HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN

MÔN: KIẾN TRÚC MÁY TÍNH

Họ và tên: Triệu Tuấn Anh
Mã sinh viên: B23DCCN053
Lớp: D23CQCN11-B
Giảng viên hướng dẫn: Trần Tiến Công

Hà Nội, 2025

MỤC LỤC

DANH SÁCH HÌNH ẢNH

DANH SÁCH MÃ NGUỒN

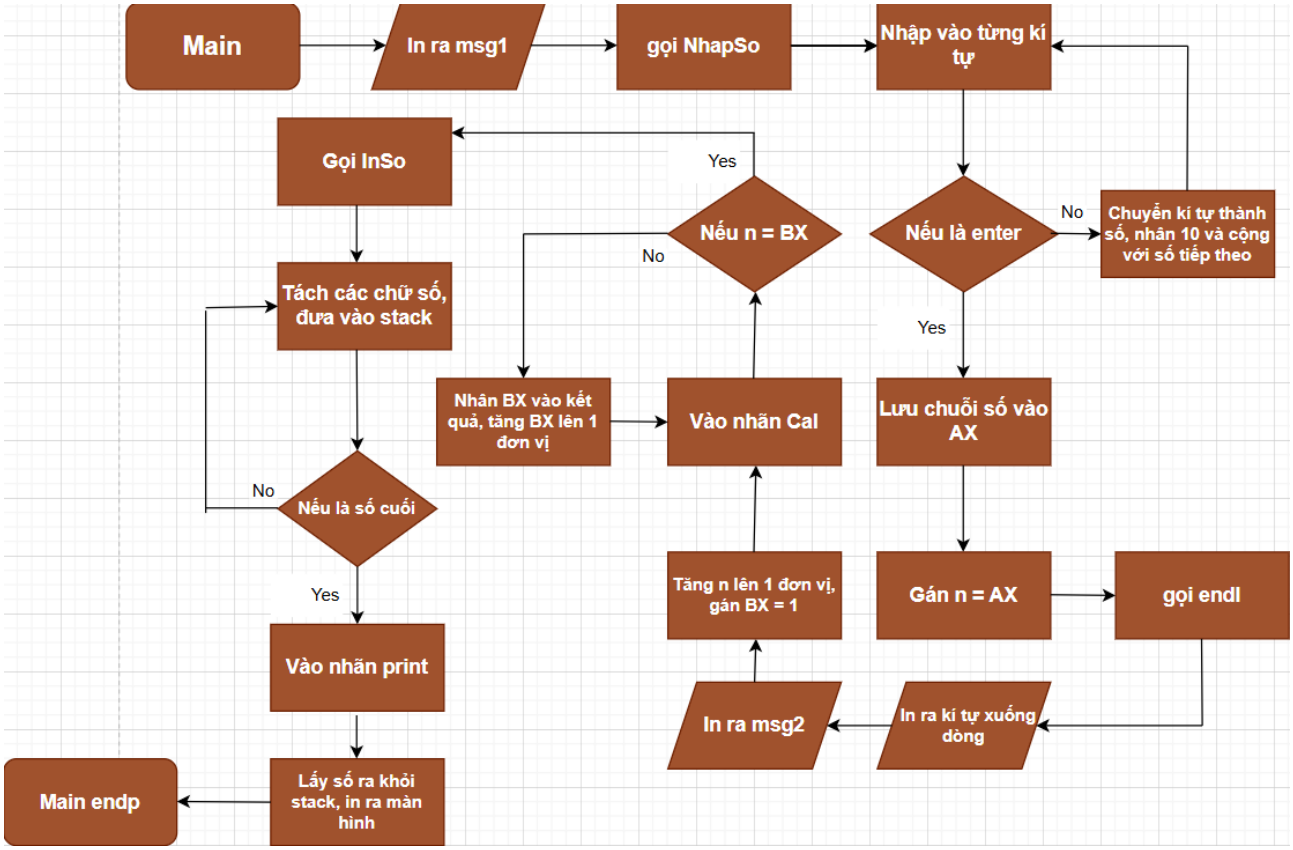
1 Phần cá nhân

1.1 Bài số 1: Lập trình hợp ngữ Assembly

1.1.1 *Câu 1*

Đề bài: Viết chương trình hợp ngữ Assembly cho phép nhập vào một số và in ra màn hình giai thừa của số đó.

Biểu diễn bằng Flowchart: hình 1



Hình 1: Flowchart tính giai thừa 1 số

Mã nguồn assembly 8086:

```
    .model small                ;Khoi tao che do bo nho la small
    .stack 100                  ;Khoi tao kich thuoc ngan xep
    .data                       ;Khoi tao cac bien
        crlf db 13, 10, '$'
        x dw ?
        y dw ?
        n dw ?
        msg1 db 'Nhap vao 1 so: $'
        msg2 db 'Giai thua cua so da nhap la: $'
    .code
    main proc                   ;Ham chinh cua chuong trinh
        mov ax, @data
        mov ds, ax              ;Khoi tao thanh ghi ds

        mov ah, 9               ;In ra msg1
```

```
        lea dx, msg1
        int 21h

        call NhapSo             ;Thuc hien nhap so
        mov n, ax               ;Luu so vua nhap vao n
        call endl               ;Xuong dong

        mov ah, 9               ;In ra msg2
        lea dx, msg2
        int 21h

        inc n                   ;Tang n len 1 don vi
        mov bx, 1               ;Khoi tao thanh ghi bx
        mov ax, 1               ;Khoi tao thanh ghi ax de luu ket qua
        Cal:
            cmp bx, n           ;So sanh bx voi n
            je break            ;Neu bx = n thi thuc hien break
            mul bx              ;Neu bx != n thi nhan ax voi bx, luu vao ax
            inc bx              ;Tang bx len 1 don vi
            jmp Cal             ;Tiep tuc lap de tinh giai thua
        break:
        call InSo               ;In ra ket qua

        mov ah, 4ch             ;Ket thuc chuong trinh
        int 21h
    main endp

    NhapSo proc                 ;Ham con de nhap so
        mov x, 0                ;Khoi tao x = 0
        mov y, 0                ;Khoi tao y = 0
        mov bx, 10              ;Khoi tao bx = 10
        nhap:
            mov ah, 1           ;Nhap 1 ki tu
            int 21h
            cmp al, 13          ;Neu ki tu la dau enter thi chay vao nhapxong
            je nhapxong
            sub al, '0'         ;Neu ki tu khong phai enter thi bien doi thanh so
            mov ah, 0
            mov y, ax           ;Luu so vua nhap vao y
            mov ax, x
            mul bx              ;Lay ax * bx, ket qua luu vao ax
            add ax, y           ;Lay ax + y, ket qua luu vao ax
            mov x, ax
            jmp nhap            ;Tiep tuc lap den khi nhap xong
        nhapxong:
            mov ax, x           ;Luu so da nhap vao thanh ghi ax
        ret
    NhapSo endp

    endl proc                    ;Ham con de xuong dong
        push ax
        push dx

        mov ah, 9                ;In ra ki tu xuong dong
        lea dx, crlf
        int 21h

        pop dx
        pop ax
        ret
    endl endp
```

```
77
78      InSo proc                   ;Ham con de in so
79          push ax
80          push bx
81          push cx
82          push dx
83
84          mov bx, 10              ;Khoi tao bx = 10
85          mov cx, 0               ;Khoi tao cx = 0
86          beforePrint:
87              mov dx, 0
88              div bx              ;Thuc hien ax / bx, phan nguyen luu vao ax, phan du luu vao dx
89              push dx             ;Day phan du vao ngan xep
90              inc cx              ;Tang cx
91              cmp ax, 0           ;Neu ax > 0 thi tiep tuc tach so
92              jg beforePrint
93          print:
94              pop dx              ;Lap phan du ra khoi ngan xep
95              mov ah, 2
96              add dx, '0'         ;Bien doi so thanh ki tu
97              int 21h             ;In ra man hinh
98              loop print          ;Lap cho den khi in xong
99
100         pop dx
101         pop cx
102         pop bx
103         pop ax
104         ret
105     InSo endp
106
107     end main
```

Listing 1: Mã nguồn câu 1

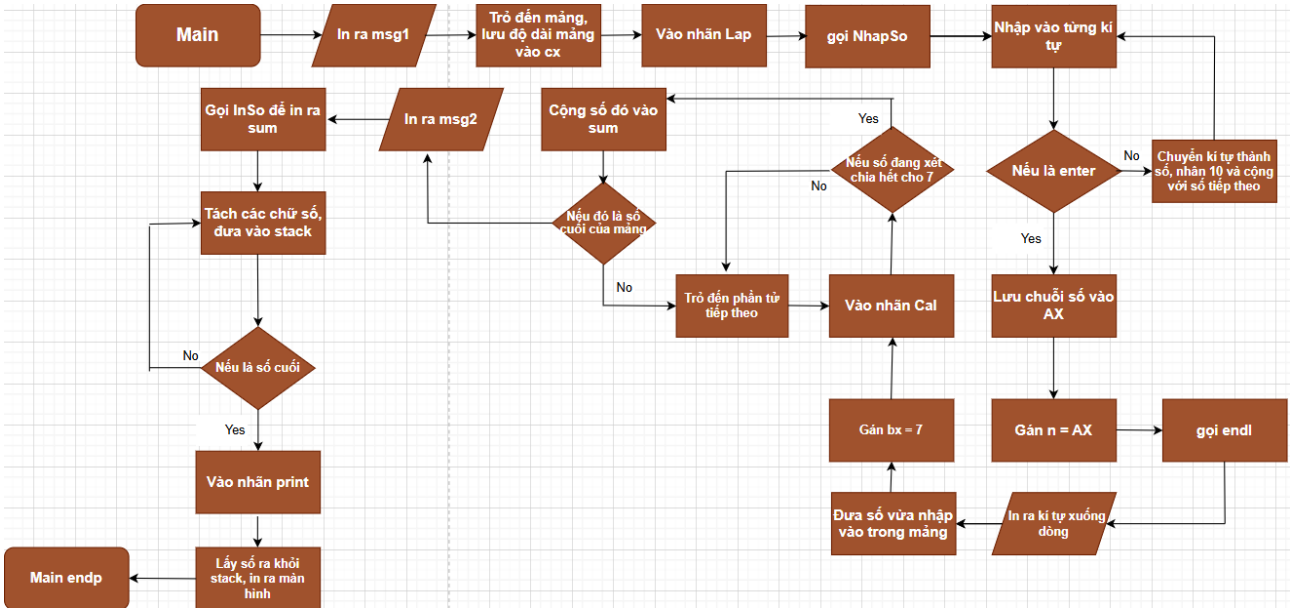
Giao diện hiển thị: hình 2

Hình 2: Giao diện hiển thị câu 1

1.1.2 *Câu 2*

Đề bài: Viết chương trình hợp ngữ cho phép nhập vào một mảng gồm 10 số có hai chữ số. Tính tổng các số chia hết cho 7. In tổng thu được ra màn hình dưới dạng thập phân.

