



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



**BÀI GIẢNG MÔN**

# **KỸ THUẬT VI XỬ LÝ**

**CHƯƠNG 4 – LẬP TRÌNH HỢP NGỮ  
VỚI 8086/8088**

## NỘI DUNG

1. Giới thiệu về hợp ngữ
2. Cú pháp của chương trình hợp ngữ
3. Dữ liệu cho chương trình hợp ngữ
4. Biến và hằng
5. Khung chương trình hợp ngữ
6. Các cấu trúc điều khiển
7. Giới thiệu phần mềm mô phỏng emu8086
8. Một số ví dụ
9. Chương trình con
10. Marco
11. Giới thiệu thiết bị ảo – Đền giao thông

## 3.1. Giới thiệu về hợp ngữ

- ❖ Hợp ngữ (Assembler) là ngôn ngữ lập trình bậc thấp, chỉ cao hơn ngôn ngữ máy;
- ❖ Hợp ngữ là ngôn ngữ gắn liền với các dòng vi xử lý (processor specific).
  - Các lệnh dùng trong hợp ngữ là lệnh của VXL
  - Chương trình hợp ngữ viết cho một VXL có thể không hoạt động trên VXL khác.
- ❖ Chương trình hợp ngữ khi dịch ra mã máy có kích thước nhỏ gọn, chiếm ít không gian nhớ.
- ❖ Hợp ngữ thường được sử dụng để viết:
  - Các trình điều khiển thiết bị
  - Các môđun chương trình cho vi điều khiển
  - Một số môđun trong nhân HĐH (đòi hỏi kích thước nhỏ gọn và tốc độ cao)

## 3.2. Cú pháp của chương trình hợp ngữ

- ❖ Trong chương trình hợp ngữ, mỗi lệnh được đặt trên một dòng – dòng lệnh;
- ❖ Lệnh có 2 dạng:
  - Lệnh thật: là các lệnh gọi nhớ của VXL
    - VD: MOV, SUB, ADD,...
    - Khi dịch, lệnh gọi nhớ được dịch ra mã máy
  - Lệnh giả: là các hướng dẫn chương trình dịch
    - VD: MAIN PROC, .DATA, END MAIN,...
    - Khi dịch, lệnh giả không được dịch ra mã máy mà chỉ có tác dụng định hướng cho chương trình dịch.
- ❖ Không phân biệt chữ hoa hay chữ thường trong các dòng lệnh hợp ngữ khi được dịch.

## 3.2. Cú pháp của chương trình hợp ngữ

### ❖ Cấu trúc dòng lệnh hợp ngữ:

[Tên] [Mã lệnh] [Các toán hạng] [Chú giải]

START: MOV AH, 100 ; Chuyển 100 vào thanh ghi AH

### ❖ Các trường của dòng lệnh:

#### ▪ Tên:

- Là nhãn, tên biến, hằng hoặc thủ tục. Sau nhãn là dấu hai chấm (:)
- Các tên sẽ được chương trình dịch gán địa chỉ ô nhớ.
- Tên chỉ có thể gồm các chữ cái, chữ số, dấu gạch dưới và phải bắt đầu bằng 1 chữ cái

#### ▪ Mã lệnh: có thể gồm lệnh thật và giả

## 3.2. Cú pháp của chương trình hợp ngữ

### ❖ Các trường của dòng lệnh:

#### ■ Toán hạng:

- Số lượng toán hạng phụ thuộc vào lệnh cụ thể
- Có thể có 0, 1 và 2 toán hạng.

#### ■ Chú giải:

- Là chú thích cho dòng lệnh
- Bắt đầu bằng dấu chấm phẩy (;)

START: MOV AH, 100 ; Chuyển 100 vào thanh ghi AH

↑                    ↑                    ↑                    ↑

Tên                    Mã lệnh    Toán hạng                    Chú giải

## 3.3. Dữ liệu cho chương trình hợp ngữ

### ❖ Dữ liệu số:

- Thập phân: 0-9
- Thập lục phân: 0-9, A-F
  - Bắt đầu bằng 1 chữ (A-F) thì thêm 0 vào đầu
  - Thêm ký hiệu H (Hexa) ở cuối
  - VD: 80H, 0F9H
- Nhị phân: 0-1
  - Thêm ký hiệu B (Binary) ở cuối
  - VD: 0111B, 1000B

### ❖ Dữ liệu ký tự:

- Bao trong cặp nháy đơn hoặc kép
- Có thể dùng ở dạng ký tự hoặc mã ASCII
  - 'A' = 65, 'a' = 97

## 3.4. Hằng và biến

### ❖ Hằng (constant):

- Là các đại lượng không thay đổi giá trị
- Hai loại hằng:
  - Hằng giá trị: ví dụ 100, 'A'
  - Hằng có tên: ví dụ MAX\_VALUE
- Định nghĩa hằng có tên:

<Tên hằng> EQU <Giá trị>

VD:

MAX	EQU	100
ENTER	EQU	13
ESC	EQU	27



## 3.4. Hằng và biến

### ❖ Biến (variable):

- Là các đại lượng có thể thay đổi giá trị
- Các loại biến:
  - Biến đơn
  - Biến mảng
  - Biến chuỗi ký tự
- Khi dịch biến được chuyển thành địa chỉ ô nhớ

## 3.4. Hằng và biến

### ❖ Định nghĩa biến đơn:

Tên biến	DB	Giá trị khởi đầu: Định nghĩa biến byte
Tên biến	DW	Giá trị khởi đầu: Định nghĩa biến word
Tên biến	DD	Giá trị khởi đầu: Định nghĩa biến double word

Ví dụ:

X	DB	10	; Khai báo biến X và khởi trị 10
Y	DW	?	; Khai báo biến Y và không khởi trị
Z	DD	1000	; Khai báo biến X và khởi trị 1000

## 3.4. Hằng và biến

### ❖ Định nghĩa biến mảng:

Tên mảng DB      D/s giá trị khởi đầu

Tên mảng DB      Số phần tử      Dup(Giá trị khởi đầu)

Tên mảng DB      Số phần tử      Dup(?)

Định nghĩa tương tự cho các kiểu DW và DD

Ví dụ:

X    DB      10, 2, 5, 6, 1      ; Khai báo mảng X gồm 5 phần tử có khởi trị

Y    DB      5      DUP(0)    ; Khai báo mảng Y gồm 5 phần tử khởi trị 0

Z    DB      5      DUP(?)    ; Khai báo mảng Z gồm 5 phần tử không khởi trị

## 3.4. Hằng và biến

- ❖ Định nghĩa biến chuỗi ký tự: có thể được định nghĩa như một chuỗi ký tự hoặc một mảng các ký tự

Ví dụ:

str1 DB 'string'

str2 DB 73H, 74H, 72H, 69H, 6EH, 67H

str3 DB 73H, 74H, 'r', 'i', 69H, 6EH, 67H

## 3.5. Khung chương trình hợp ngữ

### ❖ Khai báo qui mô sử dụng bộ nhớ:

.Model <Kiểu kích thước bộ nhớ>

### ❖ Các kiểu kích thước bộ nhớ:

- Tiny (hẹp): mã lệnh và dữ liệu gói gọn trong một đoạn
- Small (nhỏ): mã lệnh gói gọn trong một đoạn, dữ liệu gói gọn trong một đoạn
- Medium (vừa): mã lệnh không gói gọn trong một đoạn, dữ liệu gói gọn trong một đoạn
- Compact (gọn): mã lệnh gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn
- Large (lớn): mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, không có mảng lớn hơn 64K
- Huge (rất lớn): mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, có mảng lớn hơn 64K.

## 3.5. Khung chương trình hợp ngữ

### ❖ Khai báo đoạn ngăn xếp:

.Stack <Kích thước ngăn xếp>

VD:

.Stack 100H; khai báo kích thước ngăn xếp 100H=256 byte

### ❖ Khai báo đoạn dữ liệu:

.Data

;Định nghĩa các biến và hằng

;Tất cả các biến và hằng phải được khai báo ở đoạn dữ liệu

VD:

.Data

MSG	DB	'Hello!\$'
-----	----	------------

ENTER	DB	13
-------	----	----

MAX	DW	1000
-----	----	------

## 3.5. Khung chương trình hợp ngữ

❖ Khai báo đoạn mã:

.Code

; Các lệnh của chương trình

VD:

.Code

MAIN Proc ; bắt đầu chương trình chính

; các lệnh của chương trình chính

MAIN Endp ; kết thúc chương trình chính

; các chương trình con – nếu có

End MAIN

## 3.5. Khung chương trình hợp ngữ - tổng hợp

.Model Small

.Stack 100H

.Data

; khai báo các biến và hằng

.Code

MAIN Proc

; khởi đầu cho thanh ghi DS

MOV AX, @Data ; nạp địa chỉ đoạn dữ liệu vào AX

MOV DS, AX ; nạp địa chỉ đoạn dữ liệu vào DS

; các lệnh của chương trình chính

; kết thúc, trở về chương trình gọi dùng hàm 4CH của ngắt 21H

MOV AH, 4CH

INT 21H

MAIN Endp

; các chương trình con (nếu có)

END MAIN ; kết thúc toàn bộ chương trình



## 3.5. Khung chương trình hợp ngữ - ví dụ

; Chương trình in ra thông điệp: Hello World!

.Model Small

.Stack 100H

.Data

  ; khai báo các biến và hằng

  CRLF DB 13, 10, '\*' ; xuống dòng

  MSG DB 'Hello World!\$'

.Code

MAIN Proc

  ; khởi đầu cho thanh ghi DS

  MOV AX, @Data ; nạp địa chỉ đoạn dữ liệu vào AX

  MOV DS, AX     ; nạp địa chỉ đoạn dữ liệu vào DS

## 3.5. Khung chương trình hợp ngữ - ví dụ

; xuống dòng

MOV AH, 9

LEA DX, CRLF ; nạp địa chỉ CRLF vào DX

INT 21H

; hiện lời chào dùng hàm 9 của ngắt 21H

MOV AH, 9

LEA DX, MSG ; nạp địa chỉ thông điệp vào DX

INT 21H ; hiện thông điệp

; kết thúc, trở về chương trình gọi dùng hàm 4CH của ngắt 21H

MOV AH, 4CH

INT 21H

MAIN Endp

END MAIN

## 3.5. Khung chương trình hợp ngữ - ví dụ

; Chương trình in ra thông điệp: Hello  
World!

