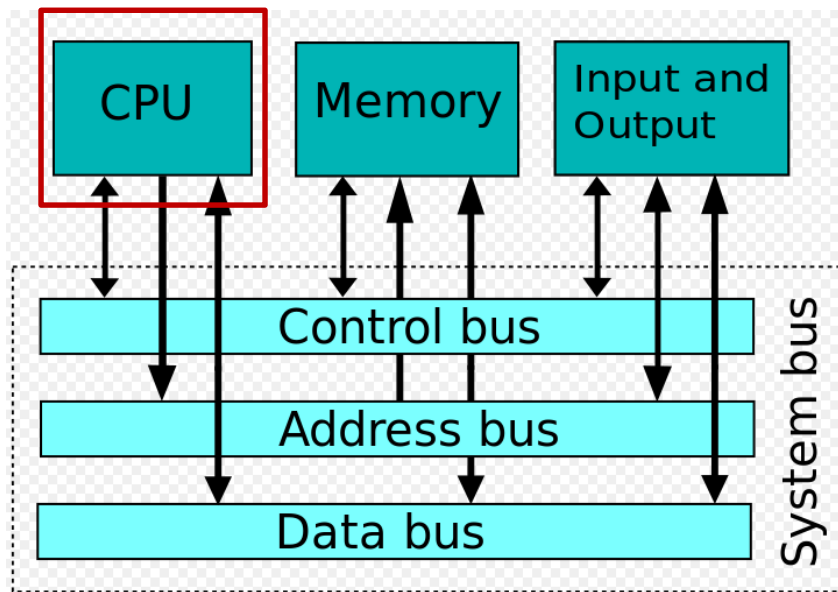


Chương 4

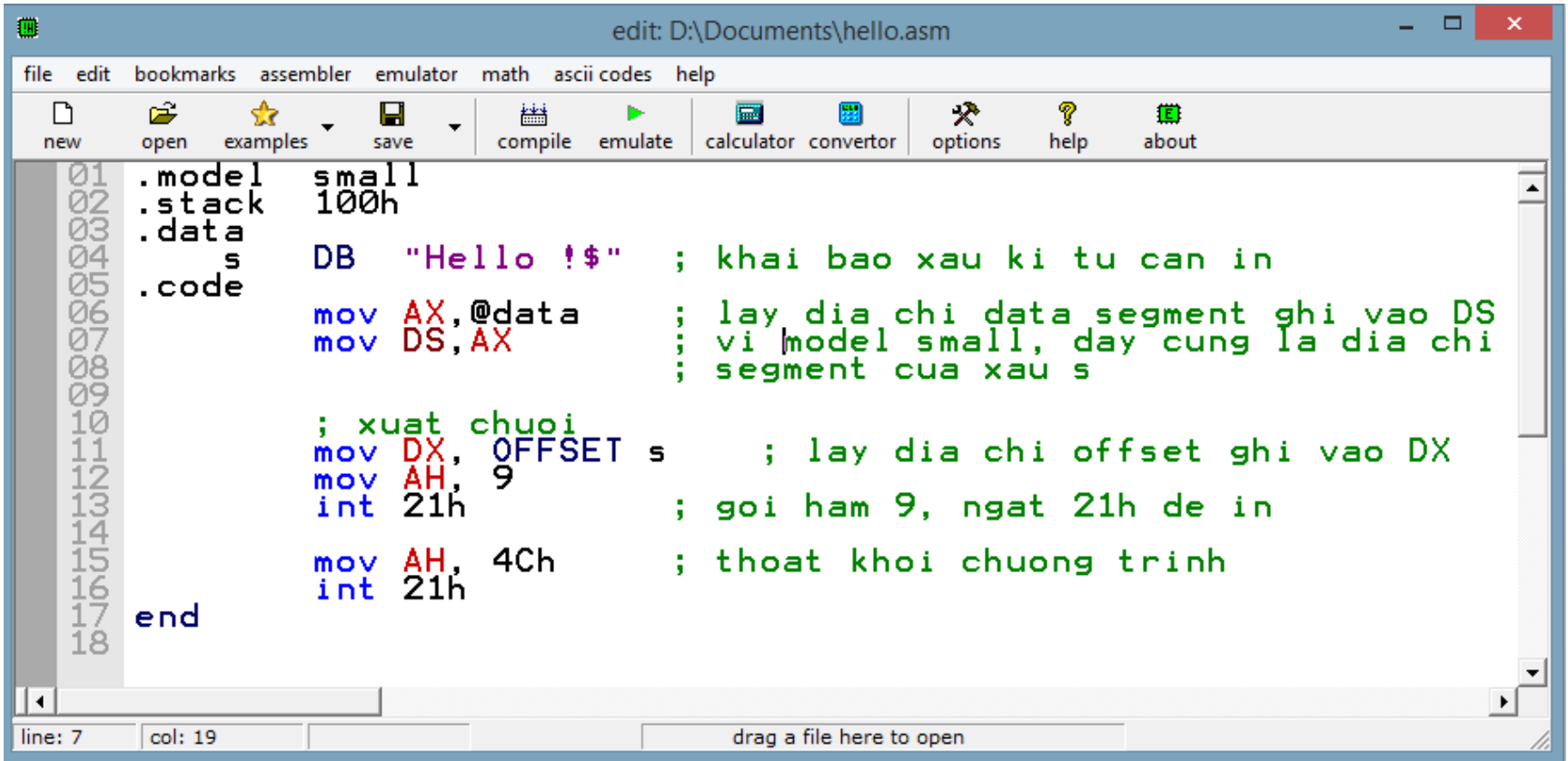
Bộ xử lý trung tâm (CPU)



Giảng viên: ThS. Phan Như Minh

Spring 2020

4.6. Ngôn ngữ ASSEMBLY



```
01 .model small
02 .stack 100h
03 .data
04 s DB "Hello !$" ; khai bao xau ki tu can in
05 .code
06 mov AX,@data ; lay dia chi data segment ghi vao DS
07 mov DS,AX ; vi model small, day cung la dia chi
08 ; segment cua xau s
09
10 ; xuat chuoai
11 mov DX, OFFSET s ; lay dia chi offset ghi vao DX
12 mov AH, 9
13 int 21h ; goi ham 9, ngat 21h de in
14
15 mov AH, 4Ch ; thoat khoi chuong trinh
16 int 21h
17 end
18
```

line: 7 col: 19 drag a file here to open

emulator: hello.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 100

registers

	H	L
AX	4C	24
BX	00	00
CX	01	20
DX	00	00
CS	F400	
IP	0204	
SS	0710	
SP	00FA	
BP	0000	
SI	0000	
DI	0000	
DS	0720	
ES	0700	

F400:0204 F400:0204

Address	Hex	Dec	Comment
F4200:	FF	255	RI
F4201:	FF	255	RI
F4202:	CD	205	=
F4203:	21	033	!
F4204:	CF	207	±
F4205:	00	000	NI
F4206:	00	000	NI
F4207:	00	000	NI
F4208:	00	000	NI
F4209:	00	000	NI
F420A:	00	000	NI
F420B:	00	000	NI
F420C:	00	000	NI
F420D:	00	000	NI
F420E:	00	000	NI
F420F:	00	000	NI

BIOS DI
INT 021h
IRET
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A
ADD [BX + SI], A

original source code

```
01 .model small
02 .stack 100h
03 .data
04 s DB "Hello ! $"
05 .code
06 mov AX, @data ; lay
07 mov DS, AX ; vi
08 ; segment cua xau s
09
10 ; xuat chuoi
11 mov DX, OFFSET s ;
12 mov AH, 9
13 int 21h ; g
14
15 mov AH, 4Ch ; t
16 int 21h
17 end
18
19
20
```

emulator screen (49x17 chars)

Hello !

clear screen change font 0/16

stack

Address	Hex	Dec	Comment
0710:0124			
0710:0122			
0710:0120			
0710:011E			
0710:011C			
0710:011A			
0710:0118			
0710:0116			
0710:0114			
0710:0112			
0710:0110			
0710:010E			
0710:010C			
0710:010A			
0710:0108			
0710:0106			
0710:0104			

flags

CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	0
DF	0

analyse

- Khái niệm về ngôn ngữ ASSEMBLY
- Khuôn dạng chỉ thị ngôn ngữ ASSEMBLY
- So sánh ngôn ngữ Assembly và ngôn ngữ bậc cao
- Một số lệnh cơ bản

1. Khái niệm

- Hợp ngữ (assembly language) là ngôn ngữ cấp thấp dùng để viết các chương trình máy tính.
- Dùng các thuật ngữ thân thiện với người dùng hơn so với ngôn ngữ máy.
- Được dịch sang ngôn ngữ máy bằng một tiện ích gọi là trình hợp dịch (assembler).

