Chương 1 (tiếp)

Ngôn ngữ Assembly và cách lập trình

Mục đích:
Giới thiệu các kiến thức, các thành phần cơ bản của chương trình con, macro, directive include, chương trình Assembly để được chương trình thực hiện dạng .com và chương trình đa tệp thuần túy Assembly.

1.6 Chương trình con

1.6.1 Ý nghĩa của chương trình con

- Làm cho chương trình có cấu trúc,
- Tiết kiệm vùng nhớ.

1.6.2 Cơ chế hoạt động khi 1 chương trình con được gọi

Khi 1 chương trình con được gọi, máy tính tiến hành các bước sau:

Bước 1: Tham số thực sẽ được đưa vào ngăn xếp (với chương trình con có đối),

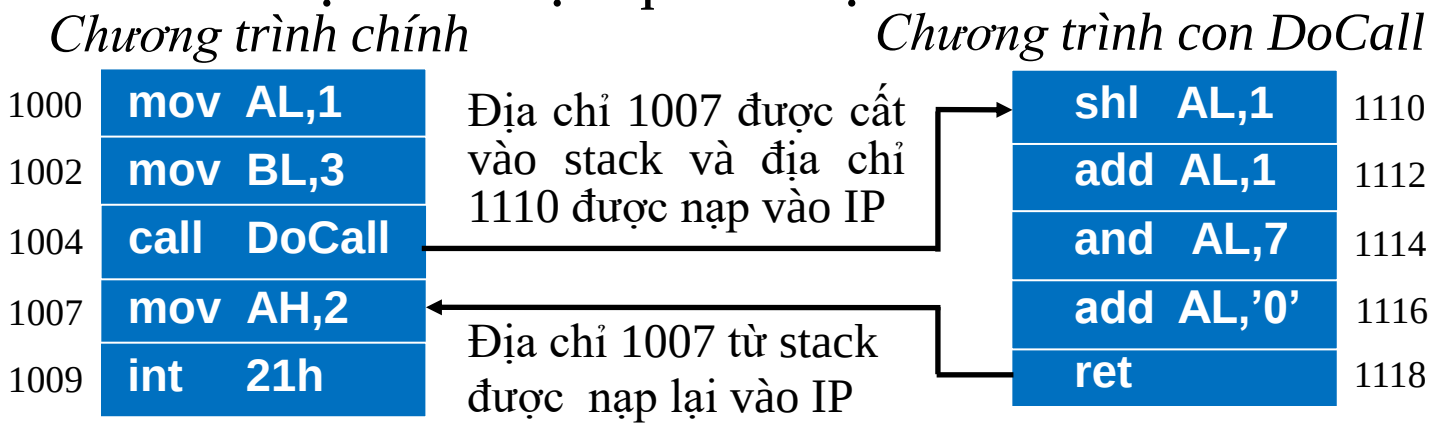
Bước 2: Địa chỉ ô nhớ chứa mã máy lệnh tiếp theo sẽ được đưa vào ngăn xếp (2 hoặc 4 byte tùy vào CT con là NEAR/FAR),

Bước 3: Hệ điều hành đưa địa chỉ ô nhớ chứa mã máy lệnh đầu của CT con vào CS:IP, rẽ nhánh vào CT con,

Bước 4: Thực hiện thân CT con cho đến khi gặp RET thì vào ngăn xếp lấy địa chỉ lệnh tiếp theo (đã cất ở Bước 2) đưa vào CS:IP và quay về chương trình đã gọi nó,

Bước 5: Tiếp tục chạy chương trình đang dở.

Cơ chế 5 bước được thể hiện qua ví dụ sau:



1.6.3 Cú pháp một chương trình con Assembly

tên chương trình con PROC [NEAR/FAR]
Bảo vệ các thanh ghi mà thân chương trình con sẽ phá vỡ
các lệnh thân chương trình con
Hồi phục các thanh ghi mà thân chương trình con đã phá vỡ
ret
tên chương trình con ENDP

trong đó:

- *Tên chương trình con*: bất kỳ một định danh nào,
- *Vấn đề NEAR/FAR*:
  - CT con là NEAR khi mã máy của CT chính và CT con cùng nằm trên 1 segment (64k) và 2 vùng nhớ chỉ khác nhau phần địa chỉ offset. Địa chỉ ô nhớ chứa lệnh tiếp theo được cất vào stack (Bước 2 mục 1.6.2) chỉ cần 2 byte phần địa chỉ offset.
  - CT con là FAR khi mã máy của CT chính và CT con nằm trên các segment khác nhau và 2 vùng nhớ không chỉ khác nhau phần địa chỉ offset mà cả phần địa chỉ segment. Địa chỉ ô nhớ chứa lệnh tiếp theo cất vào stack (Bước 2 mục 1.6.2) cần 4 byte (2 byte phần địa chỉ segment, 2 byte phần địa chỉ offset).

Mặc định (khi viết CT con không khai báo NEAR/FAR):

- Nếu khi viết CT sử dụng directive điều khiển segment dạng đơn giản thì directive .MODEL sẽ xác lập NEAR/FAR cho CT con: với các mô hình bộ nhớ tiny, small và compact thì CT con là NEAR, còn medium, large và huge thì CT con là FAR),

- Nếu khi viết CT sử dụng directive điều khiển segment dạng chuẩn thì mặc định CT con là NEAR, còn muốn CT con là FAR thì phải khai báo tường minh FAR khi viết CT con.

