

## Chương 1. NGÔN NGỮ ASM VÀ CÁCH LẬP TRÌNH (25 tiết)

### 1.1. Mở đầu

#### Giới thiệu

Ngôn ngữ Assembler là ngôn ngữ bậc thấp.

➤ **Ưu điểm :**

Vì ngôn ngữ Assembler rất gần gũi với ngôn ngữ máy nên chương trình

- + Chạy nhanh.
- + Tiết kiệm bộ nhớ.
- + Có thể lập trình truy cập qua các giao diện vào ra nhưng hiện nay các ngôn ngữ bậc cao cũng có thể làm được.

➤ **Nhược điểm**

- + Khó viết bởi vì yêu cầu người lập trình rất am hiểu về phần cứng.
- + Khó tìm sai: bao gồm sai về cú pháp (syntax) và sai về thuật toán (Algorithm). Chương trình dịch sẽ thông báo sai ta sẽ dùng debug của DOS để kiểm tra.
- + Không chuyển chương trình Assembler cho các máy tính có cấu trúc khác nhau.

➤ **Ứng dụng**

- + Viết lỗi của hệ điều hành.
- + Các chương trình trò chơi ( ngày trước).
- + Tạo virus.
- + Các chương trình đo và điều khiển sử dụng trong công nghiệp, ngày nay các vi điều khiển được sử dụng một cách rộng rãi.

### 1.2. Cài đặt chương trình dịch TASM

Hiện nay có hai chương trình dịch rất phổ biến là MASM (của hãng Microsoft) và TASM (của hãng Borland) về cơ bản là hai chương dịch này rất giống nhau nhưng khác nhau ở chỗ: khi viết lệnh push

Nếu viết :

```
push ax
push bx
push cx
```

thì cả hai chương trình đều biên dịch được. ( cách viết này theo MASM).

Còn trong TASM thì cho phép viết

```
push ax bx cx
```

#### Cài đặt chương trình dịch TASM:

➤ **Cách 1:**

Mua đĩa bản quyền nếu là đĩa mềm thì có 5 đĩa hoặc là 1 đĩa CD

Run → cmd

A:\install



vì vậy làm cho chương trình chạy nhanh hơn.

- + Giải thích: vì các thanh ghi nằm ở CPU nên dữ liệu lấy ra nhanh hơn.
- + Vùng nhớ cache là vùng nhớ nằm trong CPU.

*b) Phân loại thanh ghi*

- + Máy tính 16 bit có 14 thanh ghi.
- + Máy tính 32 bit có 16 thanh ghi.

➤ **Cấu trúc thanh ghi của máy tính 16 bit**

- + *Nhóm 1: Thanh ghi cờ*

Người lập trình ASM hay dùng trạng thái các bit cờ làm điều kiện cho các lệnh nhảy có điều kiện.

x	x	x	x	O	D	I	T	S	Z	x	A	x	P	x	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- + x: không được định nghĩa.

6 bit cờ trạng thái thể hiện các trạng thái khác nhau của kết quả sau một thao tác nào đó, trong đó 5 bit cờ đầu thuộc byte thấp của thanh cờ là các cờ giống như của bộ vi xử lý 8 bit 8085 của Intel.

- + C hoặc CF (Carry flag): cờ nhớ. CF = 1 khi có nhớ hoặc mượn từ MSB.
- + P hoặc PF (Parity flag): cờ parity. PF phản ánh tính chẵn lẻ (parity) của tổng số bit có trong kết quả. PF = 1 khi tổng số bit 1 trong kết quả là chẵn.
- + A hoặc AF (Auxiliary carry flag): cờ nhớ phụ, rất có ý nghĩa khi ta làm việc với các số BCD. AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao).
- + Z hoặc ZF (Zero flag): cờ rỗng, ZF = 1 khi kết quả bằng 0.
- + S hoặc SF (Sign flag): cờ dấu, SF = 1 khi kết quả âm.
- + O hoặc OF (Overflow flag): cờ tràn, OF = 1 khi kết quả là một số bù hai vượt ra ngoài giới hạn biểu diễn dành cho nó.

Ngoài ra bộ vi xử lý 8088 còn có các cờ điều khiển sau đây:

- + T hoặc TF (Trap flag): cờ bẫy, TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh( chế độ này cần dùng khi cần tìm lỗi trong một chương trình).
- + I hoặc IF (Interrupt enable flag): cờ cho phép ngắt, IF = 1 thì CPU cho phép các yêu cầu ngắt được tác động.
- + D hoặc DF (Direction flag): cờ hướng, DF = 1 khi CPU làm việc với chuỗi kí tự theo thứ tự từ trái sang phải (hay còn gọi D là cờ lùi).

- + *Nhóm 2: Thanh ghi đa năng: gồm 8 thanh ghi 16 bits.*

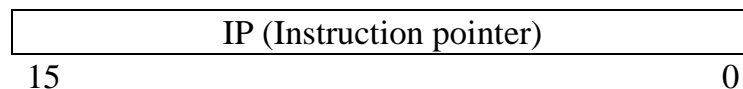
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL
SI		
DI		
BP		
SP		

- + Trong đó H(high) thể hiện các bit cao, L(low) thể hiện các bit thấp.
- + Trong 4 thanh ghi AX, BX, CX và DX có 3 cách truy cập: truy cập theo 8 bit cao hoặc theo 8 bit thấp hoặc theo cả 16 bit. Các thanh ghi còn lại chỉ có một cách truy cập.

- + AX (Accumulator, Acc): thanh chứa. Các kết quả của các thao tác thường được chứa ở đây (kết quả của phép nhân, chia). Nếu kết quả là 8 bit thì thanh ghi AL được coi là Acc.
- + BX (Base): thanh ghi cơ sở, thường chứa địa chỉ cơ sở của một bảng dùng trong lệnh XLAT(XLAT/XLATB -- Table Look-up Translation).
- + CX (Count): bộ đếm, CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP, còn CL thường chứa số lần dịch hoặc quay trong các lệnh dịch hay quay thanh ghi.
- + DX (Data): thanh ghi dữ liệu, DX cùng AX tham gia vào các thao tác của phép nhân hoặc chia các số 16 bit. DX còn dùng để chứa địa chỉ của các cổng trong các lệnh vào ra trực tiếp (IN/OUT).
- + SI (Source index): chỉ số gốc hay nguồn, SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ tương ứng với DS : SI.
- + DI (Destination index): chỉ số đích, DI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ tương ứng với DS : DI.
- + BP (Base pointer) : con trỏ cơ sở, BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS : BP.
- + SP (Stack pointer): con trỏ ngăn xếp, SP luôn trỏ vào đỉnh hiện thời của ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp ứng với SS:SP.

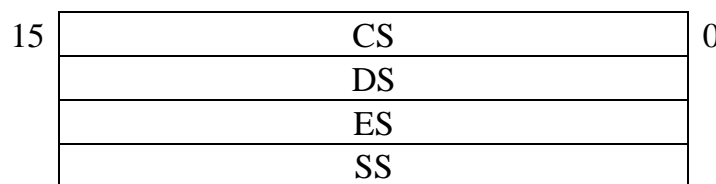
Người lập trình chỉ dùng 7 thanh ghi sau: AX, BX, CX, DX, SI, DI, BP.

- + *Nhóm 3: Thanh ghi con trỏ lệnh IP (Instruction pointer) hay PC(ProgrAM pointer)*



Nội dung trong thanh ghi IP cho biết địa chỉ offset của vùng nhớ chứa mã lệnh.

- + *Nhóm 4: Thanh ghi Segmnet (phân đoạn): 4 thanh ghi 16 bits.*

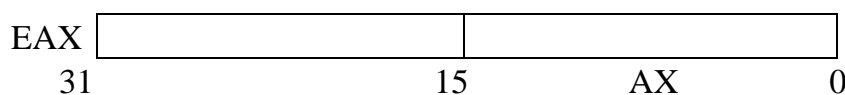


Các thanh ghi segment cho biết địa chỉ segment.

- + CS (Code segment): mã máy.
- + DS, ES: dữ liệu.
- + SS: ngăn xếp

### ➤ Cấu trúc thanh ghi của máy tính 32 bit

- + Nhóm 1+ nhóm 2 + nhóm 3 là các thanh ghi 32 bit và với chữ E ở đầu (ví dụ: EAX hay EBX)



- + Nhóm 4 vẫn là các thanh ghi 16 bit và thêm hai thanh ghi GS và FS

**1.4.2 Cách thể hiện địa chỉ ô nhớ (ROM hoặc RAM): dạng logic và dạng vật lý**

Một thanh ghi 16 bit thì trở được 64k nhưng vùng nhớ của máy tính hiện nay rất lớn do vậy phải dùng 2 thanh ghi để thể hiện địa chỉ của một ô nhớ. Và vùng nhớ được chia thành nhiều phần, mỗi phần 64k.

**a) Dạng Logic**

Địa chỉ 1 ô nhớ = segment : offset

- + Thanh ghi thứ nhất cho biết ô nhớ đó nằm ở 64k thứ mấy (địa chỉ segment).
- + Thanh ghi thứ hai cho biết khoảng cách từ đầu segment đến vị trí ô nhớ đó (địa chỉ offset).

Ví dụ: 2: 100 tức là địa chỉ của ô nhớ nằm ở vị trí 100 tính từ trên đỉnh của segment thứ hai.

**b) Dạng vật lý**

Địa chỉ ô nhớ = seg\*16 + offset

- + Cách đánh địa chỉ này hay được dùng.

**1.4.3 Các ngắt hay dùng hỗ trợ cho lập trình Assembler**

- + Hàm 1: Chờ 1 kí tự( từ bàn phím)
 

```
mov     ah,1           ; gán ah = 1 al chứa mã ASCII
                        ; al = 0 khi kí tự gõ vào là các phím chức năng( lệnh).
int      21h
```

- + Hàm 2: Hiện 1 ký tự lên màn hình từ vị trí con trỏ đang đứng.

Cách 1:

```
mov     al, mã ASCII
mov     ah,0ch
int      10h
```

Cách 2:

```
mov     dl, mã ASCII
        ; dl = mã ASCII của kí tự cần hiển thị
mov     ah,2
int      21h
```

- + Hàm 3: Hiện xâu kí tự kết thúc '\$' lên màn hình.

```
lea dx, tên biến xâu
                        ; mov dx,offset tên biến xâu
```

```
mov     ah,9
int      21h
```

- + Hàm 4: Trở về DOS

```
mov     ah,4ch
int      21h
```

**1.5. Hệ lệnh Assembler**

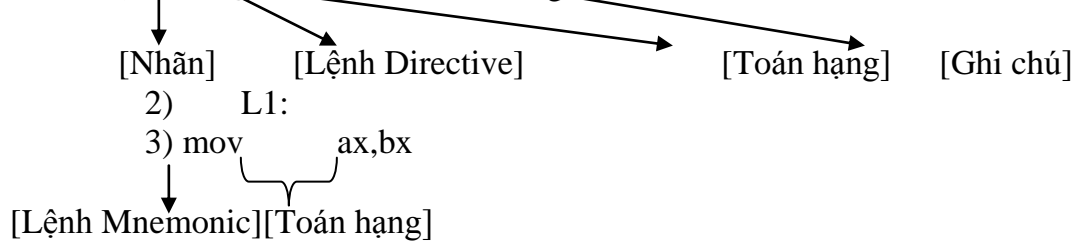
- + Tập lệnh MNEMONIC sinh mã máy để chạy chương trình.
- + Các DIRECTIVE điều khiển khi dịch chương trình.

**1.5.1. Cú pháp của một dòng lệnh ASM**

- + Mỗi một dòng chỉ được viết một lệnh.
- + [Label]            [Directive/Mnemonic]        [Operands]    [;Commnet]
- [Nhãn]            [Loại lệnh]                    [Toán hạng]    [Ghi chú]

Từ ; cho đến hết dòng là ghi chú và nó có hiệu lực chỉ trên 1 dòng.

Ví dụ: 1) X1 EQU 10 ; gán 10 cho X1



### 1.5.2. Tập lệnh Mnemonic

- Tập lệnh Mnemonic là gì? Đó là lệnh của ASM được thể hiện bằng viết tắt của tiếng Anh cho dễ hiểu.

Tiếng Anh	Lệnh dạng Mnemonic
Move	mov
Addition	add
Multiplication	mul

- Các quy ước về toán hạng

- + SRC: Toán hạng nguồn.
- + DST: Toán hạng đích.
- + REG(reg8/reg16): Toán hạng là thanh ghi
- + Data: Toán hạng là hằng số.
- + Mem: Toán hạng là biến nhớ.
- + Segreg: Toán hạng là thanh ghi segment.
- Tập lệnh MNEMONIC gồm có 6 nhóm
- + Nhóm 1: Các lệnh di chuyển dữ liệu
- + Nhóm 2: Các lệnh số học.
- + Nhóm 3: Các lệnh thao tác bit
- + Nhóm 4: Các lệnh thao tác xâu ký tự.
- + Nhóm 5: Các lệnh rẽ nhánh
- + Nhóm 6: Các hệ thống cờ

#### a) Nhóm 1: Các lệnh di chuyển dữ liệu

Tất cả lệnh trong nhóm này khi thực hiện không làm thay đổi trạng thái của các bit cờ.

##### - Lệnh mov

*Chức năng:* Đưa nội dung từ SRC đến DST

*Cú pháp:* mov DST, SRC

reg1	reg2	→	mov	ax, bx
reg	data	→	mov	cx, 100
reg	mem	→	mov	dx, value
mem	reg	→	mov	value, dx
mem	data	→	mov	value, 100
segreg	reg16	→	mov	ds, ax
reg16	segreg	→	mov	bx, cs
segreg	mem16	→	mov	cs, value
mem16	segreg	→	mov	value, cs

Chú ý:

- + Không di chuyển được giữa hai biến nhớ (mov mem1, mem2).

Thực hiện gián tiếp:

```
mov reg,mem2
```

```
mov mem1,reg
```

- + Không đưa trực tiếp dữ liệu vào thanh ghi segment (mov seg,data).

Thực hiện gián tiếp:

```
mov reg16,data
```

```
mov segreg,reg16
```

- + Sự khác nhau khi sử dụng các chế độ địa chỉ

(mov ax,bx khác với mov ax,[bx] ; đưa nội dung mà bx trỏ đến vào ax)

```
mov ax,[bx] tương đương với
```

```
mov ax,ds:[bx] (SI,DI hay BP)
```

#### - **Lệnh push**

*Chức năng:* cất 2 byte của SRC vào đỉnh ngăn xếp(stack).

*Cú pháp:*

```
PUSH SRC
```

```
reg16
```

```
mem16
```

Ví dụ: push ax

Toán hạng gốc có thể tìm được theo các chế độ địa chỉ khác nhau: có thể là thanh ghi đa năng, thanh ghi đoạn hay là ô nhớ. Lệnh này thường được dùng với lệnh POP như là một cặp đối ngẫu để xử lý các dữ liệu và trạng thái của chương trình chính(CTC) khi vào ra chương trình con(ctc).

#### - **Lệnh POP**

*Chức năng:* lấy 2 byte (1 từ) ở đỉnh ngăn xếp (stack) vào toán hạng đích.