- Ví dụ, bộ vi xử lý x86/IA-32 có thể thực hiện được chỉ thị nhị phân sau (thể hiện ở dạng ngôn ngữ máy):

10110000 01100001

- Lệnh trên tương đương với chỉ thị hợp ngữ sau:

MOV AL, 061h

- Chỉ thị này nghĩa là gán giá trị 61h vào thanh ghi AL

2. Khuôn dạng chỉ thị ngôn ngữ ASSEMBLY

- Chương trình bao gồm các dòng lệnh, mỗi lệnh trên một dòng.
- Một lệnh hợp ngữ đầy đủ gồm 4 trường:
[Nhãn lệnh:] <Tên lệnh> <Toán hạng> [;Chú thích]
- Các trường cách nhau ít nhất 1 khoảng trắng và phải theo đúng thứ tự

- [Nhãn lệnh:] là một dãy các kí tự đứng trước câu lệnh, nó được chỉ định thay thế cho địa chỉ của câu lệnh trong các đoạn lệnh lặp, rẽ nhánh ...
- <Tên lệnh> là một trong các lệnh thuộc tập lệnh hợp ngữ (lệnh gọi nhớ) của vi xử lý trên máy tính thực hiện lệnh này
- <Toán hạng> là đối tượng mà lệnh tác động vào
- [;Chú thích] dùng để làm rõ ý nghĩa của câu lệnh nếu cần, và không được dịch sang mã máy

Ví dụ:

LenhVD: MOV AX, BX ;dat gia tri thanh ghi BX vao
thanh ghi AX

ADD CL, Spt

IMUL AX, BX, 20

Quy tắc đặt tên: nhãn lệnh, tên các thủ tục và biến đặt theo quy tắc

- ✓ Dài từ 1-31 kí tự, có thể chứa chữ cái, số và các kí tự đặc biệt và không chứa khoảng trắng.
- ✓ Nếu sử dụng dấu . thì nó phải đứng đầu tiên.
- ✓ Không bắt đầu bằng số.
- ✓ Không phân biệt chữ hoa và chữ thường

Ví dụ: Counter1, @character, Sum_of_digits, .TEST ...

Dữ liệu của chương trình

- Trong một chương trình hợp ngữ chúng ta có thể biểu diễn dữ liệu dưới dạng số nhị phân, thập phân, hex hoặc kí tự
- Các số:
 - ✓ Số nhị phân được viết như một chuỗi các bit và kết thúc bằng chữ B hoặc b. VD 11110001b.
 - ✓ Số thập phân là chuỗi các chữ số thập phân và kết thúc bằng chữ D hoặc d (hoặc không có). VD 682D hoặc 682
 - ✓ Số hex phải bắt đầu bằng một chữ số thập phân và kết thúc bằng H hoặc h. VD 0ABCh

Các kí tự hay chuỗi phải đặt trong dấu “ ” hoặc ‘ ’

VD: “A” ‘hello’

Khai báo biến:

Tên biến DB|DW|DD|DQ|DT Giá trị khởi tạo

DB: định nghĩa 1 byte

DW: định nghĩa 1 word (2 bytes)

DD: định nghĩa 2 word (4 bytes)

DQ: định nghĩa 4 word (8 bytes)

DT: định nghĩa 10 bytes

- Mảng trong lập trình hợp ngữ chỉ là một chuỗi các byte nhớ hay từ nhớ.

VD để định nghĩa mảng có 3 phần tử:

B_ARRAY DB 10h,20h,30h

- Nếu địa chỉ của B_ARRAY trong bộ nhớ là 200h thì bộ nhớ sẽ như sau:

B_ARRAY	200h	10h
B_ARRAY+1	201h	20h
B_ARRAY+2	202h	30h

- Một chuỗi kí tự có thể được khởi tạo bằng bảng mã ASCII

VD: letter DB 'A'

- Khai báo hằng

 Tên hằng EQU Giá trị

VD: LF EQU 0114h

 MSG EQU 'Type your name'

Toán tử ? dùng khi khai báo biến/mảng mà không cần khởi tạo giá trị

Cấu trúc chung của chương trình hợp ngữ:

Các chế độ bộ nhớ

- ✓ Kích thước của đoạn mã và dữ liệu trong chương trình có thể được xác định bằng cách chỉ ra chế độ bộ nhớ nhờ sử dụng hướng dẫn biên dịch .MODEL
- ✓ Cú pháp: **.MODEL** **kiểu bộ nhớ**

Các kiểu bộ nhớ:

Loại	Mô tả
Tiny	Mã lệnh và dữ liệu nằm trong một đoạn
Small	Mã lệnh trong một đoạn, dữ liệu trong một đoạn
Medium	Mã lệnh không nằm trong một đoạn, dữ liệu trong một đoạn
Compact	Mã lệnh trong một đoạn, dữ liệu không nằm trong một đoạn
Large	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và không có mảng nào lớn hơn 64KB
Huge	Mã lệnh không nằm trong một đoạn, dữ liệu không nằm trong một đoạn và các mảng có thể lớn hơn 64KB

Đoạn dữ liệu chứa tất cả các định nghĩa biến, hằng
Khai báo:

Cú pháp: **.DATA**

WORD1	DW	2
WORD2	DW	10h
MSG	DB	'HELLO!'
MASK	EQU	1000111

➤ Đoạn ngăn xếp được khai báo để tạo ra một khối bộ nhớ để chứa ngăn xếp.

➤ Cú pháp:

.STACK kích thước

Trong đó kích thước là một số tùy ý, xác định kích thước của vùng ngăn xếp tính theo byte

VD: .STACK 100h

Đoạn mã chứa các lệnh của chương trình.

Khai báo:

.CODE

Bên trong đoạn mã, các lệnh được tổ chức như các thủ tục, định nghĩa thủ tục như sau:

Tên thủ tục PROC

;Thân của thủ tục

Tên thủ tục ENDP

3. So sánh ngôn ngữ Assembly và ngôn ngữ bậc

- Ngôn ngữ Assembly có ưu điểm là tốc độ cao và dung lượng file chạy sau khi dịch rất nhỏ
- Tuy nhiên cú pháp không thân thiện, số lượng tập lệnh nhiều gây khó khăn trong quá trình sử dụng
- Do tập lệnh của mỗi hệ vi xử lý khác nhau là khác nhau nên các chương trình viết bằng Assembly không thể chạy trên nhiều hệ thống khác nhau

Chuyển ngôn ngữ cấp cao thành ngôn ngữ ASM

Giả sử A và B là 2 biến từ .

Chúng ta sẽ chuyển các mệnh đề sau trong ngôn ngữ cấp cao ra ngôn ngữ ASM .

