**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**BÀI TẬP LỚN**

**KIẾN TRÚC MÁY TÍNH**

**Nhóm: Lớp N02 – Nhóm 20**

Tên thành viên

|  |  |  |
| --- | --- | --- |
| Tên thành viên | Mã số sinh viên | Bài tập phần I |
| Hồ Việt Phương | B20DCCN521 | 1,5,10,13,18 |

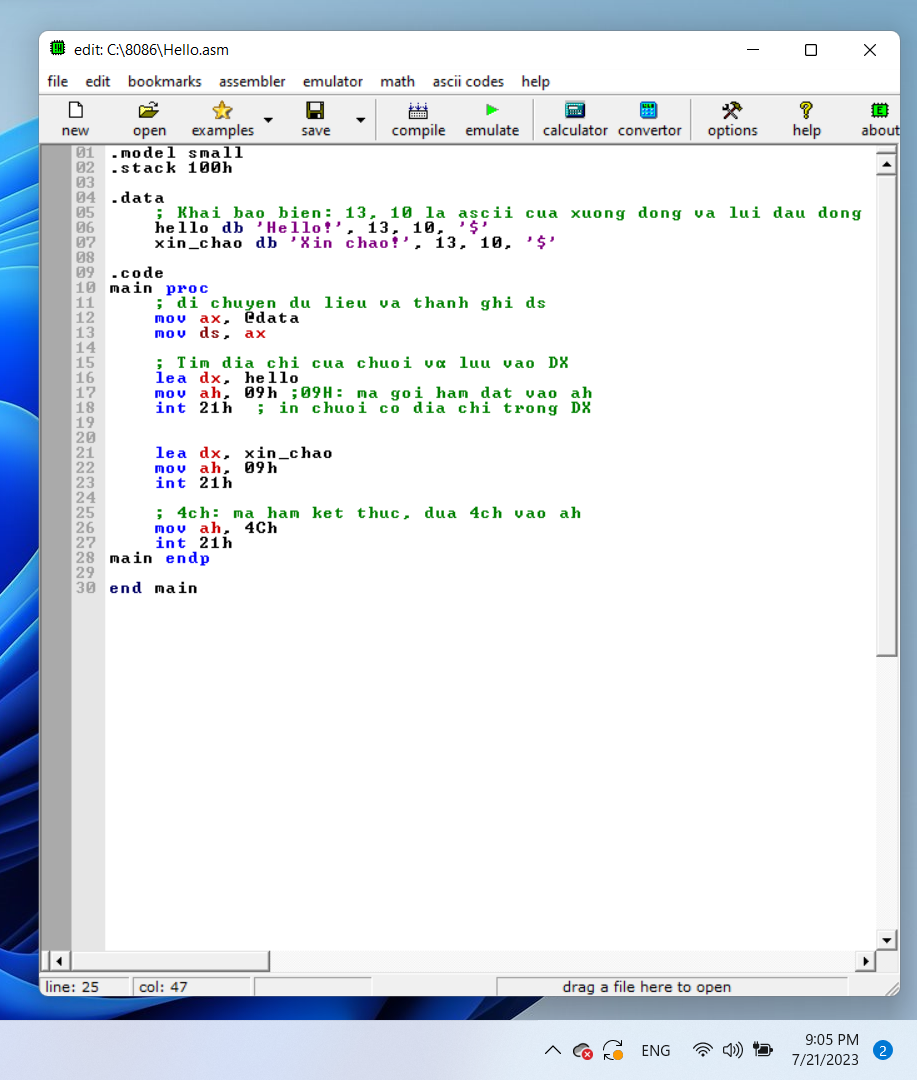
**Giảng viên hướng dẫn**: ThS. Đinh Xuân Trường

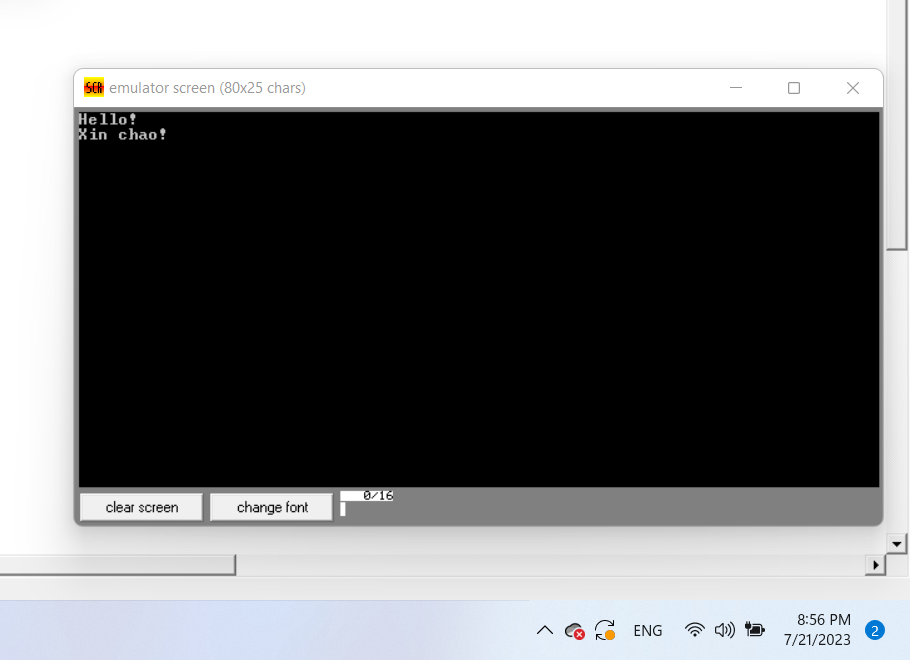
**Hà Nội 2023**

**PHẦN 1: PHẦN LÀM CÁ NHÂN**

***Bài số 1: Lập trình hợp ngữ Assembly***

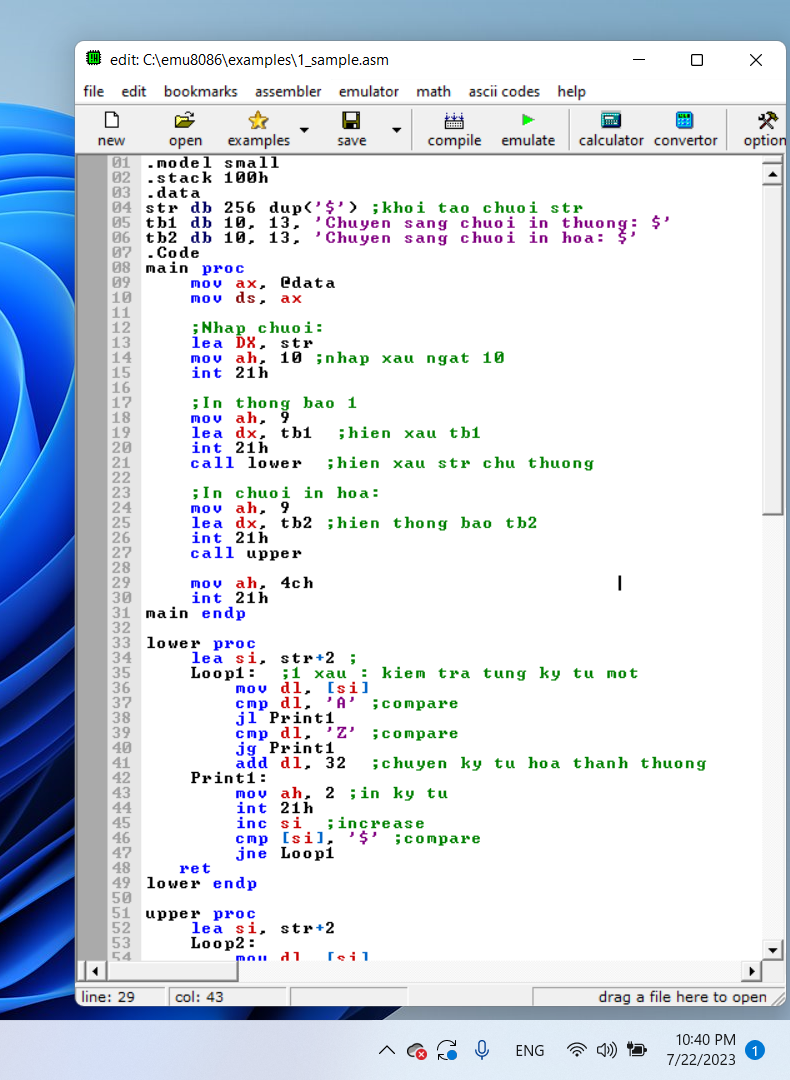
Câu 1: *Viết chương trình hợp ngữ in ra lời chào Tiếng Anh và Tiếng Việt*