*Cú pháp:*

```
POP DST
```

```
reg16
```

```
mem16
```

Ví dụ:

```
push ax
```

```
push bx
```

```
push cx
```

Đoạn Chương trình

```
pop cx
```

```
pop bx
```

```
pop ax
```

*Chú ý:* - Cơ chế PUSH/POP là LIPO( last in first out)

- Cách viết trên chỉ được sử dụng trong MASM còn trong TASM được viết như sau:

```
push ax bx cx
```

#### - **Lệnh PUSHF**

*Chức năng:* cất giá trị thanh ghi cờ vào đỉnh ngăn xếp

*Cú pháp:* PUSHF

Dữ liệu tại ngăn xếp không thay đổi, SS không thay đổi.

#### - **Lệnh POPF**

*Chức năng:* Lấy 2 byte từ đỉnh ngăn xếp rồi đưa vào thanh ghi cờ.

*Cú pháp:* POPF

Sau lệnh này dữ liệu tại ngăn xếp không thay đổi, SS không thay đổi.

- **Lệnh XCHG (Exchange 2 Operands) Trao nội dung 2 toán hạng**

*Chức năng:* Trao nội dung 2 toán hạng DST  $\longleftrightarrow$  SRC

*Cú pháp*

XCHG     DST     SRC  
             reg1     reg2  
             reg     mem

Trong toán hạng đích có thể tìm theo chế độ địa chỉ khác nhau nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là 2 ô nhớ và cũng không được là thanh ghi đoạn. Sau lệnh XCHG toán hạng chứa nội dung cũ của toán hạng kia và ngược lại.

Lệnh này không tác động đến cờ.

- **Lệnh IN**

*Chức năng:* đọc dữ liệu từ cổng vào thanh ghi AL/AX

*Cú pháp:* IN AL/AX, địa chỉ cổng

*Chú ý:*

+ Nếu địa chỉ cổng <256 thì số địa chỉ đứng trực tiếp trong lệnh IN

Ví dụ: địa chỉ cổng là 1fh

IN AL,1fh ; nội dung cổng 1fh đưa vào AL.

+ Nếu địa chỉ cổng  $\geq 256$  thì phải nhờ đến thanh ghi DX

Ví dụ: địa chỉ COM1 = 378h

mov dx,378h

in al,dx

- **Lệnh OUT**

*Chức năng:* đưa dữ liệu từ thanh ghi AL/AX ra cổng

*Cú pháp:* OUT địa chỉ cổng,AL/AX

*Chú ý:*

+ Nếu địa chỉ cổng <256 thì số địa chỉ đứng trực tiếp trong lệnh OUT

Ví dụ: địa chỉ cổng là 1fh

OUT 1fh,AL ; đưa nội dung AL ra cổng 1fh.

+ Nếu địa chỉ cổng  $\geq 256$  thì phải nhờ đến thanh ghi DX

Ví dụ: địa chỉ COM1 = 378h

mov dx,378h

out dx,al

Lệnh này không tác động đến cờ.

- **Lệnh LEA (load Effective address)**

*Chức năng:* lấy phần địa chỉ offset của biến đưa vào thanh ghi 16 bit

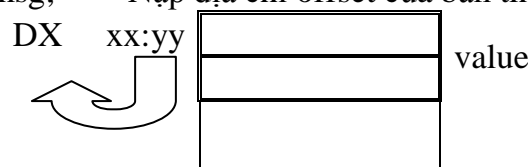
*Cú pháp:* lea                     reg16, mem

Ví dụ: lea     bx, Value hay mov bx, OFFSET Value

Đích thường là các thanh ghi: BX, CX, DX, BP, SI, DI.

Nguồn là tên biến trong đoạn DS được chỉ rõ trong lệnh hay ô nhớ cụ thể.

Ví dụ: lea     dx, msg;     Nạp địa chỉ offset của bản tin msg vào dx.



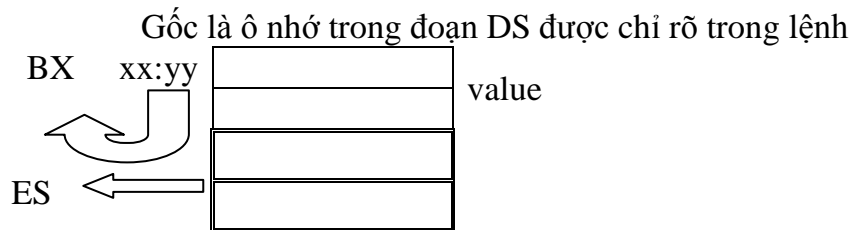


**- Lệnh LES (Load register and ES with words from memory)**

*Chức năng:* chuyển giá trị của 1 từ(word) từ một vùng nhớ vào thanh ghi đích và giá trị của từ tiếp theo sau của vùng nhớ vào thanh ghi ES.

*Cú pháp:* les reg, mem

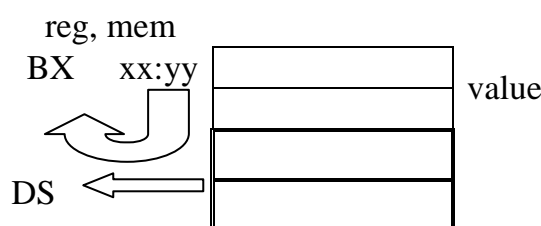
Trong đó: Đích là một trong các thanh ghi AX, BX, CX, DX, SP, BP, SI, DI.



**- Lệnh LDS (Load register and DS with words from memory)**

*Chức năng:* Nạp một từ từ bộ nhớ vào thanh ghi cho trong lệnh và 1 từ tiếp theo vào DS.

*Cú pháp:* lds



**b) Nhóm 2: Các lệnh số học**

**b1) Số có dấu và số không dấu**

- Số không dấu: Nếu nhìn vào toán hạng (độ lớn các toán hạng là 1 byte hay là 2 byte) với số không dấu thì bit cao nhất mang giá trị tại vị trí đó.

Ví dụ:

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

 255

- Số có dấu: Nếu nhìn vào toán hạng của số có dấu thì bit cao nhất sẽ mang ý nghĩa về dấu: 1 toán hạng là số âm, 0 toán hạng là số dương.

1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

↑ Nếu số không dấu thì là  $1.2^7 = 128$

Nếu số có dấu thì là - 1

**b2) Cách thể hiện một số âm của máy tính**

Máy tính thể hiện số âm bằng cách bù 2 giá trị tuyệt đối của số đó.

Ví dụ: mov ax, - 1

1 = 0000 0000 0000 0001

bù 1: 1111 1111 1111 1110

+

1

bù 2: 1111 1111 1111 1111

mov ax, - 100

100 = 0000 0000 0110 0100

1111 1111 1001 1011

+

1

bù 2: 1111 1111 1001 1100

Hầu hết các lệnh trong nhóm này khi thực hiện có thể làm thay đổi các kí tự.

**- Lệnh ADD(addition)**

**Chức năng:**  $DST \leftarrow DST + SRC$

Cộng hai toán hạng: lấy toán hạng đích cộng với toán hạng nguồn rồi đưa vào toán hạng đích.

**Cú pháp:**

add DST, SRC	
reg1	reg2 $\longrightarrow$ add ax, bx
reg	data $\longrightarrow$ add cx, 100
reg	mem $\longrightarrow$ add dx, value
mem	reg $\longrightarrow$ add value, dx
mem	data $\longrightarrow$ add value, 100

Tác động đến cờ: C, P, A, Z, S, O.

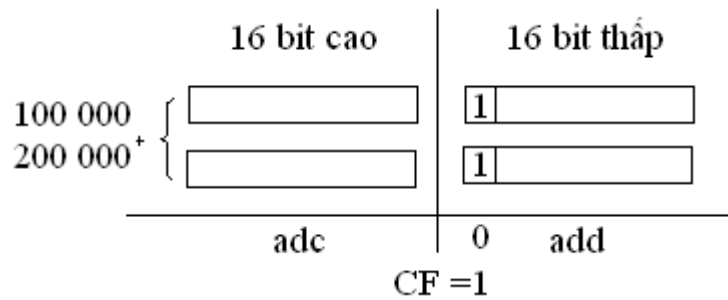
- **Lệnh ADC(Add with carry)**

**Chức năng:** cộng có nhớ,  $DST \leftarrow DST + SRC + CF$

**Cú pháp:** adc DST, SRC

Tác động đến cờ: C, P, A, Z, S, O.

Ví dụ: adc ax, bx



- **Lệnh INC(Increment Destination Register or Memory)**

**Chức năng:** Tăng toán hạng đích thêm 1.  $DST \leftarrow DST + 1$

**Cú pháp:** inc DST

Tác động đến cờ: C, P, Z, S, O.

Ví dụ: reg  $\longrightarrow$  inc ax  
~~mem~~ inc value

- **Lệnh SUB (Substraction)**

**Chức năng:** Trừ hai toán hạng,  $DST \leftarrow DST - SRC$

**Cú pháp:** sub DST, SRC

Ví dụ: sub ax, bx

Tác động đến cờ: C, P, A, Z, S, O.

Chú ý: chế độ địa chỉ không được đồng thời là 2 ô nhớ hay là thanh ghi đoạn.

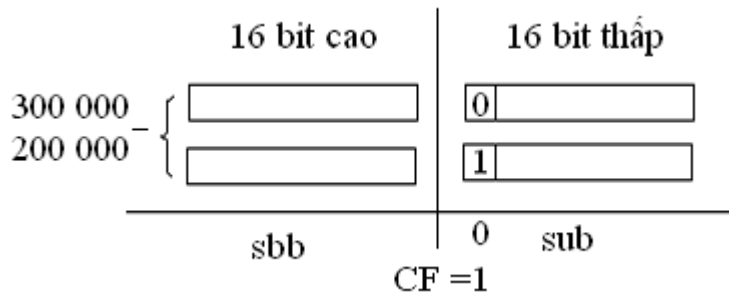
- **Lệnh SBB (Substraction with borrow)**

**Chức năng:** Trừ có mượn,  $DST \leftarrow DST - SRC - CF$

**Cú pháp:** sbb DST, SRC

Ví dụ: sbb ax, bx

Tác động đến cờ: C, P, A, Z, S, O.



- **Lệnh MUL/ IMUL (Multiply Unsigned Byte or Word/ Integer Multiplication)**

*Chức năng:* Nhân 2 toán hạng với số không dấu (MUL), số có dấu (IMUL)

*Cú pháp:* MUL(IMUL) SRC

reg  
mem

Có hai trường hợp tổ chức phép nhân

+ 8 bits \* 8 bits

Số bị nhân phải là số 8 bit để trong AL

Sau khi nhân:  $al * SRC \longrightarrow AX$

+ 16 bits \* 16 bits

Số bị nhân phải là số 16 bit để trong AX

Sau khi nhân:  $ax * SRC \longrightarrow dx:ax$

Tác động đến cờ: C, O.

Chú ý:

$al = 1111$                       1111

$bl = 0000$                       0010

$mul\ bl \longrightarrow ax = al * bl \ (255 * 2 = 510)$

$imul\ bl \longrightarrow ax = al * bl \ (-1 * 2 = -2)$

Trong phép chia thì ax, bx, dx (al,bl,dx) là ẩn

- **Lệnh DIV/IDIV (Unsigned Divide/Integer Division)**

*Chức năng:* Chia hai toán hạng với số không dấu/ số có dấu

*Cú pháp:* DIV (IDIV) SRC

reg  
mem

Hai trường hợp tổ chức phép chia

+ Nếu số 16 bits chia cho số 8 bits

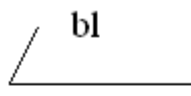
$ax \ / \ SRC$   
 $al = \text{thương} \quad ah = \text{dư}$

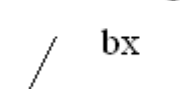
+ Nếu số 32 bits chia cho số 16 bits

$dx:ax \ / \ SRC$   
 $ax = \text{thương} \quad dx = \text{dư}$

Trong phép chia thì ax, bx, dx (al,bl,dx) là ẩn

Ví dụ:

$\text{div bl} \rightarrow \text{ax}$    
 $\text{al} = \text{thương} \quad \text{ah} = \text{dư}$

$\text{div bx} \rightarrow \text{dx:ax}$    
 $\text{ax} = \text{thương} \quad \text{dx} = \text{dư}$

- **Lệnh DEC (Decrement Destination Register or Memory)**

*Chức năng:* Giảm toán hạng đích đi 1,  $\text{DST} \leftarrow \text{DST} - 1$

*Cú pháp:*  $\text{dec DST}$

$\text{reg} \rightarrow \text{dec}$   $\text{ax}$   
 $\text{mem} \rightarrow \text{dec}$   
 value

Tác động đến cờ: C, P, Z, S, O.

- **Lệnh NEG (Negate a Operand)**

*Chức năng:* lấy bù hai của một toán hạng, đảo dấu của một toán hạng

$\leftarrow \text{DST}$   $- \text{DST}$   
*Cú pháp:*  $\text{neg DST}$   
 $\text{reg} \rightarrow \text{neg}$   $\text{ax}$   
 $\text{mem} \rightarrow \text{neg}$   
 value

Tác động đến cờ: C, P, A, Z, S, O.

- **Lệnh CMP (Compare Byte or Word)**

*Chức năng:* So sánh nội dung của hai toán hạng và dựng cờ. Sau khi thực hiện lệnh này nội dung của hai toán hạng không thay đổi.

*Cú pháp:*  $\text{cmp DST, SRC}$

Tác động đến cờ: C, P, Z, S, O.

Cách dựng cờ:

$\text{cmp DST, SRC}$   
 + Nếu  $\text{DST} < \text{SRC}$  thì  $\text{CF} = 1$ .  
 + Nếu  $\text{DST} \geq \text{SRC}$  thì  $\text{CF} = 0$ .  
 + Nếu  $\text{DST} = \text{SRC}$  thì  $\text{ZF} = 1$ .  
 + Nếu  $\text{DST} \neq \text{SRC}$  thì  $\text{ZF} = 0$ .

**c) Nhóm 3: Các lệnh thao tác bit**

Chú ý: tất cả các lệnh trong nhóm này khi thực hiện có thể làm thay đổi trạng thái các bit cờ.

- **Lệnh AND**

*Chức năng:* Thực hiện phép “và logic”, bit của kết quả bằng 1 khi 2 bit tương ứng đều bằng 1.  $\text{DST} \leftarrow \text{DST} \wedge \text{SRC}$

*Ví dụ:*

$\text{al} = 1010 \ 1010$   
 $\text{bl} = 1100 \ 1100$   
 $\text{and al, bl} = 1000 \ 1000$   
*Cú pháp:*  $\text{and DST, SRC}$

*Cách hay dùng:*

+ Tách bit:

al = xxxx xxxx

0001 0000

and al, 10h = 000x 0000

Khi dùng phép AND để che đi/ giữ lại một vài bit nào đó của một toán hạng thì bằng cách nhân logic toán hạng đó với toán hạng tức thì có các bit 0/1 ở các chỗ cần che/ giữ nguyên tương ứng.

+ Dụng cụ and DST, DST

Ví dụ: and ax, ax

Nếu  $ax < 0$  thì  $SF = 1$ .

Nếu  $ax \geq 0$  thì  $SF = 0$ .

Nếu  $ax = 0$  thì  $ZF = 1$ .

Nếu  $ax \neq 0$  thì  $ZF = 0$ .

#### - **Lệnh OR**

*Chức năng:* thực hiện phép hoặc logic, Bit của kết quả = 1 khi 1 trong 2 bit là 1.

DST ← DST V SRC

*Ví dụ:*

al = 1010 1010

bl = 1100 1100

or al, bl = 1110 1110

*Cú pháp:* or DST, SRC

Tác động đến cờ:  $C = O = 0$ , P, Z, S.

