

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

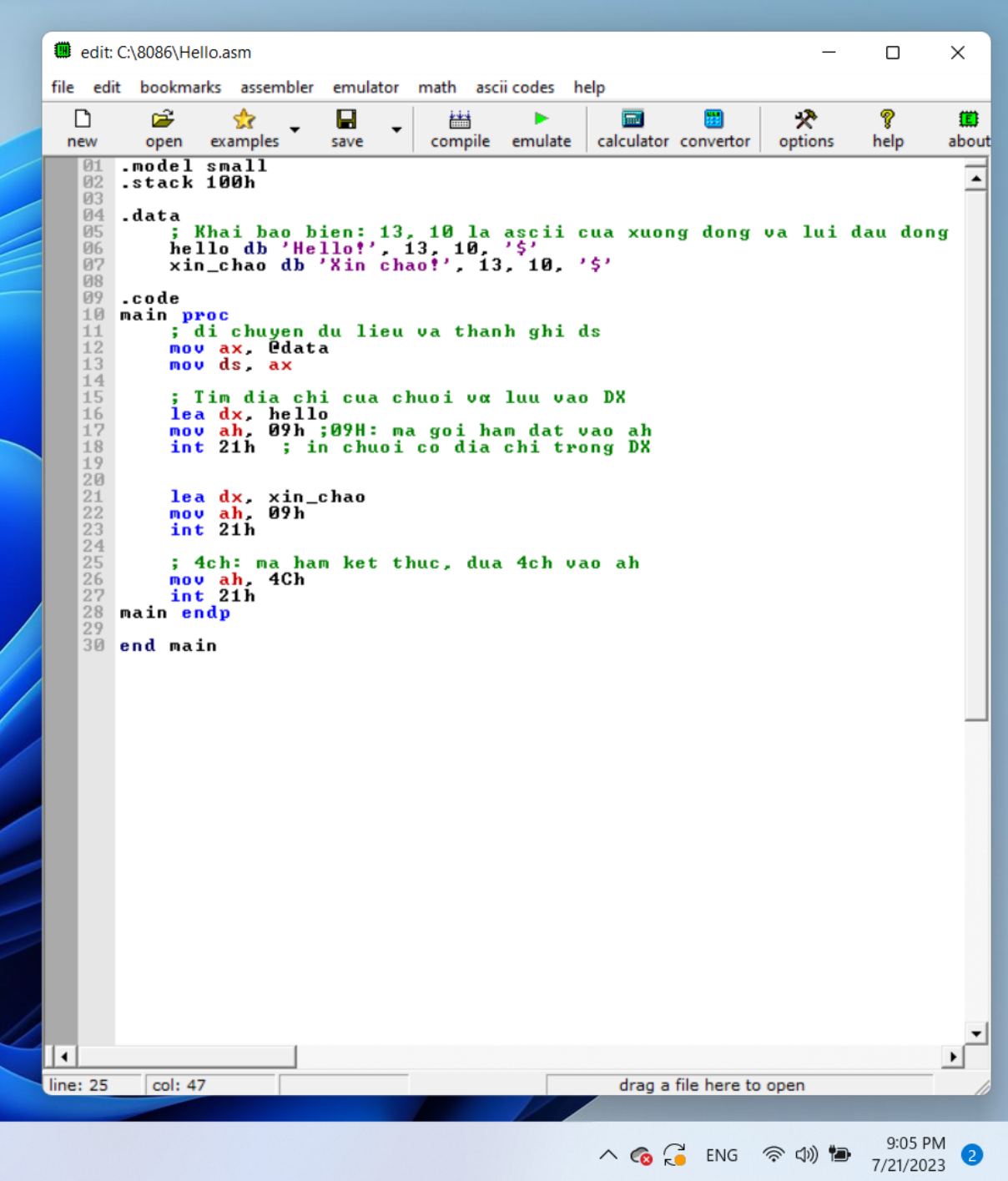
Giảng viên hướng dẫn: ThS. Đinh Xuân Trường

Hà Nội 2023

PHẦN 1: PHẦN LÀM CÁ NHÂN

Bài số 1: Lập trình hợp ngữ Assembly

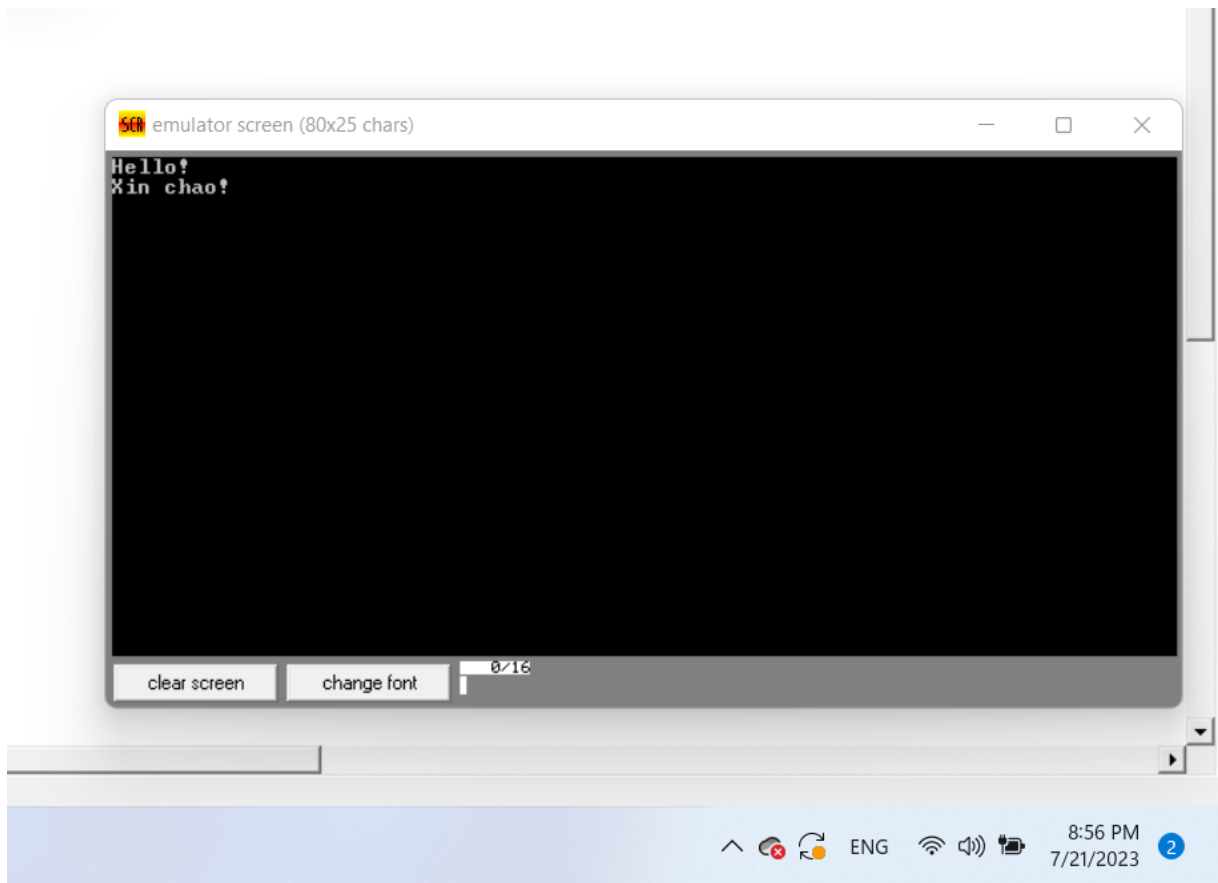
Câu 1: Viết chương trình hợp ngữ in ra lời chào Tiếng Anh và Tiếng Việt



The screenshot shows a text editor window titled "edit: C:\8086\Hello.asm". The window contains assembly code for an x86-64 system. The code is as follows:

```
01 .model small
02 .stack 100h
03
04 .data
05 ; Khai bao bien: 13, 10 la ascii cua xuong dong va lui dau dong
06 hello db 'Hello!', 13, 10, '$'
07 xin_chao db 'Xin chào!', 13, 10, '$'
08
09 .code
10 main proc
11 ; di chuyen du lieu va thanh ghi ds
12 mov ax, @data
13 mov ds, ax
14
15 ; Tim dia chi cua chuoi va luu vao DX
16 lea dx, hello
17 mov ah, 09h ; 09h: ma goi ham dat vao ah
18 int 21h ; in chuoi co dia chi trong DX
19
20
21 lea dx, xin_chao
22 mov ah, 09h
23 int 21h
24
25 ; 4ch: ma ham ket thuc, dua 4ch vao ah
26 mov ah, 4Ch
27 int 21h
28 main endp
29
30 end main
```

The status bar at the bottom of the window shows "line: 25" and "col: 47". The taskbar at the bottom of the screen shows the time "9:05 PM" and the date "7/21/2023".



