

BÀI THỰC HÀNH SỐ 1

LẬP TRÌNH HỢP NGỮ VỚI EMULATOR 8086

1. MỤC TIÊU

Sau khi kết thúc bài thực hành này, sinh viên có thể:

- Sử dụng công cụ 8086 Emulator để khảo sát các lệnh Intel-8086, lập trình hợp ngữ.
- Viết đúng cấu trúc của chương trình hợp ngữ dạng tái định (EXE).
- Đọc hiểu và sửa lỗi chương trình

2. KIẾN THỨC CHUẨN BỊ

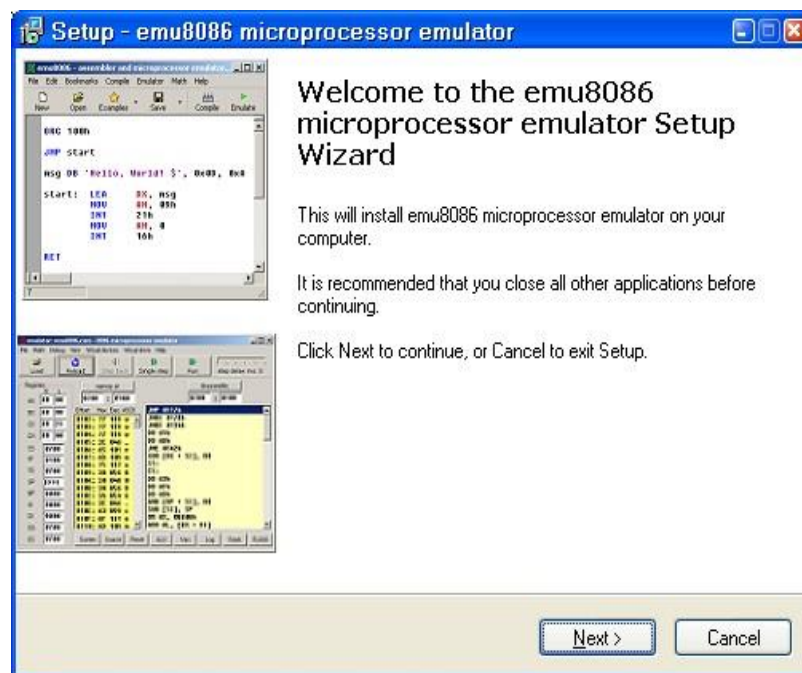
- Hướng dẫn cách cài đặt và sử dụng phần mềm 8086 Emulator
- Các thao tác cơ bản trên hệ điều hành Windows.
- Cấu trúc chương trình hợp ngữ dạng EXE.

3. NỘI DUNG THỰC HÀNH

3.1. Cài đặt và hướng dẫn sử dụng Emulator 8086

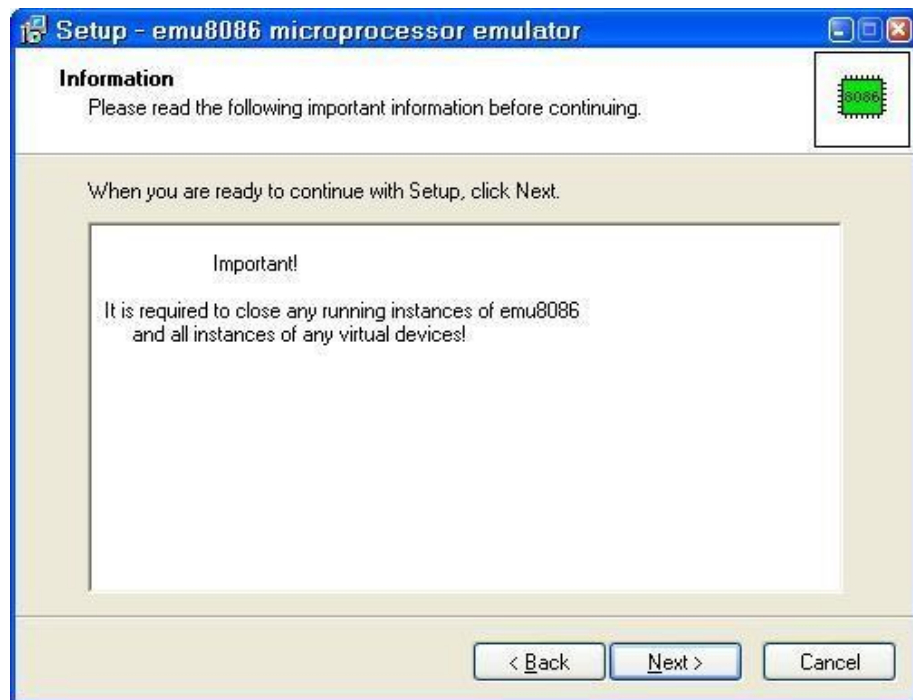
3.1.1. Cách cài đặt

Bước 1: Chạy file Setup.exe trong thư mục cài đặt. Nhấn Next để tiếp tục



Hình 1. Màn hình bắt đầu quá trình cài đặt

Bước 2: Trình cài đặt hiện một số thông tin chú ý, nhấn nút Next để bỏ qua.



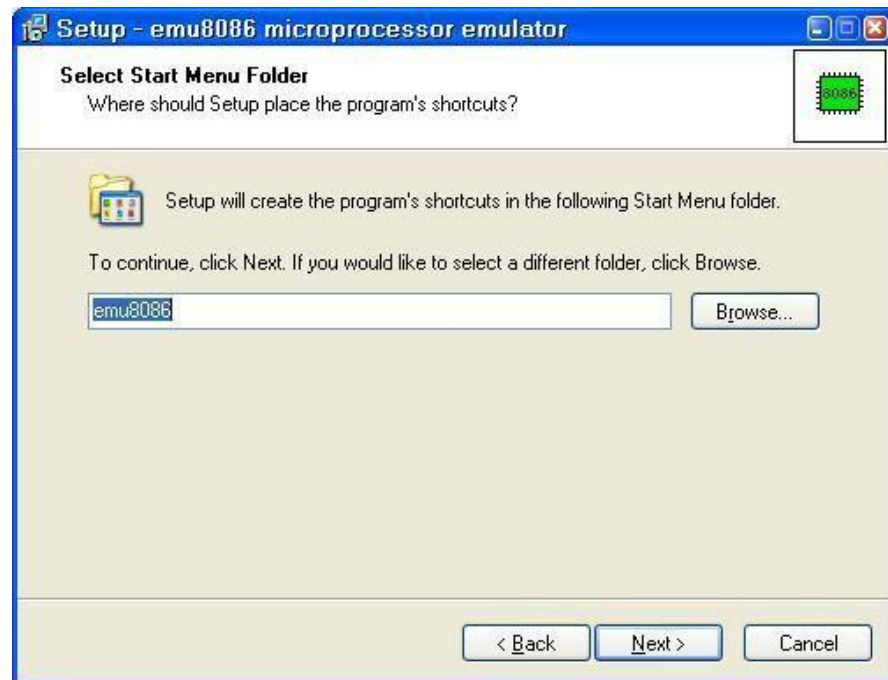
Hình 2. Thông tin chú ý trong quá trình cài đặt

Bước 3: Chọn đường dẫn đến thư mục cài đặt, mặc định là C:\emu8086. Nhấn Next để tiếp tục.



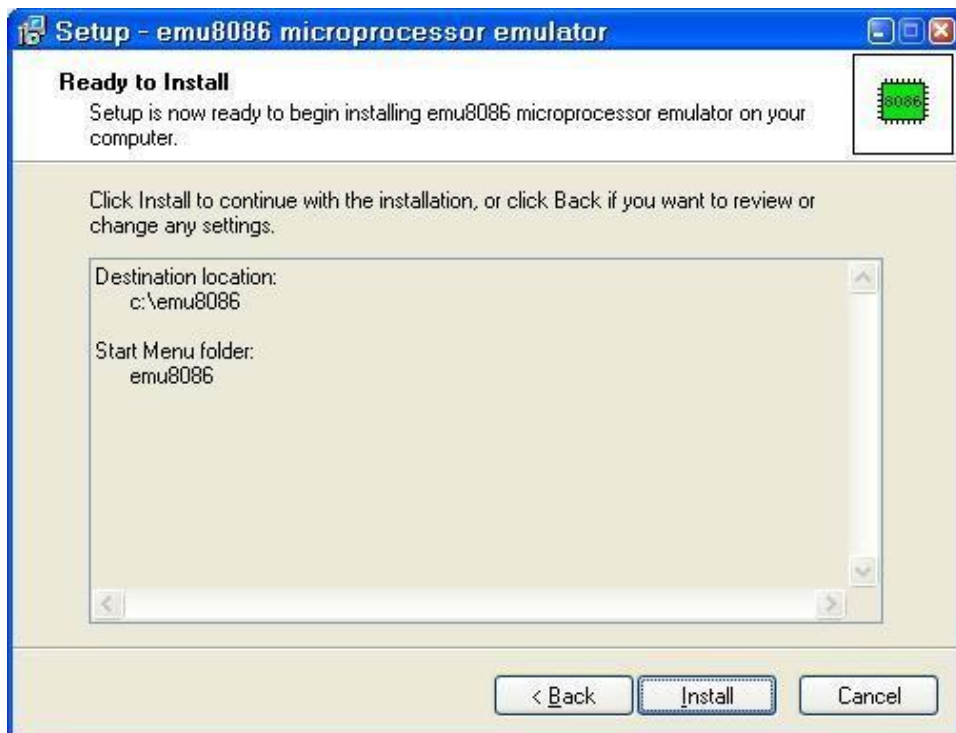
Hình 3. Hướng dẫn chọn thư mục cài đặt

Bước 4: Thông báo tạo shortcut cho chương trình, nhấn Next để tiếp tục.



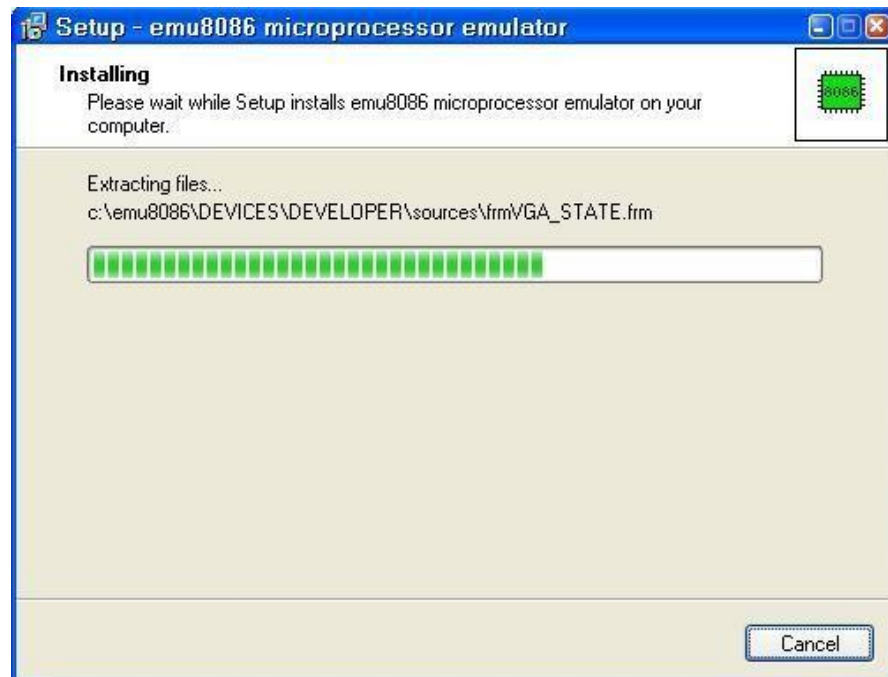
Hình 4. Thông báo tạo shortcut

Bước 5: Sau khi trình cài đặt thu thập đủ thông tin, nó hiện thông báo sẵn sàng cho việc cài đặt và tổng hợp các thông tin đã thu thập được trong các bước trước. Nhấn nút Install để tiến hành cài đặt chương trình.



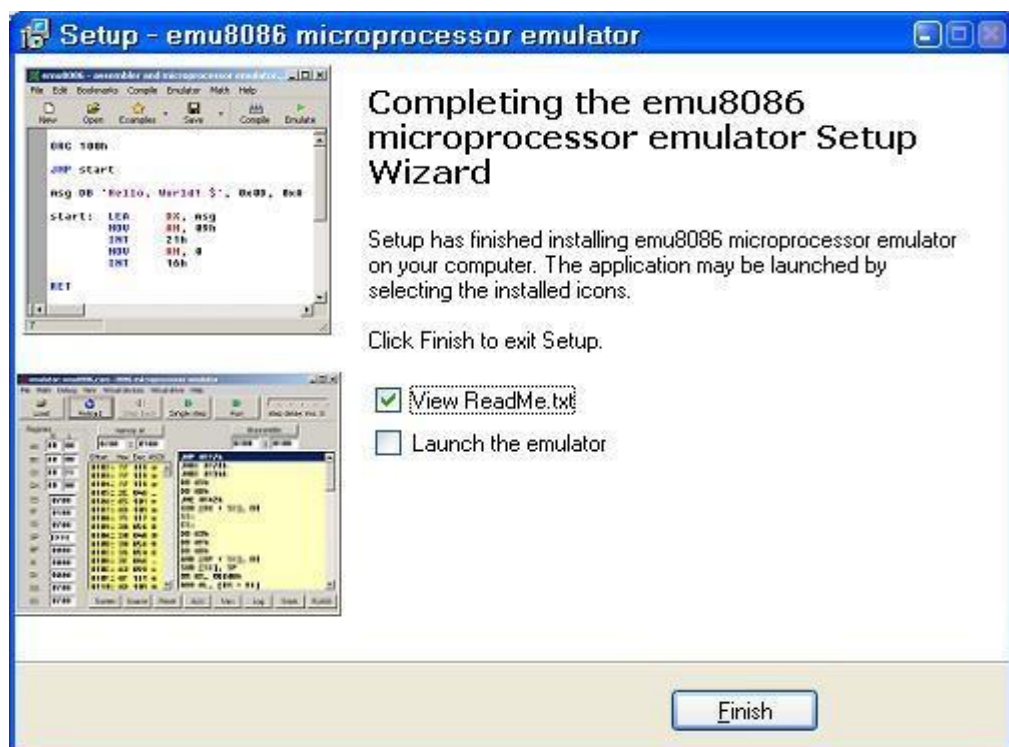
Hình 5. Thông báo sẵn sàng quá trình cài đặt

Bước 6: Chờ cho quá trình cài đặt được thực thi.



