HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÀI TẬP LỚN
KIẾN TRÚC MÁY TÍNH

Sinh viên thực hiện: Lăng Viết Thành
Mã sinh viên: B23DCCN768
Nhóm: 06
Giảng viên hướng dẫn: Trần Tiến Công

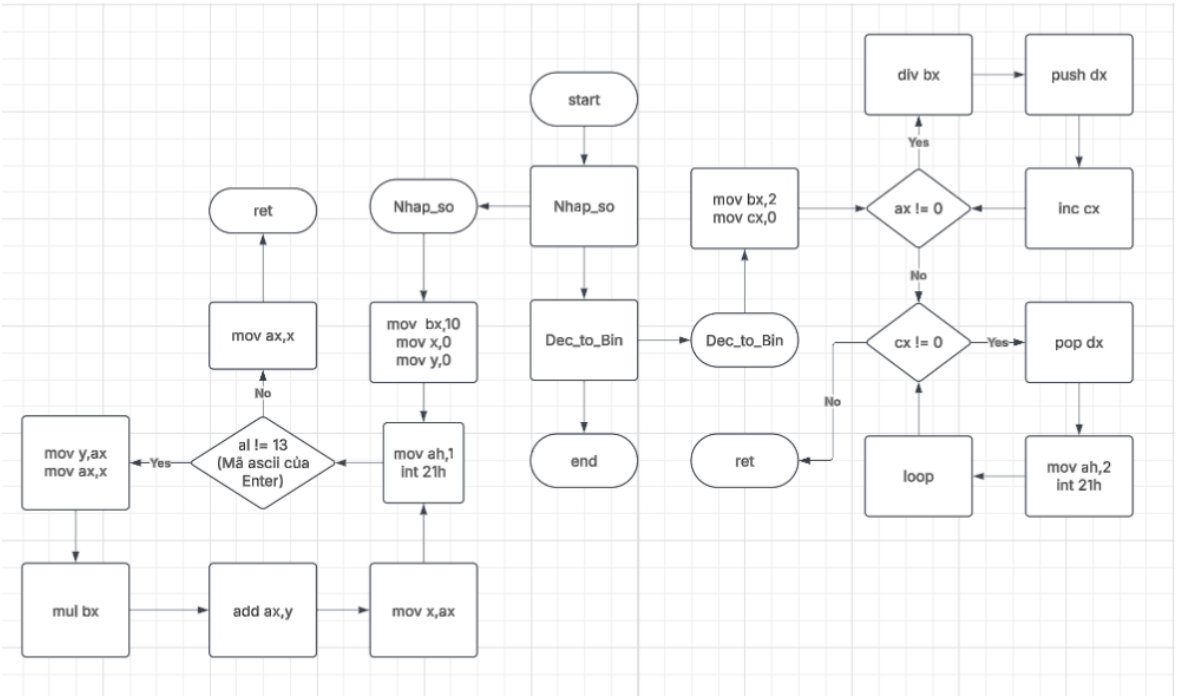
Hà Nội, 2025

Mục lục

PHẦN 1: BÀI TẬP CÁ NHÂN

BÀI 1:

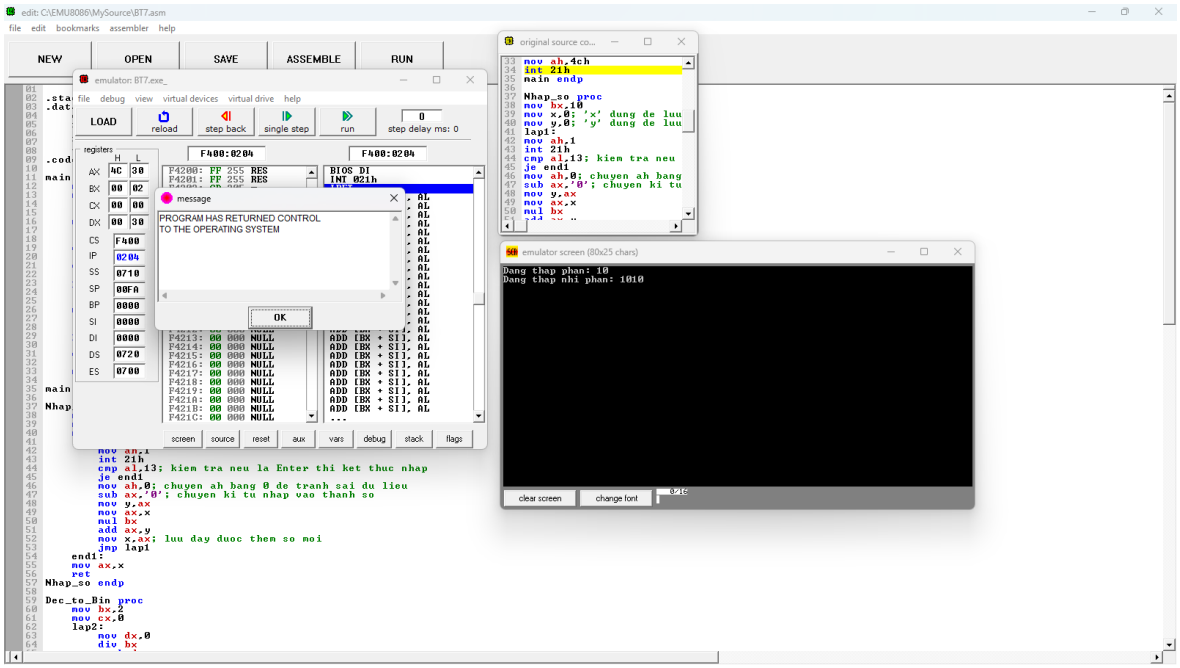
Bài số 7: Viết chương trình hợp ngữ Assembly chuyển một số từ hệ cơ số 10 sang hệ nhị phân



Hình 1: Flowchart bài 7: Chuyển số hệ 10 sang nhị phân

```
.model small
.stack 100h
.data
    crlf db 13,10,'$'
    x dw ?
    y dw ?
    tb1 db 'Dang thap phan: ','$'
    tb2 db 'Dang thap nhi phan: ','$'
.code

main proc
    mov ax,@data
    mov ds,ax

    lea dx,tb1
    mov ah,9
    int 21h

    call Nhap_so

    call endl

    push ax; day ax vao stack de ax khong bi thay doi khi in tb2

    lea dx,tb2
    mov ah,9
    int 21h

    pop ax

    call Dec_to_Bin

    mov ah,4ch
    int 21h
main endp

Nhap_so proc
    mov bx,10
    mov x,0; 'x' dung de luu day so truoc do
    mov y,0; 'y' dung de luu so vua nhap
    lap1:
        mov ah,1
        int 21h
        cmp al,13; kiem tra neu la Enter thi ket thuc nhap
        je end1
        mov ah,0; chuyen ah bang 0 de tranh sai du lieu
        sub ax,'0'; chuyen ki tu nhap vao thanh so
        mov y,ax
        mov ax,x
        mul bx
        add ax,y
        mov x,ax; luu day so duoc them so moi
        jmp lap1
    end1:
    mov ax,x
    ret
Nhap_so endp

Dec_to_Bin proc
    mov bx,2
    mov cx,0
    lap2:
        mov dx,0
        div bx
        push dx
        inc cx
        cmp ax,0
        jg lap2
    lap3:
        pop dx
        add dx,'0'
        mov ah,2
        int 21h
        loop lap3
    ret
Dec_to_Bin endp

; xuong dong
endl proc
    push ax
    push dx
    lea dx,crlf
    mov ah,9
    int 21h
    pop dx
    pop ax
    ret
endl endp

end
```

Hình 2: Code bài 7



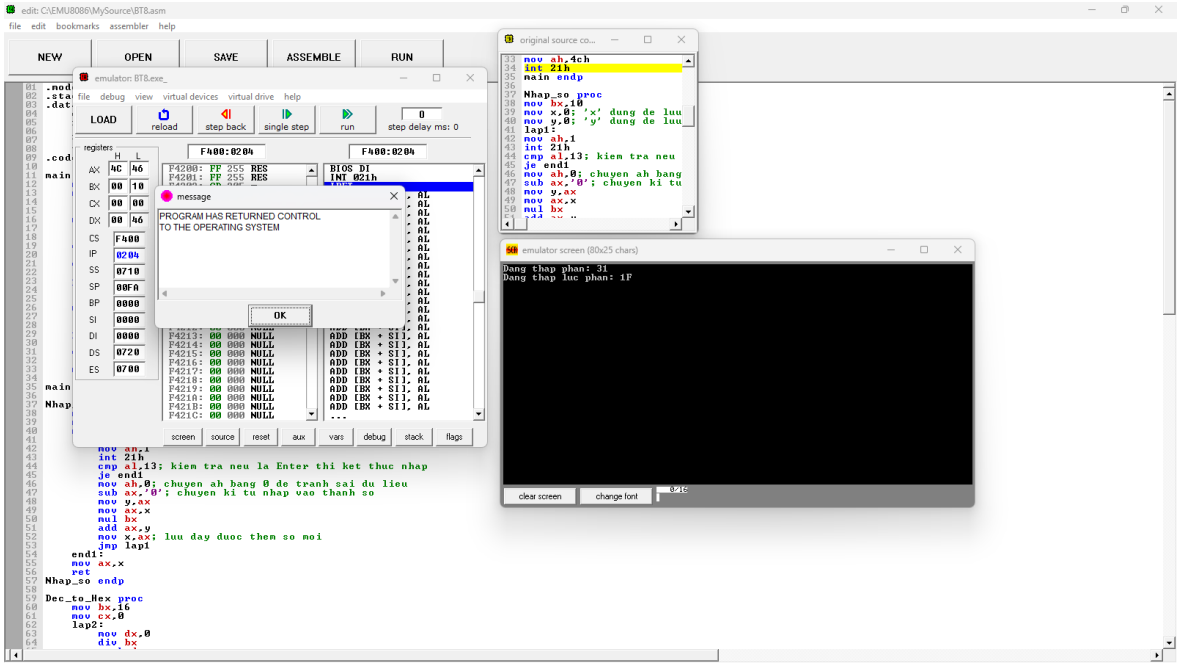
Hình 3: Testcase bài 7

Bài số 8: Viết chương trình hợp ngữ Assembly chuyển một số từ hệ cơ số 10 sang hệ cơ số 16 (Hexa)

Hình 4: Flowchart bài 8: Chuyển số hệ 10 sang hệ 16

```
.model small
.stack 100h
.data
    crlf db 13,10,'$'
    x dw ?
    y dw ?
    tb1 db 'Dang thap phan: ','$'
    tb2 db 'Dang thap luc phan: ','$'
.code

main proc
    mov ax,@data
    mov ds,ax

    lea dx,tb1
    mov ah,9
    int 21h

    call Nhap_so

    call endl

    push ax; day ax vao stack de ax khong bi thay doi khi in tb2

    lea dx,tb2
    mov ah,9
    int 21h

    pop ax

    call Dec_to_Hex

    mov ah,4ch
    int 21h
main endp

Nhap_so proc
    mov bx,10
    mov x,0; 'x' dung de luu day so truoc do
    mov y,0; 'y' dung de luu so vua nhap
    lap1:
        mov ah,1
        int 21h
        cmp al,13; kiem tra neu la Enter thi ket thuc nhap
        je end1
        mov ah,0; chuyen ah bang 0 de tranh sai du lieu
        sub ax,'0'; chuyen ki tu nhap vao thanh so
        mov y,ax
        mov ax,x
        mul bx
        add ax,y
        mov x,ax; luu day duoc them so moi
        jmp lap1
    end1:
    mov ax,x
    ret
Nhap_so endp

Dec_to_Hex proc
    mov bx,16
    mov cx,0
    lap2:
        mov dx,0
        div bx
        push dx
        inc cx
        cmp ax,0
        jg lap2
    lap3:
        pop dx
        cmp dx,10
        jl print
        sub dx,10
        add dx,'A'
        sub dx,'0'
        print:
        add dx,'0'
        mov ah,2
        int 21h
        loop lap3
    ret
Dec_to_Hex endp

; xuong dong
endl proc
    push ax
    push dx
    lea dx,crlf
    mov ah,9
    int 21h
    pop dx
    pop ax
    ret
endl endp

end
```

Hình 5: Code bài 8



Hình 6: Testcase bài 8

Bài số 14: Viết chương trình hợp ngữ Assembly tính ƯCLN và BCNN

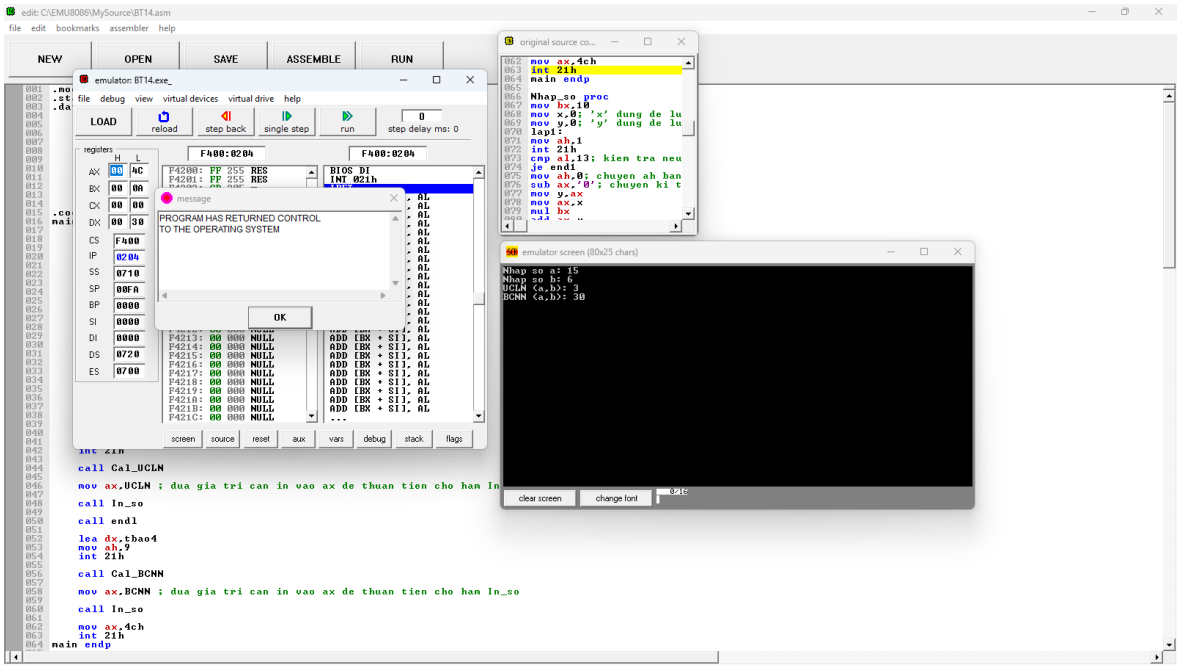
Hình 7: Flowchart bài 14: Tính ƯCLN và BCNN

```
.model small
.stack 100h
.data
    crlf db 13,10,'$'
    x dw ?
    y dw ?
    a dw ?
    b dw ?
    UCLN dw ?
    BCNN dw ?
    thao1 db 'Nhap so a: $'
    thao2 db 'Nhap so b: $'
    thao3 db 'UCLN (a,b): $'
    thao4 db 'BCNN (a,b): $'
.code
main proc
    mov ax,@data
    mov ds,ax

    lea dx,thao1
    mov ah,9
    int 21h

    call Nhap_so

    mov a,ax

    call endl

    lea dx,thao2
    mov ah,9
    int 21h

    call Nhap_so

    mov b,ax

    call endl

    lea dx,thao3
    mov ah,9
    int 21h

    call Cal_UCLN

    mov ax,UCLN ; dua gia tri can in vao ax de thuan tien cho ham In_so

    call In_so

    call endl

    lea dx,thao4
    mov ah,9
    int 21h

    call Cal_BCNN

    mov ax,BCNN ; dua gia tri can in vao ax de thuan tien cho ham In_so

    call In_so

    mov ax,4ch
    int 21h
main endp
```

```
Nhap_so proc
    mov bx,10
    mov x,0; 'x' dung de luu day so truoc do
    mov y,0; 'y' dung de luu so vua nhap
    lap1:
        mov ah,1
        int 21h
        cmp al,13; kiem tra neu la Enter thi ket thuc nhap
        je end1
        mov ah,0; chuyen ah bang 0 de tranh sai du lieu
        sub ax,'0'; chuyen ki tu nhap vao thanh so
        mov y,ax
        mov ax,x
        mul bx
        add ax,y
        mov x,ax; luu day duoc them so moi
        jmp lap1
    end1:
    mov ax,x
    ret
Nhap_so endp

In_so proc
    mov bx,10
    mov cx,0
    lap3:
        mov dx,0
        div bx
        push dx
        inc cx
        cmp ax,0
        jg lap3
    lap4:
        pop dx
        add dx,'0'
        mov ah,2
        int 21h
        loop lap4
    ret
In_so endp

Cal_UCLN proc
    mov ax,a
    mov bx,b
    lap2:
        cmp bx,0
        je end2
        mov dx,0
        div bx
        mov ax,bx
        mov bx,dx
        jmp lap2
    end2:
    mov UCLN,ax
    ret
Cal_UCLN endp
```

```
Cal_UCLN proc
    mov ax,a
    mov bx,b
    lap2:
        cmp bx,0
        je end2
        mov dx,0
        div bx
        mov ax,bx
        mov bx,dx
        jmp lap2
    end2:
    mov UCLN,ax
    ret
Cal_UCLN endp

Cal_BCNN proc
    mov ax,a
    mul b
    mov bx,UCLN
    div bx
    mov BCNN,ax
    ret
Cal_BCNN endp

endl proc
    push ax
    push dx
    lea dx,crlf
    mov ah,9
    int 21h
    pop dx
    pop ax
    ret
endl endp

end
```

Hình 8: Code bài 14

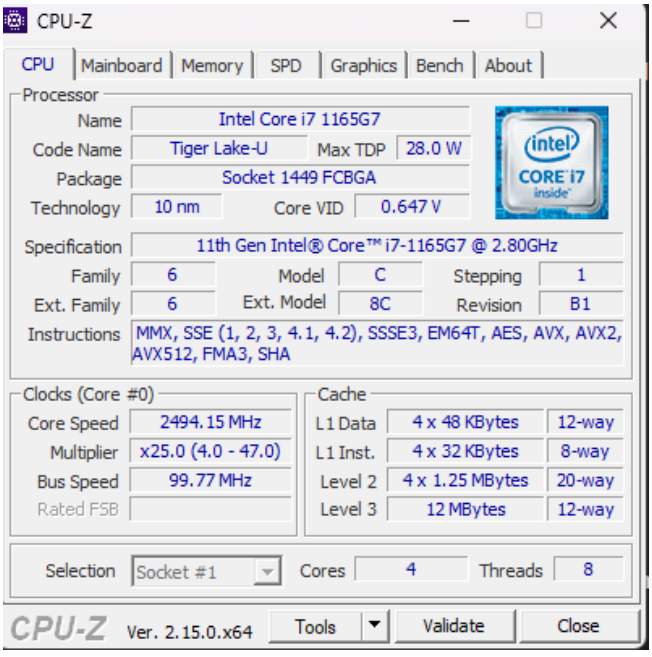


Hình 9: Testcase bài 14

BÀI 2:

1. Khảo sát cấu hình của máy và hệ thống bộ nhớ của máy đang sử dụng (Bộ nhớ trong: ROM, RAM, Cache System, Bộ nhớ ngoài: ổ đĩa cứng, CD, Thiết bị vào ra.)