Mệnh đề B=A

MOV	AX,A	; đưa A vào AX
MOV	B,AX	; đưa AX vào B

Mệnh đề A=5-A

MOV	AX,5	; đưa 5 vào AX
SUB	AX,A	; AX=5-A
MOV	A,AX	; A=5-A

cách khác :

NEG	A	;A=-A
ADD	A,5	;A=5-A

Mệnh đề A=B-2*A

MOV	AX,B	;Ax=B
SUB	AX,A	;AX=B-A
SUB	AX,A	;AX=B-2*A
MOV	A,AX	;A=B-2*A

4. Một số lệnh cơ bản

Lệnh MOV dùng để chuyển dữ liệu giữa các thanh ghi, giữa một thanh ghi và một ô nhớ hoặc chuyển trực tiếp một giá trị vào thanh ghi hay ô nhớ

MOV toán hạng đích, toán hạng nguồn

Tác dụng: lấy nội dung của toán hạng nguồn đặt vào toán hạng đích, nội dung toán hạng nguồn không thay đổi

VD: MOV AX, BX
 MOV AX, word1
 MOV AX, 0114h

- Các lệnh ADD, SUB, INC, DEC
- Lệnh ADD và SUB sử dụng để cộng hoặc trừ nội dung của 2 thanh ghi, 1 thanh ghi và 1 ô nhớ hoặc một số vào thanh ghi/ô nhớ

ADD toán hạng đích, toán hạng nguồn

SUB toán hạng đích, toán hạng nguồn

VD: ADD BL, 5
 SUB AX, BL

Cấu trúc rẽ nhánh

IF condition is true **THEN**

execute true branch statements

END IF

Hoặc

IF condition is true **THEN**

execute true branch statements

ELSE

execute false branch statements

END_IF

Ví dụ 1: Thay thế giá trị trên AX bằng giá trị tuyệt đối của nó

Thuật toán:

IF AX<0

THEN

replace AX by - AX

END-IF

Mã hoá:

;if AX<0

CMP AX,0

JNL END_IF

; no , exit

;then

NEG AX

; yes , change sign

END_IF :

Ví dụ 2: giả sử AL và BL chứa ASCII code của 1 ký tự .Hãy xuất ra màn hình ký tự trước (theo thứ tự ký tự)

Thuật toán

```
IF AL<= BL THEN
    display AL
ELSE
    display character in BL
END_IF
```

Mã hoá :

```
MOV     AH,2           ; chuẩn bị xuất ký tự
;if     AL<=BL
CMP     AL,BL          ; AL<=BL?
JNBE    ELSE_          ; no, display character in BL
;then
MOV     DL,AL
JMP     DISPLAY
ELSE_:
MOV     DL,BL
DISPLAY:
INT     21H
END_IF :
```

Rẽ nhánh nhiều hướng

Case là một cấu trúc rẽ nhánh nhiều hướng. Có thể dùng để test một thanh ghi hay, biến nào đó hay một biểu thức mà giá trị cụ thể nằm trong 1 vùng các giá trị.

Cấu trúc của CASE như sau :

CASE expression

value_1 : Statements_1

value_2 : Statements_2

.

.

value_n : Statements_n

Ví dụ

Nếu AX âm thì đặt -1 vào BX

Nếu AX bằng 0 thì đặt 0 vào BX

Nếu AX dương thì đặt 1 vào BX

Thuật toán :

CASE AX

< 0 put -1 in BX

= 0 put 0 in BX

> 0 put 1 in BX

Cài đặt

```
; case AX
                                CMP     AX,0           ; test AX
                                JL      NEGATIVE        ; AX<0
                                JE      ZERO           ; AX=0
                                JG      POSITIVE        ; AX>0
NEGATIVE:
                                MOV     BX,-1
                                JMP     END_CASE
ZERO:
                                MOV     BX,0
                                JMP     END_CASE
POSITIVE:
                                MOV     BX,1
                                JMP     END_CASE
END_CASE :
```

Rẽ nhánh với một tổ hợp các điều kiện

Đôi khi tình trạng rẽ nhánh trong các lệnh IF , CASE cần một tổ hợp các điều kiện dưới dạng :

Condition_1 AND Condition_2

Condition_1 OR Condition_2

Ví dụ 1: Đọc một ký tự và nếu nó là ký tự hoa thì in nó ra màn hình

Thuật toán :

Read a character (into AL)

IF ('A' <= character) AND (character <= 'Z') THEN

display character

END_IF

Cài đặt

; read a character

MOV AH,2

INT 21H ; character in AL

; IF ('A' <= character) AND (character <= 'Z')

CMP AL,'A' ; char >='A'?

JNGE END_IF ; no, exit

CMP AL,'Z' ; char <='Z'?

JNLE END_IF ; no exit

; then display it

MOV DL,AL

MOV AH,2

INT 21H

END_IF :

Ví dụ 2: Đọc một ký tự , nếu ký tự đó là ‘Y’ hoặc ‘y’ thì in nó lên màn hình , ngược lại thì kết thúc chương trình .

Thuật toán

Read a character (into AL)

IF (character =‘Y’) OR (character=‘y’) THEN

display it

ELSE

terminate the program

END_IF

Cài đặt

; read a character

```
MOV    AH,2  
INT     21H
```

; character in AL

; IF (character = 'y') OR (character = 'Y')

```
CMP     AL,'y'  
JE      THEN  
CMP     AL,'Y'  
JE      THEN  
JMP     ELSE_
```

; char = 'y'?

; yes , goto display it

; char = 'Y'?

; yes , goto display it

; no , terminate

THEN :

```
MOV     DL,AL  
MOV     AH,2  
INT     21H  
JMP     END_IF
```

ELSE_:

```
MOV     AH,4CH  
INT     21h
```

END_IF :

Cấu trúc lặp

Một vòng lặp gồm nhiều lệnh được lặp lại , số lần lặp phụ thuộc điều kiện

Vòng FOR

Lệnh LOOP có thể dùng để thực hiện vòng FOR .

LOOP destination_label

Số đếm cho vòng lặp là thanh ghi CX mà ban đầu nó được gán 1 giá trị nào đó . Khi lệnh LOOP được thực hiện CX sẽ tự động giảm đi 1 . Nếu CX chưa bằng 0 thì vòng lặp được thực hiện tiếp tục . Nếu CX=0 lệnh sau lệnh LOOP được thực hiện

Lưu ý rằng vòng FOR cũng như lệnh LOOP thực hiện ít nhất là 1 lần. Do đó nếu ban đầu CX=0 thì vòng lặp sẽ làm cho CX=FFFFH, tức là thực hiện lặp đến 65535 lần

```
        MOV    CX,80          ; CX chứa số lần lặp
        MOV    AH,2          ; hàm xuất ký tự
        MOV    DL,'*'        ; DL chứa ký tự '*'
TOP:
        INT     21h          ; in dấu '*'
        LOOP TOP             ; lặp 80 lần
```

Vòng WHILE

Vòng WHILE phụ thuộc vào 1 điều kiện .Nếu điều kiện đúng thì thực hiện vòng WHILE . Vì vậy nếu điều kiện sai thì vòng WHILE không thực hiện gì cả

Ví dụ : Viết đoạn mã để đếm số ký tự được nhập vào trên cùng một hàng .

MOV	DX,0	; DX để đếm số
MOV	AH,1	; hàm đọc 1 ký tự
INT	21h	; đọc ký tự vào
AL		
WHILE_:		
CMP	AL,0DH	; có phải là ký tự
CR?		
JE	END_WHILE	; đúng , thoát
INC	DX	; tăng DX lên 1
INT	21h	; đọc ký tự
JMP	WHILE_	; lặp
END_WHILE :		

Các lệnh vào ra

- Lệnh INT (**I**nterrupt) được dùng để gọi các chương trình ngắt của BIOS
 INT số hiệu ngắt
- Ngắt 21h được dùng để gọi rất nhiều hàm của DOS
- Mỗi hàm được gọi bằng cách đặt số hàm vào thanh ghi AH và gọi INT 21h

Các lệnh vào ra (tiếp)

Các hàm của ngắt 21h thường sử dụng:

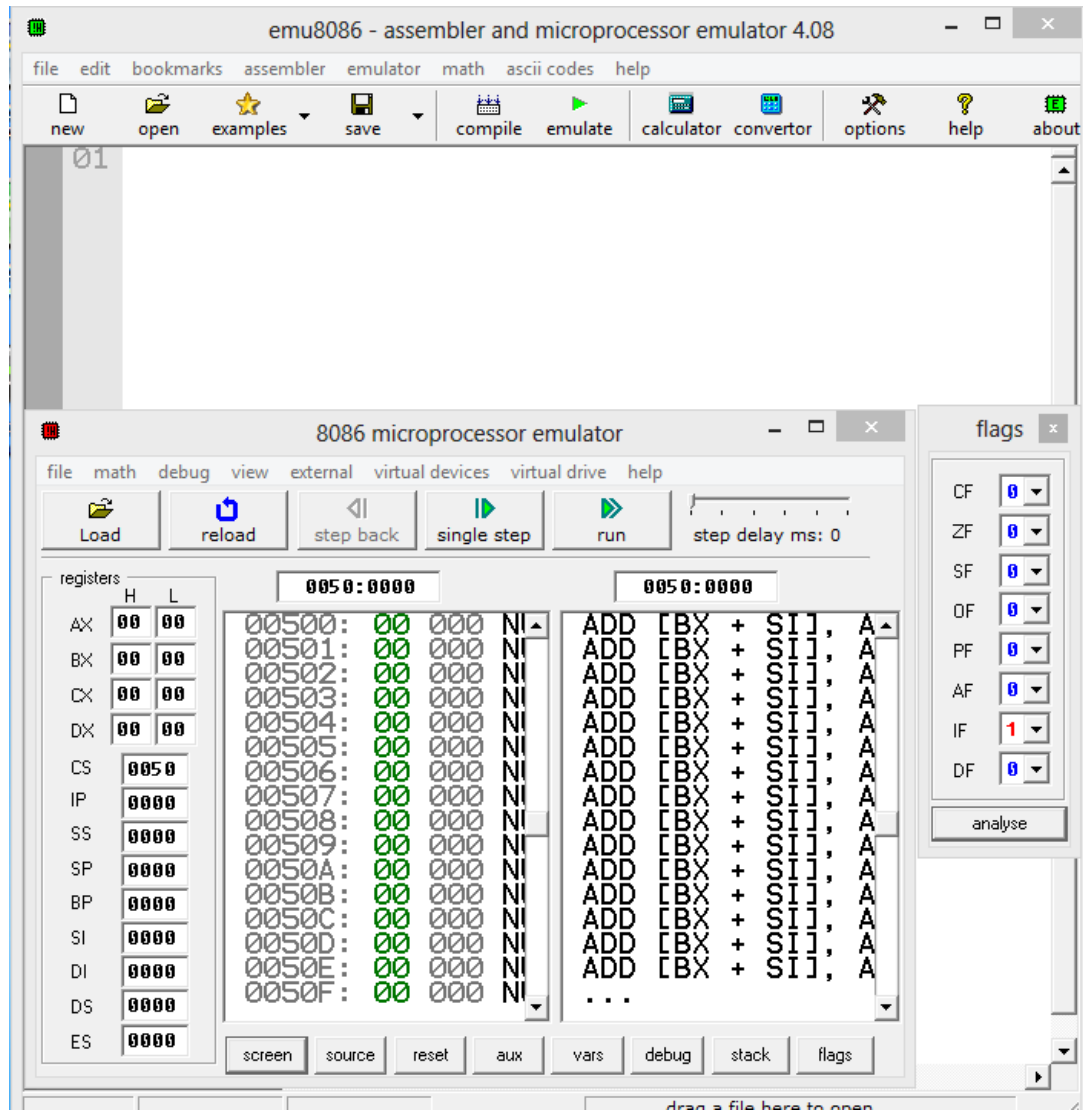
Ngắt 21h:	
AH	Ý nghĩa
1	Đọc 1 ký tự từ bàn phím, KQ lưu trong AL, nếu chưa bấm, chờ bằng được
2	In 1 ký tự ra màn hình, DL=Ký tự cần in, sau khi in: AL=DL
9	In một xâu ký tự, được trả bởi DX, xâu ký tự phải kết thúc bằng '\$'

Lập trình Assembly cho 80x86

- Công cụ: emu8086, radASM
- Chương trình emu8086 là chương trình lập trình mô phỏng cho 8086 (tương thích Intel và AMD) bao gồm bộ dịch ASM và giáo trình (tiếng anh) cho người mới bắt đầu. Chương trình có thể chạy hết hoặc chạy từng bước, ta có thể nhìn thấy các thanh ghi, bộ nhớ, stack, biến,...

Minh họa BXL 8086 trên phần mềm mô phỏng
Emu0886

Emu8086



Khung của chương trình hợp ngữ

```

TITLE      Chương trình hợp ngữ
.MODEL     Kiểu kích thước bộ nhớ      ; Khai báo quy mô sử
                                              ; dụng bộ nhớ
.STACK     Kích thước                    ; Khai báo dung lượng
                                              ; đoạn stack
.DATA
msg DB 'Hello$'                          ; Khai báo đoạn dữ liệu
.CODE
main PROC
...
CALL      Subname                        ; Gọi chương trình con
...
main ENDP

Subname   PROC                          ; Định nghĩa chương
                                              ; trình con
...
RET
Subname   ENDP
END main
  
```

Vi dụ

```

title vidu ; ten chuong trinh
.model ; kich thuoc chuong
small ; trinh
.stack ; kich thuoc stack 256
100h ; vung khai bao du
.data lieu

msg db 'Welcome to Assembly!$'
.code main proc
; khoi tao doan du lieu mov ax, @data
mov ds,ax

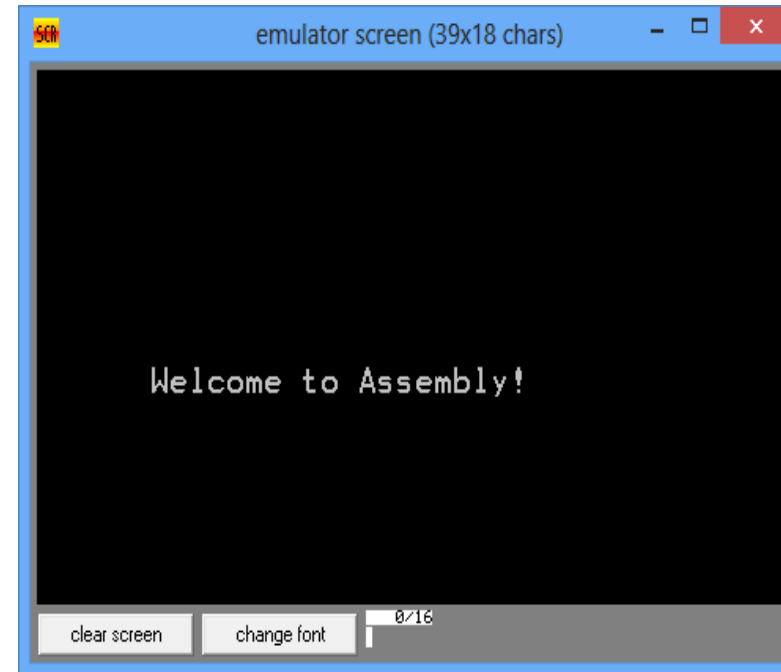
; xoa man hinh mov ah,0
int 10h

; di chuyen giua man minh mov ah,2
mov dx,0a06h ; DH=dong, DL=cot int 10h

;hien thi thong bao mov ah,9
lea dx,msg int 21h

; thoat chuong trinh mov ah, 04ch
int 21h main endp
end main

```



Nhắc lại phương pháp định địa chỉ

- ☐ **Tức thời (trực hăng)**
 - Toán hạng trong lệnh
- ☐ **Thanh ghi**
 - Toán hạng trong thanh ghi
- ☐ **Trực tiếp**
 - Địa chỉ trong lệnh là địa chỉ ô nhớ của toán hạng
- ☐ **Gián tiếp qua thanh ghi**
 - Thanh ghi chứa địa chỉ ô nhớ của toán hạng
- ☐ **Gián tiếp ô nhớ**
 - Địa chỉ trong lệnh là địa chỉ ô nhớ của toán hạng
- ☐ **Chỉ số (dịch chuyển)**
 - Địa chỉ toán hạng là tổng nội dung thanh ghi và độ dời
- ☐ **Tương đối**
 - Tổng nội dung PC và độ dời
- ☐ **Stack**
 - Thanh ghi SP chứa địa chỉ ô nhớ của toán hạng

❖ 14 thanh ghi 16 bit, 5 nhóm

☐ Thanh ghi đoạn

CS (code segment), DS (data segment),
SS (stack segment), ES (extra segment)

☐ Thanh ghi con trỏ

IP (instruction pointer), SP (stack pointer),
BP (base pointer)

☐ Thanh ghi chỉ số

SI (source index), DI (Destination index)