Biểu diễn bằng Flowchart: hình 3



Hình 3: Flowchart tính tổng số chia hết cho 7

Mã nguồn assembly 8086:

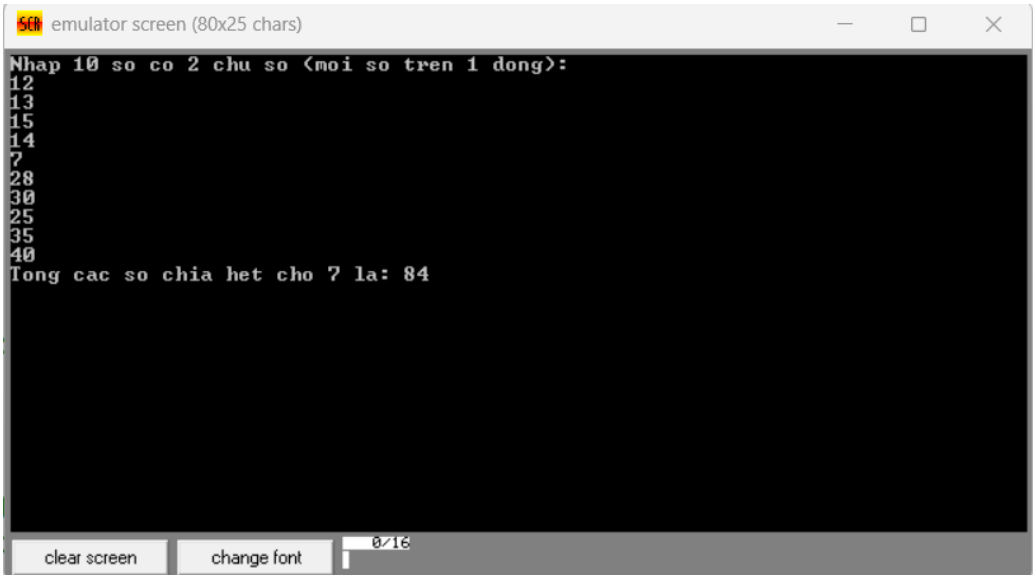
```
    .model small                ;Khoi tao che do bo nho la small
    .stack 100                  ;Khoi tao kich thuoc ngan xep
    .data                       ;Khoi tao cac bien
        crlf db 13, 10, '$'
        x dw ?
        y dw ?
        sum dw ?
        arr dw 10 dup('$')
        msg1 db 'Nhap 10 so co 2 chu so (moi so tren 1 dong):', 0Dh, 0Ah, '$'
        msg2 db 'Tong cac so chia het cho 7 la: $'
    .code
    main proc                   ;Ham chinh cua chuong trinh
        mov ax, @data
        mov ds, ax              ;Khoi tao thanh ghi ds

        mov ah, 9               ;In ra man hinh msg1
        lea dx, msg1
        int 21h

        lea si, arr             ;Thanh ghi SI tro den mang arr
        mov cx, 10              ;Luu do dai arr vao thanh ghi cx
        Lap:
            call NhapSo
            call endl
            mov [si], ax        ;Dua moi so nhap duoc vao trong mang arr
            add si, 2           ;Tang SI tro den phan tu tiep theo cua mang, vi mang dw nen
    phai +2
            loop Lap            ;Thuc hien nhap mang cho den khi du phan tu
```

```
28
29          mov cx, 10              ;Luu do dai arr vao thanh ghi cx
30          lea si, arr             ;Thanh ghi SI tro den mang arr
31          mov bx, 7               ;Luu 7 vao thanh ghi bx
32          Cal:
33              mov dx, 0           ;Gan thanh dx = 0
34              mov ax, [si]        ;Lay ra tung phan tu dua vao thanh ax
35              div bx              ;Lay ax chia bx, phan nguyen luu vao ax, phan du luu vao dx
36              cmp dx, 0           ;So sanh dx voi 0
37              jne continue        ;Neu dx != 0 thi nhay den continue
38              mov ax, [si]        ;Neu dx == 0, tuc la chia het cho 7
39              add sum, ax         ;Neu chia het cho 7 thi cong vao sum
40              continue:
41              add si, 2           ;Tang si tro den phan tu tiep theo
42              loop Cal            ;Thuc hien so sanh den phan tu cuoi cung cua mang
43
44          mov ah, 9               ;In ra msg2
45          lea dx, msg2
46          int 21h
47
48          mov ax, sum             ;Dua gia tri cua sum vao thanh ghi ax
49          call InSo               ;In ra ket qua
50          mov ah, 4ch             ;Ket thuc chuong trinh
51          int 21h
52      main endp
53
54      NhapSo proc                 ;Ham con de nhap so
55          mov x, 0                ;Khoi tao x = 0
56          mov y, 0                ;Khoi tao y = 0
57          mov bx, 10              ;Khoi tao bx = 10
58          nhap:
59              mov ah, 1           ;Nhap 1 ki tu
60              int 21h
61              cmp al, 13          ;Neu ki tu la dau enter thi chay vao nhapxong
62              je nhapxong
63              sub al, '0'         ;Neu ki tu khong phai enter thi bien doi thanh so
64              mov ah, 0
65              mov y, ax           ;Luu so vua nhap vao y
66              mov ax, x
67              mul bx              ;Lay ax * bx, ket qua luu vao ax
68              add ax, y           ;Lay ax + y, ket qua luu vao ax
69              mov x, ax
70              jmp nhap            ;Tiep tuc lap den khi nhap xong
71          nhapxong:
72              mov ax, x           ;Luu so da nhap vao thanh ghi ax
73          ret
74      NhapSo endp
75
76      endl proc                    ;Ham con de xuong dong
77          push ax
78          push dx
79
80          mov ah, 9                ;In ra ki tu xuong dong
81          lea dx, crlf
82          int 21h
83
84          pop dx
85          pop ax
86          ret
87      endl endp
88
```

```
89      InSo proc                   ;Ham con de in so
90          push ax
91          push bx
92          push cx
93          push dx
94
95          mov bx, 10              ;Khoi tao bx = 10
96          mov cx, 0               ;Khoi tao cx = 0
97          beforePrint:
98              mov dx, 0
99              div bx              ;Thuc hien ax / bx, phan nguyen luu vao ax, phan du luu vao dx
100             push dx             ;Day phan du vao ngan xep
101             inc cx              ;Tang cx
102             cmp ax, 0           ;Neu ax > 0 thi tiep tuc tach so
103             jg beforePrint
104         print:
105             pop dx              ;Lap phan du ra khoi ngan xep
106             mov ah, 2
107             add dx, '0'         ;Bien doi so thanh ki tu
108             int 21h             ;In ra man hinh
109             loop print          ;Lap cho den khi in xong
110
111         pop dx
112         pop cx
113         pop bx
114         pop ax
115         ret
116     InSo endp
117
118     end main
```

Listing 2: Mã nguồn câu 2

Giao diện hiển thị: hình 4

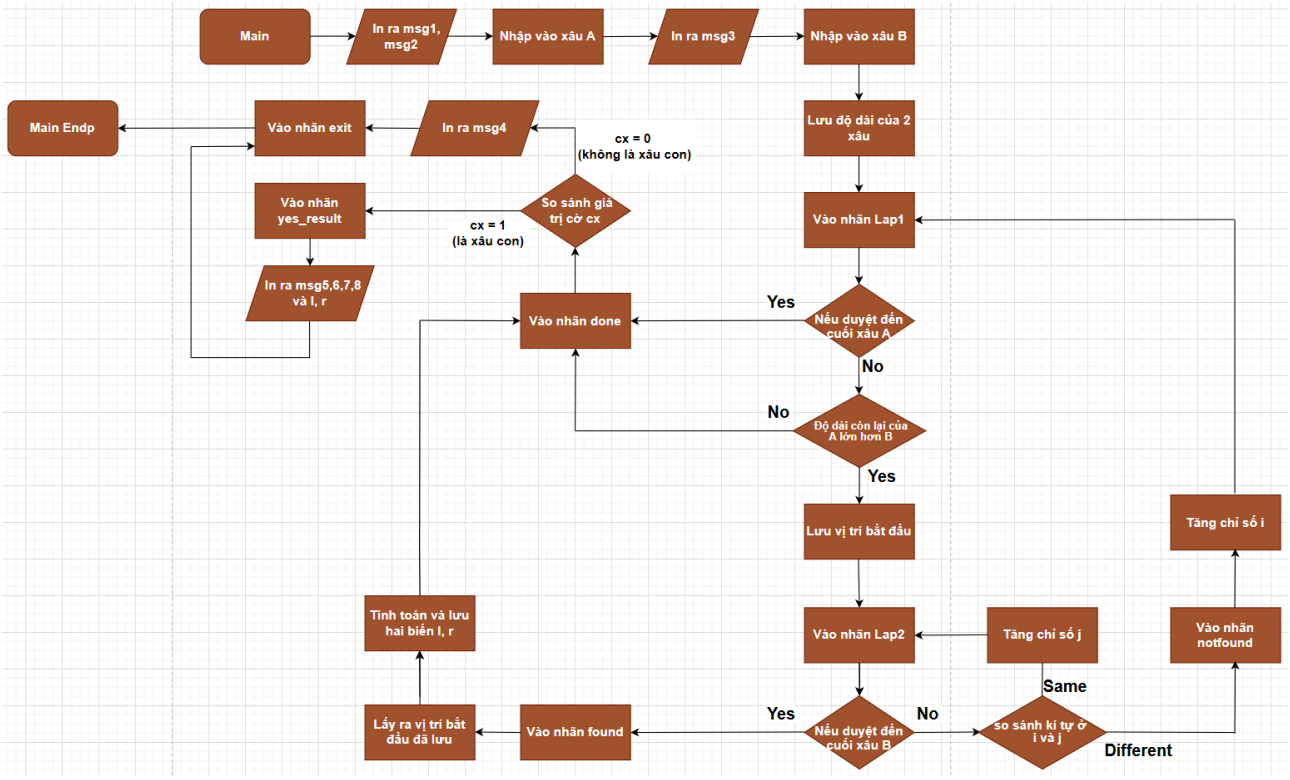


Hình 4: Giao diện hiển thị câu 2

1.1.3 *Câu 3*

Đề bài: Viết chương trình hợp ngữ cho hai chuỗi ký tự A và B có độ dài là n và m (n > m), chỉ ra xâu B có phải là xâu con của xâu A không? Nếu xâu B là xâu con của xâu A thì chỉ ra vị trí xâu B ở xâu A.

Biểu diễn bằng Flowchart: hình 5



Hình 5: Flowchart kiểm tra xâu con

Mã nguồn assembly 8086:

```
    .model small
    .stack 100
    .data
        crlf db 13, 10, '$'
        msg1 db 'Nhap vao 2 xau A, B (luu y xau A dai hon xau B)$'
        msg2 db 'Nhap vao xau A: $'
        msg3 db 'Nhap vao xau B: $'
        msg4 db 'B khong la xau con cua A$'
        msg5 db 'B la xau con cua A$'
        msg6 db 'Vi tri cua B: $'
        msg7 db 'Tu ki tu $'
        msg8 db ' den ki tu $'
        l dw ?
        r dw ?
        strA db 50, ?, 50 dup('$')
        strB db 50, ?, 50 dup('$')
        lenA db ?
        lenB db ?

    .code
    main proc
```