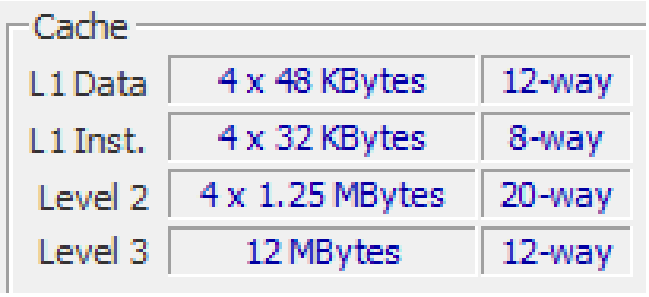
Bộ vi xử lí (CPU)



Hình 10: CPU

- CPU: Intel Core i7-1165G7
- Thế hệ: 11th Gen Tiger Lake-U
- Socket: 1449 FCBGA
- Công nghệ sản xuất: 10nm
- Số nhân / Số luồng: 4 Cores / 8 Threads
- Xung nhịp cơ bản: 2.80 GHz
- Xung nhịp thực tế: 2494.15 MHz (2.49 GHz tại thời điểm khảo sát)

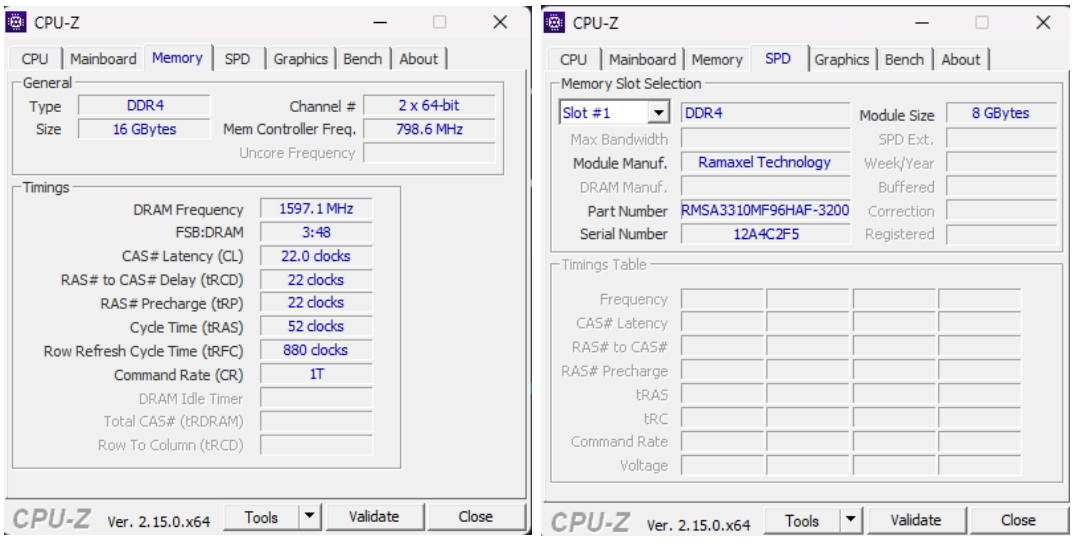
Bộ nhớ đệm (Cache)



Hình 11: Cache

- L1: 4 x 48 KB (Data) + 4 x 32 KB (Instruction)
- L2: 4 x 1.25 MB
- L3: 12 MB

Bộ nhớ trong (RAM)

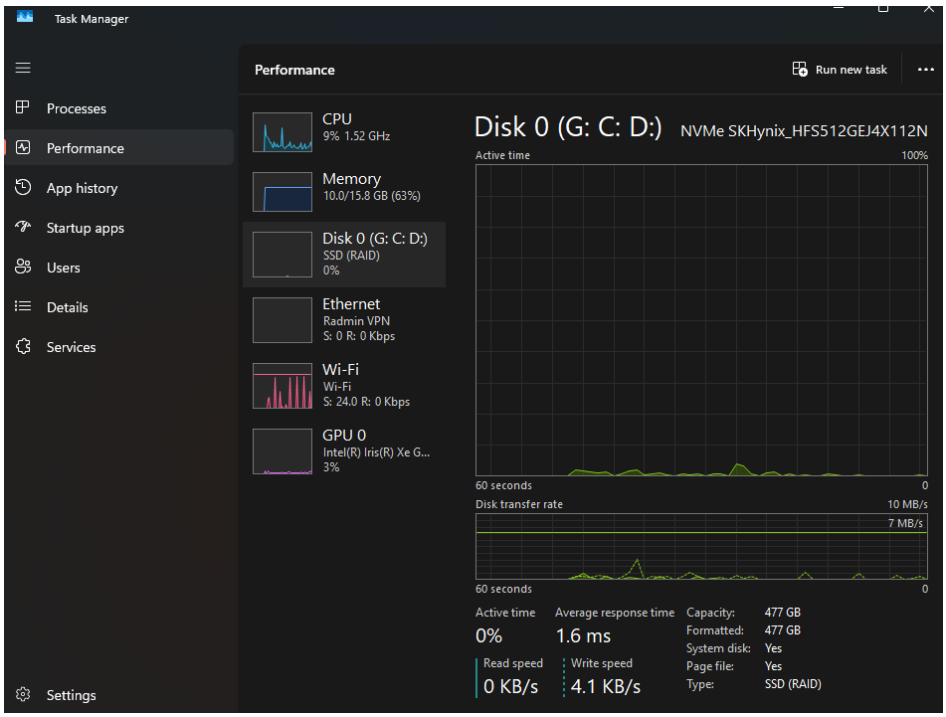


(a) Memory (b) SPD

Hình 12: Cấu hình bộ nhớ trong (RAM)

- Dung lượng tổng: 16 GB DDR4
- Tốc độ DRAM: 1597.1 MHz (tức là 3200 MHz thực tế do DDR: Double Data Rate)
- Hãng sản xuất: Ramaxel Technology
- Cấu hình: 2 kênh (Dual Channel - 2x 64-bit)
- Thông số Timing:
  - CAS Latency (CL): 22
  - RAS to CAS Delay (tRCD): 22
  - RAS Precharge (tRP): 22
  - Cycle Time (tRAS): 52

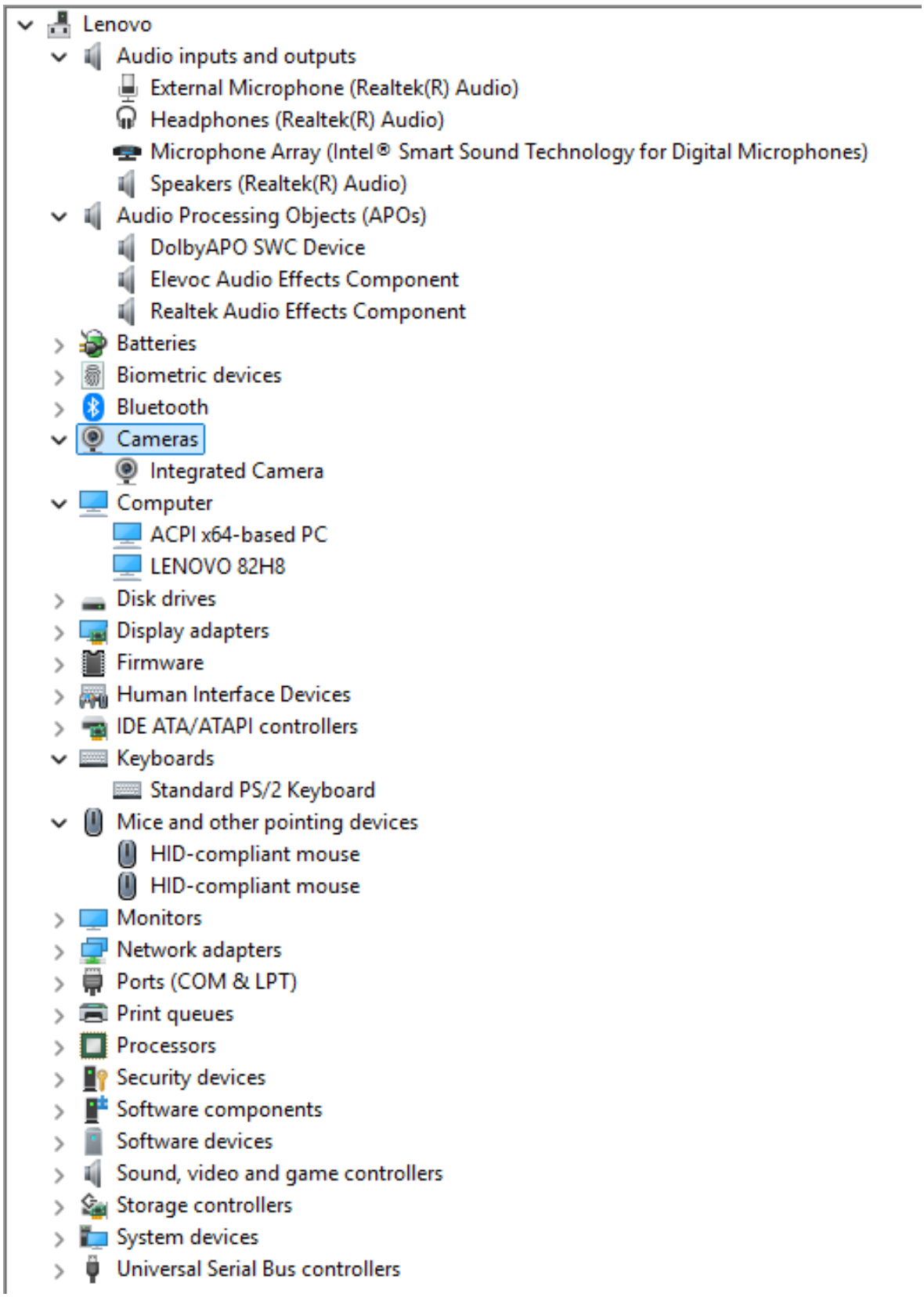
Bộ nhớ ngoài (ROM)



Hình 13: ROM

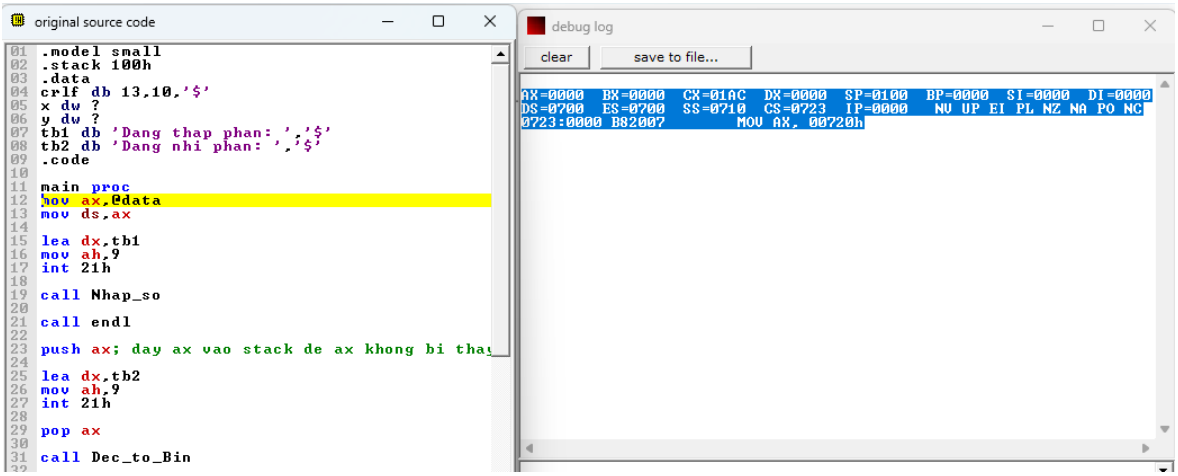
- Ổ cứng chính: SSD NVMe
- Model: SK Hynix HFS512GEJ4X112N
- Dung lượng: 512 GB (hiển thị 477 GB khả dụng)
- Tốc độ phản hồi trung bình: 1.6 ms
- Giao tiếp: NVMe (rất nhanh so với SATA SSD)

Thiết bị vào/ra

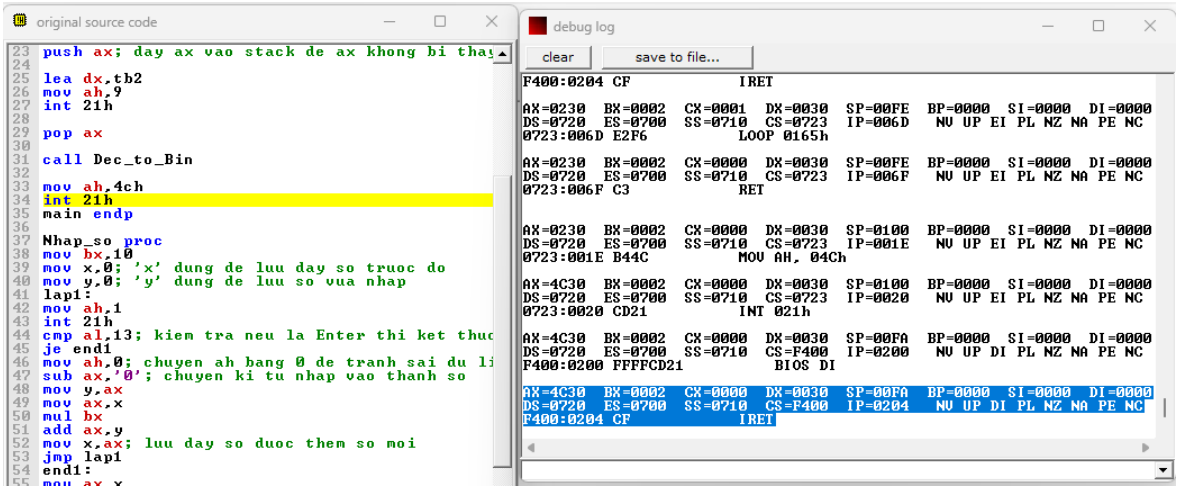


Hình 14: Divice I/O

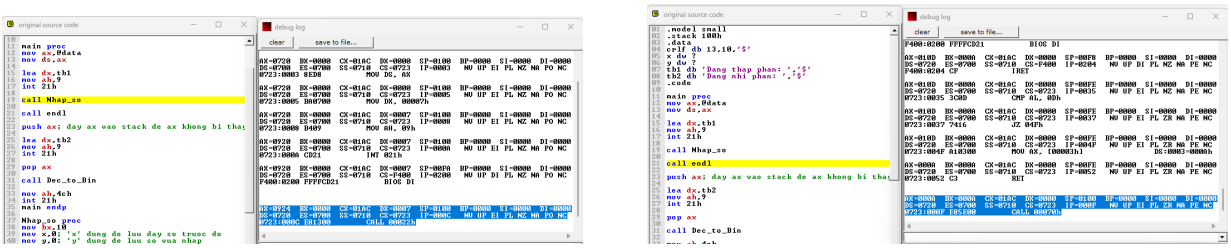
2. Dùng công cụ Debug khảo sát nội dung các thanh ghi IP, DS, ES, SS, CS, BP, SP



Hình 15: Bắt đầu chương trình



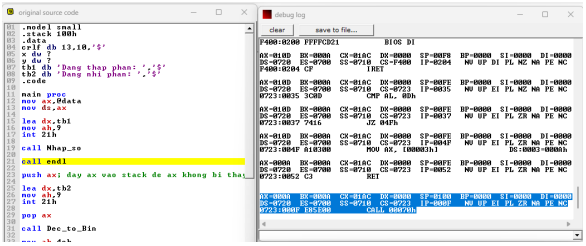
Hình 16: Kết thúc chương trình



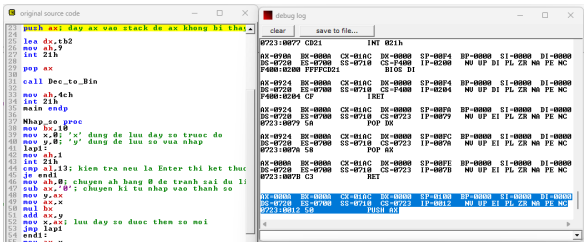
(a) Trước hàm 'Nhap_so'

(b) Sau hàm 'Nhap_so'

Hình 17: Trạng thái của hàm 'Nhap_so'

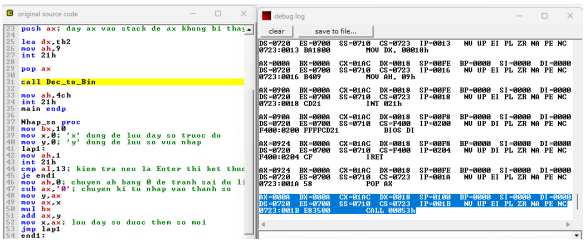


(a) Trước hàm 'endl'

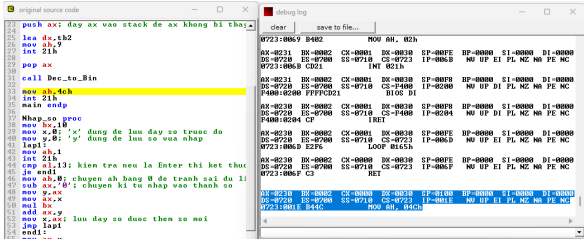


(b) Sau hàm 'Nhap_so'

Hình 18: Trạng thái của hàm 'endl'



(a) Trước hàm 'Dec_to_Bin'



(b) Sau hàm 'Dec_to_Bin'

Hình 19: Trạng thái của hàm 'Dec_to_Bin'

Bảng 1: Giá trị các thanh ghi tại từng bước khảo sát

Bước	CS	IP	SS	SP	BP	DS	ES
Bắt đầu chương trình	0723	0000	0710	0100	0000	0700	0700
Trước hàm Nhap_so	0723	000C	0710	0100	0000	0720	0700
Sau hàm Nhap_so	0723	000F	0710	0100	0000	0720	0700
Trước hàm endl	0723	000F	0710	0100	0000	0720	0700
Sau hàm endl	0723	0012	0710	0100	0000	0720	0700
Trước hàm Dec_to_Bin	0723	001B	0710	0100	0000	0720	0700
Sau hàm Dec_to_Bin	0723	001E	0710	0100	0000	0720	0700
Kết thúc chương trình	F400	0204	0710	00FA	0000	0720	0700

3. Giải thích nội dung các thanh ghi, trên cơ sở đó giải thích cơ chế quản lý bộ nhớ của hệ thống trong trường hợp cụ thể này

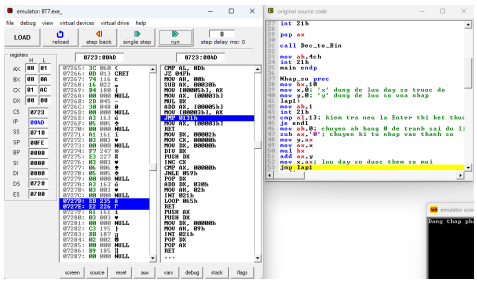
Giải thích nội dung các thanh ghi

- Thanh ghi phân đoạn - Quản lý vùng bộ nhớ
  - CS (Code Segment): Trỏ đến vùng chứa mã lệnh chương trình (vùng .code).
  - DS (Data Segment): Trỏ đến vùng chứa dữ liệu, như biến, chuỗi (vùng .data).
  - SS (Stack Segment): Trỏ đến vùng stack, nơi lưu tạm thời dữ liệu khi push, pop.
  - ES (Extra Segment): Vùng dữ liệu bổ sung, dùng khi làm việc với chuỗi.

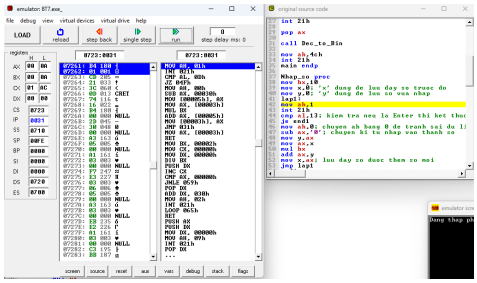
- Thanh ghi con trỏ – Quản lý vị trí trong segment
  - IP (Instruction Pointer): Trỏ tới lệnh sẽ được thực thi tiếp theo. Tự động tăng sau mỗi lệnh.
  - SP (Stack Pointer): Chỉ đến đỉnh stack hiện tại (offset) tính từ SS). Dùng trong push, pop.
  - BP (Base Pointer): Dùng để truy cập dữ liệu trong stack, thường khi truyền tham số cho thủ tục.

