



Chương 7

HỢP NGỮ



Các thành phần

- Các ký hiệu dùng trong chương trình nguồn:
 - Chương trình nguồn là tập tin văn bản được tạo ra bằng bất kỳ phần mềm soạn thảo văn bản nào cũng được và cất lên đĩa với thuộc tính mặc nhiên là .ASM.
 - Nội dung gồm các lệnh của CPU hoặc các chỉ thị của MASM theo dạng sau :
 [<tên>] <toán tử> <toán hạng> ;<chú thích>
 - **Tên** do người sử dụng đặt bằng các ký tự sau :
 a - z , A - Z , 0 - 9 , ? , @ , _ , \$, .
 và bắt đầu bằng một *ký tự khác số*.
 - Ví dụ : code, data, start, main, decode, lap01, _flags1₂

- Trong chương trình nguồn còn cho phép dùng số ở các hệ khác nhau theo qui ước :

- Số hệ 2: **xxxxxxxxB**

Ví dụ : 10110101B

- Số hệ 8: **xxxQ** hoặc **xxxO**

Ví dụ : 123Q, 705O

- Số hệ 10: mặc nhiên hoặc **xxxxD**

Ví dụ : 32767, 4509D

- Số hệ 16: **xxxxH** và bắt đầu bằng số.

Ví dụ : 1A97H, 0E05BH

- Ngoài ra còn cho phép dùng một số ký tự khác như :

: . [] () < >

3

- Tên (name) :

- Do người sử dụng đặt để dùng thay cho một địa chỉ hay giá trị.

- Không phân biệt chữ hoa chữ thường.


- Chiều dài tùy ý nhưng chỉ có 31 ký tự đầu có nghĩa.

- Tên được dùng theo ba cách sau : Nhãn, biến, ký hiệu

- Nhãn (label)

- Dùng để đánh dấu vị trí trong chương trình, thay thế cho địa chỉ.


4



Nhãn

- Có hai thuộc tính là NEAR và FAR. Nhãn NEAR dùng cho các lệnh chuyển điều khiển trong cùng segment (nhảy , gọi chương trình con hoặc lặp vòng), nhãn FAR dùng cho các lệnh chuyển điều khiển giữa các segment.
- Nhãn NEAR được dịch thành địa chỉ 2 byte, nhãn FAR được dịch thành địa chỉ 4 byte.
- Các cách khai báo nhãn.
 - **<tên>**: thường đặt trước một lệnh hoặc chỉ thị. Nhãn này có thuộc tính NEAR.
 - Ví dụ : Main:

5



- **<tên> LABEL <thuộc tính>**

Đây là dạng khai báo tường minh.

Ví dụ : Screen LABEL FAR

- **<tên> PROC [NEAR/FAR]**

...

<tên> ENDP

Dùng để khai báo một chương trình con. Tên chương trình con là nhãn.

Ví dụ : Dump PROC FAR

RET

Dump ENDP

6



■ Biến (variable) :

- Dùng thay cho địa chỉ các ô nhớ chứa dữ liệu.
- Cần phân biệt địa chỉ của một biến với nội dung của nó.
- Biến có 5 thuộc tính như bảng sau:

Thuộc tính	Kích thước biến (byte)
BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10

7




- Biến BYTE và WORD thường dùng để chứa dữ liệu tính toán.
- Biến DWORD thường dùng làm biến con trỏ (pointer) chứa địa chỉ luận lý 4 byte.
- Biến QWORD và TBYTE thường dùng trong tính toán số thực dạng chấm động.


■ Các cách khai báo biến

- **<tên> Dx <biểu thức> [, ...]**
- Trong đó Dx là DB, DW, DD, DQ hoặc DT dùng để khai báo các biến có thuộc tính tương ứng là BYTE, WORD, DWORD, QWORD và TBYTE. Chỉ thị này cấp phát bộ nhớ cho biến và khởi động trị ban đầu chính là <biểu thức>.