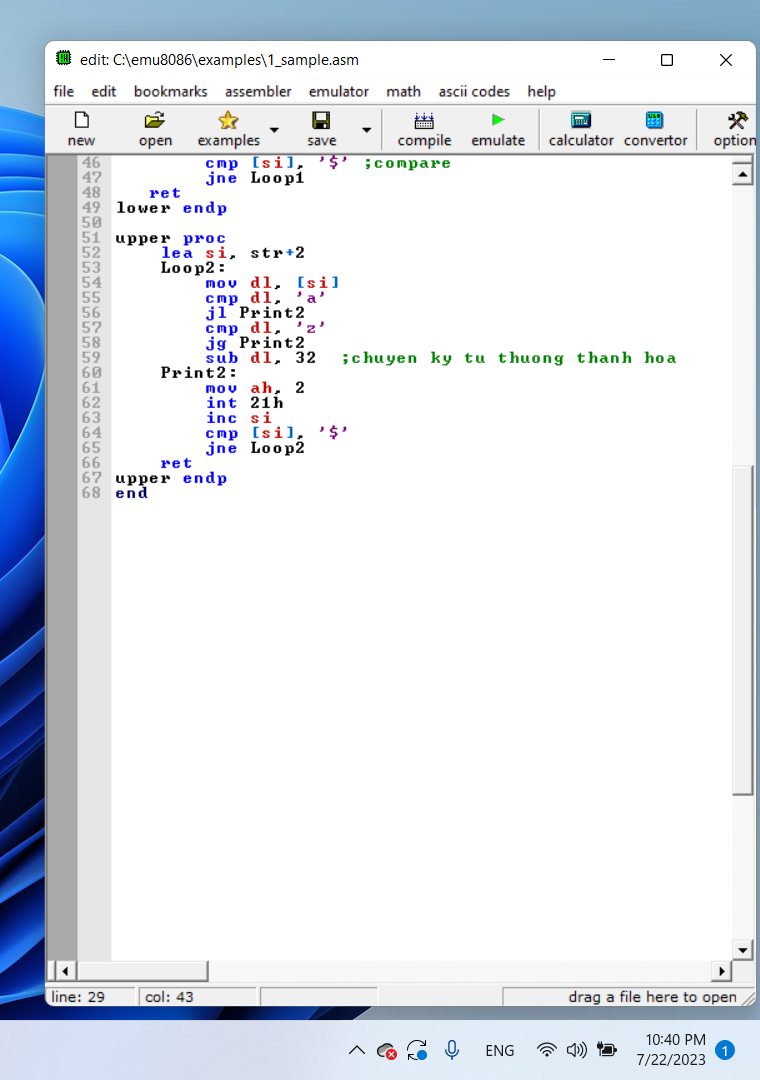


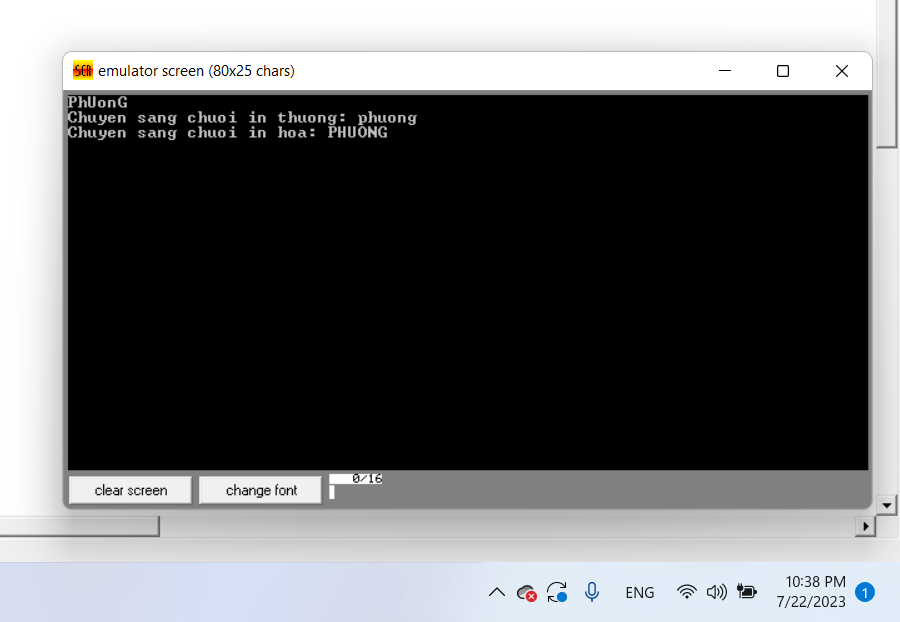


Câu 5: *Viết chương trình hợp ngữ Assembly cho phép nhập 1 chuỗi ký tự, in*

*ra màn hình chuỗi ký tự đó theo dạng viết hoa và viết thường.*

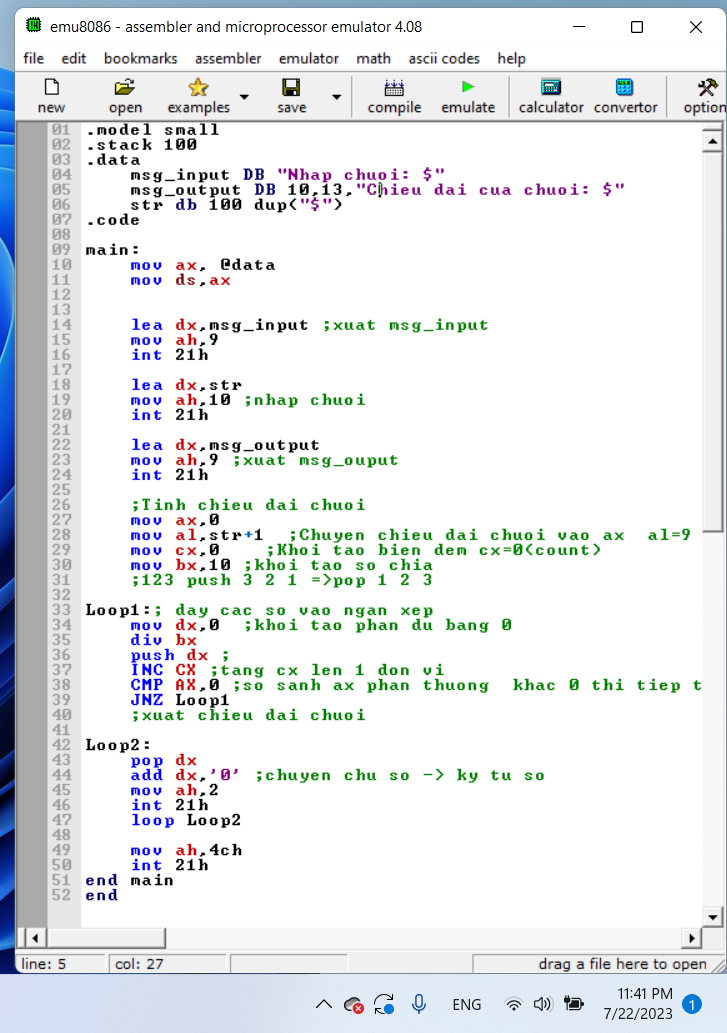


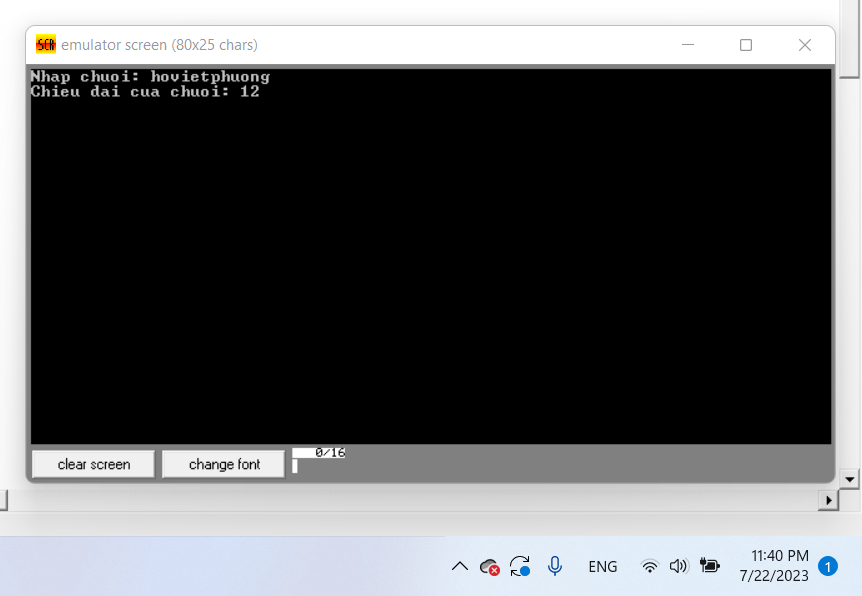




Câu 10: *Viết chương trình hợp ngữ Assembly yều cầu đếm chiều dài của một*

*chuỗi ký tự cho trước.*

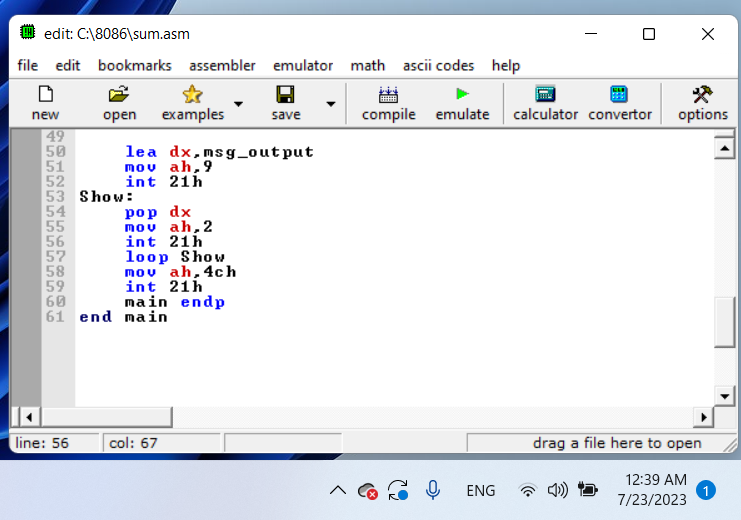


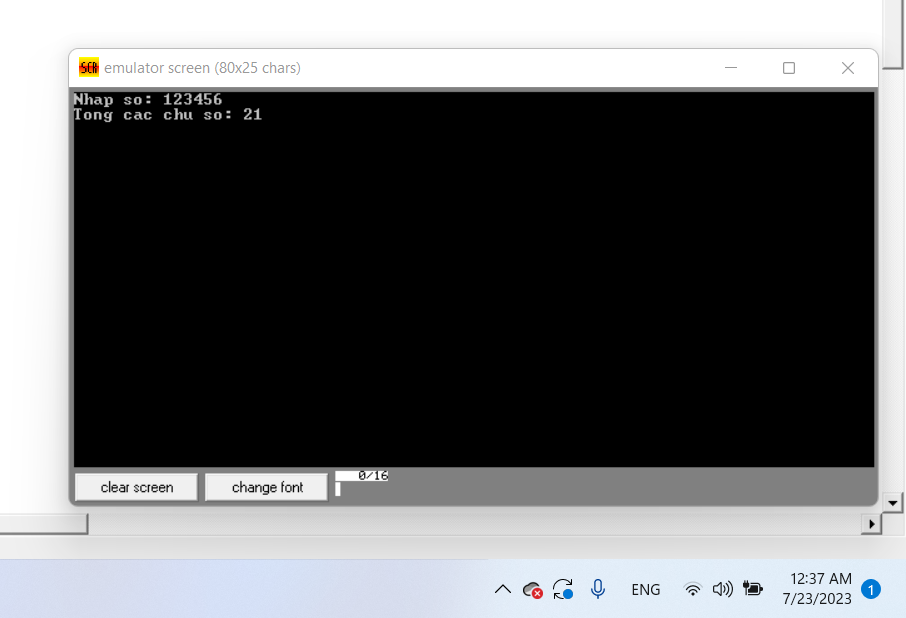


Câu 13: *Viết chương trình hợp ngữ Assembly cho phép nhập vào các số và in*

*ra màn hình tổng của các số đó.*