Giải thích cơ chế bộ nhớ

- Tổng quan về mô hình bộ nhớ (.model small):
  - .model small: code và dữ liệu nằm trong cùng một segment (phân đoạn), không vượt quá 64KB.
  - Các phân đoạn chính:
    - Code segment (CS): Chứa mã chương trình (main, Nhap_so, Dec_to_Bin, endl)
    - Data segment (DS): Chứa dữ liệu (x, y, tb1, tb2, crlf)
    - Stack segment (SS): Chứa ngăn xếp (lưu tạm ax, địa chỉ trả về,...)
    - Extra Segment (ES): Thường dùng để hỗ trợ thao tác chuỗi/dữ liệu mở rộng
- CS (Code Segment) và IP (Instruction Pointer):
  - CS:IP kết hợp để xác định địa chỉ thực của lệnh đang được thực thi
  - Khi chương trình bắt đầu:
    - CS trỏ đến đoạn mã lệnh (do hệ điều hành MS-DOS nạp).
    - IP chứa offset đến lệnh hiện tại.
  - Mỗi khi CPU thực hiện xong một lệnh, nó cập nhật IP để trỏ đến lệnh tiếp theo
  - Với những lệnh thông thường như MOV, ADD, SUB,... thì IP sẽ trỏ đến những câu lệnh một cách tuần tự từ trên xuống
  - Với những lệnh như JMP, CALL, RET thì chương trình không còn chạy tuần tự, mà sẽ nhảy đến nơi khác theo chỉ định.
    - Lệnh JMP: Sau khi thực hiện 'JMP lap1' thì IP lại trỏ đến lệnh sau label (nhãn) lap1 là lệnh MOV AH,1



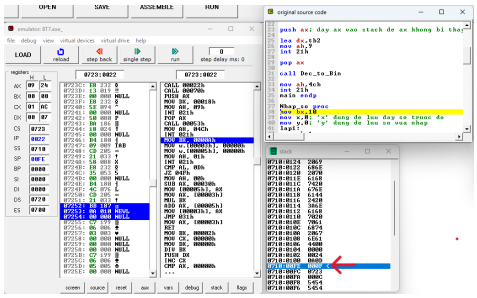
(a) IP khi đến lệnh JMP



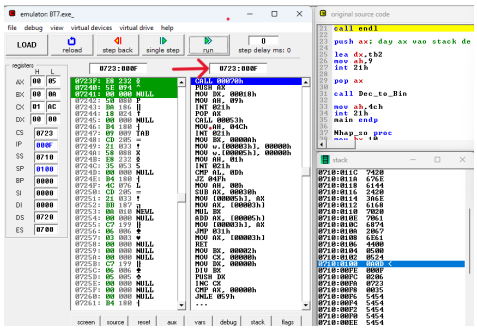
(b) IP sau thực hiện lệnh JMP

Hình 20: Lệnh JMP

∗ Lệnh CALL và RET: Khi IP chạy đến lệnh 'CALL Nhap_so' thì CPU sẽ đẩy địa chỉ của lệnh sau 'CALL Nhap_so' là 'CALL endl' vào Stack và sau khi kết thúc lệnh RET của 'Nhap_so' thì địa chỉ của lệnh 'CALL endl' được lấy ra và IP trỏ tới để chương trình tiếp tục chạy



(a) IP nhảy vào Nhap_so



(b) IP sau khi kết thúc Nhap_so

Hình 21: Lệnh CALL và RET

- DS (Data Segment):
  - Dùng để truy xuất biến trong (.data) như: x, y, tb1, tb2, crlf.
  - Được khởi tạo tại dòng:

```
mov ax,@data
mov ds,ax
```

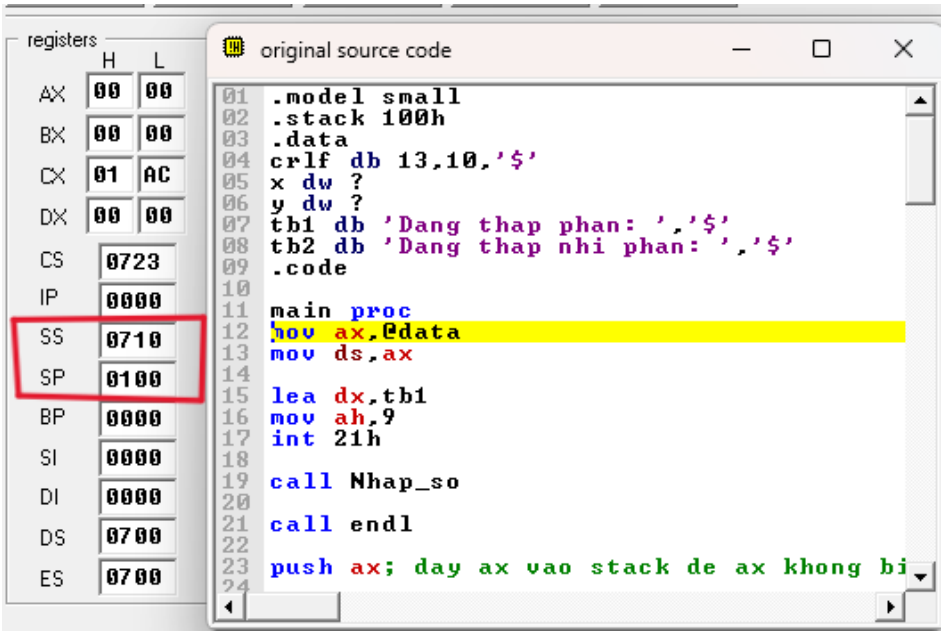
Hình 22: Khởi tạo DS

→ @data trả về địa chỉ segment chứa dữ liệu thông qua thanh ghi AX trunng gian đưa vào thanh ghi DS.

⇒ Quản lí các biến x , y, tb1, tb2, crlf.

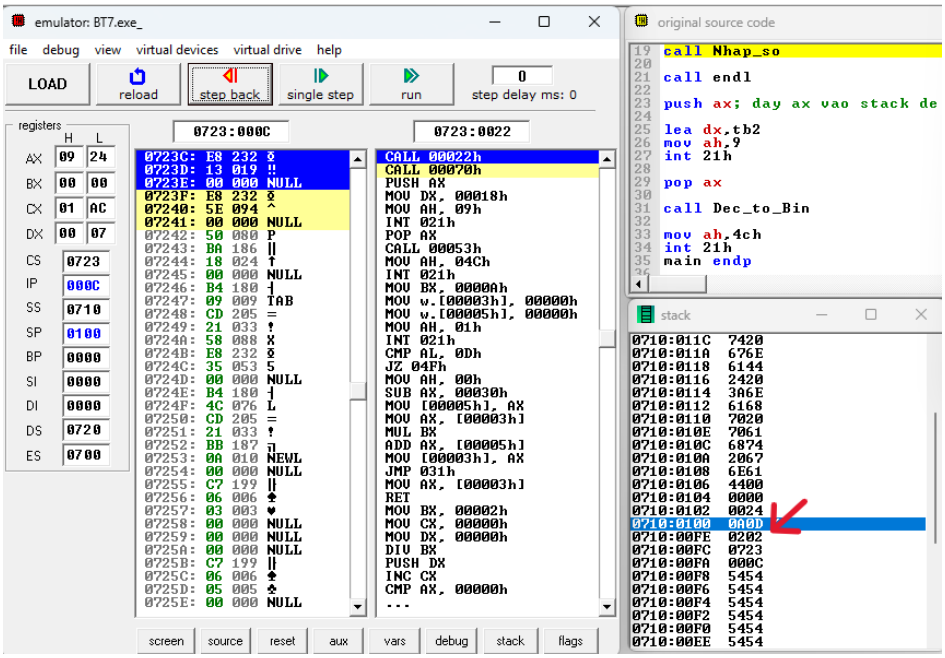
- SS (Stack Segment) và SP (Stack Pointer):
  - SS và SP dùng để truy xuất ngăn xếp.

– Stack được khai báo qua 9 (.stack 100h) → cấp 256 byte cho stack.
– Hệ thống sẽ tự động khởi tạo SS và SP khi chương trình bắt đầu.
  * SS trỏ vào segment stack.
  * SP thường khởi tạo ở đỉnh stack (offset 0100h).



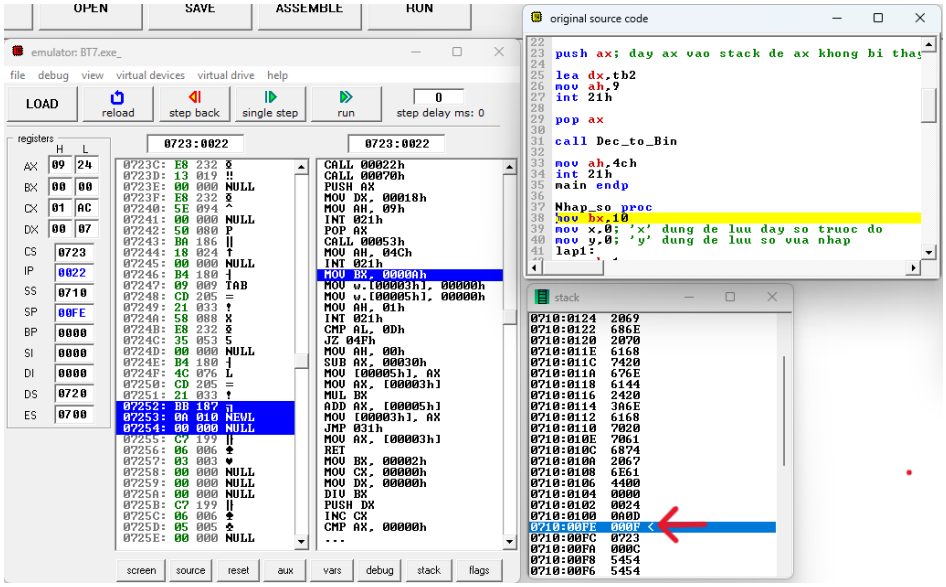
Hình 23: Địa chỉ SS và SP khi chương trình chạy

– Ví dụ khi hàm chạy đến hàm 'Nhap_so'



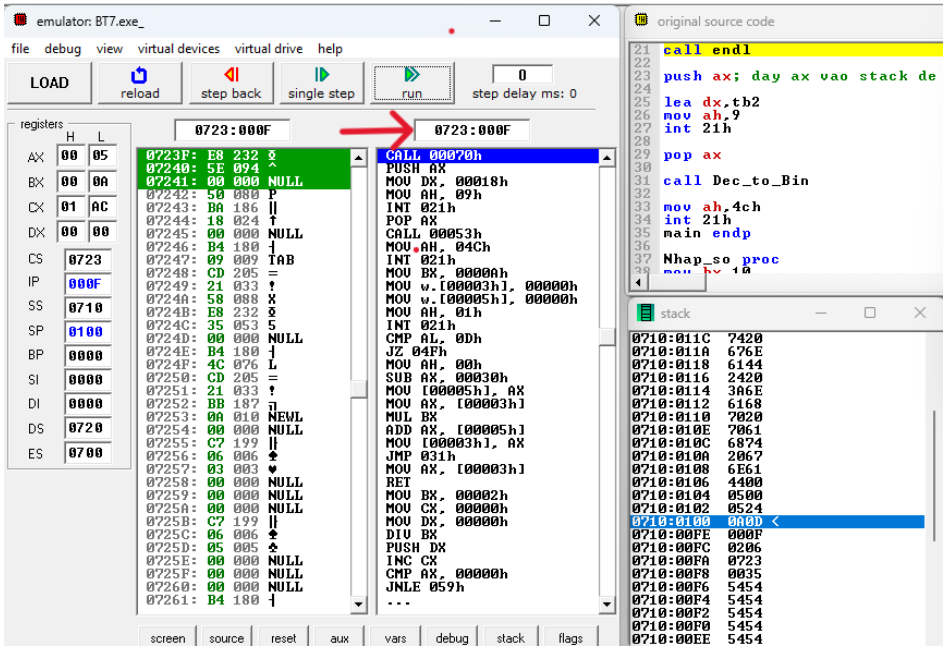
Hình 24: Stack trước hàm Nhap_so

– Địa chỉ 0710:00FE có giá trị là 0202h



Hình 25: Stack trong hàm Nhap_so

– Địa chỉ 0710:00FE đã thay đổi giá trị thành 000F ⇒ CPU đẩy địa chỉ của lệnh kế tiếp (call endl) vào Stack sau khi thực hiện hàm 'Nhap_so'.



Hình 26: Stack sau hàm Nhap_so

– Sau hàm Nhap_so thì giá trị 000F của 0710:00FE được pop ra khỏi stack và CS:IP = 0723:0022 ⇒ IP trỏ tới địa chỉ lệnh 'call endl'

- BP (Base Pointer):

  - BP thường dùng trong thao tác với stack (ví dụ như khi xử lý tham số trong thủ tục).
  - Tuy nhiên không được sử dụng trong bài 'Chuyển số thập phân thành nhị phân', nên không ảnh hưởng đến quản lý bộ nhớ trong trường hợp này.

- ES (Extra Segment):
  - ES là thanh ghi segment bổ sung, thường dùng để thao tác với các chuỗi hoặc vùng nhớ đặc biệt.
  - Trong chương trình này không sử dụng ES, nên không có quản lý bộ nhớ liên quan đến nó.

PHẦN 2: BÀI TẬP NHÓM

GIỚI THIỆU ĐỀ TÀI

- Tic Tac Toe (hay còn gọi là Cờ ca-rô) là một trò chơi giải trí cổ điển, quen thuộc với nhiều thế hệ. Với luật chơi đơn giản nhưng không kém phần hấp dẫn, trò chơi yêu cầu hai người chơi lần lượt đánh dấu X hoặc O vào một bảng gồm 9 ô (3x3), với mục tiêu tạo thành một hàng ngang, hàng dọc hoặc đường chéo gồm ba ký hiệu giống nhau. Mặc dù đơn giản về hình thức, Tic Tac Toe lại đòi hỏi sự tư duy chiến lược và khả năng phản xạ logic để vừa tấn công, vừa phòng thủ trước đối phương.

- Emu8086 là một trình mô phỏng vi xử lý Intel 8086, cho phép người dùng viết và chạy mã hợp ngữ (assembly) trong môi trường mô phỏng. Đây là công cụ hữu ích để tìm hiểu sâu về hoạt động của vi xử lý và cách phần mềm tương tác với phần cứng.

- Với đề tài "Lập trình game Tic Tac Toe trên Emu8086", nhóm chúng em mong muốn:

  1. **Rèn luyện kỹ năng lập trình Assembly:** Việc phát triển một trò chơi trên Emu8086 đòi hỏi hiểu biết sâu về ngôn ngữ lập trình hợp ngữ, quản lý bộ nhớ, xử lý ngắt và giao tiếp với người dùng ở mức độ thấp. Đây là cơ hội để chúng em nâng cao khả năng lập trình hệ thống và hiểu rõ hơn về kiến trúc máy tính.
  2. **Xây dựng logic xử lý và kiểm tra điều kiện:** Trong chế độ hai người chơi, chương trình cần kiểm tra hợp lệ các nước đi, xác định người thắng cuộc và xử lý các tình huống hòa. Điều này giúp rèn luyện kỹ năng tư duy logic và lập trình điều kiện trong môi trường hạn chế.
  3. **Tái hiện một trải nghiệm cổ điển:** Trò chơi Tic Tac Toe tuy đơn giản nhưng mang tính giải trí và cạnh tranh cao. Việc lập trình trò chơi này trên một nền tảng cổ điển như Emu8086 không chỉ mang đến trải nghiệm thú vị, mà còn giúp người chơi cảm nhận được sự hoài cổ trong một môi trường công nghệ hiện đại.

Nhóm thực hiện bao gồm:

- Nguyễn Mạnh Kha - B23DCCN418
- Lăng Viết Thành - B23DCCN768
- Nguyễn Đức Long - B23DCCN502
- Nguyễn Hoài An - B23DCCN002

Chúng em xin phép được trình bày chi tiết về đề tài: **Lập trình game Tic Tac Toe trên Emu8086.**

NỘI DUNG CHÍNH CỦA ĐỀ TÀI

1. **Mục tiêu đề tài**
   Đề tài nhằm phát triển một trò chơi Tic Tac Toe (hay còn gọi là cờ ca-rô) bằng ngôn ngữ lập trình Assembly, chạy trên môi trường giả lập Emu8086. Trò chơi cho phép hai người chơi luân phiên đánh cờ trên cùng một máy hoặc 1 người chơi đánh với máy với giao diện đơn giản, thân thiện.

2. **Giới thiệu về trò chơi**

   - **Tic Tac Toe** là trò chơi chiến lược dành cho 2 người, sử dụng một bảng gồm 9 ô (3 hàng $\times$ 3 cột).
   - Người chơi sẽ lần lượt chọn ký hiệu đại diện của mình: một người dùng **X**, người còn lại dùng **O**.
   - Mục tiêu là sắp xếp được **ba ký hiệu giống nhau** liền kề theo hàng ngang, cột dọc hoặc đường chéo để giành chiến thắng.
   - Nếu toàn bộ các ô đã được điền mà không có người chiến thắng, kết quả được tính là hòa.

   a ***Cách chơi:***

   - **Người chơi 1** bắt đầu trước, chọn ô bất kỳ trên bàn cờ và đánh dấu "**X**".
   - **Người chơi 2** tiếp tục bằng cách đặt dấu "**O**"vào ô trống còn lại.
   - Trò chơi tiếp tục cho đến khi có người thắng hoặc tất cả các ô đã được điền.
   - Nếu 3 ô theo hàng ngang, hàng dọc, đường chéo có cùng một dấu thì người đó sẽ chiến thắng (WIN), nếu đi hết tất cả các ô mà không tìm được người chiến thắng thì cả 2 cùng hòa (DRAW)

   b ***Tính năng chính:***

   - Hiển thị bàn cờ 3x3 với các ô được đánh số từ 1 đến 9.
   - Có **2 chế độ** là **người** và **máy**.
   - Cho phép người chơi nhập số tương ứng để đánh dấu vị trí mình chọn.
   - Tự động cập nhật bảng sau mỗi lượt đi.

- Kiểm tra điều kiện thắng sau mỗi lượt và đưa ra thông báo thắng cuộc nếu có.
- Nếu không ai thắng sau 9 lượt, trò chơi thông báo kết quả hòa.

c ***Đặc điểm nổi bật:***

- Hỗ trợ chơi 2 người trên cùng thiết bị.
- Hỗ trợ chơi với **máy**.
- Thiết kế đơn giản, dễ sử dụng.
- Trò chơi kinh điển mang tính đối kháng nhẹ nhàng, phù hợp với nhiều đối tượng.