☐ Thanh ghi đa dụng

☐ Thanh ghi cờ

Thanh ghi đa dụng

- ❑ AX Accumulator register
Sử dụng cho tính toán và xuất nhập
- ❑ BX Base register
Thanh ghi duy nhất có thể sử dụng chỉ số
- ❑ CX Counter register
Sử dụng cho vòng lặp
- ❑ DX Data register
Sử dụng cho xuất nhập và các lệnh nhân chia
- ❑ Các thanh ghi đa dụng có thể “chia nhỏ” thành 2 thanh ghi 8-bit (cao và thấp)
AH,AL,BH,BL,CH,CL,DH,DL

Thanh ghi cờ (Flag)

Flags Register			
Tắt	Tên	bit n	“Mô tả”
OF	Overflow	11	Tràn số có dấu
DF	Direction	10	Hướng xử lý chuỗi
IF	Interrupt	9	Cho phép ngắt
TF	Trap	8	CPU thực hiện từng bước
SF	Sign	7	Kiểm tra kết quả là số âm
ZF	Zero	6	Kiểm tra kết quả bằng 0
AF	Auxiliary Carry	4	
PF	Parity	2	Kiểm tra số bit 1 chẵn
CF	Carry	0	Tràn số không dấu

Ảnh hưởng các lệnh đến cờ

Tại 1 thời điểm CPU chỉ thực hiện 1 lệnh, kết quả cờ phản ánh tình trạng CPU sau khi thực hiện lệnh

INSTRUCTION

MOV/XCHG

ADD/SUB

INC/DEC

NEG

AFFECTS FLAGS

Không ảnh hưởng cờ

Tất cả

Tất cả trừ CF

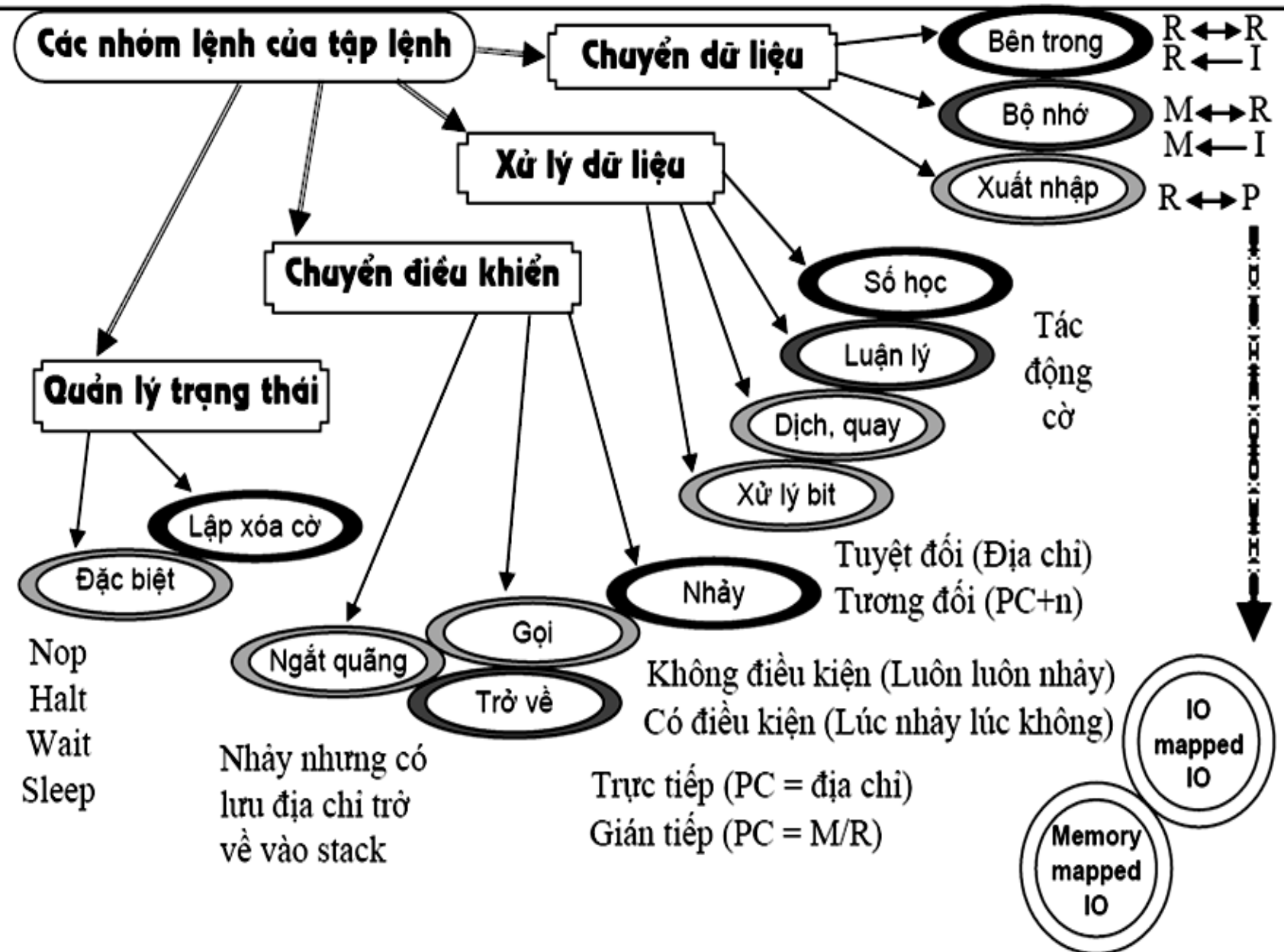
Tất cả

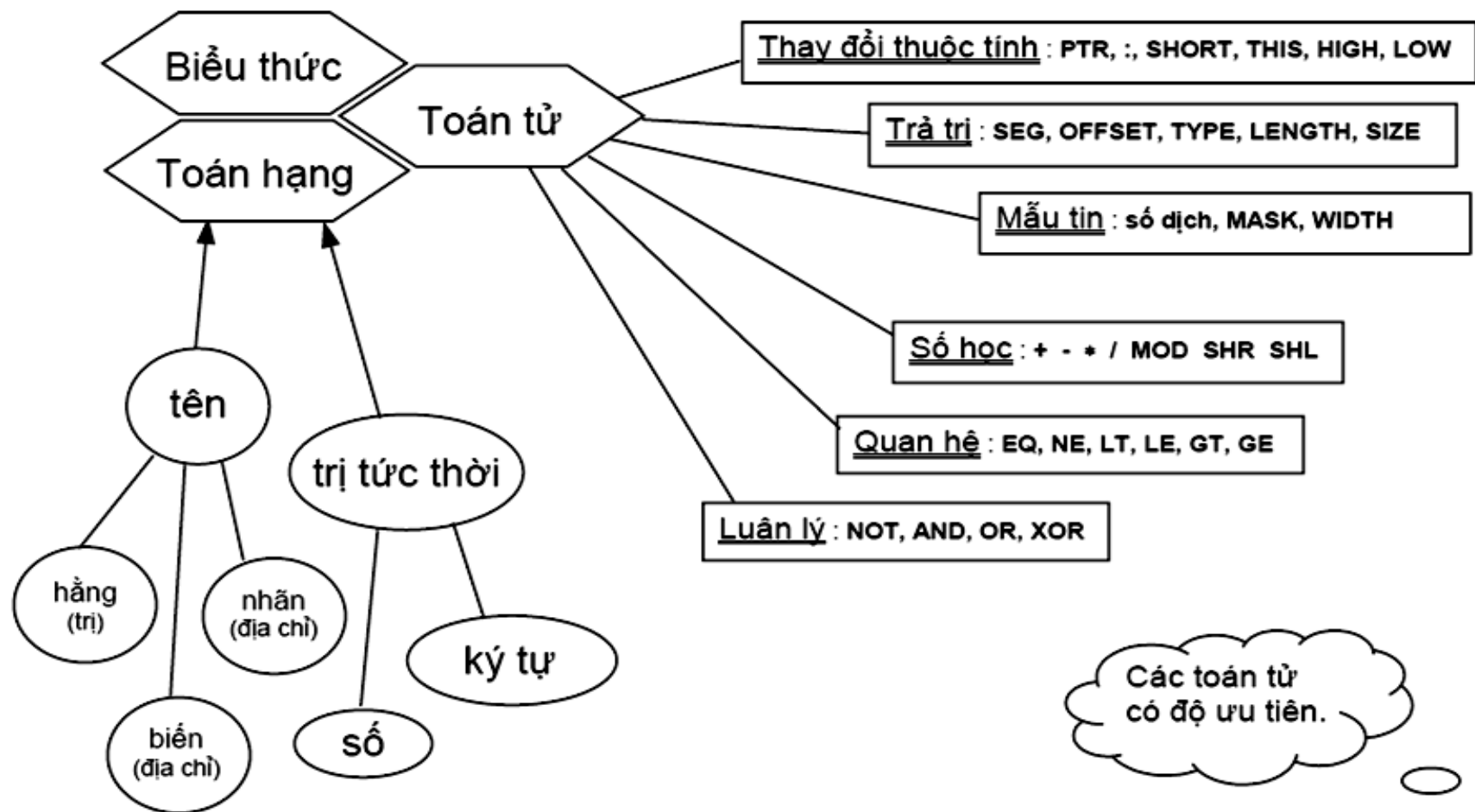
Ví dụ:

ADD AX, BX (trong đó giả sử AX=BX=0FFFFh)

Thay đổi các cờ???