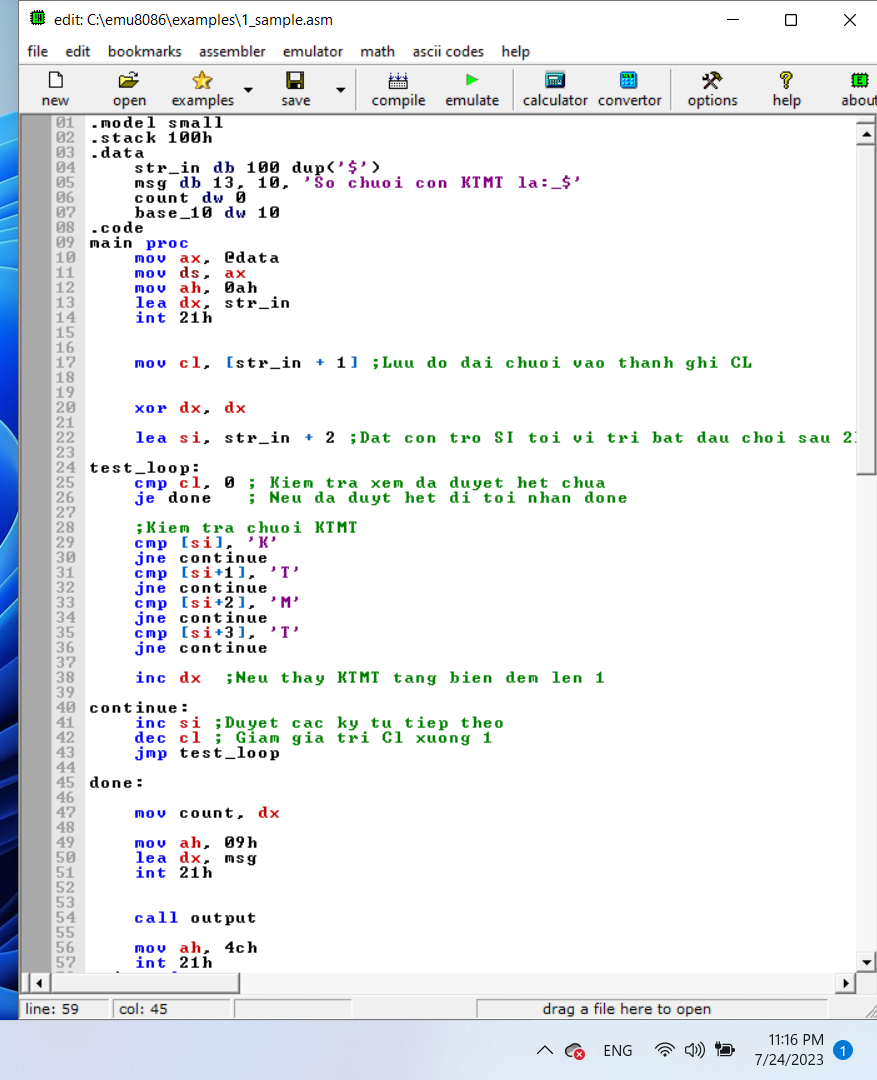




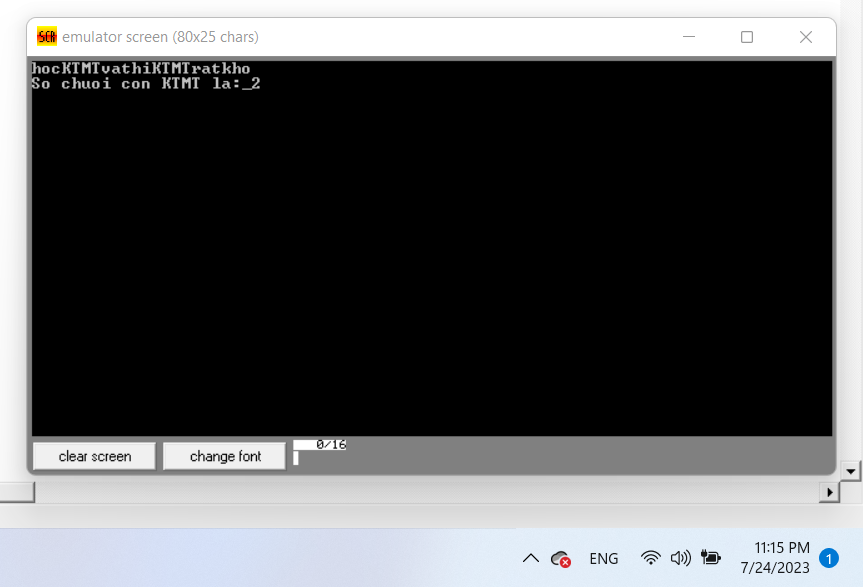


Câu 18: *Viết chương trình hợp ngữ đếm số lần xuất hiện của chuỗi con*

*"ktmt"trong một chuỗi. In kết quả dưới dạng số thập phân.*





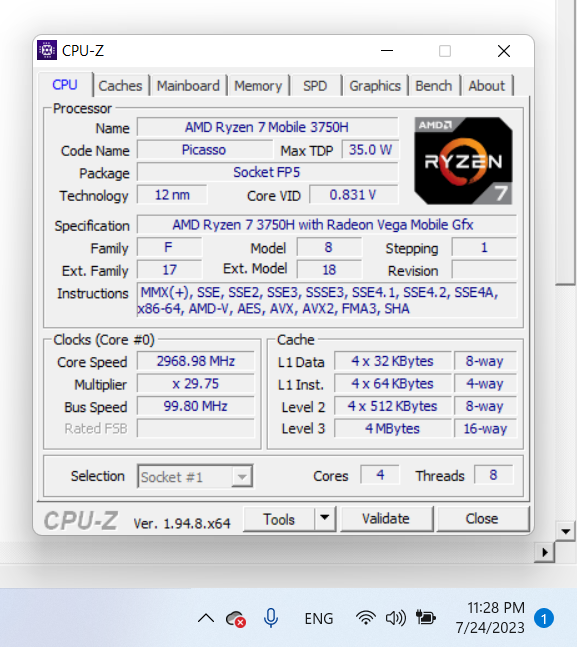
****

**Bài số 2:** Thực hành phân tích khảo sát bộ nhớ:

* 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.)

Khảo sát cấu hình của máy và hệ thống bộ nhớ của máy đang sử dụng Sử dụng phần mềm CPU-Z 64-bit v1.85.0x64:

CPU:

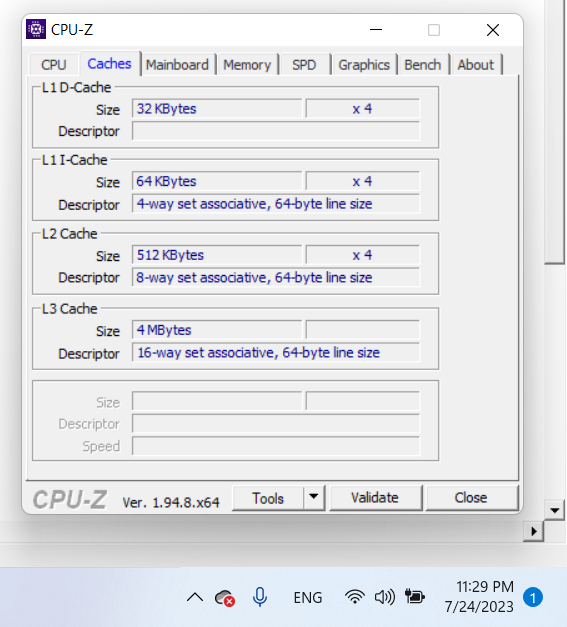


+ Bộ vi xử lí: : AMD Ryzen 7 Mobile 3750H

+ Thế hệ CPU: Renoir , tích hợp :