3. **Cấu trúc và thuật toán chương trình**

a ***Cấu trúc dữ liệu:***

- Trò chơi sử dụng một mảng một chiều gồm 9 phần tử, mỗi phần tử tương ứng với một ô trên bàn cờ.
- Ban đầu, các phần tử mảng ký tự từ '1' đến '9', đại diện cho số thứ tự các ô.

b ***Thuật toán hoạt động:***

- **Xác định lượt chơi:** Biến đếm CNT được sử dụng để kiểm soát số thứ tự lượt chơi hiện tại. Biến CNT sẽ xác định người chơi hiện tại, với quy ước người chơi '**X**' đi trước (CNT chẵn), '**O**' đi sau (CNT lẻ).
- **Nhập dữ liệu:** Người dùng sẽ nhập số đại diện cho ô họ muốn đánh. Nếu người chơi nhập sai hoặc chọn ô đã đánh, chương trình yêu cầu nhập lại.
- **Cập nhật bảng cờ:** Sau mỗi lượt, chương trình thay đổi phần tử mảng tương ứng thành '**X**' hoặc '**O**'.
- **Kiểm tra thắng cuộc:** Sau mỗi lượt, thuật toán kiểm tra 8 trường hợp thắng (3 hàng, 3 cột, 2 đường chéo).
- **Kết luận trò chơi:** Nếu có người thắng, trò chơi dừng và in ra kết quả. Nếu hết lượt chơi mà không ai thắng, kết quả được xác định là hòa.
- Bổ sung chế độ cho người chơi đấu với máy (AI).
- Người chơi đi trước với ký hiệu "**X**", máy sử dụng "**O**".
  Máy tính dựa vào thuật toán để đưa ra nước đi:
- Nếu có thể thắng ngay, máy sẽ đi để chiến thắng.
- Nếu đối thủ sắp thắng, máy sẽ chặn.
- Nếu không, máy chọn một ô trống ngẫu nhiên (ưu tiên giữa/góc).

c ***Thay đổi trong chương trình:***

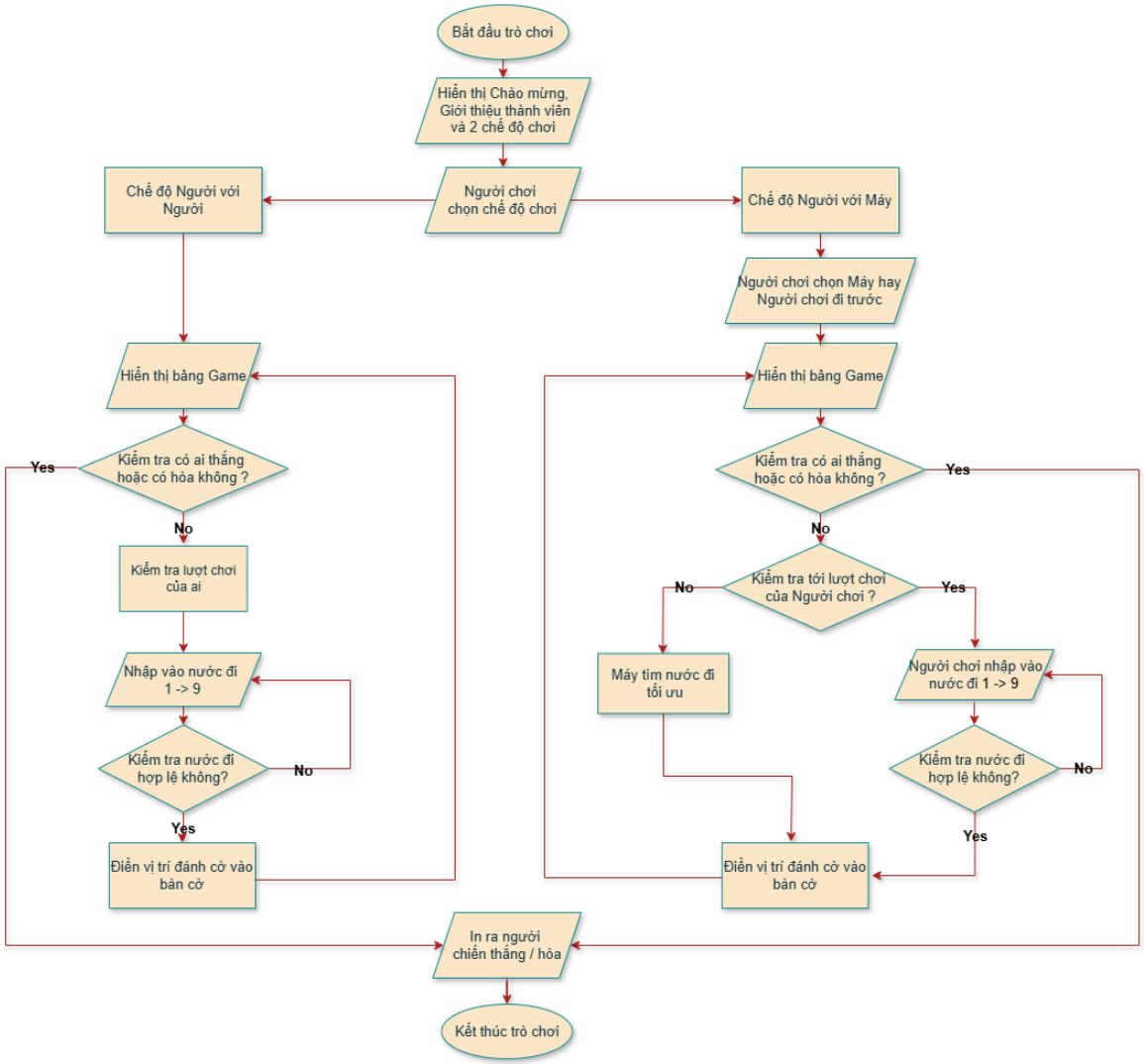
- Thêm lựa chọn chế độ chơi (người với người / người với máy).
- Viết hàm riêng cho lượt đi của máy.
- Cập nhật logic kiểm tra thắng và kết thúc trò chơi.

d ***Lợi ích:***

- Người chơi có thể giải trí một mình.
- Tăng tính tương tác và hấp dẫn cho trò chơi.
- Là bước đầu tiếp cận **tư duy AI đơn giản** trong Assembly.

MIÊU TẢ CHƯƠNG TRÌNH

Lưu đồ thuật toán (Flowcharts) của chương trình:



Phân tích chương trình: Mã nguồn game TIC TAC TOE

```
msg_choose_mode db 'You want to play with...$'
mode1 db '1 - HUMAN$'
mode2 db '2 - COMPUTER$'
msg_answer db 'Press your answer: $'
MODE2_1 db 'Who goes first?$'
MODE2_2 db '1 - YOU $'
MODE2_3 db '2 - COMPUTER$'
MODE2_4 db 'Enter your choice: $'
select_mode db ?
HUMAN_WIN db 'HUMAN WIN!$'
COMPUTER_WIN db 'COMPUTER WIN!$'
time db 'Computer are thinking next step$'
waitting db 'Please wait a second...$'
PLAYER_TURN db 'Player turn, Enter position (1-9): $'
msg_to_continue db 'Press any key to start the game!$'
msg_winner db 'CONGRATULATION $'
msg_lose db 'TRY BETTER NEXT TIME$'
```

- **BOARD**: Đại diện cho bảng Tic Tac Toe 3*times*3 gồm 9 phần tử được đánh số từ 1 đến 9
- **COLOR**: Mảng cho mã màu cho từng ký tự trong "msg_winner"
- **WELCOME**: Lời chào đầu tiên khi chương trình chạy
- **MEM1 đến MEM4**: Tên và mã SInh viên của các thành viên
- **REQ**: Thông điệp yêu cầu người chơi nhập phím để tiếp tục chương trình
- **IN_PLAYER_X / O**: Thông điệp đến lượt X / O, yêu cầu nhập vị trí ô
- **INVALID_MOVE**: Thông điệp nước đi lỗi, yêu cầu nhập lại
- **X/O_WIN_OUT, DRAW_OUT**: Thông báo kết quả trận đấu
- **cnt**: Biến đếm để xác định lượt đi của 'X' và 'O'
- **cot_doc, cot_ngang**: Dùng để vẽ bảng
- **msg_mem**: Thông điệp giới thiệu thành viên

```
.data
    BOARD db '123456789$'
    COLOR db 0Bh,0Ch,0Dh,0Eh,0Fh,0Bh,0Ch,0Dh,0Eh,0Fh,0Bh,0Ch,0Dh,0Eh,0
    WELLCOME db 'Wellcome to Tic Tac Toe By Group 22!', 13, 10, '$'
    MEM1 db 'NGUYEN HOAI AN   B23DCCN002$'
    MEM2 db 'NGUYEN MANH KHA  B23DCCN418$'
    MEM3 db 'NGUYEN DUC LONG  B23DCCN502$'
    MEM4 db 'LANG VIET THANH  B23DCCN768$'
    REQ db 'Press Any Key to Play Game!$'
    IN_PLAYER_X db 'Player X turn, Enter position (1-9): $'
    IN_PLAYER_O db 'Player O turn, Enter position (1-9): $'
    INVALID_MOVE db 'Invalid move! Try again : $'
    X_WIN_OUT db 'Player X wins!$'
    O_WIN_OUT db 'Player O wins!$'
    DRAW_OUT db 'Draw!$'
    cnt dw ?
    cot_doc db '|$'
    cot_ngang db '-------$'
    msg_mem db 'GAME MADE BY:$'
```

- **msg_choose_mode**: Thông điệp yêu cầu chọn chế độ
- **mode1, mode2**: Chế độ chơi (người/ máy)
- **msg_answer**: Thông điệp lựa chọn của người chơi
- **MODE2_1**: Thông điệp hỏi người chơi chọn người/máy đi trước
- **MODE2_2, MODE2_3**: Lựa chọn cho người/máy đi trước
- **MODE2_4**: Lựa chọn của người chơi
- **select_mode**: Lưu lựa chọn chế độ chơi với người/máy
- **HUMAN_WIN, COMPUTER_WIN**: Thông điệp kết thúc ván game khi 'X'/'O' thắng (chế độ chơi với máy)
- **msg_winner**: Thông điệp chúc mừng chiến thắng
- **msg_lose**: Thông điệp khi người chơi thua máy
- **time, waitting**: Thông điệp chờ máy tính toán
- **PLAYER_TURN**: Thông điệp đến lượt người chơi đi (chế độ chơi với máy)
- **msg_to_continue**: Thông điệp yêu cầu người chơi nhập phím bất kỳ để tiếp tục chương trình

MAIN:

```
MAIN PROC
    mov ax, @data
    mov ds, ax

    call INTRO

    mov cnt,0

    call CLEAR_SCREEN
    call MODE

    mov select_mode,al
    cmp al,1
    je GAME_VS_HUMAN
    jmp PLAYER_VS_COMPUTER
```

- **call INTRO**: Gọi hàm in lời chào
- **call CLEAR_SCREEN**: Hàm để dọn màn hình để in nội dung tiếp theo
- **call MODE**: Gọi hàm hiện lựa chọn chế độ chơi với (người/máy)
- **Nhãn GAME_VS_HUMAN:**

```
    GAME_VS_HUMAN:
        mov bx,0
        call CLEAR_SCREEN
        call PRINT_TABLE
        call CHECK_WIN

        cmp al,'X'
        je  X_WIN

        cmp al,'O'
        je O_WIN

        call CHECK_DRAW
        cmp al,1
        je GAME_DRAW

        mov ax,cnt
        mov bh,2
        div bh

        cmp ah,0
        je X_TURN
        jmp O_TURN

        CONTINUE_GAME_VS_HUMAN:
        inc cnt
        jmp GAME_VS_HUMAN
```

  - **call CLEAR_SCREEN:** Dọn màn hình
  - **call PRINT_TABLE:** In ra bảng 3x3

- **je O_WIN:** In thông điệp khi bên dùng dấu 'O' chiến thắng
- **je X_WIN:** In thông điệp khi bên dùng dấu 'X' chiến thắng
- **je GAME_DRAW:** In thông điệp hòa

- **Nhãn PLAYER_VS_COMPUTER:**

```
PLAYER_VS_COMPUTER:
    call CLEAR_SCREEN

    mov dh,6
    mov dl,28
    mov bx,0
    mov ah,2
    int 10h

    mov ah, 9
    lea dx, MODE2_1
    int 21h

    mov dh,9
    mov dl,28
    mov bx,0
    mov ah,2
    int 10h

    lea dx, MODE2_2
    mov ah,9
    int 21h

    mov dh,11
    mov dl,28
    mov bx,0
    mov ah,2
    int 10h
    lea dx, MODE2_3
    mov ah,9
    int 21h

    mov dh,13
    mov dl,28
    mov bx,0
    mov ah,2
    int 10h

    lea dx, MODE2_4
    mov ah,9
    int 21h

    mov ah, 1
    int 21h

    cmp al, '2'
    je PC_FIRST
    jmp GAME_VS_COMPUTER
```

- **CLEAR_SCREEN**: Dọn màn hình

- **mov DH,13 và mov DL, 28:** Vị trí muốn đặt con trỏ (DH vị trí hàng, DL vị trí cột)
- **mov AH, 2 và int 10h**: Thực hiện ngắt để đặt lại vị trí con trỏ
- **lea DX, MODE2_1**: Tải địa chỉ của chuỗi MODE2_1 vào DX
- **mov AH, 9 và int 21h**: Thực hiện lệnh ngắt để in chuỗi có địa chỉ được lưu trong DX

  !Các chuỗi **MODE2_2, MODE2_3, MODE2_4** được thực hiện như **MODE2_1**

- **Nhãn GAME_VS_COMPUTER:**

```
GAME_VS_COMPUTER:
    mov bx,0
    call CLEAR_SCREEN
    call PRINT_TABLE
    call CHECK_WIN

    cmp al,'X'
    je  X_WIN

    cmp al,'O'
    je O_WIN

    call CHECK_DRAW
    cmp al,1
    je GAME_DRAW

    mov ax,cnt
    mov bh,2
    div bh

    cmp ah,0
    je X_TURN
    jmp COMPUTER_TURN

    CONTINUE_GAME_VS_COMPUTER:
    inc cnt
    jmp GAME_VS_COMPUTER
```

  - **call CLEAR_SCREEN:** Dọn màn hình
  - **call PRINT_TABLE:** In ra bảng 3x3
  - **je O_WIN:** In thông điệp khi bên dùng dấu 'O' chiến thắng
  - **je X_WIN:** In thông điệp khi bên dùng dấu 'X' chiến thắng
  - **je GAME_DRAW**: In thông điệp hòa

INTRO: Thủ tục in lời chào

```
lea si,WELLCOME
mov dh,10
mov dl,20

PRINT_WELLCOME:
    mov al,[si]
    cmp al,'$'
    je DONE_PRINT_WELLCOME
    mov ah,2
    int 10h

    mov ah,9
    mov bh,0
    mov bl,15
    mov cx,1
    int 10h

    inc si
    inc dl
    jmp PRINT_WELLCOME

DONE_PRINT_WELLCOME:
call DELAY
call CLEAR_SCREEN
```

- **lea SI, WELLCOME:** Tải địa chỉ ký tự đầu tiên của chuỗi WELLCOME vào SI
- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 34:** Đặt vị trí con trỏ lại về hàng 34.
- **Nhãn PRINT_WELLCOME:**
  - **mov AL, [SI]:** Gán ký tự của chuỗi **WELLCOME** vào **AL**
  - **cmp AL, '$':** So sánh **AL** với mã ascii của '$'
  - **je DONE_PRINT_WELLCOME:** Nhảy đến nhãn **DONE_PRINT_WELLC** khi **AL** bằng '$'
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **mov AH, 9:** Hàm in ký tự có màu
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov BL, 15:** Gán mã màu trắng vào **BL** cho kí tự được lưu ở **AL**
  - **mov CX, 1:** In một lần
  - **int 10h:** Gọi ngắt để in ký tự màu
  - **inc SI:** Tăng con trỏ sang ký tự tiếp theo trên chuỗi **WELLCOME**
  - **inc DL:** Tăng vị trí cột con trỏ

- **jmp PRINT_WELLCOME:** Nhảy đến nhãn **PRINT_WELLCOME**

- **Nhãn DONE_PRINT_WELLCOME:**

```
DONE_PRINT_WELLCOME:
call DELAY
call CLEAR_SCREEN

mov dh,6
mov dl,33
mov bh,0
mov ah,2
int 10h
lea dx,msg_mem
mov ah,9
int 21h
```

  - **call DELAY:** Gọi thủ tục DELAY
  - **call CLEAR_SCREEN:** Gọi thủ tục **CLEAR_SCREEN**
  - **mov DL, 6:** Đặt vị trí con trỏ lại về cột 6
  - **mov DH, 33:** Đặt vị trí con trỏ lại về hàng 33
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **lea DX, msg_mem:** Tải địa chỉ chuỗi vào **DX**
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in chuỗi

```
mov dh,9
mov dl,27
mov bh,0
mov ah,2
int 10h
lea dx,MEM1
mov ah,9
int 21h
```

  - **mov DL, 9:** Đặt vị trí con trỏ lại về cột 9
  - **mov DH, 27:** Đặt vị trí con trỏ lại về hàng 27
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **lea DX, MEM1:** Tải địa chỉ chuỗi vào DX
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in chuỗi

    ! **MEM2, MEM3, MEM4** thực hiện như **MEM1** để in ra tên và mã SInh viên của thành viên

```
mov dh,18
mov dl,27
mov bh,0
mov ah,2
int 10h
lea dx,msg_to_continue
mov ah,9
int 21h

mov ah,1
int 21h


RET
```

- **mov DL, 18:** Đặt vị trí con trỏ lại về cột 18
- **mov DH, 27:** Đặt vị trí con trỏ lại về hàng 27
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, msg_to_continue:** Tải địa chỉ chuỗi vào **DX**
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi
- **mov AH, 1:** Hàm nhập ký tự
- **int 21h:** Gọi ngắt để in ký tự
- **RET:** Trả địa chỉ lệnh cho **IP** để quay lại **MAIN**

MODE: Thủ tục chọn chế độ chơi

```
mov bh,0
mov dh,6
mov dl,28
mov ah,2
int 10h

lea dx,msg_choose_mode
mov ah,9
int 21h
```

```
mov dh,9
mov dl,28
mov ah,2
int 10h

lea dx,mode1
mov ah,9
int 21h

mov dh,11
mov dl,28
mov ah,2
int 10h

lea dx,mode2
mov ah,9
int 21h

mov dh,13
mov dl,28
mov ah,2
int 10h

lea dx,msg_answer
mov ah,9
int 21h

mov ah,1
int 21h
sub al,'0'
```

- **mov BH,0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov DH,6:** Di chuyển con trỏ về hàng 6
- **mov DL,28:** Di chuyển con trỏ về cột 28
- **mov AH,2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX,msg_choose_mode:** Tải địa chỉ chuỗi vào **DX**
- **mov AH,9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi

- **mov DH, 9:** Di chuyển con trỏ về hàng 9, chuẩn bị in dòng đầu tiên của menu chế độ
- **mov DL, 28:** Di chuyển con trỏ về cột 28, đồng bộ vị trí dòng mới với tiêu đề
- **mov AH, 2:** Hàm đặt lại vị trí con trỏ
- **int 10h:** Gọi ngắt BIOS để áp dụng vị trí con trỏ mới
- **lea DX, mode1:** Tải địa chỉ chuỗi "**1. Player vs Player**" vào **DX** – hiển thị chế độ chơi giữa 2 người
- **mov AH, 9:** Chuẩn bị in chuỗi ra màn hình
- **int 21h:** Thực hiện in chế độ 1
- **mov DH, 11:** Di chuyển con trỏ về hàng 11 – in dòng tiếp theo là chế độ 2
- **mov DL, 28:** Di chuyển con trỏ về cột 28, giữ đồng nhất canh lề
- **mov AH, 2:** Hàm định vị con trỏ
- **int 10h:** Đặt lại con trỏ để hiển thị dòng mới
- **lea DX, mode2:** Tải địa chỉ chuỗi "**2. Player vs Computer**" vào **DX**
- **mov AH, 9:** Hàm in chuỗi ra màn hình
- **int 21h:** In chế độ chơi thứ 2 lên màn hình
- **mov DH, 13:** Di chuyển con trỏ xuống hàng 13, nơi người chơi sẽ nhập lựa chọn
- **mov DL, 28:** Đưa con trỏ về cột 28
- **mov AH, 2:** Gọi hàm định vị trí con trỏ
- **int 10h:** Áp dụng vị trí con trỏ
- **lea DX, msg_answer:** Tải địa chỉ chuỗi "**Your choice:**" vào **DX** – hướng dẫn người chơi nhập vào lựa chọn
- **mov AH, 9:** Chuẩn bị in chuỗi
- **int 21h:** Hiển thị chuỗi yêu cầu nhập dữ liệu
- **mov AH, 1:** Chuẩn bị gọi hàm đọc 1 ký tự từ bàn phím mà không cần nhấn Enter
- **int 21h:** Thực hiện đọc ký tự, lưu vào thanh ghi AL
- **sub AL, '0':** Chuyển ký tự ASCII về giá trị số nguyên (ví dụ: nếu người chơi nhập '2', AL chứa mã ASCII 50 $\rightarrow$ 50 - 48 = 2)

PRINT_TABLE: Thủ tục in bảng 3×3

```
mov dh,9
mov dl,35
mov ah,2
int 10h

push dx
lea dx,cot_ngang
mov ah,9
int 21h
pop dx
```

- **mov DH, 9:** Di chuyển con trỏ về hàng 9
- **mov DL, 35:** Di chuyển con trỏ về cột 35
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **push DX:** Đẩy **DX** vào stack để lưu vị trí con trỏ
- **lea DX, cot_ngang:** Tải địa chỉ cột ngang đầu tiên vào **DX**
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in cột ngang đầu tiên
- **pop DX:** Lấy lại vị trí con trỏ
- **Nhãn ROW:**

```
ROW:
   inc dh
   cmp dh,16
   je END_PRINT_TABLE
   mov dl,35
   mov cx,3

   mov ah,2
   int 10h

   push dx
   lea dx,cot_doc
   mov ah,9
   int 21h
   pop dx
   inc dl
   COLUMN:
        push cx
        mov al,[si]
        mov ah,2
        int 10h
        cmp al,'X'
        je PRINT_X
        cmp al,'O'
        je PRINT_O
        jmp DEFAUL
```