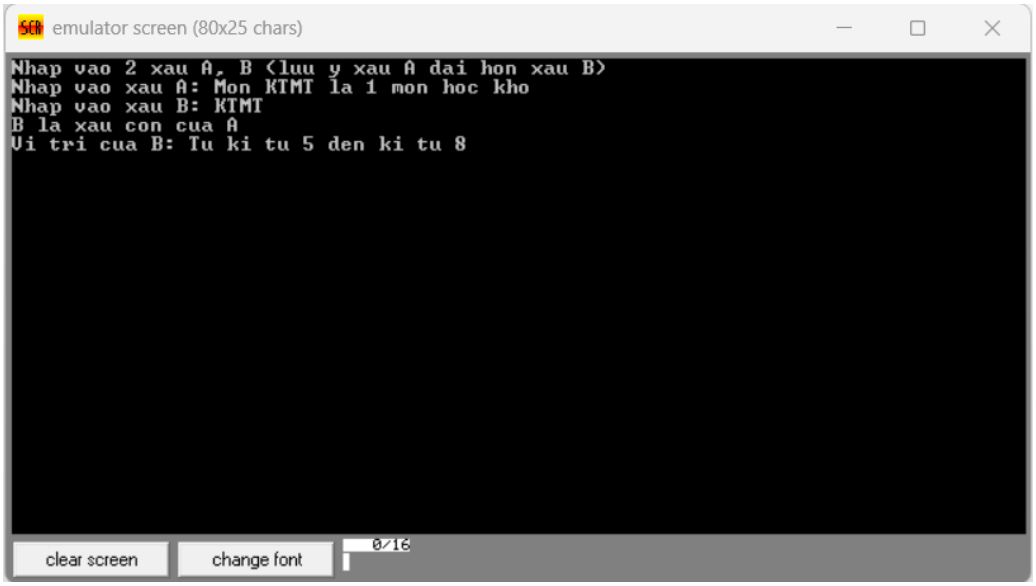
```
22          mov ax, @data
23          mov ds, ax
24
25          mov ah, 9                   ;In ra msg1
26          lea dx, msg1
27          int 21h
28          call endl
29
30          mov ah, 9                   ;In ra msg2
31          lea dx, msg2
32          int 21h
33
34          mov ah, 10
35          lea dx, strA                ;Nhap xau A
36          int 21h
37          call endl
38
39          mov ah, 9                   ;In ra msg3
40          lea dx, msg3
41          int 21h
42
43          mov ah, 10                  ;Nhap xau B
44          lea dx, strB
45          int 21h
46          call endl
47
48          mov al, [strA + 1]          ;Luu do dai strA vao lenA
49          mov lenA, al
50          mov al, [strB + 1]          ;Luu do dai strB vao lenB
51          mov lenB, al
52
53          lea si, strA + 2             ;si tro vao dau xau A
54          mov cx, 0                    ;cx = 0 => chua tim thay
55          mov dl, 0                    ;Chi so i trong A
56
57          Lap1:
58              cmp dl, lenA                ;Neu i >= lenA thi lap xong
59              jnl done
60              mov al, lenA
61              sub al, dl
62              cmp al, lenB
63              jb done                     ;Phan con lai < do dai B => khong the co xau con
64
65              push si                     ;Luu vi tri bat dau
66              lea di, strB + 2            ;di tro vao xau B
67              mov dh, 0                   ;Chi so j trong B
68              mov bx, si                  ;bx tro vao vi tri so sanh trong A
69
70          Lap2:
71              cmp dh, lenB                ;Neu j chay den het strB thi thoa man la xau con
72              je found
73              mov al, [bx]                ;So sanh chi so i va j
74              cmp al, [di]
75              jne notfound                ;Neu i != j thi vao nhan not_found
76              inc bx                      ;Neu i == j thi tang i, j
77              inc di
78              inc dh
79              jmp Lap2
80
81          found:
82              pop si                      ;Lay ra vi tri bat dau
```

```
            mov cx, 1                   ;cx = 1 => da tim thay
            mov ax, si
            sub ax, offset strA + 2     ;Tinh toan vi tri bat dau
            inc ax                      ;Tang ax vi bat dau tinh tu vi tri 1
            mov l, ax                   ;Luu gia tri vao l
            mov ax, l
            mov bl, lenB                ;Lay vi tri dau tien cong voi
            add ax, bx                  ;do dai strB de ra vi tri cuoi cung
            dec ax
            mov r, ax                   ;Luu gia tri vao r
            jmp done

        notfound:
            pop si                      ;Neu khong tim thay, tro den vi tri tiep theo trong strA
            inc si
            inc dl
            jmp Lap1

        done:
            cmp cx, 1                   ;So sanh cx voi 1
            je yes_result               ;Neu cx = 1 => vao nhan yes_result

            mov ah, 9                   ;Neu cx = 0 => in ra msg4
            lea dx, msg4
            int 21h
            jmp exit

        yes_result:
            mov ah, 9                   ;In ra msg5
            lea dx, msg5
            int 21h
            call endl

            mov ah, 9                   ;In ra msg6
            lea dx, msg6
            int 21h

            mov ah, 9                   ;In ra msg7
            lea dx, msg7
            int 21h

            mov ax, l                   ;In ra l
            call InSo

            mov ah, 9                   ;In ra msg8
            lea dx, msg8
            int 21h

            mov ax, r                   ;In ra r
            call InSo

        exit:
        mov ah, 4ch
        int 21h
    main endp

    endl proc
        push ax
        push dx
        mov ah, 9
        lea dx, crlf            ;In ra ki tu xuong dong
```

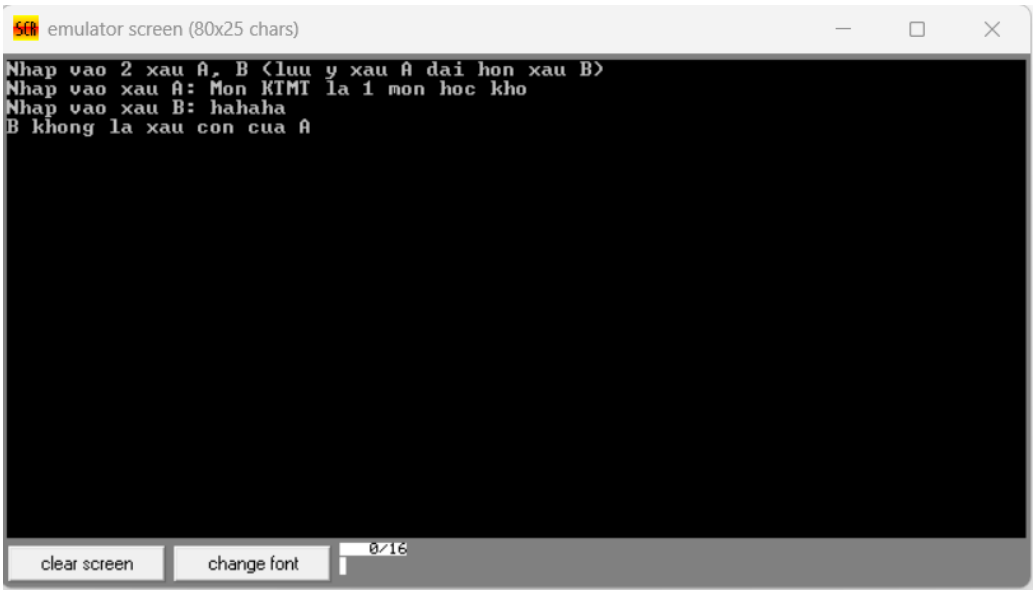
```
144         int 21h
145         pop dx
146         pop ax
147         ret
148     endl endp
149
150     InSo proc                       ;Ham in so nguyen
151         push ax
152         push bx
153         push cx
154         push dx
155
156         mov bx, 10
157         mov cx, 0
158         next_digit:
159             mov dx, 0
160             div bx
161             push dx
162             inc cx
163             cmp ax, 0
164             jg next_digit
165
166         print_digits:
167             pop dx
168             add dl, '0'
169             mov ah, 2
170             int 21h
171             loop print_digits
172
173         pop dx
174         pop cx
175         pop bx
176         pop ax
177         ret
178     InSo endp
179
180     end main
```

Listing 3: Mã nguồn câu 3

Giao diện hiển thị: hình 6 và hình 7



Hình 6: Giao diện hiển thị câu 3 - Trường hợp hợp lệ



Hình 7: Giao diện hiển thị câu 3 - Trường hợp không hợp lệ

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

1.2.1 *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)*

Phần mềm khảo sát:

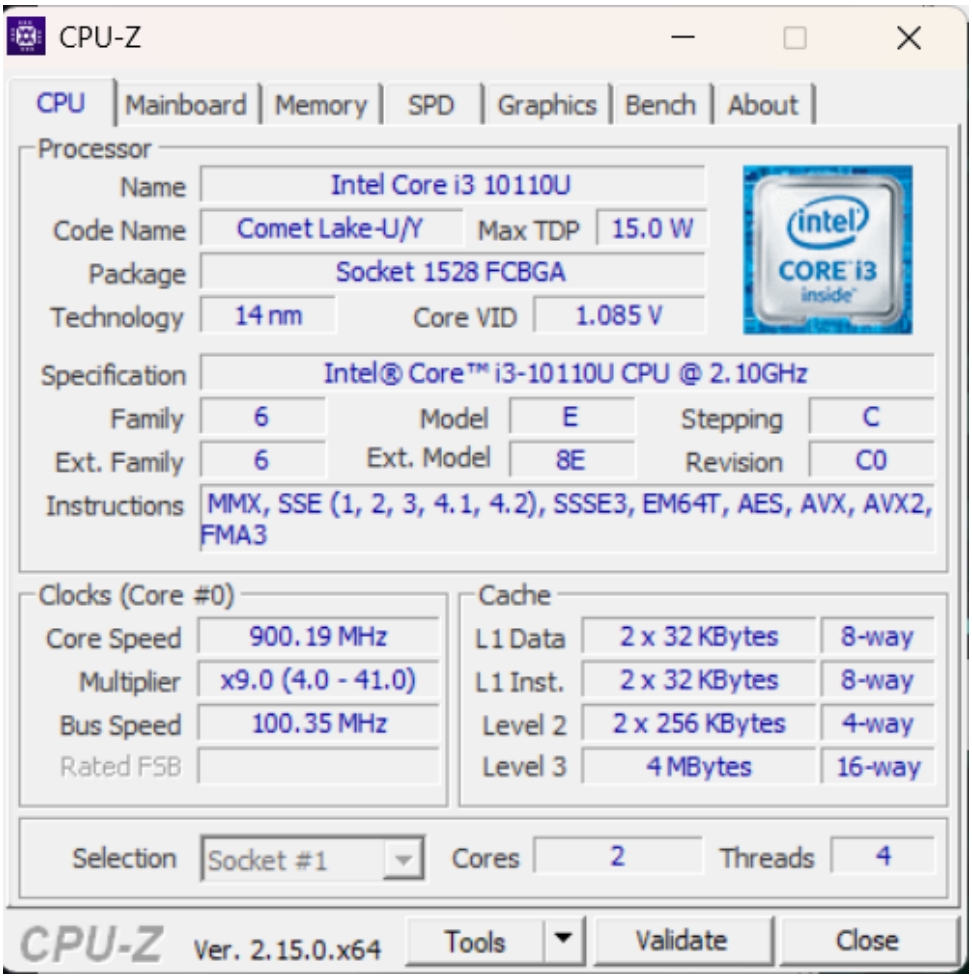
- Sử dụng phần mềm CPU-Z 64-bit Ver. 2.15
- Sử dụng các chương trình quản lí đĩa và thiết bị trên máy tính (Disk Management và Device Manager)

CPU: (Thông số chi tiết thể hiện ở Hình 8)

- **Tên CPU:** Intel Core i3-10110U

- **Kiến trúc:** Comet Lake-U/Y
- **Tiến trình sản xuất:** 14nm
- **TDP tối đa:** 15W (rất tiết kiệm điện, phù hợp cho laptop)
- **Socket:** 1528 FCBGA (dạng hàn chết lên mainboard, phổ biến ở laptop)
- **Số nhân - Số luồng:** 2 nhân, 4 luồng
- **Xung nhịp cơ bản:** 2.10 GHz (thể hiện ở dòng "Specification")
- **Xung nhịp hiện tại:** Khoảng 900 MHz
- **Bus Speed:** 100.35 MHz
- **Tập lệnh hỗ trợ:** MMX, SSE (và các biến thể), AVX, AVX2, FMA3,...

Đây là một CPU tiết kiệm điện, phổ biến trên các laptop mỏng nhẹ, hiệu năng vừa đủ cho các tác vụ văn phòng, học tập, hoặc giải trí nhẹ nhàng.