* Bộ xử lí đồ họa: Radeon
* Sản xuất trên tiến trình công nghệ: 12nm

Cache:



+ L1D-Cache (Level 1 Data Cache): lưu trữ dữ liệu

* Size: 32KB x 4 = 128 KB
* 8 đường (way)
* Ánh xạ tập kết hợp

+ L1 I-Cache (Level 1 Instruction Cache): lưu trữ lệnh

* Size: 32KB x 4 = 128 KB
* 8 đường (way)
* Ánh xạ tập kết hợp
* Kích thước 1 line: 64B

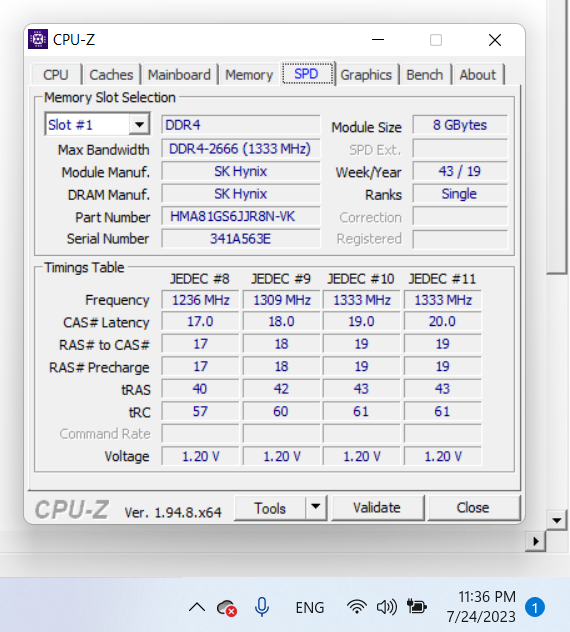
+ L2 Cache

* Size: 512KB x 4 = 2048 KB
* 8 đường (way)
* Ánh xạ tập kết hợp
* Kích thước 1 line: 64B

+ L3 Cache

* Size: 4MB
* 16 đường (way)
* Ánh xạ tập kết hợp
* Kích thước 1 line: 64B

RAM/ROM:



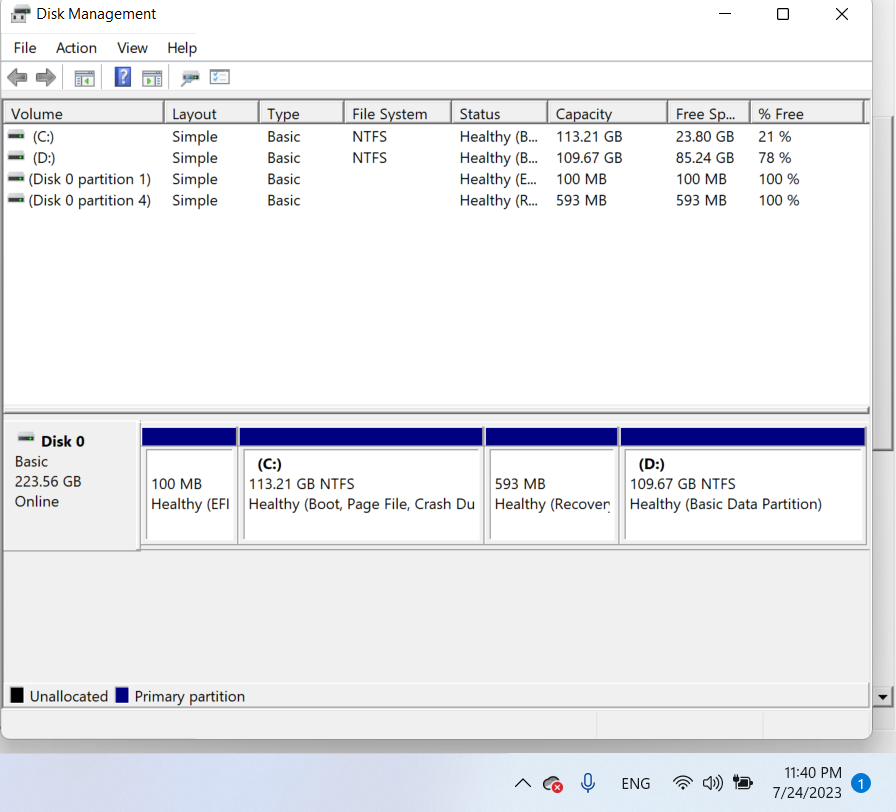
- ROM:

* Dung lượng: 256 GB
* M2. PCIe SSD

- RAM:

* Dung lượng RAM (Size): 8 GB
* Loại RAM (Type): DDR4
* Tốc độ RAM: 2666MHz
* Số lượng RAM cắm trên máy (Channel): 1 RAM (Single)
* Số lượng khe cắm RAM (Slot #): Slot #1
* Số khe cắm rời: 2
* Số khe RAM còn lại: 1
* Số RAM onboard: 0
* Hỗ trợ RAM tối đa: 32 GB
* Tốc độ băng thông tối đa (Max Bandwidth): 1333 MHz

Bộ nhớ ngoài:



- Bộ nhớ ngoài:

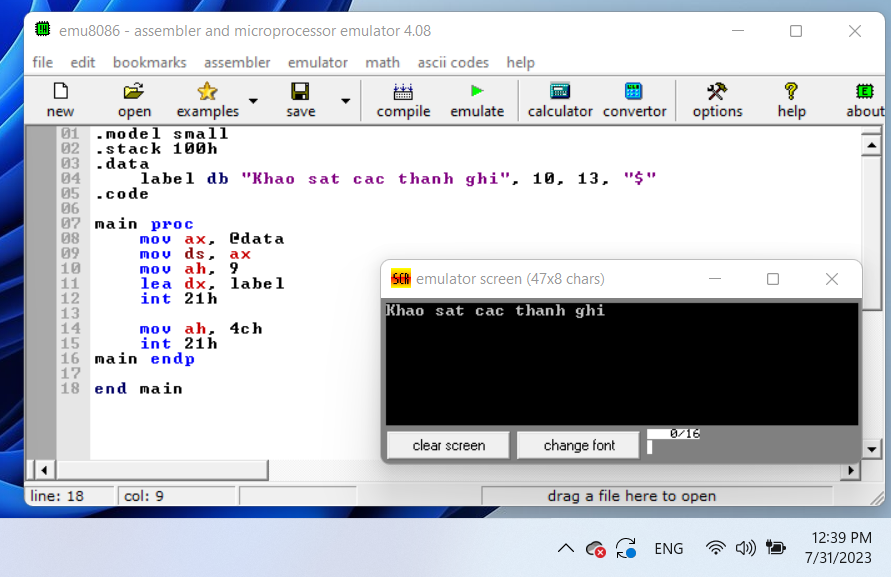
+ Ổ đĩa cứng, CD: không cung cấp

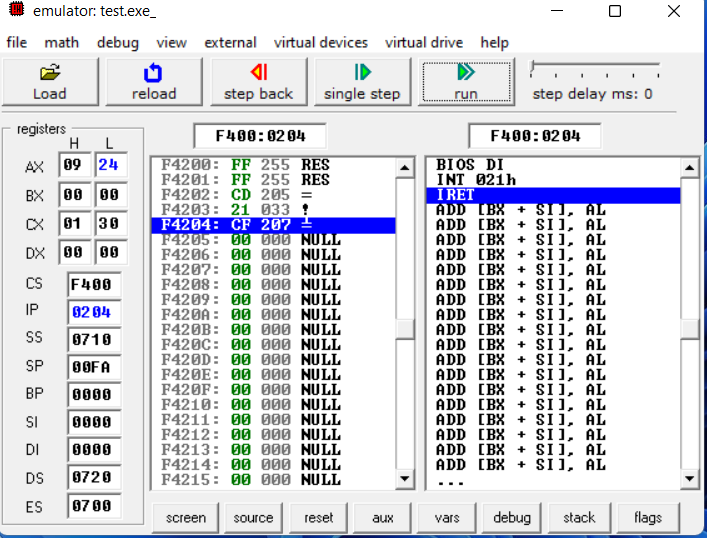
+ Thiết bị vào ra:

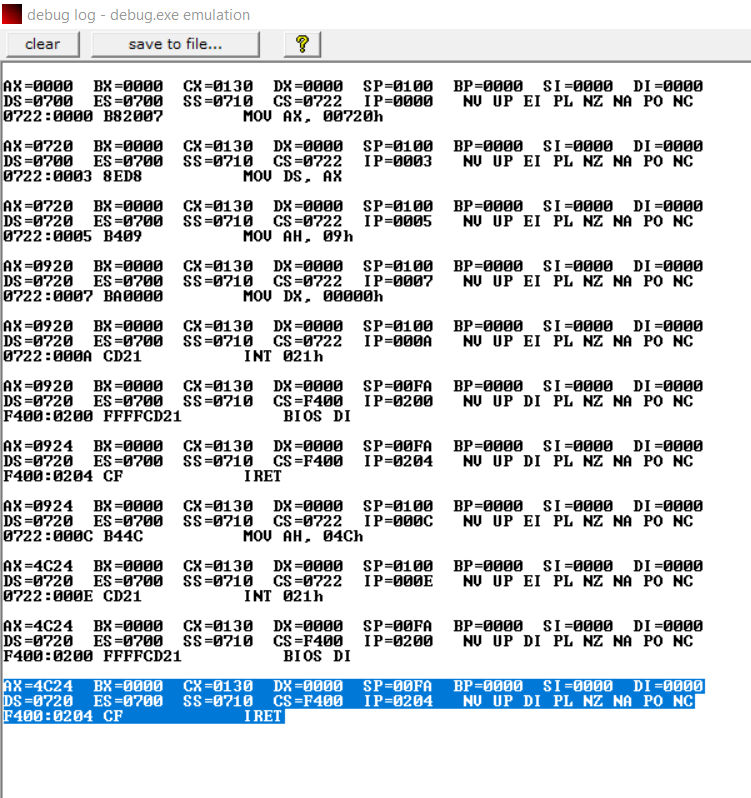
* 1 cổng USB 3.2 thế hệ 2 Type-C
* 3 cổng USB 3.2 Thế hệ 1 Type-A.
* 1 cổng HDMI (2.0b)
* 1 cổng Ethernet LAN RJ45
* 1 cổng âm thanh jack 3.5mm hỗ trợ tai nghe, micro hoặc loa ngoài.
* *Dùng công cụ Debug khảo sát nội dung các thanh ghi IP, DS, ES, SS, CS, BP, SP. Công cụ sử dụng: emu8086 microprocessor emulator*