8



- Ví dụ : Btable DB 0,1,2,3,4
Chiếm 5 byte trong bộ nhớ và Btable là tên của ô nhớ đầu tiên trong 5 byte đó (chính là ô nhớ chứa trị 0).
Dtable DD 0; Chiếm 4 byte trong bộ nhớ.
- **<tên> LABEL <thuộc tính biến>**
- Dạng khai báo biến này không cấp phát bộ nhớ cho biến mà chỉ để đặt tên cho một ô nhớ đã có. Điều này cho phép truy xuất một ô nhớ dưới những tên biến có thuộc tính khác nhau hoặc dùng để khai báo một biến có địa chỉ cố định.
- Ví dụ : Btable LABEL BYTE
- Wtable DW 0,1,2,3,4
- Btable và Wtable cùng đ/chỉ nhưng khác thuộc tính. 9




- Ký hiệu (symbol) hay hằng :
 - Dùng thay cho các giá trị, các hằng số, hằng ký tự hay hằng chuỗi để làm cho chương trình dễ hiểu, rõ ràng hơn.
 - Các cách khai báo hằng
 - **<tên> EQU <biểu thức>**
 - **<tên> = <biểu thức>**
 - trong đó <biểu thức> có thể là một ký hiệu khác, một tên gọi nhớ, một biểu thức hằng hay biểu thức địa chỉ.
 - Ví dụ :
 - ON EQU 1
 - LMAX EQU 25
 - XYZ = LMAX*4+15



BIỂU THỨC

- Biểu thức gồm 2 phần : toán hạng và toán tử
- Toán hạng : có thể là
 - Toán hạng tức thời :
 - Số ở các hệ 2, hệ 8, hệ 10 và hệ 16.
 - Hằng ký tự : 1 hoặc 2 ký tự ASCII đặt trong nháy đơn hay nháy kép.
 - Ví dụ :
`MOV AL,'@'`
`MOV CX,'OC'`
 - Chuỗi : nhiều ký tự ASCII trong nháy đơn hay nháy kép và chỉ dùng trong chỉ thị DB và mỗi ký tự chiếm 1 byte trong bộ nhớ (đĩ nhiên là không có dấu).
 - Ví dụ : `mess1 DB 'Khử niên kim nhật thử môn trung'`

11



- Toán hạng thanh ghi : tên các thanh ghi của vi xử lý 8086.
- Toán hạng bộ nhớ :
 - Cách dùng **địa chỉ hiệu dụng** cùng với các tên do người sử dụng đặt thay cho vùng địa chỉ trực tiếp.
 - Các toán hạng cấu trúc (khai báo bằng chỉ thị *STRUC*).
 - Ví dụ :
`MOV AL,tri_số[SI][2]`
`MOV reg.ax,AX`

12



Toán tử :

■ Toán tử thuộc tính :

- Mở rộng thuộc tính :PTR, :, SHORT, THIS, HIGH, LOW

■ <thuộc tính> **PTR** <biểu thức>

Trong đó <thuộc tính> có thể là BYTE, WORD, DWORD, QWORD, TBYTE, NEAR hay FAR.

■ Ví dụ : *CALL FAR PTR table[BX]*

MOV BYTE PTR array,1

■ <thanh ghi đoạn> : <biểu thức địa chỉ >

■ <tên đoạn> : <biểu thức địa chỉ >

■ <tên nhóm> : <biểu thức địa chỉ >

13



- Dùng để mở rộng, thay đổi địa chỉ đoạn của toán hạng bộ nhớ.

■ <tên đoạn> phải được khai báo bằng chỉ thị SEGMENT

■ <tên nhóm> phải được khai báo bằng chỉ thị GROUP.

■ Ví dụ : *MOV AX,CS:count*

ADD data:sum1,CX

■ **SHORT** <nhãn>

■ Dùng trong lệnh nhảy ngắn cách tương đối (-128 đến +127 byte).

■ Ví dụ : *JMP SHORT repeat*

■ **HIGH** <exp>

■ **LOW** <exp>

■ Lấy byte cao (HIGH) hay byte thấp (LOW) của biểu thức (thường là WORD).