.Model Small

.Stack 100H

.Data

; khai báo các biến và hằng

CRLF DB 13, 10, '\*' ; xuống dòng

MSG DB 'Hello World!\$'

.Code

MAIN Proc

; khởi đầu cho thanh ghi DS

MOV AX, @Data ; nạp địa chỉ đoạn  
dữ liệu vào AX

MOV DS, AX ; nạp địa chỉ đoạn  
dữ liệu vào DS

; xuống dòng

MOV AH, 9

LEA DX, CRLF ; nạp địa chỉ CRLF vào DX

INT 21H

; hiện lời chào dùng hàm 9 của ngắt 21H

MOV AH, 9

LEA DX, MSG ; nạp địa chỉ thông điệp vào  
DX

INT 21H ; hiện thông điệp

; kết thúc, trở về chương trình gọi dùng hàm  
4CH của ngắt 21H

MOV AH, 4CH

INT 21H

MAIN Endp

END MAIN

## 3.6. Các cấu trúc điều khiển

### ❖ Cấu trúc lựa chọn

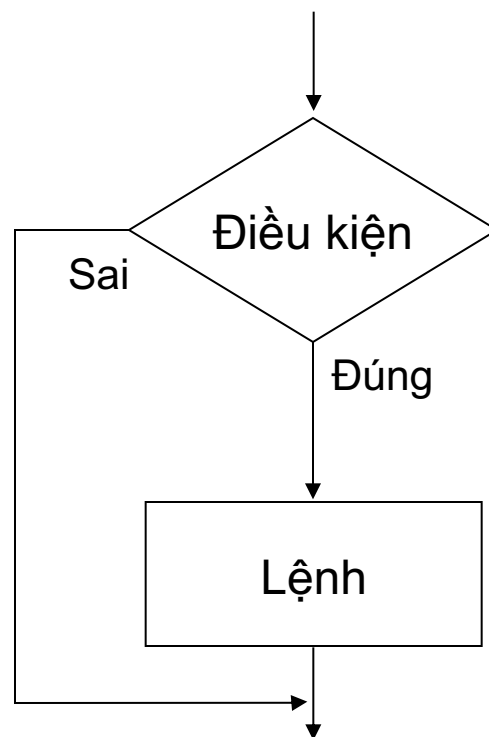
- Rẽ nhánh kiểu IF ... THEN
- Rẽ nhánh kiểu IF ... THEN ... ELSE
- Rẽ nhiều nhánh

### ❖ Cấu trúc lặp

- Lặp kiểu for
- Lặp kiểu repeat ... until

## 3.6. Các cấu trúc điều khiển - IF ... THEN

- ❖ IF điều kiện THEN thao tác
- ❖ Gán BX giá trị tuyệt đối AX
  1. `CMP AX,0`
  2. `JNL GAN`
  3. `NEG AX`
  4. `GAN: MOV BX, AX`



## 3.6. Các cấu trúc điều khiển - IF ... THEN ... ELSE

Gán bit dấu của AX cho CL:

CMP AX, 0 ; AX > 0 ?

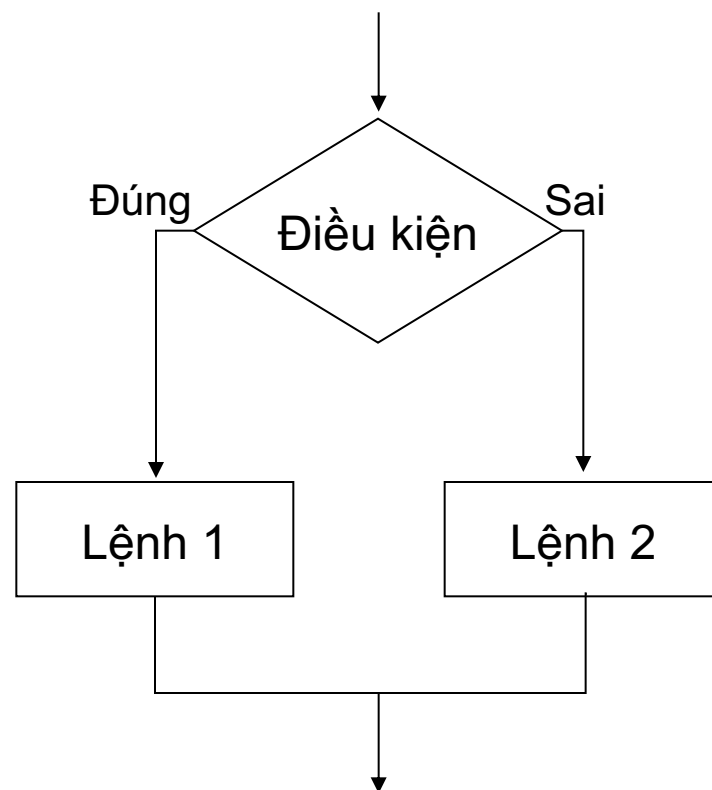
JNS DG ; đúng

MOV CL, 1 ; không,  $CL \leftarrow 1$

JMP RA ; nhảy qua nhánh kia

DG: MOV CL, 0 ;  $CL \leftarrow 0$

RA:



## 3.6. Các cấu trúc điều khiển - Rẽ nhiều nhánh

**Gán giá trị cho CX theo qui tắc:**

- Nếu  $AX < 0$  thì  $CX = -1$
- Nếu  $AX = 0$  thì  $CX = 0$
- Nếu  $AX > 0$  thì  $CX = 1$

CMP AX, 0

JL AM

JE KHONG

JG DUONG

AM: MOV CX, -1

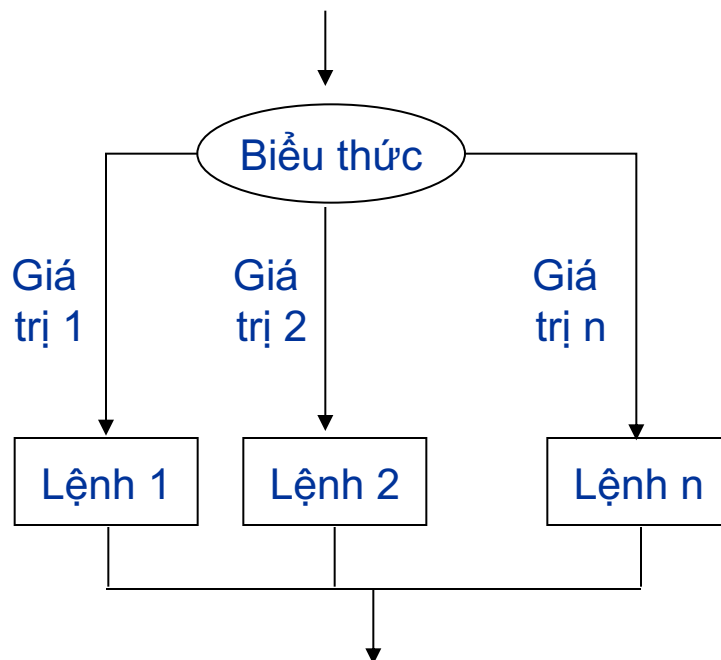
JMP RA

DUONG: MOV CX, 1

JMP RA

KHONG: MOV CX, 0

RA:

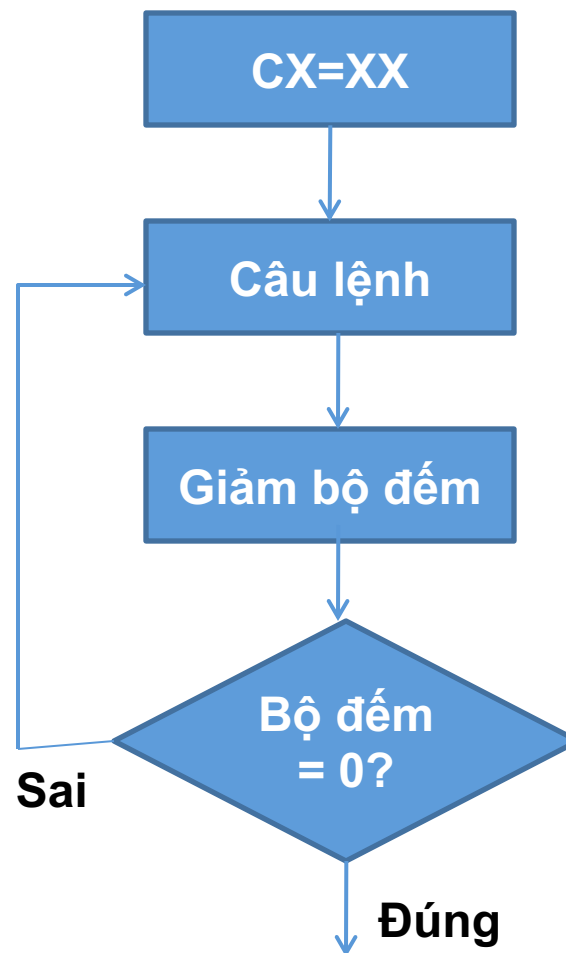


## 3.6. Các cấu trúc điều khiển – Lặp kiểu for

❖ Sử dụng lệnh LOOP

❖ Số lần lặp CX

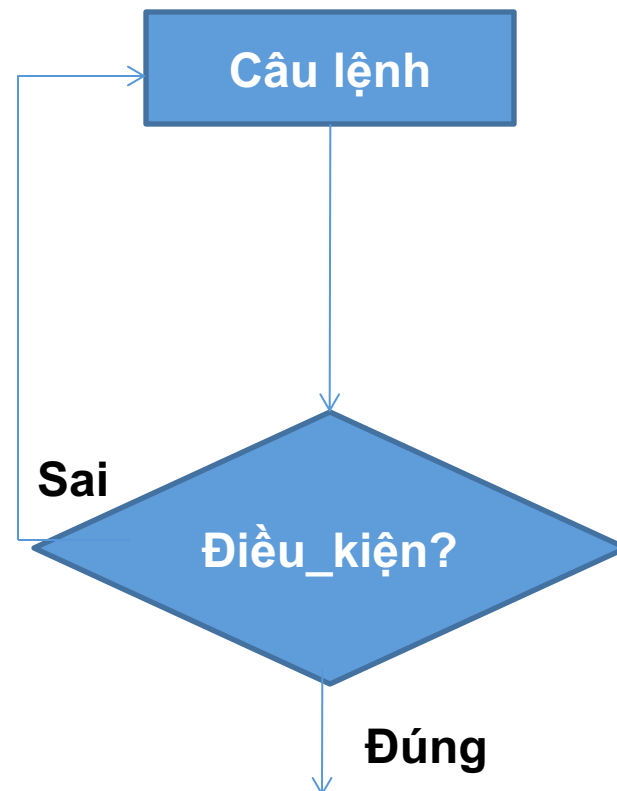
1. MOV CX,10
2. MOV AH,2
3. MOV DL,'9'
4. Hien: INT 21H
5. LOOP Hien



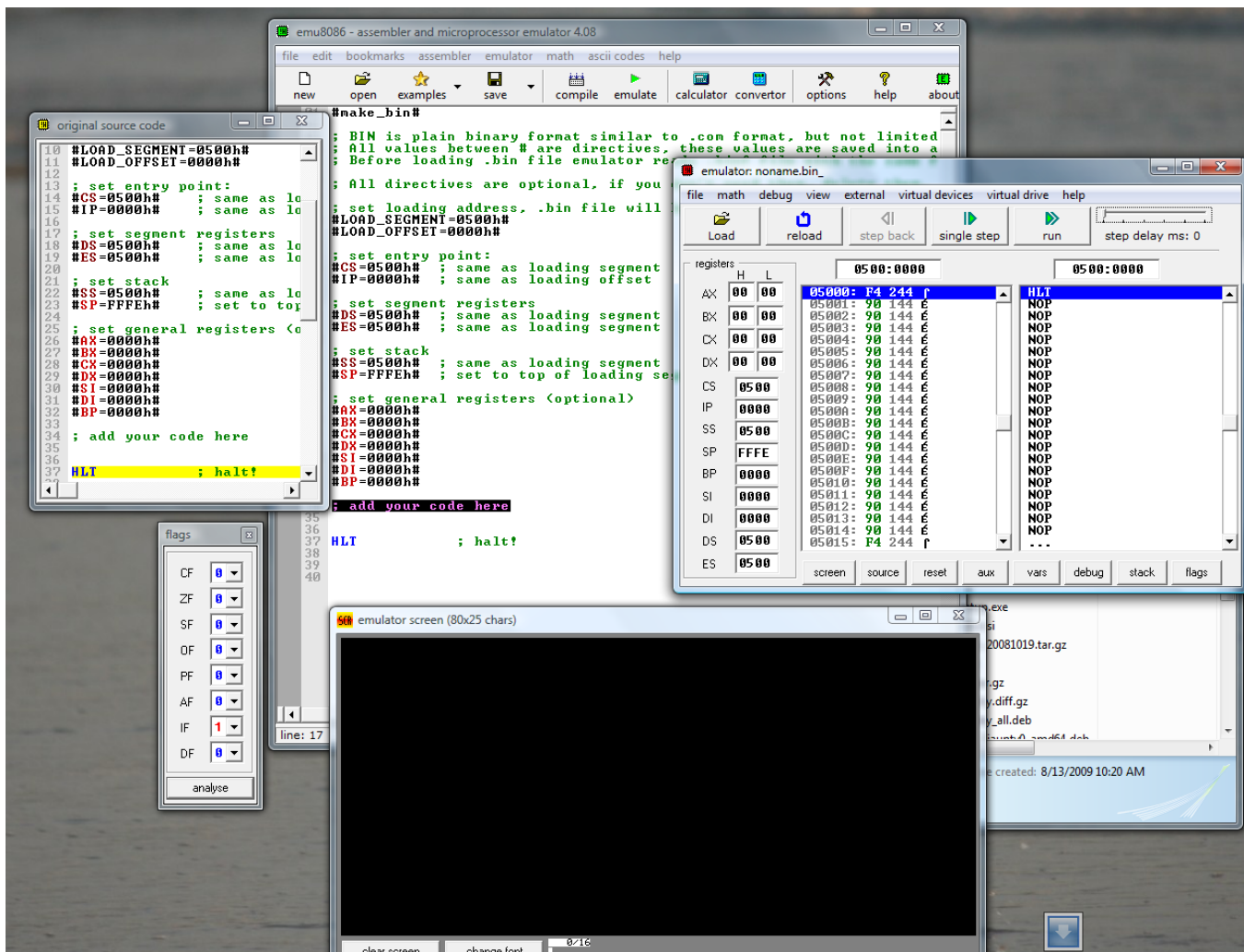


### 3.6. Các cấu trúc điều khiển – Lặp kiểu repeat ... until

1. ...
2. Tiếp:...
3. ....
4. CMP X,Y; điều kiện
5. Quay lại Tiếp nếu điều\_kiện=sai;