Hình 6. Tiến trình cài đặt đang được thực thi

Bước 7: Quá trình cài đặt thành công, nhấn nút Finish để kết thúc.



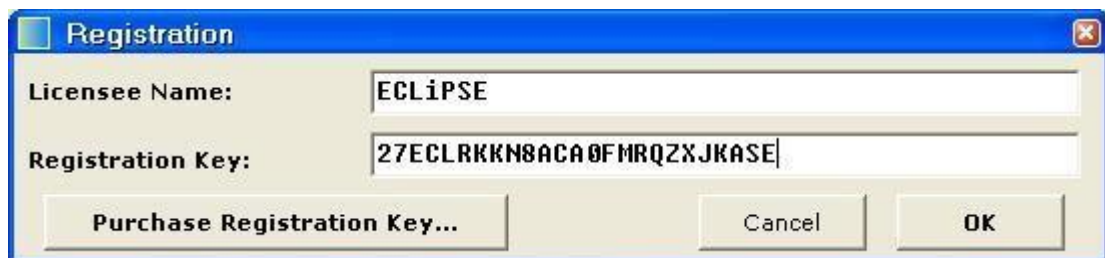
Hình 7. Quá trình cài đặt thành công

Bước 8: Chạy chương trình emu8086 (trên Desktop), giao diện cho lần đầu đăng nhập sẽ hiện ra yêu cầu chúng ta nhập mã đăng ký. Nhấn nút phía dưới (Please Enter the Registration Key) để đăng ký.



Hình 8. Cửa sổ khi chạy chương trình lần đầu

Bước 9: Nhập thông tin đăng ký (có trong file cd key(8086).txt trong thư mục cài đặt) và nhấn nút OK để kết thúc quá trình đăng ký.

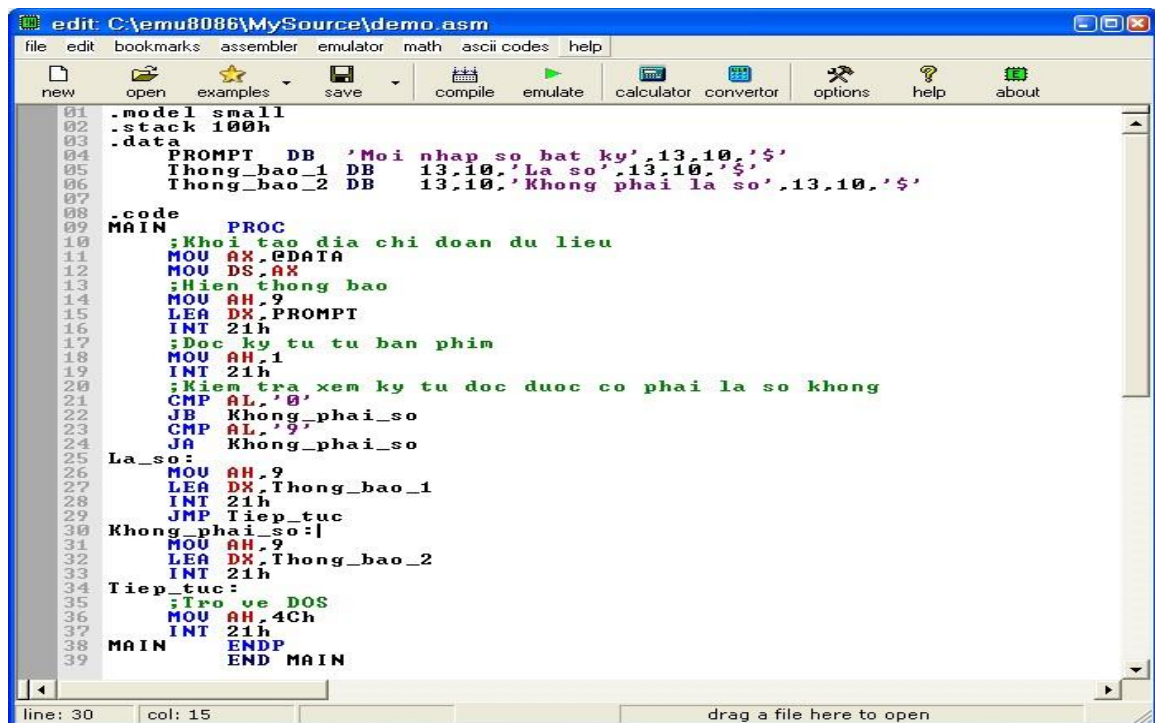


Hình 9. Cửa sổ nhập thông tin đăng ký sử dụng

3.1.2. Hướng dẫn sử dụng phần mềm 8086 Emulator

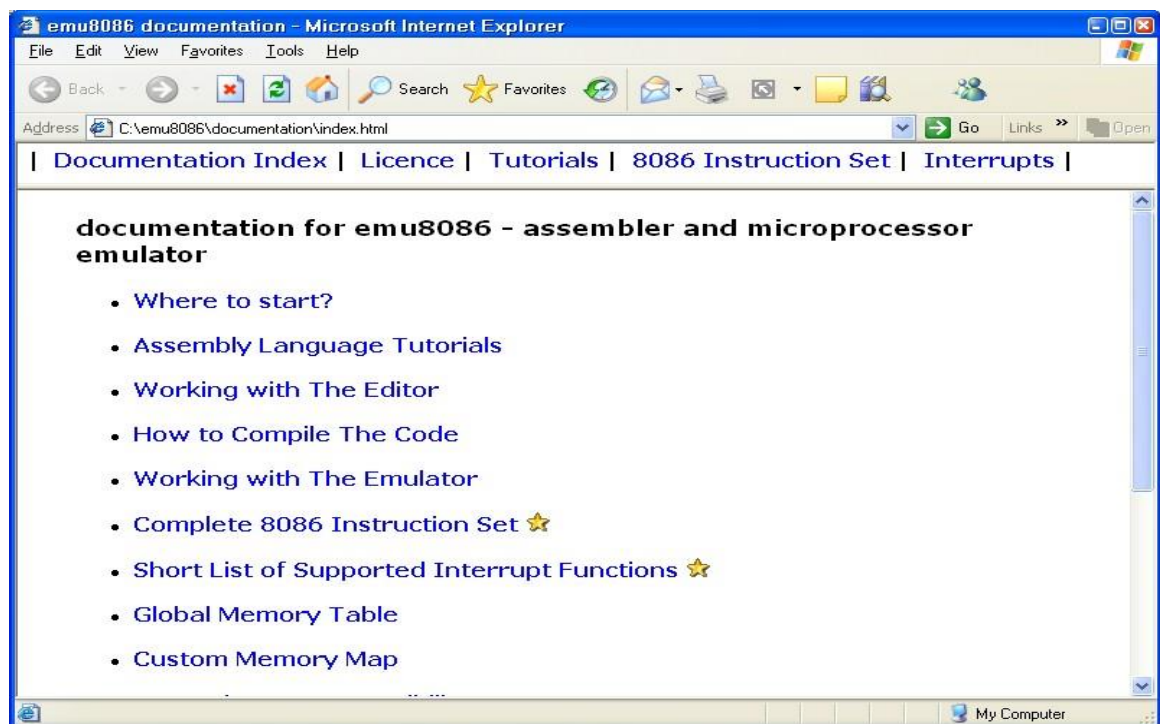
Dưới đây (Hình 10) là giao diện làm việc chính của chương trình 8086 Emulator. Chương trình cho phép chúng ta thực hiện các chức năng chính sau

- Soạn thảo mã hợp ngữ trên màn hình soạn thảo, dịch ra file .exe hoặc file .com và chạy mô phỏng, debug trực tiếp.
- Tra cứu tập lệnh của bộ vi xử lý 8086 (**Help>8086 Instruction Set**)
 - Tra cứu bảng mã ASCII (**Mục ascii codes trên menu**)
 - Thực hiện các phép toán và chuyển đổi giữa các hệ cơ số thông dụng (nhị phân, thập phân, thập lục phân) (**Mục math trên menu**)



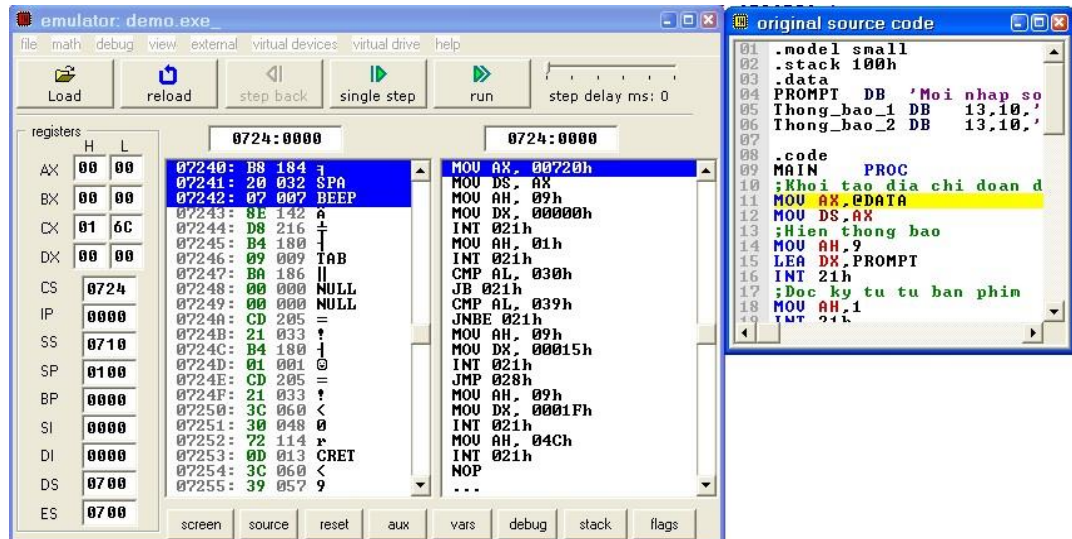
Hình 10. Giao diện chính của chương trình 8086 Emulator

Ngoài ra chương trình có phần trợ giúp rất hữu ích (Mục help trên menu). Người dùng có thể tìm hiểu cách sử dụng chương trình, tra cứu tập lệnh của vi xử lý 8086 và các ngắt cơ bản cũng như các ví dụ sử dụng các lệnh này.



Hình 11. Giao diện trợ giúp của chương trình

3.1.3. Giới thiệu giao diện gỡ lỗi (debug)



Hình 12. Giao diện chạy chương trình và gỡ lỗi

Giao diện chạy và gỡ lỗi sẽ hiện ra khi chúng ta nhấn nút **Emulate** trên thanh công cụ của chương trình. Với giao diện debug này, chúng ta có thể chạy cả chương trình hay chạy ở chế độ từng lệnh để dò lỗi. Chương trình cung cấp giao diện trực quan cho phép người lập trình:

- Theo dõi trực tiếp mã lệnh, địa chỉ và nội dung của các lệnh cũng như dữ liệu trong bộ nhớ
- Quan sát trực tiếp sự thay đổi của các thanh ghi, các biến, dữ liệu trong ngăn xếp và các cờ trong thanh ghi cờ.

3.2. Khảo sát lệnh Intel-8086

3.2.1. Nhập vào Emu8086 đoạn lệnh sau đây và dự đoán trước kết quả:

```
MOV AH, 80      ; AH ← 80 (AX = ?)
MOV AL, 86      ; AL ← 86 (AX = ?)
MOV BX, AX      ; BX ← AX (BH = ?, BL = ?)
MOV DH, BL      ; DH ← BL (DH = ?, DX = ?)
MOV DL, BH      ; DL ← BH (DL = ?, DX = ?)
MOV SI, CS      ; SI ← CS (SI = ?)
```

Thực hiện từng lệnh, sau mỗi lệnh ghi lại kết quả các thanh ghi trong ngoặc để đối chiếu với kết quả dự đoán và giải thích.

2.2. Thực hành tương tự như 2.1 đối với đoạn lệnh sau:

```
MOV AX, 8086    ; AX ← 8086 (AH = ?, AL = ?)
ADD AL, 3        ; AL ← AL + 3 (AL = ?, AX = ?)
DEC AX           ; AX ← AX - 1 (AH = ?, AL = ?, AX = ?)
SUB AH, 10h      ; AH ← AH - 10h (AH = ?, AL = ?, AX = ?)
```

2.2.3. Sinh viên chủ động lập lại ít nhất 1 lần 2.1 và 2.2 với các giá trị toán hạng khác nhau trong mỗi câu lệnh.

3.3. Lập trình hợp ngữ trên 8086

3.3.1. Cấu trúc chung một chương trình hợp ngữ

3.3.1.1. Ví dụ

Để có cái nhìn tổng quát về một chương trình hợp ngữ, ta xét ví dụ sau đây:

```
TITLE    VI DU 1
.MODEL   SMALL
.STACK   100H
.DATA
    A     DB    4
    B     DB    6
    C     DB    ?
.CODE
    MAIN   PROC
        MOV     AX, @DATA
        MOV     DS, AX

        MOV     AL, A
        ADD     AL, B
        MOV     C, AL

        MOV     AX, 4Ch
        INT     21H
    MAIN   ENDP
END MAIN
```

Bước đầu ta chưa cần quan tâm tới ý nghĩa của các câu lệnh mà chỉ cần quan tâm tới bố cục chung của chương trình, các từ khoá cơ bản, cách viết các câu lệnh,...

3.3.1.2. Giải thích

Nhìn chung, cấu trúc của một chương trình hợp ngữ có 3 phần: phần tên, phần khai báo và phần mã lệnh.

a) Phần tên

Tên chương trình được viết sau từ khoá TITLE ở đầu chương trình. Tên có thể chứa dấu cách và các ký tự đặc biệt. Thông thường phần tên sẽ cho ta biết mục đích, nhiệm vụ hoặc nội dung tóm tắt của chương trình.

Ví dụ: TITLE VI DU 1

b) Phần khai báo

Trong hợp ngữ có nhiều nội dung cần phải khai báo như kiểu bộ nhớ, ngăn xếp, biến, hằng,...

➤ **Khai báo kiểu bộ nhớ**

Kiểu bộ nhớ được viết sau từ khoá .MODEL. Kiểu bộ nhớ sẽ quy định kích thước của đoạn mã và đoạn dữ liệu trong chương trình.