 Các bước thực hiện:

* Mở file bằng phần mềm trên
* Chọn emulate trên thanh công cụ rồi chọn nút debug nằm cuối của cửa sổ vừa mở ra
* Chạy Single step để xem kết quả debug từng mã lệnh từ đầu đến cuối







* *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.*
* Khi chương trình bắt đầu chạy, hệ điều hành tự động khởi tạo các thanh ghi, vùng nhớ và cấp phát không gian địa chỉ cho chương trình.
* Tương ứng với các câu lệnh trong mã nguồn, nội dung các thanh ghi có thể thay đổi hoặc không.
* Sau mỗi câu lệnh địa chỉ IP đều sẽ thay đổi, hoặc có thể không dựa vào địa chỉ của lệnh tiếp theo. Do IP sẽ lưu địa chỉ của lệnh tiếp theo
* Ban đầu SP=0100 BP=0000 DS=0700 ES=0700 SS=0710 CS=0722 IP=0000
* Sau 2 câu lệnh đầu tiên:
* mov ax, @data

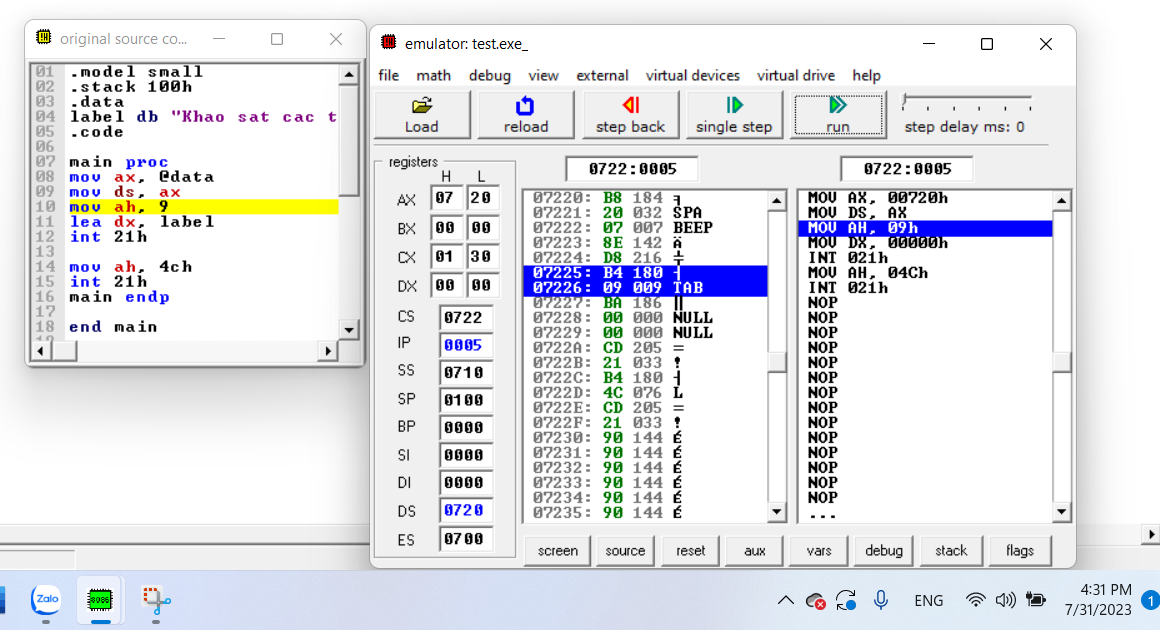
mov ds, ax

+ Đầu tiên đưa địa chỉ của phân đoạn dữ liệu (Data Segment) vào thanh ghi AX. Sau đó địa chỉ phân đoạn dữ liệu này được lưu vào thanh ghi DS.

-> Thiết lập địa chỉ của phân đoạn dữ liệu (Data Segment) trong thanh ghi DS.

Khi đó chỉ thanh ghi IP và DS thay đổi, do IP lưu địa chỉ của lệnh tiếp theo, sau 2 lệnh trên thì thanh ghi IP cũng thay đổi:

SP=0100 BP=0000 **DS=0720** ES=0700 SS=0710 CS=0722 **IP=0005**



* Sau 3 lệnh tiếp theo là:

mov ah, 9

lea dx, label

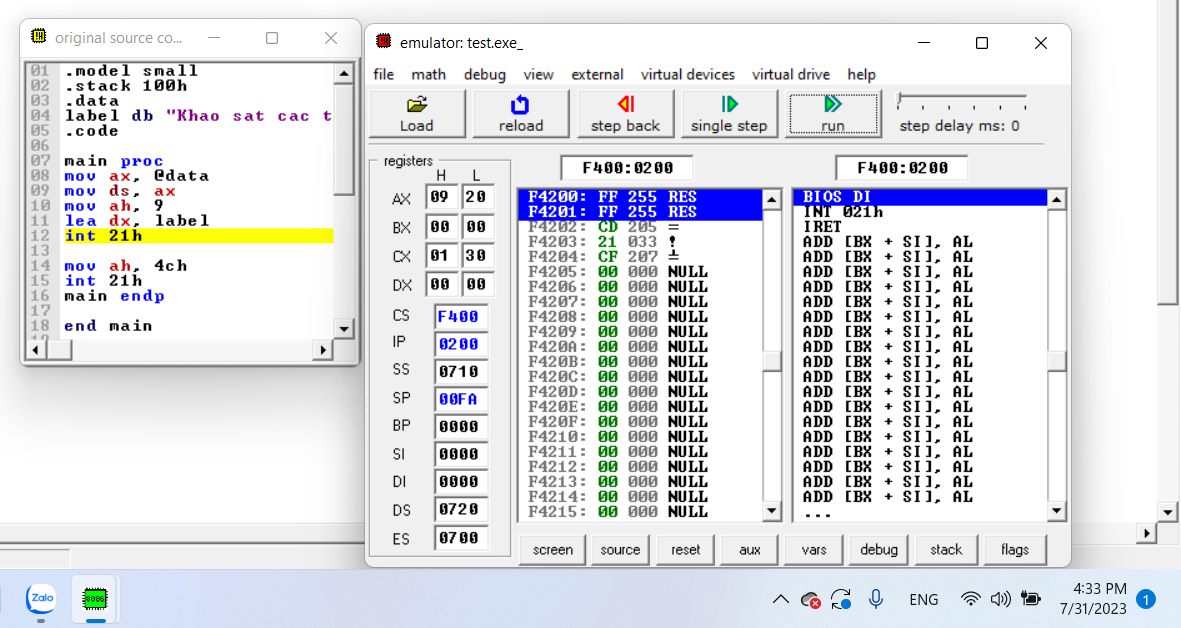
int 21h

+ Khi đó, hàm 09 của ngắt 21H sẽ làm thay đổi thanh ghi CS, IP, SP, cụ thể:

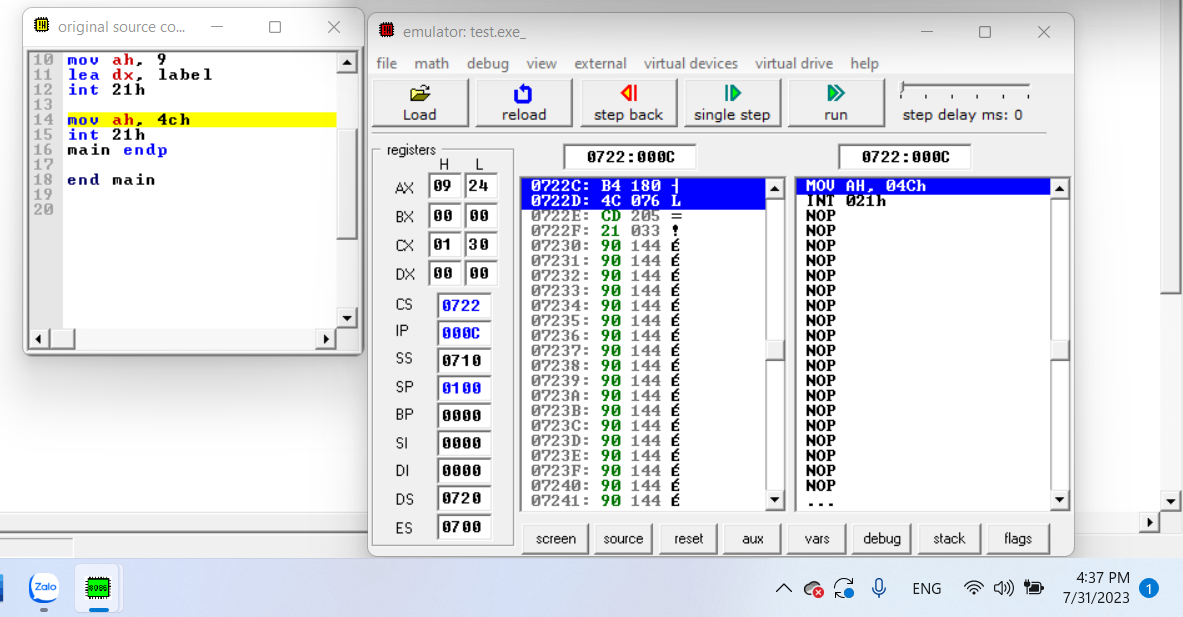
* **SP=00FA** BP=0000 DS=0720 ES=0700 SS=0710 **CS=F400** **IP=0200**