### 3.7. Giới thiệu phần mềm mô phỏng emu8086



### 3.8. Một số ví dụ - Một số dịch vụ của ngắt 21H

❖ **Hàm 1 của ngắt INT 21H:** đọc 1 ký tự từ bàn phím

Vào: AH = 1

Ra: AL = mã ASCII của ký tự cần hiện thị

AL = 0 khi ký tự gõ vào là phím chức năng

❖ **Hàm 2 của ngắt INT 21H:** hiện 1 ký tự lên màn hình

Vào: AH = 2

DL = mã ASCH của ký tự cần hiện thị.

Ra: Không

## 3.8. Một số ví dụ - Một số dịch vụ của ngắt 21H

- ❖ **Hàm 9 của ngắt INT 21H:** hiện chuỗi ký tự với \$ ở cuối lên màn hình

Vào: AH = 9

DX = địa chỉ lệch của chuỗi ký tự cần hiện thị.

Ra: Không

- ❖ **Hàm 4CH của ngắt INT 21H:** kết thúc chương trình kiểu EXE

Vào: AH = 4CH

Ra: Không

## VD1- Hiện các lời chào ta và tây

. Model Small

. Stack 100

. Data

CRLF DB 13, 10, '\$'

Chao tay DB 'hello!\$'

ChaoTa DB 'Chao ban!\$'

. Code

MAIN Proc

MOV AX, @ Data ; khởi đầu thanh ghi DS

MOV DS, AX

; hiện thị lời chào dùng hàm 9 của INT 21H

MOV AH, 9

LEA DX, ChaoTay

INT 21H

## VD1- Hiện các lời chào ta và tây

```
; cách 5 dòng dùng hàm 9 của INT 21H
LEA DX, CRLF
MOV CX, 6          ;CX chứa số dòng cách +1
LAP: INT 21H
LOOP LAP
; hiện thị lời chào dùng hàm 9 của INT 21H
LEA DX, ChaoTa
INT 21H
; trở về DOS dùng hàm 4 CH của INT 21H
MOV AH, 4CH
INT 21H
MAIN Endp
END MAIN
```

## VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

- ❖ Cho một chuỗi bất kỳ
- ❖ Đọc từng ký tự của chuỗi cũ --> chuyển sang chuỗi mới + viết hoa.
- ❖ Để giải bài toán này có thể ứng dụng các lệnh LODSB và STOSB với chuỗi đã cho. Các bước thực hiện:
  - Lấy từng ký tự của chuỗi gốc (cũ) bằng lệnh LODSB,
  - Biến đổi thành chữ hoa bằng cách trừ đi 20H,
  - Gán ký tự đã biến đổi vào chuỗi đích (mới) bằng lệnh STOSB.

## VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

```
.Model small
.Stack 100H
.Data
; source string
str1 DB 'a','5', 'B', '?', 'd', 'g', 'P','N','k','*'
      DB 10,13,'$'
; destination string
str2 DB 10 DUP(' ')
      DB '$'
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    mov es,ax
```



## VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

; make SI points to str1 and DI to str2

lea si, str1

lea di, str2

cld ; định hướng lùi

mov cx, 10 ; Chứa số byte phải đổi

Start:

lodsb ; lấy 1 ký tự chuỗi cũ

; check if it is lower case

cmp al, 'a'

jl NotLowerCase

cmp al, 'z'

jg NotLowerCase

; is lower case, convert to upper case

sub al, 20H

; store to new string

NotLowerCase: stosb

loop Start

## VD2- Đổi các ký tự thường trong 1 chuỗi thành chữ hoa

; print the original string

lea dx, str1

mov ah, 9

int 21H

; print the output

lea dx, str2

mov ah, 9

int 21H

; end program

mov ah, 4CH

int 21H

main endp

end main

## VD3- Tìm số lớn nhất trong 1 dãy

```
.Model small
.Stack 100H
.Data
; source string
list DB 1,4,0,9,7,2,4,6,2,5
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    cld
    mov cx, 9
    lea si, list    ; si points to list
    mov bl, [si]    ; max <-- 1st element
    inc si
```

## VD3- Tìm số lớn nhất trong 1 dãy

Start:

lodsb

cmp al, bl

jle BYPASS

mov bl, al; al>bl --> bl to store new max

BYPASS:

loop Start

; print the max

add bl, '0' ; digit to char

mov dl,bl

mov ah, 2

int 21H

; end program

mov ah, 4CH

int 21H

main endp

End Main

## 3.9. Tạo và sử dụng chương trình con

- ❖ Chương trình con (còn gọi là thủ tục (procedure) hoặc hàm (function)):
  - Thường gồm một nhóm các lệnh gộp lại;
  - Được sử dụng thông qua tên và các tham số.
- ❖ Ý nghĩa của việc sử dụng chương trình con:
  - Chia chức năng giúp chương trình trong sáng, dễ hiểu, dễ bảo trì;
  - Chương trình con được viết một lần và có thể sử dụng nhiều lần.