Trong chương trình ở phần 3.1.1, kiểu bộ nhớ là SMALL, nghĩa là kiểu bộ nhớ nhỏ, mã lệnh sẽ nằm trong 1 đoạn nhớ, dữ liệu nằm trong 1 đoạn nhớ. Ngoài kiểu SMALL còn có nhiều kiểu bộ nhớ khác:

MEDIUM	Mã lệnh chiếm nhiều hơn 1 đoạn
	Dữ liệu trong 1 đoạn
COMPACT	Mã lệnh trong 1 đoạn
	Dữ liệu chiếm nhiều hơn 1 đoạn
LARGE	Mã lệnh chiếm nhiều hơn 1 đoạn
	Dữ liệu chiếm nhiều hơn 1 đoạn
	Không có mảng nào lớn hơn 64KB
HUGE	Mã lệnh chiếm nhiều hơn 1 đoạn
	Dữ liệu chiếm nhiều hơn 1 đoạn
	Các mảng có thể lớn hơn 64KB

➤ **Khai báo kiểu ngăn xếp**

Kích thước ngăn xếp được viết sau từ khoá .STACK

Ví dụ: .STACK 100h

Khi đó kích thước vùng bộ nhớ làm ngăn xếp là 100h bytes (các vấn đề về ngăn xếp sẽ được trình bày sau).

➤ **Khai báo dữ liệu**

- Khai báo dữ liệu được viết sau từ khoá .DATA. Các biến trong chương trình sẽ được khai báo ở phần này.

Ví dụ: A DB 4

Trong đó: A là tên biến, DB là kiểu dữ liệu (biến kiểu DB sẽ có kích thước 1 byte), 4 là giá trị ban đầu của biến.

Ví dụ: B DW ?

Trong ví dụ này, biến B có kiểu là DW (1 word) và không có giá trị khởi tạo.

- Khai báo hằng

Ví dụ: H EQU 2Bh

Hằng H sẽ nhận giá trị bằng 2Bh

- Quy tắc đặt tên biến:
 - o Tên có chiều dài tối đa 31 ký tự.
 - o Có thể chứa các chữ cái, chữ số và các ký tự đặc biệt (? . @ _ \$ %).
 - o Không được bắt đầu bằng số.
 - o Nếu dùng dấu chấm (.) thì dấu chấm phải đứng đầu tiên.
 - o Tên không được chứa khoảng trắng.
 - o Trong hợp ngữ không phân biệt chữ hoa và chữ thường.

c) Phần mã lệnh

Phần này bao gồm các thủ tục được viết sau từ .CODE. Trong số các thủ tục này phải có 1 thủ tục làm chương trình chính, tên thủ tục đó được viết sau từ **END** ở cuối chương trình. Tên chương trình chính thường đặt là **MAIN**, cũng có thể chọn một tên khác.

➤ **Cấu trúc một thủ tục**

<Tên thủ tục> PROC

Lệnh 1

Lệnh 2

...

Lệnh n

<Tên thủ tục> ENDP

➤ **Cấu trúc chung của phần mã lệnh**

.CODE

<Tên chương trình chính> PROC

Lệnh 1

Lệnh 2

Lệnh 3

...

<Tên chương trình chính> ENDP

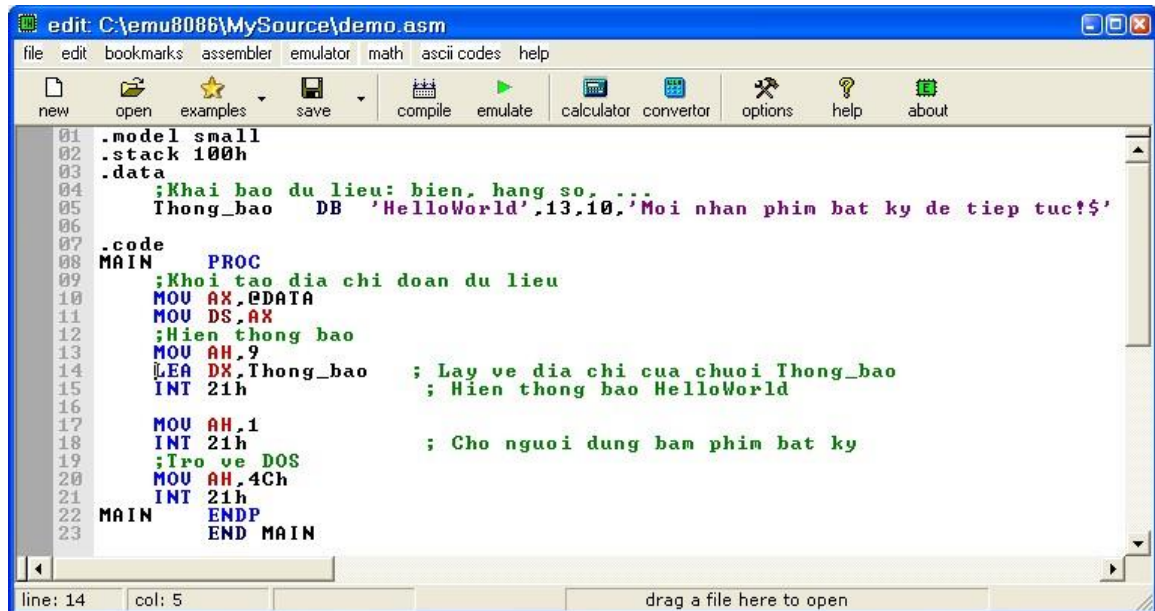
END <Tên chương trình chính>

3.3.2. Chương trình HelloWorld

Cũng như khi chúng ta học các ngôn ngữ lập trình khác, bao giờ chúng ta cũng viết một chương trình đầu tay với mục đích:

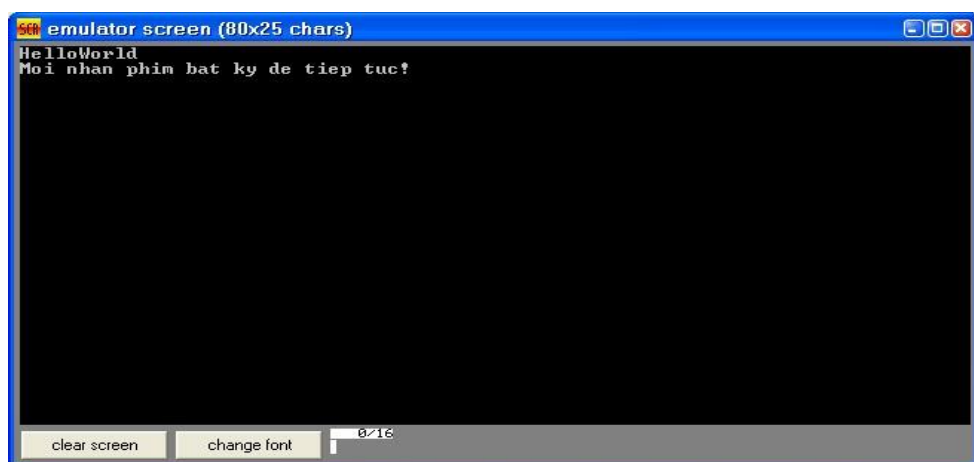
- Kiểm tra cấu trúc chương trình
- Kiểm tra trình biên dịch

Chương trình HelloWorld trong trường hợp này chỉ có chức năng hiển thị một thông báo HelloWorld và chờ người dùng nhấn phím bất kỳ để tiếp tục.



```
01 .model small
02 .stack 100h
03 .data
04 ;Khai bao du lieu: bien, hang so, ...
05 Thong_bao DB 'HelloWorld',13,10,'Moi nhan phim bat ky de tiep tục!$'
06
07 .code
08 MAIN PROC
09 ;Khoi tao dia chi doan du lieu
10 MOV AX,0DATA
11 MOV DS,AX
12 ;Hien thong bao
13 MOV AH,9
14 LEA DX,Thong_bao ; Lay ve dia chi cua chuoi Thong_bao
15 INT 21h ; Hien thong bao HelloWorld
16
17 MOV AH,1
18 INT 21h ; Cho nguoi dung bam phim bat ky
19 ;Tro ve DOS
20 MOV AH,4Ch
21 INT 21h
22 MAIN ENDP
23 END MAIN
```

Hình 13. Chương trình HelloWorld



Hình 14. Kết quả khi chạy chương trình HelloWorld

Lưu ý: - Chương trình hoàn toàn không có lỗi.

- Trong đó có những lệnh mà sinh viên chưa học đến, điều này không cần quan tâm, điều cần quan tâm trong bài thực hành này là Cấu trúc chương trình hợp ngữ.

3.3.3. Viết các chương trình đơn giản

3.3.3.1. Viết chương trình hợp ngữ dạng EXE để tính kết quả biểu thức sau, lưu trữ kết quả trong AX, lưu file với tên SUM1.ASM

$$10 + 8086 - 100h + 350 + 0Fh$$

Lưu ý: Chỉ khi báo đoạn lệnh để viết chương trình

- Dịch sửa lỗi (nếu có lỗi) và chạy chương trình.
- Dùng Emu8086 để chạy chương trình trên và kiểm tra kết quả lưu trong AX.

3.3.2. Viết chương trình hợp ngữ dạng EXE để tính kết quả biểu thức sau, lưu trữ kết quả trong AX, lưu file với tên SUM2.ASM

$$KQUA = A + B - C + D + E$$

Trong đó: KQUA, A, B, C, D, E là các biến 2 byte khai báo trong đoạn dữ liệu.

Lưu ý: Chương trình gồm 2 đoạn: Đoạn lệnh và đoạn dữ liệu để chứa các biến.

- Gán giá trị biến A = 1000, B = 10, C = 1Fh, D = 30h, E = 300Ah. Dịch và chạy chương trình.
- Dùng Emu8086 để kiểm tra kết quả của câu a.
- Áp dụng SUM2.ASM để tính biểu thức đã cho ở câu 3.3.1. Dùng Emu8086 để kiểm tra kết quả.

4. BÀI TẬP ĐỀ NGHỊ

1.1. Dùng Emu8086 để khảo sát các lệnh khác trong tập lệnh của Intel – 8086

1.2. Tự tìm hiểu thêm những chức năng khác của Emu8086.

1.3. Viết từng chương trình tính các biểu thức sau:

- | | |
|-----------------|------------------------|
| a. $15h * 250$ | d. $1000 \div 100$ |
| b. $16 * 0AF1h$ | e. $1000 \div 100h$ |
| c. $300 * 400$ | f. $3AB45Eh \div 0A1h$ |

Sử dụng Emu8086 để kiểm chứng kết quả của các chương trình đã viết.

BÀI THỰC HÀNH SỐ 2

NHẬP XUẤT KÝ TỰ

1. MỤC TIÊU

Sau khi kết thúc bài thực hành này, sinh viên có thể:

- Sử dụng được các ngắt mềm để viết được chương trình: in ký tự - chuỗi ký tự lên màn hình và nhập ký tự - chuỗi ký tự từ bàn phím.
- Hiểu được cách quản lý ký tự và ký số trong hợp ngữ.

2. KIẾN THỨC CHUẨN BỊ

- Kết quả bài thực hành 1.
- Các hàm 01h, 02h, 06h, 07h, 08h, 09h, 0Ah của ngắt 21h.
- Bảng mã ASCII

3. NỘI DUNG THỰC HÀNH

3.1. Một số lệnh thường dùng

3.1.1. Cấu trúc câu lệnh hợp ngữ

Một câu lệnh hợp ngữ có thể chia làm 2 phần: Tên lệnh và các toán hạng. Tên lệnh viết trước, toán hạng viết sau. Nếu có nhiều toán hạng thì chúng được phân cách với nhau bằng dấu phẩy (.).

<Tên lệnh> <Toán hạng 1> [, <Toán hạng 2>...] [;lời chú thích]

Ngoài ra có thể viết thêm lời chú thích để làm rõ ý nghĩa của câu lệnh. Lời chú thích được bắt đầu bằng dấu chấm phẩy (;).

Ví dụ 1: MOV DS, AX ; Chuyển nội dung của thanh ghi AX vào thanh ghi DS

Trong đó: - MOV là tên lệnh.

- DS và AX là các toán hạng.

Lưu ý: Các lệnh được trình bày trong tài liệu này hầu hết thuộc tập lệnh của bộ vi xử lý 8086, trừ một số trường hợp cụ thể sẽ có chú thích riêng.

3.1.2. Một số lệnh thường dùng

3.1.2.1. Lệnh MOV (move)

Lệnh này được sử dụng để chuyển dữ liệu giữa các thanh ghi hay ô nhớ.

Cú pháp lệnh:

MOV <Toán hạng đích>, <Toán hạng nguồn>

Trong đó: - <Toán hạng đích>: là một thanh ghi hay một ô nhớ.

- <Toán hạng nguồn>: là một thanh ghi, một ô nhớ hay một hằng số.

Dữ liệu sẽ được chuyển từ Toán hạng nguồn vào Toán hạng đích (Nội dung của toán hạng nguồn không thay đổi sau khi chuyển).

Ví dụ 2: `MOV AX, 4C00h`

→ Lệnh chuyển giá trị 4C00h vào thanh ghi AX.

Ví dụ 3: `MOV AL, A`

→ Lệnh chuyển giá trị của biến A vào thanh ghi AL.

Chú ý:

- Không được chuyển trực tiếp nội dung của 2 biến cho nhau.
- Không được chuyển trực tiếp một hằng số vào thanh ghi đoạn.
- Không được chuyển trực tiếp nội dung của 2 thanh ghi đoạn cho nhau.

Ví dụ 4: `MOV A, B` ; lệnh này không thực hiện được vì A, B đều là biến

Ví dụ 5: `MOV DS, A` ; lệnh này không thực hiện được

Ví dụ 6: `MOV CS, DS`

→ Lệnh bị sai do CS và DS đều là thanh ghi đoạn. Muốn thực hiện được điều này thì phải sử dụng một biến hay một thanh ghi khác thanh ghi đoạn làm trung gian.

Ví dụ 7: `MOV AX, DS` ; dùng AX làm trung gian
 `MOV CS, AX`

3.1.2.2. Lệnh XCHG (Exchange)

Lệnh này dùng để hoán đổi dữ liệu giữa 2 toán hạng

Cú pháp lệnh:

XCHG <Toán hạng 1>, <Toán hạng 2>

Trong đó: Các toán hạng có thể là thanh ghi đa dụng, hoặc một thanh ghi đa dụng và một ô nhớ.

Ví dụ 8: `XCHG AX, BX` ; hoán đổi nội dung của AX và BX

3.1.2.3. Lệnh ADD

Lệnh ADD sẽ thực hiện phép toán cộng Toán hạng đích với Toán hạng nguồn, kết quả chứa trong Toán hạng đích.