- *Vấn đề Bảo vệ/Hồi phục các thanh ghi trong thân CT con*:

Trong lập trình bằng ngôn ngữ Assembly rất dễ xảy ra trường hợp là CT chính và CT con cùng dùng 1 thanh ghi làm toán hạng (vì số lượng các thanh ghi là có hạn). Như vậy CT con có thể xóa mất giá trị mà CT chính đã đặt vào thanh ghi đó. Trong trường hợp này, muốn các thanh ghi vẫn giữ các giá trị mà CT chính đã đặt vào thì CT con phải tiến hành bảo vệ các thanh ghi trước khi thực hiện các lệnh của CT con để sau này trước khi ra khỏi chương trình con sẽ hồi phục lại các giá trị đó trở lại các thanh ghi. Để tiến hành cơ chế bảo vệ đó, có 2 giải pháp thường dùng:

- Cơ chế PUSH-POP: Đầu CT con nên tiến hành cất giá trị của tất cả các thanh ghi mà CT con sẽ phá vỡ vào ngăn xếp (PUSH) và trước khi ra khỏi CT con phải hồi phục lại các giá trị đó từ ngăn xếp vào các thanh ghi tương ứng (POP),

– Dùng theo qui ước: qui định dùng một số thanh ghi dành cho chương trình chính và chương trình con không được dùng đến các thanh ghi đó.

- *Vấn đề chuyển giao tham số:* Dữ liệu cần được trao đổi từ chương trình gọi (CT chính) và chương trình được gọi (CT con) và ngược lại. Có 3 cách chuyển giao tham số theo cả 2 hướng, đó là:
  – Thông qua các thanh ghi: Với cách chuyển giao này, chương trình gọi (CT chính) hoặc chương trình được gọi (CT con) chỉ cần đặt giá trị vào thanh ghi và sau đó chương trình được gọi (CT con) hoặc chương trình gọi (CT chính) sẽ sử dụng giá trị này khi thâm nhập vào thanh ghi đó,
  – Thông qua các biến ngoài (biến global): Biến khai báo toàn cục có tác dụng trong phạm vi toàn chương trình.
  – Thông qua ngăn xếp: Cách chuyển giao này thường được dùng khi liên kết ngôn ngữ bậc cao với ngôn ngữ Assembly trong trường hợp ngôn ngữ bậc cao giả thiết CT con viết bằng ngôn ngữ Assembly có đối.

Ví dụ 1: Hãy viết chương trình con nhận một số nguyên (-32768 đến 32767- giới hạn của thanh ghi 16 bit) vào từ bàn phím.

Các bước tiến hành:

– Dùng chức năng 1 của ngắt int 21h để nhận 1 ký tự từ bàn phím vào thanh ghi AL. Kiểm tra liệu ký tự vừa nhận có phải là:
  * Enter (kết thúc nhận 1 số) hoặc
  * dầu '-' (số nhận là âm) thì tiến hành dựng cờ dấu lên 1 (biến cờ dấu là 0 thể hiện số nhận là dương và biến cờ dấu bằng 1 thể hiện số nhận là âm),
– Nếu không phải 2 trường hợp trên thì ký tự nhận là 1 ký tự số và chuyển ký tự số từ dạng ASCII sang số (trừ cho 30h), sau đó cộng số vừa vào với phần số đã vào trước sau khi nhân cho 10,
– Nếu kết thúc nhận số (Enter) thì tiến hành kiểm tra liệu biến cờ dấu là 0 (số nhận dương) hay là 1 (số nhận âm). Nếu biến cờ dấu là 1 thì tiến hành đổi dấu trước khi đưa kết quả vào thanh ghi AX.
– *Chú ý*: Chương trình con này không cho phép đánh sai và sửa.

```
VAO_SO_N    PROC
        push    BX CX DX SI   ; Bảo vệ các thanh ghi mà thân CT con dùng
        mov     BX,10         ; BX=10 là 1 số hạng nhân
        xor     CX,CX         ; CX=0 (phần số đã vào trước, lúc đầu=0)
        mov     SI,CX         ; SI=biến cờ dấu (giả thiết lúc đầu là dương)
VSN1:   mov     AH,1          ; Chờ nhận 1 ký tự từ bàn phím
        int     21h
        cmp     AL,0dh        ; Ký tự  nhận có phải là Enter?
        je      VSN3          ; Đúng là Enter (kết thúc nhận số), nhảy đếnVSN3
        cmp     AL,'-'        ; Ký tự nhận có phải dấu '-' ?
        jne     VSN2          ; Không phải thì đó là 1 ký tự số, nhảy  đến VSN2
        inc     SI            ; còn nếu là dấu '-' thì đưa biến cờ dấu lên 1
        jmp     VSN1          ; Nhảy về nhận ký tự tiếp theo
VSN2:   sub     AL,30h        ; Chuyển ký tự số dạng ASCII sang số
        xor     AH,AH         ; AH=0 để AX=AL (số vừa vào)
        xchg    CX,AX         ; AX=phần số đã vào trước,CX=số vừa nhận
        mul     BX            ; AX*10 (nhân phần số đã vào trước với 10)
        add     CX,AX         ; CX=số vừa vào+phần số đa vào trước*10)
        jmp     VSN1          ; Quay về chờ nhận ký tự tiếp theo
```

```
VSN3:
        and     SI,SI           ;  Xét biến cờ dấu là 0 (dương) hay 1 (âm)
        jz      VSN4            ;  Nếu số nhận là dương, nhảy đến VSN4
        neg     CX              ;  còn số nhận là âm thì đổi dấu
VSN4:
        mov     AX,CX           ; Số nhận được sẽ đặt trong thanh ghi AX
        pop     SI DX CX BX     ; Hồi phục các thanh ghi
        ret
VAO_SO_N        ENDP
```

Ví dụ 2: Viết chương trình con hiện nội dung 1 số nguyên có trong thanh ghi AX lên màn hình dạng cơ số 10.

Các bước tiến hành:

Trước tiên phải kiểm tra số cần hiện là âm hay dương.

– *Nếu số cần hiện là dương*: tiến hành chia AX cho 10. Số bị chia dạng 32 bit được đặt ở DX:AX (trong trường hợp này DX=0) và số chia là 10 được đặt trong 1 thanh ghi đa năng, ví dụ BX. Kết quả của lệnh chia là thanh ghi AX chứa thương và DX chứa dư.