## 3.9.1 Chương trình con – Khai báo và sử dụng

### ❖ Khai báo

<name> PROC

; here goes the code  
; of the procedure ...

RET

<name> ENDP

### ❖ Sử dụng: gọi chương trình con

Call <proc\_name>

## 3.9.1 Chương trình con – Khai báo và sử dụng

```
MOV AL, 1
```

```
MOV BL, 2
```

```
CALL m2
```

```
; other instructions
```

```
MOV CX, 30
```

```
;-----
```

```
; define a proc
```

```
; input: AL, BL
```

```
; Output: AX
```

```
m2 PROC
```

```
    MUL BL           ; AX = AL * BL.
```

```
    RET              ; return to caller.
```

```
m2 ENDP
```

## 3.9.2 Chương trình con – Truyền tham số

- ❖ Phục vụ trao đổi dữ liệu giữa chương trình gọi và chương trình con;
- ❖ Các phương pháp truyền tham số:
  - Truyền tham số thông qua các thanh ghi
    - Đưa giá trị vào các thanh ghi lưu tham số cần truyền trước khi gọi hoặc trở về từ chương trình con
  - Truyền tham số thông qua các biến toàn cục
    - Biến toàn cục (định nghĩa trong đoạn dữ liệu ở chương trình chính) có thể được truy nhập ở cả chương trình chính và chương trình con.
  - Truyền tham số thông qua ngăn xếp
    - Sử dụng kết hợp các lệnh PUSH / POP để truyền tham số.



## 3.9.2 Chương trình con – Truyền tham số

### ❖ Bảo vệ các thanh ghi:

- Cần thiết phải bảo vệ giá trị các thanh ghi sử dụng trong chương trình gọi khi chúng cũng được sử dụng trong chương trình con.
- Giá trị của các thanh ghi có thể bị thay đổi trong chương trình con → sai kết quả ở chương trình gọi.

### ❖ Các phương pháp bảo vệ các thanh ghi:

- Sử dụng PUSH và POP cho các thanh ghi tổng quát, chỉ số và con trỏ;
- Sử dụng PUSHF và POPF cho thanh ghi cờ;
- Sử dụng qui ước thống nhất về sử dụng các thanh ghi.

### 3.9.3 Chương trình con – Ví dụ 1

; Find max of a list and print out the max

.Model small

.Stack 100H

.Data

; source string

list DB 1,4,0,9,7,2,4,6,2,5

.code

main proc

    ; initilize the ds and es registers

    mov ax, @Data

    mov ds,ax

    cld

    mov cx, 9

    lea si, list   ; si points to list

    mov bl, [si]   ; max <-- 1st element

    inc si

## 3.9.3 Chương trình con – Ví dụ 1

Start:

lodsb

cmp al, bl

jle BYPASS

mov bl, al; al>bl --> bl to store new max

BYPASS:

loop Start

; print the max

call printSingleDigit

; end program

mov ah, 4CH

int 21H

main endp

## 3.9.3 Chương trình con – Ví dụ 1

```
;-----  
; proc to print out a single digit number  
; input: bl to contain the digit to print  
printSingleDigit proc  
    push dx  
    push ax  
    add bl, '0' ; digit to char  
    mov dl,bl  
    mov ah, 2  
    int 21H  
    pop ax  
    pop dx  
    ret  
printSingleDigit endp  
end main
```

## 3.9.3 Chương trình con – Ví dụ 2

```
; convert lower case chars to upper cases
.Model small
.Stack 100H
.Data
; source string
str1 DB 'a','5', 'B', '?', 'd', 'g', 'P','N','k','*'
      DB 10,13,'$'
; destination string
str2 DB 10 DUP(' ')
      DB '$'
.code
main proc
    ; initilize the ds and es registers
    mov ax, @Data
    mov ds,ax
    mov es,ax
    ; make SI points to str1 and DI to str2
    lea si, str1
    lea di, str2
    cld
    mov cx, 10
```

## 3.9.3 Chương trình con – Ví dụ 2

Start:

```
lodsb  
; check if it is lower case  
cmp al, 'a'  
jl NotLowerCase  
cmp al, 'z'  
jg NotLowerCase  
; is lower case, convert to upper case  
sub al, 20H
```

```
; store to new string
```

```
NotLowerCase:
```

```
stosb
```

```
loop Start
```

```
; print the original string
```

```
lea dx, str1
```

```
call printString
```

