

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

**KHOA CÔNG NGHỆ THÔNG TIN**

**----------------------------------------**

****

**BÀI TẬP LỚN**

**KIẾN TRÚC MÁY TÍNH**

|  |  |
| --- | --- |
| Giảng viên | : Trần Tiến Công |
| Họ và tên | : Đặng Hà Vy |
| Mã sinh viên | : B23DCCN951 |
| Nhóm  Lớp | : 12  : N09 |

**Hà Nội – 2025**

,,,,,

**MỤC LỤC**

[PHẦN 1: BÀI TẬP CÁ NHÂN 3](#_Toc198227200)

[I. Bài số 1 3](#_Toc198227201)

[1. Bài 4: Viết chương trình hợp ngữ Assembly cho phép nhập 1 ký tự viết thường và in ra màn hình chữ hoa của ký tự đó. 3](#_Toc198227202)

[2. Bài số 7: Viết chương trình hợp ngữ Assembly chuyển một số từ hệ cơ số 10 sang hệ nhị phân. 4](#_Toc198227203)

[3. Bài số 13:  Viết chương trình hợp ngữ Assembly cho phép nhập vào các số và in ra màn hình tổng của các số đó. 6](#_Toc198227204)

[II. Bài số 2: 9](#_Toc198227205)

[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.) 9](#_Toc198227206)

[2. Dùng công cụ Debug khảo sát nội dung các thanh ghi IP, DS, ES, SS, CS, BP, SP. 11](#_Toc198227207)

[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. 11](#_Toc198227208)

[PHẦN 2: BÀI TẬP NHÓM 13](#_Toc198227209)

[I. Giới Thiệu Đề Tài 13](#_Toc198227210)

[II. Nội Dung Chính Của Đề Tài 14](#_Toc198227211)

[III. Miêu Tả Chương Trình 16](#_Toc198227212)

[IV. Miêu Tả Giao Diện Chương Trình 33](#_Toc198227213)

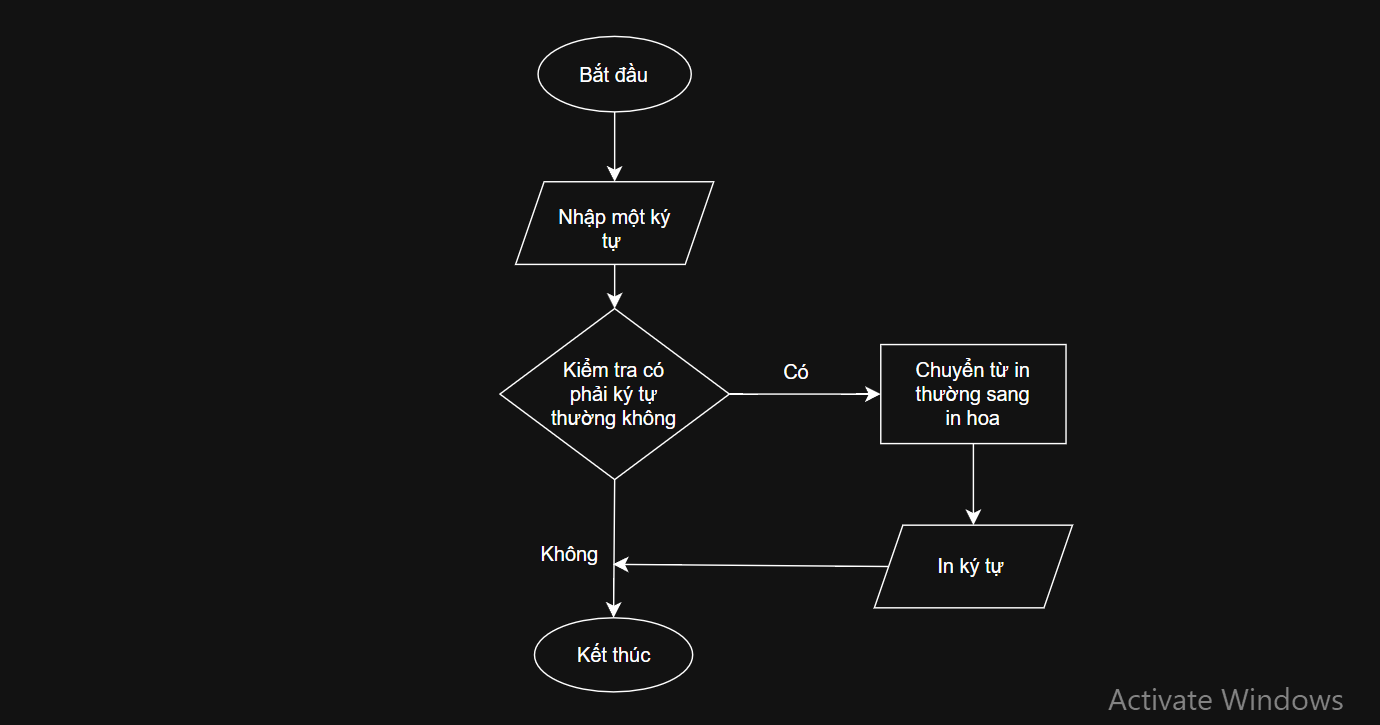
[V. Tài Liệu Tham Khảo 37](#_Toc198227214)