#### - **Lệnh XOR**

*Chức năng:* Thực hiện phép “hoặc loại trừ” 2 toán hạng, bit của kết quả bằng 1 khi 2 bit tương ứng khác nhau.

*Ví dụ:*

al = 1010 1010

bl = 1100 1100

xor al, bl = 0110 0110

*Cú pháp:* xor DST, SRC

*Cách hay dùng:*

+ Tách bit:

al = xxxx xxxx

0001 0000

and al, 10h = 000x 0000

Tác động đến cờ:  $C = O = 0$ , P, Z, S.

Ví dụ: Thực hiện  $ax = 0$

1. mov ax, 0 3 byte

2. and ax, 0

3. sub ax, ax 2 byte

4. xor ax, ax

#### - **Lệnh SHL (Shift Left)**

*Chức năng:* dịch trái các bit của toán hạng đích đi một số lần nào đó (số lần dịch được cất trong thanh ghi CL).

*Cú pháp:* SHL DST, CL



Nếu dịch một lần thì ta có thể viết trực tiếp.

VD: sar ax, 1

Nếu số lần dịch  $\geq 2$  thì phải nhờ đến CL/CX

~~sar ax, 4~~  $\equiv$  mov cl/cx, 4  
sar ax, cl/cx

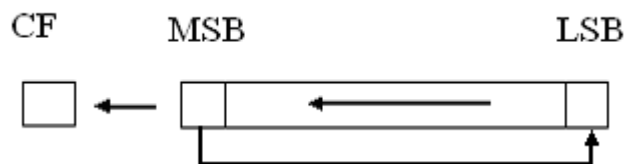
Ý nghĩa: Dịch phải 1 lần là chia đôi và làm tròn dưới với số có dấu.

#### - **Lệnh ROL( Rotate All Bits to the Left)**

Chức năng: quay vòng sang trái các bit của toán hạng đích đi một số lần nào đó (số lần dịch được cất trong thanh ghi CL). Trong mỗi lần quay giá trị bit cao nhất vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bit thấp nhất

Cú pháp: ROL DST, CL

(reg, mem)



Tác động đến cờ: C, O.

Nếu dịch một lần thì ta có thể viết trực tiếp.

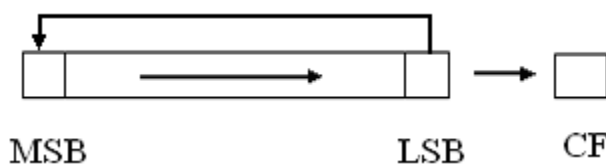
VD: rol ax, 1

#### - **Lệnh ROR**

Chức năng: quay vòng sang phải các bit của toán hạng đích đi một số lần nào đó (số lần dịch được cất trong thanh ghi CL). Trong mỗi lần quay giá trị bit thấp nhất LSB vừa chuyển vào thanh ghi cờ CF đồng thời chuyển vào bit cao nhất MSB.

Cú pháp: ROR DST, CL

(reg, mem)



Tác động đến cờ: C, O.

Nếu dịch một lần thì ta có thể viết trực tiếp.

VD: ror ax, 1

#### d) **Nhóm 4: Các lệnh làm việc với xâu**

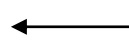
Chú ý: Chỉ có 2 lệnh trong nhóm này khi thực hiện làm thay đổi các bit cờ.

##### - **Lệnh MOVSB/MOVSX (Move String Byte or String Word)**

Chức năng: Chuyển một xâu ký tự theo từng byte(MOVSB) hay theo từng từ (MOVSX) từ vùng nhớ trở bởi DS:SI sang vùng nhớ trở bởi ES:DI. Sau mỗi

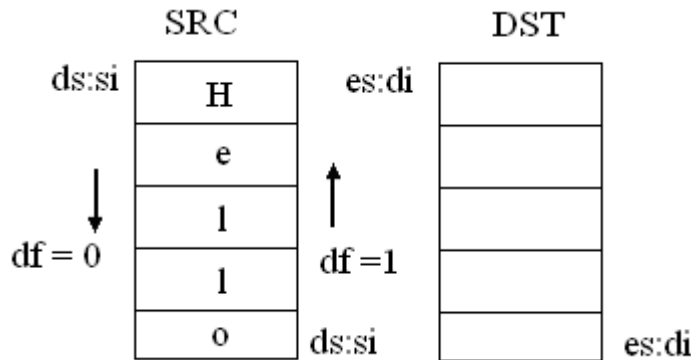
lần dịch chuyển thì giá trị của SI, DI tự động tăng lên 1 hoặc 2 khi cờ hướng DF = 0 hoặc giảm đi 1 hoặc 2 khi DF = 1.

Phần tử của Chuỗi đích



Phần tử của Chuỗi gốc

Cú pháp: MOVSB hoặc MOVSW



Chuẩn bị trước ds:si con trỏ đến đầu xâu SRC, es:di con trỏ đến đầu xâu DST  
Lệnh này không tác động đến cờ.

- **Lệnh LODSB/LODSW (Load String Byte or Word into AL/AX)**

*Chức năng:* Chuyển các kí tự theo từng byte (LODSB) hay theo từng từ (LODSW) từ vùng nhớ trỏ bởi DS:SI vào AL/AX.

*Cú pháp:* LODSB hoặc LODSW

Chuẩn bị trước ds:si con trỏ ở đầu xâu, df = 0 hay df = 1.

Lệnh này không tác động đến cờ.

- **Lệnh STOSB/STOSW (Store AL/AX in String Byte/Word)**

*Chức năng:* Chuyển các kí tự nằm ở AL(STOSB) /AX (STOSW) vào vùng nhớ trỏ bởi ES:DI.

*Cú pháp:* STOSB hoặc STOSW hoặc STOS Chuỗi đích.

Xác lập trước ES:DI trỏ đến đầu vùng nhớ, df = 0 hay df = 1.

Lệnh này không tác động đến cờ.

Nhận xét

1. movsb = lodsb + stosb
2. movsw = lodsw + stosw

- **Lệnh CMPSB/CMPSW**

*Chức năng:* So sánh hai xâu kí tự theo từng byte (CMPSB) / theo từng từ (CMPSW) giữa hai vùng nhớ trỏ bởi DS:SI và ES:DI. Lệnh này chỉ tạo cờ, không lưu lại kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi.

*Cú pháp:* CMPSB hoặc CMPSW hoặc STOS Chuỗi đích.

Xác lập trước DS:SI trỏ đến đầu xâu 1, ES:DI trỏ đến đầu xâu 2, df = 0 hay df = 1.

Tác động đến cờ: ZF = 1 khi hai xâu bằng nhau, ZF = 0 khi hai xâu khác nhau.

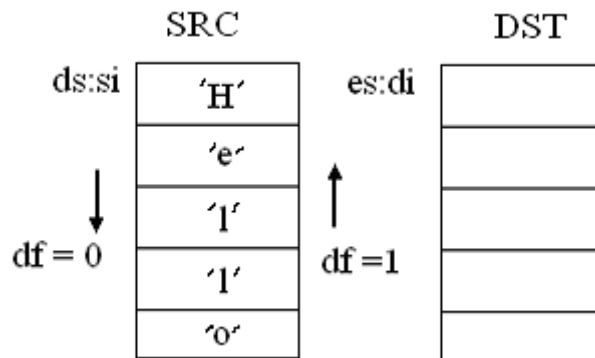
- **Tiên tố REP (Repeat String Instruction until CX = 0).**

*Chức năng:* Lặp đi lặp lại lệnh làm việc với xâu kí tự đằng sau nó cho đến khi cx = 0. Sau mỗi lần thực hiện cx tự động giảm đi 1

*Cú pháp:* mov cx, số lần

rep                      lệnh làm việc với xâu ; rep                      movsb





Thuật toán

1. DS:SI
2. ES:DI
3. D = 0
4. CX = 5
5. rep movsb; sau mỗi lần  $cx = cx - 1$  cho đến khi  $cx = 0$ .

#### e) Nhóm 5: Các lệnh rẽ nhánh

##### - Lệnh Call

*Chức năng:* Gọi chương trình con

*Cú pháp*

Call	Addr (seg:offset)
	Label
	Tên chương trình con
	reg
	mem

##### - Lệnh RET

*Chức năng:* quay về chương trình đã gọi chương trình con

*Cú pháp:* RET (nằm ở cuối chương trình con)

##### - Lệnh INT

*Chức năng:* Kích hoạt một ngắt (chuyển sang chạy chương trình con phục vụ ngắt) (Ngắt mềm).

*Cú pháp:* int n (số ngắt viết theo số hexa)

*Ví dụ:* int 21h = int 33

##### - Lệnh IRET

*Chức năng:* quay về chương trình đã kích hoạt nó từ chương trình con phục vụ ngắt.

*Cú pháp:* IRET

##### - Lệnh JMP (go to)

*Chức năng:* nhảy không điều kiện

*Cú pháp:*

jmp	Addr (seg:offset)
	Label
	Tên chương trình con
	reg
	mem

Chú ý: Bước nhảy của lệnh jump < 64k

**- Lệnh nhảy có điều kiện**

Với số không có dấu (Below/above)			Với số có dấu (Less/ greater)			Nhảy theo trạng thái các bit cờ	
Cmp DST, SRC			Cmp DST, SRC				
Jb/jnae	Nhãn Địa chỉ	Khi DST dưới SRC	Jl/jnge	Nhãn Địa chỉ	Khi DST < SRC	jc	Nhãn Địa chỉ Khi CF=1
Jbe/jna	Nhãn Địa chỉ	Khi DST dưới SRC hoặc =	Jle/jng	Nhãn Địa chỉ	Khi DST ≤ SRC	jnc	Nhãn Địa chỉ Khi CF=0
Je	Nhãn Địa chỉ	Khi DST = SRC	Je	Nhãn Địa chỉ	Khi DST = SRC	jz	Nhãn Địa chỉ Khi ZF=1
Jne	Nhãn Địa chỉ	Khi DST ≠ SRC	Jne	Nhãn Địa chỉ	Khi DST ≠ SRC	jnz	Nhãn Địa chỉ Khi ZF=0
Ja/jnbe	Nhãn Địa chỉ	Khi DST trên SRC	Jg/jnle	Nhãn Địa chỉ	Khi DST > SRC	js	Nhãn Địa chỉ Khi SF=1
Jae/jnb	Nhãn Địa chỉ	Khi DST trên / = SRC	Jge/jnl	Nhãn Địa chỉ	Khi DST ≥ SRC	jns	Nhãn Địa chỉ Khi SF=0

Chú ý: Bước nhảy các lệnh nhảy có điều kiện phải nhỏ hơn hoặc bằng 128 byte

**- Lệnh LOOP (for của ASM)**

Chức năng: lặp đi lặp lại khối lệnh ASM nằm giữa nhãn và loop cho đến khi cx = 0. Mỗi khi thực hiện một vòng lặp giá trị của CX giảm đi 1.

Cú pháp:

mov cx, số lần lặp; số lần lặp ≥ 1

Nhãn:

Khối lệnh ASM

Loop Nhãn

**f) Nhóm 6: Các lệnh thao tác với cờ**

**- Lệnh CLC (Clear CF)**

Chức năng: Xóa giá trị cờ CF về 0, CF = 0

Cú pháp: CLC

Cờ C = 0

**- Lệnh STC**

Chức năng: Đặt giá trị cờ CF lên 1, CF = 1

Cú pháp: STC

Cờ C = 1

**- Lệnh CMC**

Chức năng: Đảo giá trị hiện thời của cờ CF.

Cú pháp: CMC

Tác động đèn cờ C.

**- Lệnh CLI**

Chức năng: Xóa giá trị của cờ IF về 0 (IF = 0). Cấm toán bộ các ngắt cứng trừ ngắt MNI.

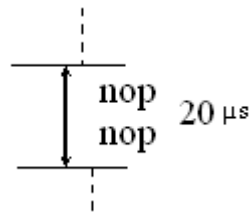
Cú pháp: CLI

Cờ IF = 0.

**- Lệnh STI**

Chức năng: Đặt giá trị của cờ IF lên 1 (IF = 1). Cho phép ngắt cứng.

- Cú pháp:* STI  
Cờ IF = 1.
- **Lệnh CLD**  
*Chức năng:* Xoá giá trị của cờ DF về 0 (DF = 0).  
*Cú pháp:* CLD  
Cờ DF = 0, dựng cờ.
  - **Lệnh STD**  
*Chức năng:* Đưa giá trị của cờ DF lên 1 (DF = 1).  
*Cú pháp:* STD  
Cờ DF = 1.
  - **Lệnh HLT**  
*Chức năng:* dừng máy  
*Cú pháp:* HLT
  - **Lệnh NOP**  
*Chức năng:* lệnh này không thực hiện gì cả  
*Cú pháp:* NOP (4 $\mu$ s)  
*Ý nghĩa:*  
Tạo trễ (delay)



Tạo vòng chờ vô tận để chờ ngắt.

```
L1:  nop
      jmp  L1
```

### 1.5.3 Các lệnh điều khiển khi dịch chương trình (directive)

#### 1.5.3.1. Các directive điều khiển segment: dạng đơn giản

(.MODEL, .STACK, .DATA, .CODE, ...)

##### a) Directive .MODEL

*Chức năng:* cho phép người lập trình xác lập vùng nhớ RAM thích hợp cho chương trình.

*Cú pháp*

. Model	Kiểu	
	Tiny	Code + data $\leq$ 64k
	Small	Code $\leq$ 64k; data $\leq$ 64k
	Compact	Code $\leq$ 64k; data $\geq$ 64k
	Medium	Code $\geq$ 64k; data $\leq$ 64k
	Large	Code $\geq$ 64k; data $\geq$ 64k
		1 array $\leq$ 64k
	Huge	Code $\geq$ 64k; data $\geq$ 64k
		1 array $\geq$ 64k

##### b) Directive .STACK

**Chức năng:** báo cho chương trình dịch của ASM biết xác lập 1 vùng nhớ RAM cho Stack. Với lệnh điều khiển này thì DOS sẽ xác lập địa chỉ đầu của ngăn xếp và giá trị đó được đưa vào thanh ghi segment SS.

**Cú pháp:** . stack độ dài (tính theo byte)

Ví dụ: . stack 100h

Nếu không có khai báo . stack thì lấy độ dài mặc định default.

### c) **Directive . DATA**

**Chức năng:** báo cho chương trình dịch của ASM biết để xác lập 1 vùng nhớ RAM cho dữ liệu chương trình.

**Cú pháp:**

.DATA

Khai báo biến

Biến trong ASM có ba loại: biến số, biến xâu kí tự và biến trường số

#### - **Khai báo biến số**

.DATA

Tên biến

Kiểu

Giá trị ban đầu/?

db ( 1 byte)

dw ( 2 byte)

dd ( 4 byte)

dp ( 6 byte)

dq ( 8 byte)

dt ( 10 byte)

trong đó 2 biến db và dw hay dùng.

Ví dụ:

.DATA

Value

dw

?

Value

db

10

#### - **Khai báo biến xâu kí tự**

.DATA

Tên biến

db

Các kí tự cách nhau bởi dấu

phẩy ,

độ lớn

dup (1

kí tự/ ?)

Ví dụ:

.DATA

xau1

db

'H','e','l','l','o'

xau2

db

100h dup('A')

xau2

db

100 dup(?)

#### - **Khai báo biến trường số**

.DATA

Tên trường số

kiểu của thành phần (Các số cách

nhau bởi dấu,)

Độ lớn

dup( 1 số/?)