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.

```

edit: C:\emu8086\examples\1_sample.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  option

01 .model small
02 .stack 100h
03 .data
04 str db 256 dup('$') ;khởi tạo chuỗi str
05 tb1 db 10, 13, 'Chuyen sang chuoi in thường: $'
06 tb2 db 10, 13, 'Chuyen sang chuoi in hoa: $'
07 .Code
08 main proc
09     mov ax, @data
10     mov ds, ax
11
12     ;Nhập chuỗi:
13     lea dx, str
14     mov ah, 10 ;nhập vào ngay 10
15     int 21h
16
17     ;In thông báo 1
18     mov ah, 9
19     lea dx, tb1 ;hiển thị tb1
20     int 21h
21     call lower ;hiển thị chuỗi thường
22
23     ;In chuỗi in hoa:
24     mov ah, 9
25     lea dx, tb2 ;hiển thị thông báo tb2
26     int 21h
27     call upper
28
29     mov ah, 4ch
30     int 21h
31 main endp
32
33 lower proc
34     lea si, str+2 ;
35     Loop1: ;lặp lại : kiểm tra từng ký tự một
36         mov dl, [si]
37         cmp dl, 'A' ;compare
38         jl Print1
39         cmp dl, 'Z' ;compare
40         jg Print1
41         add dl, 32 ;chuyển ký tự hoa thành thường
42     Print1:
43         mov ah, 2 ;in ký tự
44         int 21h
45         inc si ;increase
46         cmp [si], '$' ;compare
47         jne Loop1
48     ret
49 lower endp
50
51 upper proc
52     lea si, str+2
53     Loop2:
54         mov dl, [si]

```

line: 29 col: 43 drag a file here to open

```
edit: C:\emu8086\examples\1_sample.asm
file  edit  bookmarks  assembler  emulator  math  ascii codes  help
new  open  examples  save  compile  emulate  calculator  convertor  option

46      cmp [si], '$' ;compare
47      jne Loop1
48      ret
49  lower endp
50
51  upper proc
52      lea si, str+2
53      Loop2:
54          mov dl, [si]
55          cmp dl, 'a'
56          jl Print2
57          cmp dl, 'z'
58          jg Print2
59          sub dl, 32 ;chuyen ky tu thuong thanh hoa
60      Print2:
61          mov ah, 2
62          int 21h
63          inc si
64          cmp [si], '$'
65          jne Loop2
66      ret
67  upper endp
68  end

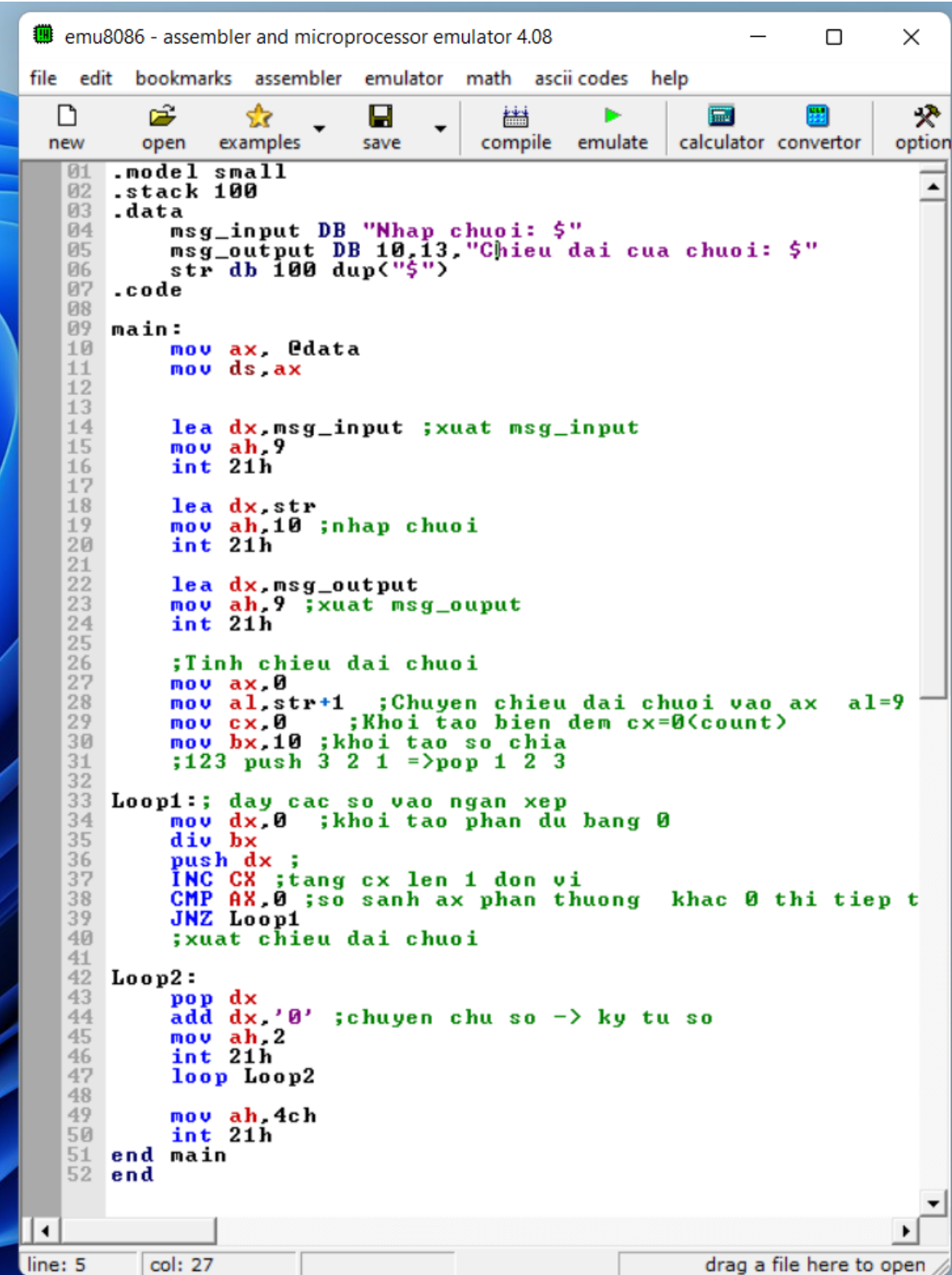
line: 29  col: 43  drag a file here to open
```

emulator screen (80x25 chars)

PhUonG
Chuyen sang chuoi in thuong: phuong
Chuyen sang chuoi in hoa: PHUONG

clear screenchange font0/16

Câu 10: Viết chương trình hợp ngữ Assembly yêu cầu đếm chiều dài của một chuỗi ký tự cho trước.



```
01 .model small
02 .stack 100
03 .data
04     msg_input DB "Nhap chuoi: $"
05     msg_output DB 10,13,"Chieu dai cua chuoi: $"
06     str db 100 dup("$")
07 .code
08
09 main:
10     mov ax, @data
11     mov ds, ax
12
13
14     lea dx, msg_input ;xuat msg_input
15     mov ah, 9
16     int 21h
17
18     lea dx, str
19     mov ah, 10 ;nhap chuoi
20     int 21h
21
22     lea dx, msg_output
23     mov ah, 9 ;xuat msg_output
24     int 21h
25
26     ;Tinh chieu dai chuoi
27     mov ax, 0
28     mov al, str+1 ;Chuyen chieu dai chuoi vao ax al=9
29     mov cx, 0 ;Khoi tao bien dem cx=0(count)
30     mov bx, 10 ;khoi tao so chia
31     ;123 push 3 2 1 =>pop 1 2 3
32
33 Loop1:; day cac so vao ngan xep
34     mov dx, 0 ;khoi tao phan du bang 0
35     div bx
36     push dx ;
37     INC CX ;tang cx len 1 don vi
38     CMP AX, 0 ;so sanh ax phan thuong khac 0 thi tiep t
39     JNZ Loop1
40     ;xuat chieu dai chuoi
41
42 Loop2:
43     pop dx
44     add dx, '0' ;chuyen chu so -> ky tu so
45     mov ah, 2
46     int 21h
47     loop Loop2
48
49     mov ah, 4ch
50     int 21h
51 end main
52 end
```

line: 5 col: 27 drag a file here to open

11:41 PM 7/22/2023

emulator screen (80x25 chars)