# **PHẦN 1: BÀI TẬP CÁ NHÂN**

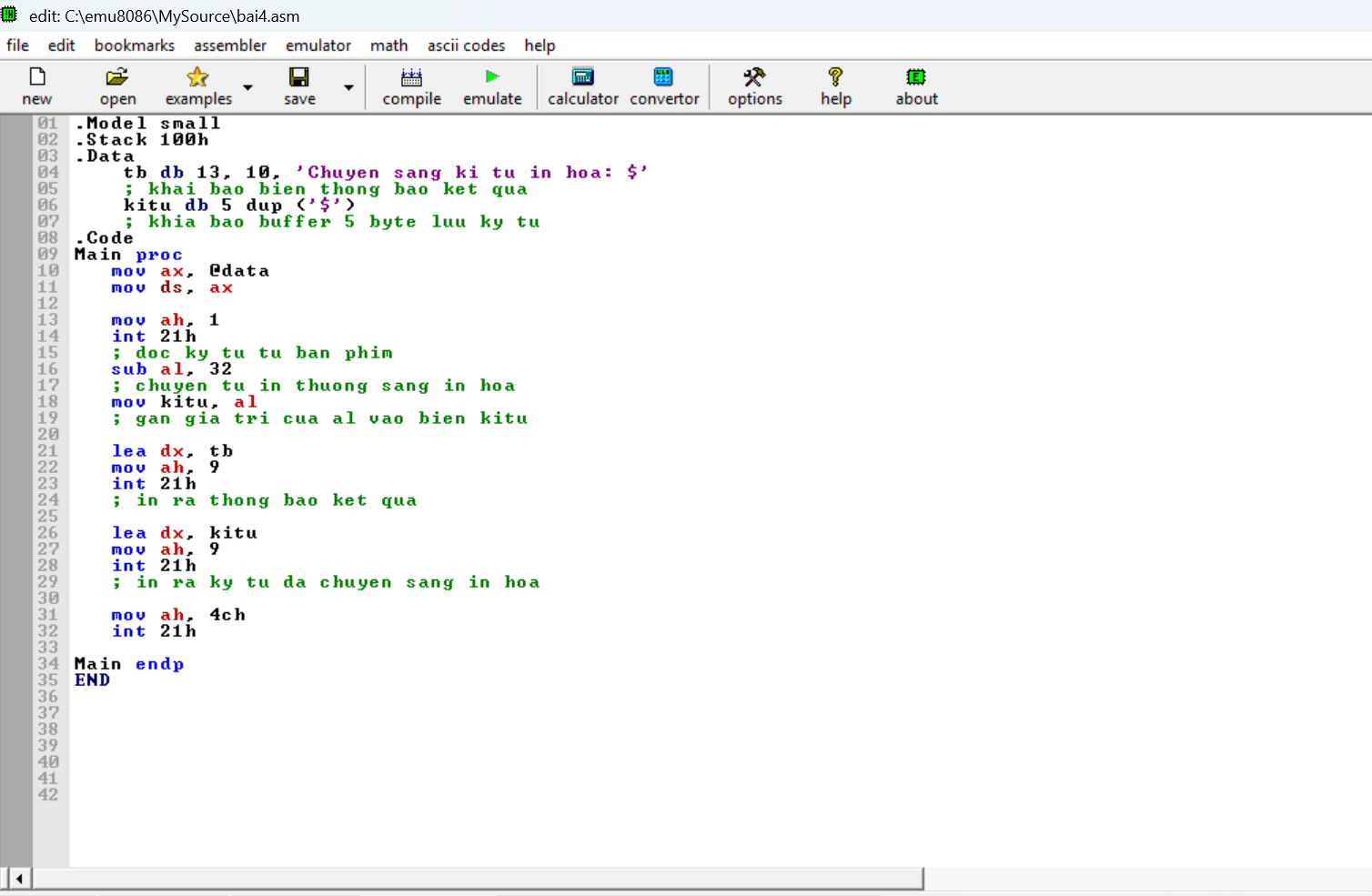
## **I. Bài số 1**

1. Bài 4: Viết chương trình hợp ngữ Assembly cho phép nhập 1 ký tự viết thường và in ra màn hình chữ hoa của ký tự đó.

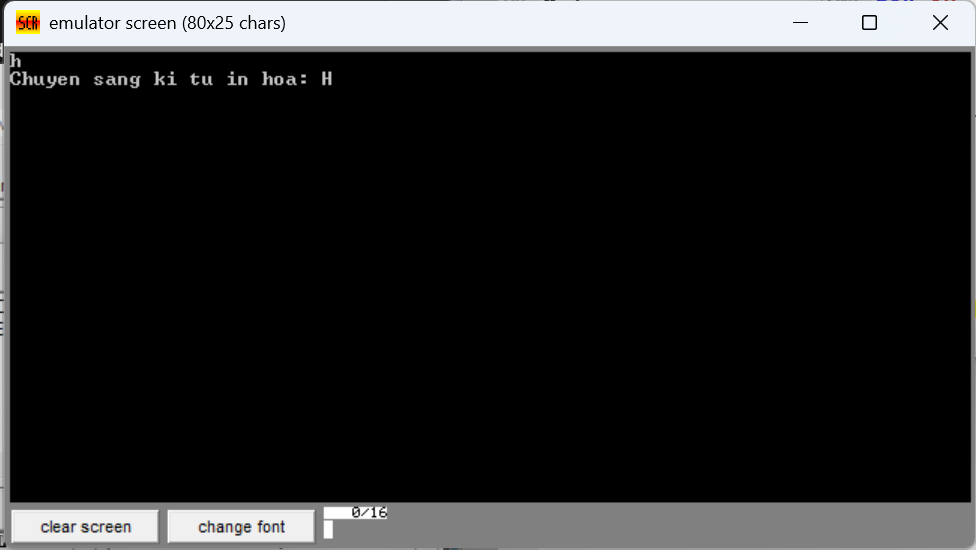
* Lưu đồ thuật toán



* Mã nguồn emu8086

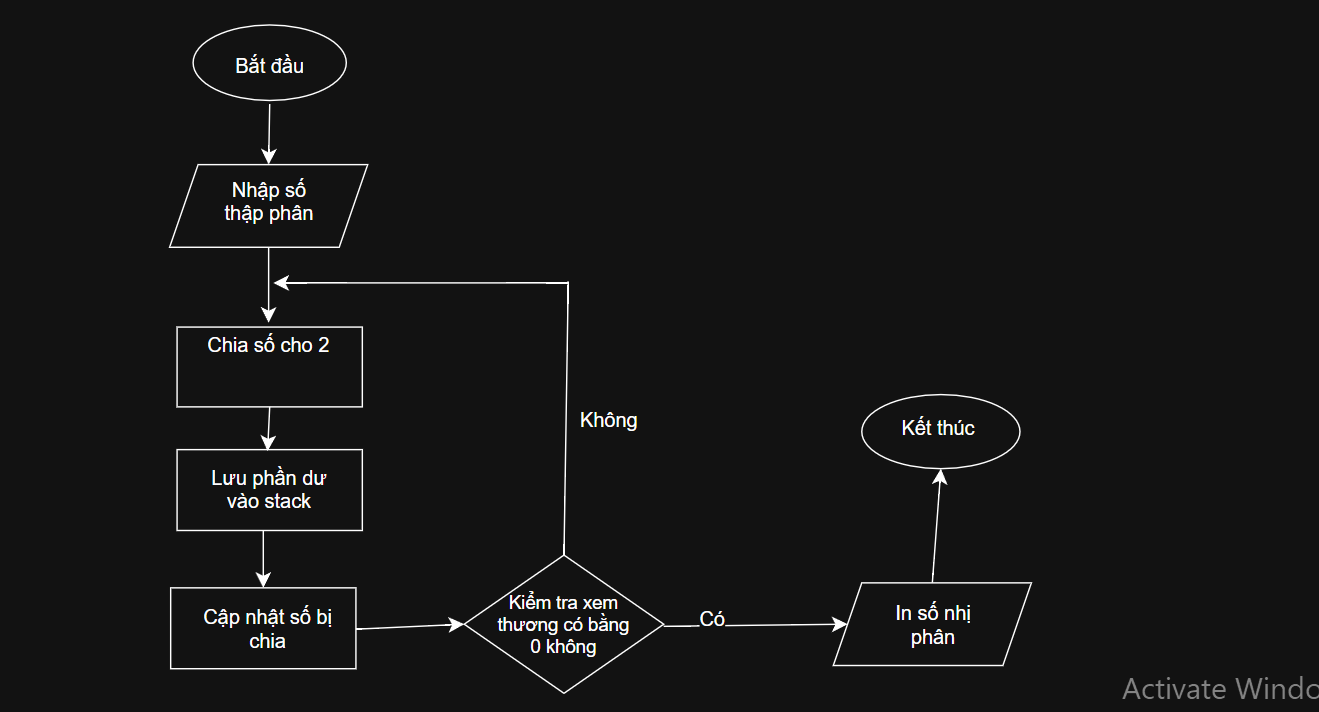


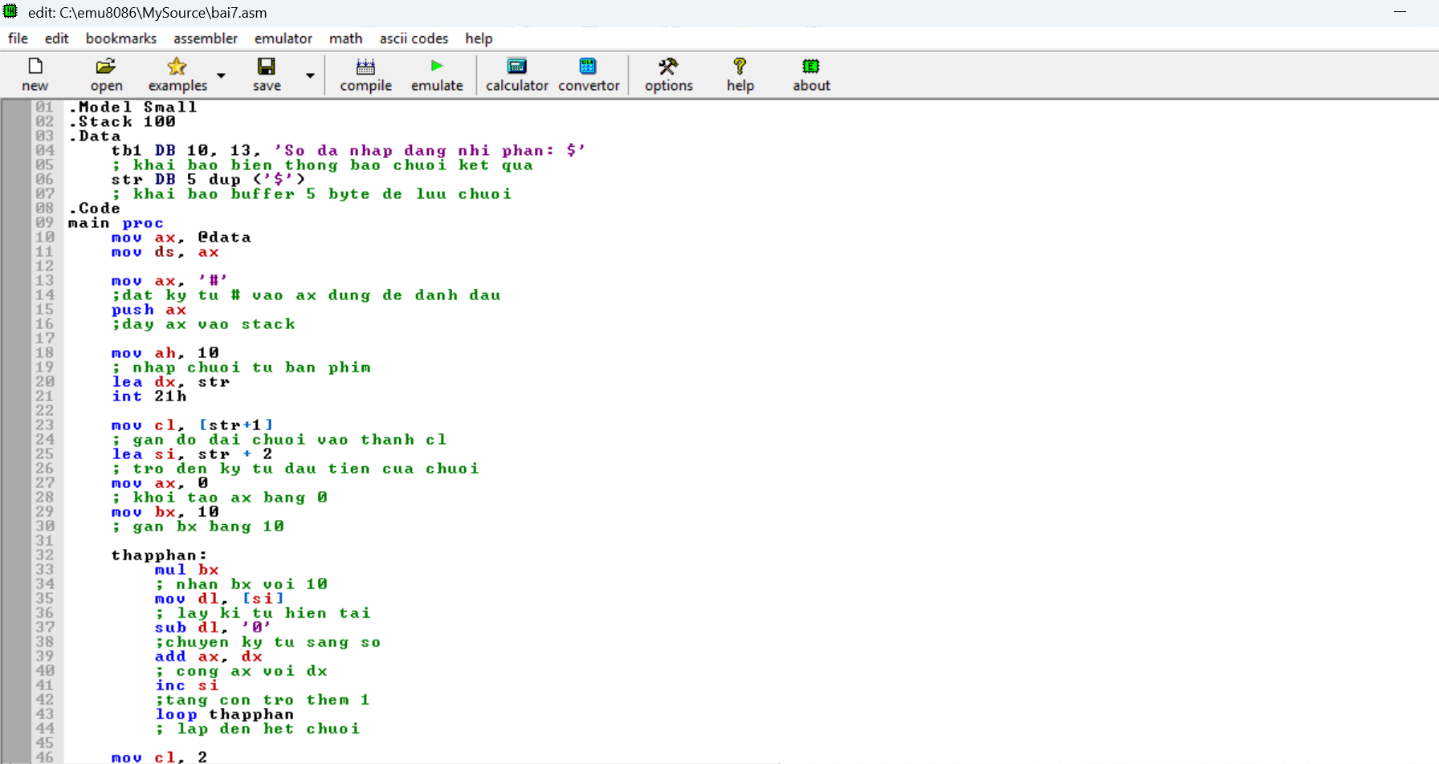
* Giao diện hiển thi

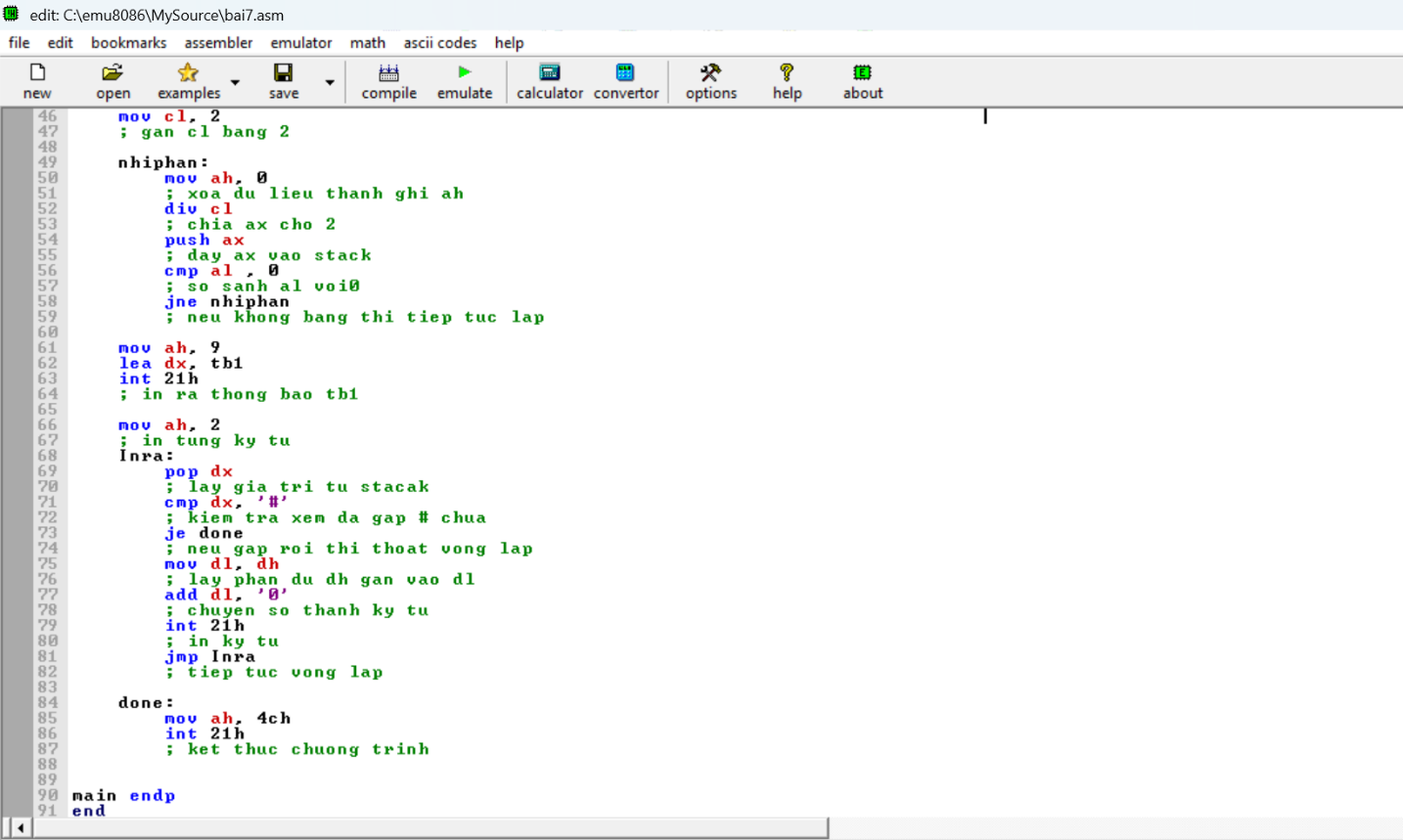


1. Bài số 7: Viết chương trình hợp ngữ Assembly chuyển một số từ hệ cơ số 10 sang hệ nhị phân.

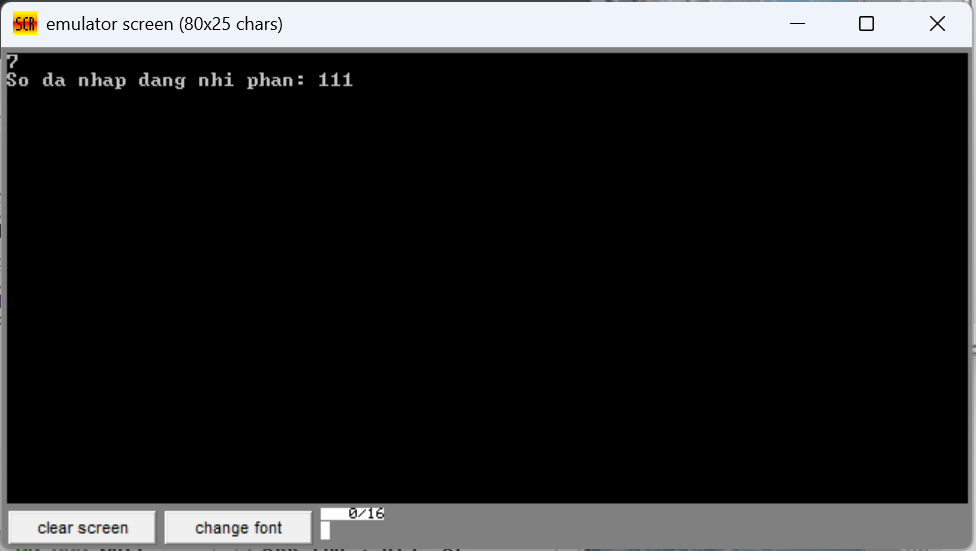
* Lưu đồ thuật toán



* Mã nguồn emu8086

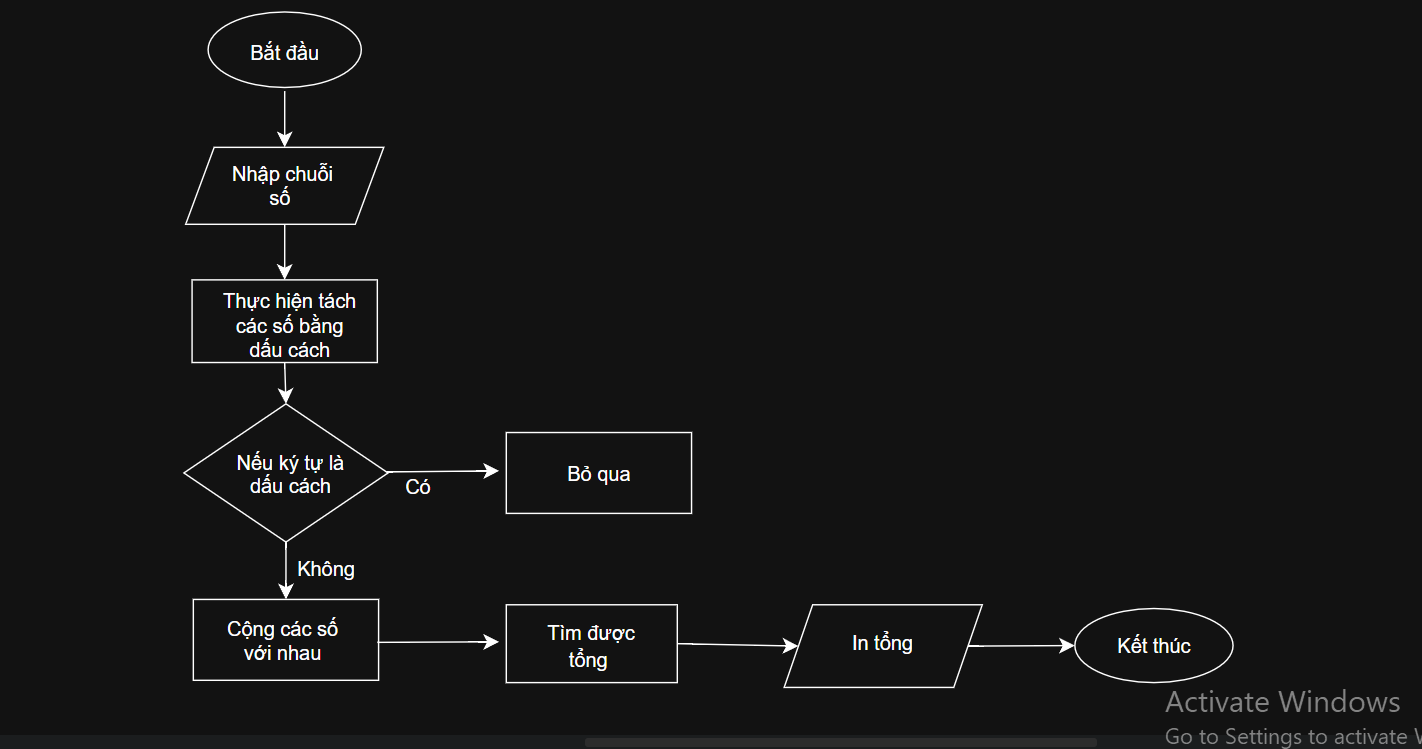


* Giao diện hiển thị

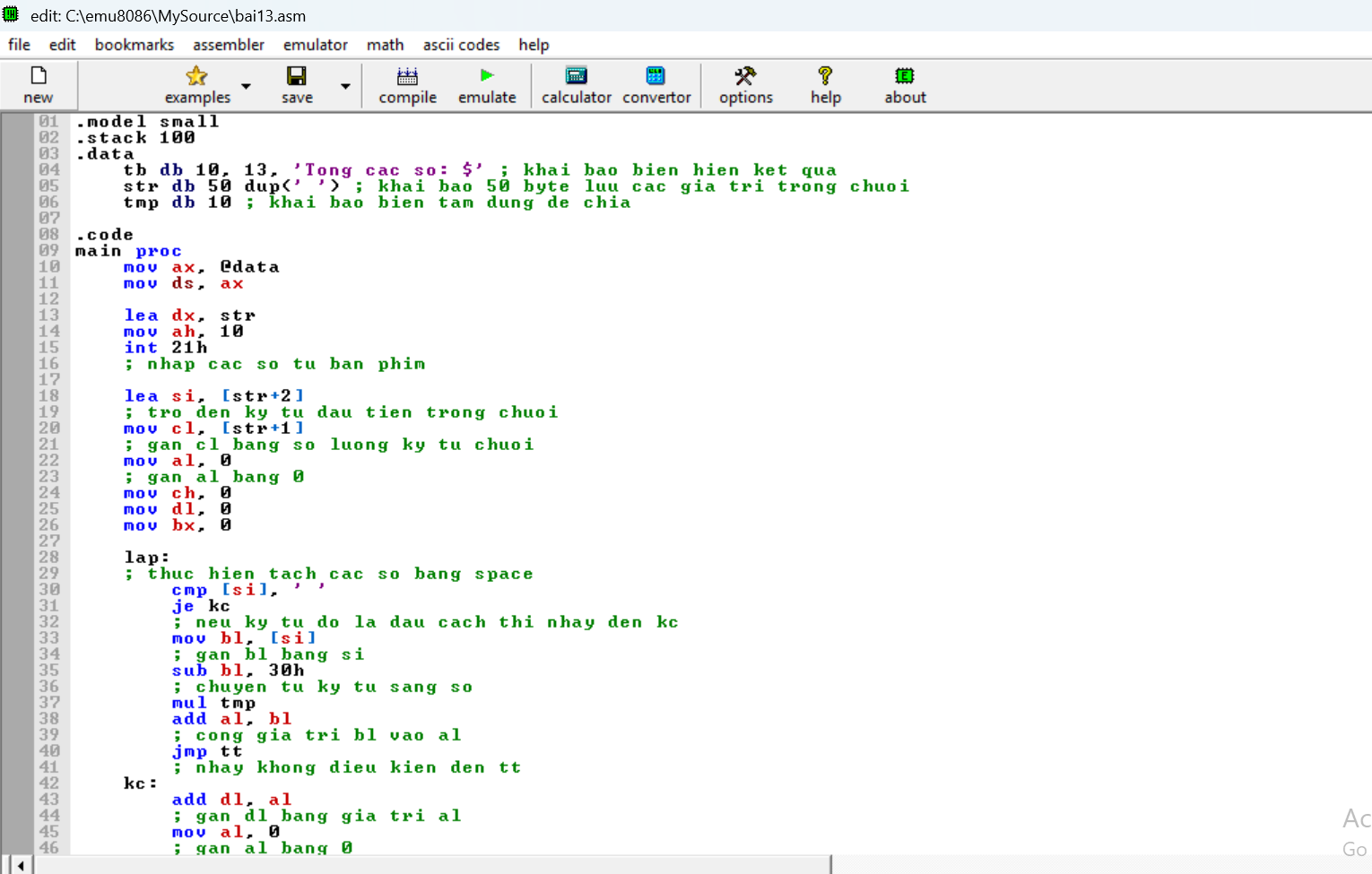


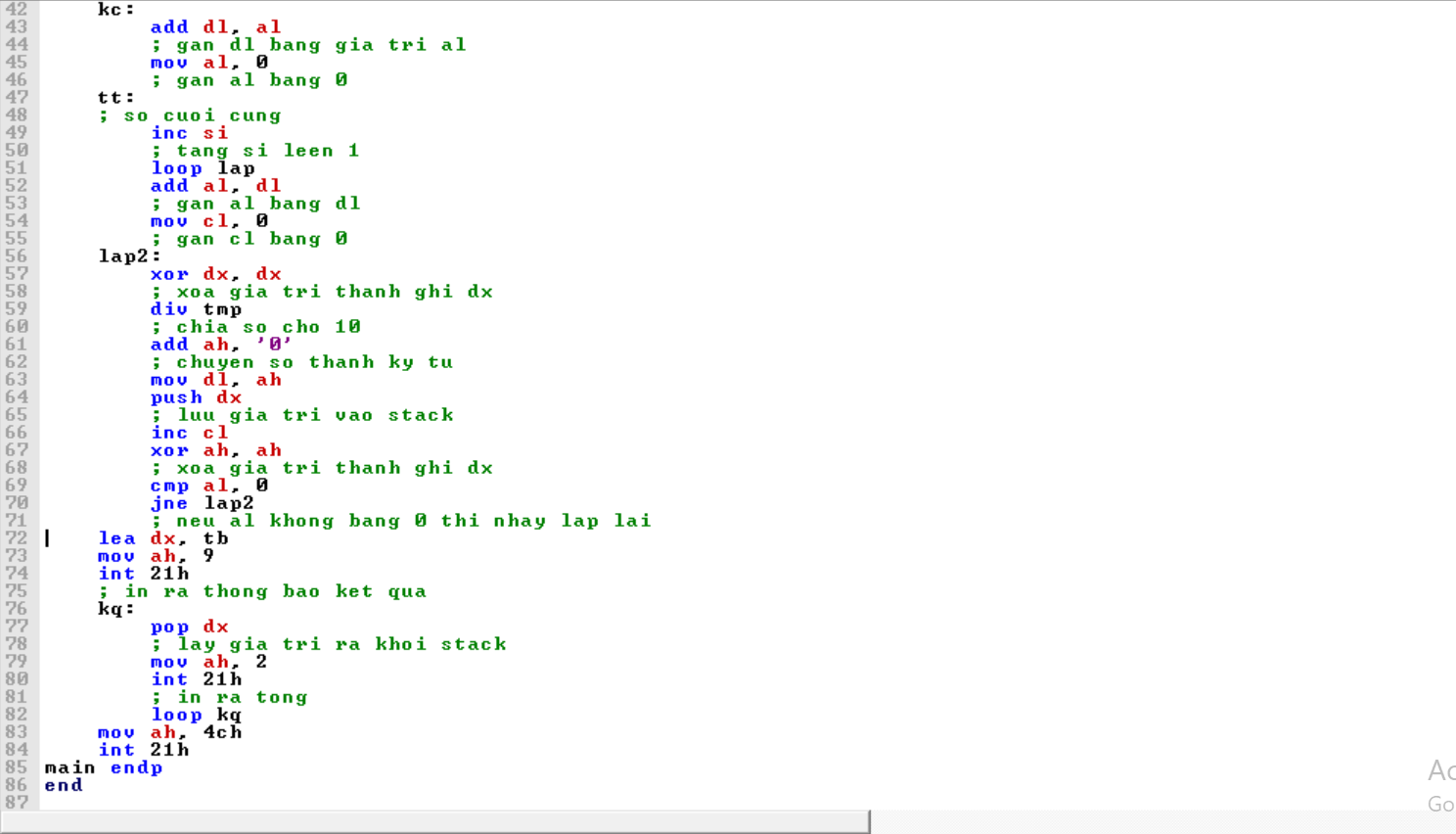
1. Bài số 13:  Viết chương trình hợp ngữ Assembly cho phép nhập vào các số và in ra màn hình tổng của các số đó.