  - **inc DH:** Tăng vị trí con trỏ lên 1 hàng

- **cmp DH, 16:** So sánh giá trị **DH** với 16
- **je END_PRINT_TABLE:** Kết thúc in bảng khi **DH** = 16
- **mov DL, 35:** Đặt lại vị trí con trỏ tại cột 15
- **mov CX, 3:** Để thực hiện loop nhãn **COLUMN**
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **push DX:** Đẩy **DX** vào stack để lưu vị trí con trỏ
- **lea DX, cot_doc:** Tải địa chỉ cột dọc đầu tiên trên 1 hàng vào **DX**
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in cột dọc đầu tiên trên 1 hàng
- **pop DX:** Lấy lại vị trí con trỏ
- **inc DL:** Tăng vị trí con trỏ lên 1 cột

- **Nhãn COLUMN:**

```
PRINT_X:
    mov bh,0
    mov bl,13
    mov ah,9
    mov cx,1
    int 10h
    jmp CONTINUE_LOOP

PRINT_O:
    mov bh,0
    mov bl,10
    mov ah,9
    mov cx,1
    int 10h
    jmp CONTINUE_LOOP

DEFAUL:

    mov bh,0
    mov bl,15
    mov ah,9
    mov cx,1
    int 10h
```

- **push CX:** Đẩy **CX** vào stack để lưu số lần loop **COLUMN**
- **mov AL, [SI]:** Gán **AL** bằng kí tự [**SI**] trong mảng **BOARD**
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **cmp AL,'X':** So sánh **AL** với kí tự **X**
- **je PRINT_X: AL** bằng **X** thì nhảy đến nhãn **PRINT_X**
- **cmp AL,'O':** So sánh **AL** với kí tự **O**
- **je PRINT_O: AL** bằng **X** thì nhảy đến nhãn **PRINT_O**

- **jmp DEFAUL:** Nhảy đến nhãn**DEFAUL**

- **Nhãn DEFAUL:**
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov BL, 15:** Gán mã màu trắng vào **BL** cho kí tự được lưu ở **AL**
  - **mov AH, 9:** Hàm in ký tự có màu
  - **mov CX, 1:** In một lần
  - **int 10h:** Gọi ngắt để in ký tự màu

- **Nhãn CONTINUE_LOOP:**

```
        CONTINUE_LOOP:
        inc dl
        mov ah,2
        int 10h
        push dx

        lea dx,cot_doc
        mov ah,9
        int 21h

        pop dx
        inc dl
        inc si
        pop cx
    loop COLUMN

    mov dl,35
    inc dh
    mov ah,2
    int 10h

    push dx

    lea dx,cot_ngang
    mov ah,9
    int 21h

    pop dx


jmp ROW
```

  - **inc DL:** Tăng vị trí con trỏ lên 1 cột
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **push DX:** Đẩy **DX** vào stack để lưu vị trí con trỏ
  - **lea DX, cot_doc:** Tải địa chỉ cột dọc vào **DX**
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in cột dọc

- **pop DX:** Lấy lại vị trí con trỏ
- **inc DL:** Tăng vị trí con trỏ lên 1 cột
- **inc SI:** Tăng con trỏ lên 1 để trỏ đến phần tử tiếp theo trong mảng **BOARD**
- **pop CX:** Lấy lại giá trị **CX** để thực hiện loop **COLUMN**
- **loop COLUMN:** Nếu **CX** khác 0 nhảy lại nhãn **COLUMN**
- **mov DL, 35:** Đặt vị trí con trỏ lại về cột 35 để thực hiện in hàng tiếp theo
- **inc DH:** Tăng vị trí con trỏ lên 1 hàng
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **push DX:** Đẩy **DX** vào stack để lưu vị trí con trỏ
- **lea DX, cot_ngang:** Tải địa chỉ cột ngang vào **DX**
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in cột ngang sau khi in hàng đầu tiên
- **pop DX:** Lấy lại vị trí con trỏ
- **jmp ROW:** Nhảy về nhãn **ROW**

X_TURN: Thủ tục lượt của X

```
mov dh ,5
mov dl ,20
mov bh,0
mov ah,2
int 10h
lea dx,IN_PLAYER_X
mov ah,9
int 21h
jmp check_move_for_X
```

- Chế độ chơi 1 (chơi với người):
  - **mov DH, 5:** Di chuyển con trỏ về hàng 5
  - **mov DL, 20:** Di chuyển con trỏ về cột 20
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **lea DX, IN_PLAYER_X:** Tải địa chỉ chuỗi vào **DX**
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in chuỗi
  - **jmp check_move_for_X:** Nhảy đến phần kiểm tra nước đi cho **X** (tự động nếu là máy tính)

```
HUMAN_TURN:
mov dh ,5
mov dl ,22
mov bh,0
mov ah,2
int 10h
lea dx,PLAYER_TURN
mov ah,9
int 21h
```

- Chế độ chơi 2 (chơi với máy):
  - **mov DH, 5:** Di chuyển con trỏ về hàng 5
  - **mov DL, 22:** Di chuyển con trỏ về cột 22
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **lea DX, PLAYER_TURN:** Tải địa chỉ chuỗi vào **DX**
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in chuỗi
- **Nhãn check_move_for_X:**

```
check_move_for_X:
    lea si,BOARD
    mov ah,1
    int 21h
    sub al,'0'
    mov ah,0
    add si,ax
    dec si
    mov bl,[si]
    cmp bl,'9'
    jle VALID_POS_FOR_X
    call INVALID_POS
    jmp check_move_for_X

VALID_POS_FOR_X:
    mov [si],'X'
    mov bl,select_mode
    cmp bl,1
    je CONTINUE_GAME_VS_HUMAN
    jmp CONTINUE_GAME_VS_COMPUTER
```

  - **check_move_for_X:** kiểm tra nước đi của người chơi **X**
  - **lea SI, BOARD:** Tải địa chỉ bảng cờ vào thanh ghi **SI**
  - **mov AH, 1:** Chuẩn bị gọi hàm nhập 1 ký tự từ bàn phím
  - **int 21h:** Thực hiện đọc ký tự nhập từ người dùng
  - **sub al, '0':** Chuyển mã **ASCII** ký tự nhập về số (VD: '1' → 1)

- **mov AH, 0:** Xóa phần cao của thanh ghi **AX** (giữ giá trị ở **AL**)
- **add SI, ax:** Dịch con trỏ **SI** đến vị trí người dùng chọn trong bảng
- **dec SI:** Giảm đi 1 vì mảng đánh số từ 0
- **mov bl, [SI]:** Lấy giá trị ô cờ người chơi chọn vào **BL**
- **cmp bl, '9':** Kiểm tra xem ô đang chọn có trống không (đã bị đánh dấu chưa)
- **jle VALID_POS_FOR_X:** Nếu ký tự trong ô bé hơn hoặc bằng '9' thì được coi là ô hợp lệ
- **call INVALID_POS:** Gọi hàm xử lý nhập sai vị trí (ô đã có người đánh)
- **jmp check_move_for_X:** Quay lại kiểm tra nước đi để nhập lại
- **VALID_POS_FOR_X:** xử lý khi vị trí được chọn là hợp lệ
- **mov [SI], 'X':** Ghi ký tự '**X**' vào ô người chơi chọn
- **mov bl, select_mode:** Lấy lại chế độ chơi để xác định nhánh tiếp theo
- **cmp bl, 1:** So sánh với chế độ 1 (PvP)
- **je CONTINUE_GAME_VS_HUMAN:** Nếu đang chơi với người thì nhảy đến xử lý lượt tiếp theo cho **O**
- **jmp CONTINUE_GAME_VS_COMPUTER:** Nếu không thì nhảy đến phần chơi tiếp với máy tính

O_TURN: Thủ tục lượt của O

```
O_TURN PROC
    mov dh ,5
    mov dl ,20
    mov bh,0
    mov ah,2
    int 10h
    lea dx,IN_PLAYER_O
    mov ah,9
    int 21h
```

- Chế độ chơi với người:
  - **mov DH, 5:** Di chuyển con trỏ về hàng 5
  - **mov DL, 20:** Di chuyển con trỏ về cột 20
  - **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
  - **mov AH, 2:** Hàm đặt vị trí con trỏ
  - **int 10h:** Gọi ngắt để đặt lại con trỏ
  - **lea DX, IN_PLAYER_O:** Tải địa chỉ chuỗi vào **DX**
  - **mov AH, 9:** Lệnh in chuỗi
  - **int 21h:** Gọi ngắt in chuỗi

- **Nhãn check_move_for_O:**

```
    check_move_for_O:
        lea si,BOARD
        mov ah,1
        int 21h
        sub al,'0'
        mov ah,0
        add si,ax
        dec si
        mov bl,[si]
        cmp bl,'9'
        jle VALID_POS_FOR_O
        call INVALID_POS
        jmp check_move_for_O

    VALID_POS_FOR_O:
        mov [si],'O'
        jmp CONTINUE_GAME_VS_HUMAN
    RET
O_TURN ENDP
```

  - **lea SI, BOARD:** Tải địa chỉ mảng bàn cờ vào thanh ghi **SI**
  - **mov AH, 1:** Chuẩn bị nhận 1 ký tự từ bàn phím
  - **int 21h:** Gọi ngắt để nhập ký tự ô muốn đánh (1 → 9)
  - **sub al, '0':** Chuyển ký tự số từ **ASCII** sang giá trị số (ví dụ: '1' → 1)
  - **mov AH, 0:** Xoá phần cao của thanh ghi **AX** để đảm bảo chính xác
  - **add SI, ax:** Tăng địa chỉ **SI** lên đúng ô người chơi chọn
  - **dec SI:** Vì mảng bắt đầu từ 0 nên giảm **SI** đi 1
  - **mov bl, [SI]:** Lấy giá trị tại ô đó vào **BL** để kiểm tra
  - **cmp bl, '9':** So sánh với ký tự '9' – giả sử các ô trống ban đầu là '1' → '9'
  - **jle VALID_POS_FOR_O:** Nếu giá trị nhỏ hơn hoặc bằng '9' → hợp lệ → chuyển đến **VALID_POS_FOR_O**
  - **call INVALID_POS:** Nếu ô đã bị đánh → gọi hàm báo lỗi
  - **jmp check_move_for_O:** Quay lại nhập lại nước đi mới
  - **VALID_POS_FOR_O:** Nhãn xử lý nếu người chơi **O** chọn ô hợp lệ
  - **mov [SI], 'O':** Ghi ký hiệu '**O**' vào ô đã chọn
  - **jmp CONTINUE_GAME_VS_HUMAN:** Nhảy đến đoạn tiếp theo của trò chơi – luân phiên lượt đánh

INVALID_POS: Thủ tục nước đi lỗi

```
INVALID_POS PROC
    call CLEAR_SCREEN
    call PRINT_TABLE
    mov dh ,5
    mov dl ,28
    mov bh,0
    mov ah,2
    int 10h
    lea dx,INVALID_MOVE
    mov ah,9
    int 21h

    RET
INVALID_POS ENDP
```

- **call CLEAR_SCREEN:** Dọn màn hình hiển thị
- **call PRINT_TABLE:** In lại bảng 3×3
- **mov DH, 5:** Di chuyển con trỏ về hàng 5
- **mov DL, 28:** Di chuyển con trỏ về cột 28
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, INVALID_MOVE:** Tải địa chỉ chuỗi vào **DX**
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi

COMPUTER_TURN: Thủ tục để lựa chọn nước đi tối ưu của Máy.

```
COMPUTER_TURN PROC
    mov bx,0
    mov cx,0

    mov dh,3
    mov dl,28
    mov ah,2
    int 10h

    lea dx,time
    mov ah,9
    int 21h

    mov dh,5
    mov dl,28
    mov ah,2
    int 10h

    lea dx,waitting
    mov ah,9
    int 21h

    call CHANCE_TO_WIN
    call CHANCE_TO_BLOCK
    call CHECK_MIDDLE_POS
    call CHECK_CORNER_POS
    call CHECK_RANDOM_POS

    RET
COMPUTER_TURN ENDP
```

- **mov DH, 3 và mov DL, 28 :** Vị trí muốn đặt con trỏ (Hàng 3, Cột 28).
- **mov AH, 2 và int 10h :** Thực hiện ngắt để đặt lại vị trí con trỏ.
- **lea DX, time:** Tải đỉa chỉ của chuỗi thông điệp "**time**" vào **DX**.
- **mov AH, 9:** Gọi hàm ngắt 9 để in 1 chuỗi.
- **int 21h:** In thông điệp Thông báo Máy "đang nghĩ nước đi ".
- **mov DH, 5 và mov DL, 28 :** Vị trí muốn đặt con trỏ (Hàng 5, Cột 28).
- **mov AH, 2 và int 10h :** Thực hiện ngắt để đặt lại vị trí con trỏ.
- **lea DX, waitting :** Tải đỉa chỉ của chuỗi thông điệp "**waitting**" vào **DX**.
- **mov AH, 9:** Gọi hàm ngắt 9 để in 1 chuỗi.
- **int 21h:** In thông điệp Thông báo " Vui lòng đợi vài giây".
- **call CHANCE_TO_WIN :** Gọi hàm Tìm nước đi (nếu có) để Máy chiến thắng .

- **call CHANCE_TO_BLOCK :** Gọi hàm Tìm nước đi (nếu có) để chặn Người chơi chiến thắng.
- **call CHECK_MIDDLE_POS :** Gọi hàm Tìm nước đi ở ô chính giữa bàn cờ (nếu có) .
- **call CHECK_CORNER_POS :** Gọi hàm Tìm nước đi (nếu có) ở 4 góc của bàn cờ.
- **call CHECK_RANDOM_POS :** Gọi hàm Tìm nước đi (nếu có) ở 4 ô còn lại của bàn cờ.
- **RET :** Hàm trả về, kết thúc hàm **COMPUTER_WIN**

ENTER_MOVE: Thủ tục gán nước đi của Máy vào bàn cờ.

```
ENTER_MOVE PROC
    mov [si], 'O'
    call DELAY
    jmp CONTINUE_GAME_VS_COMPUTER
    RET
ENTER_MOVE ENDP
```

- **mov [SI], 'O' :** Gán '**O**' (Nước đi của Máy) vào vị trí mà SI đang trỏ tới trong bàn cờ.
- **call DELAY :** Gọi hàm DELAY làm chậm chương trình.
- **jmp CONTINUE_GAME_VS_COMPUTER :** Nhảy tới nhãn tiếp tục lượt chơi tiếp theo của chế độ Người chơi với Máy.
- **RET :** Hàm trả về, kết thúc hàm **ENTER_MOVE**

CHANCE_TO_WIN: Thủ tục tìm nước đi để Máy dành chiến thắng.

```
CHANCE_TO_WIN PROC
    mov cx, 3
    lea si, BOARD

    DUYET_HANG_WIN:
        mov dh, [si]
        mov dl, [si + 1]
        mov bl, [si + 2]

        cmp dh, 'X'
        je TIEP_TUC_DUYET_HANG_WIN
        cmp dl, 'X'
        je TIEP_TUC_DUYET_HANG_WIN
        cmp bl, 'X'
        je TIEP_TUC_DUYET_HANG_WIN

        cmp dh, 'O'
        jne skip1
        cmp dl, 'O'
        jne skip1
        add si, 2
        jmp ENTER_MOVE

        skip1:
            cmp dl, 'O'
            jne skip2
            cmp bl, 'O'
            jne skip2
            add si, 0
            jmp ENTER_MOVE

        skip2:
            cmp dh, 'O'
            jne TIEP_TUC_DUYET_HANG_WIN
            cmp bl, 'O'
            jne TIEP_TUC_DUYET_HANG_WIN
            add si, 1
            jmp ENTER_MOVE

        TIEP_TUC_DUYET_HANG_WIN:
            add si, 3
            loop DUYET_HANG_WIN
```

- **mov CX, 3 :** Khởi tạo biến lặp để duyệt 3 hàng ngang trong bàn cờ.
- **lea SI, BOARD :** Nạp địa chỉ ô đầu tiên của bàn cờ vào **SI**.
- **DUYET_HANG_WIN:** Duyệt từng hàng để tìm nước đi
- **mov DH, [SI] :** Lưu kí tự đầu tiên của hàng đang duyệt vào **DH**.
- **mov DL, [SI + 1] :** Lưu kí tự thứ 2 của hàng đang duyệt vào .

- **mov bl, [SI + 2] :** Lưu kí tự thứ 3 của hàng đang duyệt vào **BL**.
- **cmp DH, 'X' / cmp DL, 'X' / cmp bl, 'X' :** So sánh 3 kí tự của hàng với kí tự 'X' (Người chơi)
- **je TIEP_TUC_DUYET_HANG_WIN :** Nếu trong hàng có kí tự 'X' tức là Máy không thể thắng thông qua hàng đó (Bắt buộc có 3 'O') thì nhảy tới nhãn **TIEP_TUC_DUYET_HANG_WIN** để tiếp tục duyệt hàng tiếp theo.
- **cmp DH, 'O' và cmp DL, 'O' :** So sánh kí tự đầu với kí tự 2 của hàng với 'O'
- **jne skip1 :** Nếu 2 kí tự không phải là 'O' tức là chưa thắng hàng đó được thì chuyển tới so sánh kí tự thứ 2 với thứ 3 của hàng.
- **add SI, 2 và jmp ENTER_MOVE :** Nếu 2 kí tự thứ 1 và 2 đều là 'O' thì tìm được nước đi để thắng ở ô thứ 3 của hàng (**add SI, 2**) và nhảy tới hàm **ENTER_MOVE** để cập nhật nước đi trong bàn cờ.

  Tương tự so sánh kí tự đầu với kí tự 2 bên trên (để kiểm tra nước đi ở ô còn lại):

- **skip1 :** Kiểm tra kí tự thứ 2 với thứ 3 liệu có thể chọn nước đi ở ô đầu tiên để thắng?
- **skip2 :** Kiểm tra kí tự thứ 1 với thứ 3 liệu có thể chọn nước đi ở ô thứ 2 để thắng?
- **TIEP_TUC_DUYET_HANG_WIN :** Nhãn thủ tục để duyệt hàng tiếp theo
- **add SI, 3 :** Trỏ tới ô đầu tiên của hàng tiếp theo.
- **loop DUYET_HANG_WIN :** Tiếp tục lặp cho tới khi hết 3 hàng để tìm nước đi có thể chiến thắng ở 1 trong 3 hàng

```
mov cx, 3
lea si, BOARD
DUYET_COT_WIN:
    mov dh, [si]
    mov dl, [si + 3]
    mov bl, [si + 6]

    cmp dh, 'X'
    je TIEP_TUC_DUYET_COT_WIN
    cmp dl, 'X'
    je TIEP_TUC_DUYET_COT_WIN
    cmp bl, 'X'
    je TIEP_TUC_DUYET_COT_WIN

    cmp dh, 'O'
    jne skip3
    cmp dl, 'O'
    jne skip3
    add si, 6
    jmp ENTER_MOVE
```

```
skip3:
    cmp dl, 'O'
    jne skip4
    cmp bl, 'O'
    jne skip4
    add si, 0
    jmp ENTER_MOVE
skip4:
    cmp dh, 'O'
    jne TIEP_TUC_DUYET_COT_WIN
    cmp bl, 'O'
    jne TIEP_TUC_DUYET_COT_WIN
    add si, 3
    jmp ENTER_MOVE

TIEP_TUC_DUYET_COT_WIN:
    inc si
    loop DUYET_COT_WIN
```

- **DUYET_COT_WIN** (Tương tự với **DUYET_HANG_WIN**) : Duyệt lần lượt 3 cột của bàn cờ để tìm nước đi có thể chiến thắng.
  - Mỗi lần duyệt 1 cột thì sẽ kiểm tra có tồn 2 kí tự đều là 'O' và kí tự còn lại khác với 'X' (Người chơi):
  - Nếu có thì tìm được thì nhảy đến **ENTER_MOVE** để cập nhật nước đi tại ô khác 'O' trong cột -> WIN
  - Nếu không thì chuyển tới kiểm tra Đường chéo chính và Đường chéo phụ.

```
lea si, BOARD
mov dh, BOARD[0]
mov dl, BOARD[4]
mov bl, BOARD[8]

cmp dh, 'X'
je TIEP_TUC_DCP_WIN
cmp dl, 'X'
je TIEP_TUC_DCP_WIN
cmp bl, 'X'
je TIEP_TUC_DCP_WIN


cmp dh, 'O'
jne skip5
cmp dl, 'O'
jne skip5
add si, 8
jmp ENTER_MOVE

skip5:
    cmp dl, 'O'
    jne skip6
    cmp bl, 'O'
    jne skip6
    add si, 0
    jmp ENTER_MOVE

skip6:
    cmp dh, 'O'
    jne TIEP_TUC_DCP_WIN
    cmp bl, 'O'
    jne TIEP_TUC_DCP_WIN
    add si, 4
    jmp ENTER_MOVE
```

- Đoạn code trên có ý nghĩa : Kiểm tra liệu có thể tìm nước đi trong Đường chéo chính (DCC) để chiến thắng hay không ?
- **mov DH, BOARD[0] :** Lưu kí tự đầu tiên của DCC vào DH.
- **mov DL, BOARD[4] :** Lưu kí tự thứ 2 của DCC vào DL.
- **mov bl, BOARD[8] :** Lưu kí tự thứ 3 của DCC vào BL.
- **cmp DH, 'X' / cmp DL, 'X' / cmp bl, 'X' :** So sánh 3 kí tự của hàng với kí tự 'X' (Người chơi) .
- **je TIEP_TUC_DCP_WIN :** Nếu trong hàng có kí tự 'X' tức là Máy không thể thắng thông qua DCC thì nhảy tới nhãn TIEP_TUC_DCP_WIN để tiếp tục duyệt Đường chéo phụ.
- **Ngược lại :** Lần lượt kiểm tra có tồn tại kí tự đầu tiên cùng kí tự thứ 2 / kí tự thứ 2 cùng kí tự thứ 3 (skip5) / kí tự đầu tiên cùng kí tự thứ 3 (skip6) trong DCC đều là 'O' .