Nhap chuoi: hovietchuong
Chieu dai cua chuoi: 12

clear screen

change font

0/16



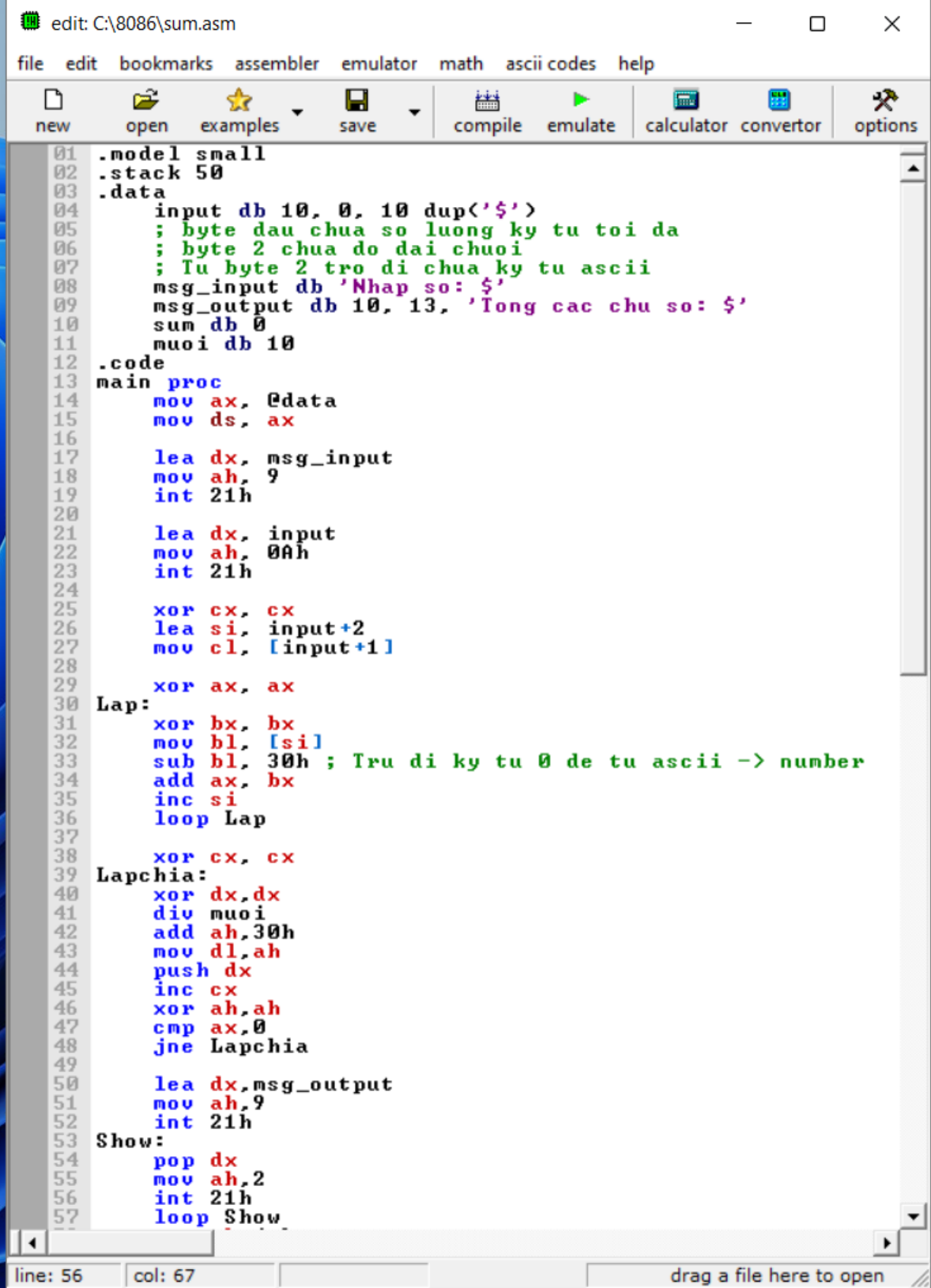
ENG



11:40 PM
7/22/2023

1

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



```
01 .model small
02 .stack 50
03 .data
04     input db 10, 0, 10 dup('$')
05     ; byte đầu chứa số lượng ký tự tối đa
06     ; byte 2 chứa độ dài chuỗi
07     ; Tu byte 2 trở đi chứa ký tự ascii
08     msg_input db 'Nhập số: $'
09     msg_output db 10, 13, 'Tổng các chu số: $'
10     sum db 0
11     muoi db 10
12 .code
13 main proc
14     mov ax, @data
15     mov ds, ax
16
17     lea dx, msg_input
18     mov ah, 9
19     int 21h
20
21     lea dx, input
22     mov ah, 0Ah
23     int 21h
24
25     xor cx, cx
26     lea si, input+2
27     mov cl, [input+1]
28
29     xor ax, ax
30 Lap:
31     xor bx, bx
32     mov bl, [si]
33     sub bl, 30h ; Trừ đi ký tự 0 để từ ascii -> number
34     add ax, bx
35     inc si
36     loop Lap
37
38     xor cx, cx
39 Lapchia:
40     xor dx, dx
41     div muoi
42     add ah, 30h
43     mov dl, ah
44     push dx
45     inc cx
46     xor ah, ah
47     cmp ax, 0
48     jne Lapchia
49
50     lea dx, msg_output
51     mov ah, 9
52     int 21h
53 Show:
54     pop dx
55     mov ah, 2
56     int 21h
57     loop Show_
```

line: 56 col: 67

drag a file here to open

12:38 AM 7/23/2023

edit: C:\8086\sum.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator convertor options

```
49  
50     lea dx,msg_output  
51     mov ah,9  
52     int 21h  
53 Show:  
54     pop dx  
55     mov ah,2  
56     int 21h  
57     loop Show  
58     mov ah,4ch  
59     int 21h  
60     main endp  
61 end main
```

line: 56 col: 67 drag a file here to open

12:39 AM 7/23/2023

emulator screen (80x25 chars)

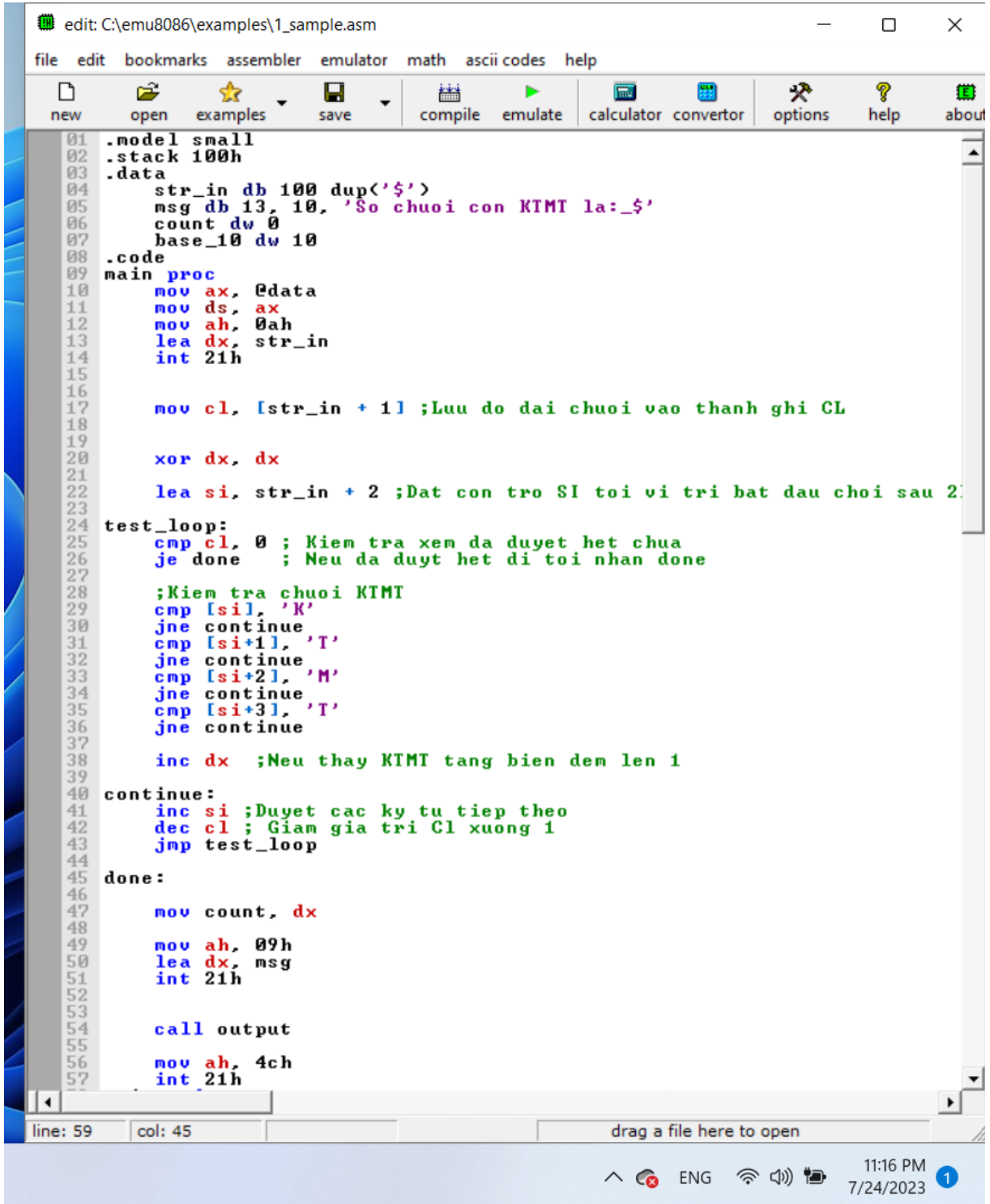
```
Nhap so: 123456  
Tong cac chu so: 21
```

