Fatima Jinnah Women University

Opening Portals of Excellence Through Higher Education

FINAL LAB PROJECT:

SUBMITTED TO:

- **ENGR. MOHAMMAD SHOAIB**

SUBMITTED BY:

- **EMAN ZAI (2022-BSE-049)**
- **AREEJ INTISHAD (2022-BSE-046)**
- **SYEDA FARWA BATOOL (2022-BSE-071)**

COURSE TITLE:

- **COMPUTER ARCHITECTURE AND LOGIC DESIGN**

TIC TAC TOE GAME:

The provided code is written in 8086 assembly language and emu8086 is mentioned in the comments, which is a popular emulator for 8086 assembly language development. The code implements a simple tic-tac-toe game for two players (X and O) using the console.

Here's a breakdown limited introduction of the code:

1. **Data Segment**:
    - It initializes various data variables such as **new_line** for formatting, the game grid **game_draw**, **game_pointer** for managing grid positions, **win_flag** to determine if a player has won, and **player** to keep track of the current player.
2. **Stack Segment**:
    - The stack segment is defined but not utilized explicitly in the provided code.
3. **Code Segment**:
    - The **start** label marks the beginning of the program.
    - **set_game_pointer** initializes the game grid's pointers.
    - The main game loop (**main_loop**) repeatedly clears the screen, displays game messages, draws the game board, and handles player inputs until a win condition is met.
    - Various subroutines (**clear_screen**, **print**, **read_keyboard**, etc.) handle specific tasks such as clearing the screen, printing messages, reading keyboard inputs, and updating the game state.
    - **change_player** toggles between players (X and O).
    - **update_draw** updates the game grid based on player inputs.
    - **check** and related subroutines (**check_line**, **check_column**, **check_diagonal**) determine if a player has won by checking the rows, columns, and diagonals.

- The game ends when a player wins (**game_over** subroutine is called), and the final state is displayed.

4. **Functionality**:
   - Players take turns entering their positions (1-9) on the game board.
   - The game checks for a win condition after each move.
   - Once a player wins, the game ends, and the winning player is displayed.
5. **Characteristics:**
   - The code is well-structured with clear subroutine divisions for different tasks.
   - It uses simple ASCII art for the game board.
   - There's no user interface other than the console, and the game is played entirely using keyboard inputs.
   - The game doesn't handle invalid inputs (e.g., entering a position that's already taken), so it might be possible to break the game flow with unexpected inputs.