Hiệu chỉnh phần dư ra dạng ASCII (cộng với 30h) và tạm cất vào ngăn xếp. Thanh ghi CX sẽ đếm số lần đã cất vào ngăn xếp. Sau mỗi lần chia, kiểm tra liệu thương (ở AX) đã bằng 0 chưa? Nếu thương khác 0, tiếp tục lấy thương chia cho 10, ngược lại nếu thương bằng 0 thì dừng việc chia và thiết lặp vòng lặp lấy các giá trị đã cất trong ngăn xếp lần lượt hiện lên màn hình,

– *Nếu số cần hiện là âm*: thì hiện dấu '-', sau đó đổi dấu số cần hiện trở thành số nguyên dương và hiện 1 số nguyên dương sau dấu '-'.

```
HIEN_SO_N     PROC
      push   AX BX CX DX      ; Bảo vệ các thanh ghi
      mov    BX,10            ; BX=số hạng chia là 10
      xor    CX,CX            ; CX=số lần cất vào ngăn xếp (lúc đầu=0)
      and    AX,AX            ; Số nằm ở AX là dương hay âm? (dựng cờ dấu)
      jns    HSN1             ; Nếu là số dương thì nhảy đến HSN1,
      push   AX               ; còn âm thì tạm cất số cần hiện vào ngăn xếp
      mov    AL,'-'
      mov    AH,0Eh           ; và hiện dấu '-' lên màn hình
      int    10h
      pop    AX               ; Hồi phục số âm đã tạm cất vào ngăn xếp
      neg    AX               ; AX=|số cần hiện|
```

```
HSN1:
    xor     DX,DX           ; DX=0
    div     BX              ; Chia giá trị có trong DX:AX cho 10 (BX)
    add     DX,30h          ; Hiệu chỉnh số phần dư ra dạng ASCII
    push    DX              ; Cất phàn dư vào ngăn xếp
    inc     CX              ; Tăng số lần cất vào ngăn xếp
    and     AX,AX           ; Liệu thương đã bằng 0?
    jnz     HSN1            ; Nếu thương #0 thì quay về  tiếp tục chia
HSN2:                       ; Vòng lặp hiện các ký tự số đã cất ở ngăn xếp
    pop     AX              ; Lấy từ ngăn xếp từng ký tự vào AL
    mov     AH,0Eh          ; Hiện ký tự ra màn hình
    int     10h
    loop    HSN2
    pop     DX CX BX AX     ; Hồi phục các thanh ghi
    ret
HIEN_SO_N       ENDP
```

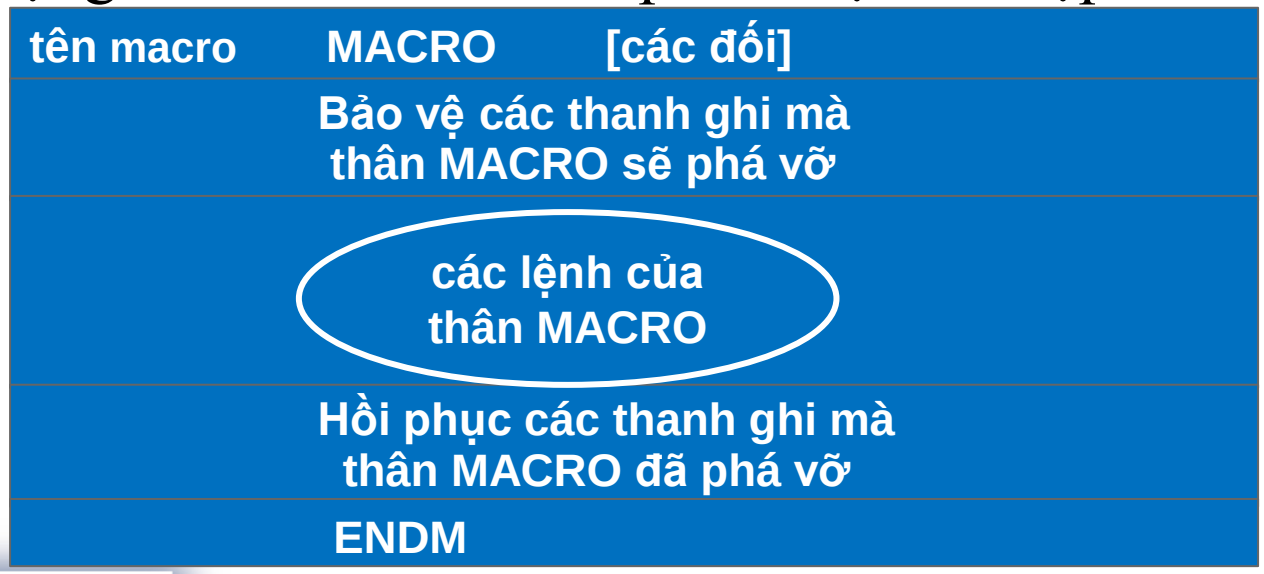
1.7 MACRO

1.7.1 Ý nghĩa của MACRO

Cho phép gán một tên thay cho 1 khối lệnh và sau đó khi chương trình dịch gặp tên này ở đâu thì khối lệnh đó sẽ được dịch và đặt khối mã lệnh đó vào vị trí gọi tên MACRO. Nói một cách khác MACRO cho phép người lập trình Assembly tạo một lệnh mới trên cơ sở gộp khối lệnh chuẩn của ngôn ngữ Assembly.