SUB AL,BL (trong đó AL=BL=80h)





Các toán tử có độ ưu tiên.

Lệnh nhập xuất chuỗi kí tự

❑ INT 21h

function number routine

09

Xuất chuỗi kí tự

Input : AH=09

DX= địa chỉ chuỗi ký tự **kết thúc bằng \$**

Output: chuỗi hiện ra màn hình

❑ Lệnh LEA (Load Effective Address)

Lấy địa chỉ offset của biến vào thanh ghi

LEA destination, source

LEA DX,MSG ; đưa địa chỉ MSG vào DX

Program Segment Prefix (PSP)

Chứa thông tin chương trình để hệ thống truy xuất
256 byte

Làm thay đổi DS, ES

MOV AX,@DATA

MOV DS,AX

@DATA tên đoạn số liệu .DATA .

Assembler sẽ chuyển @DATA thành địa chỉ.

Chương trình nhập ký tự thường đổi thành ký tự hoa

TITLE PGM3: CASE COVERT PROGRAM

.MODEL SMALL

.STACK 100H

.DATA

CR EQU 0DH

LF EQU 0AH

MSG1 DB 'enter a lower case letter:\$'

MSG2 DB 0DH,0AH,'in upper case it is :'

CHAR DB ?, '\$' ;

.CODE
MAIN PROC

; initialize ds
MOV AX,@DATA
MOV DS,AX

;print prompt user
LEA DX,MSG1 ; MOV AH,9
INT 21H ;

;input char to AL
MOV AH,1 ;
INT 21H ;

;sub to convert

SUB AL,20H ;

MOV CHAR, AL ;cất vào biến CHAR

;prompt user

LEA DX, MSG2 ;

MOV AH,9

INT 21H

;vì MSG không có dấu \$ nên tiếp tục cho đến dấu
;\$, lúc này có cả kí tự hoa

;dos exit

MOV AH,4CH

INT 21H

MAIN ENDP

END MAIN

Lệnh sử dụng stack

☐ PUSH

Cú pháp: PUSH source

source: thanh ghi, bộ nhớ hay hằng 16-bit để lưu vào stack

☐ POP

Cú pháp: POP dest

dest: thanh ghi, bộ nhớ để lưu kết quả

☐ Cặp thanh ghi xác định đỉnh stack

SS:SP

Lệnh sử dụng stack (tiếp theo)

☐ Qui tắc vào sau ra trước:

PUSH AX	1	POP CX	3
PUSH BX	2	POP BX	2
PUSH CX	3	POP AX	1

☐ PUSHA

Lưu theo thứ tự giá trị của tất cả các thanh ghi AX, BX, CX, DX, SP, BP, SI, DI vào stack

☐ POPA

Lấy ra giá trị trong stack vào các thanh ghi AX, BX, CX, DX, SP, BP, SI, DI theo thứ tự ngược lại

Lệnh XLAT

❑ Translate

“Dịch” giá trị trong AL thành giá trị mới trong bảng tại vị trí xác định theo độ dời bằng AL. Ứng dụng để chuyển đổi số liệu

❑ Cú pháp: XLAT (không có toán hạng)

‘Input’:

BX chứa địa chỉ bảng dữ liệu

AL chứa byte cần đổi

‘Output’:

AL chứa giá trị tìm thấy trong bảng tại địa chỉ BX+AL

Lệnh XLAT (tiếp theo)

Ví dụ: đổi số thập phân <16 ra kí tự HEXA

- **Khai báo bảng kí tự**

Bangkytu DB '0123456789ABCDEF'

- **Thực hiện**

MOV AL, 10 ;Nhập số cần đổi vào AL ví dụ =10

;thực hiện đổi

MOV BX, offset Bangkytu ;lấy địa chỉ bảng dữ liệu

XLAT ;AL chứa kí tự 'A'

Làm thế nào để nhập vào một số 1 chữ số trong khi hàm 01 của INT 21h chỉ nhập kí tự?

Các lệnh điều khiển

- ❑ Lệnh nhảy
 - Có điều kiện
 - Không điều kiện
- ❑ Biểu diễn ngôn ngữ cấp cao
 - Cấu trúc rẽ nhánh
 - IF
 - CASE
 - Compound condition
 - Cấu trúc lặp
 - FOR
 - WHILE
 - REPEAT

Ví dụ về lệnh nhảy

TITLE PGR3-1: CHARACTER DISPLAY

.MODEL SMALL

.STACK 100H

.CODE

MAIN PROC

MOV AH,2 ; hàm xuất kí tự

MOV CX,256 ; số kí tự cần xuất 0-255

MOV DL,0 ; DL mã ASCII của kí tự NUL

PRINT_LOOP : ; nhãn thực hiện vòng lặp

INT 21H ; thực hiện xuất kí tự

INC DL ; tăng DL lên kí tự tiếp theo

DEC CX ; đếm giảm số kí tự chưa in

JNZ PRINT_LOOP ;nhảy đến PRINT_LOOP nếu CX<>0

;DOS EXIT

MOV AH,4CH

INT 21H

MAIN ENDP

END MAIN

Nhảy có điều kiện

- ❑ Cú pháp: Jxxx destination_label
xxx là viết tắt của điều kiện
destination_label là nhãn để nhảy đến
- ❑ Nếu điều kiện được thỏa mãn thì nhảy
Nếu không thì tiếp tục thực hiện lệnh tiếp theo
JNZ == jump if not zero
- ❑ Điều kiện
Thanh ghi cờ
- ❑ Phạm vi nhảy
Không quá 126 bytes

Các lệnh nhảy có dấu

SYMBOL	DESCRIPTION	CONDITION
JG/JNLE	jump if greater than jump if not less than or equal to	ZF=0 and SF=OF
JGE/JNL	jump if greater than or equal to jump if not less or equal to	SF=OF
JL/JNGE	jump if less than jump if not greater or equal	SF<>OF
JLE/JNG	jump if less than or equal jump if not greater	ZF=1 or SF<>OF

Các lệnh nhảy không dấu

SYMBOL	DESCRIPTION	CONDITION
JA/JNBE	jump if above jump if not below or equal	CF=0 and ZF=0
JAЕ/JNB	jump if above or equal jump if not below or equal	CF=0
JB/JNAE	jump if below jump if not above or equal	CF=1
JBE/JNA	jump if below or equal jump if not above	CF=1 or ZF=1

Các lệnh nhảy 1 cờ

SYMBOL	DESCRIPTION	CONDITION
JE/JZ	jump if equal jump if equal to zero	ZF=1
JNE/JNZ	jump if not equal jump if not zero	ZF=0
JC	jump if carry	CF=1
JNC	jump if no carry	CF=0
JO	jump if overflow	OF=1
JNO	jump if not overflow	OF=0
JS	jump if sign negative	SF=1
JNS	jump if non-negative sign	SF=0
JP/JPE	jump if parity even	PF=1
JNP/JPO	jump if parity odd	PF=0

Lệnh CMP

- ❑ Thường dùng so sánh để lấy điều kiện
 - ✓ Cú pháp: CMP destination, source
 - ✓ Thực hiện phép trừ destination – source để tính cờ và từ đó xác định điều kiện cho lệnh nhảy
- ❑ Việc sử dụng lệnh nhảy có dấu hay không dấu là tùy diễn dịch của lập trình viên

Ví dụ: AX=7FFFh, BX=8000h

CMP AX,BX

CMP AX,BX

JA lon_hon ;không nhảy JG lon_hon ;nhảy

Viết đoạn chương trình lấy số lớn nhất trong AX và BX vào CX; biết AX, BX chứa số có dấu?