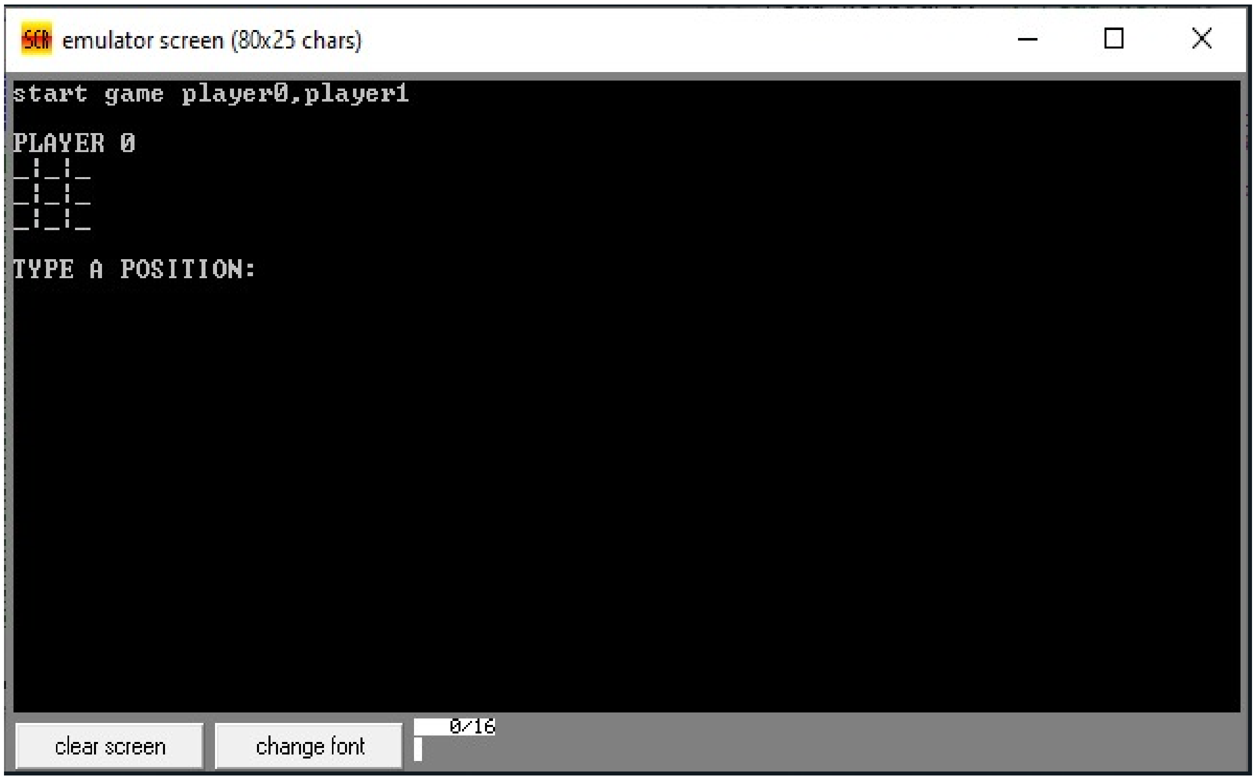
CODE:

```
; 8086 Assembly, emu8086

data segment
    new_line db 13, 10, "$"

    game_draw db "_|_|_", 13, 10
              db "_|_|_", 13, 10
              db "_|_|_", 13, 10, "$"
    game_pointer db 9 DUP(?)
    win_flag db 0
    player db "0$"
    player_message db "PLAYER $"
    win_message db " WIN!$"
    type_message db "TYPE A POSITION: $"
ends
stack segment
    dw   128  dup(?)
ends
```