* Lưu đồ thuật toán

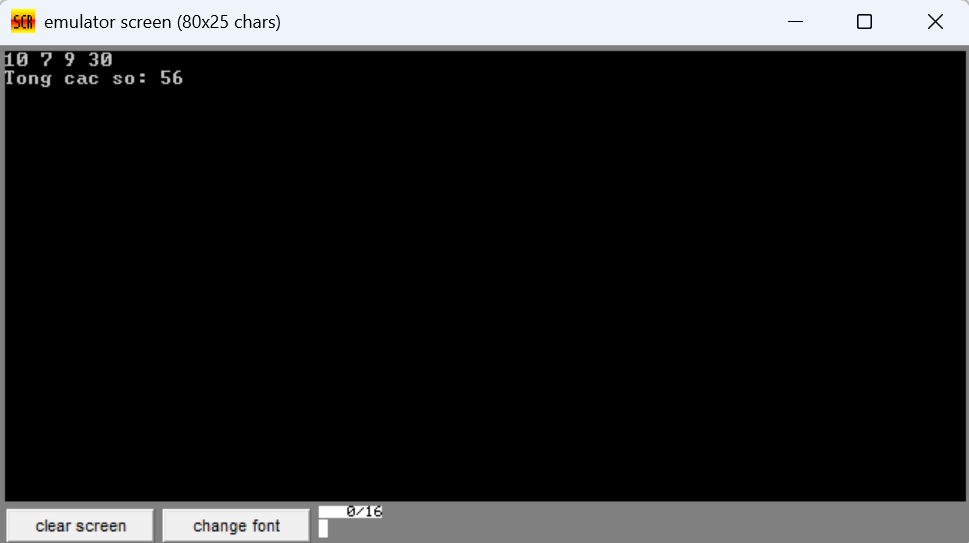


* Mã nguồn emu8086





* Giao diện hiển thị



## **II. Bài số 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.)

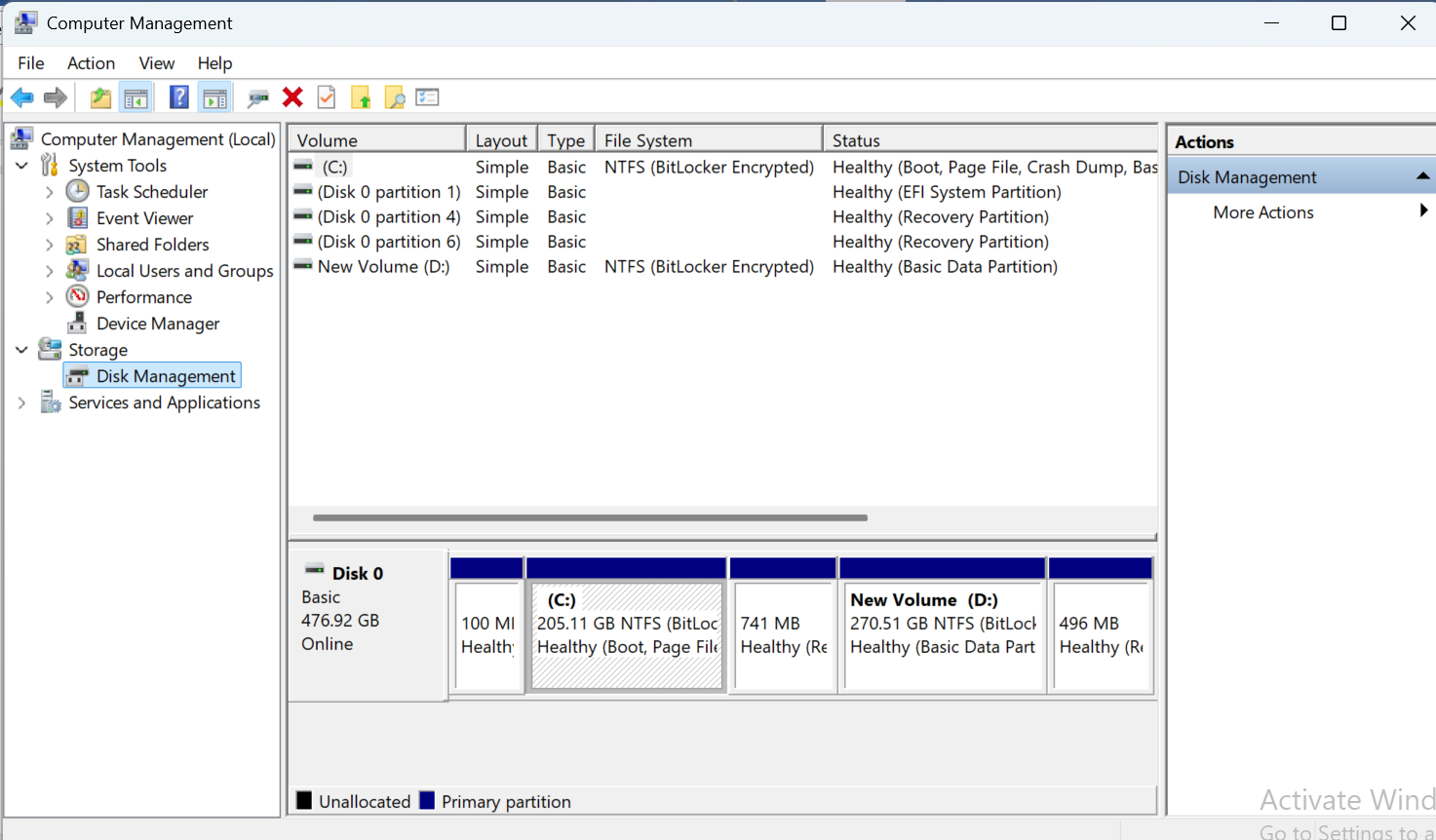
Sử dụng phần mềm CPU-Z 64-bit v1.85.0x64:



* CPU: AMD Ryzen 7 4800H
* Cache:
  + L1Data: 8 \* 32KB; 8-way
  + L1Inst.: 8 \* 32KB; 8-way
  + L2: 8 \* 512 KB; 8-way
  + L3: 8 MB; 16-way



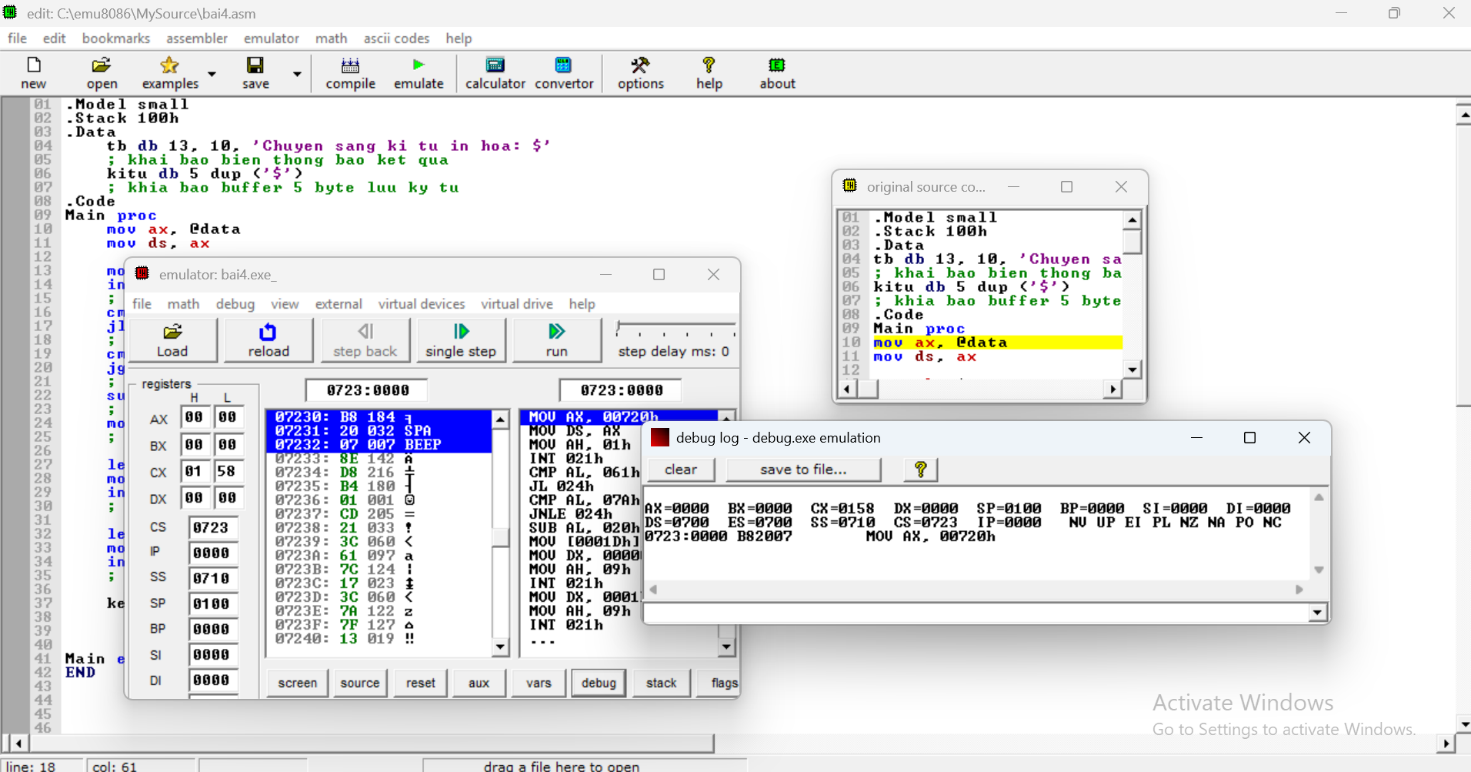
* ROM/RAM: DDR4; 16GB



* Bộ nhớ ngoài: 1 ổ SSD 512 GB
* Thiết bị vào ra:
  + Loa, mic
  + Bluetooth
  + Mạng
  + Camera
  + Bàn phím, chuột
  + Máy in
  + Các cổng kết nối khác…

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

**Khảo sát câu 4:** Viết chương trình hợp ngữ Assembly cho phép nhập 1 ký tự viết thườngvà in ra màn hình chữ hoa của ký tự đó



1. 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.