Lệnh JMP

☐ Nhảy không điều kiện

Cú pháp: JMP destination

☐ Nhảy trong đoạn CS

Khắc phục phạm vi nhảy có điều kiện

TOP:

; thân vòng lặp

DEC CX

JNZ TOP

MOV AX,BX

TOP:

; thân vòng lặp dài hơn 126 bytes

DEC CX

JNZ BOTTOM

JMP EXIT

BOTTOM:

JMP TOP

EXIT:

MOV AX,BX

Biểu diễn ngôn ngữ cấp cao

❑ Cấu trúc IF – THEN – END IF

IF (condition is true)

THEN

execute true branch statements

END IF

❑ Ví dụ:

; if AX<0

CMP AX,0

JNL END_IF ; no , exit

;then

NEG AX ; yes , change sign

END_IF :

Biểu diễn ngôn ngữ cấp cao (tt)

❑ Cấu trúc IF – THEN – ELSE – END IF

IF condition is true

THEN

exec true branch statements

ELSE

exec false branch statements

END_IF

```
IF AL<= BL
THEN
    display AL
ELSE
    display character in BL
END_IF
```



```
MOV AH,2 ; prepare
;if AL<=BL
CMP AL,BL ;AL<=BL?
JNBE ELSE_ ; no, display BL
;then
MOV DL,AL
JMP DISPLAY
ELSE_:
MOV DL,BL
DISPLAY:
INT 21H
END_IF :
```


Biểu diễn ngôn ngữ cấp cao (tt)

❑ Cấu trúc CASE

CASE expression

value_1 : Statements_1

value_2 : Statements_2

...

value_n : Statements_n

CASE AX

< 0 put -1 in BX

= 0 put 0 in BX

> 0 put 1 in BX

; case AX

CMP AX,0 ;test AX

JL NEGATIVE ;AX<0

JE ZERO ;AX=0

JG positive ;AX>0

NEGATIVE:

MOV BX,-1

JMP END_CASE

ZERO:

MOV BX,0

JMP END_CASE

POSITIVE:

MOV BX,1

JMP END_CASE

END_CASE :

Biểu diễn ngôn ngữ cấp cao (tt)

- ❑ Rẽ nhánh với tổ hợp điều kiện

Condition_1 **AND** Condition_2
Condition_1 **OR** Condition_2

Read a character (into AL)

*IF ('A' <= character) AND (character <= 'Z') THEN
 display character
END_IF*

```
;read a character
    MOV AH,2
    INT 21H ; character in AL
; IF ( 'A' <= character ) AND ( character <= 'Z' )
    CMP AL,'A'      ; char >='A'?
    JNGE END_IF     ; no, exit
    CMP AL,'Z'      ; char <='Z'?
    JNLE END_IF     ; no exit
; then display it
    MOV DL,AL
    MOV AH,2
    INT 21H
END_IF:
```

Biểu diễn ngôn ngữ cấp cao (tt)

■ Rẽ nhánh với tổ hợp điều kiện

Condition_1 **AND**

Condition_2

Condition_1 **OR**

Condition_2

*Read a character (into AL)
IF (character ='Y') OR (
character='y') THEN
 display it
ELSE
 terminate the program
END_IF*



```
;read a character
    MOV AH,2
    INT 21H ; character in AL
; IF ( character ='y' ) OR ( charater = 'Y')
    CMP AL,'y'      ; char ='y'?
    JE THEN         ; yes , display it
    CMP AL,'Y'      ; char ='Y'?
    JE THEN         ; yes , display it
    JMP ELSE_       ; no , terminate

THEN :
    MOV DL,AL
    MOV AH,2
    INT 21H
    JMP END_IF

ELSE_:
    MOV AH,4CH
    INT 21h

END_IF :
```

Biểu diễn ngôn ngữ cấp cao (tt)

❑ Cấu trúc lặp FOR

LOOP destination_label

Lặp khi CX khác 0

Số đếm trong thanh ghi CX giảm đi 1

❑ Mẫu thực hiện lệnh LOOP

```
TOP:      MOV CX,20      ; gán cho cho CX số lần lặp là 20
          ...           ; thân vòng lặp ở đây
          LOOP TOP      ; các lệnh tiếp theo sau vòng lặp
          ...
```

Cấu trúc lặp FOR (tt)

❑ LOOP luôn thực hiện ít nhất 1 lần

CX = 0 → lặp FFFFh lần

Tránh bằng lệnh JCXZ (Jump if CX is Zero)

```
        MOV CX,...           ; gán cho cho CX số lần lặp
        JCXZ SKIP
TOP:
        ...                  ; thân vòng lặp ở đây
        LOOP TOP
SKIP:
        ...                  ; các lệnh tiếp theo sau vòng lặp
```


Biểu diễn ngôn ngữ cấp cao (tt)

❑ Cấu trúc lặp WHILE

Kiểm tra và lặp chỉ khi điều kiện còn đúng

■ Ví dụ:

*Đếm số kí tự
nhập trên cùng
1 hàng (cho
đến khi gặp kí
tự CR)*



```
MOV DX,0      ; DX để đếm skt
MOV AH,1      ; hàm nhập kí tự
INT 21h       ; kí tự ở AL

WHILE_:
  CMP AL,0DH   ; nếu kt=CR
  JE END_WHILE ; đúng, thoát
  INC DX       ; DX tăng 1
  INT 21h      ; đọc kt tiếp
  JMP WHILE_   ; lặp

END_WHILE :
```

Biểu diễn ngôn ngữ cấp cao (tt)