Ví dụ:

.DATA

array1	db	100,2,21,31	
array2	dw	100h	dup(-100)
array3	dd	100	dup(?)

**Chú ý:** Nếu chương trình có khai báo biến (tức là có .DATA) thì người lập trình ASM phải đưa phần địa chỉ segment của vùng nhớ dữ liệu vào trong DS nhờ 2 lệnh sau:

```
mov reg16, @data
mov ds, reg16
```

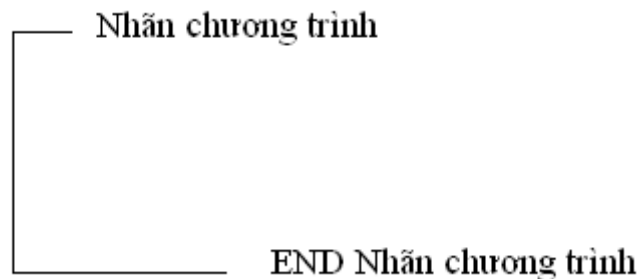
Ví dụ:

```
mov ax, @data
mov ds, ax
```

#### d) Directive **.CODE**

**Chức năng:** Báo cho chương trình dịch ASM biết để xác lập 1 vùng nhớ RAM cho phần tử mã máy của chương trình.

**Cú pháp:** .CODE



#### e) **Dạng thường thấy 1 chương trình ASM đơn giản**

(Khai báo theo directive điều khiển segment dạng đơn giản)

```
.MODEL
.STACK
.DATA
```

Khai báo biến

```
.CODE
```

Nhãn chương trình:

```
mov ax, @data
mov ds, ax
```

Thân chương trình

```
mov ah, 4ch
int 21h
```

END Nhãn chương trình

**Ví dụ 1:** Hiện 1 xâu lên màn hình

**Cách 1:** Dùng chức năng hiện 1 xâu '\$' lên màn hình

```
lea dx, tên biến xâu
mov ah, 9
int 21h
```

C:\BT>edit vd1.asm

```
.MODEL small
```

```

.STACK      100h
.DATA
M          db      'Hello, World!$'
.CODE
PS:
mov        ax, @data
mov        ds,ax*4
lea        dx, M
mov        ah,9
int        21h
mov        ah,1
int        21h
mov        ah,4ch
int        21h
END PS

```

### Cách 2: Dùng lệnh LODSB và khai báo theo dạng ngôn ngữ C

```

.MODEL      small
.STACK      100h
.DATA
M          db      'Hello, World!',0
.CODE
PS:
mov        ax, @data
mov        ds,ax
lea        si, M
cld
L1:
Lodsb
And        al,al
Jz         KT
mov        ah,0eh
int        10h
jmp        L1
KT:
mov        ah,1
int        21h
mov        ah,4ch
int        21h
END PS

```

### Cách 3: Không dùng lệnh làm việc với xâu

```

.MODEL      small
.STACK      100h
.DATA
M          db      'Hello, World!',0

```

```

.CODE
    PS:
        mov     ax, @data
        mov     ds,ax
        lea     si, M
        cld
        L1:
            mov     al,[si]; mov     al, ds:[si]
            and     al,al
            jz      KT
        mov     ah,0eh
            int     10h
        inc     si
        jmp     L1
KT:
        mov     ah,1
        int     21h
        mov     ah,4ch
        int     21h
        END PS

```

### Ví dụ 2: Hiện nội dung AX lên màn hình dạng binary

AX = -1 suy ra ct      1111    1111    1111    1111

AX = 100  $\longrightarrow$  0000 0000 0110 0100

AX = 255  $\longrightarrow$  0000 0000 1111 1111

```
C:\BT>edit    vd2.asm
```

```

.MODEL small
.STACK 100h
.CODE
    PS:
        mov     ax, số thứ -1 hoặc 100 hoặc 255
        mov     bx,ax
        mov     cx, 16

    L1:
        xor     al, al
        shl     bx,1
        adc     al,0
        add     al,30h;        hiện mã ASCII
        mov     ah,0eh
        int     10h
        loop    L1
        mov     ah,1
        int     21h
        mov     ah,4ch

```

```
int      21h
END PS
```

### Ví dụ 3: Tính 5!

#### Cách 1: không dùng biến

```
.MODEL      small
.STACK      100h
.CODE
PS:
    mov     ax, 1
    mov     cx, 5
L1:
    mul     cx
    loop    L1
    mov     ah, 1
    int     21h
    mov     ah, 4ch
    int     21h
END PS
```

#### Cách 2: dùng biến

```
.MODEL      small
.STACK      100h
.DATA
    FV      dw      ?
    FAC      dw      ?
.CODE
PS:
    mov     ax, @data
    mov     ds, ax
    mov     FV, 1
    mov     FAC, 2
    mov     cx, 4
L1:
    mov     ax, FV
    mul     FAC
    mov     FV, ax
    inc     FAC
    loop    L1
    mov     ah, 4ch
    int     21h
END PS
```

Giải thích:

cx = 4

ax = FV = 1



$dx:ax = ax * FAC = ax = 1.2$	$ax = 1.2$	$ax = 1.2.3$	$ax = 1.2.3.4$
$FV = ax = 1.2$	$ax = 1.2.3$	$ax = 1.2.3.4$	$ax = 1.2.3.4.5$
$FAC = 3$	$FV = 1.2.3$	$FV = 1.2.3.4$	$FV = 1.2.3.4.5$
$cx = 3?0$	$FAC = 4$	$FAC = 5$	$FAC = 6$
	$Cx = 2?0$	$Cx = 1?0$	$Cx = 0?0$

## f) Công cụ DEBUG

Chức năng: gỡ rối chương trình ASM

Quy ước:

- Mỗi lệnh là 1 ký tự: D, T, G, P, Q, N, L, O
- Giá trị làm việc với DEBUG là hệ hexa

Khởi động công cụ DEBUG

Cách 1: ..... \> debug      tentep.exe (.com) ↵

Cách 2: ..... \> debug ↵

- N      tentep.exe ↵ (.com)

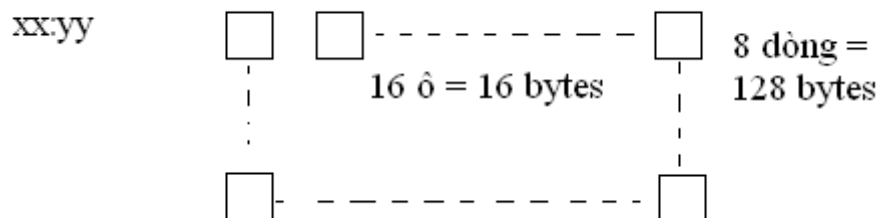
- L ↵

Các lệnh hay dùng

### - Lệnh D (Dump = Display)

Chức năng: hiện vùng nhớ lên máy tính

Cú pháp: - D địa chỉ ô đầu ; ↵ (seg:offset)

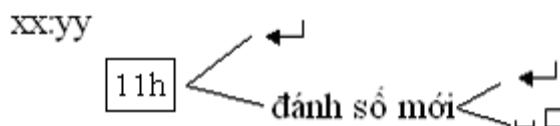


- D ↵      hiện tiếp 128 byte

### - Lệnh E (Enter)

Chức năng: hiện và sửa nội dung ô nhớ .

Cú pháp: - E địa chỉ offset ; ↵ (seg:offset)

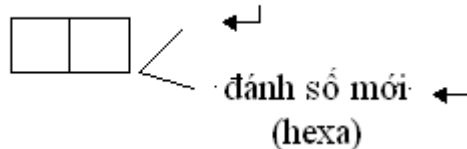


### - Lệnh R (Register)

Chức năng: hiện và sửa nội dung 1 thanh ghi.

Cú pháp: - R      Tên thanh ghi ↵ ; tên huy của thanh ghi

tên thanh ghi



### - Lệnh G (Go)

**Chức năng:** chạy từ nơi chương trình đang đứng (IP đang trỏ) đến hết chương trình.

**Cú pháp:**  $\leftarrow G$

**Giải thích:** lệnh 1

lệnh 2

#### - Lệnh T/P

**Chức năng:** cho phép chạy 1 bước.

**Cú pháp:**  $\leftarrow T/P$

Hiện nội dung các thanh ghi

AX	BX	CX	DX	SI	DI	BP	ZI
AY	ODD						
DS		ES		CS	SS	<div style="border: 1px solid black; display: inline-block; padding: 2px;">Z                      A</div>	SP

NZ                      NA      EVEN

lệnh sắp thực hiện tiếp theo.

Sự khác nhau giữa T và P

#### 1. Call

Call      tên chương trình con

- T  $\leftarrow$       nhảy vào chương trình con

- P  $\leftarrow$       chạy hết chương trình con, coi chương trình con là 1 lệnh

#### 2. INT (ngắt mềm)

int      n

- T  $\leftarrow$       nhảy vào thân chương trình con phục vụ ngắt

- P  $\leftarrow$       chạy hết chương trình con phục vụ ngắt

#### 3. Loop Nhãn

Nhan:

Loop      Nhan

- T  $\leftarrow$       chạy từng vòng một; cx = cx - 1?0

- P  $\leftarrow$       chạy toàn bộ các vòng cx      0

#### - Lệnh Q (Quit)

**Chức năng:** trở về DOS.

**Cú pháp:**  $\leftarrow Q$

#### - Lệnh U (UnAssembly)

**Chức năng:** dịch ngược từ dạng **.exe** hay **.com** sang dạng **.asm**

**Cú pháp:**      - U      địa chỉ ô nhớ đầu      ; seg: offset

địa chỉ      mã máy      lệnh mnemonic

xx:yy      eg      jmp

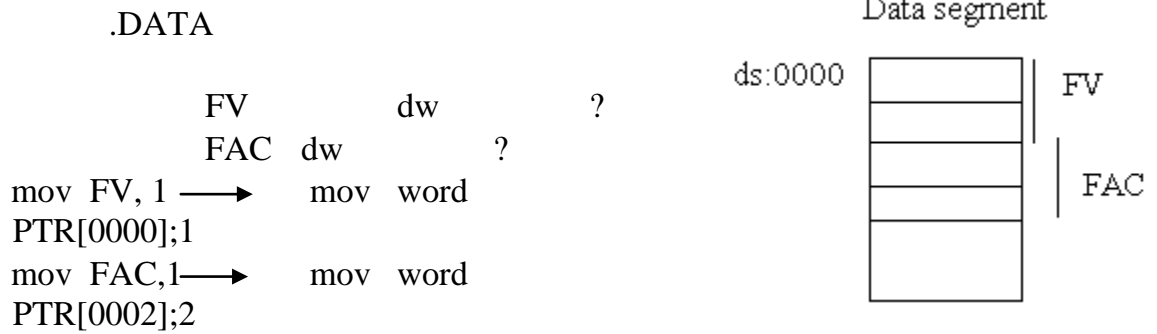
**Chú ý vấn đề khai báo biến**

1. Khai báo biến tức là xin cấp phát ô nhớ.

2. Biến nào được khai báo trước sẽ chiếm ô nhớ trước.

### 3. Biến khai báo đầu tiên sẽ có địa chỉ offset = 0000h

Giải thích



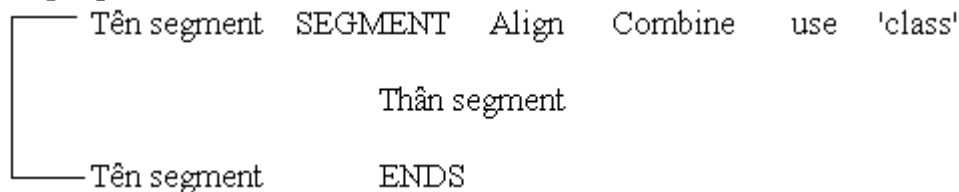
#### 1.5.3.2. Các directive điều khiển segment: dạng chuẩn

(SEGMENT, GROUP và ASSUME)

##### a) Directive **SEGMENT**

Chức năng: báo cho chương trình dịch ASM xác lập các segment cho chương trình.

Cú pháp:



- Tên Segment: bất kỳ một định danh nào.

- Align

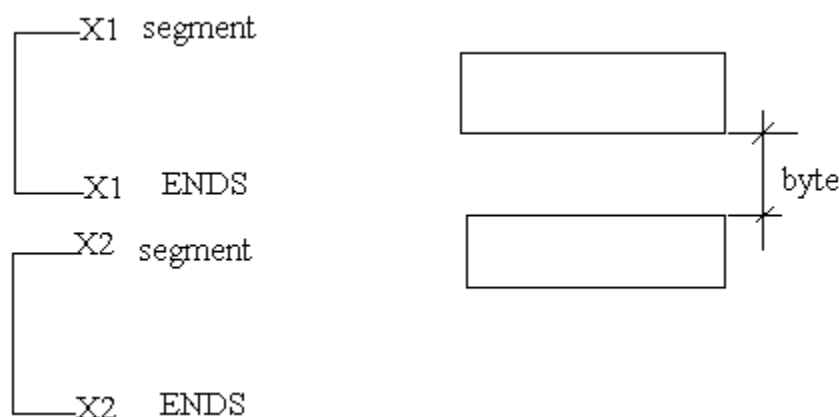
Chức năng: xác lập khoảng trống giữa segment đang khai báo với segment trước nó.

Cú pháp:

```

ALIGN
BYTE
WORD
PARA
(16 BYTE)      (Default)
PAGE
(128 BYTE)
    
```

Giải thích:



- Combine

Chức năng 1: cho phép đặt segment khai báo 1 vùng nhớ RAM theo yêu cầu.

Cú pháp:      tên segment                      SEGMENT                      at địa chỉ

                         Tên segment                      ENDS

Chức năng 1: phục vụ chương trình đa tệp thuần túy ASM, cách gộp các segment có cùng tên nằm ở các tệp khác nhau khi liên kết.

Cú pháp:

                         COMMON                      Overlay đè lên nhau  
                         PUBLIC                      Continue, Σ  
                         PRIVATE (Default)                      Không biết nhau

Ví dụ:

                         tep1.asm    tep2.asm  
X1              Segment              common              X1              Segment              common

- USE : chỉ máy tính 32 bit trở lên

            use16 →                      ASM 16 bit (default)  
            use32 →                      ASM 32 bit

- 'CLASS'

Chức năng: cho phép gộp các segment có cùng lớp lại gần nhau khi liên kết

```

X1 segment 'm'
|
X1 ENDS

X2 segment
|
X2 ENDS

X3 segment 'm'
|
X3 ENDS
    
```

Cách khai báo 3 segment của chương trình

Dạng chuẩn	Dạng đơn giản
Stack segment db 100h dup (?)      ← Stack ends	.Stack 100h
Data segment Khai báo biến Data ends	.DATA Khai báo biến
Chú ý:    mov    ax, data	Chú ý:    mov    ax, @data

mov ds, ax	mov ds, ax
Code segment Nhan CT:	.CODE Nhan CT:
Code ends ENDS Nhan CT	ENDS Nhan CT

### b) Directive GROUP

Chức năng: gộp các segment cùng loại cho dễ dàng qui chiếu.