1.7.2 Khai báo MACRO (cú pháp để tạo 1 lệnh mới)

Muốn sử dụng 1 macro thì macro phải được xác lập với cú pháp:



Ví dụ 1: Hãy xác lập (khai báo 1 lệnh mới) cho phép xóa màn hình.
Cách làm: Cơ chế màn hình ở chế độ văn bản là mỗi khi đặt chế độ thì màn hình bị xóa và con trỏ ở góc trên trái. Để xóa màn hình mà không thay đổi chế độ thì nên lấy chế độ hiện thời, sau đó đặt lại.

Lấy chế độ màn hình hiện thời: Chức năng: 0Fh của ngắt int 10h	
Vào	AH = 0Fh
Ra	AL←mode màn hình hiện thời

Đặt chế độ cho màn hình: Chức năng: 0h của ngắt int 10h	
Vào	AH = 0h; AL←số mode cần đặt
Ra	Không

MACRO sẽ có dạng sau:

```
CLRSCR  MACRO
        push    AX          ; Bảo vệ thanh ghi AX
        mov     AH,0Fh      ; Lấy mode màn hình hiện hành
        int     10h         ; AL chứa số mode hiện hành
        mov     AH,0h       ; Đặt mode màn hình
        int     10h         ; có trong AL
        pop     AX          ; Hồi phục thanh ghi AX
        ENDM
```

Ví dụ 2: Hãy xác lập (khai báo 1 lệnh mới) cho phép hiện một xâu ký tự lên màn hình.

Cách làm: Có nhiều cách làm song tốt nhất sử dụng chức năng thứ 9 của ngắt int 21h (cho phép hiện xâu kết thúc '$' lên màn hình).

```
HienString  MACRO   xau
            push    AX DX       ; Bảo vệ  các thanh ghi
            lea     DX,xau      ; Chức năng hiện 1 xâu (kết thúc bằng
            mov     AH,9        ; dấu '$' ) lên màn hình tại vị trí con
            int     10h         ; trỏ đang đứng
            pop     DX AX       ; Hồi phục các thanh ghi
            ENDM
```

Chú ý:

- *Xác định phạm vi có hiệu lực nhãn nhảy bên trong MACRO*: Khi sử dụng MACRO (bên trong có nhãn nhảy) 2 lần trở lên thì vi phạm tính chất nhãn phải là duy nhất trong một chương trình. Để tránh sự vi phạm trên hãy sử dụng directive LOCAL để xác định phạm vi có hiệu lực của nhãn với cú pháp sau:

  LOCAL tên nhãn nhảy

- *So sánh chương trình con và MACRO:*
  - Sử dụng chương trình con tiết kiệm vùng nhớ hơn MACRO,
  - Sủ dụng MACRO linh hoạt hơn chương trình con vì:
    - MACRO có đối (cho phép trao đổi tham số),
    - Với việc sử dụng các lệnh điều khiển điều kiện khi dịch chương trình có thể tạo được những mã lệnh khác nhau để thực hiện hàng loạt những nhiệm vụ gần giống nhau,
    - Có khả năng sử dụng các lệnh lặp một khối lệnh khi dịch.

1.7.3 Cách sử dụng lệnh mới (được xác lập bởi MACRO)

Một MACRO sau khi được khai báo (xác lập) thì tên MACRO trở thành một lệnh mới của ngôn ngữ Assembly.

Việc sử dụng lệnh mới chỉ đơn giản là gọi tên MACRO và thay tham số thực cho đối nếu MACRO có đối. Chương trình dịch Assembler gặp các lệnh mới đó thì nó sẽ dịch các lệnh của thân MACRO và đặt kết quả mã lệnh dịch được tại điểm đó.

Chú ý: MACRO phải được khai báo (lệnh mới phải được xác lập) trước khi sử dụng.

1.8 Directive INCLUDE và tệp INCLUDE

1.8.1 Ý nghĩa

Cho phép chèn nội dung một tệp ngoài (tệp INCLUDE) vào chương trình đang viết.

1.8.2 Cú pháp chèn

```
INCLUDE   tên ổ đĩa:\đường dẫn\tên tệp.đuôi
```

1.8.3 Vấn đề tìm tệp INCLUDE

Chương trình dịch Assembler tìm tệp INCLUDE như thế nào?

- Nếu sau directive INCLUDE chỉ rõ tên ổ đĩa, đường dẫn, tệp tệp thì tất nhiên chương trình dịch sẽ chỉ tìm theo sự xác định trên,
- Còn nếu sau directive INCLUDE chỉ xác định có tên tệp thôi thì chương trình dịch trước tiên tìm tệp đó trong thư mục hiện hành, nếu không tìm thấy thì sẽ tìm tệp đó ở danh mục xác định bởi tùy chọn –i ở dòng lệnh dịch, ví dụ như:

```
tasm  -i c:\tên các thư mục\tên tệp INCLUDE
```

- Nếu không nằm trong 2 trường hợp trên CT dịch thông báo lỗi.

1.9 Chương trình dạng .COM

1.9.1 Sự khác nhau giữa CT dang .EXE và .COM

- Với chương trình dạng .EXE có thể định nghĩa được 4 phân đoạn (segment) khác nhau, đó là: phân đoạn lệnh (code segment), phân đoạn dữ liệu (data segment), phân đoạn dữ liệu thêm (extra segment) và phân đoạn ngăn xếp (stack segment).
- Với chương trình dạng .COM chỉ có thể định nghĩa một phân đoạn duy nhất và đó là phân đoạn lệnh và do đó phải đặt dữ liệu và ngăn xếp vào phân đoạn này.