14



- Toán tử trả trị :SEG, OFFSET, TYPE, LENGTH, SIZE, .TYPE
 - **SEG** <label>
 - **SEG** <biến>
 - trả về địa chỉ đoạn của biến.
 - Ví dụ : MOV AX, SEG array
 - **OFFSET** <nhãn>
 - **OFFSET** <biến>
 - Trả về địa chỉ trong đoạn của biến.
 - Ví dụ : MOV BX, OFFSET mess
 - MOV CX, OFFSET dgroup: count

15



- **TYPE** <nhãn> / <biến>
- trả về loại khai báo của biến BYTE = 1, WORD = 2, ..., STRUC = tổng số field.
- trả về loại khai báo của nhãn NEAR=0FFFFH, FAR=0FFFFEH.
- **LENGTH** <biến>
- trả về số phần tử dãy khai báo bằng DUP.
- **SIZE** <biến>
- trả về số byte của dãy khai báo bằng DUP.
- với dãy đơn thì $SIZE = LENGTH * TYPE$.

16

■ Toán tử số học :


- * Nhân
- / Chia nguyên
- **MOD** Chia lấy số dư
- **SHR** Dịch phải, dạng sử dụng : <trị> SHR <số bit>
- **SHL** Dịch trái, dạng sử dụng : <trị> SHL <số bit>
- + Cộng
- - Trừ, âm
- Chú ý phân biệt với các lệnh SHL, SHR. Toán tử thi hành khi dịch chương trình, trong khi lệnh chỉ thi hành khi chạy chương trình.

17

■ Toán tử quan hệ : Dùng để so sánh


- **EQ** bằng
- **NE** khác
- **LT** nhỏ hơn
- **LE** nhỏ hơn hoặc bằng
- **GT** lớn hơn
- **GE** lớn hơn hoặc bằng
- Nếu đúng trả về trị 0FFFFH (-1), còn sai trả về trị 0.
- Dùng trong các chỉ thị dịch có điều kiện.
- Ví dụ : if (TRI MOD 3) **GT** 1
MOV AL,1
else
MOV AL,0
endif

18



- **Toán tử luận lý: NOT, AND, OR, XOR**
 - Ví dụ : TRI1 EQU 176
 MASK1 EQU 1
 MOV AX, TRI1 AND NOT (MASK1 SHL 2)
- **Độ ưu tiên các toán tử :**
 - MASM thi hành toán tử có độ ưu tiên cao trước hoặc từ trái sang phải nếu cùng độ ưu tiên.
 - Độ ưu tiên các toán tử được sắp theo thứ tự giảm dần như sau :

19



Độ ưu tiên các toán tử

- 1) LENGTH, SIZE, WIDTH, MASK
Trong dấu (), < >, []
Toán tử biến cấu trúc <biến> . <vùng>
- 2) Toán tử mở rộng segment " : "
- 3) PTR, OFFSET, SEG, TYPE, THIS
- 4) HIGH, LOW
- 5) *, /, MOD, SHR, SHL
- 6) +, - (cả 2 loại trừ và âm)
- 7) Toán tử quan hệ
- 8) NOT
- 9) AND
- 10) OR, XOR
- 11) SHORT, .TYPE

20



CHỈ THỊ (DIRECTIVES)

- Là các lệnh của riêng hợp ngữ để trợ giúp người lập trình trong vấn đề tổ chức bộ nhớ, dịch có điều kiện, tạo bản in theo ý muốn (và điều khiển tham khảo chéo) và các định nghĩa macro. Các chỉ thị gồm có 4 loại : chỉ thị bộ nhớ, chỉ thị dịch có điều kiện, chỉ thị về in ấn và các chỉ thị macro.
- Chỉ thị về bộ nhớ: một số chỉ thị sau :
 - ASSUME, COMMENT, DB, DW, DD, DQ, DT, END, EQU, EXTRN, GROUP, INCLUDE, LABEL, ORG, PROC, PUBLIC, RECORD, SEGMENT, STRUC.