clear screen change font 0/16

12:37 AM 7/23/2023

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.



```
001 .model small
002 .stack 100h
003 .data
004     str_in db 100 dup('$')
005     msg db 13, 10, 'Số chuỗi con KTMT là:$_$'
006     count dw 0
007     base_10 dw 10
008 .code
009 main proc
010     mov ax, @data
011     mov ds, ax
012     mov ah, 0ah
013     lea dx, str_in
014     int 21h
015
016
017     mov cl, [str_in + 1] ;Lưu độ dài chuỗi vào thanh ghi CL
018
019
020     xor dx, dx
021
022     lea si, str_in + 2 ;Đặt con trỏ SI tới vị trí bắt đầu chuỗi sau 2
023
024 test_loop:
025     cmp cl, 0 ; Kiểm tra xem đã duyệt hết chưa
026     je done ; Nếu đã duyệt hết đi tới nhãn done
027
028     ;Kiểm tra chuỗi KTMT
029     cmp [si], 'K'
030     jne continue
031     cmp [si+1], 'T'
032     jne continue
033     cmp [si+2], 'M'
034     jne continue
035     cmp [si+3], 'T'
036     jne continue
037
038     inc dx ;Nếu thấy KTMT tăng biến đếm lên 1
039
040 continue:
041     inc si ;Duyệt các ký tự tiếp theo
042     dec cl ; Giảm giá trị CL xuống 1
043     jmp test_loop
044
045 done:
046
047     mov count, dx
048
049     mov ah, 09h
050     lea dx, msg
051     int 21h
052
053
054     call output
055
056     mov ah, 4ch
057     int 21h
```

line: 59 col: 45 drag a file here to open 11:16 PM 7/24/2023

edit: C:\emu8086\examples\1_sample.asm

file edit bookmarks assembler emulator math ascii codes help

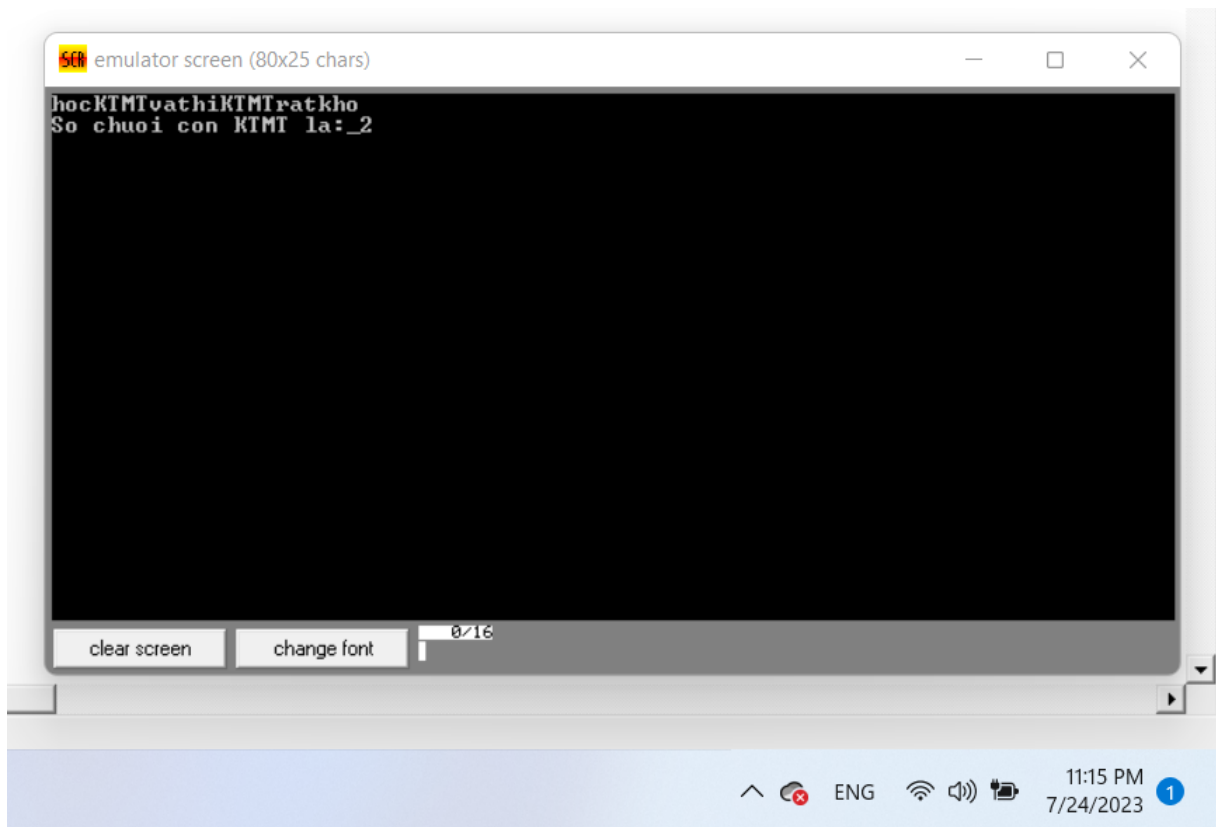
new open examples save compile emulate calculator convertor options help about

```
43      jmp test_loop
44
45 done:
46
47     mov count, dx
48
49     mov ah, 09h
50     lea dx, msg
51     int 21h
52
53
54     call output
55
56     mov ah, 4ch
57     int 21h
58 main endp
59 ; chuyen doi so lan xuat hien thanh he 10
60 output proc
61     mov ax, count
62     mov cx, 0
63
64 divide:
65     xor dx, dx
66     div base_10
67     push dx
68     inc cx
69     test ax, ax
70     jnz divide
71
72 show:
73     pop dx
74     add dl, '0'
75     mov ah, 02h
76     int 21h
77     loop show
78     ret
79 output endp
80
81 end main
82
```

line: 59 col: 45

drag a file here to open

11:16 PM 7/24/2023

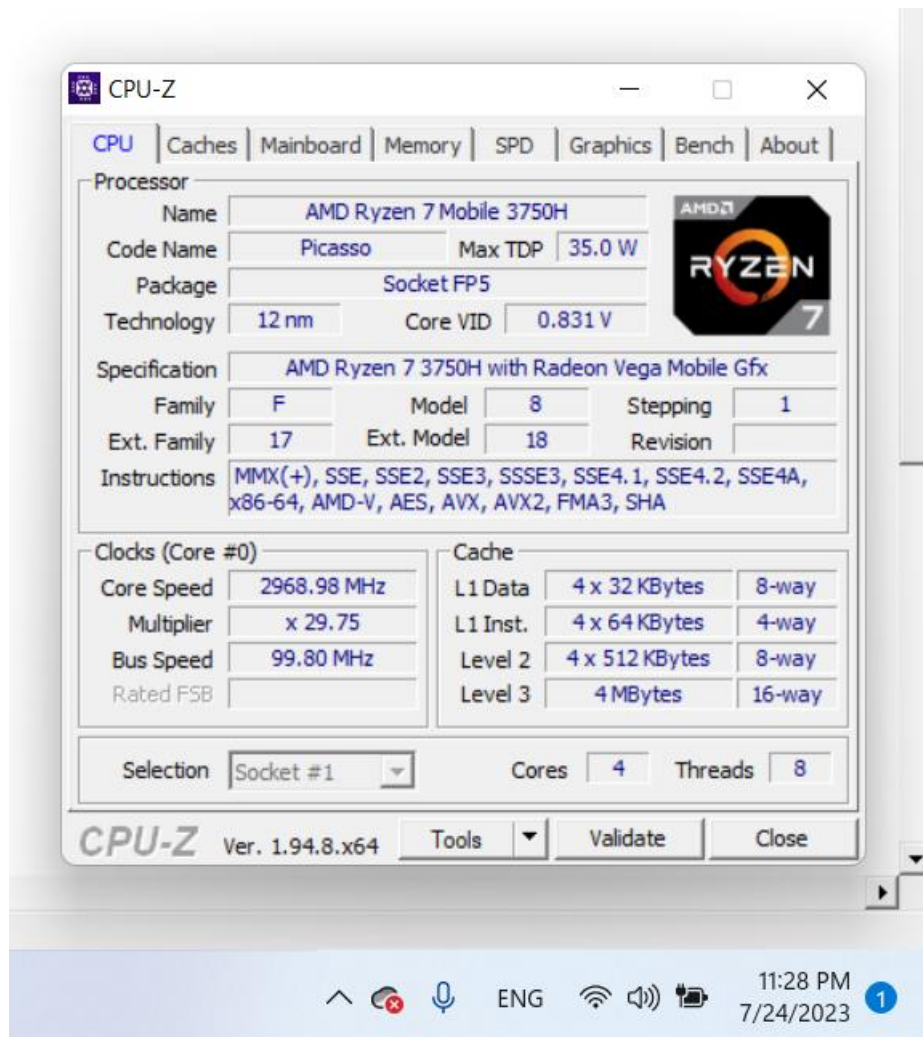


Bài số 2: Thực hành phân tích khảo sát bộ nhớ:

- 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:

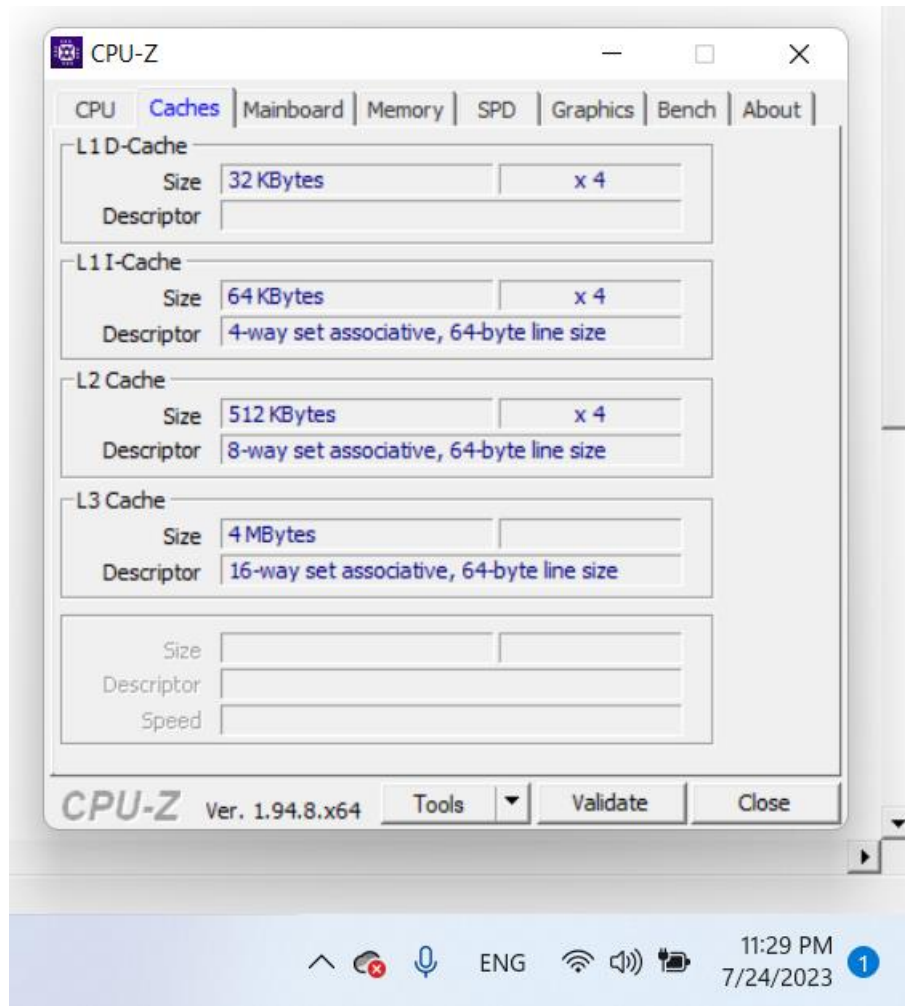


+ Bộ vi xử lí : AMD Ryzen 7 Mobile 3750H

+ 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

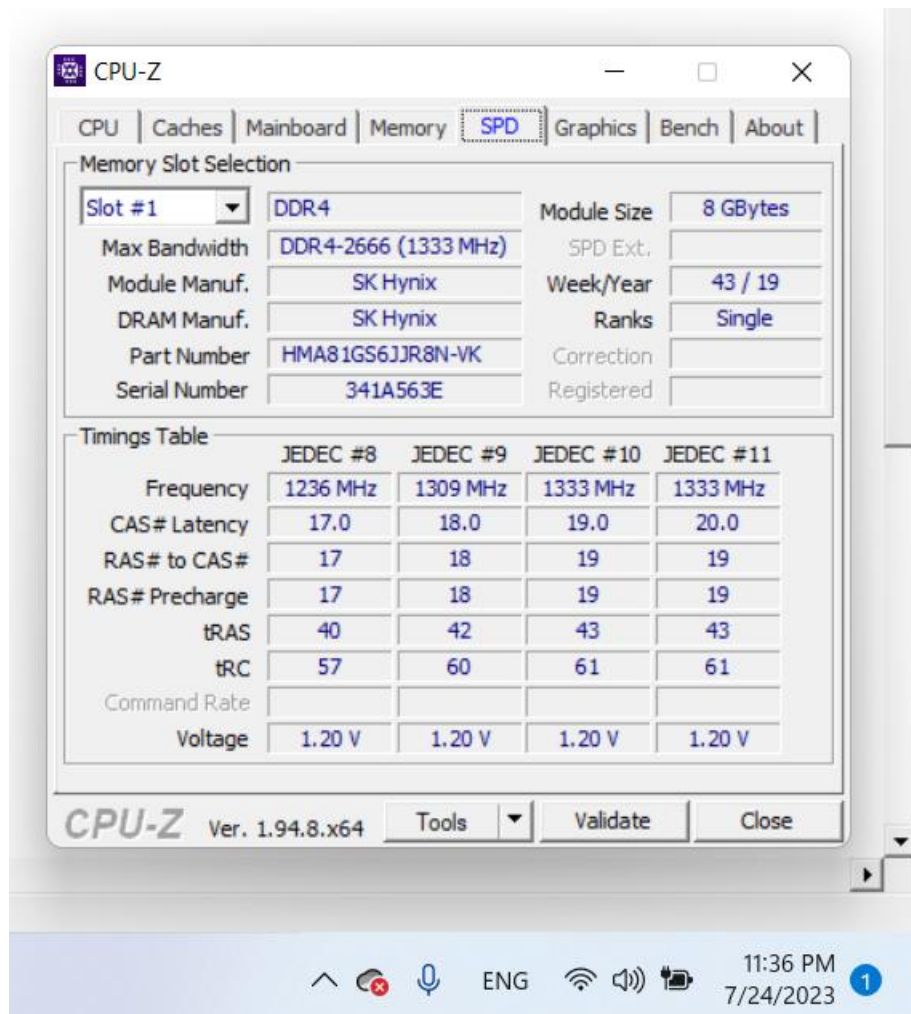
+ L2 Cache

- 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)
- Ảnh xạ tập kết hợp
- Kích thước 1 line: 64B

RAM/ROM:



- ROM:

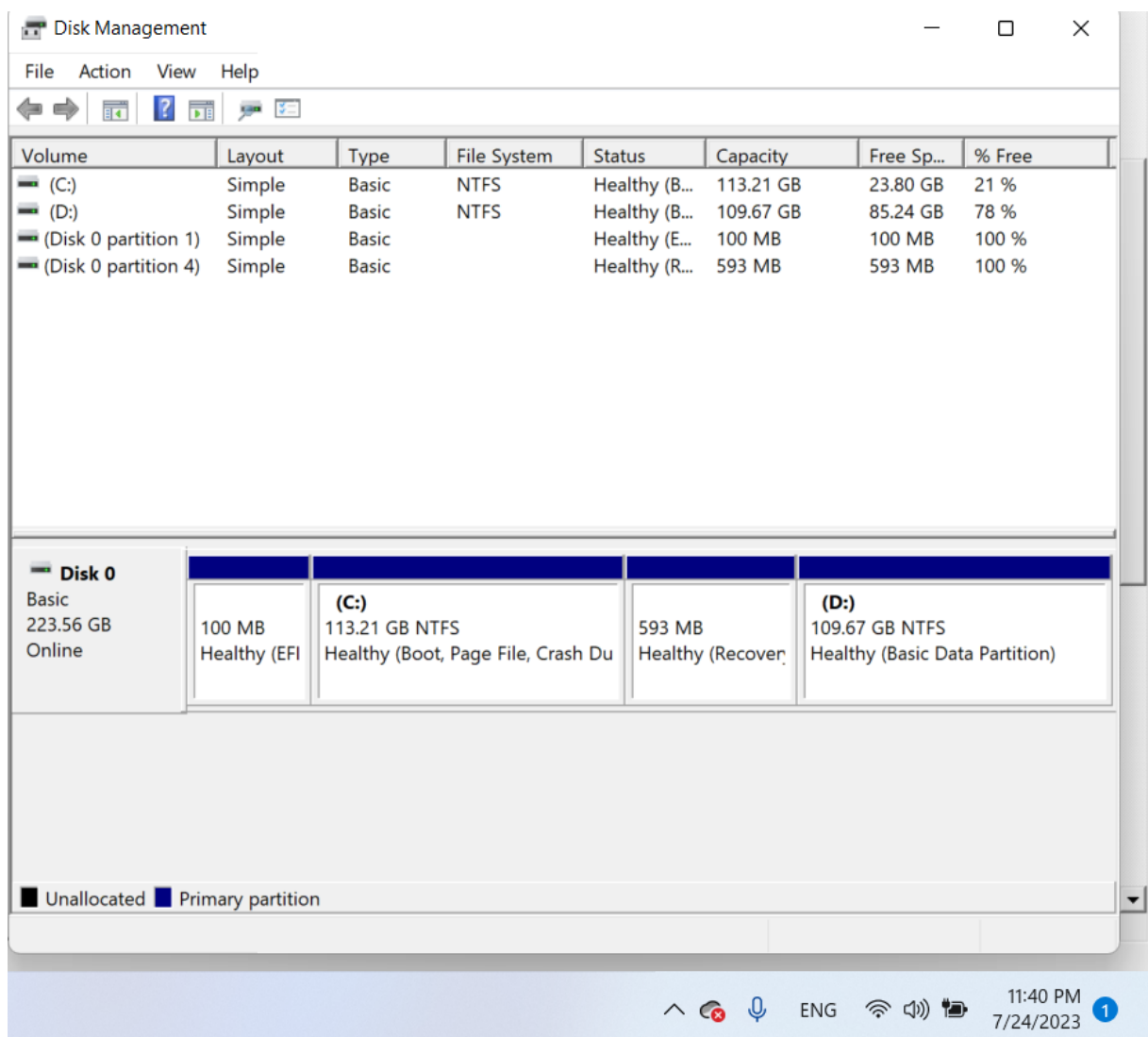
- Dung lượng: 256 GB
- M2. PCIe SSD

- 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

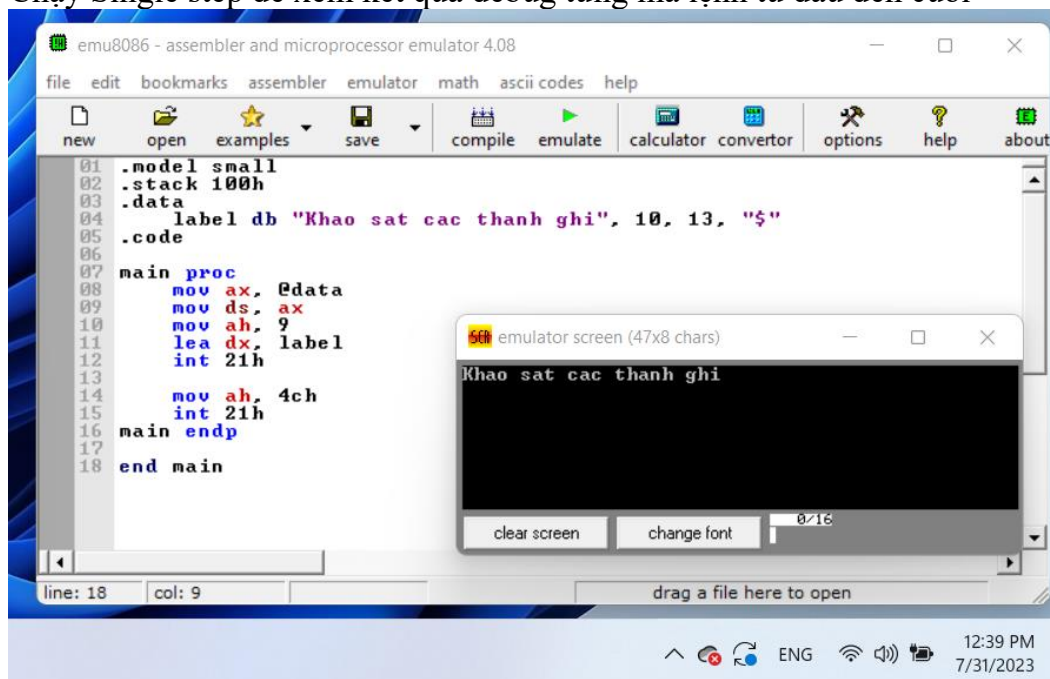
+ Thiết bị vào ra:

- 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`

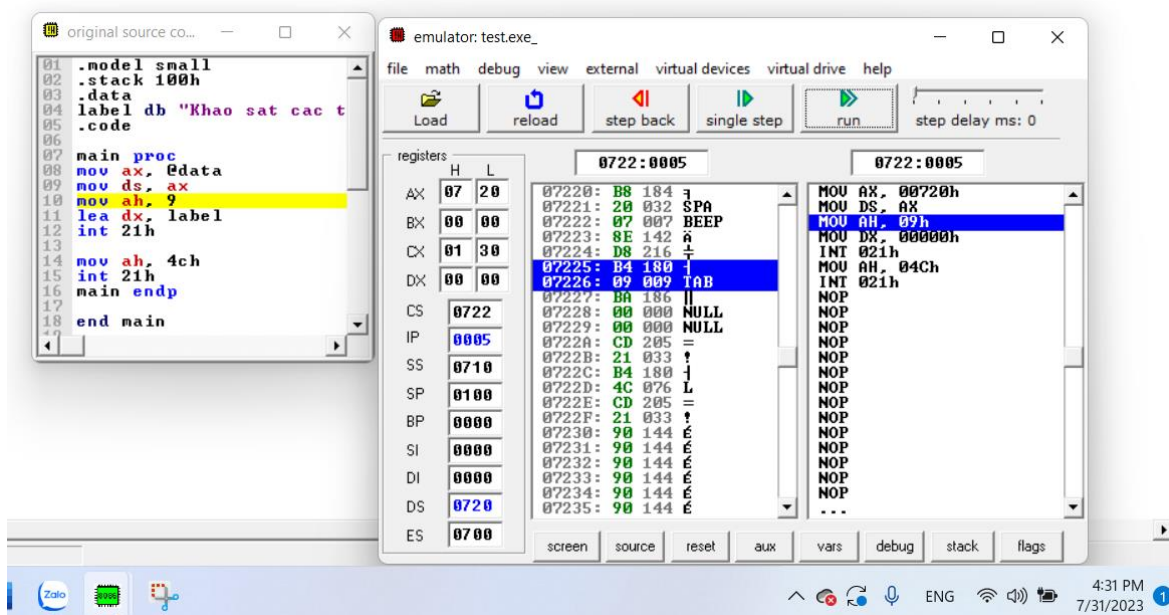
`mov ds, ax`

+ 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

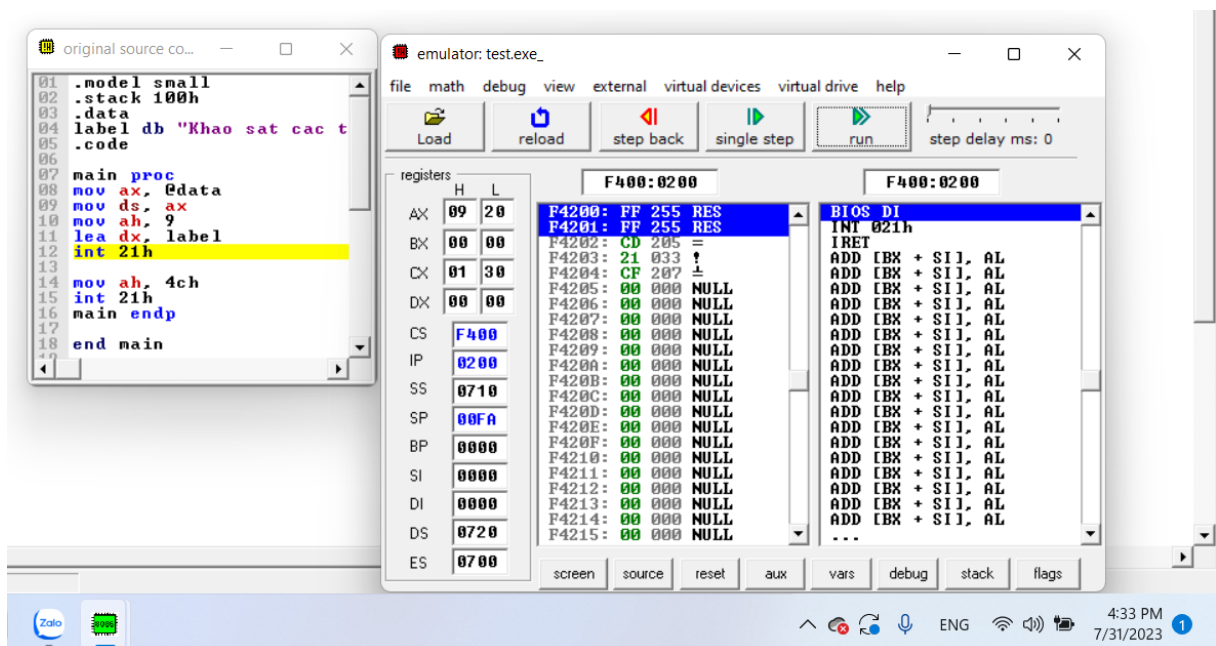
int 21h

+ 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.