1.9.2 Một số nhận xét khi viết chương trình để được dạng .COM

- *Vấn đề khai báo biến*: Chương trình dạng .COM chỉ có một phân đoạn duy nhất và đó là phân đoạn lệnh (code). Vậy thì khai báo biến ở đâu? Câu trả lời là: biến được khai báo trong vùng nhớ cấp phát cho phân đoạn mã máy. Hệ điều hành dành một phần vùng nhớ RAM cấp phát cho mã máy để khai báo biến và vì vậy chương trình dạng .COM có khai báo biến thường có dạng:

```
Nhãn CT:  [jmp  nhãn khác
          khai báo biến]
  nhãn khác:
          thân chương trình
          . . . .
```

- *Directive ORG 100h*: Các lệnh của chương trình dạng .COM khi nạp vào vùng nhớ trong (RAM) bao giờ cũng bắt đầu tại OFFSET 100h. Vùng nhớ từ 0 đến 0ffh (256 byte) là vùng PSP (Program Segment Prefix) dùng để chứa các thông tin mà hệ điều hành DOS cần đến sau khi chương trình dạng .COM được nạp vào bộ nhớ trong. Vì vậy để tạo được tệp thực hiện dạng .COM, chương trình được viết bằng ngôn ngữ Assembly phải có lệnh điều khiển (directive) ORG 100h để chỉ cho thanh ghi IP trỏ đến lệnh đầu tiên trong phân đoạn lệnh và phải đặt lệnh điều khiển này ở ngay đầu mỗi chương trình.
- *Trở về DOS*: Chương trình dạng .EXE sử dụng chức năng 4Ch của ngắt int 21h. Chương trình .COM cũng có thể sử dụng chức năng trên, song tốt hơn nên sử dụng ngắt int 20h.

1.10 Cấu trúc một chương trình Assembly

Để được dạng .EXE

[Phần khai báo MACRO,STRUC,RECORD và UNION] ; (Nếu có)

Phần khai báo segment

Dạng đơn giản

```
.MODEL   kiểu
.STACK   độ lớn          ; (tính theo byte)
[.DATA                   ; Nếu có khai báo
Khai báo biến]           ; biến
.CODE
Nhãn CT:
  [mov  AX,@data         ; Nếu có khai báo
  mov  DS,AX]            ; .DATA và biến
        thân CT chính
  mov  AH,4Ch            ; Về DOS
  int    21h
[ các chương trình con ] ; Nếu có
  END nhãn CT
```

Dạng chuẩn

```
tên _stack _segment  segment
          kiểu độ lớn dup(?)
tên_stack_segment    ends
[tên _data _segment  segment   ; Nếu có
          khai báo biến        ; khai báo
tên_data_segment     ends]     ; biến
tên _code _segment  segment
assume tên thanh ghi segment:tên segment
Nhãn CT:
  [mov  AX ,tên_data_segment   ; Nếu có khai
  mov  DS,AX]                  ; báo biến
        thân CT chính
  mov  AH,4Ch                  ; Về DOS
  int    21h
[ các chương trình con ]       ; Nếu có
tên _code _segment  ends
  END nhãn CT
```

Để được dạng .COM

[Phần khai báo MACRO,STRUC,RECORD và UNION] ; (Nếu có)

Phần khai báo segment

Dạng đơn giản	*Dạng chuẩn*
.MODEL tiny/small/compact	tên _code _segment segment
.CODE	assume CS: tên_code segment,
ORG 100h	DS: tên_code_segment,
Nhãn CT:	SS: tên_code_segment
[jmp nhãn_khác ; Nếu có khai báo	ORG 100h
khai báo biến] ; biến	Nhãn CT:
nhãn khác:	[jmp nhãn khác ; Nếu có khai
thân CT chính	khai báo biến] ; báo biến
int 20h ; Vê DOS	nhãn khác:
[các chương trình con] ; Nếu có	thân CT chính
END nhãn CT	int 20h ; Về DOS
	[các chương trình con] ; Nếu có
	tên _code _segment ends
	END nhãn CT

Chú ý:

- Phần CT con có thể đặt trước CT chính thay vì đặt sau,
- Để được CT dạng .COM khi liên kết phải có tùy chọn /t:

tlink/t tên tệp

Một số bài tập:

Trong phần bài tập, hãy giả thiết có tệp lib1.asm chứa 2 MACRO xóa màn hình (với tên clrscr) và hiện 1 xâu ký tự kết thúc bằng '$' lên màn hình (với tên là HienString) và tệp lib2.asm chứa 2 CT con nhận 1 số nguyên (với tên là VAO_SO_N) và hiện nội dung 1 số có trong AX ra màn hình dạng cơ số 10 (với tên là HIEN_SO_N). Hai tệp trên giả thiết đặt ở thư mục INCLUDE của ổ đĩa C:\.

Bài tập 1: Hãy viết chương trình tính n! ($0 \le n \le 7$). Khi chạy có dạng:

```
Hay vao n:5
Giai thua cua 5 la: 120
Co tiep tuc CT(c/k)?_
```

Để được tệp dạng .EXE:

```
INCLUDE c:\include\lib1.asm        ; Khai báo 2 MACRO (clrscr và HienString)
.MODEL small
.STACK  100h                       ; Xác lập vùng nhớ 256 byte cho STACK
.DATA
  m1 db  13,10,'Hay vao n: $'
  m2 db  13,10,'Giai thua cua $'
  m3 db  ' la : $'
  m4 db  13,10,'Co tiep tuc CT (c/k)?$'
```

```
.CODE
 PS: mov    AX,@data        ; Đưa phần địa chỉ segment của vùng nhớ
     mov    DS,AX           ; chứa dữ liệu vào DS
     clrscr                 ; Xóa màn hình
     HienString  m1         ; Hiện xâu m1
     call   VAO_SO_N        ; Vào 1 số (số vào nằm ở thanh ghi AX)
     HienString  m2         ; Hiện xâu m2
     call   HIEN_SO_N       ; Hiện số n vừa vào
     HienString  m3         ; Hiện xâu m3
     mov    CX,AX           ; Đưa số n vào CX (chỉ số vòng lặp)
     mov    AX,1            ; AX=1 (tích lúc đầu bằng 1)
     cmp    CX,2            ; So sánh số n với 2 (trường hợp n=0 và  n=1)
     jb     L2              ; Nhảy nếu n dưới 2 (n=0 và n=1)
L1:  mul    CX              ; AX*CX      DX:AX (song DX=0)
     loop   L1              ; CX=CX-1. Nếu CX≠ 0 nhảy về L1 (lặp tiếp)
L2:  call   HIEN_SO_N       ; Hiện kết quả n!
     HienString  m4         ; Hiện xâu m4
     mov    AH,1            ; Chờ 1 ký tự từ bàn phím
     int    21h
     cmp    AL,'c'          ; So sánh ký tự vào với ký tự tự vào là 'c' (Có)
     jne    Exit            ; Nếu không tiếp tục thì nhảy về Exit
     jmp    PS              ; còn tiếp tục chương trình, quay về PS
```

*

```
Exit:
        mov     AH,4Ch                  ; Về DOS
        int     21h
 INCLUDE   c:\include\lib2.asm    ; Các chương trình con
        END     PS
```