21



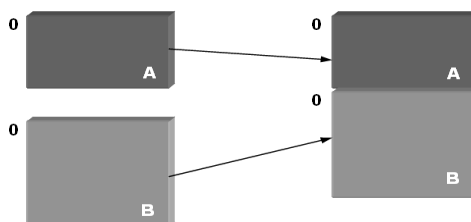
Chỉ thị segment

- <tên> SEGMENT [<align>] [<combine>] ['<class>']
...
- <tên> ENDS
 - <class> là tên lớp của đoạn dùng để ghép nhóm khi liên kết.
 - <align> Để qui định biên của các đoạn khi ghép kế tiếp nhau, cũng chính là qui định địa chỉ bắt đầu của đoạn kế như sau :

<align>		địa chỉ bắt đầu
BYTE	xxxx	xxxx (1 byte)
WORD	xxxx	xxx0 (2 byte)
DWORD	xxxx	xx00 (4 byte)
PARA	xxxx	0000 (16 byte)
PAGE	xxx0	0000 (256 byte)

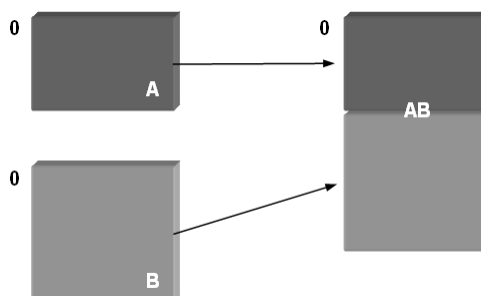
22

- `<combine>` có thể là `PUBLIC`, `COMMON`, `AT` `<exp>`, `STACK`, `MEMORY` hay không có dùng để qui định phương thức ghép các đoạn trong một lớp như sau :
 - Không có (ghép riêng biệt) : Các đoạn riêng biệt được nạp riêng. Chúng có thể liên tục về vật lý nhưng không liên tục về luận lý (hình vẽ). Mỗi đoạn có địa chỉ nền riêng của nó và có offset bắt đầu từ 0.

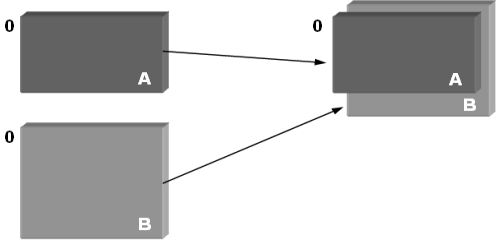


23

- `PUBLIC` và `STACK` : Các đoạn được nạp liên tục. Địa chỉ trong đoạn (offset) được tính liên tục từ đoạn đầu cho đến đoạn cuối. Hình vẽ cho thấy việc ghép hai đoạn A và B thành một đoạn duy nhất AB.




24



- **COMMON** : Các đoạn được nạp chồng lên nhau và có một địa chỉ nền duy nhất. Chiều dài đoạn lấy theo đoạn dài nhất (hình vẽ). Thường sử dụng đối với đoạn dữ liệu. Lúc đó các biến có cùng offset sẽ trùng lên nhau (có cùng nội dung).
- **MEMORY**: Các đoạn được nạp chồng lên nhau ở segment cao nhất của vùng nhớ. Kiểu này không dùng trong MS-LINK. MS-LINK xem như PUBLIC.

25



- Ta có thể dùng chỉ thị này để khai báo cho các đoạn chương trình, dữ liệu hoặc stack đều được cả.
 - Ví dụ : `code SEGMENT PARA PUBLIC 'code'`
 `...`
 `code ENDS`
 `data SEGMENT PARA PUBLIC 'data'`
 `...`
 `data ENDS`
 `stack SEGMENT STACK 'stack'`
 `...`
 `stack ENDS`

26



Chỉ thị ASSUME

- **ASSUME** <thanh ghi đoạn>: <tên> [, . . .]
- <thanh ghi đoạn> là CS, DS, ES và SS.
- <tên> là tên đoạn (được khai báo bằng chỉ thị SEGMENT), tên nhóm (được khai báo bằng chỉ thị GROUP), hay biểu thức có toán tử SEG.
- Ví dụ : **ASSUME** CS:code, DS:data, SS:stack
- Dùng để giả định thanh ghi đoạn đang được xem như chứa địa chỉ đoạn (segment) của đoạn hay nhóm chỉ định.
- Mục đích là để MASM tính địa chỉ trong đoạn (offset) của biến hoặc nhãn trong chương trình.