- **jmp ENTER_MOVE :** Nếu tìm thấy thì nhảy tới nhãn ENTER_MOVE để cập nhật ô Máy có thể đi để chiến thắng :
  - **add SI, 8 :** Nước đi ở ô thứ 3 của DCC
  - **add SI, 0 :** Nước đi ở ô đầu tiên của DCC
  - **add SI, 4 :** Nước đi ở ô thứ 2 của DCC
- **jne TIEP_TUC_DCP_WIN :** Nếu không tìm được nước đi trong DCC sau khi duyệt 3 trường hợp thì chuyển sang Tìm nước đi ở Đường chéo phụ.

```
    TIEP_TUC_DCP_WIN:
        mov dh, BOARD[2]
        mov dl, BOARD[4]
        mov bl, BOARD[6]

        cmp dh, 'X'
        je XONG_8_DUONG_WIN
        cmp dl, 'X'
        je XONG_8_DUONG_WIN
        cmp bl, 'X'
        je XONG_8_DUONG_WIN

        cmp dh, 'O'
        jne skip7
        cmp dl, 'O'
        jne skip7
        add si, 6
        jmp ENTER_MOVE

        skip7:
            cmp dl, 'O'
            jne skip8
            cmp bl, 'O'
            jne skip8
            add si, 2
            jmp ENTER_MOVE

        skip8:
            cmp dh, 'O'
            jne XONG_8_DUONG_WIN
            cmp bl, 'O'
            jne XONG_8_DUONG_WIN
            add si, 4
            jmp ENTER_MOVE

    XONG_8_DUONG_WIN:
    RET
CHANCE_TO_WIN ENDP
```

- **TIEP_TUC_DCP_WIN :** Tương tự đoạn code bên trên, Nhãn này có ý nghĩa : Kiểm tra liệu có thể tìm nước đi trong Đường chéo phụ (DCP) để chiến

thắng hay không ?

- Nếu tìm được 2 kí tự bất kì trong DCP đều là 'O' (Máy) thì Máy cần đi ngay ở ô còn lại để chiến thắng :
  - **add SI, 6 :** Chọn nước đi ở ô thứ 3 của DCC.
  - **add SI, 2 :** Chọn nước đi ở ô đầu tiên của DCC.
  - **add SI, 4 :** Chọn nước đi ở ô thứ 2 của DCC.
- **XONG_8_DUONG_WIN :** Sau khi không tồn tại kí tự đầu tiên cùng kí tự thứ 2 / kí tự thứ 2 cùng kí tự thứ 3 (skip7) / kí tự đầu tiên cùng kí tự thứ 3 (skip8) trong DCP đều là 'O' thì tức là Máy KHÔNG THỂ CHIẾN THẮNG TRONG LƯỢT CHƠI NÀY sau khi xét 8 đường có thể chiến thắng được (3 Hàng, 3 Cột, 2 Dường chéo).
- **RET :** Trả về, kết thúc hàm CHANCE_TO_WIN.

CHANCE_TO_BLOCK: Thủ tục tìm nước đi để chặn Người chơi chiến thắng.

```
CHANCE_TO_BLOCK PROC


    mov cx, 3
    lea si, BOARD
    DUYET_HANG_BLOCK:
        mov dh, [si]
        mov dl, [si + 1]
        mov bl, [si + 2]

        cmp dh, 'O'
        je TIEP_TUC_DUYET_HANG_BLOCK
        cmp dl, 'O'
        je TIEP_TUC_DUYET_HANG_BLOCK
        cmp bl, 'O'
        je TIEP_TUC_DUYET_HANG_BLOCK

        cmp dh, 'X'
        jne skip9
        cmp dl, 'X'
        jne skip9
        add si, 2
        jmp ENTER_MOVE
```

```
skip9:
    cmp dl, 'X'
    jne skip10
    cmp bl, 'X'
    jne skip10
    add si, 0
    jmp ENTER_MOVE

skip10:
    cmp dh, 'X'
    jne TIEP_TUC_DUYET_HANG_BLOCK
    cmp bl, 'X'
    jne TIEP_TUC_DUYET_HANG_BLOCK
    add si, 1
    jmp ENTER_MOVE

TIEP_TUC_DUYET_HANG_BLOCK:
    add si, 3
    loop DUYET_HANG_BLOCK
```

- **mov CX, 3 :** Khởi tạo biến lặp để duyệt 3 hàng ngang trong bàn cờ.
- **lea SI, BOARD :** Nạp địa chỉ ô đầu tiên của bàn cờ vào SI.
- **DUYET_HANG_BLOCK:** Duyệt từng hàng để tìm nước đi.
- **mov DH, [SI] :** Lưu kí tự đầu tiên của hàng đang duyệt vào DH.
- **mov DL, [SI + 1] :** Lưu kí tự thứ 2 của hàng đang duyệt vào DL.
- **mov bl, [SI + 2] :** Lưu kí tự thứ 3 của hàng đang duyệt vào BL.
- **cmp DH, 'O' / cmp DL, 'O' / cmp bl, 'O' :** So sánh 3 kí tự của hàng với kí tự 'O' (Máy)
- **je TIEP_TUC_DUYET_HANG_BLOCK :** Nếu trong hàng có kí tự 'X' tức là Người chơi không thể thắng thông qua hàng đó (Bắt buộc có 3 'X') thì nhảy tới nhãn TIEP_TUC_DUYET_HANG_BLOCK để tiếp tục duyệt hàng tiếp theo.
- **cmp DH, 'X' và cmp DL, 'X' :** So sánh kí tự đầu với kí tự 2 của hàng với 'X'.
- **jne skip9 :** Nếu 2 kí tự không phải là 'X' tức là Người chơi chưa thắng hàng đó được (Không cần chặn hàng đó) thì chuyển tới so sánh kí tự thứ 2 với thứ 3 của hàng.
- **add SI, 2 và jmp ENTER_MOVE :** Nếu 2 kí tự thứ 1 và 2 đều là 'X' thì tìm được nước đi để chặn Người chơi thắng ở ô thứ 3 của hàng (add SI, 2) và nhảy tới hàm ENTER_MOVE để cập nhật nước đi trong bàn cờ.
- Tương tự so sánh kí tự đầu với kí tự 2 bên trên (để kiểm tra nước đi ở ô còn lại):

- **skip1 :** Kiểm tra kí tự thứ 2 với thứ 3 liệu có thể chọn nước đi ở ô đầu tiên để chặn Người chơi thắng?
- **skip2 :** Kiểm tra kí tự thứ 1 với thứ 3 liệu có thể chọn nước đi ở ô thứ 2 để chặn Người chơi thắng?

- **TIEP_TUC_DUYET_HANG_BLOCK :** Nhãn thủ tục để duyệt hàng tiếp theo
- **add SI, 3 :** Trỏ tới ô đầu tiên của hàng tiếp theo.
- **loop DUYET_HANG_BLOCK :** Tiếp tục lặp cho tới khi hết 3 hàng để tìm nước đi có thể chặn Người chơi chiến thắng ở 1 trong 3 hàng

```
mov cx, 3
lea si, BOARD
DUYET_COT_BLOCK:
    mov dh, [si]
    mov dl, [si + 3]
    mov bl, [si + 6]

    cmp dh, 'O'
    je TIEP_TUC_DUYET_COT_BLOCK
    cmp dl, 'O'
    je TIEP_TUC_DUYET_COT_BLOCK
    cmp bl, 'O'
    je TIEP_TUC_DUYET_COT_BLOCK

    cmp dh, 'X'
    jne skip11
    cmp dl, 'X'
    jne skip11
    add si, 6
    jmp ENTER_MOVE
```

```
    skip11:
        cmp dl, 'X'
        jne skip12
        cmp bl, 'X'
        jne skip12
        add si, 0
        jmp ENTER_MOVE
    skip12:
        cmp dh, 'X'
        jne TIEP_TUC_DUYET_COT_BLOCK
        cmp bl, 'X'
        jne TIEP_TUC_DUYET_COT_BLOCK
        add si, 3
        jmp ENTER_MOVE

    TIEP_TUC_DUYET_COT_BLOCK:
        inc si
        loop DUYET_COT_BLOCK
```

- **DUYET_COT_BLOCK** (Tương tự với **DUYET_HANG_BLOCK**) : Duyệt lần lượt 3 cột của bàn cờ để tìm nước đi có thể chặn Người chơi chiến thắng.

- Mỗi lần duyệt 1 cột thì sẽ kiểm tra có tồn 2 kí tự đều là 'X' và kí tự còn lại khác với 'O' (Máy):
  - Nếu có thì tìm được thì nhảy đến ENTER_MOVE để cập nhật nước đi tại ô khác 'X' trong cột -> Chặn người chơi chiến thắng thành công.
  - Nếu không thì chuyển tới kiểm tra Đường chéo chính và Đường chéo phụ.

```
lea si, BOARD
mov dh, BOARD[0]
mov dl, BOARD[4]
mov bl, BOARD[8]

cmp dh, 'O'
je TIEP_TUC_DCP_BLOCK
cmp dl, 'O'
je TIEP_TUC_DCP_BLOCK
cmp bl, 'O'
je TIEP_TUC_DCP_BLOCK


cmp dh, 'X'
jne skip13
cmp dl, 'X'
jne skip13
add si, 8
jmp ENTER_MOVE
```

```
skip13:
    cmp dl, 'X'
    jne skip14
    cmp bl, 'X'
    jne skip14
    add si, 0
    jmp ENTER_MOVE

skip14:
    cmp dh, 'X'
    jne TIEP_TUC_DCP_BLOCK
    cmp bl, 'X'
    jne TIEP_TUC_DCP_BLOCK
    add si, 4
    jmp ENTER_MOVE
```

- Đoạn code trên có ý nghĩa : Kiểm tra liệu có thể tìm nước đi trong Đường chéo chính (DCC) để chặn Người chơi chiến thắng hay không ?
- **mov DH, BOARD[0] :** Lưu kí tự đầu tiên của DCC vào DH.
- **mov DL, BOARD[4] :** Lưu kí tự thứ 2 của DCC vào DL.
- **mov bl, BOARD[8] :** Lưu kí tự thứ 3 của DCC vào BL.
- **cmp DH, 'O' / cmp DL, 'O' / cmp bl, 'O' :** So sánh 3 kí tự của hàng với kí tự 'O' (Máy) .

- **je TIEP_TUC_DCP_BLOCK :** Nếu trong hàng có kí tự 'O' tức là Người chơi không thể thắng thông qua DCC thì nhảy tới nhãn **TIEP_TUC_DCP_BLOCK** để tiếp tục duyệt Đường chéo phụ.
- Lần lượt kiểm tra có tồn tại kí tự đầu tiên cùng kí tự thứ 2 / kí tự thứ 2 cùng kí tự thứ 3 (**skip13**) / kí tự đầu tiên cùng kí tự thứ 3 (skip14) trong DCC đều là 'X' .
- **jmp ENTER_MOVE :** Nếu tìm thấy thì nhảy tới nhãn **ENTER_MOVE** để cập nhật ô Máy có thể đi để chặn Người chơi chiến thắng :
  - **add SI, 8 :** Nước đi ở ô thứ 3 của DCC
  - **add SI, 0 :** Nước đi ở ô đầu tiên của DCC
  - **add SI, 4 :** Nước đi ở ô thứ 2 của DCC
- **jne TIEP_TUC_DCP_BLOCK :** Nếu không tìm được nước đi trong DCC sau khi duyệt 3 trường hợp thì chuyển sang Tìm nước đi ở Đường chéo phụ.

```
    TIEP_TUC_DCP_BLOCK:
        mov dh, BOARD[2]
        mov dl, BOARD[4]
        mov bl, BOARD[6]

        cmp dh, 'O'
        je XONG_8_DUONG_BLOCK
        cmp dl, 'O'
        je XONG_8_DUONG_BLOCK
        cmp bl, 'O'
        je XONG_8_DUONG_BLOCK

        cmp dh, 'X'
        jne skip15
        cmp dl, 'X'
        jne skip15
        add si, 6
        jmp ENTER_MOVE
        skip15:
            cmp dl, 'X'
            jne skip16
            cmp bl, 'X'
            jne skip16
            add si, 2
            jmp ENTER_MOVE

        skip16:
            cmp dh, 'X'
            jne XONG_8_DUONG_BLOCK
            cmp bl, 'X'
            jne XONG_8_DUONG_BLOCK
            add si, 4
            jmp ENTER_MOVE

    XONG_8_DUONG_BLOCK:
    RET
CHANCE_TO_BLOCK ENDP
```

- **TIEP_TUC_DCP_BLOCK :** Tương tự đoạn code **TIEP_TUC_DCC_BLOCK** bên trên, Nhãn này có ý nghĩa : Kiểm tra liệu có thể tìm nước đi trong Đường chéo phụ (DCP) để chặn Người chơi chiến thắng hay không ?
- Nếu tìm được 2 kí tự bất kì trong DCP đều là 'X' (Người chơi) thì cần chặn ngay ở ô còn lại :
  - **add SI, 6 :** Chặn ở ô thứ 3 của DCP.
  - **add SI, 2 :** Chặn ơ ô đầu tiên của DCP.
  - **add SI, 4 :** Chặn ở ô thứ 2 của DCP.
- **XONG_8_DUONG_BLOCK :** Sau khi không tồn tại kí tự đầu tiên cùng kí tự thứ 2 / kí tự thứ 2 cùng kí tự thứ 3 (skip15) / kí tự đầu tiên cùng kí tự thứ 3 (skip16) trong DCP đều là 'X' thì tức là Máy KHÔNG THỂ CHẶN NGƯỜI CHƠI CHIẾN THẮNG TRONG LƯỢT CHƠI NÀY sau khi xét 8 đường có thể chặn (3 Hàng, 3 Cột, 2 Đường chéo).
- **RET :** Trả về, kết thúc hàm **CHANCE_TO_BLOCK**.

CHECK_MIDDLE_POS: Thủ tục kiểm tra Máy có thể chọn nước đi ở ô Chính giữa của bàn cờ không ?

```
CHECK_MIDDLE_POS PROC
    lea si, BOARD
    cmp BOARD[4], 'X'
    je CANT_INSERT_MIDDLE
    cmp BOARD[4], 'O'
    je CANT_INSERT_MIDDLE
    add si, 4
    jmp ENTER_MOVE

    CANT_INSERT_MIDDLE:

    RET
CHECK_MIDDLE_POS ENDP
```

- **lea SI, BOARD :** Nạp địa chỉ ô đầu tiên của bàn cờ (ô [0][0]) vào SI.
- **cmp BOARD[4], 'X' :** So sánh kí tự ở chính giữa bàn cờ với 'X'
- **cmp BOARD[4], 'O' :** So sánh kí tự ở chính giữa bàn cờ với 'O'
- **je CANT_INSERT_MIDDLE :** Nhảy tới nhãn **CANT_INSERT_MIDDLE** nếu kí tự chính giữa là 'X' hoặc 'O' (Không thể chọn nước đi ở ô chính giữa).
- Nếu chọn được nước đi chính giữa thì :
  - **add SI, 4 :** Trỏ SI tới ô chính giữa của bàn cờ
  - **jmp ENTER_MOVE :** Nhảy tới hàm **ENTER_MOVE** để cập nhật nước đi của Máy ở ô SI đang trỏ.

- **RET:** Trả về, kết thúc hàm CHECK_MIDDLE_POS.

CHECK_CORNER_POS: Thủ tục kiểm tra Máy có thể tìm nước đi ở 1 trong 4 ô góc của bàn cờ không ?

```
CHECK_CORNER_POS PROC
    lea si, BOARD
    GOC0:

        cmp BOARD[0], 'X'
        je GOC2
        cmp BOARD[0], 'O'
        je GOC2

        jmp ENTER_MOVE

        GOC2:
            cmp BOARD[2], 'X'
            je GOC6
            cmp BOARD[2], 'O'
            je GOC6
            add si, 2
            jmp ENTER_MOVE

        GOC6:
            cmp BOARD[6], 'X'
            je GOC8
            cmp BOARD[6], 'O'
            je GOC8
            add si, 6
            jmp ENTER_MOVE

        GOC8:
            cmp BOARD[8], 'X'
            je NOT_FOUNDED_CORNER_POS
            cmp BOARD[8], 'O'
            je NOT_FOUNDED_CORNER_POS
            add si, 8
            jmp ENTER_MOVE

    NOT_FOUNDED_CORNER_POS:
    RET
CHECK_CORNER_POS ENDP
```

- **lea SI, BOARD :** Nạp địa chỉ ô đầu tiên của bàn cờ vào SI.
- **GOC0 :** Nhãn tìm kiếm nước đi ở ô [0][0] (Góc trên bên trái)
- **cmp BOARD[0], 'X' :** So sánh kí tự ở góc [0][0] với 'X'.
- **cmp BOARD[0], 'O' :** So sánh kí tự ở góc [0][0] với 'O'.
- **je GOC2 :** Nếu là 'X' hoặc 'O' thì không chọn nước đi góc [0][0] được -> nhảy tới GOC2

- Nếu tìm được thì chọn được nước đi ở ô SI đang trỏ (góc [0][0]).
- Tương tự tìm nước đi ở ô [0][0] (GOC0) ta có :
  - **GOC2 :** Tìm ở ô [0][2] nếu tìm thấy thì **add SI, 2 :** Chọn ô [0][2]
  - **GOC6 :** Tìm ở ô [2][0] nếu tìm thấy thì **add SI, 6 :** Chọn ô [2][0]
  - **GOC8 :** Tìm ở ô [2][2] nếu tìm thấy thì **add SI, 8 :** Chọn ô [2][0]
- **jmp ENTER_MOVE :** Nếu tìm được nước đi ở 1 trong 4 ô góc thì nhảy tới ENTER_MOVE để cập nhật nước đi của Máy.
- **je NOT_FOUND_CORNER_POS :** Không tìm được nước đi ở 4 ô góc thì nhảy tới NOT_FOUND_CORNER_POS : Kết thúc tìm nước đi ở 4 ô góc.
- **RET :** Trả về, kết thúc hàm **CHECK_CORNER_POS**

CHECK_RANDOM_POS : Thủ tục tìm Máy nước đi ở 4 ô còn lại trong bàn cờ.

```
CHECK_WIN PROC
    mov cx,3
    lea si,BOARD
    check_row:
        mov bl,[si]
        cmp bl,[si+1]
        jne next_row
        cmp bl,[si+2]
        jne next_row
        jmp WIN
        next_row:
            add si,3
            loop check_row
    mov cx,3
    lea si,BOARD
    check_column:
        mov bl,[si]
        cmp bl,[si+3]
        jne next_column
        cmp bl,[si+6]
        jne next_column
        jmp WIN
        next_column:
            add si,1
            loop check_column
```