Để được tệp dạng .COM:

```
INCLUDE c:\include\lib1.asm    ; Khai báo 2 MACRO (clrscr và HienString)
.MODEL tiny
.CODE
        ORG   100h
 PS: jmp Start
  m1  db  13,10,'Hay vao n: $'
  m2  db  13,10,'Giai thua cua $'
  m3  db  ' la : $'
  m4  db  13,10,'Co tiep tuc CT (c/k)?$'
 Start:
                  *
 Exit:
       int      20h                     ; Về DOS
 INCLUDE   c:\include\lib2.asm ; Các chương trình con
        END     PS
```

Chú ý: Khi liên kết nhớ tùy chọn /t (tlink/t tên tệp).

Bài tập 2: Hãy viết chương trình tính a^n (a là số nguyên; n là số nguyên dương). Khi chạy ví dụ có dạng:

```
Hay vao a: -4
Hay vao n: 3
-4 luy thua 3 la: -64
Co tiep tuc CT(c/k)?_
```

Cách giải: a^n có nghĩa giá trị của a được nhân n lần. Do vậy thiết lập vòng lặp n lần việc nhân giá trị của a.

Để được tệp dạng .EXE:

```
INCLUDE c:\include\lib1.asm   ; Khai báo 2 MACRO (clrscr và HienString)
_stack  segment
  db  100h dup(?)             ; Xác lập vùng nhớ 256 byte cho STACK
_stack   ends
data      segment             ; Xác lập vùng nhớ RAM để cấp phát cho biến
  m1 db  13,10,'Hay vao a: $'
  m2 db  13,10,'Hay vao n: $'
  xh  db  13,10,'$'
  m3 db  '  luy thua $'
  m4 db  ' la : $'
  m5 db  13,10,'Co tiep tuc CT (c/k)?$'
data      ends
```

```
code      segment
   assume CS:code,DS:data,SS:_stack
  PS:
        mov    AX,data          ; Đưa phần địa chỉ segment của vùng nhớ
        mov    DS,AX            ; cấp phát cho biến vào DS
        clrscr                  ; Xóa màn hình
        HienString  m1          ; Hiện xâu m1
        call    VAO_SO_N        ; Nhận giá trị a
        mov    BX,AX            ; BX=a
        HienString  m2          ; Hiện xâu m2
        call    VAO_SO_N        ; Nhận giá trị n
        mov    CX,AX            ; CX=n
        HienString  xh          ; Đưa con trỏ xuống hàng và về đầu dòng
        mov    AX,BX            ; AX=a (vì BX chứa giá trị của a)
        call    HIEN_SO_N       ; Hiện giá trị của a
        HienString  m3          ; Hiện xâu m3
        mov    AX,CX            ; AX=n (vì CX chứa giá trị n)
        call    HIEN_SO_N       ; Hiện giá trị của n
        HienString  m4          ; Hiện xâu m4
        mov    AX,1             ; AX=1 (AX chứa tích, lúc đầu bằng 1)
```

```
        and     CX,CX                   ; Dựng cờ Z (liệu n=0?)
        jz      L2                      ; Nếu n=0 thì nhảy đến L2
L1:                                     ; còn n≠ 0 thì tiến hành vòng lặp
        mul     BX                      ; AX*BX=DX:AX (song DX=0)
        loop    L1                      ; CX=CX-1? 0
L2:
        call    HIEN_SO_N               ; Hiện kết quả a^n
        HienString  m5                  ; Hiện xâu m5
        mov     AH,1                    ;  Chờ 1 ký tự từ bàn phím
        int     21h
        cmp     AL,'c'                  ; Liệu AL='c' (tiếp tục chương trình)
        jne     Exit                    ; Không tiếp tục chương trình
        jmp     PS                      ; Tiếp tục chương trình (về lại PS)
Exit:
        mov     AH,4Ch                  ; Về DOS
        int     21h
INCLUDE c:\include\lib2.asm             ; Chứa 2 chương trình con
code    ends
END     PS
```

Để được dạng .COM:

```
INCLUDE c:\include\lib1.asm    ; Khai báo 2 MACRO (clrscr và HienString)
code       segment
 assume CS:code,DS:code,SS:code
         ORG  100h
PS: jmp Start
  m1 db  13,10,'Hay vao a: $'     ; Vùng nhớ dành cho khai báo biến
  m2 db  13,10,'Hay vao n: $'
  xh  db  13,10,'$'
  m3 db  '  luy thua $'
  m4 db  ' la : $'
  m5 db  13,10,'Co tiep tuc CT (c/k)?$'
Start:
                *
Exit:
        int       20h                    ; Về DOS
INCLUDE c:\include\lib2.asm    ; Chứa 2 chương trình con
code         ends
END          PS
```