Cú pháp lệnh: ***ADD <Toán hạng đích>, <Toán hạng nguồn>***

Trong đó: - <Toán hạng đích>: là một thanh ghi hay một ô nhớ.

- <Toán hạng nguồn>: là một thanh ghi, một ô nhớ hay một hằng số.

- <Toán hạng đích>, <Toán hạng nguồn>: không đồng thời là 2 ô nhớ.

Ví dụ 9: `ADD AX, 10` ; tăng nội dung của thanh ghi AX lên 10

`ADD BX, AX` ; cộng nội dung của AX và BX, kết quả lưu ở BX

3.1.2.4. Lệnh SUB

Lệnh SUB sẽ lấy Toán hạng đích trừ đi Toán hạng nguồn, kết quả chứa trong Toán hạng đích.

Cú pháp lệnh: **SUB <Toán hạng đích>, <Toán hạng nguồn>**

Trong đó: - <Toán hạng đích>: là một thanh ghi hay một ô nhớ.

- <Toán hạng nguồn>: là một thanh ghi, một ô nhớ hay một hằng số.

- <Toán hạng đích>, <Toán hạng nguồn>: không đồng thời là 2 ô nhớ.

Ví dụ 10: SUB AX, 10 ; giảm nội dung của thanh ghi AX xuống 10
SUB AX, B ; trừ nội dung của thanh ghi AX cho biến B

3.1.2.5. Lệnh INC (Increment) và DEC (Decrement)

Lệnh INC sẽ tăng toán hạng đích lên 1 (cộng toán hạng đích với 1)

Lệnh DEC sẽ giảm toán hạng đích xuống 1 (trừ toán hạng đích đi 1)

Cú pháp lệnh:

INC <Toán hạng đích>

DEC <Toán hạng đích>

Trong đó: <Toán hạng đích>: là một thanh ghi hoặc một ô nhớ.

Ví dụ 11: INC AH ; Cộng nội dung của thanh ghi AH với 1
DEC B ; Trừ giá trị của biến B cho 1.

3.1.2.6. Lệnh NEG (Negative)

Lệnh NEG có tác dụng đổi dấu của toán hạng đích.

Cú pháp lệnh: **NEG <Toán hạng đích>**

Trong đó: <Toán hạng đích>: là một thanh ghi hoặc một ô nhớ.

Ví dụ 12: NEG AL

3.2. Nhập xuất dữ liệu

3.2.1. Chương trình ngắt

Chương trình ngắt là những chương trình con đã được viết sẵn nhằm thực hiện những chức năng cơ bản khi thao tác với máy tính. Các chương trình con này được phân phối kèm theo các phần mềm điều khiển hệ thống như BIOS, Hệ điều hành. Mỗi chương trình có một số hiệu riêng (0, 1, 2, ...). Khi lập trình ta có thể sử dụng các chương trình con có sẵn này bằng cách dùng lệnh INT (interrupt).

Cú pháp lệnh: **INT <số hiệu ngắt>**

Ví dụ 13: INT 21h

Lệnh trên sẽ gọi thực hiện chương trình ngắt số 21h (đây là số hiệu ngắt hay sử dụng nhất của DOS).

Ví dụ 14: INT 13h

Lệnh trên sẽ gọi thực hiện chương trình ngắt số 13h (đây là số hiệu ngắt của BIOS, sùn thao tác với đĩa từ).

Chi tiết về các ngắt và chương trình xử lý ngắt sẽ trình bày sau.

3.2.2. Chức năng nhập - xuất của ngắt 21h

Ngắt 21h của DOS cung cấp rất nhiều chức năng khác nhau, mỗi chức năng có một số hiệu riêng (0, 1, 2,...). Trong phần này ta chỉ quan tâm đến chức năng nhập - xuất dữ liệu.

Các sử dụng ngắt 21h: Trước khi gọi ngắt cần xác định số hiệu chức năng, số hiệu đó được đặt vào thanh ghi AH. Ngoài ra cũng cần quan tâm tới các tham số khác (chương trình ngắt sẽ sử dụng thanh ghi nào? Giá trị của chúng bằng bao nhiêu?...).

3.2.2.1. Chức năng hiện 1 ký tự ra màn hình

Đây là chức năng số 2 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 2

DL = Mã ASCII của ký tự cần hiển thị

Ra: AL chứa mã ASCII của ký tự cần hiển thị

Ví dụ 15: Chương trình sử dụng chức năng số 2 của ngắt 21h để in ký tự 'B' ra màn hình. Hãy soạn thảo và lưu lại thành tập tin HienKyTu.asm.

```
TITLE HIEN KY TU

.MODEL SMALL

.STACK 100H

.CODE

    MAIN    PROC

        MOV AH, 2      ; Chức năng số 2
        MOV DL, 'B'    ; Ký tự cần hiển thị
        INT 21h        ; Gọi ngắt 21h

        MOV AH, 4Ch    ; Kết thúc
        INT 21h

    MAIN    ENDP

END MAIN
```

- Dịch sửa lỗi (nếu có) và chạy chương trình để xem kết quả in ra màn hình.
- Các dòng lệnh nào thực hiện chức năng in ký tự '**B**' ra màn hình? Các dòng lệnh khác dùng làm gì?
- Sửa lại chương trình trên để in ra màn hình ký tự '**D**'. Chạy chương trình và kiểm chứng kết quả.
- Viết chương trình để in ra màn hình số **9**.

Ta có thể sử dụng mã ASCCI của ký tự hoặc viết trực tiếp ký tự giữa 2 dấu nhảy đơn như trong chương trình trên (nói chung các ký tự và chuỗi ký tự trong hợp ngữ pahir có dấu nhảy đơn ở 2 đầu).

Ngoài chức năng số 2, trong chương trình còn sử dụng chức năng 4Ch của ngắt 21h. Chức năng này có tác dụng kết thúc chương trình và trả lại quyền điều khiển cho hệ điều hành DOS.

3.2.2.2. Chức năng hiện 1 chuỗi ký tự ra màn hình

Đây là chức năng số 9 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 9

DX = Địa chỉ offset của vùng nhớ chứa chuỗi ký tự

Ví dụ 16: Chương trình sử dụng chức năng số 9 của ngắt 21h để in ra màn hình dòng chữ: KHOA CNTT. Hãy soạn thảo và lưu lại thành tập tin HienChuoiKyTu.asm

```
TITLE HIEN CHUOI KY TU

.MODEL SMALL

.STACK 100H

.DATA

    ChuoiKT DB 'KHOA CNTT$'

.CODE

MAIN    PROC

    MOV AX, @DATA
    MOV DS, AX

    MOV AH, 9          ; Chức năng số 9
    LEA DX, ChuoiKT    ; Lấy đchỉ chuỗi ký tự đặt vào DX
    INT 21h            ; Gọi ngắt 21h

    MOV AH, 4Ch ; Kết thúc
    INT 21h

MAIN ENDP

END MAIN
```

- Dịch sửa lỗi (nếu có) và chạy chương trình để xem kết quả trên màn hình.
- Viết lại chương trình để in ra chuỗi “Tuong DH SPKT Vinh Long”.
- Sửa khai báo biến ChuoiKT có dạng như sau:

ChuoiKT DB ‘Truong DH SPKT’, 10, 13, ‘Vinh Long\$’

Dịch và chạy chương trình để xem kết quả. Trong khai báo biến ChuoiKT, 2 giá trị **10, 13** có ý nghĩa gì trong việc in chuỗi ra màn hình?

- Sửa lại chương trình để in ra màn hình số 2016

Trong chương trình trên có một số điểm cần lưu ý:

- Chuỗi ký tự cần hiển thị phải được kết thúc bằng dấu \$
- Nếu trong chương trình có sử dụng khai báo .DATA thì ở đầu chương trình chính phải có các lệnh :

MOV AX, @DATA

MOV DS, AX

Mục đích để đặt địa chỉ segment của đoạn dữ liệu vào thanh ghi DS.

- Chương trình trên có sử dụng lệnh LEA (Load Effective Address). Lệnh này sẽ lấy địa chỉ offset của toán hạng nguồn đặt vào toán hạng đích.

Cú pháp lệnh như sau:

LEA <Toán hạng đích>, <Toán hạng nguồn>

Trong đó: - <Toán hạng đích>: là một thanh ghi đa dụng.

- <Toán hạng nguồn>: là một ô nhớ.

3.2.2.3. Chức năng nhập 1 ký tự từ bàn phím

Đây là chức năng số 1 của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 1

Ra: AL chứa mã ASCII của ký tự

Các lệnh cụ thể như sau:

MOV AH, 1

INT 21h

Khi gặp các lệnh trên, chương trình sẽ dừng lại chờ ta gõ một ký tự từ bàn phím, mã ASCII của ký tự đó sẽ được cất trong thanh ghi AL.

Ví dụ 17: Viết chương trình nhập 1 ký tự từ bàn phím, đổi ký tự đó thành ký tự in hoa rồi hiện ra màn hình. Hãy soạn thảo và lưu lại thành tập tin DoiKT.asm

```
TITLE DOI KY TU
.MODEL SMALL
.STACK 100H
.CODE
```

```

MAIN      PROC
          MOV AH, 1          ; Chức năng số 1: Nhập 1 ký tự
          INT 21h

          SUB AL, 20h        ; đổi sang ký tự in hoa

          MOV AH, 2          ; Chức năng số 2: Hiện 1 ký tự
          MOV DL, AL
          INT 21h

          MOV AH, 4Ch        ; Kết thúc
          INT 21h

MAIN      ENDP
END MAIN

```

- Dịch sửa lỗi (nếu có) và chạy chương trình để xem kết quả trên màn hình.
- Ký tự nhập được lưu trữ ở đâu và được CPU quản lý ở dạng thức gì? (Dùng EMU8086 sswwe khảo sát)
- Sửa chương trình để đọc ký tự bằng chức năng số 8 của ngắt 21h.
- Chạy chương trình và so sánh hoạt động của hai chức năng 1 và 8.

Giải thích: Mã ASCII của ký tự thường lớn hơn ký tự in hoa tương ứng là 20h. Muốn chuyển từ ký tự in thường thành ký tự in hoa thì chỉ việc lấy mã ASCII của nó trừ đi 20h.

3.2.2.4. Chức năng nhập 1 chuỗi ký tự từ bàn phím

Đây là chức năng số 0Ah của ngắt 21h. Các tham số cần thiết để gọi ngắt như sau:

Vào: AH = 0Ah

DS:DX = Địa chỉ của vùng đệm ký tự nhận dữ liệu

Trước khi gọi hàm này vùng đệm (Buffer) có dạng sau:

Offset	Ý nghĩa
0	Số ký tự tối đa của chuỗi được nhập vào
1	Số ký tự thật sự nhập vào (không kể Enter)
2, 3, 4, ...	Các ký tự đã nhập (kể cả ký tự Enter)

Ví dụ:

```

Buffer  DB 81 ; Chiều dài tối đa của chuỗi ký tự
          DB 0  ; Số ký tự thực đã nhập của chuỗi
          DB 81 DUP(0) ; Nơi chứa các ký tự nhập vào

```

Hàm này khi nhập vào kết thúc bằng dấu ENTER.

Ví dụ 18: Nhập 1 chuỗi ký tự từ bàn phím.

```
TITLE NHAP CHUOI KY TU
.MODEL SMALL
.STACK 100H
.DATA
    Buffer      DB 30
                DB 0
                DB 30 DUP(0)
    Tbao DB 'Hay nhap vao 1 chuoi: $'
.CODE
    MAIN      PROC
                MOV AX, @DATA
                MOV DS, AX
                MOV AH, 9      ; Chức năng số 9: hiện 1 chuỗi kt
                LEA DX, Tbao
                INT 21h

                MOV AH, 0Ah    ; Chức năng số 0Ah: nhập 1 chuỗi kt
                LEA DX, Buffer
                INT 21h

                MOV AH, 4Ch    ; Kết thúc
                INT 21h
    MAIN ENDP
END MAIN
```

- Dịch sửa lỗi (nếu có) và chạy chương trình trong từng trường hợp sau:
 1. Nhập từ bàn phím chuỗi có ít hơn 30 ký tự.
 2. Nhập từ bàn phím chuỗi có nhiều hơn 30 ký tự.
- Giá trị của [Buffer + 1] trong mỗi trường hợp bằng bao nhiêu?
- Tại sao không thể nhập nhiều hơn 30 ký tự? Chuỗi ký tự nhập vào được lưu trữ ở biến nào?
- Sửa chương trình để có thể nhập nhiều hơn 30 ký tự (60 ký tự chẳng hạn).
- Khả năng tối đa của chức năng 0Ah của ngắt 21h là nhận chuỗi bao nhiêu ký tự?

4. BÀI TẬP ĐỀ NGHỊ

1. Viết chương trình sử dụng chức năng số 1 của ngắt 21h để nhận 1 ký tự từ bàn phím. Dùng 1 biến để lưu trữ ký tự đã nhận được (do sinh viên tự đặt tên biến), sau đó sử dụng chức năng số 2 của ngắt 21h để in ra màn hình ký tự được đang lưu trong biến ấy. Chương trình phải có đủ các câu thông báo nhập và xuất.

Ví dụ: *Hay go 1 phim: B*

Ky tu nhan duoc la: B

2. Sửa lại chương trình ở câu 1 sao cho không cần sử dụng biến để lưu trữ ký tự mà kết quả chạy chương trình vẫn không thay đổi.
3. Viết chương trình nhận vào 1 ký tự từ bàn phím, sau đó in ra màn hình ký tự kế trước và kế sau của ký tự vừa nhập.

Ví dụ: *Hay go 1 phim: B*
 Ky tu ke truooc la: A
 Ky tu ke sau la: C

4. Viết chương trình cho phép nhập từ bàn phím tên của 1 người, sau đó in ra màn hình chuỗi có dạng như sau:

Xin chao <tên_đã_nhập>

Ví dụ: Khi chạy chương trình, nhập vào là: Nguyen Van Anh
Chuỗi in ra màn hình sẽ là: Xin chao Nguyen Van Anh

BÀI THỰC HÀNH SỐ 3

CẤU TRÚC RỄ NHÁNH – VÒNG LẶP

1. MỤC TIÊU

Sau khi kết thúc bài thực hành này, sinh viên có thể:

- Hiểu cách so sánh hai số trong hợp ngữ.
- Hiểu cách thay đổi thứ tự thực hiện các lệnh.
- Biết cách sử dụng các lệnh so sánh, nhảy và lặp.

2. KIẾN THỨC CHUẨN BỊ

- Kết quả bài thực hành 2.
- Thanh ghi cờ và các cờ trạng thái.
- Các lệnh so sánh, nhảy và lặp.