Hình 8: Giao diện phần mềm CPU-Z - Tab CPU

Cache: (Thông số chi tiết thể hiện ở Hình 9)

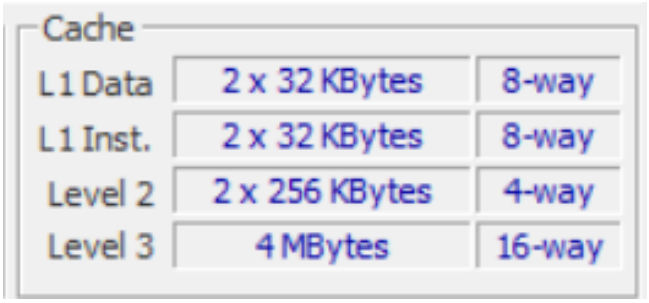
- **L1 Cache (Level 1 Cache):**

  - **Data:** 2 × 32 KB
  - **Instruction:** 2 × 32 KB
  - **Tổ chức:** 8-way associative (8 đường liên kết)
  - *Ghi chú:* Đây là bộ nhớ cache nhanh nhất và nhỏ nhất, chia riêng biệt cho dữ liệu và lệnh. Mỗi nhân có cache riêng.

- **L2 Cache (Level 2 Cache):**
  - **Dung lượng:** 2 × 256 KB
  - **Tổ chức:** 4-way associative
  - *Ghi chú:* Mỗi nhân có 256 KB cache L2 riêng biệt, trung gian giữa L1 và L3.

- **L3 Cache (Level 3 Cache):**
  - **Dung lượng:** 4 MB (chia sẻ giữa các nhân)
  - **Tổ chức:** 16-way associative
  - *Ghi chú:* Đây là cache lớn nhất và được chia sẻ chung cho tất cả các nhân CPU, giúp giảm độ trễ khi trao đổi dữ liệu giữa các nhân.

Cache cấp thấp hơn (L1) thì nhỏ nhưng cực nhanh, trong khi cấp cao hơn (L3) thì lớn hơn nhưng tốc độ chậm hơn. Cấu trúc nhiều cấp như vậy giúp tối ưu tốc độ truy xuất bộ nhớ của CPU.



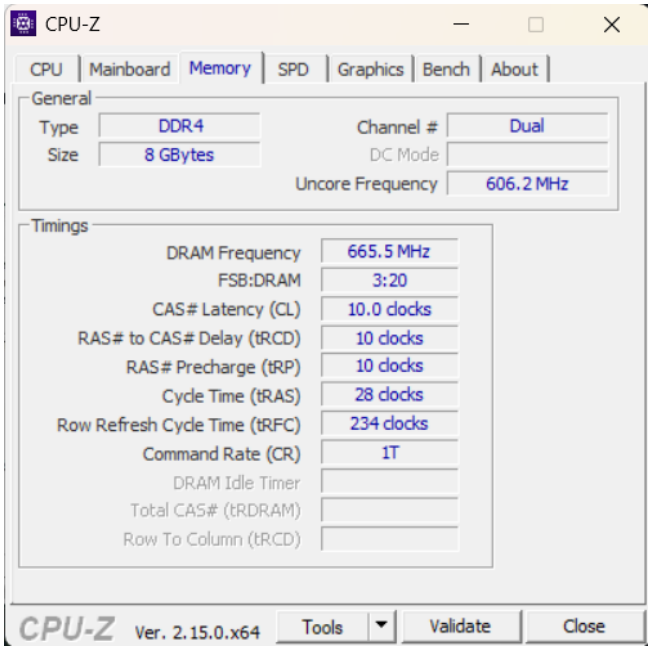
Hình 9: Giao diện phần mềm CPU-Z - Bộ nhớ Cache

RAM: (Thông số chi tiết thể hiện ở Hình 10)

- **Loại RAM:** DDR4
- **Dung lượng:** 8 GB
- **Số kênh (Channel):** Dual (2 kênh)
- **Tần số thực tế (DRAM Frequency):** 665.5 MHz
- **Tỉ lệ FSB:DRAM:** 3:20
- **Độ trễ CAS (CL):** 10.0 clocks
- **RAS to CAS Delay (tRCD):** 10 clocks

- **RAS Precharge (tRP):** 10 clocks
- **Cycle Time (tRAS):** 28 clocks
- **Row Refresh Cycle Time (tRFC):** 234 clocks
- **Command Rate (CR):** 1T

RAM của hệ thống này có cấu hình cân bằng, đáp ứng tốt nhu cầu phổ thông, đồng thời nhờ Dual Channel mà hiệu suất cũng khá ổn định và mượt mà trong các tác vụ đa nhiệm.



Hình 10: Giao diện phần mềm CPU-Z - Bộ nhớ

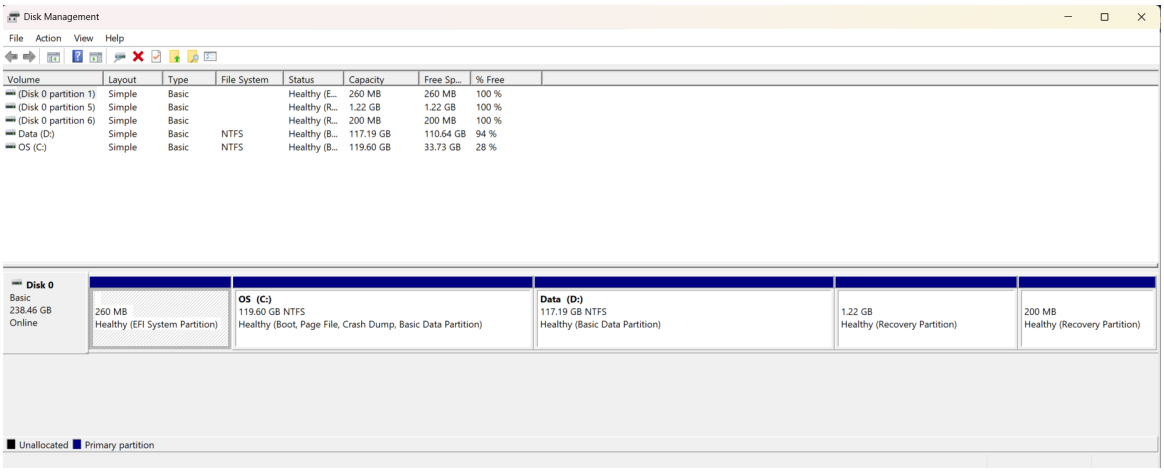
Bộ nhớ ngoài: (Thông số chi tiết thể hiện ở Hình 11)

- **Ổ cứng:** 1 ổ cứng vật lý (Disk 0)
- **Dung lượng tổng:** 238.46 GB
- **Cấu trúc phân vùng:**
  - **OS (C:)** 119.60 GB (NTFS)
    - Phân vùng chứa hệ điều hành Windows.
    - Còn trống: 33.73 GB (khoảng 28% dung lượng).
  - **Data (D:)** 117.19 GB (NTFS)
    - Phân vùng lưu trữ dữ liệu cá nhân.
    - Còn trống: 110.64 GB (khoảng 94% dung lượng).
  - **Phân vùng hệ thống và khôi phục:**
    - 260 MB (EFI System Partition) – phân vùng khởi động.

  - 1.22 GB (Recovery Partition) – phân vùng phục hồi hệ thống.
    Dùng để khôi phục máy về trạng thái ban đầu (Factory Reset) như lúc mới mua.
  - 200 MB (Recovery Partition) – phân vùng phục hồi hệ thống.
    Dùng để sửa lỗi khởi động (Startup Repair), khôi phục hệ thống (System Restore), reset PC (Reset This PC), hoặc truy cập Command Prompt khi hệ điều hành gặp sự cố.

- **Hệ thống tập tin:** NTFS cho các phân vùng chính.

Ổ đĩa đang được phân vùng hợp lý, tách riêng dữ liệu và hệ điều hành. Dung lượng dữ liệu còn trống nhiều, không có phân vùng lỗi, hệ thống đang vận hành ổn định.



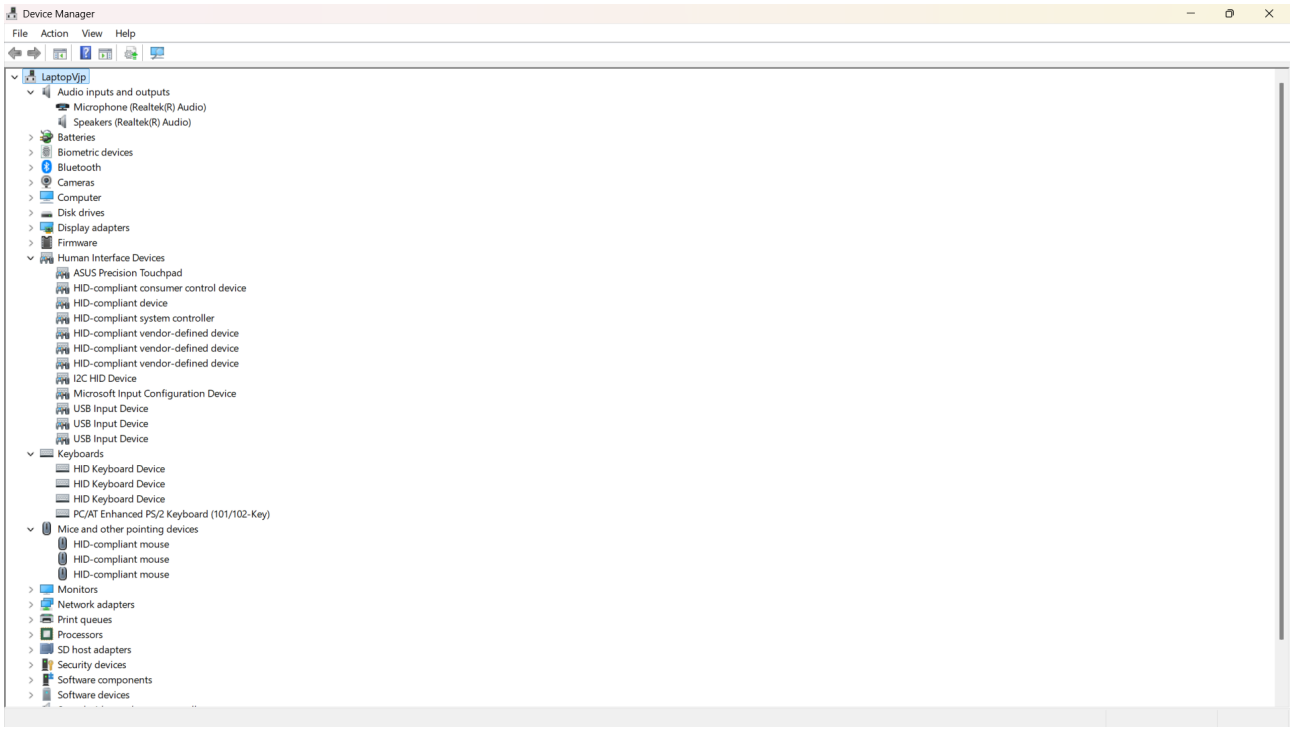
Hình 11: Giao diện chương trình Disk Management

Các thiết bị vào ra: (Thông số chi tiết thể hiện ở Hình 12)

- **Thiết bị âm thanh (Audio inputs and outputs):**
  - **Microphone:** Realtek(R) Audio: microphone tích hợp.
  - **Speakers:** Realtek(R) Audio: loa tích hợp laptop.
- **Thiết bị nhập liệu giao tiếp người dùng (Human Interface Devices):**
  - **ASUS Precision Touchpad:** touchpad của laptop ASUS.
  - **Nhiều thiết bị HID-compliant khác (chuột, bàn di, điều khiển hệ thống, v.v.):** các thiết bị phụ trợ nhập liệu như chuột rời, touchpad, bàn phím rời, các nút multimedia...
- **Bàn phím (Keyboards):**
  - **3 thiết bị HID Keyboard Device:** bàn phím laptop + bàn phím ảo + bàn phím rời.
  - **PC/AT Enhanced PS/2 Keyboard (101/102-Key):** bàn phím vật lý mặc định gắn với mainboard qua cổng PS/2 hoặc emulated PS/2.
- **Chuột và các thiết bị trỏ khác (Mice and other pointing devices):**

– **3 thiết bị HID-compliant mouse:** touchpad, chuột ngoài, hoặc thiết bị trỏ phụ (ví dụ: trackpoint, bút stylus...).