Bài tập 3: Hãy viết chương trình tính tổng 1 dãy số nguyên. Khi chương trình chạy yêu cầu ví dụ có dạng:

```
Hay vao so luong thanh phan cua day so: 4
  a[0]= -100
  a[1]=20
  a[2]=-15
  a[3]=150
Day so vua vao la: -100  20  -15  150
Tong day la: 55
Co tiep tuc CT(c/k)?_
```

Chương trình có dạng:

```
INCLUDE c:\include\lib1.asm    ; Khai báo 2 MACRO (clrscr và HienString)
.MODEL  small
.STACK  100h                   ; Xác lập vùng nhớ 256 byte cho STACK
.DATA
  m1 db  13,10,'Hay vao so luong thanh phan: $'
  m2 db  13,10,'    a[$'
  m3 db  ' ]= $'
  m4 db 13,10,'    Day so vua vao la: $'
  space  db '   $'
  m5 db  13,10,'   Tong day la: $'
  m6 db  13,10,'Co tiep tuc CT (c/k)?$'
```

```
 sltp   dw    ?
 i      dw    ?
 a      dw    100 dup(?)          ; Khai báo trường số
.CODE
 PS:
      mov    AX,@data            ; Đưa phần địa chỉ segment của vùng nhớ
      mov    DS,AX               ; cấp phát cho biến vào DS
      clrscr                     ; Xóa màn hình
      HienString  m1             ; Hiện xâu m1
      call   VAO_SO_N            ; Vào số lượng thành phần
      mov    sltp,AX             ; Đưa số lượng thành phần vào biến sltp
      mov    CX,AX               ; CX=sltp (chỉ số vòng lặp)
      lea    BX,a                ; BX con trỏ offset  đến ô nhớ chưa a[0]
      mov    i,0                 ; Chỉ số (lúc đầu là 0)
; Vòng lặp nhận các số của dãy số đưa vào mảng a
 L1: HienString  m2              ; Hiện xâu m2  (‘a[‘)
      mov    AX,i                ; Đưa chỉ số vào AX
      call   HIEN_SO_N           ;Hiện chỉ số của mảng
      HienString  m3             ; Hiên xâu m3  (‘]=‘)
      call   VAO_SO_N            ; Vào giá trị các thành phần của dãy số
      mov    [BX],AX             ; Đưa vào mảng
```

```
    inc     i                   ; Tăng chỉ số
    add     BX,2                ; BX trỏ đến thành phần tiếp theo của mảng
    loop    L1
    HienString  m4              ; Hiện xâu m4
    mov     CX,sltp             ; CX=sltp (chỉ số vòng lặp)
    lea     BX,a                ; BX con trỏ offset  đến ô nhớ chưa a[0]
; Vòng lặp hiện các số của mảng lên màn hình
L2: mov     AX,[BX]             ; Đưa a[i] vào AX
    call    HIEN_SO_N           ;Hiện a[i]
    HienString  space           ; Hiện các dấu cách
    add     BX,2                ; BX trỏ đến thành phần tiếp theo của mảng
    loop    L2
    HienString  m5              ; Hiện xâu m5
    mov     CX,sltp             ; CX=sltp (chỉ số vòng lặp)
    lea     BX,a                ; BX con trỏ offset  đến ô nhớ chưa a[0]
    xor     AX,AX               ; AX=tổng dãy (lúc đầu bằng 0)
; Vòng lặp tính tổng dãy
L3: add     AX,[BX]
    add     BX,2                ; BX trỏ đến thành phần tiếp theo của mảng
    loop    L3
```

```
    call    HIEN_SO_N          ; Hiện kết quả
    HienString  m6             ; Hiện xâu m6
    mov     AH,1               ;  Chờ 1 ký tự từ bàn phím
    int     21h
    cmp     AL,'c'             ; Liệu AL='c' (tiếp tục chương trình)
    jne     Exit               ; Không tiếp tục chương trình
    jmp     PS                 ; Tiếp tục chương trình (về lại PS)
  Exit:
    mov     AH,4Ch             ; Về DOS
    int     21h
INCLUDE c:\include\lib2.asm    ; Chứa 2 chương trình con
END         PS
```

Chú ý: Thay vì khai báo 2 biến sltp và i (các biến 2 byte), có thể dùng thanh ghi, ví dụ:

sltp … thay bằng thanh ghi SI,

i … thay bằng thanh ghi DI,

còn mảng a thì bắt buộc phải khai báo biến.

1.11 Chương trình đa tệp thuần túy Assembly

Một chương trình viết trên nhiều tệp có những thuận lợi sau:

- Cho phép nhiều người cùng tham gia viết một chương trình lớn,
- Sửa module (tệp) nào, chỉ cần dịch lại module đó,
- Mỗi module thường giải quyết một vấn đề nên dễ hiểu, dễ tìm sai.

Vấn đề nảy sinh cần giải quyết: làm sao các nhãn (tên biến, tên chương trình con, tên hằng, tên cấu trúc,…) dùng chung phải hiểu nhau. Để giải quyết vấn đề này, chương trình dịch Assembler có trang bị 2 directive, đó là :PUBLIC và EXTRN.

1.11.1 Directive PUBLIC

Chức năng: Báo cho chương trình dịch Assembler biết nhãn nào nằm trong module (tệp) này đã được xác lập và cho phép các module (tệp) khác dùng mà không cần xác lập lại.

Cú pháp: PUBLIC tên nhãn

xác lập nhãn

• Với nhãn là tên biến nhớ:	• Với nhãn là tên chương trình con
.DATA **PUBLIC tên biến** ***Khai báo biến***	**.CODE** **PUBLIC tên chương trinh con** ***Chương trình con***

1.11.2 Directive EXTRN

Chức năng: Báo cho chương trình dịch Assembler biết module này xin phép được dùng các nhãn đã được xác lập và cho phép bằng khai báo PUBLIC ở các module khác.