-
- The screenshot shows the 8086 emulator interface. The 'original source code' window on the left contains the following assembly code:
- ```

10 mov ah, 9
11 lea dx, label
12 int 21h
13
14 mov ah, 4ch
15 int 21h
16 main endp
17
18 end main
19
20

```
- The 'emulator: test\_exe\_' window on the right shows the execution state. The 'registers' window displays the following values:
- |    | H    | L  |
|----|------|----|
| AX | 09   | 24 |
| BX | 00   | 00 |
| CX | 01   | 30 |
| DX | 00   | 00 |
| CS | 0722 |    |
| IP | 000C |    |
| SS | 0710 |    |
| SP | 0100 |    |
| BP | 0000 |    |
| SI | 0000 |    |
| DI | 0000 |    |
| DS | 0720 |    |
| ES | 0700 |    |
- The 'source' window shows the current instruction: `MOV AH, 04Ch`. The 'stack' window shows the current instruction: `INT 021h`. The 'screen' window shows the current state of the screen.

- 
- The screenshot displays the Z80 emulator interface. On the left, a text editor window titled 'original source co...' shows assembly code with the following content:
- ```

10 mov ah, 9
11 lea dx, label
12 int 21h
13
14 mov ah, 4ch
15 int 21h
16 main endp
17
18 end main
19
20

```
- The main emulator window, titled 'emulator: test_exe_', features a menu bar (file, math, debug, view, external, virtual devices, virtual drive, help) and a toolbar with buttons for Load, reload, step back, single step, run, and a step delay slider set to 0 ms. Below the toolbar, the 'registers' panel is visible, showing the following values:
- | | H | L |
|----|------|----|
| AX | 4C | 24 |
| BX | 00 | 00 |
| CX | 01 | 30 |
| DX | 00 | 00 |
| IP | F400 | |
| CS | 0200 | |
| SS | 0710 | |
| SP | 00FA | |
| BP | 0000 | |
| SI | 0000 | |
| DI | 0000 | |
| DS | 0720 | |
| ES | 0700 | |
- Below the registers, the memory dump is displayed in two columns. The left column shows memory addresses from F4200 to F4215, and the right column shows memory addresses from BIOS D1 to BIOS D1F. The memory dump is as follows:
- | Address | Value | Comment |
|---------|--------|---------|
| F4200 | FF 255 | RES |
| F4201 | FF 255 | RES |
| F4202 | CD 205 | = |
| F4203 | 21 033 | ! |
| F4204 | CF 207 | ± |
| F4205 | 00 000 | NULL |
| F4206 | 00 000 | NULL |
| F4207 | 00 000 | NULL |
| F4208 | 00 000 | NULL |
| F4209 | 00 000 | NULL |
| F420A | 00 000 | NULL |
| F420B | 00 000 | NULL |
| F420C | 00 000 | NULL |
| F420D | 00 000 | NULL |
| F420E | 00 000 | NULL |
| F420F | 00 000 | NULL |
| F4210 | 00 000 | NULL |
| F4211 | 00 000 | NULL |
| F4212 | 00 000 | NULL |
| F4213 | 00 000 | NULL |
| F4214 | 00 000 | NULL |
| F4215 | 00 000 | NULL |
- At the bottom of the emulator window, there are tabs for screen, source, reset, aux, vars, debug, stack, and flags. The 'screen' tab is currently selected, showing a blank screen.

*** 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

I. Giới thiệu đề tài

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

II. 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.

2. 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 hiệu (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,...

[Int 10/AH=01h](#) - VIDEO - SET TEXT-MODE CURSOR SHAPE
[Int 10/AH=02h](#) - VIDEO - SET CURSOR POSITION
[Int 10/AH=03h](#) - VIDEO - GET CURSOR POSITION AND SIZE
[Int 10/AH=04h](#) - VIDEO - READ LIGHT PEN POSITION (except VGA)
[Int 10/AH=04h](#) - HUNTER 16 - GET CURSOR ADDRESS
[Int 10/AH=05h](#) - VIDEO - SELECT ACTIVE DISPLAY PAGE
[Int 10/AX=0500h](#) - VIDEO - Corona/Cordata BIOS v4.10+ - SET GRAPHICS BITMAP BUFFER
[Int 10/AX=050Fh](#) - VIDEO - Corona/Cordata BIOS v4.10+ - GET GRAPHICS BITMAP BUFFER
[Int 10/AX=0580h](#) - VIDEO - PCjr, Tandy 1000 - GET CRT/CPU PAGE REGISTERS

[Int 15/AH=85h](#) - OS HOOK - SysReq KEY ACTIVITY (AT,PS)
[Int 15/AX=8500h](#) - HUNTER 16 - RESTORE POWER MENU (APM)
[Int 15/AX=8501h](#) - HUNTER 16 - SET POWER MENU (APM)
[Int 15/AH=86h](#) - BIOS - WAIT (AT,PS)
[Int 15/AH=86h](#) - HUNTER 16 - GET/SET SCREEN ATTRIBUTE TABLE




[Int 16/AH=00h](#) - KEYBOARD - GET KEYSTROKE
[Int 16/AH=01h](#) - KEYBOARD - CHECK FOR KEYSTROKE
[Int 16/AH=02h](#) - KEYBOARD - GET SHIFT FLAGS
[Int 16/AH=03h](#) - KEYBOARD - SET TYPEMATIC RATE AND DELAY
[Int 16/AH=04h](#) - KEYBOARD - SET KEYCLICK (PCjr only)

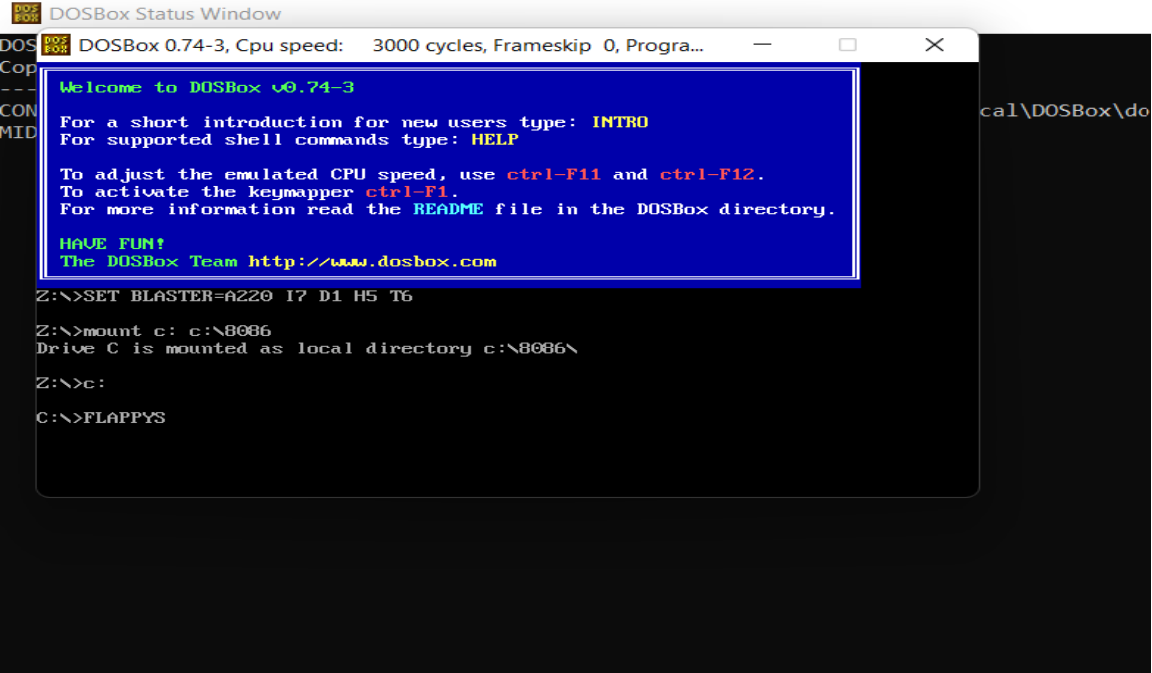
3. Giới thiệu về trình giả lập DOSBox

- 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.