+ SP và CS bị thay đổi là do:

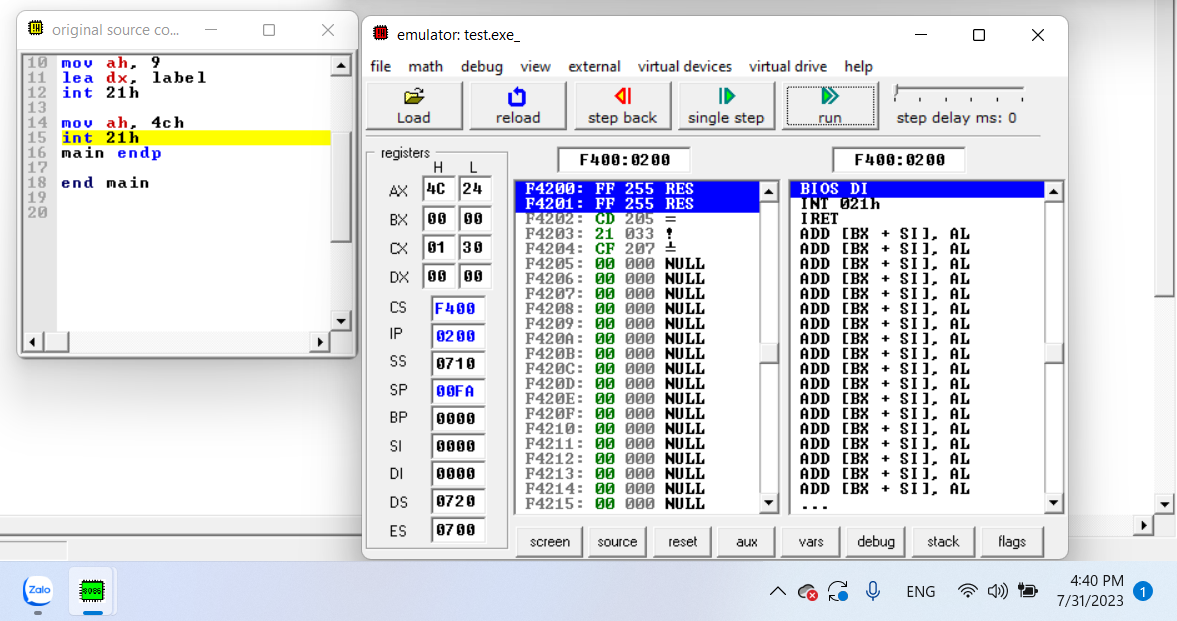
Hàm 09 của ngắt 21H: Dừng lệnh hiện tại, đưa giá trị con trỏ lệnh CS:IP hiện tại (địa chỉ của lệnh sau lệnh gọi ngắt) vào lưu trữ ở hai phần tử trên đỉnh Stack. Xác định địa chỉ trong bảng vector ngắt chứa vector ngắt, lấy giá trị tại đây (2 phần tử liên tiếp) để đưa vào cặp thanh ghi con trỏ lệnh CS:IP (địa chỉ của lệnh đầu tiên của chương trình con phục vụ hàm ngắt). Bắt đầu thực hiện chương trình con phục vụ ngắt cho đến khi gặp lệnh kết thúc chương trình này. Lấy nội dung của hai phần tử trên đỉnh Stack đặt vào lại cặp thanh ghi con trỏ lệnh để quay trở lại tiếp tục thực hiện lệnh sau lệnh gọi ngắt trong chương trình.



* Đến hàm 4CH của ngắt 21H để kết thúc chương trình, tương tự như cách giải thích sự thay đổi của hàm 09 của ngắt 21H trên.
* **SP=0100** BP=0000 DS=0720 ES=0700 SS=0710 **CS=0722** **IP=000C**



* Đến lệnh int21h:
* **SP=00FA** BP=0000 DS=0720 ES=0700 SS=0710 **CS=F40**0 **IP=0200**



**\* Vậy nội dung của từng thanh ghi là:**

- **IP** (Instruction Pointer): con trỏ lệnh, IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo này ứng với CS:IP

- **DS** (Data Segment): Thanh ghi đoạn dữ liệu, lưu địa chỉ đoạn chứa dữ liệu (các biến) trong chương trình

- **ES** (Extra data Segment) :Thanh ghi đoạn dữ liệu mở rộng, lưu địa chỉ đoạn chứa dữ liệu mở rộng trong chương trình.

- **SS** (Stack Segment): Thanh ghi đoạn ngăn xếp, lưu địa chỉ đoạn của vùng ngăn xếp.

- **CS** (Code Segment): Thanh ghi đoạn mã lệnh, lưu địa chỉ đoạn chứa mã lệnh chương trình của người sử dụng

- **BP** (Base Pointer): con trỏ cơ sở. BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS, Thanh ghi con trỏ nền dùng để lấy số liệu từ ngăn xếp. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS:BP

- **SP**: Thanh ghi con trỏ ngăn xếp luôn chỉ vào địa chỉ đỉnh ngăn xếp.

**PHẦN II: BÀI TẬP NHÓM**

**CHỦ ĐỀ 6: LẬP TRÌNH GAME TỰ CHỌN**

**LẬP TRÌNH GAME FLAPPY BIRD**

1. **Giới thiệu đề tài**

* Mô phỏng lại tựa game nổi tiếng Flappy Bird trên trình giả lập vi xử lý 8086

1. **Nội dung chính và kiến thức sử dụng**

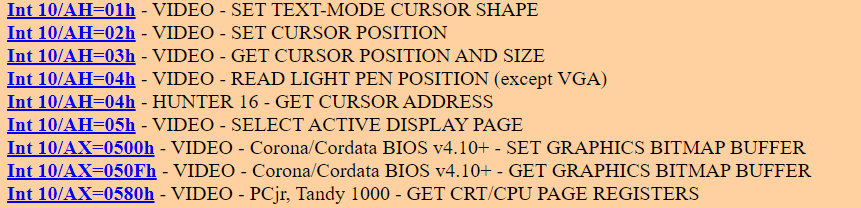
* Các kiến thức về lập trình assembly 8086 đặc biệt về các hàm ngắt, trình giả lập DOSBox.

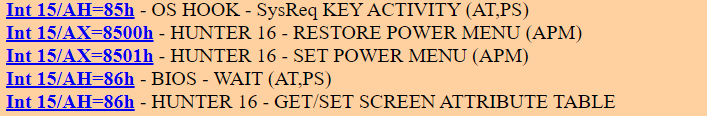
1. **Ngắt và xử lý ngắt trong hệ 8086**

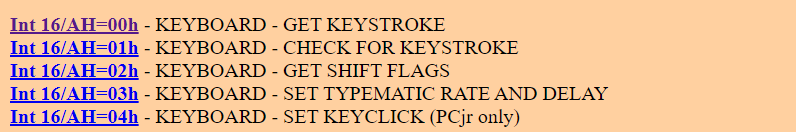
* Ngắt là việc tạm dừng chương trình đang chạy để CPU có thể chạy một chương trình khác nhằm xử lý một yêu cầu do bên ngoài đưa tới CPU như yêu cầu vào/ ra hoặc do chính yêu cầu của bên trong CPU như lỗi khi tính toán.
* Trong cách tổ chức trao đổi dữ liệu thông qua việc thăm dò trạng thái sẵn sàng của thiết bị ngoại vi, trước khi tiến hành bất kỳ cuộc trao đổi nào dữ liệu CPU phải dành toàn bộ thời gian vào việc xác định trạng thái sẵn sàng làm việc của thiết bị ngoại vi. Để tận dụng khả năng của CPU để làm thêm được nhiều công việc khác nữa, chỉ khi nào có yêu cầu trao đổi dữ liệu thì mới yêu cầu CPU tạm dừng công việc hiện tại để phục vụ trao đổi dữ liệu. Sau khi hoàn thành việc trao đổi dữ liệu thì CPU lại phải quay về để làm tiếp công việc bị đang bị gián đoạn.
* Trong các tín hiệu của CPU 8086 có tín hiệu cho các yêu cầu ngắt che được INTR và không che được NMI, chính các tín hiệu này sẽ được sử dụng vào việc đưa các yêu cầu ngắt từ bên ngoài đến CPU.