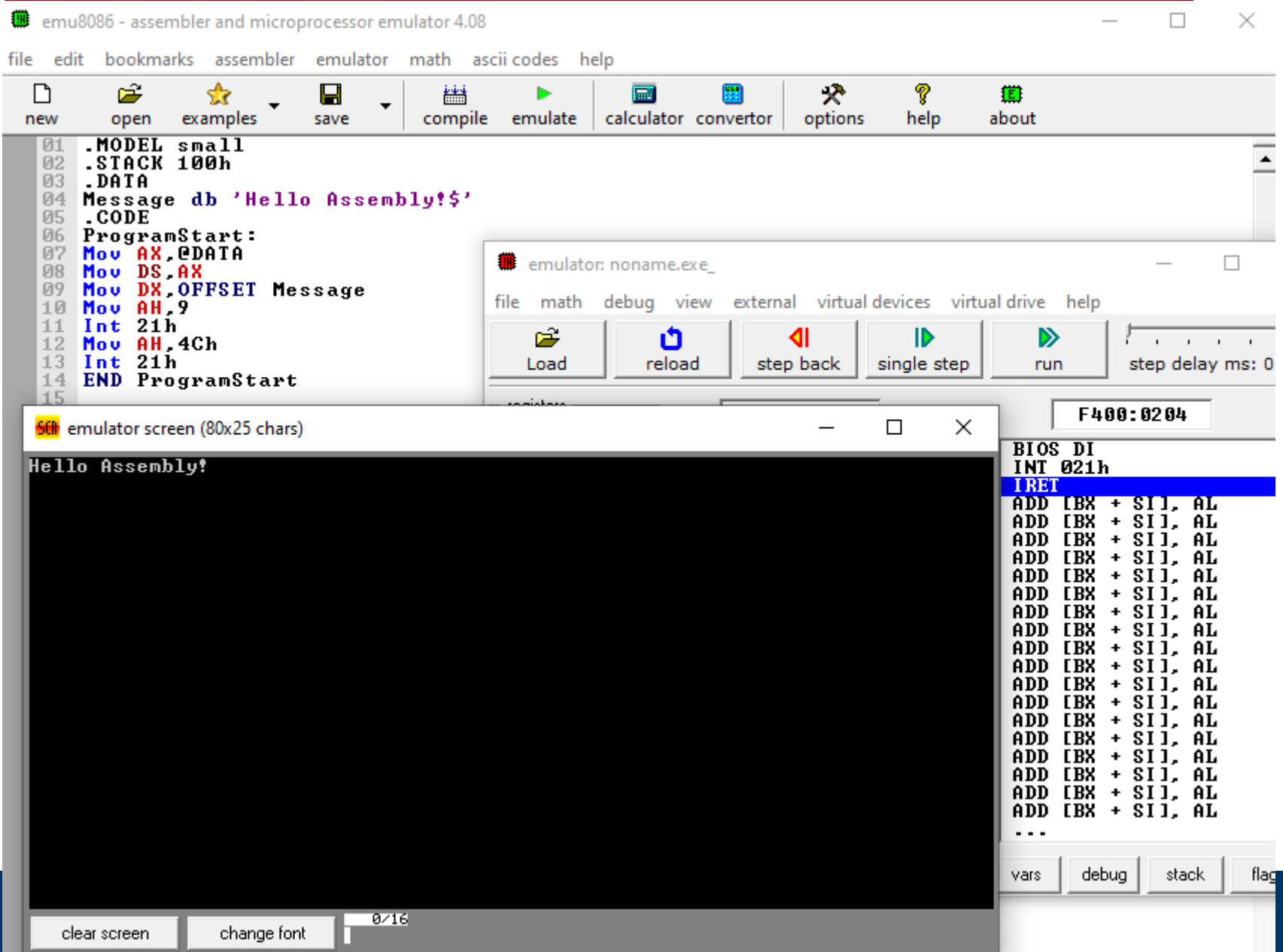
❑ Cấu trúc lặp REPEAT...UNTIL

Thực hiện công việc và kiểm tra điều kiện, nếu điều kiện vẫn sai thì lặp lại.

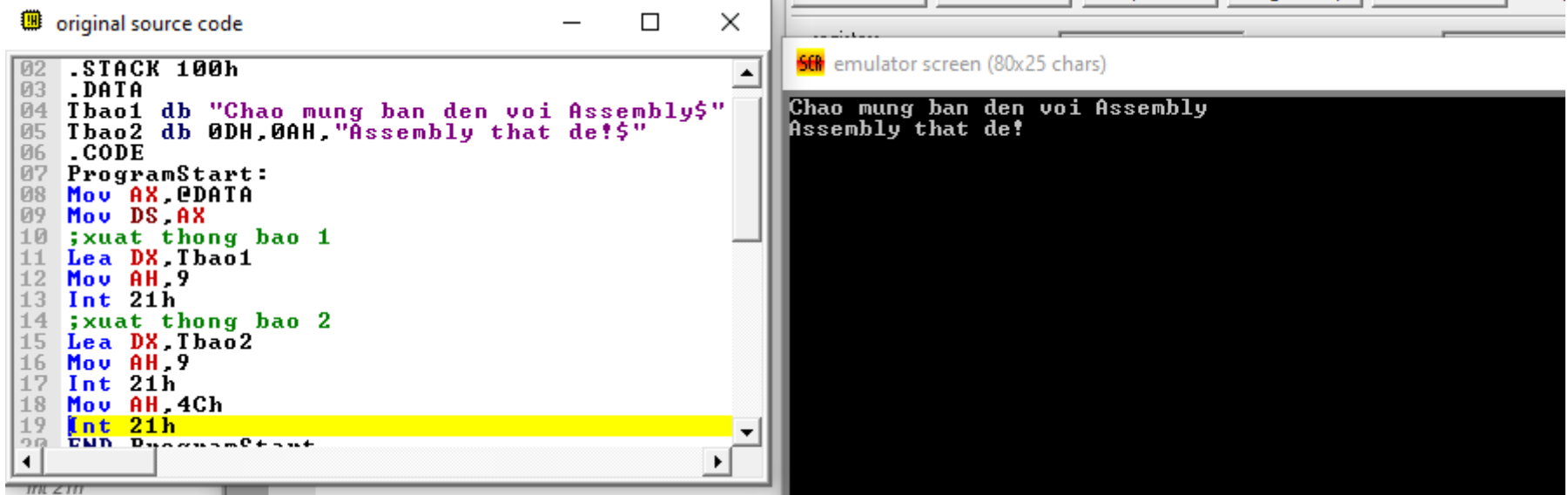
Ví dụ viết đoạn mã nhập vào kí tự cho đến khi gặp kí tự khoảng trắng .

```
        MOV AH,1      ; đọc kí tự
REPEAT_:
        INT 21h ; kí tự ở AL
;until
        CMP AL,' '    ; AL=' '?
        JNE REPEAT_
        ...           ;các lệnh tiếp theo sau vòng lặp
```

Bài 1: Viết chương trình hiện ra câu “Hello Assembly”



Bài 2: Viết chương trình hiện ra hai câu “Chao mung ban den voi Assembly” “Assembly that de!” Mỗi câu trên một dòng.



The image shows two windows from an assembly development environment. The left window, titled 'original source code', contains the following assembly code:

```
02 .STACK 100h
03 .DATA
04 Tbao1 db "Chao mung ban den voi Assembly$"
05 Tbao2 db 0DH,0AH,"Assembly that de!$"
06 .CODE
07 ProgramStart:
08 Mov AX,@DATA
09 Mov DS,AX
10 ;xuat thong bao 1
11 Lea DX,Tbao1
12 Mov AH,9
13 Int 21h
14 ;xuat thong bao 2
15 Lea DX,Tbao2
16 Mov AH,9
17 Int 21h
18 Mov AH,4Ch
19 Int 21h
20 End ProgramStart
```

The right window, titled 'emulator screen (80x25 chars)', shows the output of the program:

```
Chao mung ban den voi Assembly
Assembly that de!
```

Bài 3: Viết chương trình yêu cầu nhập một ký tự và xuất ra màn hình ký tự vừa nhập

```
01 .model small
02 .stack
03 .data
04 IBao1 db "Hay nhap mot ky tu: $"
05 IBao2 db 0DH,0AH,"Ky tu da nhap: $"
06 KyTu db ?
07 .code
08 ProgramStart:
09 Mov ax,@data
10 Mov ds,ax
11 ; nhac nhap
12 Lea dx, IBao1
13 Mov ah, 9
14 int 21h
15 ; nhap 1 ky tu
16 Mov ah, 1
17 Int 21h
18 Mov KyTu, al
19 ; thong bao ket qua
20 lea dx, IBao2
21 mov ah, 9
22 int 21h
23 ; hien thi ky tu da nhap
```

emulator: noname.exe_

file math debug view external virtual devices virtual drive help

Load reload step back single step run step delay ms: 0

registers

	H	L
AX	4C	66
BX	00	00
CX	01	56
DX	00	66

F400:0204

F4200:	FF	255	RES
F4201:	FF	255	RES
F4202:	CD	205	=
F4203:	21	033	!
F4204:	CF	207	±
F4205:	00	000	NULL

BIOS DI

INT 021h

IRET

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

ADD [BX + SI], AL

...

vars debug stack flags

emulator screen (80x25 chars)

```
Hay nhap mot ky tu: f
Ky tu da nhap: f
```

Bài 4: Viết chương nhập vào một ký tự. Chuyển ký tự đó sang ký tự hoa

```

01 .MODEL SMALL
02 .STACK 100h
03 .DATA
04 Msg1 DB 'Nhap vao ki tu thuong : $'
05 Msg2 DB 0Dh,0Ah,'Chuyen sang ki tu hoa la : '
06 Char DB ?, '$'
07 .CODE
08 Main PROC
09 MOV AX,@DATA
10 MOV DS,AX
11 ; In ra thong bao 1
12 LEA DX,Msg1
13 MOV AH,9
14 INT 21h
15 ; Nhap vao 1 ki tu thuong va d
16 MOV AH,1
17 INT 21h ; Doc 1 ki tu thuong v
18 SUB AL,20h ; Doi thanh ki tu h
19 MOV Char,AL
20 ; Hien len chu hoa
21 LEA DX,Msg2
22 MOV AH,9
23 INT 21h
24 ; Ket thuc chuong trinh
25 MOV AH,4Ch
26 INT 21h
27 Main ENDP

```

SCM emulator screen (80x25 chars)

```
hap vao ki tu thuong: h
huyen sang ki tu hoa la: H
```