3. TÓM TẮT LÝ THUYẾT

3.1. Thanh ghi cờ và các cờ trạng thái

Thanh ghi cờ dài 16 bit, mỗi bit được gọi là một cờ và có công dụng riêng. Dưới đây là các vị trí của các cờ:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				OF	DF	IF	TF	SF	ZF		AF		PF		CF

Ta thấy bộ vi xử lý 8086 mới sử dụng 9 bit của thanh ghi cờ, sau đây là tên và chức năng của các cờ:

Bit	Tên cờ	Ký hiệu
0	Cờ nhớ (Cary Flag)	CF
2	Cờ chẵn lẻ (Parity Flag)	PF
4	Cờ nhớ phụ (Auxiliary Flag)	AF
6	Cờ Zero (Zero Flag)	ZF
7	Cờ dấu (Sign Flag)	SF
11	Cờ tràn (OverFlow Flag)	OF
8	Cờ bẫy (Trap Flag)	TF
9	Cờ ngắt (Interrupt Flag)	IF
10	Cờ định hướng (Direction Flag)	DF

Các cờ chỉ làm hai nhóm khác nhau:

- Nhóm cờ trạng thái: gồm 6 cờ CF, PF, AF, ZF, SF, OF
- Nhóm cờ điều khiển: gồm 3 cờ TF, IF, DF

Mỗi khi một lệnh trong chương trình được thực hiện thì trạng thái của bộ vi xử lý lại thay đổi, sự thay đổi này được phản ánh trong các cờ trạng thái. Để hiểu rõ hơn vấn đề này chúng ta xem xét một vài cờ trạng thái hay dùng.

3.1.1. Cờ nhớ CF

Ví dụ 1: Xét các lệnh sau đây:

MOV AX, 0FFFFh

ADD AX, 1

Trước khi thực hiện lệnh ADD thì AX = FFFFh = 1111 1111 1111 1111b = 65535

Sau khi thực hiện phép cộng với 1 thì AX bằng bao nhiêu?

1111 1111 1111 1111b
+ 1
1 0000 0000 0000b

Thanh ghi AX dài 16 bit nên sau lệnh AD thì AX = 0. Phép cộng đã không còn chính xác do kết quả vượt quá phạm vi chứa của AX (gọi là hiện tượng tràn khi cộng số không dấu). Khi đó cờ CF được thiết lập bằng 1.

Như vậy, cờ CF sẽ được thiết lập khi thực hiện phép cộng có nhớ ở bit Msb hoặc khi thực hiện phép trừ có vay ở bit Msb.

3.1.2. Cờ Zero CF

Ví dụ 2: Xét các lệnh sau đây:

MOV CX, 2Ah

SUB CX, 2Ah

Sau khi thực hiện lệnh SYB thì CX = 0, cờ ZF được thiết lập bằng 1.

Như vậy, cờ ZF sẽ được thiết lập khi kết quả của lệnh vừa thực hiện bằng 0.

3.1.3. Cờ tràn OF

Ví dụ 3: Xét các lệnh sau đây:

MOV AX, 7FFFh

ADD AX, 7FFFh

Trước khi thực hiện lệnh ADD thì AX = 7FFFh = 0111 1111 1111 1111b = 32767

Sau khi thực hiện phép cộng với 1 thì AX bằng bao nhiêu?

0111 1111 1111 1111b
+ 0111 1111 1111 1111b
1111 1111 1111 1110b

Sau lệnh ADD thì $AX = FFFEh$. Nếu coi đây là số không dấu thì $AX = 65634$, không có hiện tượng tràn, cờ $CF = 0$. Nhưng nếu coi đây là số có dấu thì $AX = -2$, phép cộng đã không còn chính xác do kết quả vượt quá phạm vi chứa của AX (gọi là hiện tượng tràn khi cộng số có dấu). Khi đó cờ OF được thiết lập bằng 1.

Như vậy, cờ CF sẽ được thiết lập khi xuất hiện hiện tượng tràn trong phép tính với số có dấu.

3.2. Các lệnh có điều kiện

3.2.1. Ví dụ

Xét chương trình hợp ngữ sau

```
TITLE Lenh nhay
.MODEL    SMALL
.STACK    100H
.CODE
    MAIN    PROC
        NHAPLAI:
            MOV    AH, 1            ;Chức năng số 1: Nhập 1 ký tự
            INT    21h
            CMP    AL, ' '          ;Kiểm tra ký tự nhập
            JZ     NHAPLAI
            ...
            MOV    AH, 4Ch          ;Kết thúc
            INT    21h
        MAIN    ENDP
    END MAIN
```

Giải thích: Chương trình trên sẽ nhập 1 ký tự từ bàn phím, kiểm tra xem đó có phải là ký tự khoảng trắng ' ' hay không, nếu đúng thì tiến hành nhập lại. Quá trình đó được thực hiện nhờ lệnh so sánh CMP và lệnh nhảy JZ .

3.2.1.1. Lệnh so sánh CMP (Compare)

Cú pháp:

$CMP <Toán\ hạng\ đích>, <Toán\ hạng\ nguồn>$

Lệnh có tác dụng tương tự lệnh SUB , nó thực hiện phép trừ giữa Toán hạng đích và Toán hạng nguồn. Sự khác biệt là kết quả của phép tính không được lưu vào Toán hạng đích như trong lệnh SUB mà tính chất của kết quả được thể hiện thông qua cờ.

Ví dụ 4: $CMP \quad AL, ' '$

→ Lệnh này sẽ lấy nội dung của AL trừ đi cho $20h$ (mã ASCII của ký tự khoảng trắng). Nếu kết quả bằng 0, tức là $AL = 20h$ ($AL = ' '$) thì cờ ZF sẽ được thiết lập bằng 1. Trạng thái của các cơ sẽ được dùng làm điều kiện cho các lệnh nhảy.

Ví dụ 5: So sánh 2 số nguyên dương

MOV AH, 1 ; AH ← 1
 MOV AL, 2 ; AH ← 2
 CMP AH, AL ; CF ← 1, ZF ← 0 vì AH < AL

Sau khi thực hiện các lệnh trên, cờ Carr bật (CF = 1), báo hiệu rằng AH < AL

3.2.1.2. Lệnh nhảy JZ

Lệnh JZ là lệnh nhảy khi cờ CF = 1

Cú pháp lệnh: **JZ <nhãn>**

Trong chương trình ở ví dụ trên, lệnh JZ sẽ kiểm tra cờ ZF, nếu cờ ZF = 1 thì sẽ nhảy đến nhãn NHAPLAI, nghĩa là thực hiện lại các lệnh nhập dữ liệu.

3.2.1.3. Một số lệnh nhảy có điều kiện thường dùng

Việc kiểm tra trạng thái của các cờ khi sử dụng lệnh nhảy gây rất nhiều khó khăn cho người lập trình (do có quá nhiều lệnh nhảy, khó nhớ, không hợp với tư duy thông thường,...). Để khắc phục điều này, người ta thường sử dụng lệnh nhảy kèm với lệnh CMP theo quy tắc sau:

CMP <Toán hạng đích>, <Toán hạng nguồn>

Điều kiện nhảy	Lệnh nhảy không dấu	Lệnh nhảy có dấu
Đích > Nguồn	JA/JNBE	JG/JNLE
Đích < Nguồn	JB/JNAE	JL/JNGE
Đích = Nguồn	JE/JZ	JE/JZ
Đích ≥ Nguồn	JAE/JNB	JGE/JNL
Đích ≤ Nguồn	JBE/JNA	JLE/JNG
Đích ≠ Nguồn	JNE/JNZ	JNE/JNZ

Một số từ viết tắt:

A: Above (lớn hơn) = G: Greater than

B: Below (nhỏ hơn) = L: Less than

E: Equal (bằng)

3.3. Lệnh nhảy không điều kiện

Các lệnh nhảy có điều kiện mà ta đã nghiên cứu có một nhược điểm là không thể nhảy quá xa. Các lệnh đó chỉ có thể nhảy tới một nhãn đứng trước nó không quá 126 byte hoặc không đứng sau quá 127 byte.

Để khắc phục điều này có thể sử dụng lệnh nhảy không điều kiện JMP.

Cú pháp lệnh: **JMP <nhãn>**

Vị trí của nhãn phải nằm trên một đoạn nhớ với lệnh nhảy JMP.

Ví dụ 6: Xét đoạn lệnh sau:

```
MOV AH, 1      ;Nhập 1 ký tự
INT 21h

CMP AL, 'Z'    ;So sánh ký tự nhập với 'Z'
JA  KetThuc    ;Nếu AL > 'Z' thì nhảy tới nhãn KetThuc

...           ;Các lệnh khác

KetThuc:
    MOV  AH, 4Ch
    INT 21h
```

Đoạn lệnh trên chỉ được thực hiện khi khoảng giữa lệnh JA và vị trí đặt nhãn KetThuc không quá 127 byte. Tuy nhiên nếu khoảng cách đó vượt quá giới hạn cho phép thì ta có thể khắc phục bằng phương pháp “nhảy hai bước” nhờ lệnh JMP (đầu tiên nhảy tới một “Nhãn trung gian”, sau đó mới nhảy đến nhãn KetThuc)

```
MOV AH, 1      ;Nhập 1 ký tự
INT 21h

CMP AL, 'Z'    ;So sánh ký tự nhập với 'Z'
JA NhanTG      ;Nếu AL > 'Z' thì nhảy tới nhãn KetThuc
JMP TiepTuc

NhanTG:
    JMP KetThuc

TiepTuc:
...           ;Các lệnh khác

KetThuc:
    MOV  AH, 4Ch
    INT 21h
```

❖ Các trường hợp khác

- JMP SHORT <Nhãn> ;(short jump)

Kiểu này chỉ nhảy trong phạm vi từ -128 đến +127 byte so với vị trí hiện tại.

- JMP FAR <Nhãn> ; (far jump)

Kiểu này nhảy đến bất kỳ chỗ nào.

3.4. Lệnh lặp

Bằng cách dùng các lệnh nhảy có thể tạo ra vòng lặp. Tuy nhiên, để viết chương trình tiện lợi và ngắn gọn, có thể dùng thêm các lệnh lặp như LOOP, LOOPZ,...

Cú pháp: **LOOP <Label>**

→ Lệnh lặp LOOP tự động giảm CX một đơn vị, sau đó kiểm tra CX có bằng 0, nếu không bằng thì nhảy đến nhãn <Label>.

Cú pháp: **LOOPZ <Label>**

→ Lệnh lặp LOOPZ tự động giảm CX một đơn vị, sau đó kiểm tra CX có bằng 0 hoặc cờ ZF có bật không (ZF = 1), nếu cả 2 đều không xảy ra thì nhảy đến nhãn <Label>.

Như vậy, cấu trúc lệnh có sẵn trong hợp ngữ để thực hiện các vòng lặp biết trước số lần lặp được viết như sau:

MOV CX, <Số lần lặp>

NHAPLAI:

... ;Các lệnh cần lặp

LOOP NHAPLAI

Ví dụ 7: Đoạn chương trình in ra màn hình các ký tự số từ 0 đến 9 với I là biến được khai báo giá trị ban đầu bằng 0 (tức I DB 0).

```
MOV    CX, 10      ;Số lần lặp
Lap:
    MOV    DL, I
    ADD    DL, 30h   ;Đổi số nguyên sang kí tự số tương ứng

    MOV    AH, 2     ;Chức năng số 2: hiện ký tự
    INT    21h
    INC    I         ;Tăng biến I lên 1
    LOOP   Lap
```

4. NỘI DUNG THỰC HÀNH

4.1. Cấu trúc rẽ nhánh IF

Đối với cấu trúc rẽ nhánh thì vị trí của nhãn sẽ đứng sau lệnh nhảy.

Cấu trúc: <Lệnh nhảy>

...

<Nhãn>

Ví dụ 8: Nhập một ký tự từ bàn phím, nếu ký tự nhập vào là ký tự thường thì đổi sang ký tự hoa và in ra màn hình.

Thuật toán như sau:

- Nhập một ký tự KT
- IF ($KT \leq 'z'$) AND ($KT \geq 'a'$) THEN Đổi KT sang in hoa
- Hiện KT ra màn hình.

Giải

```

TITLE    DOI KI TU
.MODEL   SMALL
.STACK   100H
.CODE
MAIN     PROC
    MOV   AH, 1      ;Nhập một ký tự
    INT   21h
    CMP   AL, 'z'
    JA    HienChu ;Nếu AL > 'z' thì hiện ký tự ra màn hình
    CMP   AL, 'a'
    JB    HienChu ;Nếu AL < 'a' thì hiện ký tự ra màn hình

    SUB   AL, 20h     ;đổi ký tự sang in hoa

    HienChu:
        MOV   AH, 2      ;Chức năng số 2: hiện ký tự
        MOV   DL, AL
        INT   21h

        MOV   AH, 4Ch ;Kết thúc
        INT   21h
MAIN ENDP
END MAIN

```

- Dịch và chạy chương trình ở những trường hợp khác nhau để xem kết quả.
- Nếu ký tự nhập vào không phải là ký tự chữ thì kết quả in ra màn hình là gì?
Tại sao?

4.2. Cấu trúc lặp

4.2.1. Lặp với số lần lặp không biết trước (*while, repeat*)

Đối với cấu trúc lặp nói chung thì vị trí của nhãn sẽ đứng trước lệnh nhảy.

Cấu trúc: <Nhãn>

 ...

 <Lệnh nhảy>

Ví dụ 9: Nhập một ký tự số từ bàn phím ('0', '1', ..., '9'). Nếu ký tự nhập vào không phải số thì nhập lại, ngược lại thì đổi nó sang số thập phân tương ứng.

Giải

```
TITLE VI DU LAP
.MODEL SMALL
.STACK 100H
.CODE
    MAIN PROC
        NhapLai:
        MOV AH, 1          ;Nhập 1 ký tự
        INT 21h

        CMP AL, '0'
        JB NhapLai         ;Nếu AL < '0' thì nhập lại
        CMP AL, '9'
        JA NhapLai         ;Nếu AL > '9' thì nhập lại

        SUB AL, 30h        ;Đổi sang số thập phân tương ứng
        ...                ;Các lệnh khác

        MOV AH, 4Ch        ;Kết thúc
        INT 21h
    MAIN ENDP
END MAIN
```

Giải thích:

- Ký tự '0' có mã ASCII bằng 30h
- Ký tự '1' có mã ASCII bằng 31h
- ...
- Ký tự '9' có mã ASCII bằng 39h

Để đổi từ ký tự số sang số thập phân tương ứng, ta lấy mã ASCII của nó đem trừ đi 30h.