Cú pháp:

tên nhóm                      GROUP                      tên các segment  
   Khai báo các segment

Giải thích:

```
Data1      segment
                M1      db      ?
Data1      ends
Data2      segment
                M2      dw      ?
Data2      ends
```

Code segment

PS:

```
mov     ax, data1
mov     ds, ax
mov     cl, M1
```

```
mov     ax, data2
mov     ds, ax
mov     cl, M2
```

Ta làm group như sau:

```
Nhom_DL      GROUP      data1,data2
Data1      segment
                M1      db      ?
Data1      ends
Data2      segment
                M2      dw      ?
Data2      ends
Code segment
PS:
mov     ax, nhom_DL
mov     cl, M1
mov     dx, M2
```

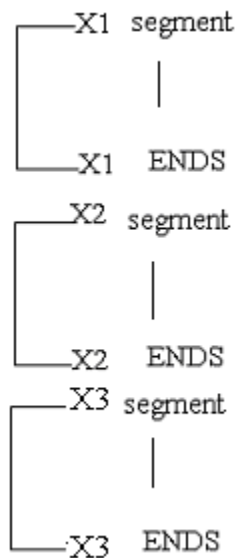
### c) Directive ASSUME

Chức năng: cho biết segment khai báo thuộc loại segment nào

Cú pháp:

assume      tên thanh ghi segment : tên segment

Giải thích



assume      cs:x3, ds:x2,ss:x1

Chú ý: assume thường là dòng đầu của code segment

### **Dạng chương trình ASM đơn giản (dạng chuẩn)**

Stack segment

db 100h dup (?)

Stack ends

Data segment

Khai báo biến

Data ends

Code segment

Assume cs:code, ds:data, ss:\_stack

Nhan CT:

mov ax, data2

mov ds,ax

mov ah, 4ch

int 21h

code ends

END Nhan CT

Bài tập: Hiện chuỗi kí tự '\$'

Stack segment



Chức năng: con trỏ đến các thành phần của biến nhớ (cho phép lấy từng byte).

Cú pháp:

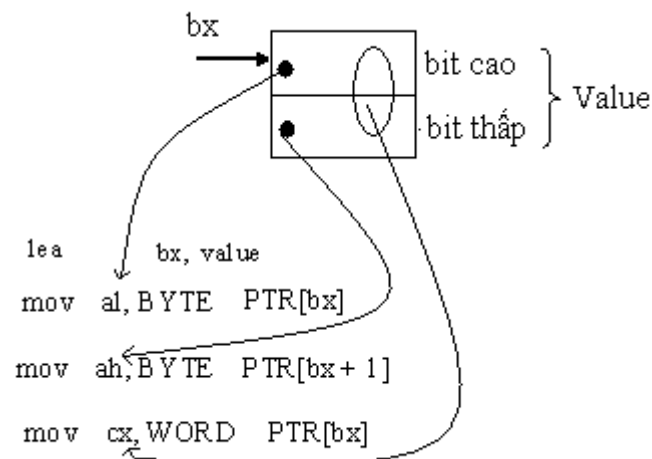
Kiểu PTR [thanh ghi]

Ví dụ:

.DATA

Value dw ?

.CODE



## 1.6. Chương trình con

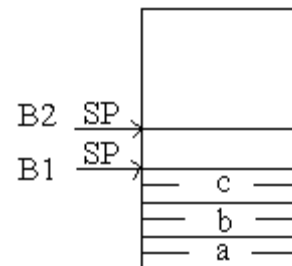
### 1.6.1. Ý nghĩa của chương trình con

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

### 1.6.2. Cơ chế khi một chương trình con bị gọi

Cơ chế có 5 bước:

- Bước 1: Tham số thực đưa vào stack
- Bước 2: Địa chỉ lệnh tiếp theo đưa vào stack
- Bước 3: Hệ điều hành biết được địa chỉ đầu của chương trình con. Do vậy hệ điều hành đưa địa chỉ đầu của chương trình con vào CS:IP → rẽ nhánh vào chương trình con.
- Bước 4: Thực hiện chương trình con cho đến khi gặp return thì vào stack lấy địa chỉ lệnh tiếp theo (đã cất ở bước 2 để đưa vào CS:IP) và quay về chương trình đã gọi nó.
- Bước 5: tiếp tục chương trình đang thực hiện dở.



### 1.6.3. Cú pháp một chương trình con ASM

Tên chương trình con PROC [near/far]

Bảo vệ các thanh ghi mà thân chương trình con phá vỡ.

Các lệnh ASM của thân chương trình con.

Hồi phục các thanh ghi mà thân chương trình con đã phá

vỡ.

RET

Tên chương trình con

ENDP



### Nhận xét

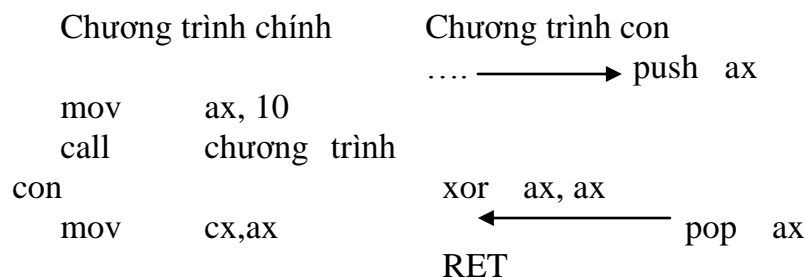
1. Chương trình con thuần túy ASM không có đối.
2. Vấn đề near/ far
  - Chương trình con là near khi mã máy của chương trình con và mã máy của chương trình chính là cùng nằm trong 1 segment, địa chỉ của chương trình con và chương trình chỉ khác nhau phần địa chỉ offset. Cho nên địa chỉ lệnh tiếp theo cất vào stack (Bước 2, mục 1.6.2) chỉ cần 2 byte offset.
  - Chương trình con là far khi mã máy của chương trình con và mã máy của chương trình chính nằm trên các segment khác nhau, địa chỉ của chương trình con và chương trình chính khác nhau cả về phần địa chỉ segment. Cho nên địa chỉ lệnh tiếp theo cất vào stack (Bước 2, mục 1.6.2) phải cần 4 byte offset ( 2 byte segment và 2 byte offset).

### Default:

- Với chương trình được khai báo directive dạng đơn giản thì directive MODEL sẽ cho biết chương trình con là near hay far  
 Nếu .MODEL tiny/small/compact thì chương trình con là NEAR(mã máy< 64k)  
 Nếu .MODEL medium/large/huge thì chương trình con là FAR(mã máy>64k)
- Với chương trình con được viết theo directive dạng chuẩn thì mặc định là near. Còn muốn chương trình con là far thì phải viết far khi viết chương trình con.

3. Vấn đề cần bảo vệ thanh ghi và phục hồi các thanh ghi trong thân chương trình con.

Ví dụ:



Bảo vệ và hồi phục các thanh ghi và thân chương trình con phá vỡ tốt nhất bằng cơ chế PUSH và POP.

Ví dụ 1: Hãy viết chương trình con ASM cho phép nhận một số nguyên (-32768 ~ 32767) từ bàn phím kết thúc nhận một số nguyên bằng phím Enter (13 = 0dh).

Kết quả nằm trong thanh ghi ax. Chú ý không cho phép đánh sai và sửa.

- a) Nhận số nguyên dương

- Dùng hàm nhận kí tự

```
mov ah, 1
```

```
int 21h
```

Suy ra al chứa mã ASCII của kí tự.

al – 30h: thành mã ASCII chuyển thành số

- Số vừa đưa vào sẽ cộng phần số đã cất vào trước \*10

- b) Nhận một số nguyên âm

- Có 1 biến cờ dấu: 0 là số dương, 1 là số âm.

Nếu phát hiện kí tự đầu là dấu âm thì biến cờ dấu sẽ bằng 1.

Nhận một số nguyên dương sau đó hỏi biến cờ dấu. Nếu cờ dấu = 1 thì chuyển sang số bù 2 để đổi dấu.

```

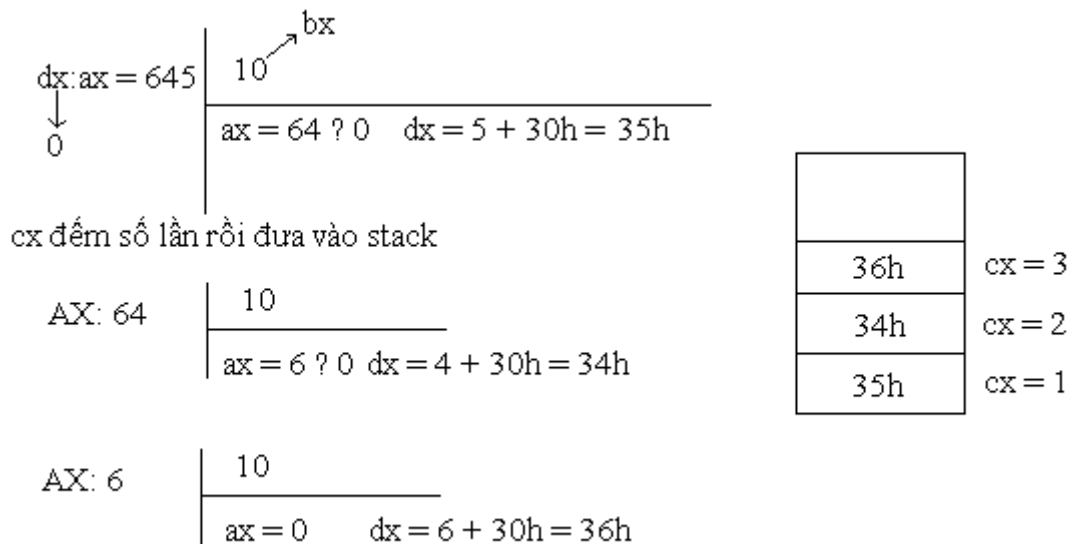
VAO_SO_N      PROC
                push  bx      cx      dx, si
                mov   bx,10
xor             cx, cx; cx = 0 cx = phần số đã vào trước
mov  si, cx; SI = biến cờ dấu
VSN1:
mov  ah, 1
int  21h
cmp  al, 13; Enter?
je   VSN3
cmp  al, '-'
jne  VSN2
inc  si
jmp  VSN1
VSN2:
sub  al, 30h
xor  ah, ah
exchg ax, ax; Đổi chỗ số vừa vào và số đã vào trước
mul  bx
add  cx, ax
jmp  VSN2
VSN3:
and  si, si
jz   VSN4
neg  cx
VSN4:
mov  ax, cx
pop  SI      dx      cx      bx
VAO_SO_N      ENDP

```

Ví dụ 2: Viết chương trình con hiện nội dung có trong AX ra ngoài màn hình dạng cơ số 10.

Thuật toán:

a) AX chứa số nguyên dương



vòng loop

```

X:   pop    ax
      mov    ah, 0ch
      int     10h
      loop X
    
```

b) AX chứa số âm

Kiểm tra  $AX \leq 0$

- Nếu  $AX \leq 0$  hiện dấu ra màn hình sau đó đổi dấu AX rồi hiện như một số nguyên dương sau dấu trừ.

- Chương trình

```

HIEN_SO_N      PROC
                push    ax      bx      cx      dx
                mov     bx, 10
                xor     cx, cx
                and     ax, ax
                jns     HSN1
                push    ax
                mov     al, '-'
                mov     ah, 0eh
                int     21h
                pop     ax
                neg     ax
                HSN1:
                xor     dx, dx
                div     bx
                add     dx, 30h
                push    dx
                inc     cx
                and     ax, ax
                jnz     HSN1
                HSN2:
                pop     ax
    
```

```

                                mov     ah,0eh
                                int      10h
                                loop HSN2
                                pop      dx     cx     bx     ax
                                ret
HIEN_SO_N      END

```

## 1.7. MACRO

### 1.7.1. Ý nghĩa

Cho phép người lập trình ASM tạo lập 1 lệnh ASM mới, trên cơ sở tập lệnh chuẩn của ASM.

### 1.7.2. Khai báo (xác lập) MACRO

Cú pháp:

```

Tên Marco      Marco [đối]
    Bảo vệ các thanh ghi mà thân Marco phá vỡ
    Các lệnh ASM trong thân Marco
    Hồi phục các thanh ghi mà thân Marco đã phá vỡ
ENDM

```

Ví dụ: Hãy khai báo 1 Marco tạo 1 lệnh mới cho phép xoá toàn bộ màn hình

Cơ chế màn hình ở chế độ text, mỗi lần đặt mode cho màn hình thì màn hình sẽ bị xoá và con trỏ đứng ở góc trên bên trái.

Set mode:

```

mov  al, số mode
mov  ah,0
int   10h

```

Get mode

```

mov  ah, 0fh
int   10h

```

Clrscr MARCO

```

                                push  ax
                                mov    ah,0fh;      get mode
                                int     10h
                                mov     ah,0
                                int      10h;      set mode
                                pop     ax
                                ENDM

```

**Ví dụ 2:** Khai báo 1 Marco cho phép hiện 1 xâu lên màn hình

```

Hienstring  MARCO
                                push  ax     dx
                                lea     dx, xau
                                mov     ah,9
                                int     21h
                                pop     dx     ax
                                ENDM

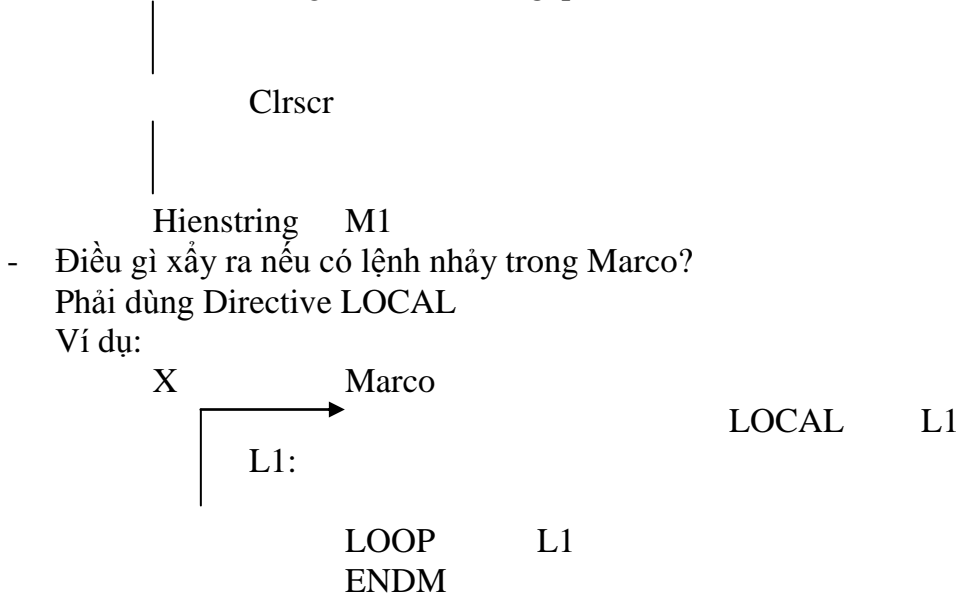
```

### 1.7.3 Cách dùng MACRO đã được xác lập

Sau khi 1 marco đã được khai báo thì tên marco được tạo thành 1 lệnh mới của ASM. Sử dụng bằng cách viết tên marco và thay tham số thực cho đối.

Chú ý:

Cơ chế của chương trình dịch khi gặp lệnh mới



## 1.8. Directive INCLUDE

### 1.8.1. Ý nghĩa

- Cho phép chèn khối lệnh nằm ở 1 tệp ngoài chương trình đang viết

### 1.8.2 Cú pháp chèn

include ổ đĩa:\đường dẫn\ tên tệp.đuôi

### 1.8.3. Cơ chế khi chương trình dịch TASM gặp directive INCLUDE

include ổ đĩa:\đường dẫn\ tên tệp.đuôi

### Các bước thực hiện

B1: tìm tệp đứng sau directive INCLUDE

B2: Mở tệp đó.