- **lea SI, BOARD :** Nạp địa chỉ ô đầu tiên của bàn cờ vào SI.
- **O1 :** Nhãn tìm kiếm nước đi ở ô [0][1]
- **cmp BOARD[1], 'X' :** So sánh kí tự ở [0][1] với 'X'.
- **cmp BOARD[1], 'O' :** So sánh kí tự ở [0][1] với 'O'.
- **je O3 :** Nếu là 'X' hoặc 'O' thì không chọn nước đi ô [0][1] được -> nhảy tới O3

- **add SI, 1 :** Nếu tìm được thì chọn được nước đi ở ô [0][1].
- Tương tự tìm nước đi ở ô [0][1] (O1) ta có :
- **O3 :** Tìm ở ô [1][0] nếu tìm thấy thì **add SI, 3** : Chọn ô [1][0]
- **O5 :** Tìm ở ô [1][2] nếu tìm thấy thì **add SI, 5** : Chọn ô [1][2]
- **O7 :** Tìm ở ô [2][1] nếu tìm thấy thì **add SI, 7** : Chọn ô [2][1]
- **jmp ENTER_MOVE :** Nếu tìm được nước đi ở 1 trong 4 ô góc thì nhảy tới ENTER_MOVE để cập nhật nước đi của Máy.
- Do chắc chắn tìm được nước đi cho máy ở 1 lượt chơi (Nếu chưa ai thắng / hòa) nên nếu xét hết 8 ô và cuối cùng chỉ còn O7 (ô [2][1]) thì ô [2][1] chính là nước đi mà máy chọn được ở lượt chơi này.
- **RET :** Trả về, kết thúc hàm **CHECK_RANDOM_POS**

Check_WIN: Thủ tục check ai sẽ là người chiến thắng

- **Mục Tiêu:** duyệt hàng, cột, 2 đường chéo nếu cả 3 trùng ký tự 'X' hoặc 'O' thì sẽ WIN
- **Kiểm tra hàng và cột:** Xác định xem có hàng ngang hoặc cột nào có 3 ô giống nhau không, tức là người chơi đã thắng theo hàng hoặc cột.
- **mov cx, 3:** Hàm gán giá trị cho thanh ghi cx là 3
- **lea si,BOARD:** Nạp địa chỉ ô đầu tiên của bàn cờ vào si
- **check_row:** Vòng lặp kiểm tra mỗi hàng có ký tự giống nhau hay không
  - **mov bl,[si]:** gán giá trị thanh gi bl bằng [si] địa chỉ đầu tiên của bàn cờ
  - **cmp bl, [si+1]:** so sánh ký tự ô [0][0] với ô [0][1]
  - **jne next_row:** nếu không bằng nhảy đến nhãn next_row
  - **cmp bl,[si+2]:** nếu không nhảy vào **next_row** so sánh tiếp ô [0][0] với ô [0][2]
  - nếu bằng thì nhảy đến lệnh **jmp WIN**
  - **next_row:** nếu ô [0][1] hoặc ô [0][2] không bằng ô đang xét thì
  - **add si,3:** di chuyển đến hàng thứ 2
  - **loop check_row:** giảm cx đi 1 nhảy đến **check_row** check lại lặp lại tương tự đến hết hàng thứ 3 nếu không nhảy vào **WIN**

- **check_column:** Tương tự như check row nhưng cần trỏ si đến vị trí kế tiếp của cột là 3 và 6 và vì kiểm tra cột nên vào lệnh next_column chỉ cần trỏ si lên 1(đến cột kế tiếp)

```
    lea si,BOARD
    check_cheo1:
        mov bl,[si]
        cmp bl,[si+4]
        jne check_cheo2
        cmp bl,[si+8]
        jne check_cheo2
        jmp WIN
    check_cheo2:
        lea si,BOARD
        mov bl,[si+2]
        cmp bl,[si+4]
        jne NO_WIN
        cmp bl,[si+6]
        jne NO_WIN
        jmp WIN
    WIN:
        mov al,bl
        RET
    NO_WIN:
        mov al,0
    RET
CHECK_WIN ENDP
```

- Kiểm tra đường chéo phụ và đường chéo chính: Xác định xem có đường chéo nào có 3 ô giống nhau không, tức là người chơi đã thắng.
- **lea si,BOARD:** Nạp địa chỉ ô đầu tiên của bàn cờ vào si
- **check_cheo1:** Vòng lặp kiểm tra đường chéo chính có cả 3 ký tự giống nhau hay không
- **mov bl,[si]:** gán giá trị thanh gi bl bằng [si] địa chỉ đầu tiên của bàn cờ
- **cmp bl, [si+4]:** so sánh ký tự ô [0][0] với ô [1][2] chéo phải xuống
- **jne check_cheo2:** nếu không bằng nhảy đến nhãn kiểm tra đường chéo phụ
- **cmp bl,[si+8]:** nếu không nhảy vào **check_cheo2** so sánh tiếp ô [0][0] với ô [2][2]
- nếu không bằng nhảy đến nhãn **NO_WIN**
- nếu bằng thì nhảy đến nhãn **WIN**
- **check_cheo2:** Tương tự như check_cheo1 nhưng cần trỏ si đến vị trí kế tiếp của ô là 4 và 6 và vì kiểm tra đường chéo phụ nên vị trí đầu tiên cần xét là 2
- **CHECK_WIN_ENDP:** kết thúc hàm **CHECK_WIN**

X_WIN: Thủ tục khi bên dấu X thắng

```
X_WIN PROC
    call MSG_FOR_WINNER

    mov al,select_mode
    cmp al,2
    je msg_for_human

    mov dh ,5
    mov dl ,31
    mov ah,2
    int 10h
    lea dx,X_WIN_OUT
    mov ah,9
    int 21h
    jmp GAME_END

    msg_for_human:
    mov dh ,5
    mov dl ,34
    mov ah,2
    int 10h

    lea dx,HUMAN_WIN
    mov ah,9
    int 21h
    jmp GAME_END
    RET
X_WIN ENDP
```

- **call MSG_FOR_WINNER:** Gọi thủ tục MSG_FOR_WINNER
- **mov AL, select_mode:** Gán giá trị biến select_mode cho AL
- **cmp AL, 2:** So sánh giá trị AL với 2
- **je msg_for_human:** Nhảy đến nhãn msg_for_human để in thông điệp khi chơi

Chế độ chơi 2 người:

- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 31:** Đặt vị trí con trỏ lại về hàng 31
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, X_WIN_OUT:** Tải địa chỉ chuỗi vào DX
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi

- **jmp GAME_END:** Nhảy đến nhãn GAME_END

Chế độ chơi với máy:

- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 34:** Đặt vị trí con trỏ lại về hàng 34
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, HUMAN_WIN:** Tải địa chỉ chuỗi vào DX
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi
- **jmp GAME_END:** Nhảy đến nhãn GAME_END

O_WIN: Thủ tục khi bên dấu O thắng

```
O_WIN PROC
    mov al,select_mode
    cmp al,2
    je msg_for_computer

    call MSG_FOR_WINNER
    mov dh ,5
    mov dl ,31
    mov ah,2
    int 10h
    lea dx,O_WIN_OUT
    mov ah,9
    int 21h
    jmp GAME_END

    msg_for_computer:

    call MSG_FOR_LOSER
    mov dh ,5
    mov dl ,32
    mov ah,2
    int 10h
    lea dx,COMPUTER_WIN
    mov ah,9
    int 21h
    jmp GAME_END
    RET
O_WIN ENDP
```

- **mov AL, select_mode:** Gán giá trị biến select_mode cho AL
- **cmp AL, 2:** So sánh giá trị AL với 2

- **je msg_for_computer:** Nhảy đến nhãn msg_for_computer để in thông điệp khi chơi với máy
- **call MSG_FOR_WINNER:** Gọi thủ tục MSG_FOR_WINNER

Chế độ chơi 2 người:

- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 31:** Đặt vị trí con trỏ lại về hàng 31
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, O_WIN_OUT:** Tải địa chỉ chuỗi vào DX
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi
- **jmp GAME_END:** Nhảy đến nhãn GAME_END

Chế độ chơi với máy:

- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 34:** Đặt vị trí con trỏ lại về hàng 34
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, COMPUTER_WIN:** Tải địa chỉ chuỗi vào DX
- **mov AH, 9:** Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi
- **jmp GAME_END:** Nhảy đến nhãn GAME_END

CHECK_DRAW: Thủ tục check 2 người chơi hòa nhau

- **AL = 1** nếu là hòa
- **AL = 0** nếu không phải hòa (tức là còn ô trống để chơi tiếp).

```
CHECK_DRAW PROC
    mov cx,9
    lea si,BOARD
    check_draw_loop:
        mov bl,[si]
        cmp bl,'X'
        je  continue_check_draw_loop
        cmp bl,'O'
        je  continue_check_draw_loop
        jmp IS_NOT_DRAW
        continue_check_draw_loop:
            inc si
            loop check_draw_loop
    jmp IS_DRAW
    IS_NOT_DRAW:
        mov al,0
        RET
    IS_DRAW:
        mov al,1
    RET
CHECK_DRAW ENDP
```

- **mov cx,9:** CX được gán giá trị 9, đại diện cho 9 ô trên bàn cờ.
- **lea si, BOARD:** Trỏ SI tới vị trí đầu tiên của BOARD
- **check_draw_loop:** Vòng lặp kiểm tra từng ô
- **mov bl,[si]:** Lấy giá trị tại vị trí SI và đưa vào BL
- **cmp bl,'X' :** So sánh kí tự đang được trỏ với 'X'
- **je continue_check_draw_loop :** Nếu ô đó chứa ký tự 'X', thì chuyển đến nhãn **continue_check_draw_loop** để tiếp tục vòng lặp.
- **cmp bl,'O' :** So sánh kí tự đang được trỏ với 'O'
- **je continue_check_draw_loop :** Nếu ô đó chứa ký tự 'O', cũng tiếp tục vòng lặp.
- **jmp IS_NOT_DRAW:** Nếu ô đó không phải 'X' hoặc 'O', tức là ô trống, thì không phải hòa, nhảy đến nhãn **IS_NOT_DRAW**.
- **continue_check_draw_loop:** Tiếp tục vòng lặp:
- **inc si:** Tăng SI để kiểm tra ô tiếp theo.

- **loop check_draw_loop:** loop sẽ giảm CX đi 1 và nếu CX chưa bằng 0 thì quay lại nhãn **check_draw_loop**.
- **jmp IS_DRAW:** Nếu đã kiểm tra hết 9 ô mà không phát hiện ô trống nào thì là hòa, nhảy đến **IS_DRAW**.
- **Các nhãn kết thúc:**
- **IS_NOT_DRAW:** Trò chơi chưa hòa (còn ô trống), trả về **AL = 0.**
- **mov al,0**
- **RET**
- **IS_DRAW:** Trò chơi hòa (tất cả ô đều đã được đánh), trả về **AL = 1.**
- **mov al,1**
- **RET**
- **CHECK_DRAW ENDP:** Kết thúc thủ tục **CHECK_DRAW**.

GAME_DRAW: Thủ tục khi game hòa

```
GAME_DRAW PROC
    mov dh ,5
    mov dl ,36
    mov ah,2
    int 10h
    lea dx,DRAW_OUT
    mov ah,9
    int 21h
    jmp GAME_END
    RET
GAME_DRAW ENDP
```

- **mov DL, 5:** Đặt vị trí con trỏ lại về cột 5
- **mov DH, 36:** Đặt vị trí con trỏ lại về hàng 36
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **lea DX, DRAW_OUT:** Tải địa chỉ chuỗi vào DX
- **mov AH, 9:**Lệnh in chuỗi
- **int 21h:** Gọi ngắt in chuỗi
- **jmp GAME_END:** Nhảy đến nhãn GAME_END

MSG_FOR_WINNER: Thủ tục in lời chúc mừng chiến thắng

```
call MSG_FOR_WINNER

mov al,select_mode
cmp al,2
je msg_for_human

mov dh ,5
mov dl ,31
mov ah,2
int 10h
lea dx,X_WIN_OUT
mov ah,9
int 21h
jmp GAME_END

msg_for_human:
mov dh ,5
mov dl ,34
mov ah,2
int 10h

lea dx,HUMAN_WIN
mov ah,9
int 21h
jmp GAME_END
```

- **mov CX, 15:** Gán độ dài chuỗi để thực hiện loop
- **xor SI, SI:** Reset con trỏ về đầu chuỗi
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov DH, 3:** Đặt vị trí con trỏ lại về hàng 3
- **mov DL, 31:** Đặt vị trí con trỏ lại về cột 31
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ

Nhãn print_congratulation:

- **push CX:** Đẩy giá trị CX vào stack
- **mov AH, 9:** Hàm in ký tự có màu
- **mov CX, 1:** In một lần
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov BL, COLOR[SI]:** Gán mã màu theo thứ tự trong chuỗi COLOR vào BL thông qua SI

- **mov AL, msg_winner[SI]:** Gán ký trong chuỗi msg_winner vào AL thông qua SI
- **int 10h:** Gọi ngắt để in ký tự màu
- **inc SI:** Tăng con trỏ
- **inc DL:** Tăng vị trí cột con trỏ
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **pop CX:** Lấy giá trị ở đầu stack gán vào CX để thực hiện loop
- **loop print_congratulation:** Nhảy đến nhãn print_congratulation khi CX khác 0

MSG_FOR_LOSER: Thủ tục in khi người chơi thua máy

```
mov cx,20
xor si,si
mov bh,0
mov dh,3
mov dl,28
mov ah,2
int 10h
print_lose:
    push cx

    mov ah,9
    mov cx,1
    mov bh,0
    mov al,msg_lose[si]
    mov bl,12
    int 10h

    inc si
    inc dl
    mov ah,2
    int 10h

    pop cx
    loop print_lose
```

- **mov CX, 20:** Gán độ dài chuỗi để thực hiện loop
- **xor SI, SI:** Reset con trỏ về đầu chuỗi
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov DH, 3:** Đặt vị trí con trỏ lại về hàng 3
- **mov DL, 28:** Đặt vị trí con trỏ lại về cột 28

- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ

Nhãn print_lose:

- **push CX:** Đẩy giá trị CX vào stack
- **mov AH, 9:** Hàm in ký tự có màu
- **mov CX, 1:** In một lần
- **mov BH, 0:** Điều khiển con trỏ tại trang đang hiển thị
- **mov AL, msg_lose[SI]:** Gán ký trong chuỗi msg_lose vào AL thông qua SI
- **mov BL, 12:** Gán mã màu đỏ vào BL
- **int 10h:** Gọi ngắt để in ký tự màu
- **inc SI:** Tăng con trỏ
- **inc DL:** Tăng vị trí cột con trỏ
- **mov AH, 2:** Hàm đặt vị trí con trỏ
- **int 10h:** Gọi ngắt để đặt lại con trỏ
- **pop CX:** Lấy giá trị ở đầu stack gán vào CX để thực hiện loop
- **loop print_lose:** Nhảy đến nhãn print_lose khi CX khác 0

CLEAR_SCREEN: Thủ tục xóa màn hình

```
CLEAR_SCREEN PROC
    mov ax,3
    int 10h
    RET
CLEAR_SCREEN ENDP
```

- **mov AX, 3:** Hàm xóa màn hình
- **int 10h:** Gọi ngắt xóa màn hình

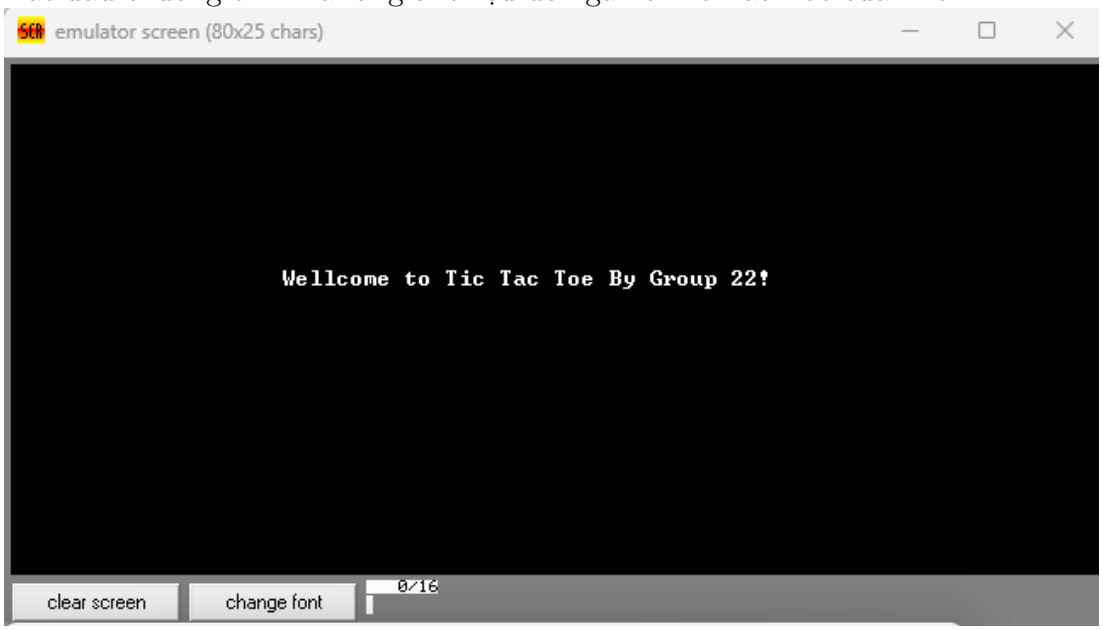
DELAY: Thủ tục delay màn hình hiển thị

```
    mov cx,0123h
    delay_screen:
        nop
        loop delay_screen
    RET
```

- **mov CX,0123H:** Gán giá trị cho CX để thực hiện vòng lặp
- **Nhãn delay_screen:**
  - **nop:** Không làm gì cả nhưng vẫn tốn thời gian CPU.
  - **loop delay_screen:** Nhảy đến nhãn delay_screen khi CX khác 0

GIAO DIỆN CHƯƠNG TRÌNH

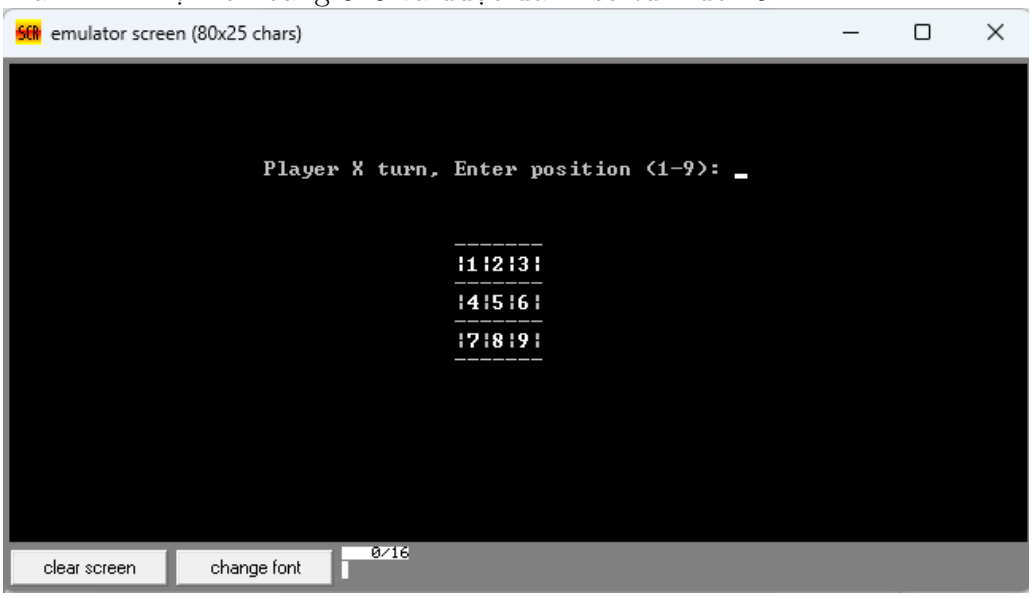
- Bắt đầu chương trình là lời giới thiệu đến game Tic Tac Toe của nhóm 22.