* **CS (Code Segment):** Giá trị 0723h, không đổi. CS chứa địa chỉ bắt đầu của đoạn mã chương trình, nơi chứa các lệnh cần thực hiện.
* **IP (Instruction Pointer):** Thay đổi từ 0000h đến 0003h. IP trỏ đến địa chỉ của lệnh tiếp theo cần thực thi, giúp CPU biết lệnh nào sẽ được xử lý kế tiếp.
* **DS (Data Segment):** Giá trị 0700h, không đổi. DS chứa địa chỉ bắt đầu của vùng dữ liệu, nơi lưu trữ các chuỗi và biến do người lập trình khai báo.
* **ES (Extra Segment):** Giá trị 0700h, không đổi. ES dùng để hỗ trợ thao tác với vùng dữ liệu mở rộng.
* **SS (Stack Segment):** Giá trị 0710h, không đổi. SS chứa địa chỉ bắt đầu của vùng ngăn xếp (stack), dùng để lưu tạm các giá trị trong quá trình thực thi như địa chỉ trả về từ hàm.
* **SP (Stack Pointer):** Giá trị 0100h, không đổi. SP là con trỏ đến đỉnh ngăn xếp hiện tại, xác định vị trí tiếp theo để PUSH hoặc POP dữ liệu.
* **BP (Base Pointer):** Giá trị 0000h, không đổi. BP được sử dụng trong các thao tác truy cập vào stack.
* **SI (Source Index):** Giá trị 0000h, không đổi. SI thường dùng làm chỉ số nguồn trong các thao tác truyền dữ liệu hoặc xử lý chuỗi.

Hệ thống sử dụng **mô hình phân đoạn bộ nhớ**, trong đó địa chỉ vật lý được tính theo công thức:

* Địa chỉ vật lý = Segment × 16 + Offset.

Cách quản lý này giúp chia bộ nhớ thành các khu vực riêng biệt như mã lệnh, dữ liệu và ngăn xếp, giúp việc truy cập và bảo vệ bộ nhớ rõ ràng và hiệu quả hơn.

# **PHẦN 2: BÀI TẬP NHÓM**

## **I. Giới Thiệu Đề Tài**

Rapid Ball là một trò chơi mô phỏng đơn giản, lấy cảm hứng từ trò chơi cổ điển Pong – một trong những trò chơi điện tử đầu tiên ra đời từ những năm 1970. Với lối chơi dễ hiểu nhưng đầy thử thách, Rapid Ball trở thành một ví dụ điển hình trong việc rèn luyện lập trình hợp ngữ, đồng thời mang lại cảm giác giải trí thú vị.

Trong trò chơi Rapid Ball, người chơi sẽ điều khiển một thanh chắn (paddle) nằm ở đáy màn hình để đỡ quả bóng đang rơi từ trên xuống. Mỗi lần bóng được đỡ thành công, người chơi sẽ ghi được điểm; nếu không đỡ được, người chơi sẽ mất một mạng. Trò chơi kết thúc khi người chơi mất hết tất cả các mạng. Với cơ chế phản xạ và ghi điểm đơn giản, trò chơi mang lại cảm giác hồi hộp và hấp dẫn.

Emu8086 là một trình mô phỏng vi xử lý Intel 8086 mạnh mẽ, cho phép lập trình, biên dịch và chạy trực tiếp mã Assembly trong môi trường DOS mô phỏng. Đây là công cụ lý tưởng để học tập và kiểm chứng các nguyên lý của kiến trúc máy tính và hệ điều hành thời kỳ đầu. Đề tài “Lập trình game Rapid Ball trên Emu8086” tập trung vào phát triển một trò chơi nhỏ nhưng đầy đủ chức năng, hoàn toàn bằng ngôn ngữ Assembly, giúp sinh viên thực hành tư duy thuật toán và xử lý tương tác phần cứng.

Với việc chọn đề tài này, chúng em mong muốn:

* Rèn luyện tư duy lập trình hợp ngữ: Việc xây dựng một trò chơi hoàn chỉnh bằng Assembly giúp sinh viên làm quen với cách thao tác với bộ nhớ, hiển thị ký tự, điều khiển phím và xử lý logic.
* Làm chủ các ngắt hệ thống: Trò chơi tận dụng các ngắt BIOS như INT 10h để hiển thị và INT 16h để đọc phím. Đây là kiến thức quan trọng để làm việc ở tầng thấp của hệ điều hành.
* Vận dụng thuật toán điều khiển đơn giản: Chương trình cần xử lý va chạm giữa bóng và tường hoặc paddle, cập nhật điểm và trạng thái trò chơi. Điều này đòi hỏi khả năng thiết kế thuật toán trong môi trường không có hàm thư viện hỗ trợ.
* Tái hiện trải nghiệm trò chơi cổ điển: Trò chơi được viết với giao diện đơn sắc, theo phong cách cổ điển trên màn hình văn bản 80x25 ký tự. Điều này không chỉ mang tính hoài niệm mà còn phù hợp với khả năng hiển thị của môi trường Emu8086.

Dưới đây, chúng em xin được trình bày báo cáo chi tiết cho đề tài “Lập trình trò chơi Rapid Ball trên Emu8086” – sản phẩm do nhóm thực hiện bằng hợp ngữ x86. Chúng em kính mong nhận được sự góp ý và đánh giá từ thầy.

## **II. Nội Dung Chính Của Đề Tài**

1. Tổng quan về trò chơi

Trò chơi **Rapid Ball** là một trò chơi điện tử đơn giản thuộc thể loại bắn bóng (brick breaker). Người chơi điều khiển một thanh chắn (paddle) ở đáy màn hình để phản xạ quả bóng bay. Mỗi lần bóng chạm thanh chắn hoặc đập vào các vật thể mục tiêu sẽ giúp người chơi ghi điểm, và mất một mạng nếu bóng rơi ra khỏi đáy màn hình. Mục tiêu của trò chơi là ghi được càng nhiều điểm càng tốt trước khi hết mạng.

Trò chơi được điều khiển bằng bàn phím như sau:

* **←** và **→**: Di chuyển thanh chắn sang trái hoặc sang phải.
* **SPACE**: Phát bóng để bắt đầu ván chơi hoặc tiếp tục sau khi tạm dừng.
* **ESC**: Tạm dừng trò chơi hoặc thoát khỏi chương trình.

Chương trình gồm các chức năng chính sau:

* Khởi tạo màn hình trò chơi, vẽ thanh chắn và bóng ở vị trí ban đầu.
* Đọc phím bấm và di chuyển thanh chắn tương ứng, đảm bảo thanh chắn không đi ra khỏi biên giới màn hình.
* Tính toán và cập nhật vị trí của quả bóng mỗi chu kỳ. Bóng có tọa độ (x, y) và vận tốc (dx, dy) được cộng dồn vào tọa độ để di chuyển.
* Kiểm tra va chạm của bóng với tường trái, phải và trần trên; nếu va chạm thì đổi chiều vận tốc tương ứng (bóng bật ngược lại). Nếu bóng va chạm với thanh chắn, bóng cũng đổi chiều và người chơi được cộng thêm điểm.
* Giảm số mạng khi bóng vượt qua đáy màn hình (không đập vào thanh chắn). Khi số mạng về 0, trò chơi kết thúc và hiển thị thông báo **Game Over**.
* Cập nhật và hiển thị điểm số, số mạng còn lại lên màn hình sau mỗi lần bóng di chuyển hoặc khi có sự kiện va chạm.

Đặc điểm kỹ thuật của trò chơi:

* Giao diện đồ họa 2D đơn giản, sử dụng chế độ đồ họa trên EMU8086 (ví dụ VGA 320×200).
* Sử dụng thao tác trực tiếp với bộ nhớ video (segment A000h) để vẽ quả bóng, thanh chắn và điểm số.
* Chương trình viết bằng hợp ngữ 8086 thuần túy, không sử dụng hàm API bên ngoài. Mã nguồn được chia thành các thủ tục (procedures) rõ ràng để xử lý từng chức năng.
* Tối ưu hóa thao tác với bộ nhớ nhằm đảm bảo cập nhật màn hình liên tục mỗi khung hình (frame) cho chuyển động mượt mà.

2. Giải thuật chính và cài đặt

Chương trình được tổ chức theo cấu trúc chính gồm phần khởi tạo và vòng lặp game liên tục cho đến khi kết thúc. Các bước chính bao gồm:

1. **Khởi tạo**: Thiết lập chế độ đồ họa thích hợp, khởi tạo giá trị ban đầu cho thanh chắn (ở giữa đáy màn hình) và quả bóng (thường ở phía trên thanh chắn). Biến lưu trữ tọa độ và vận tốc (dx, dy) của quả bóng được gán giá trị ban đầu.
2. **Vòng lặp game**: Lặp lại các bước sau cho đến khi số mạng còn lại bằng 0:
   * Đọc phím bấm của người chơi: Nếu nhấn **←** hoặc **→**, cập nhật vị trí thanh chắn tương ứng (cộng hoặc trừ tọa độ x). Đảm bảo thanh chắn không di chuyển vượt ra ngoài biên giới màn hình.
   * Cập nhật vị trí bóng: Mỗi chu kỳ, hoành độ và tung độ của bóng được cập nhật bằng cách cộng thêm giá trị dx và dy.
   * Kiểm tra va chạm:
     + Nếu bóng chạm **tường trái/phải** (tọa độ x đạt giới hạn min/max), đổi dấu dx để bóng bật ngược chiều ngang.
     + Nếu bóng chạm **trần trên** (tọa độ y đạt giá trị giới hạn trên), đổi dấu dy để bóng bật ngược chiều dọc.
     + Nếu bóng chạm **thanh chắn** (tọa độ y của bóng trùng hoặc vượt đến vị trí thanh chắn), đổi dấu dy và cộng thêm điểm cho người chơi.
     + Nếu bóng vượt **đáy màn hình** (tung độ y vượt dưới thanh chắn), giảm số mạng của người chơi. Nếu mạng còn 0, thoát vòng lặp để kết thúc.
   * Cập nhật giao diện: Xóa vẽ cũ và vẽ lại quả bóng, thanh chắn tại vị trí mới. Cập nhật hiển thị điểm số và số mạng.
3. **Kết thúc**: Khi vòng lặp game kết thúc (hết mạng), hiển thị thông báo **Game Over** và dừng chương trình hoặc chờ nhấn phím thoát.

Các tham số như tọa độ và vận tốc của bóng, vị trí thanh chắn được lưu trữ trong biến hoặc thanh ghi phù hợp. Mỗi lần cập nhật, chương trình so sánh tọa độ với các giới hạn màn hình để xử lý va chạm. Việc vẽ đồ họa được thực hiện bằng cách ghi trực tiếp các giá trị vào bộ nhớ video, đảm bảo giao diện được cập nhật theo thời gian thực cho người chơi.

## **III. Miêu Tả Chương Trình**

1. Lưu đồ thuật toán

**A diagram of a game

AI-generated content may be incorrect.**

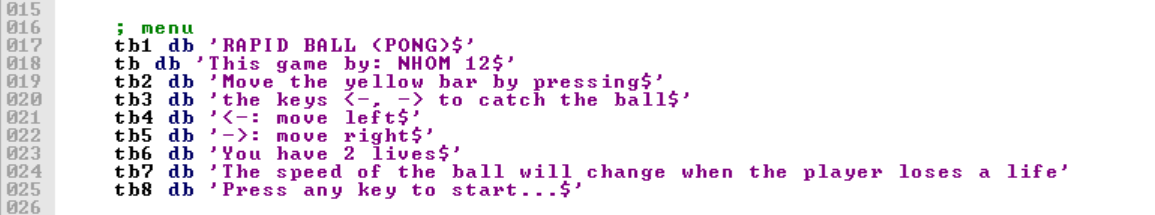
**Dữ liệu đầu vào:** Người chơi nhấn phím ←, → để di chuyển thanh đỡ; phím SPACE để tạm dừng/tiếp tục; phím ESC để thoát chương trình.

**Dữ liệu đầu ra:** Màn hình hiển thị quả bóng di chuyển, thanh đỡ, số mạng (Lives), điểm số (Score) và các thông báo “Press SPACE to start”, “PAUSED”, “GAME OVER”

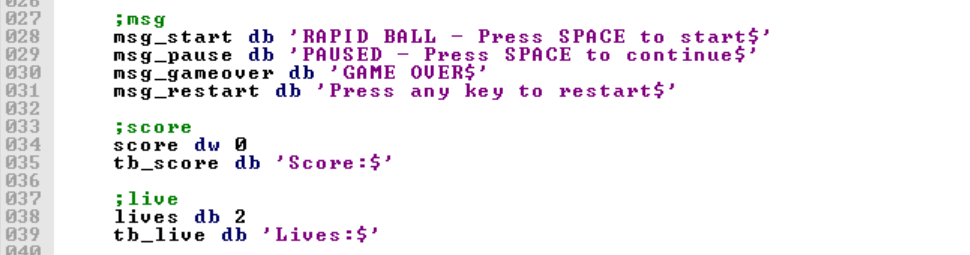
2. Phân tích chương trình: <https://ideone.com/zMDsfb>



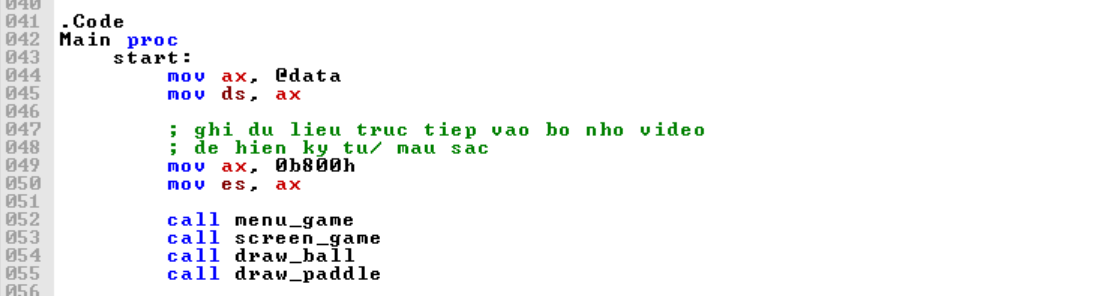
* **.Model small**: Chọn mô hình bộ nhớ “small”
* **.Stack 100h**: Cấp 0x100 (256) byte cho ngăn xếp.
* **.Data**: Khởi đầu phân đoạn dữ liệu.
* **ball\_x dw 40**: Khởi tạo vị trí cột ban đầu của bóng
* **ball\_y dw 12**: Khơi tạo vị trí hang ban đầu của bóng
* **ball\_dx dw 1**: Khởi tạo tốc độ di chuyển bóng theo cột
* **ball\_dy dw 1**: Khởi tạo tốc độ di chuyển bóng theo hàng
* **paddle\_x dw 35**: Khởi tạo vị trí cột ban đầu của thanh đỡ
* **paddle\_y dw 23**: Khởi tạo vị trí hàng ban đầu của thanh đỡ
* **paddle\_width dw 12**: Khởi tạo chiều dài thanh đỡ
* **game\_active db 0**: Khai báo biến điều khiển trạng thái game
* **last\_hit db 0**: Khai báo biến va chạm cuối
* **heart db 03h**: Khai báo biến vẽ hình trái tim