Laptop đang sử dụng đa dạng và hiệu quả các thiết bị vào ra, từ đó mang lại trải nghiệm sử dụng tốt nhất cho người dùng.



Hình 12: Giao diện chương trình Device Manager

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

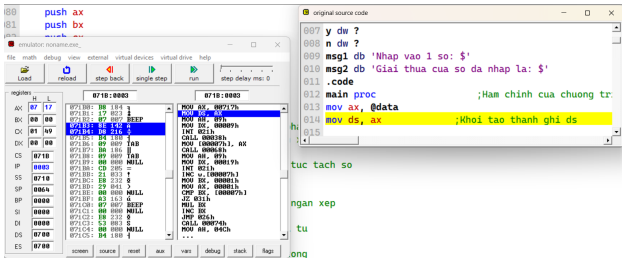
Phần mềm khảo sát:

- Sử dụng phần mềm emu8086 microprocessor emulator

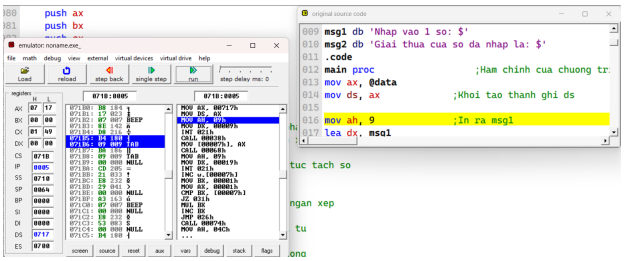
Các bước thực hiện:

- Mở file .asm 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

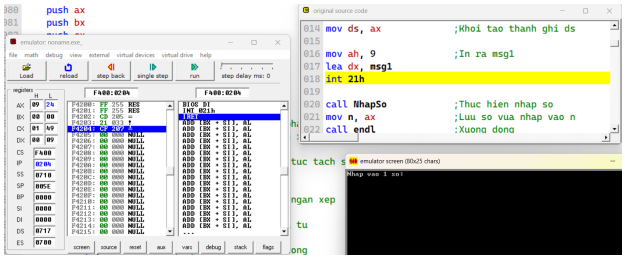
Các kết quả khi chạy single step: Kết quả chi tiết thể hiện ở các bước trong Hình 13



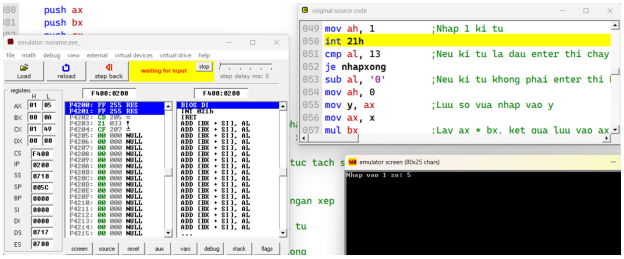
(a) Bước 1



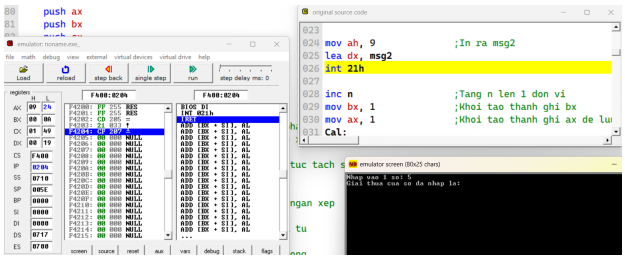
(b) Bước 2



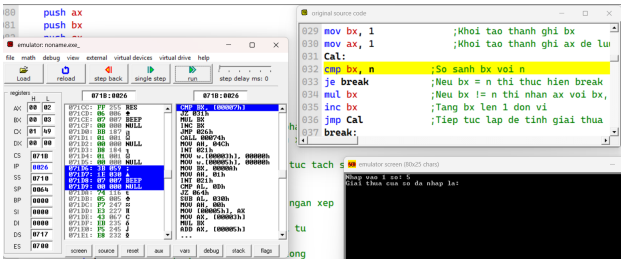
(c) Bước 3



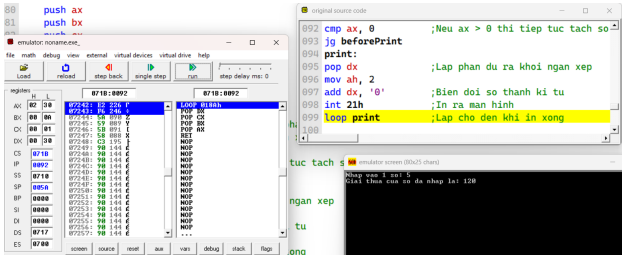
(d) Bước 4



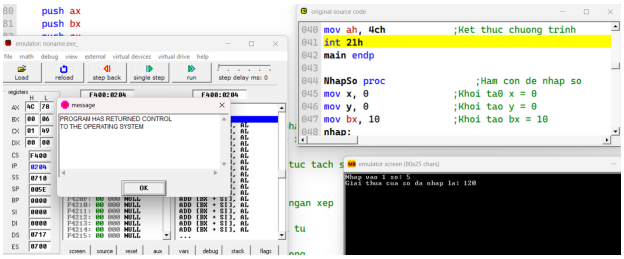
(e) Bước 5



(f) Bước 6



(g) Bước 7



(h) Bước 8

Hình 13: Các bước chạy single step khảo sát nội dung các thanh ghi.

1.2.3 *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.*

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.

- **IP (Instruction Pointer):**
  - **IP** là thanh ghi trỏ đến địa chỉ của lệnh tiếp theo sẽ được thực thi trong mã máy.

  - Khi một chương trình được thực thi, **IP** được cập nhật để trỏ đến lệnh tiếp theo trong mã nguồn, giúp CPU biết lệnh nào sẽ được thực thi tiếp theo.

- **DS (Data Segment) và SI (Source Index):**
  - **DS** là thanh ghi chỉ đến phân đoạn dữ liệu, nơi dữ liệu chương trình được lưu trữ.
  - **SI** thường được sử dụng trong các phép toán dữ liệu và là một trong các thanh ghi chỉ địa chỉ.
  - Khi chương trình yêu cầu truy cập dữ liệu từ bộ nhớ, **DS** và **SI** thường được sử dụng để xác định vị trí của dữ liệu.

- **SS (Stack Segment) và BP (Base Pointer):**
  - **SS** là thanh ghi chỉ đến phân đoạn ngăn xếp (stack segment), nơi lưu trữ các giá trị cục bộ và địa chỉ của các hàm.
  - **BP** thường được sử dụng để trỏ đến địa chỉ cơ sở của ngăn xếp (base of stack).
  - Ngăn xếp được sử dụng để lưu trữ giá trị trung gian và địa chỉ trả về từ các hàm con trong quá trình thực thi chương trình.

- **SP (Stack Pointer):**
  - **SP** là thanh ghi chỉ đến đỉnh của ngăn xếp (top of stack).
  - Khi dữ liệu được đẩy (push) hoặc rút (pop) ra khỏi ngăn xếp, **SP** sẽ thay đổi để chỉ đến vị trí mới nhất trong ngăn xếp.
  - **SP** cũng được sử dụng để cấp phát không gian mới cho dữ liệu trong ngăn xếp.

Cụ thể, trong trường hợp chương trình Assembly cho phép nhập vào một số và in ra màn hình giai thừa của số đó, cơ chế quản lý bộ nhớ như sau:

- **Khởi tạo bộ nhớ:**
  - Khi chương trình bắt đầu, hệ điều hành cấp phát các đoạn mã (code), dữ liệu (data) và ngăn xếp (stack) cho chương trình.
  - Đoạn mã chứa các lệnh chương trình, đoạn dữ liệu chứa các biến, và ngăn xếp dùng để lưu trữ dữ liệu tạm thời.

- **Phân đoạn bộ nhớ:**
  - **CS** (Code Segment) trỏ tới đoạn mã của chương trình.
  - **DS** (Data Segment) trỏ tới đoạn dữ liệu của chương trình.
  - **SS** (Stack Segment) trỏ tới đoạn ngăn xếp của chương trình.

- **Quản lý ngăn xếp:**
  - Ngăn xếp được sử dụng để lưu trữ tạm thời dữ liệu và địa chỉ trả về khi gọi hàm.

– **SP** được khởi tạo với giá trị từ khai báo `.Stack 100`, nghĩa là kích thước ngăn xếp là 100 byte.

Diễn giải nội dung của các câu lệnh trong mã nguồn và ảnh hưởng đến các thanh ghi:

- **Khởi tạo đoạn dữ liệu:**

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

```

DS trỏ tới đoạn dữ liệu, **AX** làm trung gian để chuyển địa chỉ đoạn dữ liệu vào **DS**.

- **Hiển thị thông báo:**

```
    mov ah, 9
    lea dx, msg1
    int 21h

```

hoặc

```
    mov ah, 9
    lea dx, msg2
    int 21h

```

Gắn **AH** = 9, **DX** trỏ tới địa chỉ chuỗi thông báo cần in ra màn hình.

- **Nhập dữ liệu:** Gọi hàm nhập số:

```
    call NhapSo

```

Trong hàm `NhapSo`:

```
    NhapSo proc
    mov x, 0
    mov y, 0
    mov bx, 10
    nhap:
        mov ah, 1
        int 21h
        cmp al, 13
        je nhapxong
        sub al, '0'
        mov ah, 0
        mov y, ax
        mov ax, x
        mul bx
        add ax, y
        mov x, ax
        jmp nhap
    nhapxong:
        mov ax, x
    ret
    NhapSo endp

```

– Gắn **AH** = 1 để nhập ký tự từ bàn phím, ký tự nhập vào lưu trong **AL**.

– Sau khi nhập xong, số nhập được gán vào thanh ghi **AX**.

- **Tính giai thừa:**

```
    inc n
    mov bx, 1
    mov ax, 1
    Cal:
        cmp bx, n
        je break
        mul bx
        inc bx
        jmp Cal

```

AX lưu kết quả phép nhân, **BX** lưu biến đếm sử dụng trong nhãn `Cal`.

- **In ra màn hình:** Gọi hàm in số:

```
    call InSo

```

Trong hàm `InSo`:

```
    InSo proc
    push ax
    push bx
    push cx
    push dx

    mov bx, 10
    mov cx, 0
    beforePrint:
        mov dx, 0
        div bx
        push dx
        inc cx
        cmp ax, 0
        jg beforePrint
    print:
        pop dx
        mov ah, 2
        add dx, '0'
        int 21h
        loop print

    pop dx
    pop cx
    pop bx
    pop ax
    ret
    InSo endp

```

– Đẩy **AX**, **BX**, **CX**, **DX** vào ngăn xếp để bảo vệ dữ liệu.

– Phân tách từng chữ số từ **AX** đưa vào stack, rồi lần lượt lấy ra in ra màn hình.

2 Phần làm nhóm

2.1 Giới thiệu đề tài

Trò chơi *Snake* (hay còn gọi là **rắn săn mồi**) là một trò chơi điện tử đơn giản nhưng kinh điển. Với lối chơi dễ hiểu nhưng đầy thử thách, Snake thường được lựa chọn làm bài tập thực hành trong các môn học về lập trình hoặc hệ thống máy tính.

Trong khuôn khổ môn học **Kiến trúc máy tính**, nhóm chúng em thực hiện đề tài xây dựng trò chơi Snake bằng ngôn ngữ Assembly trên trình giả lập **emu8086** - một môi trường mô phỏng vi xử lý Intel 8086 giúp người học tiếp cận sâu hơn với kiến trúc phần cứng và cách thức vận hành của một hệ thống máy tính ở mức thấp.

Mục tiêu đề tài: Triển khai thành công trò chơi Snake với đầy đủ chức năng cơ bản như: điều khiển rắn bằng phím, sinh mồi ngẫu nhiên,... Ngoài ra, đề tài còn hướng tới việc khai thác hiệu quả các dịch vụ hệ thống qua ngắt BIOS/DOS (như `INT 10h`, `INT 16h`), từ đó nâng cao kỹ năng lập trình Assembly, hiểu rõ hơn về hoạt động của CPU, bộ nhớ, và giao tiếp phần cứng.

Giới hạn đề tài: Đề tài giới hạn ở môi trường **chế độ văn bản** trên giả lập emu8086, không sử dụng đồ họa bitmap hay chế độ đồ họa nâng cao. Tốc độ xử lý và di chuyển của rắn được điều chỉnh bằng kỹ thuật trễ đơn giản, không sử dụng đa luồng hay xử lý song song. Tất cả logic trò chơi đều được viết thủ công bằng Assembly, không sử dụng thư viện hỗ trợ bên ngoài.

Nội dung báo cáo: Dưới đây, chúng em:

- **Phạm Tùng Dương**: làm phần 1 (in ra `msgstart`, `msgover`, viết các hàm `wait_for_enter`, `randomizeMeal` và nhãn `game_over`).
- **Triệu Tuấn Anh**: làm phần 2 (làm nhãn `game_loop` và các nhãn nằm trong `game_loop`).
- **Lê Huy Đức**: làm phần 3 (các hàm `shownewhead`, `move_snake` và `score_plus`).

xin phép được trình bày về đề tài **Lập trình Snake Game trên Emu8086**

2.2 Nội dung tổng quan về đề tài

1. Nội dung chính

Xây dựng trò chơi **Snake (rắn săn mồi)** bằng ngôn ngữ Assembly trên trình giả lập **Emu8086**. Trò chơi mô phỏng cơ chế rắn di chuyển, ăn mồi và phát triển độ dài, đồng thời xử lý va chạm với tường hoặc chính thân rắn để xác định điều kiện kết thúc.

2. Tổng quan về trò chơi

- Snake là trò chơi một người chơi, trong đó người chơi điều khiển một con rắn di chuyển liên tục trên màn hình để tìm và ăn các mồi xuất hiện ngẫu nhiên.
- Mỗi khi ăn được mồi, rắn sẽ dài ra và điểm số tăng lên.

- Trò chơi kết thúc khi rắn đâm vào chính thân mình.
- **Cách chơi:**
  - Điều khiển bằng các phím mũi tên: $\uparrow \downarrow \leftarrow \rightarrow$
  - Rắn di chuyển tự động theo hướng hiện tại
  - Không được quay ngược 180° tức thời
  - Ăn mồi để tăng độ dài và điểm số
- **Các chức năng chính:**
  - Khởi tạo màn hình và vị trí ban đầu
  - Sinh mồi ngẫu nhiên không trùng rắn
  - Cập nhật chuyển động theo điều khiển
  - Kiểm tra va chạm thân/tường
  - Hiển thị điểm số khi kết thúc

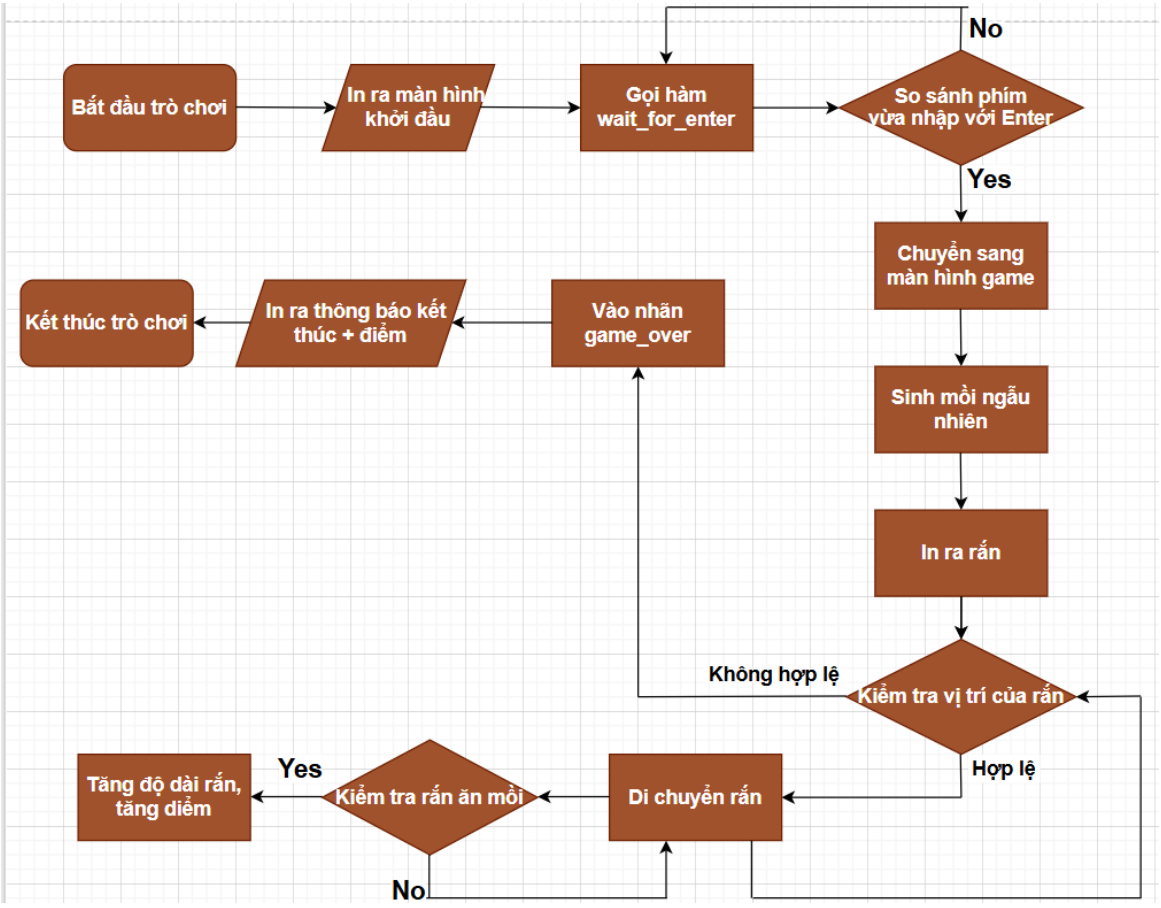
3. Giải thuật và kỹ thuật

- **Cấu trúc dữ liệu:**
  - Mảng tọa độ các đoạn thân rắn
  - Vị trí mồi sinh ngẫu nhiên
- **Thuật toán chính:**
  - Di chuyển: đầu tiến, đuôi xóa (trừ khi ăn mồi)
  - Kiểm tra trùng mồi để tăng điểm
  - Phát hiện va chạm thân/tường
- **Kỹ thuật xử lý:**
  - Ngắt `INT 10h` để vẽ màn hình
  - Ngắt `INT 16h` đọc phím
  - Vòng lặp điều khiển rắn

2.3 Phân tích chương trình

2.3.1 *Lưu đồ thuật toán (Flowchart)*

Flowchart của thuật toán: Hình 14



Hình 14: Flowchart Snake Game

Trong đó:

- **Dữ liệu đầu vào:** Người dùng nhập phím `Enter` để bắt đầu trò chơi, sử dụng các phím mũi tên ↑ ↓ ← → để điều khiển rắn di chuyển.
- **Dữ liệu đầu ra:** Khi bắt đầu, in ra tên trò chơi và thông báo yêu cầu người dùng nhập `Enter` để chơi. Khi kết thúc, in ra thông báo `"Game Over"` và điểm của người chơi.

2.3.2 *Phân tích chi tiết*

Mã nguồn chi tiết của thuật toán tại: `http://bit.ly/3GG2OQN`

Phân tích chương trình:

```
.model small
.stack 100h
.data
    snake dw 10Dh, 10Ch, 10Bh, 10Ah, 150 dup(?)
    s_size db 4,0
    tail dw ?

    left  equ 4Bh
    right equ 4Dh
    up    equ 48h
    down  equ 50h
```

```
12
13          cur_dir db right
14          old_dir db right
15
16          mealX db ?
17          mealY db ?
18
19          score db '0','0','0','0','$'
20
21          ;2 bien msgstart va msgover nhu hinh duoi
22      .code
```

```
msgstart db 5 dup(0Ah), 7 dup(20h)
     db              "   _____                    _                  _____                        ",0Dh,0Ah
     db 7 dup(20h), "  / ____|                  | |                / ____|                       ",0Dh,0Ah
     db 7 dup(20h), " | (___    _ __    __ | | _____      | |  __  __ _ _ __ ___   ___ ",0Dh,0Ah
     db 7 dup(20h), "  \___ \ | ''_ \/ _` |/ / _ \     | | |_ |/ _` | ''_ ` _ \/ _ \",0Dh,0Ah
     db 7 dup(20h), "  ____) || | | | (_| |   <  __/     | |__| | (_| | | | | | |  __/",0Dh,0Ah
     db 7 dup(20h), " |_____/ |_| |_|\__,_|_|\_\___|     \_____|\__,_|_| |_| |_|\___|",0Dh,0Ah,0Ah, 0Ah, 0Ah, 0Ah
     db 22 dup(20h)," Using arrow keys to move the snake                              ",0Dh,0Ah
     db 27 dup(20h)," Press Enter to start. $                                         "
msgover db 5 dup(0Ah), 11 dup(20h)
    db           ,"   _____                          ____                          ",0Dh,0Ah
    db 11 dup(20h),"  / ____|                        / __ \                         ",0Dh,0Ah
    db 11 dup(20h)," | |  __  __ _ _ __ ___   ___  | /  \ |__   _____  _____      ",0Dh,0Ah
    db 11 dup(20h)," | | |_ |/ _` | ''_ ` _ \/ _ \ | |  | |\ \ / / _ \| ___ /     ",0Dh,0Ah
    db 11 dup(20h)," | |__| | (_| | | | | | |  __/ | \__/ | \ \/ /  __/|     \     ",0Dh,0Ah
    db 11 dup(20h),"  \_____|\__,_|_| |_| |_|\___|  \____/   \__/ \___||__|\__\    ",0Dh,0Ah,0Ah,0Ah
    db 28 dup(20h),"  Your score is: $                                              ",0Dh,0Ah
```

Hình 15: Biến `msgstart` và `msgover`

Trong đoạn code trên:

- `.model small`: chọn mô hình bộ nhớ "small", nghĩa là segment code và data riêng biệt, mỗi segment tối đa 64KB.
- `.stack 100h`: đặt vùng stack có kích thước 256 bytes (100h = 256).
- `.data`: bắt đầu khai báo biến.
- `snake`: khai báo con rắn ban đầu có 4 ô từ (10,1) đến (13,1), có 150 ô trống để thêm khi rắn ăn được mồi.
- `s_size`: lưu độ dài rắn, ban đầu là 4.
- `tail`: lưu vị trí đuôi rắn.
- `left, right, up, down`: định nghĩa các biến ứng với phím mũi tên trên bàn phím.
- `cur_dir`, `old_dir`: lưu hướng hiện tại và hướng cũ của rắn, mặc định ban đầu là `right`.
- `mealX`, `mealY`: lưu tọa độ X, Y của mồi.
- `score`: lưu điểm của người chơi, tối đa 4 chữ số.
- `msgstart`: lưu nội dung màn hình bắt đầu.
- `msgover`: lưu nội dung khi trò chơi kết thúc.