B3: Chèn khối lệnh vào directive INCLUDE

B4: Dịch khối lệnh đó

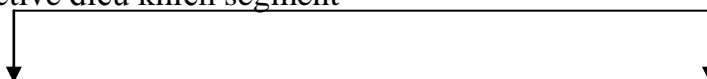
B5: Đóng tệp

**Chú ý:** Nếu dùng directive INCLUDE với 1 tệp 2 lần trở lên thì không cho phép dùng lệnh nhảy trong đó.

### Dạng thường thấy 1 chương trình ASM phức tạp

(Khai báo MARCO, STRUC, UNION ..)

Các Directive điều khiển segment



Dạng đơn giản Dạng chuẩn	
<pre> .MODEL    small .STACK    100h .DATA ; Khai báo biến .CODE Nhãn CT:     [mov    ax, @data      mov     ds, ax]     Thân CT chính           mov     ah, 4ch     int     21h     [ Các CT con]     END Nhãn CT         </pre>	<pre> Stack    segment           db    100h Stack    ends Data     segment           Khai báo biến Data     ends Code     segment Assume   cs:code, ds:data, ss:stack Nhãn CT:     [mov     ax, data      mov      ds, ax]     Thân CT chính           mov     ah, 4ch     int     21h     [ Các CT con] code     ends     END Nhãn CT         </pre>

Giả thiết: lib1.asm

```

                                Clrscr
                                hiện string
lib2.asm                        VAO_SO_N        PROC
                                RET
                                VAO_SO_N        END
                                HIEN_SO_N       PROC
                                RET
                                HIEN_SO_N       ENDP
        
```

## BÀI TẬP

**Bài 1:** So sánh 2 số nguyên và hiện số có giá trị bé lên màn hình.

Khi chạy chương trình yêu cầu có dạng:

- Xoá màn hình.

```

Hay vào số thứ nhất: - 152
Hay vào số thứ hai: 31
Số bé là: -152
Có tiếp tục CT(C/K)?
        
```

Hướng dẫn:

Tạo file C:\BT>edit

sosanh.asm

```

Include lib1.asm
.MODEL small
.STACK 100h
.DATA
    M1 db 13,10, ' Hay vào số thứ nhất: $'
    M2 db 13,10, ' Hay vào số thứ hai: $'
    M3 db 13,10, ' Số bé là: $'
    M4 db 13,10, ' Có tiếp tục CT (C/K): $'
.CODE
    PS: mov ax, @data
        mov ds, ax
        clrscr
        Hienstring M1
        call VAO_SO_N
        mov bx, ax
        Hienstring M2
        call VAO_SO_N
        Hienstring M3
        cmp ax, bx
        jl L1
        xchg ax, bx
    L1: call Hien_so_N
        Hienstring M4
        mov ah,1
        int 21h
        cmp al,'c'
        jne exit
        jmp PS
    Exit: mov ah,4ch
        int 21h
        Inculde lib2.asm
        END PS

```

**Bài 2:** Tính  $n!$  (0 - 7)

Chỉ chương trình chạy yêu cầu:

- Xóa màn hình

```

Hay vào số n: 7 ↵
Giải thừa của 7 là: 5040 ↵
Có tiếp tục CT (C/K)?

```

C:\BT&gt;edit gth.asm ↵

```

Include lib1.asm
_stack segment
    db 100h dup(?)
_stack ends
Data segment
    M1 db 13,10, ' Hay vào số n: $'

```

```

M2    db    13,10, ' Giai thua cua '
M3    db    13,10, ' la: $'
M4    db    13,10, ' Co tiep tục CT (C/K): $'
FV     dw    ?
FAC    dw    ?

```

Data ends

Code segment

Assume cs:code, ds:data, ss:stack

PS:

```

mov    ax, data
mov    ds, ax
clrscr
Hienstring M1
call   VAO_SO_N
Hienstring M2
call   VAO_SO_N
Hienstring M3
mov    FV, 1
mov    FAC, 2
mov    cx, ax
cmp    cx, 2
jb     L1
dec    cx
L1:
mov    ax, FV
mul    FAC
mov    FV, ax
inc    FAC
loop   L2
L2:
mov    ax, FV
call   HIEN_SO_N
Hienstring M4
mov    ah, 1
int     21h
jmp    al, 'c'
jne     Exit
jmp    PS

```

Exit:

```

mov    ah, 4ch
int     21h
Include lib2.asm

```

Code ends

END PS

**Bài 3:**  $a^n$  (a là số nguyên, n là số nguyên dương)

Khi chương trình chạy yêu cầu có dạng

- Xóa màn hình



```

Hay vào a: - 4 ←
Hay vào n: 3 ←
-4 lũy thừa 3 là: - 64
Có tiếp tục CT (C/K)?
    
```

```

C:\BT>edit lt.asm
Include lib1.asm
.MODEL small
.STACK 100h
.DATA
    M1 db 13,10, ' Hay vào a: $'
    M2 db 13,10, ' Hay vào n: $'
    Crlf db 13, 10, '$'
    M3 db ' ' , 'Lũy thừa ' : '$'
    M4 db ' ' , 'là : $'
    M5 db 13,10, ' Có tiếp tục CT (C/K): $'
.CODE
PS:
    mov ax, @data
    mov ds, ax
    clrscr
    Hienstring M1
    call VAO_SO_N
    mov cx, ax; cx = n
    Hienstring Crlf
    mov ax, bx
    call HIEN_SO_N
    Hienstring M3
    mov ax, bx
    call HIEN_SO_N
    Hienstring M4
    mov ax, 1
    and cx, ax
    jz L1
    L1:
        mul ax; ax*bx để vào ax
        loop L2
    L2:
        call HIEN_SO_N
        Hienstring M5
        mov ah,1
    int 21h
    cmp al, 'c'
    jne Exit
    jmp PS

Exit:
    mov ah,4ch
    
```

```

int      21h
Include  lib2.asm
END      PS

```

**Bài 4:** trung bình cộng các số nguyên

Khi chương trình chạy yêu cầu có dạng:

```

Hay vào số thứ nhất: -12
Hay vào số thứ hai: -15
TBC hai số là: -13,5
Có tiếp tục CT (C/K)?

```

```

C:\BT>edit      tbc.asm
Include lib1.asm
_Stack segment
    db      100h dup(?)
_Stack ends
Data segment
    M1      db      13,10, ' Hay vào số thứ nhất: $'
    M2      db      13,10, ' Hay vào số thứ hai:  $'
    M3      db      13,10, ' TBC hai số là: $'
    M4      db      ' ' $'
    M5      db      '.5 $'
    M6      db      13,10, ' Có tiếp tục CT (C/K): $'
Data ends
Code segment
    Assume   cs:code, ds:data, ss:stack
PS:

```

```

    mov     ax, data
    mov     ds, ax
    clrscr
    Hienstring M1
    call    VAO_SO_N
    mov     bx, ax
    Hienstring M2
    call    VAO_SO_N
    Hienstring M3
    add     ax, bx
    and     ax, ax
    jns     L1
    Hienstring M4
    neg     ax;      đổi dấu = lấy bù 2
    L1:
                shr     ax, 1; dịch phải 1 lần
    pushf
    call    HIEN_SO_N
    popf
    jnc     L2
    Hienstring M5

```

L2:

```

Hienstring M6
mov ah,1
int 21h
cmp al,'c'
jne Exit
jmp PS

```

Exit:

```

mov ah,4ch
int 21h
Include lib2.asm

```

Code ends

END PS

**Bài 5:** Tính tổng 1 dãy số nguyên.

Yêu cầu:

- Nhập số lượng thành phần.
- Nhận các số đưa vào mảng
- Hiện các số vừa vào ra màn hình
- Tính tổng.
- Hiện kết quả

Yêu cầu khi chương trình chạy có dạng:

```

Hay vào sltp: 4
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
Day so vua vào la: 1 2 3 4
Co tiep tục ct (C/K)?

```

```

C:\BT>edit tong.asm
INCLUDE lib1.asm
.MODEL small
.STACK 100h
.DATA
M1 db 13,10, ' Hay vào số lượng thành phần: $'
M2 db 13,10, 'a [ $'
M3 db ' ] $'
M4 db 13,10, ' Day số vừa vào là : $'
M5 db ' $'
M6 db 13,10, ' Tổng day là : $'
M7 db 13,10, ' Có tiếp tục CT (C/K): $'
sltp dw
i dw
a dw 100h dup(?); khai báo mảng
.CODE
PS:
mov ax, @data
mov ds, ax

```

```

    clrscr
    Hienstring M1
    call VAO_SO_N
    mov sltp,ax
    mov cx, ax
    lea bx, a; lấy phần địa chỉ offset a[0] đưa vào bx
    mov i, 0
    L1:                                     ; Nhập các số đưa vào mảng
        Hienstring M2
    mov ax, i
    call HIEN_SO_N
    Hienstring M3
    call VAO_SO_N
    mov [bx], ax
    inc i
    add bx, 2 ; tăng 2 lần bx tức + 2 vào bx
    loop L1

    Hienstring M4
    mov cx, sltp
    lea bx, a ; bx trở vào a[0]

L2:                                     ; lấy các số hiện lên màn hình
    mov ax, [bx]
    call HIEN_SO_N
    Hienstring M5
    add bx, 2
    loop L2

    Hienstring M6
    mov cx, sltp
    lea bx, a
    xor ax, ax
    L3:
    add ax, [bx]
    add bx, 2
    loop L3
    call HIEN_SO_N
    Hienstring M7
    mov ah, 1
    int 21h
    cmp al, 'c'
    jne Exit
    jmp PS
Exit:
    mov ah, 4ch
    int 21h

```

```

        Include      lib2.asm
END      PS

```

Bổ xung:

Tính tổng dương, tổng âm

L3:

```

        mov  dx, [bx]
        and  dx, dx
        js   L4      ; jns
        add  ax, dx

```

L4:

```

        add  bx, 2
        loop L3

```

Tính tổng chẵn, tổng lẻ

L3:

```

        mov  dx, [bx]
        shr  dx, 1
        jc   L4      ; jnc
        add  ax, [bx]

```

L4:

```

        add  bx, 2
        loop L3

```

**Bài 6:** Số nguyên tố

C:\BT>edit snt.asm

INCLUDE lib1.asm

.MODEL small

.STACK 100h

.DATA

```

M1  db  13,10, ' Hay vào số giới hạn: $'
M2  db  13,10, ' Các số nguyên tố vào từ 2 đến $'
M3  db  ' ' , ' là: $'
M4  db  ' ' , ' $'
M5  db  13,10, ' Tiếp tục CT (C/K): $'
so  dw

```

.CODE

PS:

```

        mov  ax, @data
        mov  ds, ax
        clrscr
        Hienstring M1
        call VAO_SO_N
        mov  bx, ax ;bx = số giới hạn
        Hienstring M2
        call HIEN_SO_N
        Hienstring M3
        mov  i, 0
        L1:

```

```

inc    so
mov    ax, so
cmp    ax, bx
ja     KT                ; so sánh đưa ra kết thúc
mov    cx, ax
shr    ax, 1             ; cx = so/2
L2:
cmp    cx, 2
jb     HIEN
xor     dx, dx
div    cx
and    dx, dx
jz     L1
mov    ax, so
loop   L2
HIEN:
Call   HIEN_SO_N
Hienstring M4
jmp    L1
KT:
Hienstring M5
mov    ah, 1
int    21h
cmp    al, 'c'
jne     Exit
jmp     PS
Exit:
mov    ah, 4ch
int    21h
Include lib2.asm
END     PS

```

## 1.9. Chương trình đa tệp

### 1.9.1. Ý nghĩa

Cho phép nhiều người cùng tham gia viết 1 chương trình lớn.

Làm sao các nhãn dùng chung (tên biến nhớ, tên chương trình con) phải hiểu nhau. Để giải quyết vấn đề này chương trình dịch ASM có trang bị hai directive đó là PUBLIC (cho phép) và EXTRN (xin phép)

### 1.9.2. Directive **PUBLIC**

Chức năng: báo cho chương trình dịch ASM biết module (tệp) này cho phép các tệp khác được dùng những nhãn nào mà không cần xác lập lại.

Cú pháp:

```

PUBLIC    Tên nhãn
          Xác lập nhãn

```

- Nhãn là tên biến

```

.DATA
PUBLIC    Tên biến

```

## Khai báo biến

Ví dụ:

```
.DATA
PUBLIC      x,y
            x   db   ?
            y   dw   ?
```

- Nhân là tên chương trình con

```
.CODE
PUBLIC      Tên chương trình con
Tên chương trình con  PROC
```

RET

```
Tên chương trình con  ENDP
```

**1.9.3. Directive EXTRN**

Chức năng: báo cho chương trình dịch ASM biết tệp này xin phép dùng các nhãn mà các modul khác đã xác lập và cho phép.

Cú pháp:

```
EXTRN      Tên nhãn: Kiểu
```

- Với nhãn là biến nhớ

```
.DATA
EXTRN      Tên biến: Kiểu

PUBLIC
            BYTE
            db
            WORD
            dw
            DWORD
            dd
```

Ví dụ:

```
.DATA
EXTRN      X:BYTE, Y:WORD
```

- Nhãn là tên chương trình con

```
.CODE
EXTRN      Tên chương trình con:PROC
```

Ví dụ: n!

Anh A (gt1.asm) - Nhận n - Gọi chương trình con tính n! (do B viết) - Hiện kết quả	<u>n</u> , <u>FV</u> , <u>GT</u> PUBLIC, EXTRN, EXTRN
Anh B(gt2.asm): Viết chương trình con tính n!	<u>n</u> , <u>FV</u> , <u>GT</u> EXTRN,    PUBLIC, PUBLIC

Viết CT

```
C:\BT>edit  gt1.asm
Include  lib1.asm
.MODEL   small
```

```

.STACK      100h
.DATA
M1  db      13,10, ' Hay vào số n: $'
M2  db      13,10, ' Giai thừa của $'
M3  db      '_', ' la: $'
M4  db      13,10, ' Có tiếp tục CT (C/K): $'
PUBLIC  n
        n      dw      ?
        EXTRN  FV:Word
.CODE
EXTRN  Factorial:PROC
PS:    mov    ax, @data
        mov    ds, ax
        clrscr
        Hienstring  M1
        call  VAO_SO_N
        mov    n, ax
        Hienstring  M2
        call  HIEN_SO_N
        Hienstring  M3
        call  Factorial
        mov    ax, FV
        call  Hien_so_N
        Hienstring  M4
        mov    ah,1
        int    21h
        cmp    al,'c'
        jne     exit
        jmp     PS
Exit:
        mov    ah,4ch
        int    21h
        Inculde  lib2.asm
        END PS
C:\BT>edit  gt2.asm
.MODEL      small
.DATA
        EXTRN  n:Word
        PUBLIC
        FV      dw      ?
        FAC      dw      ?
.CODE
        PUBLIC  Factorial
        Factorial  PROC
                mov    FV,1
                mov    FV,2
                mov    cx,n

```



```

        cmp    cx, 2
        jb     L2
        dec    cx
L1:
        mov    ax, FV
        mul    FAC
        mov    FV, ax
        inc    FAC
        loop   L1
L2:
        ret
Factorial    ENDP
END

```

#### 1.9.4. Cách dịch và liên kết

**Bước 1:** Dịch từng tệp .asm sang .obj

```

VD: C:\BT>tasm  gt1.asm → gt1.obj
      C:\BT>tasm  gt2.asm → gt2.obj

```

**Bước 2:** Gộp các tệp .obj thành 1 tệp .exe

Cú pháp:

```

tlink  tep1 + tep2 + ....+ tepn → tep1.exe
VD: C:\BT>tlink  gt1.asm + gt2.asm → gt1.exe

```

Chú ý: Khi khai báo directive điều khiển segment dạng chuẩn cho chương trình đa tệp.