* **tb (1 => 8):** Thông báo lời giới thiệu mở đầu



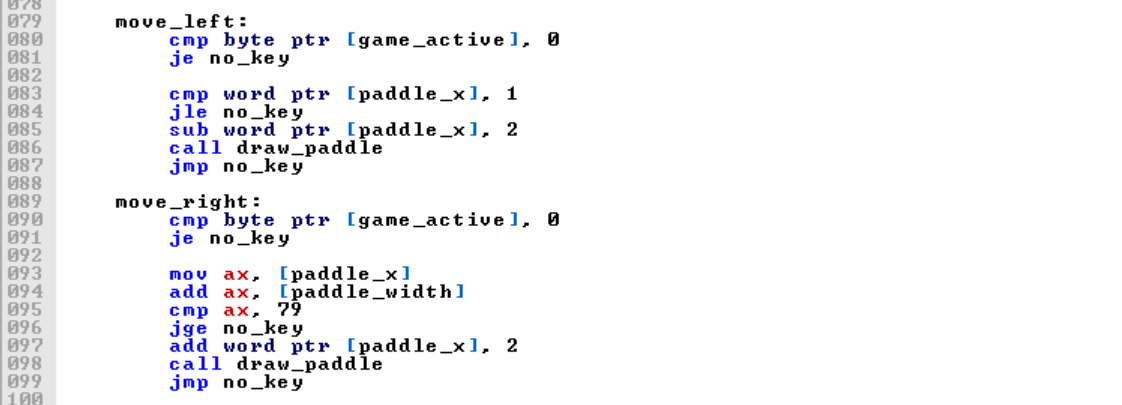
* **msg\_start**: Thông báo khi game mới bắt đầu.
* **msg\_pause**: Thông báo khi tạm dừng.
* **msg\_gameover**: Thông báo kết thúc game.
* **msg\_restart**: Hướng dẫn nhấn phím để chơi lại.
* **score dw 0**: Khởi tạo biến điểm bằng 0
* **tb\_score db 'Score:$'**: Hiện chuỗi ‘score’ trên màn hình
* **lives db 2**: Khởi tạo số mạng ban đầu bằng 2
* **tb\_live db 'Lives:$'**: Hiện chuỗi ‘lives’ trên màn hình



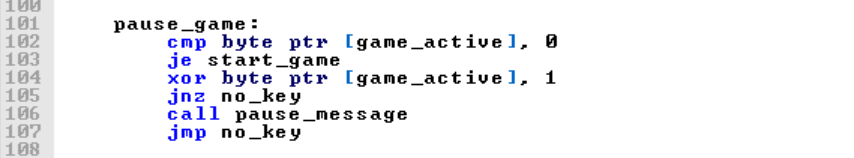
* **Main proc**: Khai báo thủ tục chính.
* **start:** Nhãn điểm bắt đầu.
* **mov ax, @data**: Đưa offset của phân đoạn Data vào AX.
* **mov ds, ax**: Gán DS = AX → DS trỏ tới Data.
* **mov ax,0B800h**: Ghi dữ liệu trực tiếp vào bộ nhớ video
* **mov es,ax**: Hiện ký tự màu sắc
* **call menu\_game**: Hiển thị menu hướng dẫn.
* **call screen\_game**: Vẽ khung game, in “Lives/Score”, thông báo chờ bắt đầu
* **call draw\_ball**: Vẽ bóng
* **call draw\_paddle**: Vẽ thanh đỡ



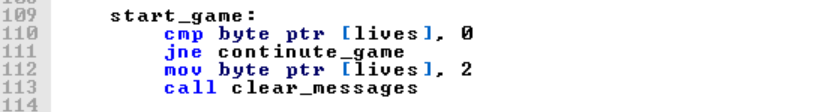
* **check\_game:** Nhãn đầu vòng lặp chính.
* **mov ah,1 / int 16h**: Kiểm tra phím: AH=1 → ZF=1 nếu không có phím.
* **jz no\_key**: Nếu chưa có phím, nhảy tới **no\_key.**
* **mov ah,0 / int 16h**: Đọc phím vào AL/AH.
* **cmp ah, 4Bh / je move\_left**: Nếu AH=4Bh (mũi tên trái), nhảy **move\_left**.
* **cmp ah, 4Dh / je move\_right**: Nếu AH=4Dh (mũi tên phải), nhảy **move\_right.**
* **cmp al, 20h / je pause\_game**: Nếu AL=20h (SPACE), nhảy **pause\_game**.
* **cmp al, 1Bh / je exit\_game**: Nếu AL=1Bh (ESC), nhảy **exit\_game**.
* **jmp no\_key**: Nếu không phải phím nào trên, quay lại xử lý **no\_key.**



* **cmp byte ptr [game\_active], 0**: Nếu game chưa được khởi chạy (0), không cho di chuyển thanh đỡ
* **je no\_key**: Nếu bằng 0 → nhảy qua **no\_key**.
* **cmp word ptr [paddle\_x], 1**: Kiểm tra xem paddle\_x đã ở sát biên trái (cột 1) chưa.
* **jle no\_key**: Nếu ≤1 → không di chuyển.
* **sub word ptr [paddle\_x], 2**: Giảm paddle\_x đi 2 (dịch thanh đỡ sang trái).
* **call draw\_paddle**: Vẽ lại thanh đỡ tại vị trí mới.
* **jmp no\_key**: Quay lại nhãn **no\_key** để tiếp tục vòng lặp chính.
* **mov ax, [paddle\_x]**: Lấy toạ độ trái paddle vào AX.
* **add ax, [paddle\_width]**: Cộng thêm chiều rộng → AX = cột bên phải của paddle.
* **cmp ax, 79**: So sánh với biên phải (cột 79).
* **jge no\_key**: Nếu ≥79 → không di chuyển.
* **add word ptr [paddle\_x],2**: Tăng paddle\_x thêm 2 (dịch phải).
* **call draw\_paddle**: Vẽ lại paddle.
* **jmp no\_key**: Quay lại vòng lặp chính.



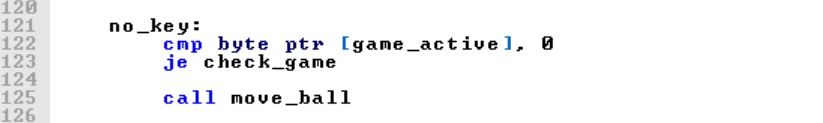
* **cmp byte ptr [game\_active],0**: Kiểm tra game đang chạy hay chưa.
* **je start\_game**: Nếu chưa chạy (0) → nhảy qua start\_game để khởi động.
* **xor byte ptr [game\_active],1**: Đảo bit game\_active: 1→0 (pause), 0→1 (resume).
* **jnz no\_key**: Nếu kết quả ≠0 (game vừa resume) → quay lại no\_key.
* **call pause\_message**: Nếu mới pause (game\_active=0), in thông báo “PAUSED…”.
* **jmp no\_key**: Tiếp tục vòng lặp.



* **cmp byte ptr [lives], 0**: Kiểm tra nếu đã hết mạng.
* **jne continute\_game**: Nếu còn mạng (>0) → nhảy qua **continue\_game.**
* **mov byte ptr [lives], 2**: Nếu hết, cấp thêm 2 mạng.
* **call clear\_messages**: Xoá thông báo cũ.



* **mov byte ptr [game\_active],1**: Bật cờ game active = chạy.
* **mov byte ptr [last\_hit],0**: Reset cờ va chạm cuối.
* **call clear\_messages**: Xoá thông báo (nếu có).
* **jmp no\_key**: Quay lại vòng lặp chính.



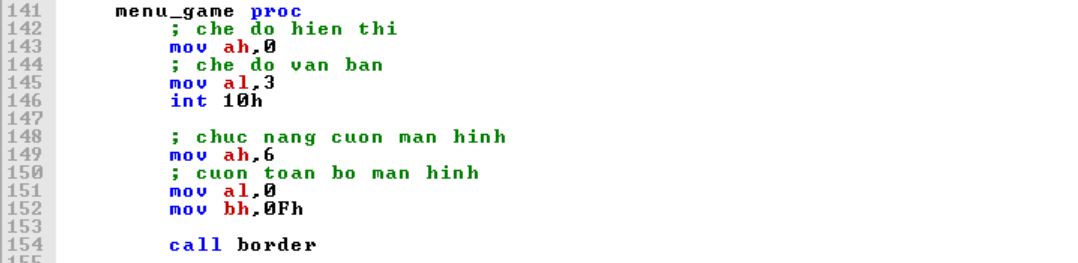
* **cmp byte ptr [game\_active], 0**: Nếu game đang pause → nhảy về check\_game.
* **call move\_ball**: Nếu game\_active =1, cập nhật vị trí và vẽ bóng.



* **mov cx, 0001h / mov dx, 0000h**: Thiết lập khoảng delay (1 tick).
* **mov ah, 86h / int 15h**: Gọi BIOS để đợi (điều khiển tốc độ).
* **jmp check\_game**: Quay lại vòng lặp xử lý phím.

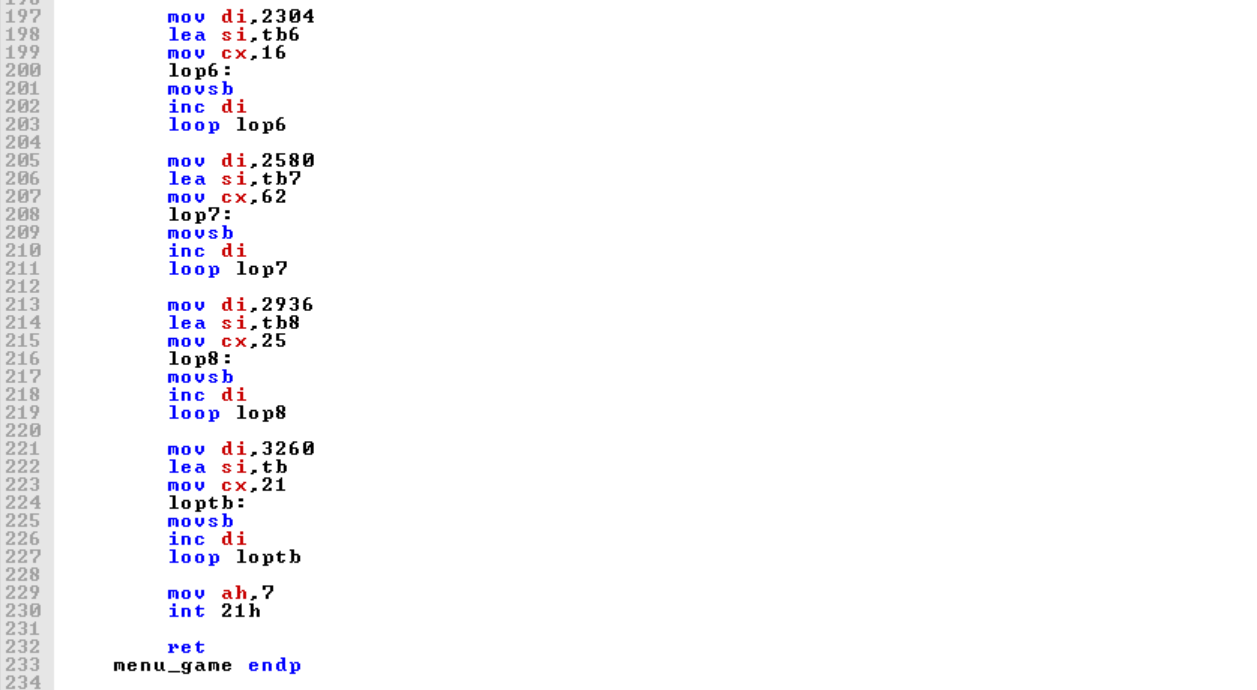


* **mov ax, 0003h / int 10h**: Chuyển video về mode 3 (text 80×25 mặc định).
* **mov ax, 4C00h / int 21h**: Kết thúc chương trình trả về DOS.

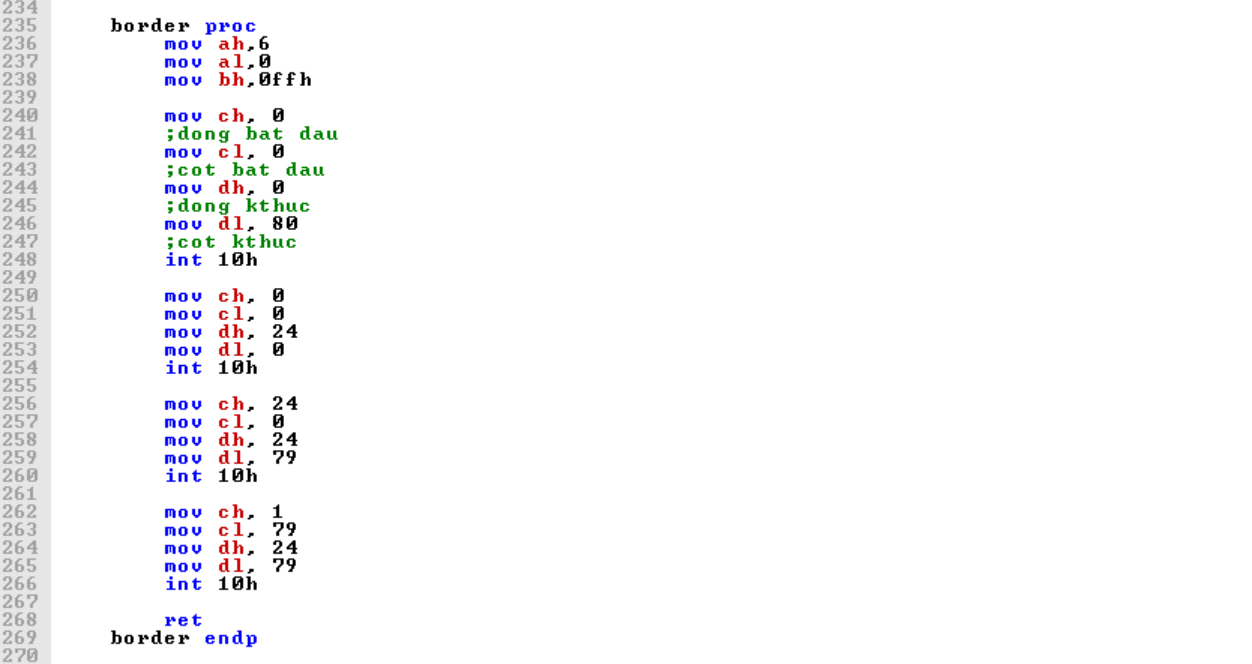


* **mov ah,0 / mov al,3 / int 10h**: Chuyển sang text mode 3.
* **mov ah,6 / mov al,0 / mov bh,0Fh / int 10h**: Cuộn toàn màn hình với nền trắng-chữ đen.
* **call border**: Vẽ khung viền màn hình.