27



Chỉ thị END

- **END** [<biểu thức>] : Dùng để kết thúc chương trình
 - <biểu thức> là địa chỉ bắt đầu chạy chương trình.
 - Không cung cấp biểu thức trong trường hợp dịch chương trình có nhiều module. Lúc đó chỉ có một module được định ra địa chỉ bắt đầu chạy chương trình còn các module còn lại thì không cung cấp biểu thức.
 - Khi viết chương trình để tạo thư viện cũng không cung cấp địa chỉ chạy chương trình sau END.
 - Ví dụ : **END** start
 - Muốn viết chương trình để dịch ra tập tin .COM thì địa chỉ chạy chương trình phải định ở 100h. Lúc đó phải dùng thêm chỉ thị ORG để định địa chỉ dịch như sau :
 - ```

 ORG 100h
 main: jmp main0
 ...
 END main

```

28

## Chỉ thị Dx : Khai báo biến

- **<tên biến> Dx <biểu thức> [, <biểu thức>....]**
- Khi sử dụng, Dx phải viết là DB, DW, DD, DQ hay DT dùng để định nghĩa cho các loại biến lần lượt là BYTE, WORD, DWORD, QWORD, TBYTE.
- <biểu thức> có thể là:
  - Biểu thức hằng.
  - Biểu thức địa chỉ (dùng trong DW và DD).
  - Ký tự '?' (trị không xác định).
  - Chuỗi ASCII (chỉ dùng với DB).
  - <số phần tử> DUP (<biểu thức>[,...])(dùng để khai báo dãy).
- Ví dụ :
 


|        |           |                          |     |                 |
|--------|-----------|--------------------------|-----|-----------------|
| tri    | <b>DB</b> | 0,1,2,3                  |     |                 |
| jtable | <b>DD</b> | subr1,subr2,subr3,subr4  |     |                 |
| mess   | <b>DB</b> | 'The quick brown fox \$' |     |                 |
| day1   | <b>DW</b> | 100                      | DUP | (?,2 DUP (0,?)) |
| buffer | <b>DB</b> | 1024                     | DUP | ('BUFFER@ @')   |

29

- **<tên>EQU <biểu thức hằng>**
- **<tên>= <biểu thức hằng>**
- Dùng để khai báo hằng.
- Dùng EQU thì trị của hằng là không đổi trong suốt chương trình.
- Dùng dấu = thì trị của hằng có thể thay đổi bằng một định nghĩa hằng khác vẫn dùng dấu bằng (=).
- Ví dụ :
 


|     |            |    |
|-----|------------|----|
| OFF | <b>EQU</b> | 0  |
| ON  | =          | 1  |
| ... |            |    |
| ON  | =          | -1 |

30



- **INCLUDE** <tên tập tin>
- Dùng để chen thêm một phần chương trình nguồn từ trên đĩa vào đúng vị trí của chỉ thị trong khi dịch.
- Chỉ thị này giúp người lập trình có thể tách bớt các phần dữ liệu hay chương trình nào mà không cần sửa đổi nữa thành tập tin riêng để làm giảm kích thước chương trình nguồn, tiện cho việc sửa chương trình khi có lỗi.
- Cũng có thể tách phần các khai báo dùng chung cho nhiều module, tiết kiệm không gian đĩa khi lưu trữ chương trình nguồn.
- Ví dụ : **INCLUDE** MACRO.ASM

31




## IV. CẤU TRÚC CHUNG CỦA CHƯƠNG TRÌNH ASM

- **Khai báo dạng đơn giản**  
 [phần khai báo Macro, STRUC, RECORD, UNION] (nếu có)  
 chú ý : phần này có thể đặt chỗ khác, nhưng phải trước khi được sử dụng  
 .MODEL kiểu  
 .STACK độ lớn (tính theo byte)  
 .DATA  
     các khai báo biến  
 .CODE  
 Nhãn :  
     MOV AX, @DATA  
     MOV DS, AX  
     Thân chương trình  
     ...  
     ...  
     MOV AH, 4Ch ; lệnh trở về DOS  
     INT 21H  
     [các chương trình con] (nếu có)  
 END Nhãn