<b>Tep1.asm</b>	<b>Tep2.asm</b>
Data segment PUBLIC	Data segment PUBLIC
PUBLIC n	EXTRN n:Wod
n dw ?	
Data ends	Data ends

### 1.10. Biến hỗn hợp : Directive STRUC, RECORD và UNION

#### 1.10.1 Cấu trúc STRUC

**Ý nghĩa:** xác lập 1 kiểu khai báo trong đó các thành phần có thể khác kiểu nhau.

**Cú pháp**

- Xác lập kiểu khai báo mới

```

Tên cấu trúc      STRUC
                  Các thành phần
Tên cấu trúc      ENDS

```

Ví dụ:

```

Person  STRUC
        Name    db    60 dup(?)
        Age     db    ?
        Income   dw    ?
Person  ENDS

```

Khai báo biến vừa xác lập

Sau khi 1 cấu trúc được xác lập thì tên của cấu trúc trở thành 1 kiểu khai báo biến.

.DATA

```
US_president    person<'G.BUSH', 64, 20000>
                x                               dw                ?
```

### 1.10.2 Directive UNION

Ý nghĩa: Xác lập 1 kiểu khai báo biến dùng chung vùng nhớ RAM.

Giải thích

```
.DATA
                x          db          ?
                y          dw          ?
```

Sử dụng 1 phần hard disk để lưu lại giá trị của biến.

## 1.11. Xây dựng chương trình Assembly để được tệp thực hiện dạng .COM

### 1.11.1 Sự khác nhau chương trình dạng COM và EXE

#### - Chương trình dạng .COM

Tất cả code, data, stack đều nằm trong 1 segment

#### - Chương trình dạng .EXE

Code, data, stack nằm trên các segment khác nhau.

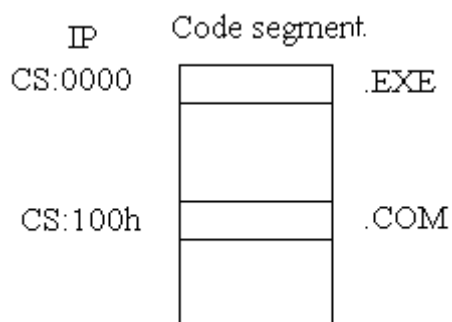
### 1.11.2 Làm thế nào để có được chương trình dạng .COM

- Từ DOS Ver5.0 trở về trước: có 1 chương trình EXE2BIN.EXE dùng để chuyển 1 tệp .EXE sang >COM

- Từ DOS Ver6.0 đến các phiên bản sau này: không có tệp EXE2BIN.EXE nên phải viết chương trình ASM có dạng đặc biệt để sau khi dịch, liên kết để chuyển sang .COM

### 1.11.3 Các vấn đề cần lưu ý

#### - Directive ORG 100h



#### - Khai báo biến

Với chương trình dạng .COM chỉ có 1 segment và đó là code segment. Vậy khai báo biến ở đâu? Khai báo biến ở code segment và được tiến hành như sau:

.CODE

Nhãn Chương trình [ jmp    Nhãn khác

Khai báo biến

Nhãn khác]

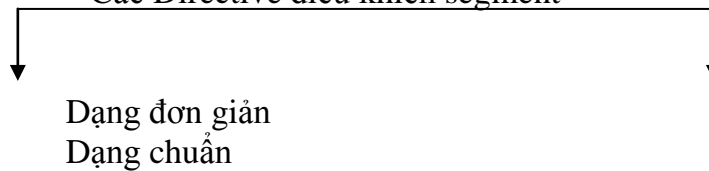
#### - Trở về DOS

(.EXE + .COM) mov ah, 4ch int 21h	(.COM) int 20h
---	-------------------

#### 1.11.4 Dạng thường thấy 1 chương trình ASM để được dạng COM

(Khai báo Marco, Struc, Union)

Các Directive điều khiển segment



<pre> .MODEL tiny .CODE     ORG 100h Nhãn CT:     [jmp Nhãn khác       Khai báo biến       Nhãn khác]       Thân CT chính      int 20h     [ Các CT con]     END Nhãn CT                 </pre>	<pre> Code segment     ORG 100h  Assume cs:code, ds:code, ss:code Nhãn CT:     [jmp Nhãn khác       Khai báo biến       Nhãn khác]       Thân CT chính      int 20h     [ Các CT con]     code ends     END Nhãn CT                 </pre>
---	--

Chú ý : khi dịch ta dùng lệnh tlink/t để dịch sang dạng .COM

Bài tập

**Bài 1:** Chia 2 số nguyên trong đó số bị chia là số nguyên, số chia là nguyên dương.

Vào số bị chia:   
 Vào số chia:   
 Thương là:   
 Có tiếp tục CT (C/K)?

```

C:\BT>edit chia.asm
INCLUDE lib1.asm
.MODEL small
.STACK 100h
.DATA
    M1 db 13,10, ' Hay vào số bị chia: $'
    M2 db 13,10, ' Hay vào số chia: $'
    M3 db 13,10, ' Thuong la: $'
    M4 db ' ': $
    M5 db '$'
    
```

```

M6    db    13,10, ' Co tiep tục CT (C/K): $'
.CODE
PS:
    mov     ax, @data
    mov     ds, ax
    clrscr
    Hienstring M1
    mov     bx,ax ;bx = so gioi han
    Hienstring M2
    call    VAO_SO_N
    xchg    ax, bx

    Hienstring M3
    and     ax, ax ; kiểm tra có phải là số âm hay ko?
    jns     L1
    Hienstring M4
    Neg     dx
    L1:
    xor     dx, ax
    div     bx
    call    HIEN_SO_N
    and     dx, dx
    jz      KT
    Hienstring M5
    mov     cx, 2
    mov     si, 10
L2:
    mov     ax, dx
    div     dx
    call    HIEN_SO_N
    and     dx, dx
    jz      KT
    loop    L2
KT:
    Hienstring M5
    mov     ah,1
    int     21h
    cmp     al,'c'
    jne     Exit
    jmp     PS
Exit:
    mov     ah,4ch
    int     21h
    Inculde lib2.asm
    END PS

```

**Dạng .COM**

```
C:\BT>edit chiacom.asm
```

```

Include    lib1.asm
.MODEL     tiny
.Code

org        100h

PS:
    Jmp     Start
        M1  db    13,10, ' Hay vào so bị chia: $'
        M2  db    13,10, ' Hay vào so chia: $'
        M3  db    13,10, ' Thuong la: $'
        M4  db    '      : $'
        M5  db    ' $'
        M6  db    13,10, ' Co tiep tục CT (C/K): $'

Start:
Clrscr
Hienstring M1
call VAO_SO_N
mov  bx,ax ;bx = so giới hạn
Hienstring M2
call VAO_SO_N
xchg ax, bx

Hienstring M3
and  ax, ax ; kiểm tra có phải là số âm hay ko?
jns  L1
Hienstring M4
Neg  dx
L1:
xor   dx, ax
div   bx
call HIEN_SO_N
and  dx, dx
jz    KT
Hienstring M5
mov  cx, 2
mov  si, 10
L2:
mov  ax, dx
div   dx
call HIEN_SO_N
and  dx, dx
jz    KT
loop L2
KT:
Hienstring M5
mov  ah,1
int  21h
cmp  al,'c'

```

```

        jne      exit
        jmp     PS
Exit:
        int     20h
        Include lib2.asm
END PS

```

**Bài 2:** Tính tổng  $\sum_{i=1}^N -i = -\sum_{i=1}^N i$

Khi chạy chương trình yêu cầu:

```

Nhập số n: 5
Tong so tu -1 den -5 la:
Co tiep tục chương trình (C/K)?

```

```

C:\BT>edit    sum.asm
Dạng .EXE
INCLUDE    lib1.asm
Stack segment
        Db          100h          dup(?)
Stack ends
Data segment
        M1    db     13,10, ' Nhập số n: $'
        M2    db     13,10, ' Tong số tu -1 den -  : $'
        M3    db     '      la : $'
        M4    db     13,10, ' Co tiep tục CT (C/K): $'
Data ends
Code segment
Assume     cs:code, ds: data, ss: stack
PS:
        mov     ax, data
        mov     ds, ax
        clrscr
        Hienstring    M1
        call    VAO_SO_N
        Hienstring    M2
        call    HIEN_SO_N
        Hienstring    M3
        mov     cx, ax
        dec     cx
L1:
        add     ax, cx
        loop    L1
        reg     ax
        call    HIEN_SO_N
        Hienstring    M4
        mov     ah, 1

```

```

        int    21h
        cmp    al, 'c'
        jne    Exit
        jmp    PS
Exit:
        mov    ah, 4ch
        int    21h
        Include lib2.asm
        Code end
        END PS
Dạng >COM
C:\BT>edit    sumcom.asm
Include      lib1.asm
.MODEL       tiny
.Code segment
            org          100h
            assume       cs:code, ds: data, ss: stack

PS:
        Jmp     Start
M1      db      13,10, 'Nhap so n: $'
M2      db      13,10, 'Tong tu -1 den - : $'
M3      db      '          la : $'
M4      db      13,10, ' Co tiep tục CT (C/K): $'
Start:
        clrscr
        Hienstring M1
        call  VAO_SO_N
        Hienstring M2
        call  HIEN_SO_N
        Hienstring M3
        mov   cx, ax
        dec   cx
L1:
        add   ax, cx
        loop  L1
        reg   ax
        call  HIEN_SO_N
        Hienstring M4
        mov   ah, 1
        int   21h
        cmp   al, 'c'
        jne   Exit
        jmp   PS
Exit:
        int   20h

```

```

        Include lib2.asm
        Code ends
    END PS
    Chú ý: dùng tlink/t

```

### Một số lưu ý khi sử dụng thanh ghi thay cho biến nhớ

- Nguyên tắc chung: cố gắng sử dụng thanh ghi thay cho biến nhớ trong trường hợp có thể, chương trình sẽ chạy nhanh hơn.
- Các loại biến: biến xâu và biến trường số (*không được dùng thanh ghi*), biến số (db, dw) dùng thanh ghi được (dd, dp, dt: không dùng).
- Các thanh ghi có thể dùng thay biến nhớ AX(ah, al), CX, BX, SI, DI, BP.
- Các thanh ghi không được phép thay biến nhớ: CS, DS, SS, IP, SP, FLAG
- Thanh ghi AX: có thể đứng làm toán hạng cho hầu hết các lệnh ASM. Ngoại lệ làm toán hạng ẩn trong các lệnh MUL/IMUL và DIV/IDIV.

Ví dụ:

```
mul    bx    ; ax*bx → dx: ax
```

Trong các lệnh IN/OUT chỉ có al có thể thực hiện hai lệnh này, không có thanh ghi nào thay thế được.

IN al, địa chỉ cổng

OUT địa chỉ cổng, al/ax

- Thanh ghi BX giống như AX ngoại trừ  
Người lập trình có thể dùng bx làm con trỏ offset (SI/DI)  
Ví dụ: lea bx, a
- Thanh ghi CX : chỉ số của lệnh loop, trong các lệnh dịch, quay với số lần lớn hơn hoặc bằng 2.

```
sar ax, 4 ≡ mov cl/cx, 4
               sar ax, cl/cx
```

- Thanh ghi DX  
Ngoại lệ: Toán hạng ẩn mul/imul và div/idiv, địa chỉ cổng khi  $\geq 256$   
Ví dụ: địa chỉ cổng COM1 là 378h  
~~IN AL, 378h~~ ≡ mov dx, 378h  
                  in al/ax
- Thanh ghi SI, DI  
Ngoại lệ: Là con trỏ offset trong các lệnh làm việc với xâu. Người lập trình ASM có thể dùng SI, DI làm địa chỉ offset của biến nhớ
- Thanh ghi BP  
Ngoại lệ: làm con trỏ offset của stack khi liên kết ngôn ngữ bậc cao với ASM khi hàm có đối.



## Chương 2: LIÊN KẾT CÁC NGÔN NGỮ BẬC CAO VỚI ASM

*Mục đích:* Tận dụng sức mạnh của các ngôn ngữ bậc cao và tốc độ của ASM.

*Cách liên kết:* Bất kỳ một ngôn ngữ bậc cao nào liên kết với ASM đều phải tuân theo 2 cách sau:

Cách 1: Inline Assembly.

cách 2: Viết tách tệp của ngôn ngữ bậc cao và tệp của ASM

### 2.1 Liên kết Pascal với ASM

#### 2.1.1 Inline ASM

*Cơ chế:* Chèn khối lệnh ASM vào chương trình được viết bằng Pascal.

*Cú pháp:*

```
    Các câu lệnh Pascal
    ASM
        các câu lệnh ASM
    end;
```

Các câu lệnh Pascal

*Ví dụ:* So sánh 2 số và hiện số lớn hơn ra màn hình.

SS.Pas

*Uses* crt;

*Label* L1

Var

*s1,s2 :Integer;*

Begin

*clrscr;*

*write ('nhap so thu nhât : '); readln(s1);*

*write ('nhap so thu hai : '); readln(s2);*

ASM

*mov ax,s1*

*mov bx,s2*

*cmp ax,bx*

*jg ll*

*xchg ax,bx*

*ll:*

*mov s1,ax*

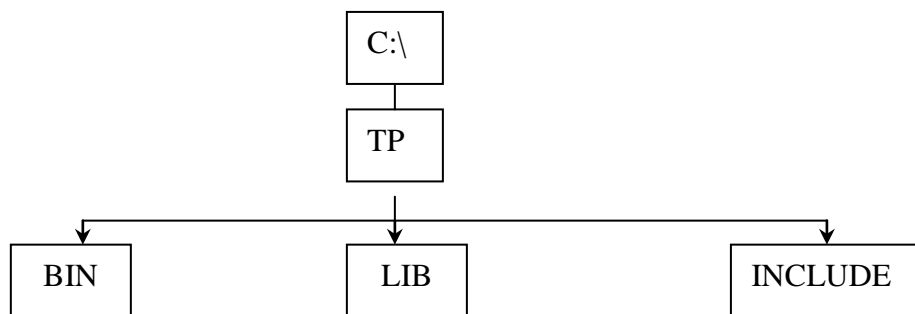
*end;*

*write ('So lon hon la : ', s1:5);*

*readln;*

*end.*

Cách dịch và liên kết:



TP.exe: Đây là chương trình dịch của TP với các tùy chọn được xác lập bởi menu options.

TPC.exe: Đây là chương trình dịch của TP với các tùy chọn được xác lập trên dòng lệnh dịch.

Cú pháp: **tpc -ml -lc:\tp\include -Lc:\tp\lib ss**

Ưu điểm: Rất dễ liên kết và viết.

Nhược điểm: Các lệnh ASM được dịch nhờ bởi chương trình dịch của TP có sai sót.

### 2.1.2 Viết tách biệt tập ngôn ngữ Pascal và tập ASM

Các vấn đề nảy sinh cần giải quyết: có 4 vấn đề

**Vấn đề 1:** Đa tệp do đó phải khai báo PUBLIC và EXTRN với các nhãn dùng chung.

*Khái báo Pascal:*

Bất kể một khai báo nào của Pascal đều là Public do đó không cần phải khai báo tường minh public.