4.2.2. Lặp với số lần lặp biết trước (For)

Ví dụ 10: In ra màn hình 10 số nguyên theo thứ tự : 0, 1, 2, 3, ..., 9

Giải:

```
TITLE VI DU LAP FOR
.MODEL SMALL
.STACK 100h
.DATA
    I DB 0          ;Khai báo giá trị biến I bằng 0
.CODE
    MAIN PROC
        MOV AX, @DATA
        MOV DS, AX
```

Lặp:

```
MOV    DL, I
ADD    DL, 30h    ;Đổi số nguyên sang ký tự số
MOV    AH, 2      ;Chức năng số 2: in ký tự
INT    21h
```

```
INC    I          ;Tăng biến I lên 1
CMP    I, 10
JNZ    Lap        ;Nếu I = 10 thì lặp lại
```

```
MOV    AH, 4Ch    ;Kết thúc
INT    21h
```

MAIN ENDP

END MAIN

- Dịch và chạy chương trình để xem kết quả.
- Sửa chương trình để in ra màn hình lần lượt các ký tự từ 'A' đến 'Z'.
- Sửa chương trình để in ra màn hình lần lượt các ký tự từ 'Z' đến 'A'.
- Sửa chương trình sao cho giữa các ký tự có 1 khoảng trống (Z Y ... B A).
- Dùng lệnh LOOP để viết lại chương trình đã được trình bày ở trên theo cấu trúc vòng lặp For.

5. BÀI TẬP

1. Viết chương trình nhập 1 ký tự từ màn hình và xuất câu thông báo tương ứng như sau:
 - Nếu ký tự nhập là 'S' hoặc 's' thì in ra "Good morning!"
 - Nếu ký tự nhập là 'T' hoặc 't' thì in ra "Good afternoon!"
 - Nếu ký tự nhập là 'C' hoặc 'c' thì in ra "Good evening!"
2. Viết chương trình phân biệt được 4 lại ký tự nhập từ bàn phím: "Ký tự chữ HOA", "Ký tự chữ thường", "Ký tự số" và "Ký tự khác".
3. Viết chương trình nhập từ bàn phím 1 ký tự chữ thường. Sau đó in ra màn hình lần lượt các ký tự từ ký tự nhận được đến ký tự 'z' sao cho giữa các ký tự có khoảng trống.
4. Viết chương trình cho phép nhập số nguyên n ($0 < n < 10$) và một ký tự k. In ra màn hình theo dạng sau:

Với n = 4 và k = @. In ra màn hình

```
@  @  @  @
@  @  @  @
@  @  @  @
@  @  @  @
```

5. Viết chương trình cho phép nhập số nguyên n ($0 < n < 10$) và một ký tự k . In ra màn hình theo dạng sau:

Với $n = 5$ và $k = @$. In ra màn hình

```
@   @   @   @   @   và   @   @   @   @   @
@               @               @   @
@               @               @
@               @               @   @
@   @   @   @   @   @   @   @   @   @   @
```

6. Viết chương trình cho phép nhập 2 số nguyên n và m ($0 < n, m < 10$) và một ký tự k . In ra màn hình theo dạng sau:

Với $n = 3$, $m = 4$ và $k = @$. In ra màn hình

```
@   @   @   @
@   @   @   @
@   @   @   @
```

BÀI THỰC HÀNH SỐ 4

NGĂN XẾP VÀ THỦ TỤC

1. MỤC TIÊU

Sau khi kết thúc bài thực hành này, sinh viên có thể:

- Hiểu được cơ chế hoạt động của ngăn xếp, quá trình gọi một thủ tục.
- Biết các sử dụng ngăn xếp, khai báo và gọi thủ tục.
- Biết các tạo và sử dụng macro.

2. KIẾN THỨC CHUẨN BỊ

- Lý thuyết về ngăn xếp, thủ tục và macro.
- Kết quả của các bài thực hành trước.

3. TÓM TẮT LÝ THUYẾT

3.1. Ngăn xếp

3.1.1. Một số lưu ý

- Ngăn xếp (stack) là vùng nhớ đặc biệt được truy cập theo cơ chế “vào sau ra trước” (**LIFO – Last In First Out**), nghĩa là dữ liệu nào đưa vào sau sẽ được lấy ra trước.
- Ngăn xếp gồm nhiều phần tử, mỗi phần tử là một từ (2 byte).
- Vị trí của ngăn xếp trong bộ nhớ được xác định bởi cặp thanh ghi SS:SP (SS chứa địa chỉ đoạn, SP chứa địa chỉ ô của đỉnh ngăn xếp). Khi chưa sử dụng, ngăn xếp rỗng, vị trí được xác định bởi SP lúc đó là đáy ngăn xếp.

3.1.2. Khai báo

.STACK <kích thước của ngăn xếp>

Ví dụ 1: Khai báo vùng ngăn xếp có kích thước 256 byte:

.STACK 100h

3.1.3. Cách sử dụng

3.1.3.1. Đưa dữ liệu vào ngăn xếp

Để cất dữ liệu vào đỉnh của ngăn xếp ta dùng lệnh **PUSH**, cách viết lệnh như sau:

PUSH <Toán hạng nguồn>

Trong đó: <Toán hạng nguồn> là một thanh ghi hay một biến có kích thước 16 bit.

Sau lệnh Push giá trị của toán hạng nguồn vẫn giữ nguyên.

Ví dụ 2: **PUSH AX** → Lệnh cất nội dung của thanh ghi AX vào ngăn xếp.

PUSH A → Lệnh cất nội dung của biến A vào ngăn xếp và A phải là biến kiểu word

Ngoài ra, còn có lệnh:

- PUSHW <hằng> : để đưa trực tiếp một hằng 16 bit vào ngăn xếp.
- PUSHF : đưa nội dung thanh ghi cờ vào đỉnh ngăn xếp.

3.1.3.2. Lấy dữ liệu khỏi ngăn xếp

Để lấy dữ liệu khỏi ngăn xếp ta dùng lệnh POP, cách viết lệnh như sau:

POP <Toán hạng nguồn>

Trong đó: <Toán hạng nguồn> là một thanh ghi hay một biến có kích thước 16 bit.

Việc lấy dữ liệu khỏi ngăn xếp sẽ đồng thời giải phóng ô nhớ đang chứa dữ liệu (tức là có thể dùng nó để chứa dữ liệu khác).

Ví dụ 3: POP AX → Lệnh lấy dữ liệu từ ngăn xếp đặt vào thanh ghi AX

POP A → Lệnh lấy dữ liệu từ ngăn xếp đặt vào biến A và A phải là biến kiểu word

Ngoài ra, còn có lệnh POPF để lấy giá trị (2 byte) ở đỉnh ngăn xếp đưa vào thanh ghi cờ.

3.1.4. Ứng dụng của ngăn xếp

Ví dụ 4: Hãy chuyển đổi nội dung của thanh ghi đoạn DS và thanh ghi đoạn CS.

→ Do không thể chuyển trực tiếp nội dung của 2 thanh ghi đoạn cho nhau (xem lại phần lệnh MOV) nên ta sử dụng ngăn xếp để làm trung gian: dữ liệu được chuyển từ DS vào ngăn xếp, sau đó lấy từ ngăn xếp chuyển vào ES:

PUSH DS ; Cất DS vào ngăn xếp

POP ES ; Lấy dữ liệu từ ngăn xếp đặt vào ES

Ví dụ 5: Viết chương trình nhập vào 1 ký tự từ bàn phím, hiện nó ở đầu dòng kế tiếp.

TITLE nhập và hiện thi ký tự bằng ngăn xếp

.MODEL SMALL

.STACK 100H

.CODE

MAIN PROC

MOV AH, 1 ; Chức năng số 1: nhập ký tự
INT 21h

PUSH AX ; Cất ký tự vào ngăn xếp

MOV AH, 2 ; Đưa con trỏ về đầu dòng tiếp theo
MOV DL, 0Dh
INT 21h
MOV DL, 0Ah
INT 21h

```

POP    DX          ;Lấy ký tự từ ngăn xếp đặt vào DX
INT    21h         ;Hiển thị ký tự

MOV    AH, 4Ch     ;Kết thúc
INT    21h

```

MAIN ENDP

END MAIN

Giải thích: Ký tự nhập vào được cất ở thanh ghi AL. Để đưa con trỏ xuống đầu dòng tiếp theo thì phải hiển thị hai ký tự có mã ASCII là 0Dh (CR: về đầu dòng) và 0Ah (LF: xuống dòng). Quá trình hiển thị hai ký tự này sẽ làm thanh ghi AL thay đổi (xem lại chức năng số 2 của ngắt 21h). Do đó, phải lưu ký tự ban đầu vào ngăn xếp trước khi xuống dòng, khi nào muốn hiển thị ký tự này thì lấy nó ra từ ngăn xếp.

Ví dụ 6: Viết lệnh thực hiện các công việc sau:

- Lưu nội dung thanh ghi cờ vào AX.
- Xoá thanh ghi cờ.

Ta không thể tác động đến thanh ghi cờ bằng các lệnh thông thường đã học như MOV, ADD, SUB, AND, OR,... Bộ vi xử lý 8086 cung cấp hai lệnh thao tác với thanh ghi cờ liên quan đến ngăn xếp là:

PUSHF ; cất nội dung thanh ghi cờ vào ngăn xếp.

POPF ; lấy dữ liệu từ ngăn xếp đặt vào thanh ghi cờ.

Sử dụng 2 lệnh này ta có thể giải quyết được yêu cầu đặt ra ở trên:

PUSHF

POP AX ; chuyển nội dung thanh ghi cờ từ ngăn xếp vào AX

XOR BX, BX ; xoá BX (BX = 0)

PUSH BX ; đặt giá trị 0 vào ngăn xếp

POPF ; chuyển giá trị 0 từ ngăn xếp vào thanh ghi cờ (xoá các cờ)

3.1.5. Cách thức làm việc của ngăn xếp

3.1.5.1. Kích thước ngăn xếp

Kích thước ngăn xếp được khai báo ở đầu chương trình hợp ngữ sau từ khoá .STACK

Ví dụ 7: .STACK 100h

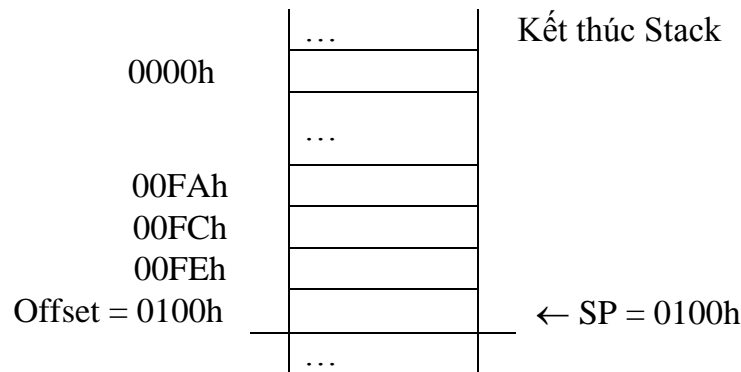
Khi đó ngăn xếp có kích thước bằng 100h byte (256 byte hay 128 word).

Mỗi lệnh Push sẽ chiếm 1 word của ngăn xếp, như vậy ngăn xếp khai báo như trên sẽ cho phép cất tối đa 128 lần. Người lập trình sẽ phải tính toán để khai báo ngăn xếp có kích thước hợp lý nhất (không quá thừa hay thiếu).

3.1.5.2. Cấu trúc của ngăn xếp

Dữ liệu được lấy ra khỏi ngăn xếp theo trình tự ngược lại so với khi cất vào, nghĩa là cất vào sau thì sẽ được lấy ra trước (LIFO – Last IN First Out).

Dữ liệu được cất vào ngăn xếp theo trật tự ngược lại so với các đoạn nhớ khác (từ địa chỉ cao xuống địa chỉ thấp). Giả sử khai báo ngăn xếp là .STACK 100h thì ngăn xếp sẽ bắt đầu tại địa chỉ offset = 0100h và kết thúc tại địa chỉ offset = 0000h. Dưới đây là mô hình của ngăn xếp khi chưa có dữ liệu:



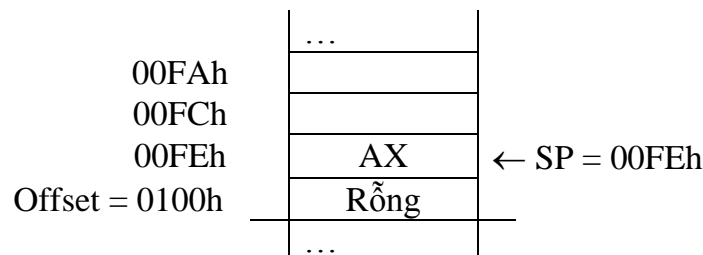
Khi ngăn xếp chưa có dữ liệu thì SP trở tới ô nhớ có địa chỉ cao nhất trong ngăn xếp. Sau mỗi lệnh PUSH thì SP sẽ giảm đi 2 để trở tới ô tiếp theo của ngăn xếp, dữ liệu sẽ được cất vào ô nhớ do SP trở tới.

Ví dụ 8:

❖ Cất nội dung 3 thanh ghi AX, BX, CX vào ngăn xếp:

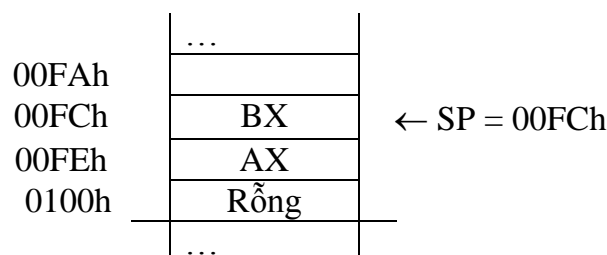
PUSH AX
PUSH BX
PUSH CX

- Sau lệnh PUSH AX



Do SP giảm đi 2 rồi mới cất thanh ghi AX vào ngăn xếp nên sẽ tạo ra một ô rỗng ở địa chỉ cao nhất.

- Sau lệnh PUSH BX



- Sau lệnh PUSH CX

	...	
00FAh	CX	← SP = 00FAh
00FCh	BX	
00FEh	AX	
0100h	Rỗng	
	...	