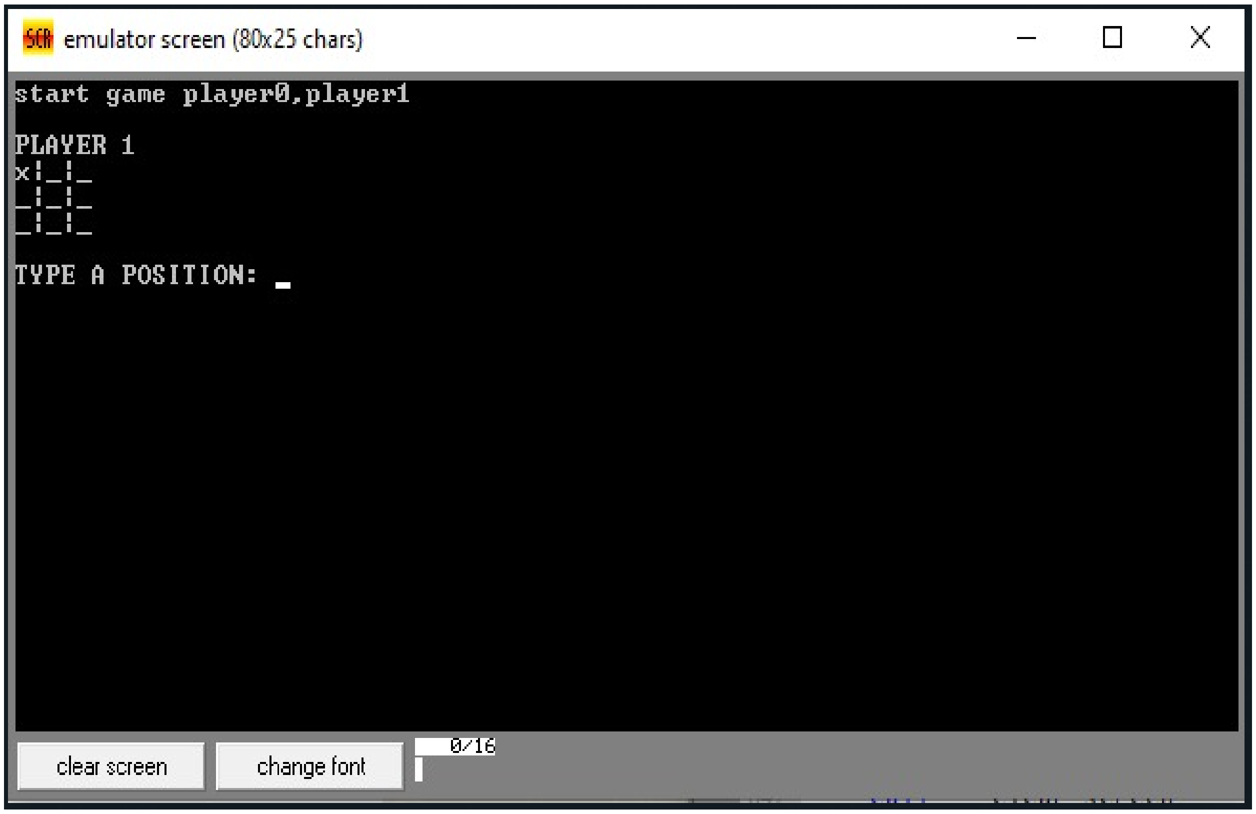
```
extra segment
ends
code segment
start:
    ; set segment registers
    mov     ax, data
    mov     ds, ax
    mov     ax, extra
    mov     es, ax
    ; game start
    call    set_game_pointer
main_loop:
    call    clear_screen
    lea     dx, game_start_message
    call    print
    lea     dx, new_line
    call    print
    lea     dx, player_message
    call    print
    lea     dx, player
    call    print
    lea     dx, new_line
    call    print
    lea     dx, game_draw
    call    print
    lea     dx, new_line
    call    print
    lea     dx, type_message
    call    print
    ; read draw position
    call    read_keyboard
    ; calculate draw position
    sub     al, 49
    mov     bh, 0
    mov     bl, al
    call    update_draw
    call    check
    ; check if game ends
    cmp     win_flag, 1
    je      game_over
    call    change_player
    jmp     main_loop
change_player:
    lea     si, player
    xor     ds:[si], 1
    ret
```

```
update_draw:
    mov     bl, game_pointer[bx]
    mov     bh, 0
    lea     si, player
    cmp     ds:[si], "0"
    je      draw_x
    cmp     ds:[si], "1"
    je      draw_o
    draw_x:
    mov     cl, "x"
    jmp     update
    draw_o:
    mov     cl, "o"
    jmp     update
    update:
    mov     ds:[bx], cl
    ret
check:
    call    check_line
    ret
check_line:
    mov     cx, 0
    check_line_loop:
    cmp     cx, 0
    je      first_line
    cmp     cx, 1
    je      second_line
    cmp     cx, 2
    je      third_line
    call    check_column
    ret
    first_line:
    mov     si, 0
    jmp     do_check_line
    second_line:
    mov     si, 3
    jmp     do_check_line
    third_line:
    mov     si, 6
    jmp     do_check_line
    do_check_line:
    inc     cx
    mov     bh, 0
    mov     bl, game_pointer[si]
    mov     al, ds:[bx]
    cmp     al, "_"
    je      check_line_loop
    inc     si
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_line_loop
    inc     si
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_line_loop
    mov     win_flag, 1
    ret
check_column:
    mov     cx, 0
    check_column_loop:
    cmp     cx, 0
    je      first_column
    cmp     cx, 1
    je      second_column
    cmp     cx, 2
    je      third_column
    call    check_diagonal
    ret
    first_column:
    mov     si, 0
    jmp     do_check_column
    second_column:
    mov     si, 1
    jmp     do_check_column
    third_column:
    mov     si, 2
    jmp     do_check_column
    do_check_column:
    inc     cx
    mov     bh, 0
    mov     bl, game_pointer[si]
    mov     al, ds:[bx]
    cmp     al, "_"
    je      check_column_loop
    add     si, 3
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_column_loop
    add     si, 3
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_column_loop
    mov     win_flag, 1
    ret
check_diagonal:
    mov     cx, 0
    check_diagonal_loop:
    cmp     cx, 0
    je      first_diagonal
    cmp     cx, 1
    je      second_diagonal
    ret
    first_diagonal:
    mov     si, 0
    mov     dx, 4 ;fasiulhaq
    jmp     do_check_diagonal
    second_diagonal:
    mov     si, 2
    mov     dx, 2
    jmp     do_check_diagonal
    do_check_diagonal:
    inc     cx
    mov     bh, 0
    mov     bl, game_pointer[si]
    mov     al, ds:[bx]
    cmp     al, "_"
    je      check_diagonal_loop
    add     si, dx
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_diagonal_loop
    add     si, dx
    mov     bl, game_pointer[si]
    cmp     al, ds:[bx]
    jne     check_diagonal_loop
    mov     win_flag, 1
    ret
game_over:
    call    clear_screen
    lea     dx, game_start_message
    call    print
    lea     dx, new_line
    call    print
    lea     dx, game_draw
    call    print
    lea     dx, new_line
    call    print
    lea     dx, game_over_message
    call    print
    lea     dx, player_message
    call    print
    lea     dx, player
    call    print
    lea     dx, win_message
    call    print
    jmp     fim
set_game_pointer:
    lea     si, game_draw
    lea     bx, game_pointer
    mov     cx, 9
    loop_1:
    cmp     cx, 6
    je      add_1
    cmp     cx, 3
    je      add_1
    jmp     add_2
    add_1:
    add     si, 1
    jmp     add_2
    add_2:
    mov     ds:[bx], si
    add     si, 2
    inc     bx
    loop    loop_1
    ret
print:      ; print dx content
    mov     ah, 9
    int     21h
    ret
clear_screen:       ; get and set video mode
    mov     ah, 0fh
    int     10h
    mov     ah, 0
    int     10h
    ret
read_keyboard:  ; read keybord and return content in ah
    mov     ah, 1
    int     21h
    ret
fim:
    jmp     fim

code ends
end start
```