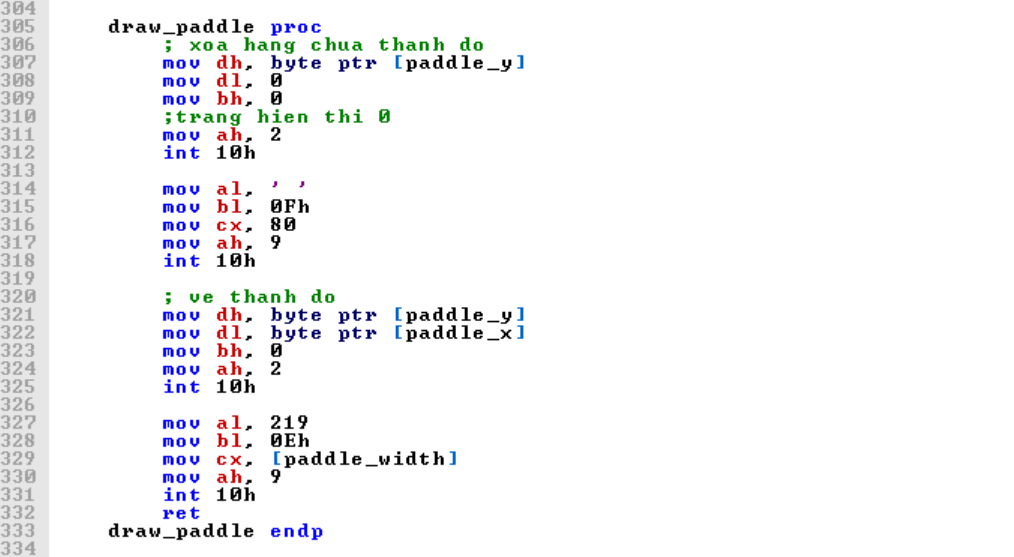
* **mov di, 704**: Đặt [ES:DI] vào vị trí hàng 1, cột 64 (màn 80×25).
* **lea si, tb1**: SI trỏ đến xâu tb1.
* **mov cx, 17**: Số ký tự của tb1.
* **lop1: / movsb / inc di / loop lop1**: Sao chép từng ký tự từ [SI] → [ES:DI] đến khi CX = 0 thì kết thúc vòng lặp
* **mov ah,7 / int 21h**: Chờ người dùng nhấn phím để tiếp tục.
* **ret**: Trở về Main.



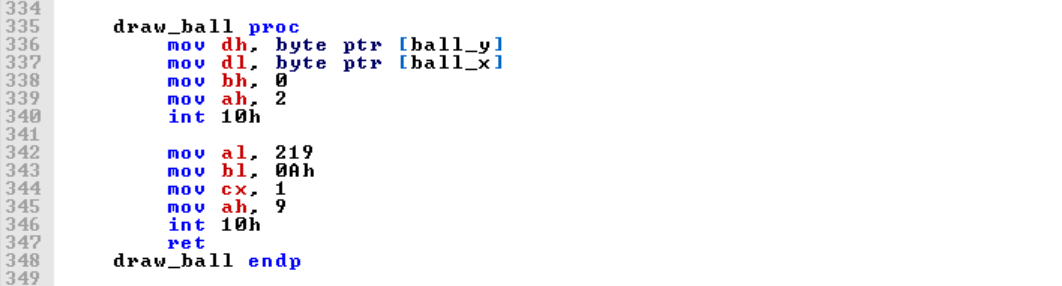
* **mov ah, 6**: Chức năng cuộn màn hình
* **mov al, 0**: AL=0 => Cuộn toàn bộ màn hình
* **mov bh, 0FFh**: thuộc tính nền màu trắng
* **mov ch, 0 / mov cl, 0**: chỉ đến dòng bắt đầu và cột bắt đầu
* **mov dh, 0 / mov dl, 80**: chỉ đến dòng kết thúc và cột kết thúc
* **int 10h**: Thực thi cuộn – ở đây thực chất khoanh vùng biên trên cùng
* **ret**: Trả về hàm



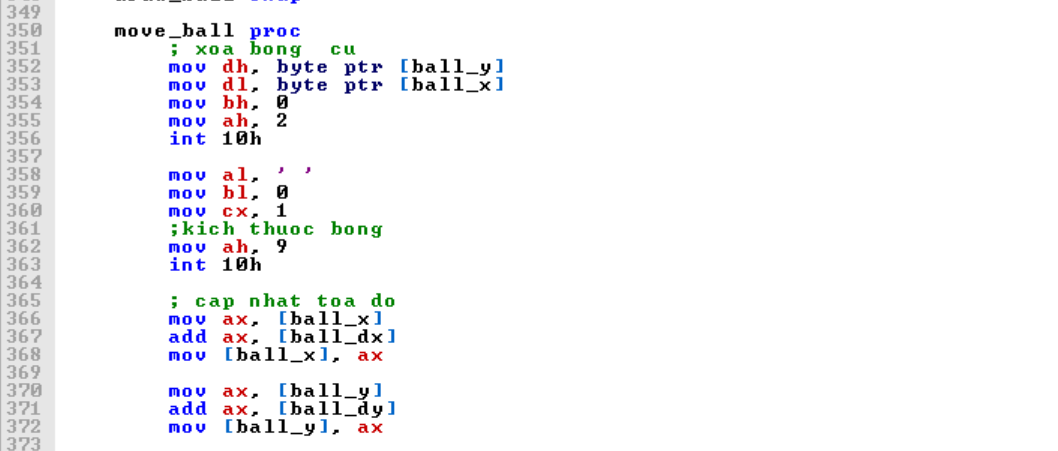
* **mov ax,3 / int 10h**: Chuyển sang video mode 3 (text 80×25).
* **mov ah, 6 / mov al,0 / mov bh,0Eh**: Chuẩn bị scroll (dùng để tái tạo khung) với attribute 0Eh (vàng nền đen).
* **mov dx, 0000h / mov ah,2 / int 10h**: INT 10h AH=2 đặt con trỏ tại DH=0, DL=0 (đầu màn).
* **mov al, 205**: AL=205 (ký tự “═”)
* **mov bl, 0Fh**: thuộc tính màu trắng sáng
* **mov cx, 80**: CX=80 lần in
* **mov ah, 9 / int 10h**: INT 10h AH=9 in CX ký tự AL với màu BL → vẽ biên trên.
* **call change\_live\_score**: Gọi thủ tục in Lives/Score.
* **mov di, 1650**: [ES:DI] = vị trí in msg\_start (hàng 20, cột 50).
* **lea si, msg\_start**: in chuỗi thông báo bắt đầu
* **mov cx, 33**: Độ dài chuỗi
* **lopst: movsb/inc di/loop lopst**: Copy từng ký tự qua video memory.
* **ret**: Trả về.



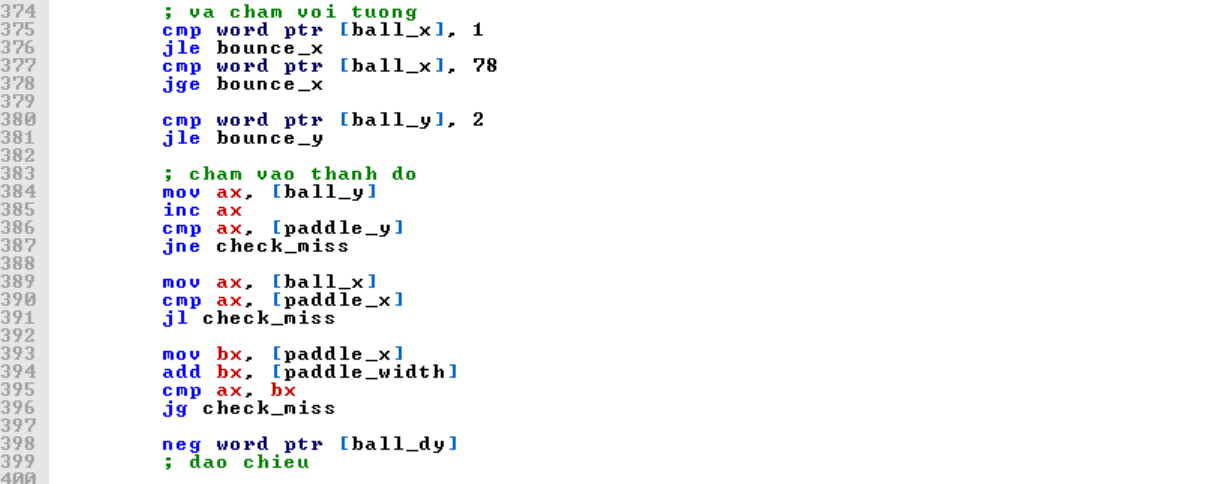
* **mov dh,[paddle\_y] / mov dl,0 / mov ah,2 / int 10h**: Dịch con trỏ đến đầu dòng hiện tại của paddle.
* **mov al,' ' / mov bl,0Fh / mov cx,80 / mov ah,9 / int 10h**: xóa hang in thanh đỡ
* **mov dh,[paddle\_y] / mov dl,[paddle\_x] / mov ah,2 / int 10h**: Dịch con trỏ đến vị trí bắt đầu vẽ paddle.
* **mov al,219 / mov bl,0Eh / mov cx,[paddle\_width] / mov ah,9 / int 10h**: In ký tự block (219) paddle\_width lần, màu vàng (0Eh).
* **ret**: Trả về.



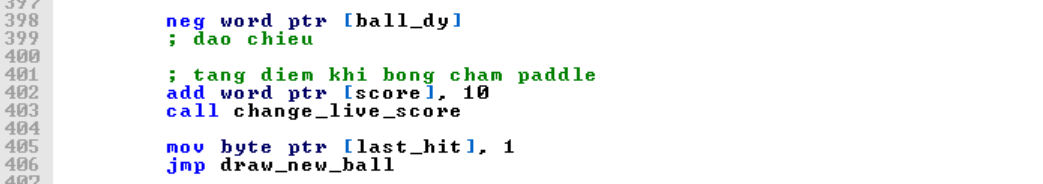
* **mov dh,[ball\_y] / mov dl,[ball\_x] / mov ah,2 / int 10h**: Dịch con trỏ đến vị trí bóng.
* **mov al, 219:** Ký tự block
* **mov bl, 0Ah:** Thuộc tính màu xanh lá
* **mov cx, 1:** block 1 ký tự
* **mov ah,9 / int 10h**: In 1 ký tự block màu xanh lá (0Ah).
* **ret**: Trả về.



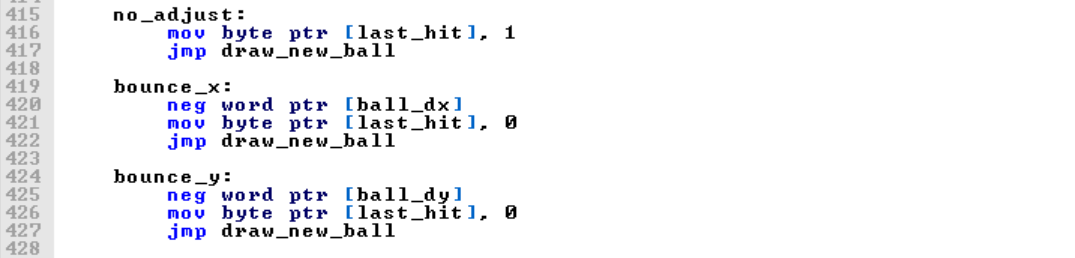
* **mov dh,[ball\_y]/dl,[ball\_x]/ah,2/int 10h**: Dịch con trỏ tới vị trí bóng cũ
* **mov al,' '/bl,0/cx,1/ah,9/int 10h**: Thay thế thuộc tính màu xanh bằng dấu cách để ẩn bóng cũ
* **mov ax,[ball\_x]/add ax,[ball\_dx]/mov [ball\_x],ax** : Trỏ tới vị trí cột mới
* **mov ax,[ball\_y]/add ax,[ball\_dy]/mov [ball\_y],ax** : Trỏ tới vị trí hàng mới



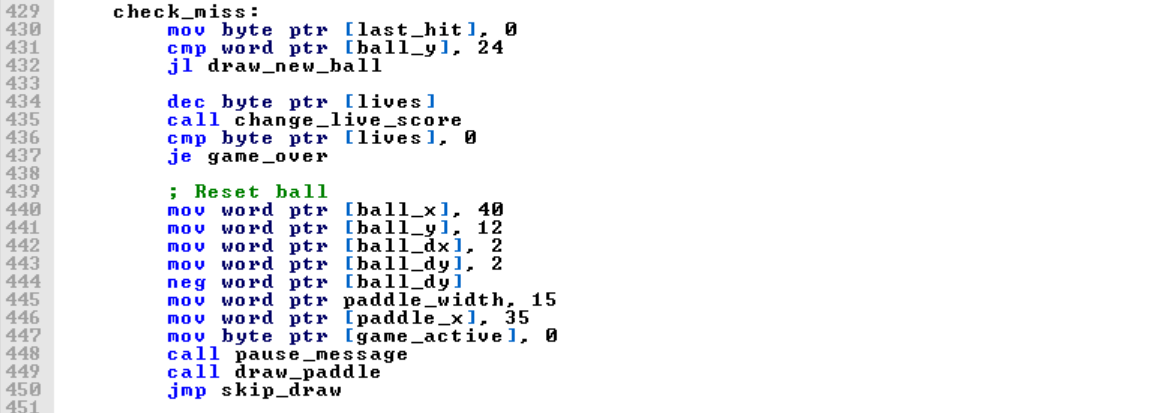
* **cmp ball\_x, 1:** so sánh vị trí bóng với cột biên đầu tiên
* **jle bounce\_x:** nếu nhỏ hơn hoặc bằng thì nhảy đến bounce\_x
* **cmp [ball\_x], 78:** so sánh vị trí bóng với cột biên thứ hai
* **jge boune\_x:** nếu lớn hơn hoặc bằng 78 thì nhảy đến boune\_x
* **cmp [ball\_y], 2:** so sánh vị trí bóng với hàng thứ 2
* **jle boune\_y:** nếu nhỏ hơn hoặc bằng thì nhảy đến bounce\_y
* **mov ax, [ball\_y] /inc ax /cmp ax, [paddle\_y]:** Kiểm tra xem vị trí hàng của bóng có trùng vào vị trí chứa thanh đỡ không
* **jne check\_miss**: nếu không bằng thì nhảy đến **check\_miss**
* **mov ax, [ball\_x] /cmp ax, [paddle\_x]:** Kiểm tra xem vị trí bóng có chạm vào biên trái của thanh đỡ không
* **jl:** nếu nhỏ hơn thì nhảy đến **check\_miss**
* **mov bx, [paddle\_x] /add bx, [paddle\_width] /cmp ax, bx:** Kiểm tra xem vị trí bóng có trùng với biên phải của thanh đỡ không
* **jg:** nếu lơn hơn thì nhảy đến **check\_miss**