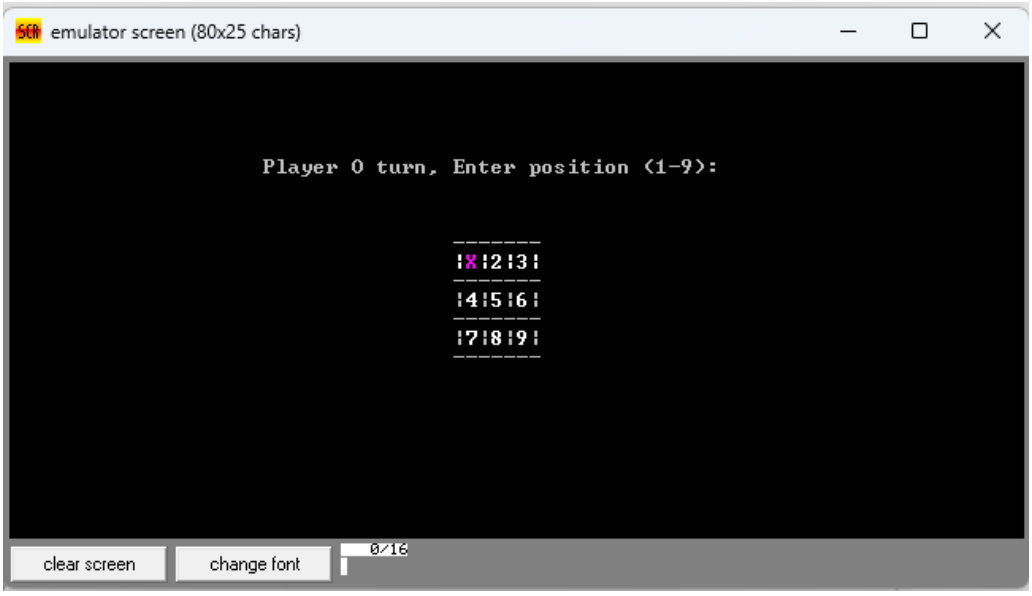
- Tiếp theo màn hình hiện lên tên và mã SInh viên của 4 thành viên đã làm game, và dòng cuối yêu cầu người dùng nhập một phím bất kì để bắt đầu trò chơi.

- Màn hình hiện lên 2 lựa chọn chế độ chơi là chơi với người (**1 - HUMAN**) hoặc chơi với máy (**2 - COMPUTER**), người chơi nhập câu trả lời ở dòng "**Press your answer**".

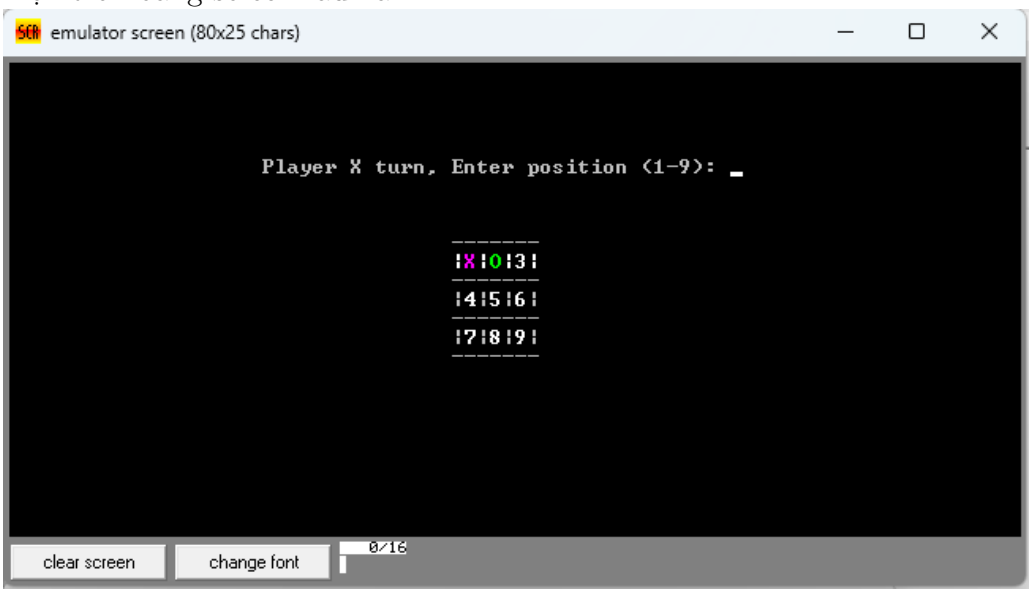
- **Chế độ chơi với người (1 - HUMAN) :**
  - Màn hình hiện lên bảng 3x3 và được đánh số từ 1 đến 9.



  - Người chơi với dấu 'X' sẽ đi trước, sau khi người chơi 'X' chọn ô thì dấu 'X' hiện trên bảng sẽ có màu hồng.

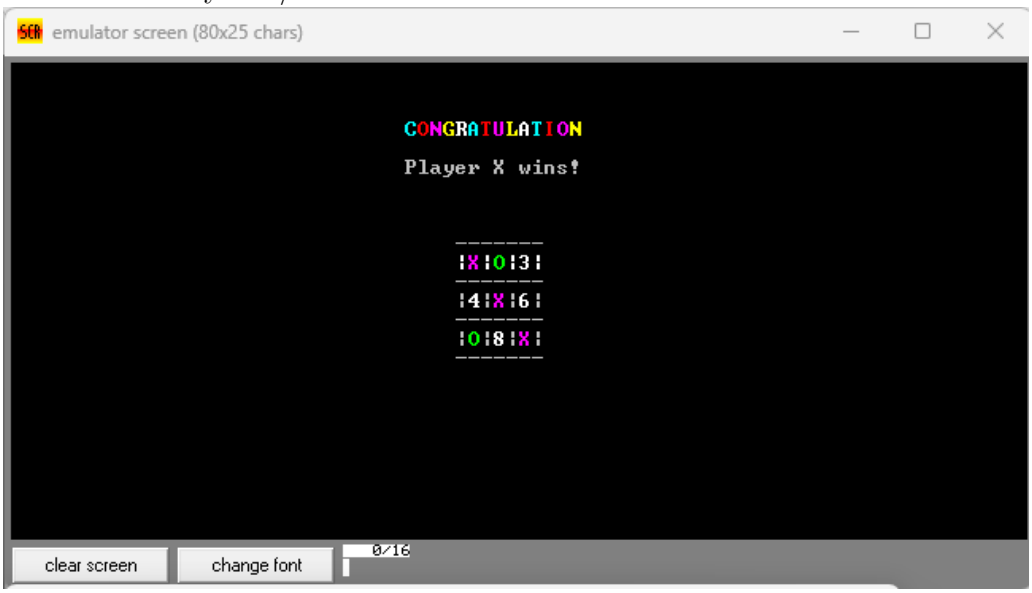


– Người chơi với dấu ‘O’ sẽ đi sau, sau khi người chơi ‘O’ chọn ô thì dấu ‘O’ hiện trên bảng sẽ có màu xanh.

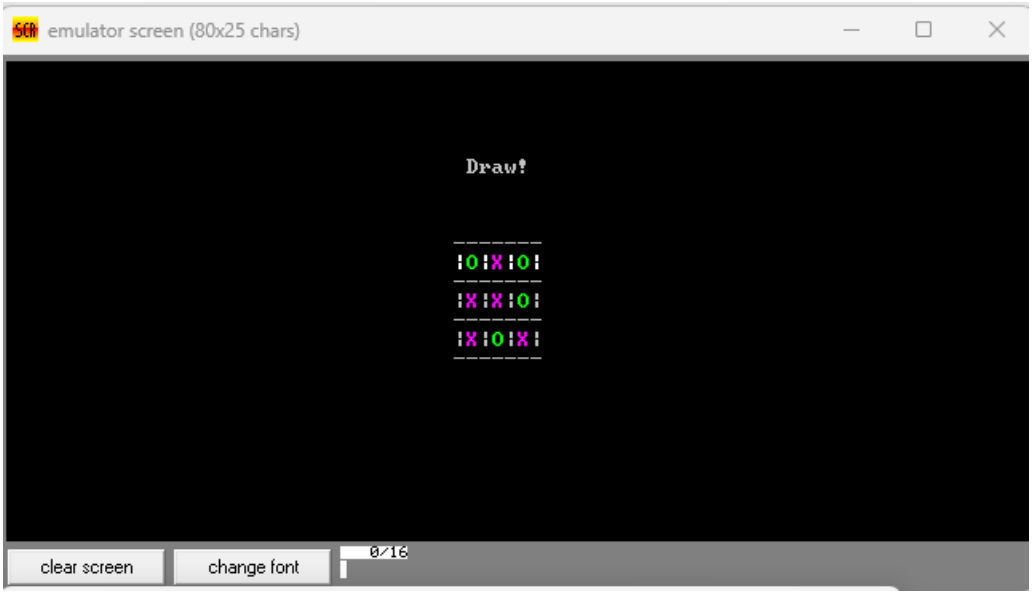


– Nếu người chơi chọn một ô đã được chọn thì sẽ bị hệ thống cho là nước đi lỗi và màn hình sẽ hiện lên dòng chữ “Invalid move! Try again” người chơi sẽ phải chọn lại nước đi khác.

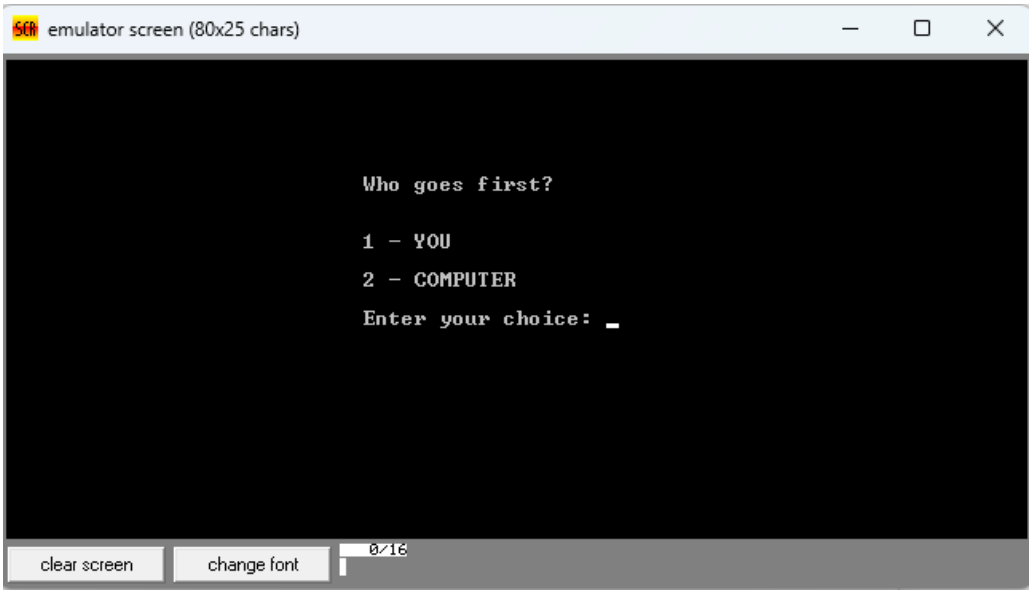
– Khi 1 người chơi chiến thắng thì màn hình sẽ hiện lên "CONGRATULATION" và "Player X/O wins".



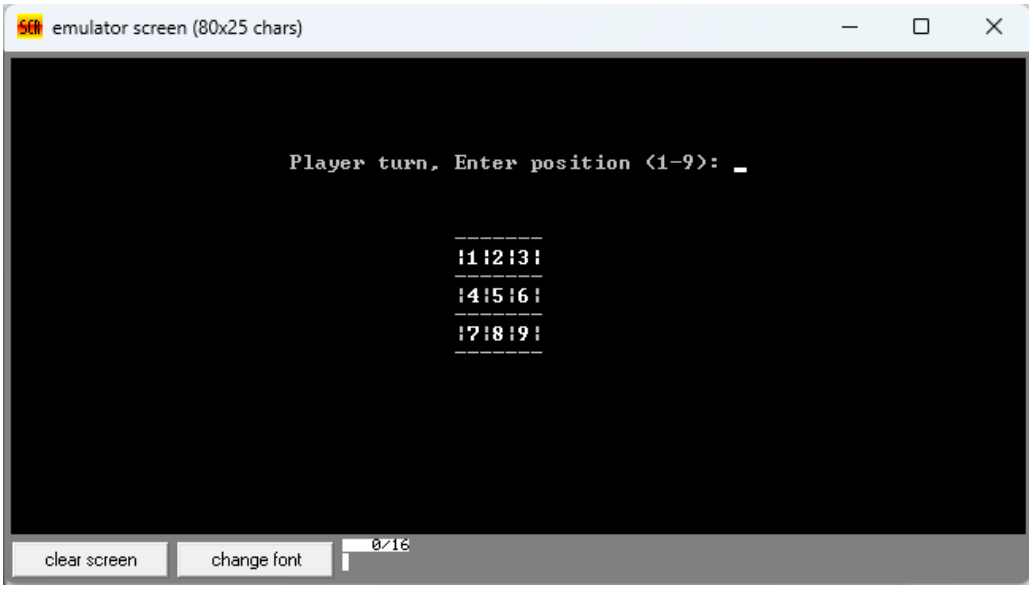
– Còn nếu hòa thì màn hình sẽ hiện lên " Draw".



- **Chế độ chơi với máy (2 - COMPUTER) :**
  - Sau khi chọn chế độ chơi với máy, màn hình sẽ hiện lên 2 lựa chọn cho người chơi (1 - YOU) hay máy (2 - COMPUTER) sẽ là người đi trước, người chơi nhập câu trả lời ở dòng "Enter your choice".
    - Nếu chọn 1 - YOU thì người chơi sẽ đi trước với dấu 'X'
    - Nếu chọn 2 - COMPUTER thì máy sẽ đi trước với dấu 'O'

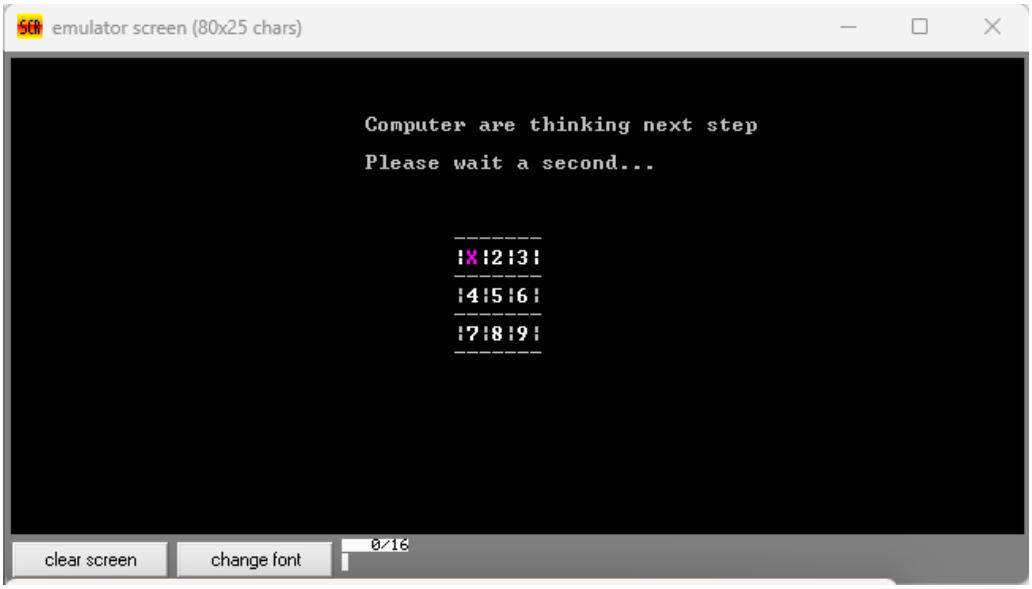


  - Khi đến lượt người chơi thì màn hình sẽ hiện lên "Player turn, Enter poSItion (1-9)" để người chơi chọn ô đánh dấu 'X'.

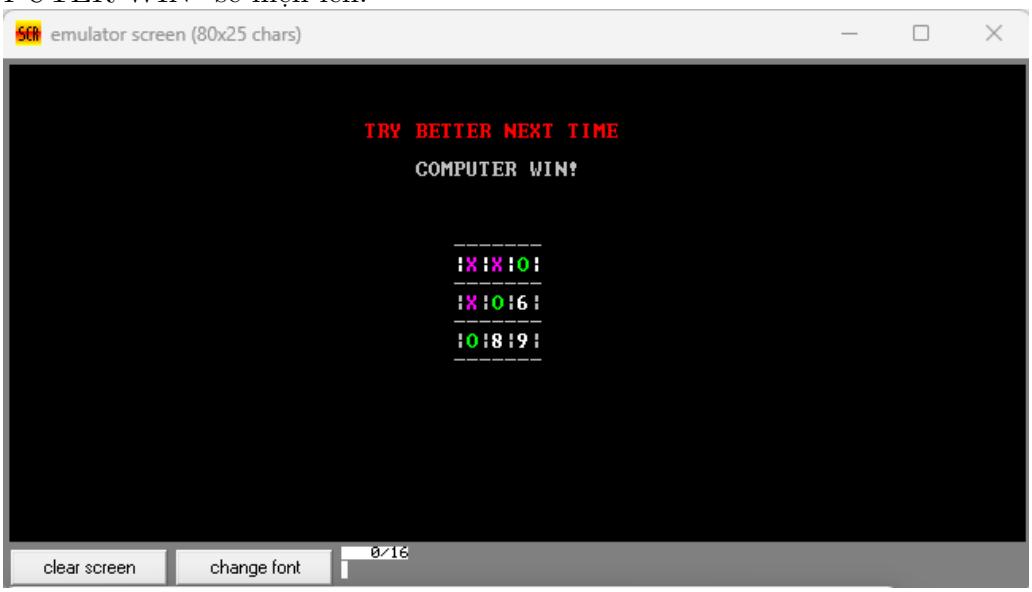


– Nếu người chơi chọn ô đã được chọn thì sẽ tính là nước đi lỗi và màn hình sẽ hiện lên thông báo như chế độ chơi 2 người.

– Khi đến lượt máy thì màn hình sẽ hiện lên màn hình đợi để máy tính toán nước đi.

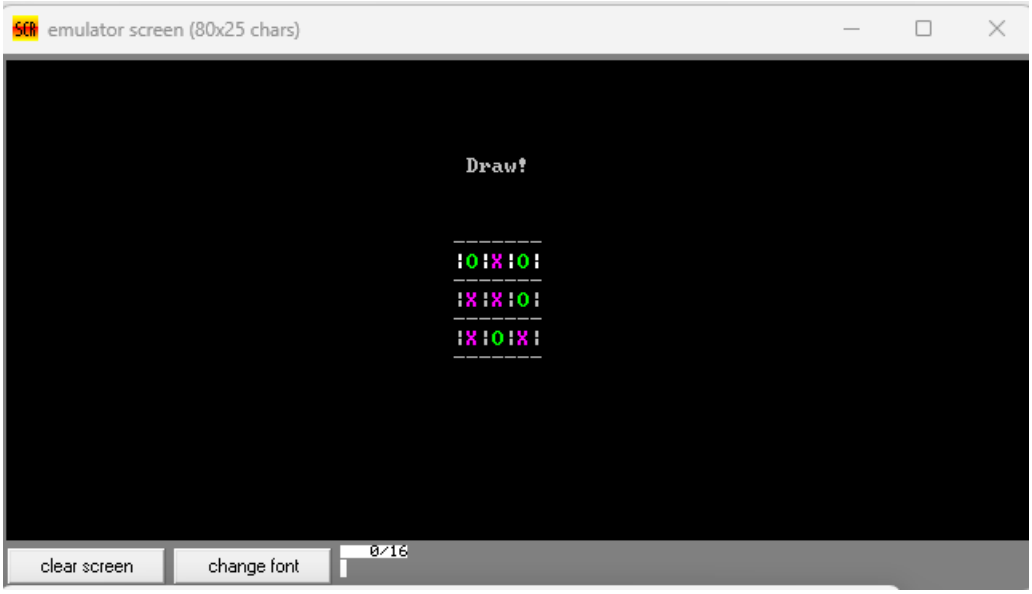


– Nếu máy thắng thì thông báo "TRY BETTER NEXT TIME" và "COMPUTER WIN" sẽ hiện lên.



– Nếu người chơi thắng thì thông báo "CONGRATULATION" và "HUMAN WIN" sẽ hiện lên.

– Còn hòa thì thông báo sẽ hiện lên như chế độ chơi 2 người.



MÃ NGUỒN

SOURCE CODE LATEX

END