 21H        ; Waits for the user to input a single character and stores it in AL.

    CMP AL, 121    ; Check if input is 'y' (ASCII 121).
    JZ INIT         ; Jump to INIT to restart the game if true.

    CMP AL, 89    ; Check if input is 'Y' (ASCII 89).
    JZ INIT        ; Jump to INIT to restart the game if true.

    ; IF INPUT IS 'Y'/'y' THEN REPEAT THE GAME
    CMP AL, 110  ; Check if input is 'n' (ASCII 110).
    JZ EXIT        ; Jump to EXIT to terminate the program.

    CMP AL, 78    ; Check if input is 'N' (ASCII 78).
    JZ EXIT       ; Jump to EXIT to terminate the program.

    ; IF INPUT IS 'N'/'n' THEN EXIT THE GAME

    ; IF INPUT IS INVALID
        ; Reset the cursor to the "Try Again" prompt position
        MOV AH, 2
        MOV BH, 0
        MOV DH, 10
        MOV DL, 24
        INT 10H

    LEA DX, WI   ; Address of the "Wrong Input" text.
    MOV AH, 9    ; DOS function to display a string.
    INT 21H      ; Displays the string.

    MOV AH, 7 ; INPUT WITHOUT ECHO
    INT 21H

    LEA DX, EMP  ;Address of an empty string.
    MOV AH, 9     ; DOS function to display a string.
    INT 21H       ; Clears the previous prompt area.

    JMP TRYAGAIN ; PROMPT THE TRY AGAIN
```

```
        ; SET CURSOR
        MOV AH, 2
        MOV DH, 16
        MOV DL, 20
        INT 10H

    JMP INPUT
; ADJUST |
; SETTING BOARD POSITION AS INPUT MARK
    C1U:
        CMP C1, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C1, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C1, CL
        JMP CHECK
    C2U:
        CMP C2, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C2, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C2, CL
        JMP CHECK
    C3U:
        CMP C3, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C3, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C3, CL
        JMP CHECK
    C4U:
        CMP C4, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C4, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C4, CL
        JMP CHECK
    C5U:
        CMP C5, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C5, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C5, CL
        JMP CHECK
    C6U:
        CMP C6, 88   ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C6, 79   ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C6, CL
        JMP CHECK
    C7U:
        CMP C7, 88    ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C7, 79    ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C7, CL
        JMP CHECK
    C8U:
        CMP C8, 88    ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C8, 79    ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C8, CL
        JMP CHECK
    C9U:
        CMP C9, 88    ; CHECKING IF THE CELL IS ALREADY 'X'
        JZ TAKEN
        CMP C9, 79    ; CHECKING IF THE CELL IS ALREADY 'O'
        JZ TAKEN

        MOV C9, CL
        JMP CHECK
```

Key Variables

- **MOVES:** Tracks the number of moves made during the game.
- **CL:** Holds the current player's symbol (X or O).
- **C1 to C9:** Represent the nine cells on the game board.
- **TKN:** Message: "Cell is already taken."
- **EMP:** Blank line used to clear previous messages.
- **WI:** Message: "Invalid Input."

Primary Functions

1. **Input Management:** Ensures that player input is valid.
2. **Cell Validation:** Prevents overwriting occupied cells.
3. **Board Update:** Places the current player's symbol in the selected cell.
4. **Game State Check:** Verifies if a winning condition is met after every move.

Conclusion

This segment of the code ensures:

- Invalid inputs are handled gracefully.
- Players cannot overwrite occupied cells.
- The game board is updated correctly with each move.
- The game state is checked to determine if a player has won or if the game should continue.

10) TRY AGAIN

This section of the code handles the functionality for restarting or exiting the game based on user input. It also manages invalid inputs and prompts the user accordingly.

```
TRYAGAIN:
        ; CLEAR SCREEN
        MOV AX,0600H
        MOV BH,07H
        MOV CX,0000H
        MOV DX,184FH
        INT 10H

        ; SET CURSOR
        MOV AH, 2
        MOV BH, 0
        MOV DH, 10
        MOV DL, 24
        INT 10

    LEA DX, TRA    ; Address of the "TRY AGAIN" prompt text.
    MOV AH, 9      ; DOS function to display a string.
    INT 21H        ; Displays the string at the address DX points to.

    MOV AH, 1
    INT 21H        ; Waits for the user to input a single character and stores it in AL.

    CMP AL, 121    ; Check if input is 'y' (ASCII 121).
    JZ INIT         ; Jump to INIT to restart the game if true.

    CMP AL, 89    ; Check if input is 'Y' (ASCII 89).
    JZ INIT        ; Jump to INIT to restart the game if true.

    ; IF INPUT IS 'Y'/'y' THEN REPEAT THE GAME
    CMP AL, 110   ; Check if input is 'n' (ASCII 110).
    JZ EXIT        ; Jump to EXIT to terminate the program.

    CMP AL, 78    ; Check if input is 'N' (ASCII 78).
    JZ EXIT        ; Jump to EXIT to terminate the program.

    ; IF INPUT IS 'N'/'n' THEN EXIT THE GAME

    ; IF INPUT IS INVALID
        ; Reset the cursor to the "Try Again" prompt position
        MOV AH, 2
        MOV BH, 0
        MOV DH, 10
        MOV DL, 24
        INT 10H

    LEA DX, WI   ; Address of the "Wrong Input" text.
    MOV AH, 9    ; DOS function to display a string.
    INT 21H      ; Displays the string.

    MOV AH, 7 ; INPUT WITHOUT ECHO
    INT 21H

    LEA DX, EMP  ;Address of an empty string.
    MOV AH, 9    ; DOS function to display a string.
    INT 21H      ; Clears the previous prompt area.

    JMP TRYAGAIN ; PROMPT THE TRY AGAIN
```

This section of the code handles the functionality for restarting or exiting the game based on user input. It also manages invalid inputs and prompts the user accordingly.

1. Clears the screen and displays the "Try Again" prompt.
2. Waits for user input:
    - **'Y'/'y':** Restarts the game.
    - **'N'/'n':** Exits the program.
    - **Invalid input:** Displays an error message and re-prompts.
3. Ensures the user provides valid input before proceeding.

    This routine ensures smooth user interaction and robust error handling.