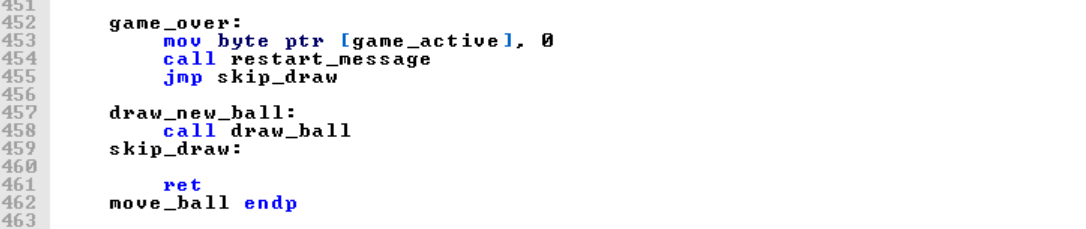
* **neg [ball\_dy]**: đảo chiều di chuyển của bóng
* **add [score], 10**: nếu chạm thanh đỡ thì tăng điểm thêm 10
* **call change\_live\_score**: gọi hàm change\_live\_score để in điểm số
* **mov [last\_hit], 1:** đánh dấu đã bóng chạm vào thanh đỡ
* **jmp draw\_new\_ball:** mặc định nhảy đến nhãn draw\_new\_ball để vẽ bóng mới



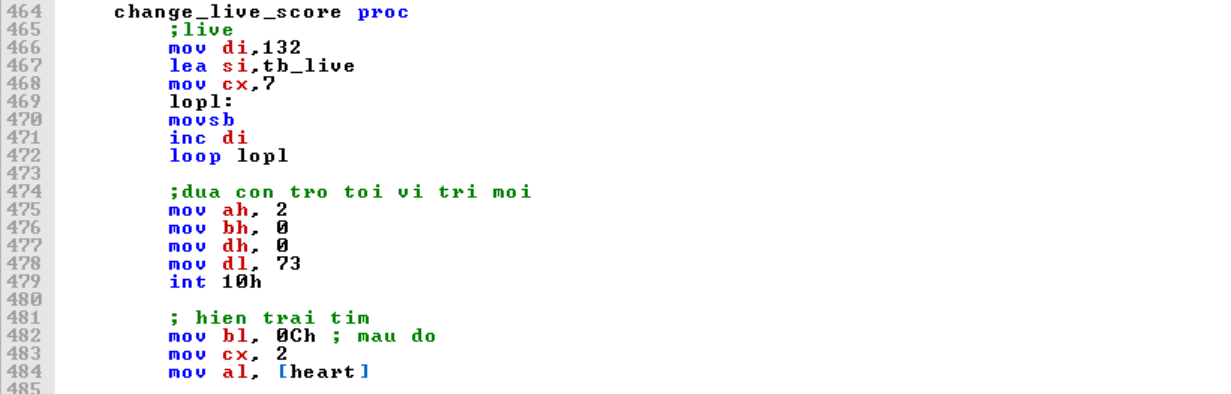
* **bounce\_x /bounce\_y:** nhãn dùng để đổi chiều khi va chạm với tường và vẽ bóng mới



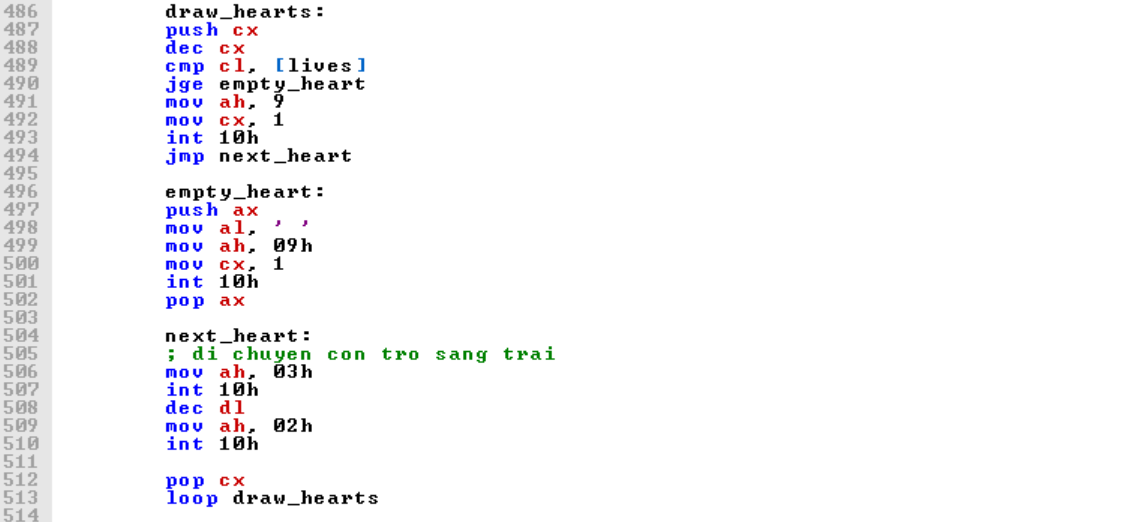
* **cmp [ball\_y], 24:** So sánh vị trí bóng với hàng cuối cùng
* **jl draw\_new\_ball**: Nếu nhỏ hơn thì vẽ bóng bình thường
* **dec [lives]**: Giảm mạng người chơi khi không đỡ được bóng
* **call change\_live\_score**: gọi hàm để xử lý mạng
* **cmp lives, 0:** so sánh người chơi còn mạng hay không
* **je game\_over:** nếu không còn mạng nào thì nhảy đến game\_over để kết thúc
* Nếu không bằng thì khởi tạo lại các biến để tiếp tục chơi



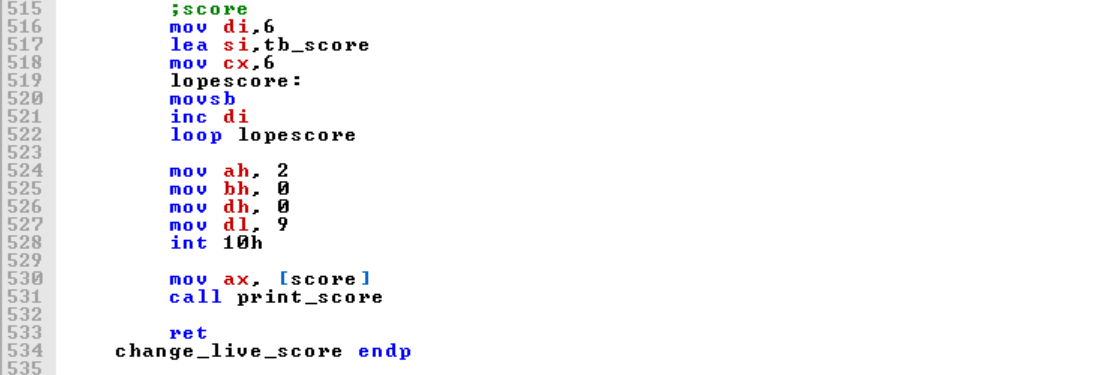
* **game\_over**: nhãn dung để kết thuc trò chơi và gọi hàm **restart\_game** để in điểm số và thông báo kết thúc
* **draw\_new\_ball**: Gọi draw\_ball vẽ tại toạ độ mới.
* **ret**: Trả về Main.



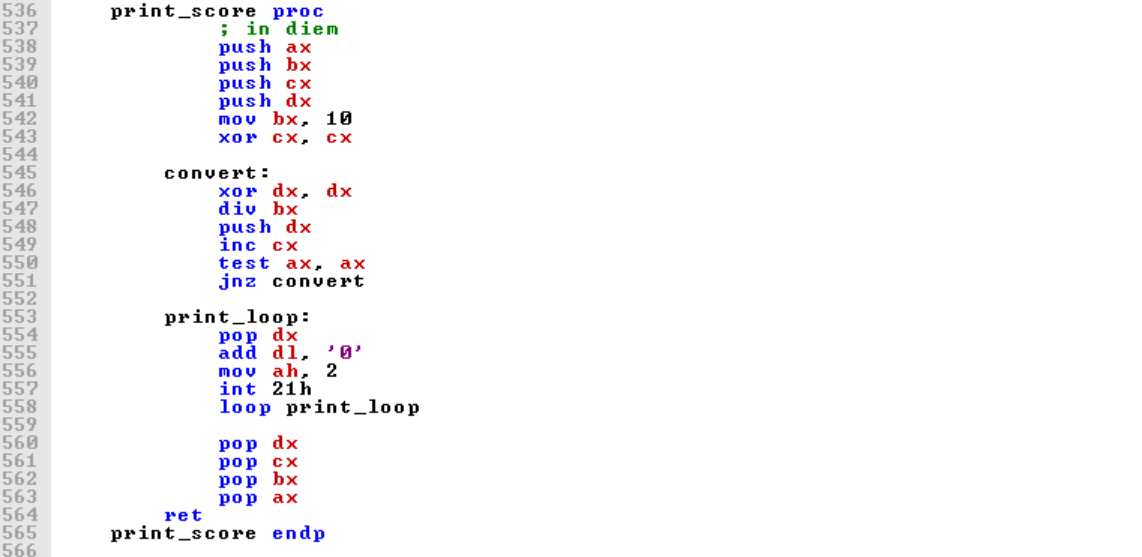
* **mov di, 132**: Đặt [ES:DI] vào offset 132 (hàng 1, cột 52
* **lea si,tb\_live**: SI trỏ đến chuỗi "Lives:$".
* **mov cx, 7**: Chuỗi dài 7 ký tự.
* **lopl:** Nhãn bắt đầu vòng copy.
* **movsb**: Copy byte từ [DS:SI] → [ES:DI].
* **inc di**: Dịch DI sang phải 1 cột.
* **loop lopl**: Giảm CX, nếu ≠0 lại lặp.
* **mov ah, 2**: AH=2 (INT 10h)
* **mov bh, 0**: Trang màn hình 0.
* **mov dh, 0**: Hàng 0.
* **mov dl, 73**: Cột 73.
* **int 10h**: Dịch con trỏ đến (0,73)
* **mov bl, 0Ch**: Thuộc tính màu đỏ
* **mov cx, 2**: gắn giá trị cx bằng 2
* **mov al, [heart]**: gắn al bằng biến heart để in ký tự



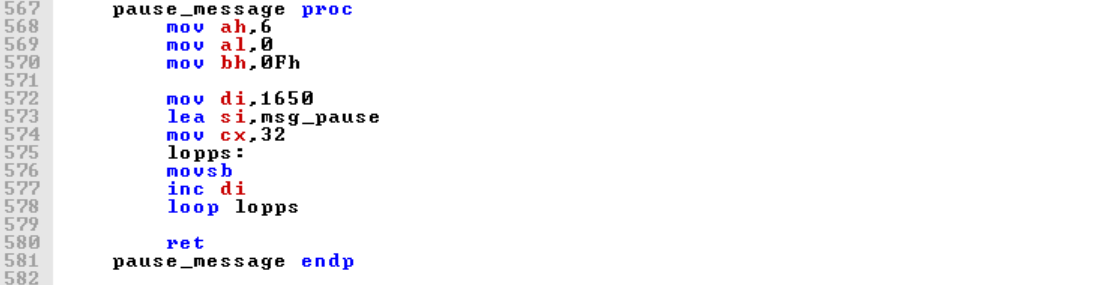
* **draw\_hearts:** Nhãn hiển thị mạng bằng hình trái tim
* **push cx**: Lưu CX tạm.
* **dec cx**: Giảm CX để sử dụng CL.
* **cmp cl, [lives]**: So sánh chỉ số với số mạng.
* **jge empty\_heart**: Nếu lớn hơn hoặc bằng, in ô trống.
* **mov ah,9 / mov cx,1 / int 10h**: INT 10h AH=9 in AL (trái tim) 1 lần.
* **jmp next\_heart**: Nhảy qua bước in trống.
* **empty\_heart:** Nhãn in ô trống.
* **push ax**: Lưu AX vì ta sửa AL.
* **mov al,' '**: AL = dấu cách.
* **mov ah,9 / mov cx,1 / int 10h**: In dấu cách 1 lần.
* **pop ax**: Khôi phục AX (ký tự heart).
* **next\_heart:** Nhãn chuẩn bị vẽ tiếp.
* **mov ah,3 / int 10h**: AH=3 INT 10h đọc cursor vào CH/DH.
* **dec dl**: Giảm DL (cột) sang trái 1.
* **mov ah,2 / int 10h**: Đặt cursor tại (CH,DL).
* **pop cx**: Khôi phục CX ban đầu.
* **loop draw\_hearts**: Lặp lại cho đến CX=0.