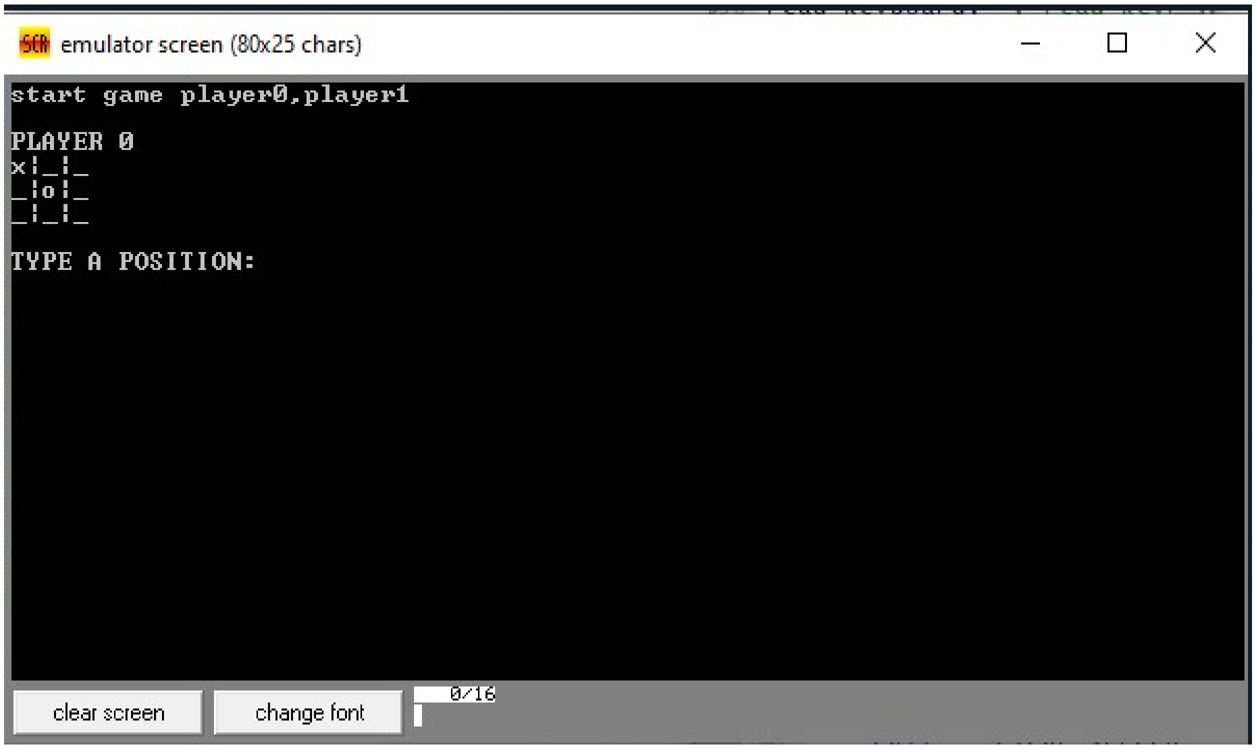
OUTPUT:



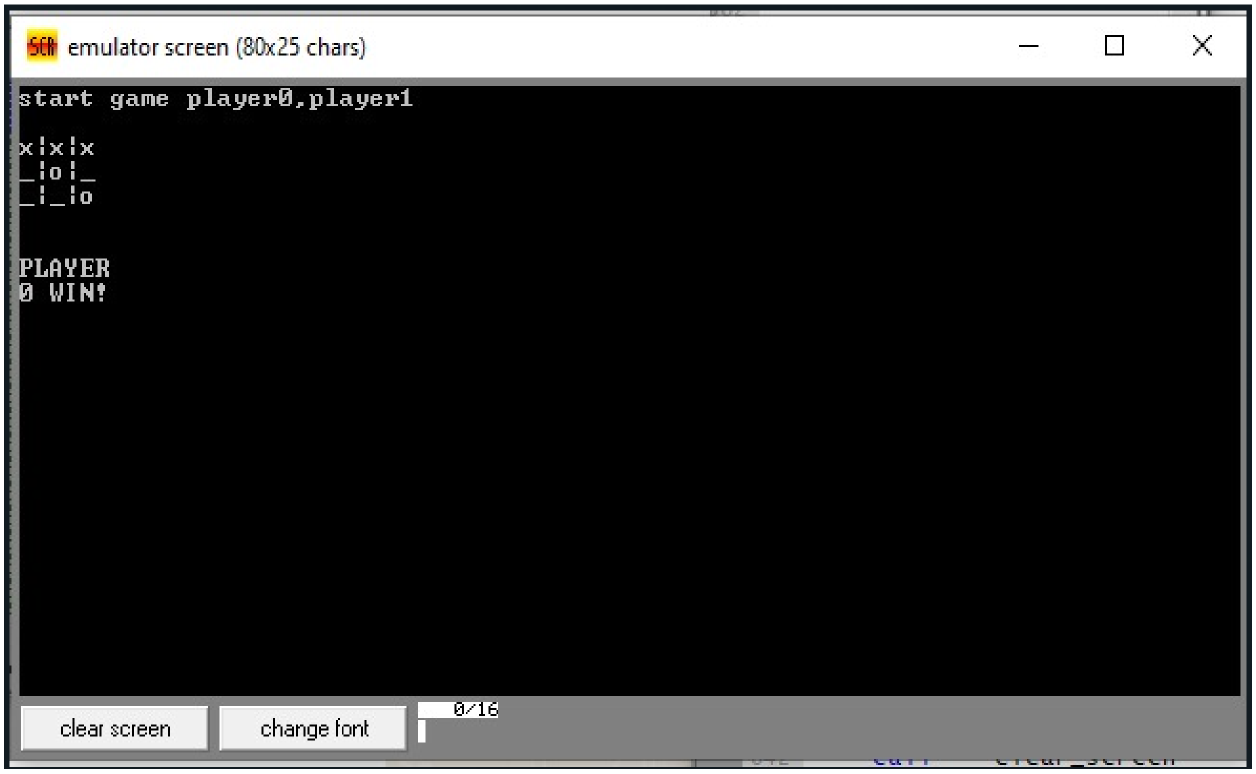
```
start game player0,player1

PLAYER 0
_|_|_
_|_|_
_|_|_

TYPE A POSITION:
```



```
start game player0,player1

PLAYER 1
x|_|_
_|_|_
_|_|_

TYPE A POSITION: _
```



```
start game player0,player1

PLAYER 0
x¦_¦_
_¦o¦_
_¦_¦_

TYPE A POSITION:
```



```
start game player0,player1

x¦x¦x
_¦o¦_
_¦_¦o


PLAYER
0 WIN!
```