Cú pháp: EXTRN tên nhãn:kiểu

trong đó kiểu có các dạng hay dùng sau:

BYTE … biến 1 byte ; WORD … biến 2 byte

DWORD… biến 4 byte ; PROC … nhãn là tên chương trình con

Bài tập: Viết chương trình n! (0≤n≤7) gồm 2 tệp sau:

– Tệp chương trình chính (lấy tên main.asm) có các nhiệm vụ sau:
  * Nhận n từ bàn phím,
  * Gọi hàm tính n! (ở tệp sub.asm) và hiện kết quả lên màn hình.
– Tệp chương trình con (sub.asm) : Viết hàm tính n!.

Những nhãn dùng chung (cả 2 tệp đều dùng), đó là: biến n, tên chương trình con (GIAITHUA) và kết quả của n! (FV).

Tệp chương trình chính (main.asm):

```
INCLUDE c:\include\lib1.asm    ; Khai báo 2 MACRO (clrscr và HienString)
.MODEL  small
.STACK  100h                   ; Xác lập vùng nhớ 256 byte cho STACK
.DATA
  gt1 db  13,10,'Hay vao n: $'
  gt2 db  13,10,' Giai thua cua $'
  gt3 db  ' la: $'
  gt4 db  13,10,'Co tiep tuc CT (c/k)?$'
  PUBLIC n
   n   dw  ?
  EXTRN  FV: word
.CODE
      EXTRN  GIAITHUA:PROC      ; Xin phép dùng chương trình con
 PS:
      mov    AX,@data           ; Đưa phần địa chỉ segment của vùng nhớ
      mov    DS,AX              ; cấp phát cho biến vào DS
      clrscr                    ; Xóa màn hình
      HienString  gt1           ; Hiện xâu gt1
```

```
    call    VAO_SO_N            ; Vào số n (số vừa vào để ở AX)
    HienString  gt2             ; Hiện xâu gt2
    call    HIEN_SO_N           ; Hiện số n
    HienString  gt3             ; Hiện xâu gt3
    mov     n,AX                ; Đưa giá trị n (AX) vào biến n
    call    GIAITHUA            ; Gọi hàm tính n! (ở tệp sub.asm)
    mov     AX,FV               ; Đưa kết quả từ FV vào AX
    call    HIEN_SO_N           ; Hiện kết quả lên màn hình
    HienString  gt4             ; Hiện xâu gt4
    mov     AH,1                ;  Chờ 1 ký tự từ bàn phím
    int     21h
    cmp     AL,'c'              ; Liệu AL='c' (tiếp tục chương trình)
    jne     Exit                ; Không tiếp tục chương trình nhảy đến Exit
    jmp     PS                  ; còn tiếp tục chương trình thì trở về PS
Exit:
    mov     AH,4Ch              ; Về DOS
    int     21h
INCLUDE c:\include\lib2.asm     ; Chứa 2 chương trình con
END         PS
```

Module chương trình con (sub.asm)

```
.MODEL  small
.DATA
  PUBLIC FV
   FV    dw    ?
  EXTRN n: word
  Fac   dw    ?
.CODE
  PUBLIC  GIAITHUA          ; Cho phép tệp khác dùng chương trình con
 GIAITHUA PROC
        mov    CX,n         ; Đưa giá trị n vào CX (chỉ số vòng lặp)
        mov    FV,1         ; FV=1
        mov    Fac,2        ; Fac=2
        cmp    CX,2         ; So sánh giá trị n (CX) với 2
        jb     L2           ; Nếu n<2 (n=0 hoặc 1) thì nhảy đến L2
        dec    CX           ; Giảm CX đi 1
  L1: mov    AX,FV          ; AX=FV
        mul    Fac          ; AX*Fac -> DX:AX (trường hợp DX=0)
        mov    FV,AX        ; FV=AX
        inc    Fac          ; Tăng Fac lên 1
        loop   L1           ; CX=CX-1 ? 0
```

```
L2:
    ret
GIAITHUA  ENDP
END
```

1.11.3 Cách dịch và liên kết chương trình đa tệp

Bước 1: Dịch từng tệp nguồn có phần mở rộng .asm sang tệp có phần mở rộng .obj. Chú ý là tệp nào có lỗi chỉ sửa tệp đó và dịch lại tệp đó.

Với ví dụ trên sẽ tiến hành như sau:

tasm main để được main.obj (nếu không có lỗi)

tasm sub để được sub.obj (nếu không có lỗi)

Bước 2: Gộp các tệp có phần mở rộng .obj thành một tệp có phần mở rộng .exe với tên tệp là tên tệp đứng đầu (thường là tên tệp chứa chương trình chính).

Với ví dụ trên sẽ như sau:

tlink main + sub để được main.exe

Tổng kết bài học:

Như vậy là chúng ta đã đề cập đến các vấn đề tiếp theo của ngôn ngữ Assembly, mà cụ thể là:

- Vấn đề chương trình con thông qua việc trình bày ý nghĩa, cơ chế khi một chương trình con được gọi và cú pháp của chương trình con Assembly cùng các ví dụ minh họa,
- Vấn đề MACRO thông qua việc trình bày ý nghĩa, khai báo MACRO cùng các ví dụ minh họa và cách dùng MACRO đã được xác lập,
- Vấn đề directive INCLUDE để chèn nội dung của một tệp ngoài vào chương trình đang viết và cơ chế tiến hành,
- Cách viết một chương trình Assembly để sau khi dịch và liên kết nhận được tệp thực hiện với đuôi .COM,
- Tổng kết phần lập trình với ngôn ngữ Assembly bằng cú pháp thường thấy của một chương trình Assembly cùng các bài tập ví dụ, và
- Cuối cùng đề cập đến chương trình đa tệp thuần túy Assembly.

Để kiểm tra kiến thức của mình, đề nghị các bạn trả lời các câu hỏi trắc nghiệm và làm các bài tập.