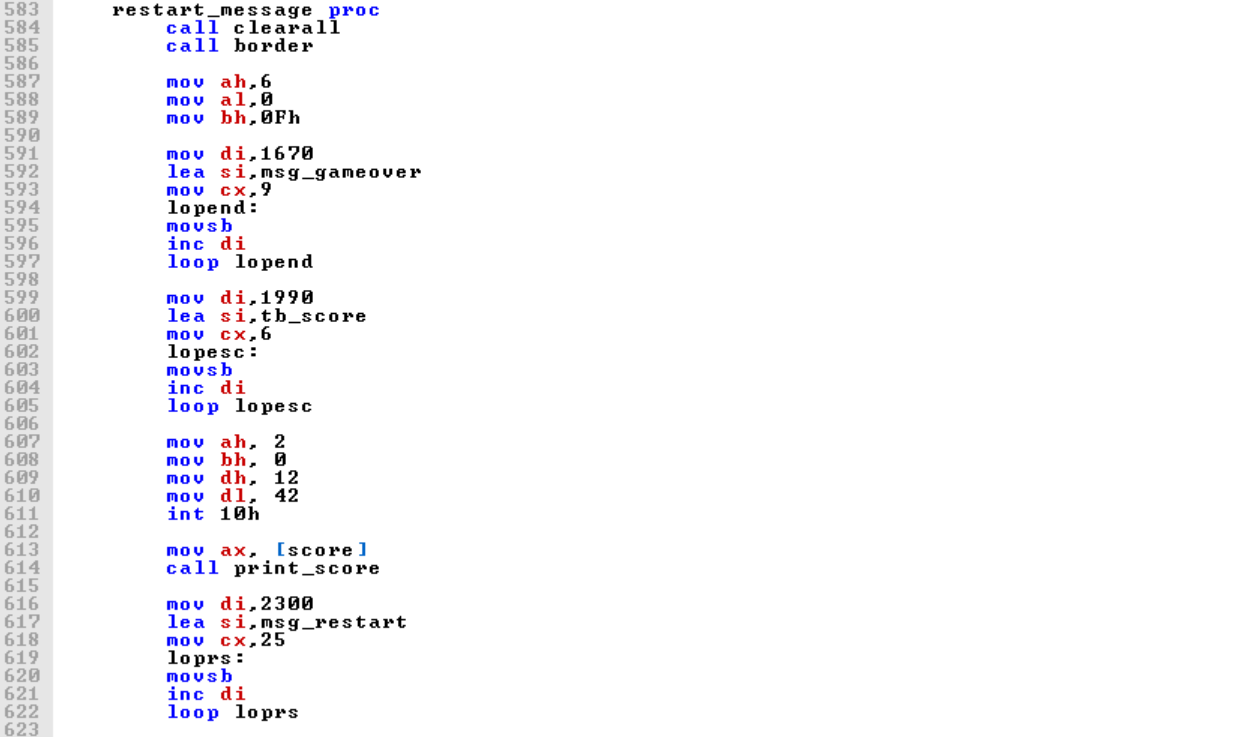
* **mov di, 6**: [ES:DI] trỏ tới hàng 0, cột 6
* **lea si, tb\_score**: gắn si bằng giá trị biến tb\_score
* **mov cx, 6**: Độ dài 6 ký tự.
* **lopescore:** Nhãn copy.
* **movsb / inc di / loop lopescore**: Copy từng ký tự.
* **mov ah,2/bh,0/dh,0/dl,9/int 10h**: Đặt con trỏ tới hàng 0 cột 9
* **mov ax,[score]**: Lấy biến score vào AX.
* **call print\_score**: Gọi thủ tục in số.
* **ret**: Trả về Main.



* **push ax/bx/cx/dx**: Lưu thanh ghi.
* **mov bx, 10**: Thiết lập hệ cơ số 10.
* **xor cx, cx**: CX=0, dùng đếm chữ số.
* **convert:** Nhãn chia số.
* **xor dx, dx**: DX=0 để tránh dư thừa.
* **div bx**: AX ÷ 10 → AX=quotient, DX=remainder.
* **push dx**: Lưu remainder (chữ số) lên stack.
* **inc cx**: Tăng đếm chữ số.
* **test ax, ax/jnz convert**: Nếu quotient ≠0, lặp.
* **print\_loop:** Nhãn in chữ số.
* **pop dx**: Lấy chữ số (0–9).
* **add dl, '0'**: Chuyển thành ký tự ASCII.
* **mov ah,2 /int 21h**: AH=2 INT 21h in DL.
* **loop print\_loop**: Lặp CX lần.
* **pop dx/cx/bx/ax**: Khôi phục thanh ghi.
* **ret**: Kết thúc.



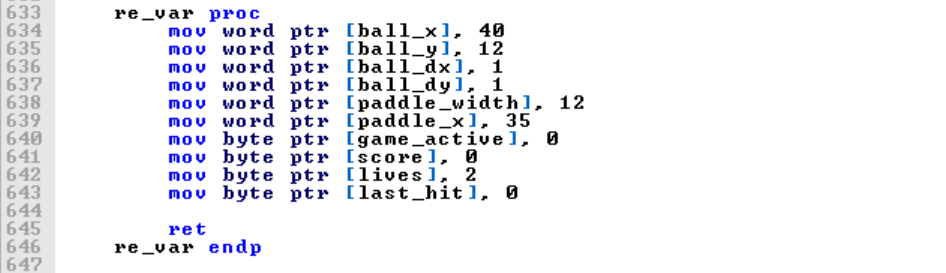
* **mov di, 1650**: [ES:DI] vị trí in msg\_pause.
* **lea si, msg\_pause**: gắn si bằng giá trị của msg\_pausse
* **mov cx, 32**: Độ dài 32 ký tự.
* **lopps: movsb/inc di/loop lopps**: Copy chuỗi.
* **ret**: Trả về.



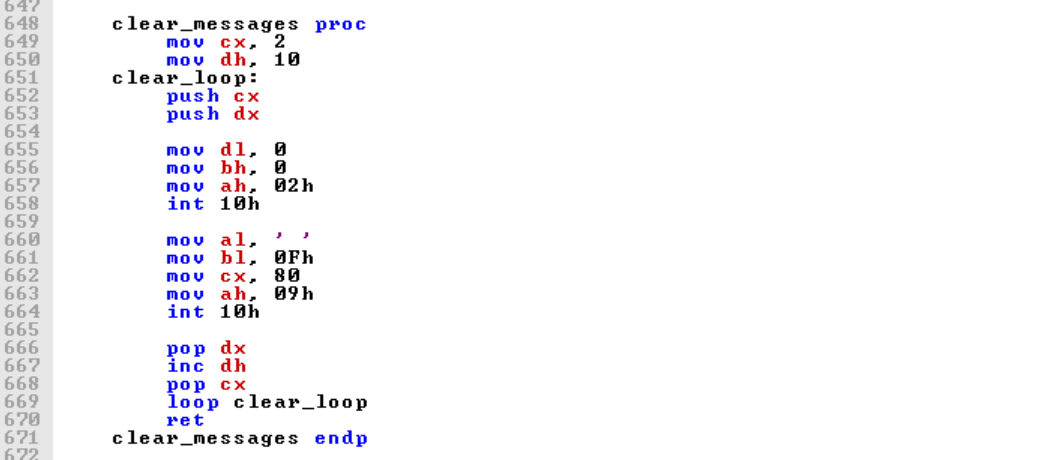
* **call clearall**: Xoá toàn màn hình.
* **call border**: Vẽ khung.
* **mov di, 1670**: ES:DI vị trí in “GAME OVER”.
* **lea si, msg\_gameover**: SI → "GAME OVER$".
* **mov cx, 9**: Độ dài 9 ký tự.
* **lopend: movsb/inc di/loop lopend**: Copy.
* **mov di, 1990**: Vị trí in “Score:”.
* **lea si, tb\_score** / **mov cx,6** / **lopesc:** Copy "Score:$".
* **mov ah, 2 /mov bh, 0 /mov dh, 12 /mov dl, 42 /int 10h**: Đặt con trỏ tới hàng 12 cột 42
* **mov ax, [score]** /**call print\_score**: Gọi hàm print\_score để in điểm số
* **mov di, 2300**: Vị trí in “Press any key…”.
* **lea si, msg\_restart** / **mov cx,25** / **loprs:** Copy chuỗi 25 ký tự.



* **mov ax, 1 /int 16h**: Chờ phím ấn bất kỳ
* **call clearall**: Xoá toàn màn hình.
* **call re\_var**: Reset biến.
* **jmp start**: Quay về nhãn điểm bắt đầu.
* **ret**: Trả về



* Reset **ball\_x = 40**, **ball\_y = 12**: vị trí khởi tạo.
* **ball\_dx = 1**, **ball\_dy = 1**: vận tốc ban đầu.
* **paddle\_width = 12**, **paddle\_x = 35**: paddle khởi tạo.
* **game\_active = 0**: tạm dừng.
* **score = 0**, **lives = 2**, **last\_hit = 0**: điểm, mạng và cờ va chạm reset.
* **ret**: Trả về.



* **mov cx, 2**: Xóa 2 dòng.
* **mov dh,10**: Bắt đầu từ hàng 10.
* **push cx /push dx**: Đẩy giá trị vào stack
* **mov dl, 0 /mov bh, 0 /mov ah, 2 /int 10h**: Đặt con trỏ tới vị trí chỉ định
* **mov al,' ' /mov bl, 0Fh /mov cx, 80 /mov ah, 9 /int 10h**: In 80 dấu cách xóa dòng.
* **pop dx**: Xóa dx khỏi stack
* **inc dh**: Sang dòng tiếp.
* **pop cx**: Xóa cx khỏi stack
* **loop clear\_loop**: Lặp lại 2 lần.
* **ret**: Trả về.



* **xor cx, cx**: CX=0
* **mov dh, 24 /mov dl, 79**: Vùng cuộn từ (0,0) đến (24,79).
* **mov bh, 7**: Thuộc tính nền xám, chữ đen
* **mov ax, 700h /int 10h**: xóa toàn màn hình text mode.
* **ret**: Trả về Main.
* **END**: Kết thúc file.

## **IV. Miêu Tả Giao Diện Chương Trình**

* Sau khi khởi động, chương trình hiển thị màn hình chơi như sau:

A screenshot of a computer

AI-generated content may be incorrect.

**Chú thích:**

* **Khung viền**: được tạo bằng lệnh INT 10h AH=6 để tô màu vùng viền, mô phỏng đường kẻ bao quanh màn hình.
* **Score**: nằm góc trái phía trên, cập nhật điểm mỗi khi bóng đập vào paddle ( mỗi lần bóng đập vào paddle sẽ được cộng 10 điểm )
* **Lives**: hiển thị 2 trái tim (♥) ở góc phải trên cùng, giảm mỗi khi người chơi để bóng rơi.
* **Paddle**: thanh chắn do người chơi điều khiển bằng các phím ← và →, nằm sát cuối màn hình.
* **Ball**: di chuyển tự động trên màn hình, phản xạ khi đụng tường hoặc paddle.
* Sau đó, ấn bất kì phím nào để game được bắt đầu:

A screenshot of a computer

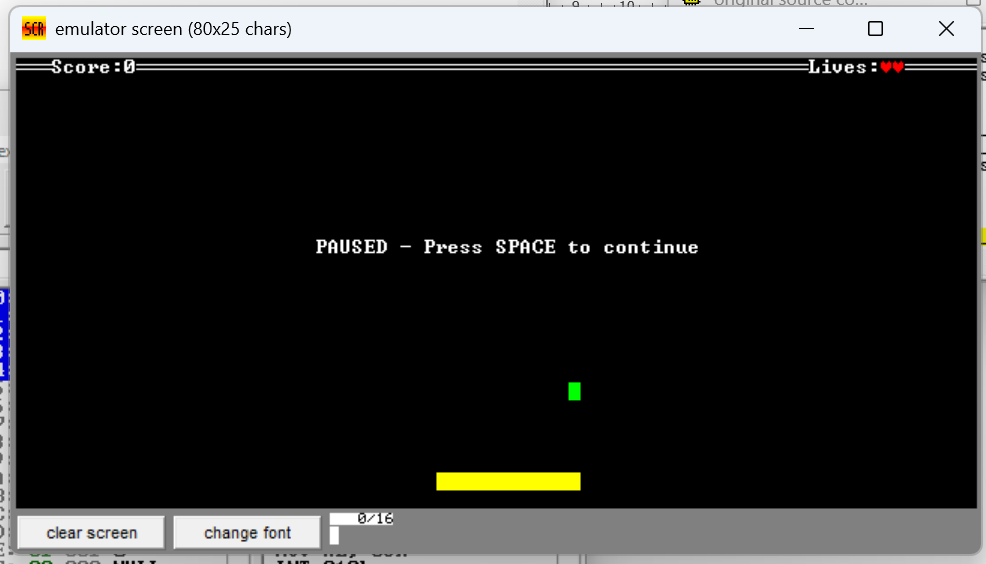
AI-generated content may be incorrect.

* Ấn SPACE để bắt đầu chơi, chương trình hiển thị màn hình chơi như sau:

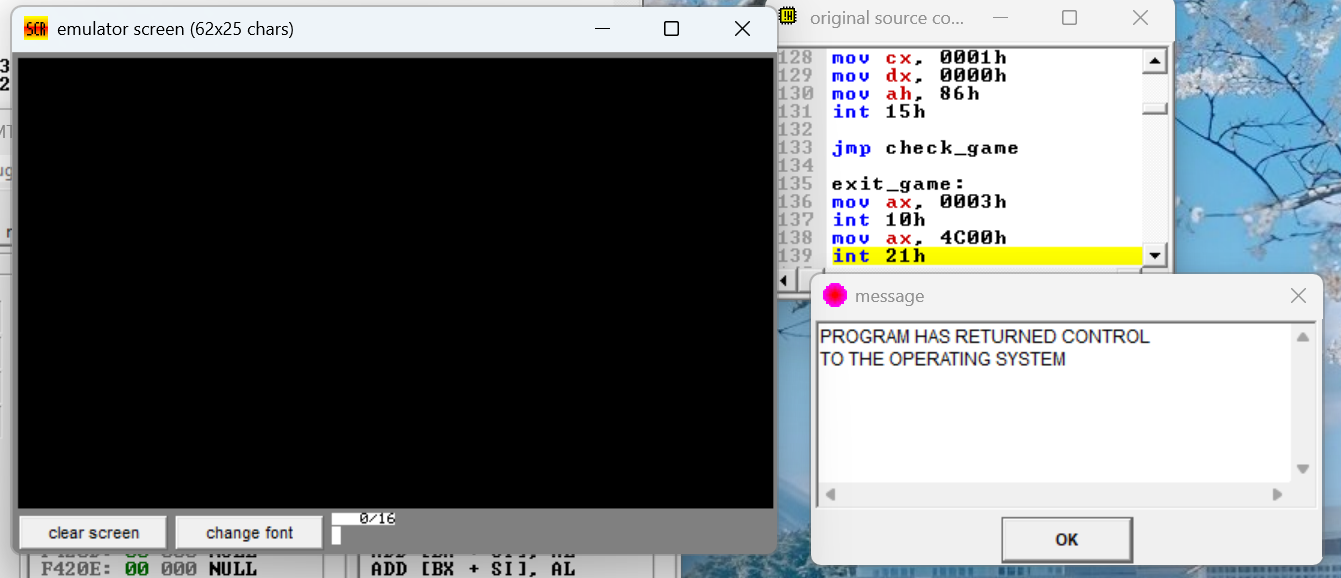
A screenshot of a computer

AI-generated content may be incorrect.

* Ấn SPACE để tạm dừng trong lúc chơi



* Nếu ấn ESC chương trình sẽ kết thúc



* Nếu bóng chạm viền, người chơi thua và màn hình sẽ hiển thị:

A screenshot of a computer

AI-generated content may be incorrect.

* Nếu chơi thua 2 lượt màn hình sẽ hiển thị:

A screenshot of a computer

AI-generated content may be incorrect.

* Game kết thúc, phần điểm sẽ được hiển thị ở dòng thử 2, ấn bất kì phím nào để tiếp tục chơi.

## **V. Tài Liệu Tham Khảo**

1.The Art of Assembly Language – Randall Hyde:

<https://www.plantation-productions.com/Webster/>

2. PC Assembly Language – Paul A. Carter:

<https://pacman128.github.io/static/pcasm-book.pdf>