```
    main proc
    mov ax, @data
    mov ds, ax

    mov ah, 9
    lea dx, msgstart
    int 21h

    mov ax, 40h
    mov es, ax

    call wait_for_enter

    mov al, 1
    mov ah, 5
    int 10h

    call randomizeMeal
```

Trong đoạn code trên:

- `main proc`: bắt đầu hàm main.
- `mov ax, @data` | `mov ds, ax`: dùng thanh ghi `ax` làm trung gian để gán các giá trị trong `.data` vào thanh ghi `ds`.
- `mov ah, 9` | `lea dx, msgstart` | `int 21h`: sử dụng ngắt 9 để in ra màn hình biến msgstart.
- `mov ax, 40h` | `mov es, ax`: dùng thanh ghi `ax` làm trung gian để gán `40h` cho `es`, nhằm lưu các thông số dòng, cột của màn hình trò chơi.
- `call wait_for_enter`: gọi hàm `wait_for_enter`, hàm này có nội dung như sau:

```
        wait_for_enter proc
            push ax
            wait_loop:
                mov ah, 0
                int 16h
                cmp al, 13
                jne wait_loop
            pop ax
            ret
        wait_for_enter endp

```

  - `push ax`: đẩy thanh ghi `ax` vào `stack` để bảo toàn dữ liệu.
  - `wait_loop`: vào nhãn `wait_loop`
  - `mov ah, 0` | `int 16h`: dùng ngắt 0 nhập kí tự, không in lên màn hình.
  - `cmp al, 13`: so sánh kí tự vửa nhập với kí tự `Enter`.
  - `jne wait_loop`: nếu không phải là kí tự `Enter` thì tiếp tục lặp.

  - `pop ax` | `ret`: trả lại giá trị cho thanh ghi `ax` và kết thúc hàm.

- `mov al, 1` | `mov ah, 5` | `int 10h`: chuyển sang trang màn hình mới.
- `call randomizeMeal`: gọi hàm `randomizeMeal`, hàm này có nội dung như sau:

```
        randomizeMeal proc
            mov ah, 00h
            int 1ah

            mov ax, dx
            xor dx, dx
            mov cx, 25
            div cx
            mov mealY, dl

            mov ah, 00h
            int 1ah

            mov ax, dx
            xor dx, dx
            mov cx, 80
            div cx
            mov mealX, dl

            mov dh, mealY
            mov cx, w.s_size
            xor bx, bx

            check_snake:
                cmp dx, snake[bx]
                je randomizeMeal

                add bx, 2
                loop check_snake

            mov ah, 02h
            mov bh, 01h
            int 10h

            mov al, 04h
            mov bl, 0eh
            mov cx, 1
            mov ah, 09h
            int 10h

            ret
        randomizeMeal endp

```

  - `mov ah, 00h` | `int 1ah` | `mov ax, dx` | `xor dx, dx`: lấy 1 giá trị ngẫu nhiên, dùng `dx` làm trung gian để lưu giá trị đó vào `ax`, sau đó gán lại `dx` = 0.
  - `mov cx, 25` | `div cx` | `mov mealY, dl`: lấy giá trị ngẫu nhiên vừa có chia cho 25 (số dòng) rồi gán cho `mealY`.
  - 7 dòng code tiếp theo: tương tự như `mealY`, tính và gán giá trị cho `mealX`.

- `mov dh, mealY` | `mov cx, w.s_size` | `xor bx, bx`: gán tọa độ của mồi vào `dx`, chiều dài rắn vào `cx`, đưa `bx` về 0 để trỏ vào đoạn đầu của rắn.
- `check_snake`: vào nhãn `check_snake`.
- `cmp dx, snake[bx]` | `je randomizeMeal`: so sánh vị trí mồi với từng đoạn của rắn, nếu trùng nhau thì tạo lại mồi.
- `add bx, 2` | `loop check_snake`: nếu không trùng nhau thì kiểm tra đến đoạn tiếp theo của con rắn.
- Các dòng code còn lại: in ra mồi trên màn hình.

```
game_loop:
    call shownewhead

    mov dx, snake[0]
    mov si, w.s_size
    add si, w.s_size
    sub si, 2
    mov cx, w.s_size
    sub cx, 4
    jz no_death
deathloop:
    cmp dx, snake[si]
    je game_over
    sub si, 2
    dec cx
    jnz deathloop
no_death:
    mov si, w.s_size
    add si, w.s_size
    sub si, 2
    mov ax, snake[si]
    mov tail, ax

    call move_snake

    mov dx, snake[0]
    mov al, mealX
    mov ah, mealY
    cmp ax, dx
    jne hide_old_tail
    mov al, s_size
    inc al
    mov s_size, al
    mov ax, tail
    mov bh, 0
    mov bl, s_size
    add bl, s_size
    sub bl, 2
    mov snake[bx], ax
    call scoreplus
    call randomizeMeal
    jmp no_hide_old_tail

hide_old_tail:
    mov dx, tail

    mov ah, 02h
    int 10h
```

```
49
50          mov al, ' '
51          mov ah, 09h
52          mov bl, 0Fh
53          mov cx, 1
54          int 10h
55
56      no_hide_old_tail:
57          mov ah, 01h
58          int 16h
59          jz no_key
60
61          mov ah, 00h
62          int 16h
63          cur_dir
64          mov cur_dir, ah
65
66      no_key:
67          jmp game_loop
```

Trong đoạn code trên:

– `call shownewhead`**: gọi đến hàm** `shownewhead` **có nội dung như sau**:

```
1           shownewhead proc
2               mov dx, snake[0]
3
4               mov ah, 02h
5               int 10h
6
7               mov al, 002
8               mov ah, 09h
9               mov bl, 0Ah
10              mov bh, 01h
11              mov cx, 1
12              int 10h
13
14              ret
15          shownewhead endp
16
```

∗ `mov dx, snake[0]` | `mov ah, 02h` | `int 10h`: lưu vị trí đoạn đầu của rắn vào `dx`, đồng thời di chuyển con trỏ đến vị trí đó.

∗ Các dòng code còn lại: hiển thị rắn lên màn hình (chiều dài mặc định là 4).

– 5 dòng code tiếp theo: lưu độ dài của rắn và trỏ tới đuôi rắn.

– `sub cx, 4` | `jz no_death`: Kiểm tra nếu độ dài rắn bằng 4 (độ dài mặc định) => rắn không thể chết, nhảy vào nhãn `no_death`.

– `deathloop`: vào nhãn `deathloop`.

– `cmp dx, snake[si]` | `je game_over`: Kiểm tra nếu đầu rắn và thân rắn trùng nhau => rắn chết, nhảy vào nhãn `game_over`. Nhãn này có nội dung như sau:

```
        game_over:
            xor dx, dx

            mov ah, 02h
            int 10h

            mov ah, 9
            lea dx, msgover
            int 21h

            mov ah, 9
            lea dx, score
            int 21h

```

  * `xor dx, dx`: đặt lại `dx` = 0.
  * `mov ah, 02h` | `int 10h`: đưa con trỏ về lại vị trí `dx`.
  * Các dòng còn lại: in ra màn hình thông báo `Game Over` và điểm của người chơi.

– 3 dòng tiếp theo: nếu không trùng nhau, nhảy đến các đoạn còn lại của rắn và tiếp tục kiểm tra.

– `no_death:` vào nhãn `no_death`.

– 5 dòng tiếp theo: xác định lại vị trí đuôi rắn (có thể bị thay đổi nếu nhảy vào `deathloop`), sau đó lưu vào `tail` qua thanh ghi `ax`.

– `call move_snake`: vào hàm `move_snake`. Hàm này có nội dung như sau:

```
        move_snake proc
            mov di, w.s_size
            add di, w.s_size
            sub di, 2

            mov cx, w.s_size
            dec cx

            move_array:
                mov ax, snake[di-2]
                mov snake[di], ax
                sub di, 2
                loop move_array

            getdir:
                cmp cur_dir, left
                je move_left
                cmp cur_dir, right
                je move_right
                cmp cur_dir, up
                je move_up
                cmp cur_dir, down
                je move_down

            get_old_dir:
                mov al, old_dir
```

```
27                  mov cur_dir, al
28                  jmp getdir
29
30              move_left:
31                  cmp old_dir, right
32                  je get_old_dir
33                  mov al, b.snake[0]
34                  dec al
35                  mov b.snake[0], al
36                  cmp al, -1
37                  stop_move
38                  jne stop_move
39                  mov al, es:[4ah]
40                  dec al
41                  mov b.snake[0], al
42                  jmp stop_move
43
44              move_right:
45                  cmp old_dir, left
46                  je get_old_dir
47                  mov al, b.snake[0]
48                  inc al
49                  mov b.snake[0], al
50                  cmp al, es:[4ah]
51                  jb stop_move
52                  mov b.snake[0], 0
53                  jmp stop_move
54
55              move_up:
56                  cmp old_dir, down
57                  je get_old_dir
58                  mov al, b.snake[1]
59                  dec al
60                  mov b.snake[1], al
61                  cmp al, -1
62                  jne stop_move
63                  mov al, es:[84h]
64                  mov b.snake[1], al
65                  jmp stop_move
66
67              move_down:
68                  cmp old_dir, up
69                  je get_old_dir
70                  mov al, b.snake[1]
71                  inc al
72                  mov b.snake[1], al
73                  cmp al, es:[84h]
74                  jbe stop_move
75                  mov b.snake[1], 0
76                  jmp stop_move
77
78              stop_move:
79                  mov al, cur_dir
80                  mov old_dir, al
81              ret
82          move_snake endp
83
```

∗ 4 dòng đầu: tính toán vị trí đuôi rắn và lưu vào `cx`.

* `dec cx`: giảm `cx` đi 1 đơn vị (sử dụng để đếm số vòng lặp).
* `move_array:` vào nhãn `move_array`. Nhãn này có nhiệm vụ lần lượt lưu vị trí trước đó của rắn vào ví trí tiếp theo (di chuyển rắn).
* `getdir:` vào nhãn `getdir`. Nhãn này sẽ kiểm tra hướng đi hiện tại, sau đó gọi vào các nhãn tương ứng.
* `get_old_dir`: vào nhãn `get_old_dir`. Nhãn này sẽ được vào nếu người chơi nhấn các phím khác ngoài phím mũi tên, khi đó rắn sẽ tiếp tục đi theo vị trí cũ.
* `move_left`: vào nhãn `move_left`.
* `cmp old_dir, right | je get_old_dir`: nếu rắn đang sang phải, tiếp tục đi theo hướng cũ (vì rắn không thể quay 180°).
* 3 dòng tiếp theo: nếu rắn đang đi lên/xuống, trừ tọa độ x của đầu rắn đi 1 đơn vị (chuyển hướng rắn sang trái).
* `cmp al, -1 | jne stop_move`: nếu tọa độ đầu rắn vẫn còn ở trong màn hình thì vào nhãn `stop_move`.
* 4 dòng tiếp theo: nếu tọa độ đầu rắn ra khỏi biên trái màn hình thì đưa đầu rắn sang biên bên phải (tạo hiệu ứng đi xuyên tường), rồi nhảy vào nhãn `stop_move`.
* `move_right | move_up | move_down`: các nhãn này có logic xử lí tương tự như nhãn `move_left`.
* `stop_move`: vào nhãn `stop_move`. Nhãn này có nhiệm vụ lưu hướng đi hiện tại vào hướng đi cũ (được sử dụng khi người dùng nhấn trùng hướng di chuyển của rắn).

– `mov dx, snake[0] | mov al, mealX | mov ah, mealY`: lưu tọa độ đầu rắn vào `dx`, tọa đồ mồi vào `ax`.

– `cmp ax, dx | jne hide_old_tail`: nếu tọa độ đầu rắn khác tọa độ mồi (rắn chưa ăn được mồi) thì vào nhãn `hide_old_tail`.

– 9 dòng tiếp theo: nếu rắn ăn được mồi thì tăng độ dài rắn, đồng thời thêm tọa độ đuôi mới cho rắn.

– `call scoreplus | call randomizeMeal`: gọi tới các hàm `randomizeMeal` để cập nhật mồi mới và `scoreplus` để cộng thêm điểm cho người chơi. Trong đó, hàm `scoreplus` có nội dung như sau:

```
1         scoreplus proc
2             mov al, score[3]
3             inc al
4             cmp al, '9'
5             jg inc_second
6             mov score[3], al
7             ret
8
9         inc_second:
10            mov score[3], '0'
```

```
            mov al, score[2]
            inc al
            cmp al, '9'
            jg inc_third
            mov score[2], al
            ret

        inc_third:
            mov score[2], '0'
            mov al, score[1]
            inc al
            cmp al, '9'
            jg inc_fourth
            mov score[1], al
            ret

        inc_fourth:
            mov score[1], '0'
            mov al, score[0]
            inc al
            mov score[0], al
            ret
        scoreplus endp

```

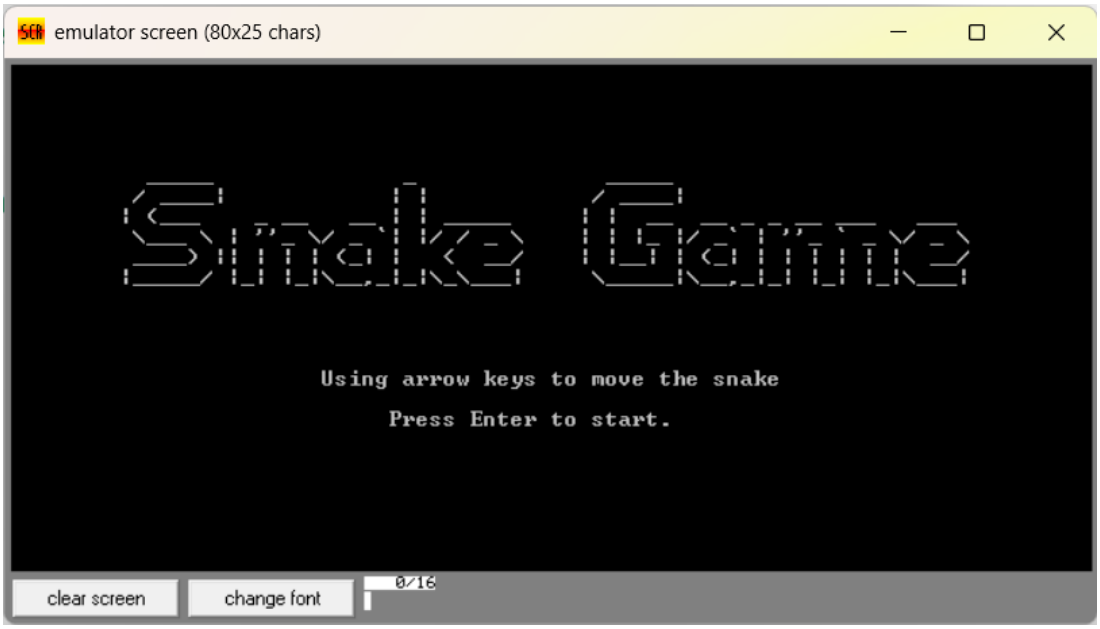
* `mov al, score[3]` | `inc al`: lưu số cuối vào `al`, đông thời tăng giá trị số đó lên 1 đơn vị (vì rắn vừa ăn được mồi).
* `cmp al, '9'` | `jg inc_second`: so sánh số đó với 9, nếu lớn hơn thì vào nhãn `inc_second`.
* `mov score[3], al` : nếu số nhỏ hơn 9 thì lưu kết quả và kết thúc hàm.
* `inc_second` | `inc_third` | `inc_fourth`: các nhãn này có nội dung tương tự như phần trên, lần lượt cộng điểm cho số hàng chục, trăm và nghìn.

```
    mov ah, 4ch
    int 21h
```

Cuối cùng, 2 dòng code trên được chạy ngay sau nhãn `game_over` để kết thúc chương trình.

2.4 Kiểm tra giao diện chương trình

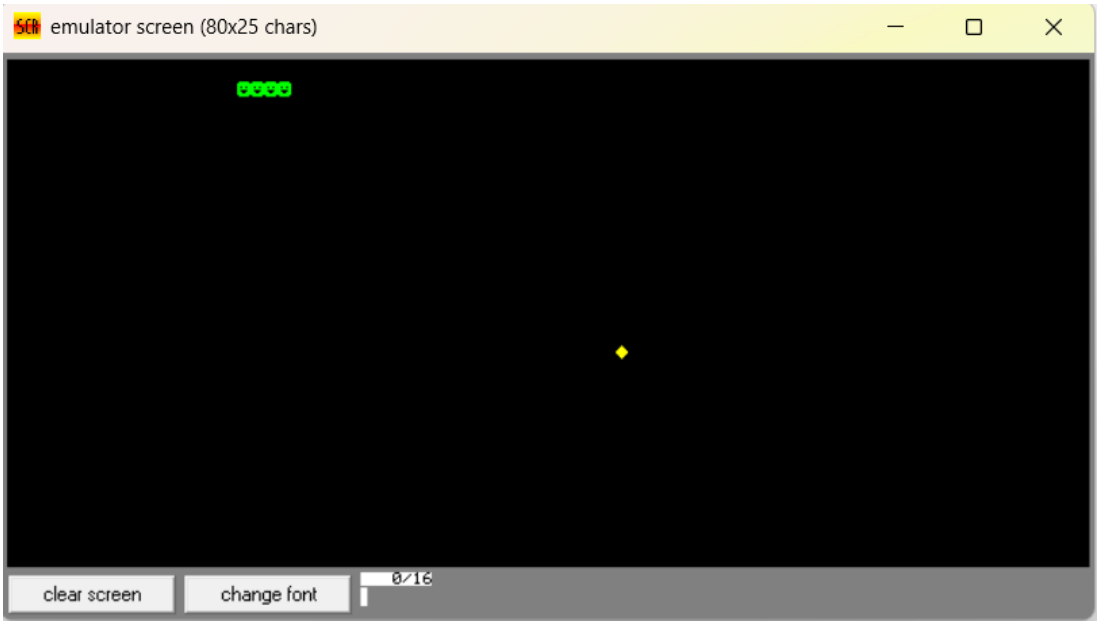
Trong phần này, nhóm chúng em sẽ trình bày kết quả khi chạy chương trình:



Hình 16: Giao diện khởi đầu game

Màn hình khởi đầu bao gồm:

- **Tiêu đề game:** Dòng chữ "Snake Game"được thiết kế bắt mắt.
- **Hướng dẫn:** Thông báo yêu cầu người chơi nhấn phím `Enter` để bắt đầu.
- **Tính năng:** Màn hình sẽ chuyển sang giao diện chính khi người dùng nhấn đúng phím `Enter`.



Hình 17: Giao diện chính của game

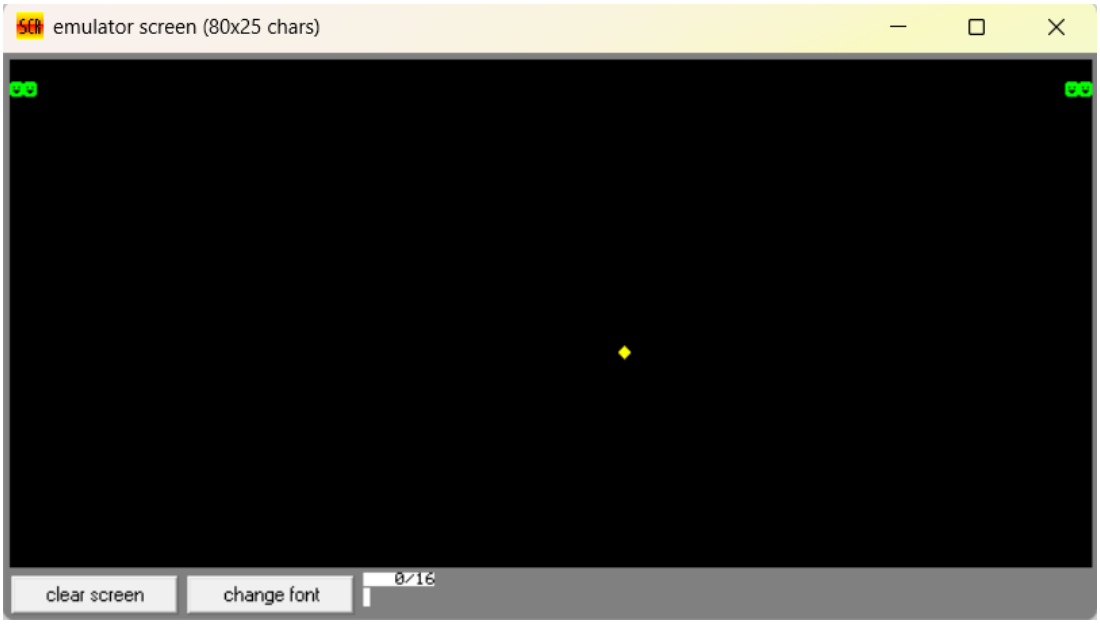
Giao diện chơi game bao gồm:

- **Con rắn:**

  - Độ dài ban đầu: 4 ô.
  - Màu sắc mặc định: Xanh lá (■).

- **Mồi:**
  - Kích thước: 1 ô.
  - Màu sắc mặc định: Vàng (■).

- **Điều khiển:** Sử dụng các phím mũi tên ↑ ↓ ← → để di chuyển



Hình 18: Cơ chế xuyên tường

Đặc điểm nổi bật:

- Khi rắn va chạm với tường, thay vì kết thúc game, rắn sẽ xuất hiện ở phía đối diện.
- Cơ chế này tạo độ khó vừa phải và tăng tính thú vị cho trò chơi.

Hình 19: Màn hình kết thúc game

Màn hình kết thúc bao gồm:

- **Trạng thái cuối:** Vị trí rắn và mồi khi game kết thúc.
- **Thông báo:** Dòng chữ "Game Over"nổi bật.
- **Điểm số:** Hiển thị tổng điểm người chơi đạt được.

KẾT LUẬN

Sau quá trình học tập và thực hiện, dự án đã hoàn thành với hai phần chính:
(1) ba bài tập cá nhân giúp củng cố kiến thức nền tảng
(2) bài tập nhóm là trò chơi *Snake Game* được lập trình bằng ngôn ngữ Assembly, thể hiện sự phối hợp và vận dụng kỹ năng lập trình cấp thấp trong thực tiễn.

Dự án không chỉ giúp rèn luyện tư duy logic, kỹ năng xử lý sự kiện và quản lý bộ nhớ, mà còn nâng cao tinh thần làm việc nhóm, trách nhiệm và sáng tạo trong quá trình phát triển phần mềm.

Chúng em xin chân thành cảm ơn thầy đã hướng dẫn tận tình và tạo điều kiện để thực hiện bài tập lớn này.

Mọi góp ý và nhận xét từ thầy và các bạn sẽ là nguồn động lực để chúng em tiếp tục hoàn thiện kỹ năng trong học tập và công việc sau này.

Hà Nội, tháng 5 năm 2025