## 3.9.3 Chương trình con – Ví dụ 2

```
; print the output
    lea dx, str2
    call printString

; end program
    mov ah, 4CH
    int 21H
main endp

; -----
; proc to print a string
; input: DX to contain the relative address of the string
printString proc
    push ax    ; store AX into stack
    mov ah, 9
    int 21H
    pop ax    ; restore AX from stack
    ret
printString endp
end main
```

### 3.9.3 Chương trình con – Ví dụ 3

```
; Sort a list to accending order  
; print out the original and sorted lists
```

```
.Model small
```

```
.Stack 100H
```

```
.Data
```

```
LIST_COUNT EQU 10
```

```
list DB 1,4,0,3,7,2,8,6,2,5
```

```
CRLF DB 13,10,'$'
```

```
.code
```

```
main proc
```

```
; initilize the ds and es registers
```

```
mov ax, @Data
```

```
mov ds,ax
```

```
; print the original list
```

```
mov cx, LIST_COUNT
```

```
lea si, list
```

```
call printList
```



### 3.9.3 Chương trình con – Ví dụ 3

```
lea si, list    ; si points to list
mov bl, 1       ; main counter
MainLoop:
    mov al, [si] ; al <-- [si]
    mov di, si
    mov bh, bl   ; sub-counter
    mov dx, di   ; dx to store min position
    SubLoop:
        inc di
        inc bh
        cmp al, [di]
        jle NotMin
        mov al, [di]
        mov dx, di
    NotMin:
        cmp bh, LIST_COUNT
        je ExitSub
        jmp SubLoop
ExitSub:
```

### 3.9.3 Chương trình con – Ví dụ 3

; swap the position if min is different from first place

mov di, dx

cmp si, di

je NoSwap

call swapMemLocation

NoSwap:

inc bl

cmp bl, LIST\_COUNT

je ExitMain

inc si

jmp MainLoop

ExitMain:

## 3.9.3 Chương trình con – Ví dụ 3

```
; print the new line chars  
lea dx, CRLF  
call printString  
  
; print the sorted list  
mov cx, LIST_COUNT  
lea si, list ; si points to list  
call printList  
  
; end program  
mov ah, 4CH  
int 21H  
main endp
```

### 3.9.3 Chương trình con – Ví dụ 3

```
;-----  
; swap the value of 2 memory locations  
; input: si points to the 1st memory location  
;      di points to the 2nd memory location  
swapMemLocation proc  
    push ax  
    mov al, [si]  
    mov ah, [di]  
    mov [si], ah  
    mov [di], al  
    pop ax  
    ret  
swapMemLocation endp
```

## 3.9.3 Chương trình con – Ví dụ 3

```
;-----  
; print the list  
; input: SI to store the start address of the list  
;      CX to store the number of elements  
printList proc  
    push dx  
    StartPrint:  
        mov dl, [si]  
        call printSingleDigit  
        inc si  
        loop StartPrint  
    pop dx  
    ret  
printList endp
```

## 3.9.3 Chương trình con – Ví dụ 3

```
; print a string ending with $  
;input: DX to point to string  
printString proc  
    push ax  
    mov ah, 9  
    int 21H  
    pop ax  
    ret  
printString endp  
;-----  
; proc to print out a single digit number  
; input: dl to contain the digit to print  
printSingleDigit proc  
    push ax  
    add dl, '0' ; digit to char  
    mov ah, 2  
    int 21H  
    pop ax  
    ret  
printSingleDigit endp  
end main
```

## 3.10 Tạo và sử dụng macro

- ❖ Macro là một đoạn mã được đặt tên và có thể được chèn vào bất cứ vị trí nào trong đoạn mã của chương trình
- ❖ Đặc điểm của macro:
  - Macro hỗ trợ danh sách các tham số
  - Macro chỉ tồn tại khi soạn thảo mã. Khi dịch, các macro sẽ được thay thế bằng đoạn mã thực của macro.
  - Nếu một macro không được sử dụng, mã của nó sẽ bị loại khỏi chương trình sau khi dịch.
  - Macro nhanh hơn thủ tục/hàm do mã của macro được chèn trực tiếp vào chương trình và nó không đòi hỏi cơ chế gọi thực hiện (lưu địa chỉ) và trở về (khôi phục địa chỉ trở về) như chương trình con.

## 3.10 Tạo và sử dụng macro

### ❖ Định nghĩa macro:

name MACRO [parameters,...]

<instructions>

ENDM

### ❖ Sử dụng macro:

<macro\_name> [real parameters]



## 3.10 Tạo và sử dụng macro

### ❖ Ví dụ

```
MyMacro  MACRO  p1, p2, p3  
    MOV AX, p1  
    MOV BX, p2  
    MOV CX, p3  
ENDM  
;  
MyMacro 1, 2, 3  
MyMacro 4, 5, DX
```

Được chuyển thành sau dịch:

```
MOV AX, 00001h  
MOV BX, 00002h  
MOV CX, 00003h  
MOV AX, 00004h  
MOV BX, 00005h  
MOV CX, DX
```