- ❖ Muốn lấy nội dung 3 thanh ghi ra khỏi ngăn xếp phải tiến hành theo trình tự ngược lại:

POP CX

POP BX

POP AX

- Sau lệnh POP CX

	...	
00FAh	-	← SP = 00FCh
00FCh	BX	
00FEh	AX	
0100h	Rỗng	
	...	

Dữ liệu tại ô nhớ do SP trở tới (SP = 00FAh) sẽ được nạp vào thanh ghi CX, sau đó SP sẽ tăng lên 2 để trở tới ô cao hơn. Ô nhớ chứa CX đã được giải phóng nên ta không cần quan tâm đến nội dung bên trong nó nữa.

- Sau lệnh POP BX

	...	
00FAh	-	← SP = 00FEh
00FCh	-	
00FEh	AX	
0100h	Rỗng	
	...	

Dữ liệu tại ô nhớ có offset = 00FCh được nạp vào thanh ghi BX.

- Sau lệnh POP ZX

	...	
00FAh	-	← SP = 0100h
00FCh	-	
00FEh	-	
0100h	Rỗng	
	...	

Dữ liệu tại ô nhớ có offset = 00FEh được nạp vào thanh ghi AX.

3.2. Thủ tục

3.2.1. Cấu trúc thủ tục

Ở bài thực hành số 1 đã trình bày về cấu trúc của một thủ tục và vị trí của nó trong chương trình hợp ngữ. Trong phần này ta sẽ đề cập tới cấu trúc của các thủ tục thông thường (không được chọn làm chương trình chính).

<Tên thủ tục> PROC ; bắt đầu thủ tục

Lệnh 1

Lệnh 2

Lệnh 3

...

RET ; trở về chương trình chính

<Tên thủ tục> ENDP ; kết thúc thủ tục

Ví dụ 9: Viết một thủ tục đưa con trỏ màn hình xuống đầu dòng kế tiếp.

Để đưa con trỏ xuống đầu dòng tiếp theo ta cần hiển thị ký tự CR (0Dh) và LF (0Ah). Ta đặt tên thủ tục này là Writeln.

Thủ tục được viết như sau:

```
Writeln PROC
    MOV AH, 2
    MOV DL, 0Dh
    INT 21h
    MOV DL, 0Ah
    INT 21h
    RET
Writeln ENDP
```

3.2.2. Sử dụng thủ tục

Để gọi một thủ tục từ chương trình chính ta sử dụng lệnh CALL, cú pháp lệnh như sau:

CALL <Tên thủ tục>

❖ Cấu trúc của một chương trình hợp ngữ có sử dụng thủ tục như sau:

TITLE <Tên chương trình>

.MODEL <Kiểu bộ nhớ>

.STACK <Kích thước ngăn xếp>

.DATA

<Khai báo dữ liệu>

.CODE

<Chương trình chính> PROC

Lệnh 1

Lệnh 2

Lệnh 3

...

CALL <Tên thủ tục> ;Gọi thủ tục

<Chương trình chính> ENDP

<Tên thủ tục> PROC

Lệnh 1

Lệnh 2

Lệnh 3

...

RET ; Trở về chương trình chính

<Tên thủ tục> ENDP

...Các thủ tục khác

END <chương trình chính>

Ví dụ 10: Viết chương trình nhập 1 ký tự từ bàn phím rồi hiện ký tự đó ở đầu dòng tiếp theo (có sử dụng thủ tục Writeln đã được viết ở ví dụ 9).

TITLE Vi du 10

.MODEL SMALL

.STACK 100H

.CODE

MAIN PROC

MOV AH, 1 ;Nhập 1 ký tự

INT 21h

PUSH AX ;Cất ký tự vào ngăn xếp

CALL Writeln ;Đưa con trỏ về đầu dòng tiếp theo

POP DX ;Lấy ký tự từ ngăn xếp đưa vào DX

MOV AH, 2 ; Hiện ký tự

INT 21h

MOV AH, 4Ch ;Kết thúc

INT 21h

MAIN ENDP

```

Writeln  PROC    ;T/tục đưa con trỏ về đầu dòng kế tiếp
        MOV     AH, 2
        MOV     DL, 0Dh
        INT     21h
        MOV     DL, 0Ah
        INT     21h
        RET
Writeln  ENDP
END MAIN

```

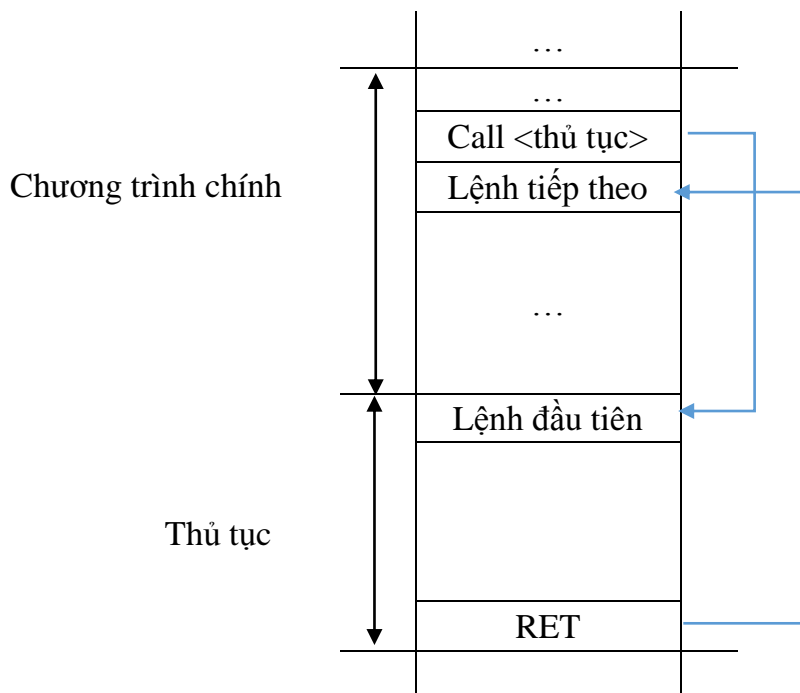
3.2.3. Quan hệ giữa thủ tục và ngăn xếp

Khi gọi lệnh CALL một thủ tục thì các lệnh của thủ tục đó sẽ được thi hành. Vậy làm cách nào để quay trở lại chương trình chính sau khi thủ tục thi hành xong?

Để hiểu được quá trình này ta cần nghiên cứu thêm về trình tự thực hiện lệnh của bộ vi xử lý 8086. Đoạn mã lệnh có địa chỉ segment nằm trong thanh ghi CS, còn offset của các lệnh sẽ được đặt vào thanh ghi IP (Instruction Pointer). Như vậy cặp thanh ghi CS:IP chứa địa chỉ của ô nhớ nào thì lệnh tại ô nhớ đó sẽ được thi hành.

Khi sử dụng lệnh CALL thì các công việc sau đây sẽ thực hiện:

- Cất địa chỉ của lệnh đứng sau lệnh CALL (trong chương trình chính) vào ngăn xếp.
- Nạp địa chỉ lệnh đầu tiên của thủ tục vào cặp thanh ghi CS:IP (tức là thi hành lệnh này).
- Lần lượt các lệnh trong thủ tục sẽ được thi hành cho tới khi gặp lệnh RET. Lệnh RET sẽ lấy địa chỉ lệnh từ ngăn xếp (do lệnh CALL cất trước đó) rồi nạp vào thanh ghi CS:IP. Như vậy quyền điều khiển đã được trả về chương trình chính (xem sơ đồ bên dưới).



3.3. Macro

3.3.1. Khai báo và gọi Macro

Macro là một dãy các câu lệnh của Assembler mà có thể xuất hiện nhiều lần trong một chương trình. Giống như chương trình con đó, mỗi macro có một tên riêng.

Cú pháp khai báo:

```
Name MACRO [Parameter_List]
    ;Mã lệnh trong macro
ENDM
```

Macro gồm 3 phần:

- **Phần header:** Dùng để khai báo một macro.

```
Name MACRO [Parameter_List]
```

Trong đó:

- Name: Tên của macro cần định nghĩa.
- Parameter_List: Là một dãy các tham số cách nhau bởi dấu phẩy.

- **Phần thân:** một tập hợp các câu lệnh của Assembler mà macro cần định nghĩa.

- **Phần cuối:** Là chỉ dẫn ENDM, dùng để đánh dấu điểm kết thúc của macro.

Ví dụ: Muốn xuất ký tự 'D' ra màn hình ta viết các lệnh

```
MOV AH, 02h
MOV DL, 'D'
INT 21h
```

Nếu muốn xuất thêm ký tự 'E' ra màn hình ta phải viết thêm các câu lệnh trên lại:

```
MOV AH, 02h
MOV DL, 'E'
INT 21h
```

Vậy nếu chương trình cần xuất nhiều ký tự khác nhau ra màn hình ta phải viết lại đoạn chương trình trên nhiều lần. Để rút gọn chương trình ta có thể viết chương trình con Out_Char.

```
; DL = Chứa ký tự cần xuất
Out_Char PROC
MOV AH, 02h
INT 21h
RET
Out_Char ENDP
```

Nếu dùng chương trình con thì chương trình có ngắn gọn hơn, nhưng chúng ta còn phải dùng đến hai câu lệnh. Nếu tham số càng nhiều ta càng viết nhiều câu lệnh hơn trước khi gọi chương trình con. Macro có thể giải quyết vấn đề này ngắn gọn hơn như sau:

```
@Out_Char MACRO Char
    MOV AH, 02h
    MOV DL, Char
    INT 21h
ENDM
```

Muốn xuất ký tự 'D' và 'E' ra màn hình ta chỉ cần viết đơn giản như sau:

```
@Out_Char 'D'
@Out_Char 'E'
```

Rõ ràng macro viết ngắn gọn hơn chương trình con nhiều.

4. NỘI DUNG THỰC HÀNH

BÀI THỰC HÀNH SỐ 5

MẢNG VÀ CHUỖI KÝ TỰ

1. MỤC TIÊU

Sau khi kết thúc bài thực hành này, sinh viên có thể:

- Viết được chương trình xử lý chuỗi ký tự bằng các lệnh xử lý chuỗi.
- Sử dụng thủ tục, macro vào các chương trình xử lý chuỗi ký tự.

2. KIẾN THỨC CHUẨN BỊ

- Bảng mã ASCII.
- Kết quả của các bài thực hành trước.
- Các chức năng 01h, 02h, 08h, 09h, 0Ah của ngắt 21h và các lệnh xử lý chuỗi như MOVSB, MOVSW, STOSB, STOSW, LODSB, LODSW,...

3. TÓM TẮT LÝ THUYẾT

3.1. Mảng 1 chiều (chuỗi)

3.1.1. Khai báo mảng

Mảng một chiều gồm một chuỗi liên tiếp các byte hay word trong bộ nhớ.

Ở bài thực hành số 2 ta đã từng sử dụng khai báo:

ChuoiKT DB 'KHOA CONG NGHE THONG TIN\$'

Thực chất khai báo này sẽ chiếm một vùng nhớ 25 ô nhớ trong đoạn dữ liệu và đặt vào đó các ký tự tương ứng:

'K'
'H'
'O'
'A'
' '
'C'
...

Cách khai báo như trên tương đương với cách khai báo sau đây:

ChuoiKT DB 'K', 'H', 'O', 'A', 'CONG', 'NGHE', 'THONG TIN\$'

Và cũng tương đương:

ChuoiKT DB 4Bh, 48h, 4Fh, 41h, 'CONG', 'NGHE', 'THONG TIN\$'

Các khai báo đó được gọi là khai báo liệt kê, tức là sẽ tập ra trong bộ nhớ một mảng có số lượng phần tử xác định, đồng thời khởi tạo luôn giá trị cho từng phần tử. Dưới đây sẽ đề cập đến các phương pháp khai báo tổng quát.

3.1.1.1. Khai báo mảng Byte

Mảng Byte là mảng mà mỗi phần tử có kích thước 1 byte.

Các cách khai báo:

Cách 1: <Tên mảng> DB <liệt kê các phần tử của mảng>

Cách 2: <Tên mảng> DB <số phần tử của mảng> DUP (giá trị khởi tạo)

Ví dụ 1:

A DB 10h, 12h, 30, 40

→ Khai báo trên tập mảng A có 4 phần tử, mỗi phần tử có kích thước 1 byte.

B DB 50 DUP (0)

→ Khai báo trên tạo mảng B có 50 phần tử, giá trị ban đầu của các phần tử bằng 0

C DB 100 DUP (?)

→ Khai báo trên tạo mảng C có 100 phần tử, không khởi tạo giá trị ban đầu cho các phần tử.

3.1.1.2. Khai báo mảng Word

Mảng Word là mảng mà mỗi phần tử có kích thước 1 word.

Các cách khai báo:

Cách 1: <Tên mảng> DW <liệt kê các phần tử của mảng>

Cách 2: <Tên mảng> DW <số phần tử của mảng> DUP (giá trị khởi tạo)

Ví dụ 2:

A DB 10h, 12h, 30, 40

→ Khai báo trên tập mảng A có 4 phần tử, mỗi phần tử có kích thước 1 word.

B DW 50 DUP (?)

3.1.2. Các phần tử của mảng một chiều

Tên mảng chính là một biến ứng với phần tử đầu tiên của mảng. Các phần tử tiếp theo có thể được xác định bằng cách lấy địa chỉ phần tử đứng trước cộng với kích thước của nó.

Ví dụ 3: M DB 10, 20, 30, 40

Các phần tử của mảng có thể được ký hiệu như sau: (kích thước của mỗi phần tử trong mảng M là 1 byte)

	Ký hiệu	Giá trị
Phần tử 1	M	10
Phần tử 2	M + 1	20
Phần tử 3	M + 2	30
Phần tử 4	M + 3	40

Ví dụ 4: N DW 1, 6, 20, 10, 15

Các phần tử của mảng có thể được ký hiệu như sau: (kích thước của mỗi phần tử trong mảng N là 2 byte)

	Ký hiệu	Giá trị
Phần tử 1	N	1
Phần tử 2	N + 2	6
Phần tử 3	N + 4	20
Phần tử 4	N + 6	10
Phần tử 5	N + 8	15

Ví dụ 5: Cho mảng A gồm 12 phần tử, các phần tử có kiểu Byte. Hãy đổi giá trị của phần tử đầu tiên và phần tử cuối cùng cho nhau.