32






### Model : Khái báo quy mô kiểu dùng bộ nhớ

| Kiểu                 | Mô tả                                                                                  |
|----------------------|----------------------------------------------------------------------------------------|
| <b>Tiny (Hẹp)</b>    | mã lệnh và dữ liệu gói gọn trong một đoạn                                              |
| <b>Small (Nhỏ)</b>   | mã lệnh nằm trong 1 đoạn, dữ liệu 1 đoạn                                               |
| <b>Medium (TB)</b>   | mã lệnh nằm trong nhiều đoạn, dữ liệu 1 đoạn                                           |
| <b>Compact (Gọn)</b> | mã lệnh nằm trong 1 đoạn, dữ liệu trong nhiều đoạn                                     |
| <b>Large (lớn)</b>   | mã lệnh nằm trong nhiều đoạn, dữ liệu trong nhiều đoạn, không có mảng nào lớn hơn 64 K |
| <b>Huge (đồ sộ)</b>  | mã lệnh nằm trong nhiều đoạn, dữ liệu trong nhiều đoạn, các mảng có thể lớn hơn 64 K   |

33




### KHUNG CỦA CHƯƠNG TRÌNH ASM

- **Khái báo dạng chuẩn**  
 [phân khai báo Macro, STRUC, RECORD, UNION]  
 Chú ý : Có thể đặt chỗ khác, nhưng phải trước khi dùng

|           |                                     |        |         |                    |
|-----------|-------------------------------------|--------|---------|--------------------|
| StackName | SEGMENT                             | alig   | combine | 'class'            |
|           | Kiểu                                | Độ lớn | DUP     | (? hoặc 1 giá trị) |
| StackName | ENDS                                |        |         |                    |
| DataName  | SEGMENT                             | alig   | combine | 'class'            |
|           | Các khai báo biến                   |        |         |                    |
| DataName  | ENDS                                |        |         |                    |
| CodeName  | SEGMENT                             | alig   | combine | 'class'            |
| ASSUME    | tên_thanh_ghi_segment : tên_segment |        |         |                    |

34



## KHUNG CHƯƠNG TRÌNH

---


Nhãn :

```

MOV AX, Tên_Data_Seg
MOV DS, AX
Thân chương trình
...
...
MOV AH, 4CH
INT 21H
[các chương trình con] (nếu có)
CodeName ENDS
END Nhãn

```


35



### ■ Ví dụ : C.trình Hello.asm dùng để in ra dòng chữ hello, world.


- Stackseg      SEGMENT    PARA    STACK      'stack'  
DB 200 DUP(?)
- Stackseg      ENDS
- Dataseg        SEGMENT    WORD            'data'  
Message DB 'Hello, world', 0Dh, 0Ah, '\$'
- Dataseg        ENDS
- Codeseg        SEGMENT    WORD            'code'
- ASSUME        CS:Codeseg, DS:Dataseg, SS:Stackseg

36



- Begin :
- MOV AX, Dataség
- MOV DS,AX
- MOV DX, OFFSET Message
- MOV AH, 09H
- INT 21H
- MOV AH, 4CH
- INT 21H
- Codeség ENDS
- END Begin
- **Chú ý :** ASSUME DS:Dataség không tự động nạp địa chỉ của đoạn Dataség vào thanh ghi DS, ta phải nạp trực tiếp bằng 2 lệnh MOV AX, Dataség, MOV DS,AX


37



### Các bước thực hiện C.trình asm trên máy PC

- Dùng một chương trình soạn thảo bất kỳ (ở chế độ soạn thảo chương trình – program mode) để soạn thảo chương trình.
- Lưu tập tin chương trình trên đĩa có đuôi .asm. Ngôn ngữ assembly thuần túy không phân biệt chữ hoa, chữ thường.
- Dịch chương trình assembly vừa tạo để tạo ra tập tin đối tượng .obj dùng trình dịch MASM của Microsoft hoặc trình dịch TASM của hãng Borland.
- Liên kết các tập tin đối tượng để tạo tập tin khả thi.
- Chạy thử