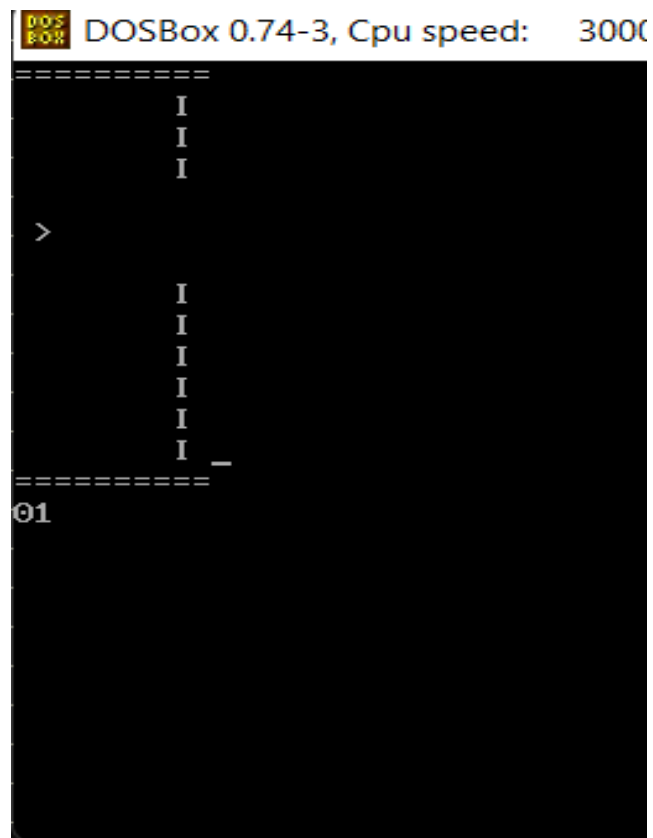
 FLAPPY.ASM	7/27/2023 10:16 PM	ASM File	8 KB
 FLAPPY	7/20/2023 4:27 PM	Application	1 KB
 FLAPPY.OBJ	7/20/2023 4:26 PM	OBJ File	1 KB

The image shows a DOSBox Status Window and a DOSBox Command Prompt. The Status Window is titled "DOSBox Status Window" and displays the following text: "Welcome to DOSBox v0.74-3", "For a short introduction for new users type: INTRO", "For supported shell commands type: HELP", "To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.", "To activate the keymapper ctrl-F1.", "For more information read the README file in the DOSBox directory.", "HAVE FUN!", "The DOSBox Team http://www.dosbox.com". The Command Prompt is titled "DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra..." and displays the following commands and output: "Z:\>SET BLASTER=A220 I7 D1 H5 T6", "Z:\>mount c: c:\8086", "Drive C is mounted as local directory c:\8086\...", "Z:\>c:", "C:\>FLAPPYS".

```
DOSBox Status Window
DOSBox 0.74-3, Cpu speed: 3000 cycles, Frameskip 0, Progra...
Welcome to DOSBox v0.74-3
For a short introduction for new users type: INTRO
For supported shell commands type: HELP
To adjust the emulated CPU speed, use ctrl-F11 and ctrl-F12.
To activate the keymapper ctrl-F1.
For more information read the README file in the DOSBox directory.
HAVE FUN!
The DOSBox Team http://www.dosbox.com

Z:\>SET BLASTER=A220 I7 D1 H5 T6
Z:\>mount c: c:\8086
Drive C is mounted as local directory c:\8086\
Z:\>c:
C:\>FLAPPYS
```

III. GIAO DIỆN CHƯƠNG TRÌNH



IV. 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

```
; data segment
data segment
    assume ds:data
    score_0 db 0
    score_1 db 0
    bird_x db 2
    bird_y db 0
    pipe_x db 11
    pipe_y db 0
    cursor_x db 0
    cursor_y db 0
data ends
```

- **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.

```

code segment
start:

    push ax
    mov ax, data
    mov ds, ax
    pop ax

    mov ah, 00h
    mov al, 03h
    int 10h

```

2. 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 việc render (In ra các thành phần của game lên màn hình console).

a) Xử lý logic game

- Logic ống xuất hiện liên tục:

```

game_loop:
    ; reset ong
    cmp pipe_x, 0
    ja pipe_x_gt_zero

    mov pipe_x, 11

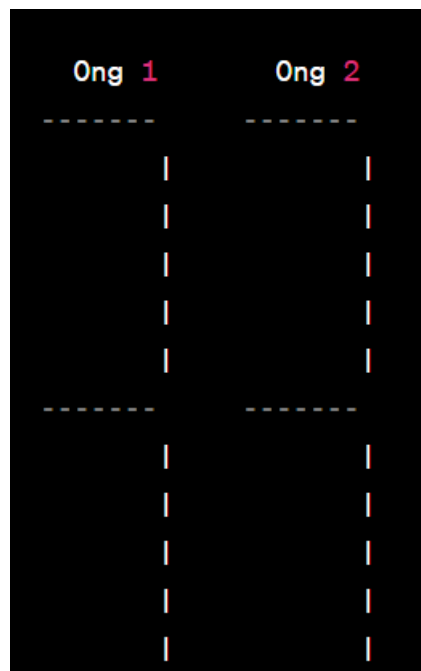
    ; xu ly diem so
    inc score_0
    cmp score_0, 9
    jbe dont_increment_tens
    mov score_0, 0
    inc score_1
    ; set lai vi tri cua ong khi vuot vao tao khe giua ong
    dont_increment_tens:

    cmp pipe_y, 5
    jb pipe_y_gt_5
    mov pipe_y, 0
    jmp pipe_x_gt_zero
pipe_y_gt_5:
    inc pipe_y

pipe_x_gt_zero:
    ; Giam vi tri cua ong lenh di 1 tu phai qua trai
    dec pipe_x

```

- Ban đầu $pipe_x = 11$ (Ở phía cuối cùng bên phải của 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 sang trái. Nếu $pipe_x \leq 0$ 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 \Rightarrow 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 \Rightarrow 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

```

check_key:
; sleep for 250ms (250000 microseconds -> 3D090h)
mov cx, 03h
mov dx, 05150h
mov ah, 86h
int 15h

; Neu khong co du lieu trong bo dem ban phim
mov ah, 0Bh
int 21h
cmp al, 0h
je is_new_key_false

is_new_key_true:
; check xem co phim o bo dem ban phim khong
mov ah, 0
int 16h

; Neu la phim escs
cmp ah, 01h
jne not_exit
mov ah, 4ch
int 21h
not_exit:

; giam vi tri nhan vat di 2
sub bird_y, 2

; dat lai bo dem ban phim
mov ah, 0Ch
int 21h

is_new_key_false:
; Tang vi tri truc tung cua nhan vat
inc bird_y

```

- 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 đếm bàn phím cập nhật phím mới là 3ms = 3s. 015015h = 250ms tức 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 đếm 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):

```

check_collision:
; collision with ground
mov ah, 13
mov bh, [bird_y]
cmp ah, bh
jb render
; check nếu nhân vật ở giữa ống
mov ah, [pipe_x]
mov bh, [bird_x]
cmp ah, bh
jne render
mov ah, [bird_y]
mov bh, [pipe_y]
add bh, 4
cmp ah, bh
je render
inc bh
cmp ah, bh
je render
sub bh, 2
cmp ah, bh
je render

game_over:
mov ah, 4ch
int 21h

```

- 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

b) Xử lý công việc render (In các thành phần game lên màn hình)

- Render phần nền dưới, nền trên, điểm


```

render:
    ; ve nen duoi
    mov dl, 10
    mov ah, 06h
    int 21h
    mov dl, 13
    int 21h
    mov cx, 10
draw_grass_bottom:
    dec cx
    mov dl, '='
    int 21h
    jnz draw_grass_bottom

    ; Tao dong moi
    mov dl, 10
    mov ah, 06h
    int 21h
    mov dl, 13
    int 21h

    ; print score
    mov dl, score_1
    add dl, '0'
    mov ah, 06h
    int 21h
    mov dl, score_0
    add dl, '0'
    int 21h

    ;cursor at x=dl=0, y=dh=0
    mov dl, 00h
    mov dh, 00h
    mov ah, 2h
    mov bh, 0
    int 10h

    ; Ve nen tren
    mov cx, 10
draw_grass_top:
    dec cx
    mov dl, '='
    int 21h
    jnz draw_grass_top

    mov dh, 00h
    int 10h

```

- 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

```

mov cursor_y, 0
render_y:
    inc cursor_y

    mov dl, 10
    mov ah, 06h
    int 21h
    mov dl, 13
    int 21h

    mov cursor_x, 0
    render_x:
        inc cursor_x

        ; render bird
        render_1:
            mov ah, [cursor_x]
            mov bh, [bird_x]
            cmp ah, bh
            jne render_2
            mov ah, [cursor_y]
            mov bh, [bird_y]
            cmp ah, bh
            jne render_2
            mov dl, '>'
            mov ah, 06h
            int 21h
            jmp render_x_end

        ; render pipe
        render_2:
            mov ah, [cursor_x]
            mov bh, [pipe_x]
            cmp ah, bh
            jne render_3
            mov ah, [cursor_y]
            mov bh, [pipe_y]
            add bh, 4
            cmp ah, bh
            je render_3
            inc bh
            cmp ah, bh
            je render_3
            sub bh, 2
            cmp ah, bh
            je render_3
            mov dl, '|'
            mov ah, 06h
            int 21h
            jmp render_x_end

        ; render empty space
        render_3:
            mov dl, ' '
            mov ah, 06h
            int 21h
            jmp render_x_end

    render_x_end:
        cmp cursor_x, 10
        jb render_x

    render_y_end:
        cmp cursor_y, 12
        jb render_y

```

- Sử dụng 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 trở 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 vị trí ống và nhân vật thì in khoảng trắng.
- Đặt thêm nhãn để kết thúc 2 vòng lặp khi `cursor_x > 10` và `cursor_y > 12`.

V. 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>