Phần tử đầu tiên của mảng là: A

Phần tử cuối cùng của mảng là: A + 11

MOV AL, A

MOV BL, A + 11

MOV A, BL

MOV A+11, AL

3.1.3. Các chế độ định địa chỉ

Việc truy cập trực tiếp tới các phần tử của mảng thông qua cách viết: <Tên mảng> + <Khoảng cách> (như trong ví dụ 3, 4, 5) gây rất nhiều bất tiện trong lập trình. Một phương pháp khác, mềm dẻo hơn là sử dụng thanh ghi để chứa <Khoảng cách> hoặc chứa địa chỉ của từng phần tử. Bằng việc thay đổi nội dung của các thanh ghi → có thể truy nhập vào các phần tử khác của mảng.

Các thanh ghi có thể được sử dụng là: BX, BP, SI, DI.

3.1.3.1. Dùng thanh ghi chứa địa chỉ của phần tử

Giả sử thanh ghi SI đang chứa địa chỉ offset của một ô nhớ nào đó, cách viết [SI] sẽ trả về nội dung của ô nhớ đó.

Nếu sử dụng thanh ghi BX, DI, SI để chứa địa chỉ offset thì địa chỉ segment sẽ được chứa trong DS. Còn nếu sử dụng thanh ghi BP thì SS sẽ chứa segment.

Ví dụ 6: Cho mảng sau: A DB 10, 12, 3, 4, 9, 5, 7, 6. Hãy tính tổng các phần tử của mảng (cất tổng vào AL)

Ta sẽ sử dụng thanh ghi SI lần lượt trở tới từng phần tử của mảng để thực hiện phép tính tổng.

XOR AL, AL ; xoá AL trống để chứa tổng

LEA SI, A ; SI chứa địa chỉ offset của phần tử đầu tiên của mảng

```
MOV CX, 8      ; số lần lặp (mảng có 8 phần tử)
Lap:
    ADD AL, [SI] ; cộng phần tử của mảng vào AL
    INC SI      ; SI trở tới phần tử tiếp theo
LOOP Lap
```

→ Cách viết như trên được gọi là *Chế độ địa chỉ gián tiếp thanh ghi*.

3.1.3.2. Dùng thanh ghi chứa <Khoảng cách>

Trong phương pháp này, muốn truy nhập vào một phần tử của mảng thì cần phải biết được <Khoảng cách> từ phần tử đó tới đầu mảng. Các phần tử của mảng sẽ được ký hiệu như sau:

<Tên mảng> [Thanh ghi]

Trong đó: Thanh ghi sẽ chứa <Khoảng cách> của phần tử tính từ đầu mảng.

Ví dụ 7: Ký hiệu: A[BX]

Trong đó:

- A là tên mảng
- BX: là thanh ghi chứa <khoảng cách>
- Nếu BX = 0 thì A[BX] chính là phần tử đầu tiên của mảng.

Ví dụ 8: Viết lại đoạn chương trình ở ví dụ 7 bằng một cách khác.

```
XOR AL, AL      ; xoá AL rỗng để chứa tổng
XOR BX, BX      ; <Khoảng cách> = 0: phần tử đầu tiên
MOV CX, 8       ; số lần lặp (mảng có 8 phần tử)
Lap:
    ADD AL, A[BX] ; cộng phần tử của mảng vào AL
    INC BX        ; tăng <Khoảng cách> để trở tới phần
                  ; tử tiếp theo
LOOP Lap
```

Ngoài các ký hiệu A[BX] còn có thể sử dụng các ký hiệu khác tương đương như [A + BX], [BX + A], A + [BX], [BX] + A.

Nếu sử dụng thanh ghi BX hoặc BP trong cách viết trên thì gọi là *Chế độ địa chỉ cơ sở*, còn nếu sử dụng SI hay DI thì gọi là *Chế độ địa chỉ chỉ số*.

3.2. Các lệnh thao tác với chuỗi

Như đã nói ở phần trước, khi khai báo mảng một chiều thì nó sẽ chiếm một chuỗi liên tiếp các byte hay word trong đoạn dữ liệu. Muốn truy nhập vào các ô nhớ trong đoạn dữ liệu cần phải xác định được địa chỉ segment và offset của chúng.

Địa chỉ segment của dữ liệu được chứa trong thanh ghi DS. Trong các thao tác giữa hai mảng dữ liệu khác nhau, người ta thường sử dụng thêm thanh ghi đoạn ES để chứa segment của ô nhớ.

Địa chỉ offset có thể được chứa trong nhiều thanh ghi khác nhau. Trong các lệnh thao tác với chuỗi sắp trình bày, có 2 cặp thanh ghi hay được sử dụng là DS:SI và ES:DI. Nghĩa là, nếu dùng DS để chứa segment thì SI sẽ chứa offset, và nếu ES chứa segment thì DI sẽ chứa offset.

Để ES cũng chứa địa chỉ của đoạn dữ liệu giống với DS thì ở đầu chương trình chính phải có các lệnh:

```
MOV AX, @DATA
```

```
MOV DS, AX
```

```
MOV ES, AX
```

3.2.1. Lệnh chuyển chuỗi (Moving a String)

Lệnh này còn được gọi là lệnh sao chép chuỗi.

3.2.2.1. Chuyển một lần

Cú pháp lệnh:

Dạng 1: MOVSB

→ Lệnh có tác dụng sao chép 1 byte dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI

Dạng 2: MOVSW

→ Lệnh có tác dụng sao chép 1 word dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI

Ví dụ 1: Xét 2 chuỗi được khai báo như sau:

.DATA

Chuoi1 DB "Khoa CNTT\$"

Chuoi2 DB 10 DUP(?)

Hãy sao chép nội dung của Chuoi1 sang Chuoi2

Giải:

Để thực hiện yêu cầu trên ta sẽ lần lượt sao chép từng byte của Chuoi1 sang Chuoi2.

Chuoi1 10 byte	'K'	← DS:SI
	'h'	
	'o'	
	'a'	
	','	
	'C'	
	'N'	
	'T'	
	'T'	
	'\$'	
Chuoi2 10 byte	?	← ES:DI
	?	
	...	

Muốn sao chép byte đầu tiên (ký tự 'K') thì DS:SI phải chứa địa chỉ đầu của Chuoi1, ES:DI phải chứa địa chỉ đầu của Chuoi2. Điều này được thực hiện bởi các lệnh sau:

LEA SI, Chuoi1 ; SI chứa địa chỉ offset của Chuoi1
 LEA DI, Chuoi2 ; DI chứa địa chỉ offset của Chuoi2
 MOVSB ; Chuyển 1 byte

Mỗi chuỗi có độ dài là 10 byte nên phải lặp lại quá trình trên 10 lần thì mới sao chép xong.

Lưu ý:

- ✓ Mỗi khi sao chép xong 1 byte thì phải tăng SI và DI lên 1 để nó trở tới ô nhớ tiếp theo. Sau mỗi lệnh MOVSB thì SI và DI sẽ tự động tăng lên 1 nếu cờ DF = 0 (SI và DI sẽ tự động giảm nếu cờ DF = 1). Như vậy, vấn đề là phải xóa được DF trước khi thi hành lệnh MOVSB. Điều này được thực hiện nhờ lệnh CLD (Clear Direction Flag):

LEA SI, Chuoi1 ; SI chứa địa chỉ offset của Chuoi1
 LEA DI, Chuoi2 ; DI chứa địa chỉ offset của Chuoi2
 CLD ; Xóa cờ định hướng, DF = 0
 MOVSB ; Chuyển 1 byte

- ✓ Ngược lại với lệnh CLD là lệnh STD (Set Direction Flag), lệnh này sẽ thiết lập cờ DF = 1. Ta có thể sử dụng lệnh STD để chuyển các byte dữ liệu theo chiều ngược lại.

Chương trình đầy đủ:

TITLE Ví dụ Chuoi

.MODEL SMALL

.STACK 100H

.DATA

Chuoi1 DB "Khoa CNTT\$"

Chuoi2 DB 10 DUP (?)

.CODE

MAIN PROC

MOV AX, @DATA

MOV DS, AX

MOV ES, AX ;DS và ES chứa segment của đoạn dữ liệu

MOV CX, 10 ;Số lần lặp

LEA SI, Chuoi1 ;SI chứa địa chỉ offset của Chuoi1

LEA DI, Chuoi2 ;DI chứa địa chỉ offset của Chuoi2

CLD ;Xóa cờ định hướng, DF = 0

Lap:

MOVSB ;Thực hiện lặp 10 lần

LOOP Lap

MOV AH, 9h ;Hiển thị Chuoi2 để kiểm tra kết quả

LEA DX, Chuoi2

INT 21h

MOV AH, 4Ch ;Kết thúc

INT 21h

MAIN ENDP

END MAIN

Ví dụ 2: Làm lại ví dụ 1 bằng cách dùng lệnh MOVSW

Giải: Do lệnh MOVSW mỗi lần sao chép 2 byte nên chỉ phải thực hiện lặp 5 lần, các lệnh cụ thể như sau:

MOV CX, 5 ;Số lần lặp

LEA SI, Chuoi1 ;SI chứa địa chỉ offset của Chuoi1

LEA DI, Chuoi2 ;DI chứa địa chỉ offset của Chuoi2

CLD ;Xóa cờ định hướng DF = 0

Lap:

MOVSW ; Thực hiện 5 lần

LOOP Lap

3.2.2.2. Chuyển nhiều lần

Các lệnh MOVSB và MOVSW mỗi lần chỉ chuyển được 1 byte hay 1 word, do đó khi cần chuyển nhiều dữ liệu thì phải sử dụng vòng lặp, điều này làm chương trình phức tạp thêm. Thay vì sử dụng vòng lặp, ta có thể sử dụng các lệnh chuyển nhiều lần dưới đây.

Cú pháp lệnh:

Dạng 1: REP MOVSB

→ Lệnh có tác dụng sao chép nhiều byte dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI, số byte cần chuyển chứa trong thanh ghi CX.

Dạng 2: REP MOVSW

→ Lệnh có tác dụng sao chép nhiều word dữ liệu từ ô nhớ có địa chỉ DS:SI sang ô nhớ có địa chỉ ES:DI, số word cần chuyển chứa trong thanh ghi CX.

Ví dụ: Để thực hiện việc sao chép nội dung Chuoi1 sang Chuoi2 trong ví dụ ở phần trước, ta có thể viết lại các lệnh như sau:

```
MOV CX, 10 ;Số byte cần chuyển
LEA SI, Chuoi1 ;SI chứa địa chỉ offset của Chuoi1
LEA DI, Chuoi2 ;SI chứa địa chỉ offset của Chuoi2
CLD ;Xóa cờ định hướng, DF = 0
REP MOVSB
```

hoặc:

```
MOV CX, 5 ;Số byte cần chuyển
LEA SI, Chuoi1 ;SI chứa địa chỉ offset của Chuoi1
LEA DI, Chuoi2 ;SI chứa địa chỉ offset của Chuoi2
CLD ;Xóa cờ định hướng, DF = 0
REP MOVSW
```

3.2.2. *Lệnh chuyển dữ liệu từ thanh ghi vào chuỗi (Store a String)*

Lệnh này còn được gọi là lệnh nạp chuỗi

Cú pháp lệnh:

Dạng 1: LODSB

→ Lệnh có tác dụng chuyển 1 byte dữ liệu từ ô nhớ có địa chỉ DS:SI vào thanh ghi AL

Dạng 2: LODSW

→ Lệnh có tác dụng chuyển 1 word dữ liệu từ ô nhớ có địa chỉ DS:SI vào thanh ghi AX

Ví dụ: Xét chuỗi sau đây:

.DATA

ChuoiKT DB "Viet Nam"

Hãy hiển thị chuỗi ra màn hình

Giải:

Vì chuỗi không kết thúc bằng dấu '\$' nên không thể hiển thị chuỗi bằng chức năng số 9 của ngắt 21h. Ta sẽ cho hiển thị lần lượt các ký tự của chuỗi bằng chức năng số 2 của ngắt 21h (các tham số: AH = 2, DL = Mã ASCII của ký tự cần hiển thị). Chuỗi có 8 ký tự nên cần 8 lần lặp.

Đầu tiên cần chuyển từng ký tự của chuỗi vào thanh ghi AL bằng lệnh LODSB, sau đó chuyển từ AL sang DL, rồi gọi chức năng số 2 của ngắt 21h.

Chương trình đầy đủ:

TITLE Ví dụ

.MODEL SMALL

.STACK 100H

.DATA

ChuoiKT DB "Viet Nam"

.CODE

MAIN PROC

MOV AX, @DATA

MOV DS, AX ;DS chứa segment của đoạn dữ liệu

MOV CX, 8 ;Số lần lặp bằng 8

LEA SI, ChuoiKT ;SI chứa địa chỉ offset của ChuoiKT

CLD ;Xóa cờ định hướng, DF = 0

Lap:

```
        LODSB                ;Chuyển ký tự từ chuỗi vào AL
                                ;SI tự động tăng lên 1 (để trỏ tới ký tự tiếp theo)

        MOV DL, AL           ;Chuyển ký tự vào DL
        MOV AH, 2            ;Hiển thị ký tự
        INT 21h

    LOOP Lap
```

```
    MOV AH, 4Ch              ;Kết thúc
    INT 21h

    MAIN ENDP
```

END MAIN

4. NỘI DUNG THỰC HÀNH

Bài 1: So sánh hai chuỗi oldpass và newpass. Nếu hai chuỗi này giống nhau thì kết luận giống nhau và ngược lại.

Bài 2: Di chuyển 33 bytes từ nội dung của string1 sang string2, sau đó in nội dung của string2 ra màn hình.

Bài 3: Tìm ký tự “A” có trong một chuỗi ký tự bất kỳ, nếu có thì in ra câu thông báo là có ký tự “A” trong chuỗi ký tự và ngược lại.

Bài 4: Giả sử có các khai báo sau:

```
String1 DB 'FGHIJ'
String2 DB 'ABCDE'

        DB 5 DUP(?)
```

Hãy viết các lệnh chuyển String1 vào chuỗi String2 để tạo ra chuỗi ‘ABCDEFGHIIJ’

Bài 5: Viết các lệnh đổi chuỗi String1 và String2 trong bài tập 4 (có thể sử dụng 5 byte sau của Chuoi2 để làm vùng nhớ trung gian).

Bài 6: Giả sử đã khai báo chuỗi sau đây:

```
String DB 'TH*C G*S* AR* b*ASTS'
```

Hãy viết lệnh đổi các dấu * thành chữ ‘E’.

Bài 7: Giả sử đã khai báo chuỗi sau đây:

```
String1 DB 'T H I S I S A T E S T'
String2 DB 11 DUP (?)
```

Viết các lệnh chép chuỗi String1 vào chuỗi String2 và bỏ đi những khoảng trắng