38



## Các bước thực hiện chương trình

- Cú pháp đơn giản để dịch chương trình là
  - MASM tên\_file[.asm] <enter>
  - hoặc
  - TASM tên\_file[.am] <enter>
- Cú pháp đơn giản để liên kết là
  - LINK tên\_file[.obj] <enter> (đối với MASM)
  - TLINK tên\_file[.obj] <enter> (đối với TASM)
- Chạy thử : tên\_file[.exe] <enter>


39



## TẬP TIN .EXE VÀ .COM

- Sự khác nhau giữa C.trình .COM và .EXE :  
Kích thước chương trình, cách định nghĩa segment và cách khởi động chương trình.
- **Kích thước chương trình :**
  - Chương trình .EXE có thể có kích thước bất kỳ
  - Chương trình .COM bị giới hạn trong 1 segment và tối đa chỉ bằng 64KB bao gồm cả PSP. PSP là một khối 256 byte mà HĐH chèn ngay trước một chương trình .COM hay .EXE khi HĐH nạp chương trình từ đĩa vào bộ nhớ.


40



## TẬP TIN .EXE VÀ .COM

- C. trình .COM luôn nhỏ hơn chương trình .EXE tương ứng
- **Các segment :**
  - Chương trình EXE phải định nghĩa cả 3 segment : stack, data, code
  - Chương trình COM chỉ cần định nghĩa 1 segment code (dữ liệu và stack được lồng vào code segment)
- **Khởi động :**
  - **Chương trình .COM :** Khi HĐH nạp chương trình .COM từ đĩa vào bộ nhớ, HĐH tự động khởi động các thanh ghi segment bằng cách nạp địa chỉ của vùng PSP cho các thanh ghi này. Vì các thanh ghi segment CS và DS sẽ chứa đúng địa chỉ segment, do đó chương trình của ta không cần nạp chúng.


41



## TẬP TIN .EXE VÀ .COM


- **Chương trình .EXE :** khi HĐH nạp chương trình .EXE từ đĩa vào trong bộ nhớ, thực hiện các thao tác sau :
  - Tạo ra một tiền segment chương trình PSP (program segment prefix) 256 byte (100h) ở địa chỉ chia hết cho 16 trong bộ nhớ
  - Lưu chương trình cần thực thi trong bộ nhớ ngay sau PSP
  - Nạp địa chỉ của PSP vào các thanh ghi segment DS và ES
  - Nạp địa chỉ của segment mã vào thanh ghi CS và nạp địa chỉ offset của chỉ thị đầu tiên trong segment mã vào thanh ghi IP (thường là 0)

42



- nạp địa chỉ của stack vào thanh ghi SS, và nạp con trỏ stack SP kích thước của vùng stack.
- chuyển điều khiển tới chương trình để thực thi, thường bắt đầu ở chỉ thị đầu tiên trong segment mã.
- Theo cách mô tả trên thì trình nạp của HĐH khởi động chính xác các thanh ghi CS, SS, còn các thanh ghi DS và ES được nạp địa chỉ của PSP chứ không phải địa chỉ của segment dữ liệu, đó chính là lý do tại sao ta phải khởi động thanh ghi DS tới địa chỉ của segment dữ liệu.

43



## Cấu trúc của chương trình .COM

- **Dạng đơn giản :**
  - [Khai báo MACRO, STRUC, RECORD, UNION]
  - .MODEL Tiny hoặc Small
  - .CODE
  - ORG 100H
  - ProgramStart:
  - JMP Start
  - Khai báo các biến (dữ liệu)
  - Start :
  - Thân của C.trình (không cần k/động các t/ghi segment)
  - MOV AH, 4CH ; trở về DOS
  - INT 21H
  - (có thể thay hai lệnh trên bằng 1 lệnh là INT 20h)
  - [khai báo các chương trình con]
  - END ProgramStart