1. **Các loại ngắt trong 8086**

* Nhóm các ngắt cứng (Hardware Interrupts): đó là các yêu cầu ngắt CPU do tín hiệu đến từ chân INTR và NMI. Ngắt cứng INTR là yêu cầu ngắt che được. Các lệnh CLI và STI có ảnh hưởng trực tiếp tới trạng thái của cờ IF trong bộ vi xử lý, tức là ảnh hưởng tới việc CPU có nhận biết yêu cầu ngắt tại chân này hay không. Yêu cầu ngắt tại chân INTR có thể có kiểu ngắt N nằm trong khoảng 0-FFH. Kiểu ngắt này phải được đưa vào bit dữ liệu để CPU có thể đọc được khi có xung trong chu kỳ trả lời chấp nhận ngắt.
* Nhóm các ngắt mềm (Software Interrupts): khi CPU thực hiện các lệnh ngắt dạng INT N, trong đó N là số hiểu (kiểu) ngắt nằm trong khoảng 00-FFH (0-255).
* Nhóm các hiện tượng ngoại lệ: đó là các ngắt do lỗi nảy sinh trong quá trình hoạt động của CPU như phép chia cho 0, xảy ra khi tràn tính toán.
* Để phục vụ cho việc tương tác với bàn phím, xử lý các sự kiện trong game nhóm em sử dụng các hàm ngắt mềm có sẵn trong vi xử lý 8086 ví dụ như hàm ngắt 10h, 15h, 16h, 21h,…





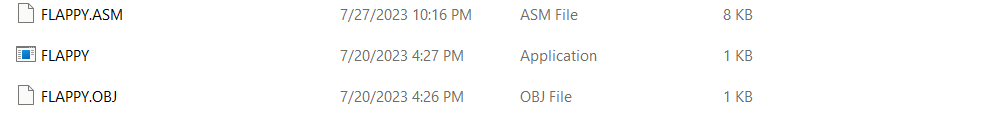


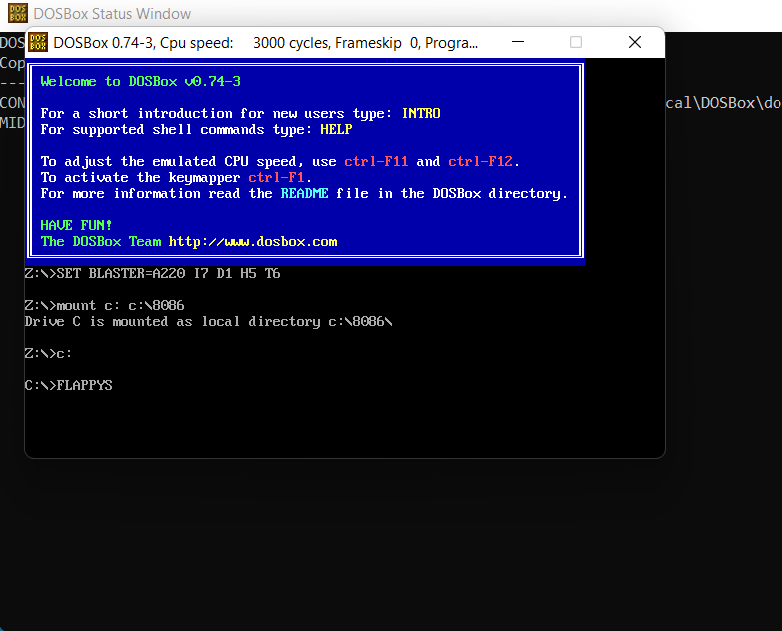
1. **Giới thiệu về trình giả lập DOSBox**

* DOSBox là một trình giả lập máy tính dựa trên DOS (Disk Operating System) được phát triển để chạy các ứng dụng và trò chơi DOS trên các hệ điều hành hiện đại. DOSBox được thiết kế để giúp người dùng tái tạo và trải nghiệm các ứng dụng DOS cổ điển và các trò chơi retro trên các hệ điều hành như Windows, macOS, Linux và nhiều nền tảng khác.



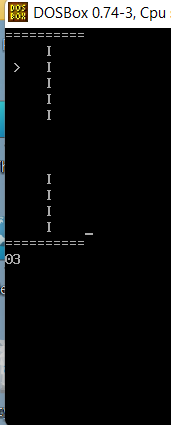
* Do hạn chế của trình biên dịch trên emu8086 không hỗ trợ một số các ngắt và một số mã ASM đặc biệt nên nhóm em đã sử dụng giả lập DOSBox là môi trường giả lập DOS chính xác nhất hỗ trợ đầy đủ các hàm ngắt và tính năng nhóm em sử dụng.
* Do DOSBox chỉ chạy các tệp tin có đuôi .COM và đuôi .EXE nên chúng ta phải chuyển đổi file ASM thành file COM phần mềm biên dịch mã lệnh như TASM(Turbo Assembler) để biên dịch mã lệnh Assembly thành tệp nhị phân (.OBJ) sau đó liên kết các file đó lại thành file COM và vào DOSBox mở file là ứng dụng sẽ được thực thi.





1. **GIAO DIỆN CHƯƠNG TRÌNH**

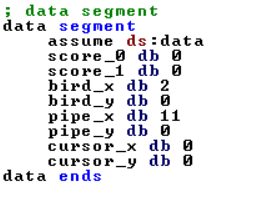
****

****

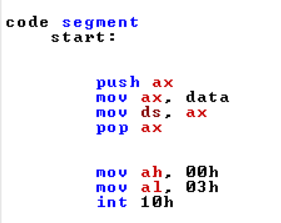
1. **MIÊU TẢ CHƯƠNG TRÌNH (GIẢI THÍCH MÃ NGUỒN)**

* Ý tưởng để xây dựng nên game này đó là xây dựng một vòng lặp xử lý liên tục các sự kiện khi đang diễn ra trò chơi như in ra nhân vật, in ra ống,…Khi sự kiện va chạm xảy ra (collision) thì sẽ xây dựng một nhãn game\_over cho vòng lặp của game kết thúc.

1. **Khởi tạo các biến**



* **Score\_0:** Điểm của game ở hàng đơn vị
* **Score\_1:** Điểm của game ở hàng chục
* **Bird\_x:** Vị trí của nhân vật theo trục hoành
* **Bird\_y:** Vị trí của nhân vật theo trục tung
* **Pipe\_x:** Vị trí của ống theo trục hoành
* **Pipe\_y:** Vị trí của ống theo trục tung
* **Cursor\_x, cursor\_y:** Tạo 2 biến con trỏ cho mục đích render
* Sử dụng hàm ngắt 10h đưa số hiệu 00h vào ax và 03h vào ah để thực hiện thiết lập chế độ hiển thị 80x25 (Độ phân giải của cửa sổ game) với 16 màu và 8 trang.

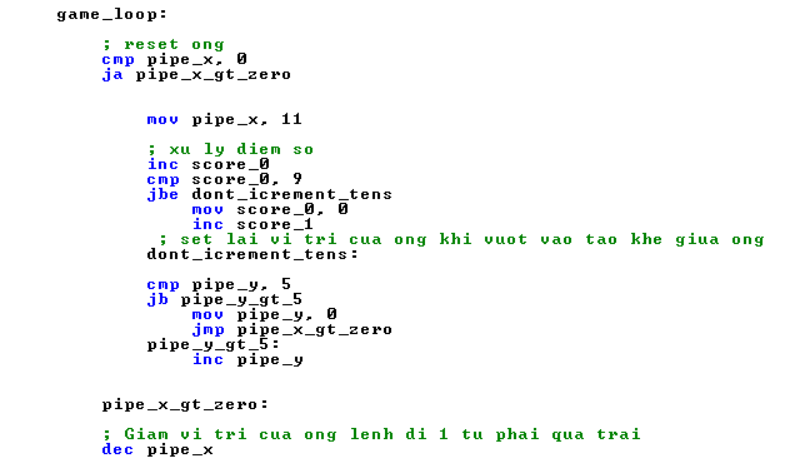


1. **Giải thích vòng lặp của game**

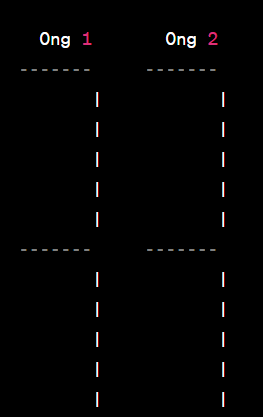
* Vòng lặp được gán nhãn **game\_loop** xử lý 2 công việc chính là: Xử lý logic của game và thực hiện viêc render (In ra các thành phần của game lên màn hình console).

1. **Xử lý logic game**

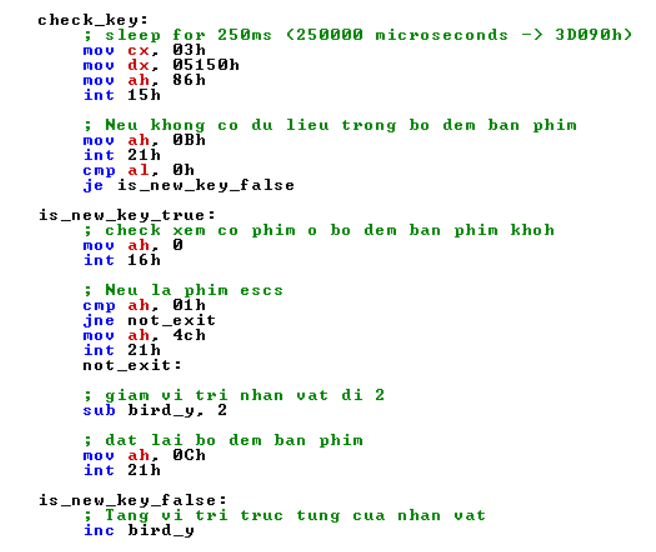
* Logic ống xuất hiện liên tục:



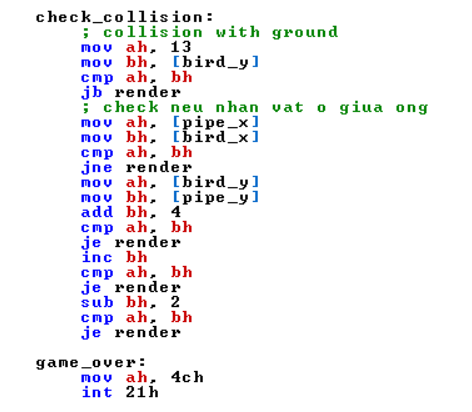
* Ban đầu *pipe\_x = 11* **(**Ở phía cuối cùng bên phải cửa sổ game**)**. Nếu *pipe\_x* > 0 nhảy đến nhãn *pipe\_x\_gt\_zero* giảm giá trị của ***pipe\_x*** đi 1 để ống di chuyển sáng trái. Nếu *pipe\_x* <= 0 sẽ đặt lại giá trị của *pipe\_x* = 11.
* **Cập nhật điểm số:** Mỗi khi chim bay qua một ống (tức là khi *pipe\_x* giảm về 0 và *pipe\_y* tăng lên 1), chúng ta sẽ tăng giá trị của *score\_0* lên 1. Điều này có nghĩa là mỗi khi chim vượt qua một ống, điểm số của người chơi sẽ tăng lên một đơn vị.
* Nếu *score\_0* > 9 set lại giá trị *score\_0* = 0 và tăng *score\_1*lên 1 để đếm điểm hàng chục.
* **Tạo khoảng hở giữa 2 ống:** Mỗi lần vòng lặp chạy *pipe\_y* được tăng lên 1 => ống sẽ di chuyển xuống dưới màn hình. Kiểm tra tiếp nếu *pipe\_y* > 5 (đã di chuyển qua 5 hàng từ trên màn hình xuống) thì gán lại *pipe\_y* = 0 đồng thời giảm *pipe\_x* đi 1 => Ống xuất hiện lại từ bên phải màn hình, tạo ra một cặp ống mới với khoảng hở giữa chúng.



* Xử lý sự kiện nhấn phím



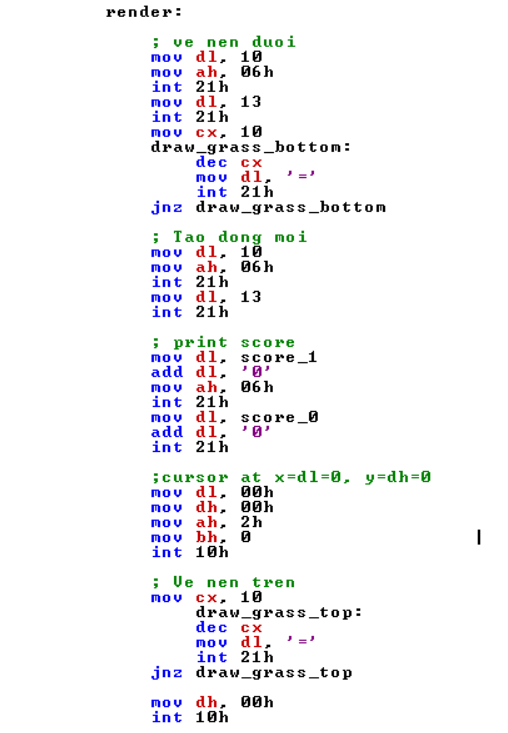
* Sử dụng hàm ngắt 15h của DOS để thực hiện khoảng thời gian chờ giữa 2 lần nhấn phím và giảm tốc độ giữa những lần cập nhật màn hình.
* *Mov cx, 03h* đưa giá trị 3 và thanh ghi CX tức là mỗi lần bộ đệm bàn phím cập nhật phím mới là 3ms = 3s. 015015h = 250ms tưc là mỗi lần nhấn phím thì màn hình game sẽ cập nhật trễ đi 250ms = 2.5s để tốc độ game có thể phù hợp với con người có thể theo dõi.
* Nếu có dữ liệu trong bộ đệm bàn phím nhảy sẽ nhảy đến nhãn *is\_new\_key\_true*. Kiểm tra xem phím bấm có phải phím *Esc* hay không. Nếu là phím *Esc* game sẽ dừng lại còn không sẽ nhảy đến nhãn *not\_exit*: giảm vị trí của *bird\_y* đi 2 đơn vị nhân vật bird sẽ bay lên còn không có phím nào được nhấn nhảy đến nhãn *is\_new\_key\_false* tăng tọa độ của nhân vật lên 1 đơn vị nhân vật sẽ rơi xuống.
* Xử lý sự kiện va chạm (collision):



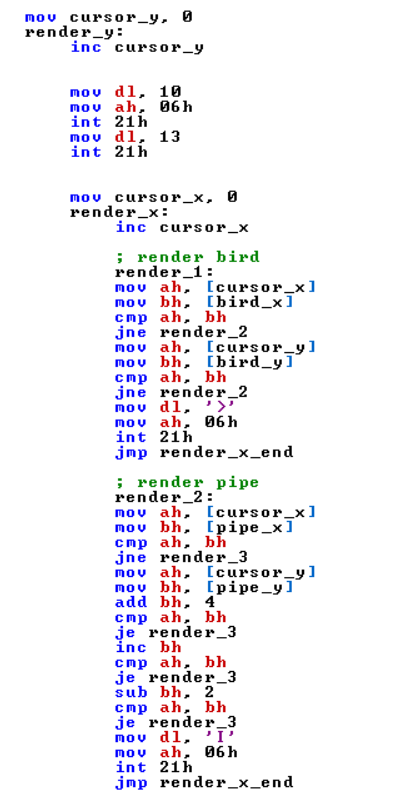
* + - Kiểm tra các trường hợp nếu hợp lệ tiếp tục in các thành phần của game lên màn hình còn không thì sẽ đến nhãn *game\_over*.
    - Nếu *bird\_y < 13* => Nhân vật vẫn chưa rơi xuống tận cùng => Nhảy đến nhãn *render* tiếp tục in các thành phần khác lên màn hình.
    - Nếu *bird\_x* != *pipe\_x* => Tiếp tục render.
    - Nếu *bird\_x = pipe\_x* (Nhân vật ở cùng vị trí với ống nước) => so sánh *bird\_y* với *pipe\_y + 4* (Khoảng trống giữa 2 ống). Nếu *bird\_y* = *pipe\_y + 4* => Nhân vật đang vượt qua ống => Tiếp tục render.
    - Vẫn trong trường hợp *bird\_x = pipe\_x* nếu *bird\_y != pipe\_y + 4* so sánh *bird\_y* với *pipe\_y + 4 – 2* (Vị trí nhân vật sát với ống vì bird\_x = 2) => Nếu bằng nhau thì tiếp tục gọi hàm render.
    - Còn tất cả các trường hợp khác là người chơi xảy ra va chạm nhảy đến nhãn game\_over kết thúc trò chơi

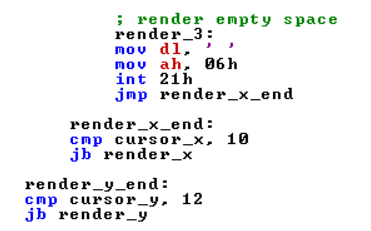
1. **Xử lý công việc render (In các thành phần game lên màn hình)**

* Render phần nền dưới, nền trên, điểm

****

* Đặt giá trị 10 vào thanh ghi DL và 06h vào thanh ghi AH. Lệnh int 21h được gọi để gọi hàm để chuyển đến ký tự xuống dòng (line feed).
* Tiếp theo, lệnh mov dl, 13 và int 21h được sử dụng để chuyển đến ký tự trở về đầu dòng. Trong mỗi lần lặp, lệnh dec cx giảm giá trị của thanh ghi CX đi 1 đơn vị.
* Lệnh mov dl, '=' đặt ký tự "=" vào thanh ghi DL. Cuối cùng, lệnh int 21h được gọi in ra màn hình console 10 dấu “=”.
* Xuống dòng để render điểm của người chơi.
* Di chuyển con trỏ lên đầu dòng thực hiện công việc render nền trên tương tự
* Render những thành phần còn lại





* Sử dùng 2 vòng lặp lồng nhau để render những thành phần còn lại của trò chơi bao gồm nhân vật, ống và khoảng cách giữa 2 ống.
* Đầu tiên, ta khởi tạo biến cursor\_y với giá trị 0, đại diện cho vị trí hiện tại của con trỏ theo trục y (dọc) trên màn hình console.
* Sau đó, ta bắt đầu một vòng lặp render\_y, đưa con trỏ xuống dòng và lùi về đầu dòng. Sau khi di chuyển con trỏ đến đầu dòng mới, ta bắt đầu một vòng lặp render\_x để điều khiển việc vẽ các đối tượng trên dòng hiện tại.
* Tăng biến cursor\_x để trỏ tới cột mới và kiểm tra xem ở vị trí 2 con trỏ x và y có cùng vị trí với nhân vật hay vị trí ống hay không. Nếu có ở vị trí nhân vật thì in dấu “ >”, nếu có ở vị trí ống thì in “I”, còn nếu không phải 2 ví trí ống và nhân vật thì in khoảng trắng.
* Đặt thêm nhãn để kế thúc 2 vòng lặp khi cursor\_x > 10 và cursor\_y > 12.

1. **TÀI LIỆU THAM KHẢO**

* Bảng tra mã ngắt: <http://www.oldlinux.org/Linux.old/docs/interrupts/int-html/int.htm>
* Setup DOSBox: YouTube
* Ngắt và xử lý ngắt 8086: Bài Giảng Kỹ Thuật Vi Xử Lý – Phạm Hoàng Duy
* Code and Tutorials: Youtuber Pirxie <https://www.youtube.com/@pirxie1127>