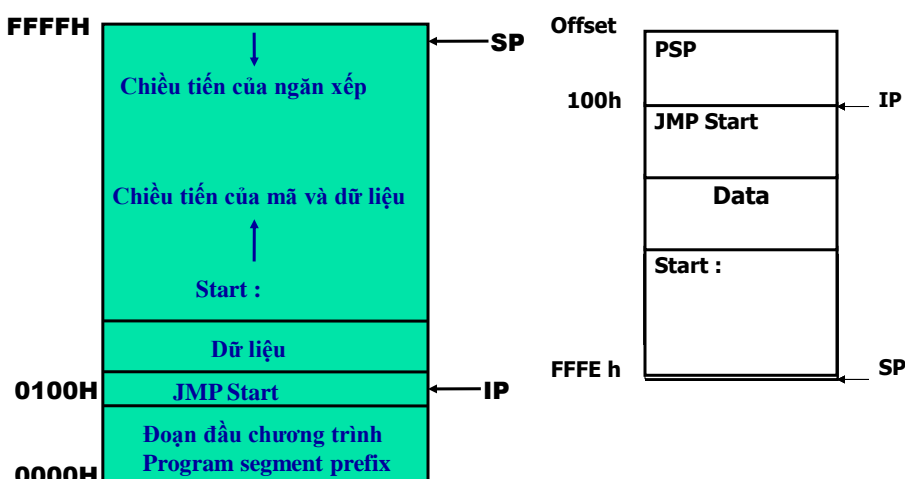
44

## Cấu trúc của chương trình .COM

- Khai báo dạng chuẩn
  - [Khai báo MACRO, STRUC, RECORD, UNION]
  - CODE SEGMENT
  - ORG 100H
  - ASSUME CS:CODE, DS:CODE, SS: CODE
  - ProgramStart:
    - JMP START
    - Khai báo các biến
  - START:
    - Các lệnh
    - MOV AH, 4CH ; trở về DOS
    - INT 21H ;(có thể dùng 1 lệnh INT 20h)
    - [khai báo các chương trình con]
  - CODE ENDS
  - END ProgramStart

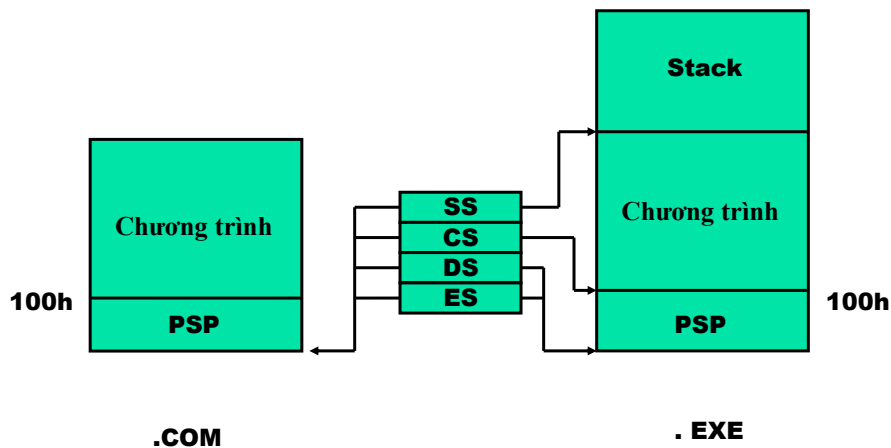
45

## Minh họa cấu trúc chương trình .COM



46

## So sánh chương trình .EXE và .COM



47

## Biên dịch chương trình .COM

### ■ Dùng MASM

- C. trình nguồn viết dưới dạng .COM hay .EXE đều phải được dịch sang tập tin .OBJ và được liên kết để tạo ra chương trình thực thi .EXE
- Nếu viết chương trình ở dạng .EXE, ta có thể thực thi chương trình ngay, nhưng nếu viết chương trình nguồn ở dạng .COM, trình liên kết sẽ thông báo lỗi như sau :
- Warning : No stack Segment
- Ta bỏ qua thông báo này, vì chương trình .COM thực sự không định nghĩa stack segment.
- Bây giờ ta chuyển tập tin .exe vừa biên dịch sang .com dùng EXE2BIN như sau
- EXE2BIN    tên\_file[.exe]    tên\_file.com

48





## Dịch chương trình .COM

- **Dùng TASM**

- Ta tiến hành biên dịch sang file .obj bình thường :  
TASM tên\_file.asm
- Liên kết bằng Tlink với lựa chọn t như sau :
- TLINK /t tên\_file[.obj]

- **Ví dụ 1 :** Viết chương trình tính tổng một dãy số nguyên để được dưới dạng .COM