Với các nhãn là biến nhớ thì Pascal luôn giành lấy để khai báo Public

Với các nhãn là tên chương trình con thì ASM viết chương trình con nên Pascal sẽ sử dụng chương trình con -> Pascal phải xin phép sử dụng như sau:

- Chương trình con là thủ tục: Procedure tên\_thủ\_tục [đối]; external;
- Chương trình con là hàm: Function tên\_hàm [đối]: Kiểu; external;

*Khai báo của ASM:*

Giống như đa tệp thuần túy ASM

- Với nhãn là tên biến nhớ:

.Data extrn tên\_biến\_nhớ : kiểu

Kiểu của ASM	TP
Byte	Char
Word	Integer
DWord	Real

- Với nhãn là tên chương trình con :

.Code

Public tên\_chương\_trình\_con

tên\_chương\_trình\_con Proc

:

Ret

tên\_chương\_trình\_con endp

## **Vấn đề 2:** Vấn đề near/far của chương trình con

Quy định chung của chương trình dịch TP

- Nếu chương trình con cùng nằm trên 1 tệp với chương trình chính hoặc chương trình con nằm ở phần implementation của Unit thì chương trình con đó là near.
- Nếu chương trình con nằm ở phần Interface của Unit thì chương trình đó là far.

*Ngoại lệ:*

- Directive {\$F<sup>+</sup>}: Báo cho chương trình dịch TP biết chương trình con nào nằm sau Directive {\$F<sup>+</sup>} là far.
- Directive {\$F<sup>-</sup>}: Báo cho chương trình dịch của TP biết những chương trình con nào nằm sau Directive {\$F<sup>-</sup>} phải tuân thủ quy định chung của chương trình dịch TP

## **Vấn đề 3:** Cách chương trình dịch TP tìm tệp để liên kết:

Directive {\$L}

Cú pháp : {\$L tên\_tệp [.obj]}

## **Vấn đề 4:** Tên hàm ASM mang giá trị quay về

Muốn tên hàm ASM mang giá trị quay về dạng 2 byte phải đặt giá trị đó vào thanh ghi Ax trước khi có lệnh Ret.

Muốn tên hàm mang giá trị 4 bytes thì phải đặt giá trị đó vào thanh ghi DX:AX trước khi có lệnh Ret.

Nhận xét:

Người viết Pascal quan tâm đến vấn đề: 1, 2, 3.

Người viết ASM quan tâm đến vấn đề: 1, 4.

### **Phương pháp 1: Chương trình con không đối. Chuyển giao tham số thông qua khai báo biến toàn cục.**

Ví dụ: Tính  $a^n$ .

vd1.pas

- Nhập giá trị a, n
- Gọi chương trình con tính  $a^n$  do asm viết
- Hiện kết quả.

vd2.asm: chương trình tính  $a^n$

vd1.pas

Uses crt;

Var

a,n: Integer

{ \$F+ }

function a\_mu\_n: integer; external;

{ \$L vd2 [.obj] }

{ \$F }

Begin

clrscr;

writeln(' Chương trình tính a mu n !');

write ('Nhập số a: '); readln(a);

write ('Nhập số n: '); readln(n);

write (a, 'lấy thừa ', n, ' là : ', a\_mu\_n : 5 );

readln;

End.

vd2.asm

.model large

.data

EXTRN a:word, n:word

.code

Public a\_mu\_n

a\_mu\_n proc

mov bx,a

mov cx,n

mov ax,1

and cx,cx

```

        jz      kt
lap:
        imul    bx
        loop   lap
kt:
        ret
a_mu_n   endp
end

```

Cách dịch và liên kết

b1: Dịch tệp .asm sang .obj

c:\asm> tasm vd2 -> vd2.obj

b2: Dịch .pas và liên kết

C:\asm> tpc -ml vd1 -> vd1.exe

## Phương pháp 2: Chương trình con có đối. Chuyển giao tham số thông qua Stack

*Nguyên lý:* Chúng ta đều biết chương trình con không ASM không có đối. Tuy nhiên khi liên kết Pascal với ASM thì Pascal giả thiết chương trình con ASM có đối. Số lượng đối và kiểu đối do Pascal giả thiết. Với giả thiết đó khi gọi chương trình con, Pascal phải đưa tham số thực vào Stack (theo chiều từ trái qua phải).

```

Cơ chế:  function      test(b1:integer, b2:integer, b3: integer): integer; external;
          :
          test (a,b,c)

```

Bước1: Tham số thực đưa vào Stack theo chiều từ phải qua trái

Bước 2: Địa chỉ lệnh tiếp theo đưa vào Stack (4 byte)

Bước 3: Hệ điều hành đưa địa chỉ đầu của chương trình con ASM vào CS:IP -> chuyển sang chương trình con .

```

.model    large
.code
    Public test
Test      Proc
    Push  bp
    mov  bp,sp
    Thân chương trình con ASM
    pop  bp
    ret  n      ; n là số lượng byte mà tham số thực chiếm trong Stack.
Test      endp

```

Ví dụ: Tính  $a^n$  đối với hàm có đối

lt1.pas

Uses crt;

Var a,n : integer;

{ $F^+$ }

```

    function lt( b1: integer, n2: integer): Integer; External;
{$L lt2}
{$F}
Begin
    clrscr;
    write('Nhap so a: '); readln(a);
    write ('Nhap so n: '); readln(n);
    write ('ket qua la: ' lt(a,n): 5);
    readln;
End.

```

lt2.asm

.model large

.code

Public lt

lt Proc

push bp

mov bp,sp

mov bx,[bp + 8]

mov cx,[bp + 6]

mov ax,1

and cx,cx

jz kt

lap:

imul bx

loop lap

kt:

pop bp

ret 4

lt endp

end

Dịch như sau:

Tasm lt2 -> lt.obj

Tcp -ml lt1 ->lt1.exe

### **Bài tập: Trung bình cộng 2 số**

Cách1: Hàm không đổi

TBC.asm

Uses crt;

Var s1,s2,flag : Integer;

{\$F+}

```

    function      tb(): Integer; external;
{$L tbc2}
{$F}
Begin
    clrscr;
    flag := 0;
    Write ( ' Nhập số thu nhất: ');      readln(s1);
    Write( ' Nhập số thu hai: ');        readln(s2);
    Write( ' Trung bình cộng 2 số là: ', 0.5*flag + tb:5);
    readln;

```

End.

tbc2.asm

.model large

.data

extrn s1: word , s2: word, flag: word

.code

public tb

tb proc

mov ax,s1

mov bx,s2

add ax,bx

sar ax,1

jnc ll

mov flag,1

L1: ret

tb end

End

Cách 2: Hàm có 3 đối

TBC.asm

Uses crt;

Var s1,s2, flag : Integer;

{\$F<sup>+</sup>}

function tb(f:integer, n1: integer, n2:Integer): Integer; external;

{\$L tbc2}

{\$F}

Begin

clrscr;

flag := 0;

Write ( ' Nhập số thu nhất: '); readln(s1);

```

Write(' Nhap so thu hai: ');      readln(s2);
Write(' Trung binh cong 2 so la: ', 0.5*flag + tb(flag,s1,s2):5);
readln;
End.
tbc2.asm
.model      large
.code
    public tb
tb  proc
    push    bp
    mov     bp,sp
    mov     ax,{bp+8}
    mov     bx,{bp+6}
    add     ax,bx
    sar     ax,1
    jnc     ll
    mov     cx,1
    mov     {bp + 10},cx
L1:
    pop     bp
    ret     6
tb  end
End

```

### **Bài tập 1:** Tính tổng của dãy số nguyên

Trong đó: Pascal

- Nhận số lượng các thành phần
- Nhận các số của mảng
- Hiện các số của mảng ra màn hình
- Gọi ctc tính tổng do ASM tính
- Hiện tổng

ASM: Viết chương trình con tính tổng

Giải

Viết một chương trình pascal T1.pas

```

uses      crt;
label     L1;
type      //cho phép khai báo xác lập kiểu khai báo biến mới
m = array [1..100] of Integer;
Var
    sltp, i: Integer;

```



```

a:      m;
tl:     char;
{$F+}           //báo hàm xấp khai báo la far
function      sum(mang:m, n:integer): Integer // do ASM thực hiện
{$L T2}        //hàm đó nằm ở file T2.obj
{$F}           //các hàm dùng sau theo chuẩn P
Begin
  L1:
  clrscr;
  Write ('nhap so thanh phan sltp = ': );   readln(sltp);
  write('nhap vao day cua cac thanh phan');
  for   I:=1 to sltp do      begin
                                write ('a[' ,I,']= '); readln(a[i]);
                                end
  write (' Day so vua nhap vao la: ');
  for I:= 1 to sltp do write(a[i], ' ');
  writeln;
  write('co tiep tục không C/K ? ');
  tl := readkey;
  if (tl='c') then goto L1;
  readln;
END.

```

#### T2.ASM

```

.MODEL      large
.code
  public sum
sumproc  //a: d/c cuar a0 dc đưa vào stack mat 4 byte do offset+seg, cat vaof theo
          //chiều từ trái qua phải,
  push  bp
  mov   bp,sp
  mov   cx,[bp+6]
  les   bx,[bp+8]
//lay 2 byte đưa vào BX và 2 byte tiếp theo vào ES
  xor   ax,ax
lap:
  add   ax,es:[bx]
  add   bx,2
  loop lap
  pop   bp

```

```
ret    6    //tra lai 6 byte 4 byte cho a, 2 byte cho sltp
end
```

Dịch và liên kết:

b1: Dịch ASM sang .OBJ  
 c:\tuan t2 -> T2.obj  
 T2.obj nằm ở {\$L T2}.  
 b2: Dịch và liên kết P  
 c:\tuan>tpc -ml t1 ->t1.exe

Sử dụng directive ARG

**Lý do:** cho phép người viết chương trình con ASM (trong trường hợp có đối) viết đúng chương trình con mà không biết cấu trúc của Stack.

**Cú pháp:** tên chương trình con PROC

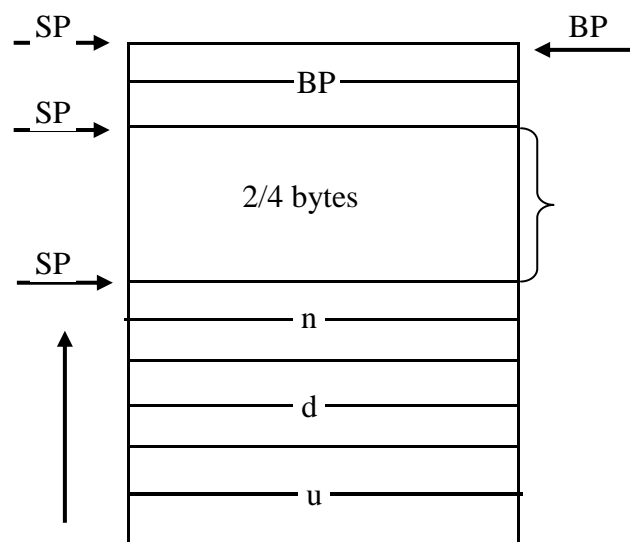
ARG tên đối : kiểu = Retbytes (tên đối được sắp xếp từ phải sang trái)

## Bài tập 2: Tính tổng cấp số cộng khi biết n, d, u1

```
Pascal:      csc1.pas
Uses      crt;
Var n,d,u1:Integer;
{$F+}
function      csc(n1: integer, n2: integer, n3: integer ):integer; external;
{$L csc2} //tìm ở tệp csc2.obj, không có đường dẫn thì ở thư mục hiện hành
{$F}        //báo theo chuan P
Begin
    write ('nhap vao n = '); readln(n);
    write('nhap vao d = '); readln(d);
    write('nhap vao u1 = '); readln(d);
    write('tong cap so cong = ', csc(n,d,u):5);
End.
```

Viet ASM: csc2.asm (không dùng directive)

```
cach1:
.model      large
.code
    public csc
csc proc
    push bp
    mov      bp,cs
    mov      ax,[bp+6]
    mov      bx,[bp+8]
    mov      cx,[bp+10]
    mov      dx,ax
```



```

dec cx
lap:
add     dx,bx
add     ax,dx
loop lap
pop bp
ret 6
csc endp
end
cach2:
.model   large
.code
    public csc
csc proc
    ARG n3:word, n2:word, n1:word= Retbytes
    push bp
movbp,cs
movax,n3
movbx,n2
movcx,n1
movdx,ax
dec cx
lap:
add dx,bx
add ax,dx
loop lap
pop bp
ret Retbytes
csc endp
end

```

## 2.2 Liên kết c/c<sup>++</sup> với ASM

### 2.2.1. Inline Assembly

**Cơ chế:** Chèn khối lệnh ASM vào chương trình được viết bằng C/C<sup>++</sup>

**Cú pháp:**

```

    Các câu lệnh C
    ASM lệnh ASM
    ASM lệnh ASM
    ASM lệnh ASM
    Các câu lệnh C

```

hoặc cách khác:

Các câu lệnh C

```
ASM {           // dấu ngoặc phải cùng một dòng
    khối lệnh ASM
}
```

Các câu lệnh C

**Ví dụ**      Tính Tổng 2 số nguyên  
                 Tong.C

```
#include <stdio.h>
```

```
#include <conio.h>
```

```
int s1,s2
```

```
Void Main(void) //nếu hàm f() không có giá trị trả về thì ct sẽ mặc định là int
{
```

```
clrscr();
```

```
printf("\n nhap vao so thu nhat : "); scanf("%d",&s1);
```

```
printf("\n nhap vao so thu hai : "); scanf("%d",&s2);
```

*//nếu không có format thì không báo lỗi và cũng không hiện ra màn hình*

```
ASM {
```

```
mov ax,s1
```

```
mov bx,s2
```

```
mov ax,bx
```

```
mov s1,ax
```

```
}
```

```
printf("\n tong cua hai so la %d ", s1);
```

```
getch();
```

```
}
```

Dịch và liên kết :

Giả sử chúng ta cất giữ file trong thư mục ASM

```
C:\ASM>tcc -ms -IC:\tc\include -LC:\tc\lib tong.c
```

*//phần trên có thể dịch được nếu chúng ta đã khai báo trong Autobot thì bất cứ ở đâu cùng gọi được tcc. còn không chúng ta phải viết như sau:*

```
C:\ASM>c:\tc\bin\tcc -ms -IC:\tc\include -LC:\tc\lib tong.c
```

*uđ:*            Dễ liên kết

*Nhược điểm:*

- Khối lệnh ASM được dịch nhờ bởi TC -> không chuẩn
- Không cho phép có nhãn nhảy trong khối lệnh ASM được chèn vào C -> khối lệnh chèn vào không linh hoạt và không mạnh.

### 2.2.2 Viết tách biệt C/C++ và tệp ASM

Một số vấn đề nảy sinh cần giải quyết khi viết tách biệt, có 3 vấn đề

**Vấn đề1:** (đa tệp)

Chúng ta phải liên kết các file với nhau do đó chúng ta phải khai báo Public và External với các nhãn dùng chung.

*Khai báo trong C/C++*

PUBLIC: Bất kỳ một khai báo nào của C/C++ đều là Public, nên không cần khai báo tường minh. Với nhãn là biến nhớ cho phép ASM khai báo Public và c/c++ xin phép được dùng

cú pháp:

Extern	kiểu	tên biến	ASM
	char		db
	Int		dw
	float		dd

EXTERNAL: Khai báo để được phép dùng chương trình con của ASM

Extern kiểu tên hàm ([đối]);

**Khai báo của ASM: Giống như đa tệp thuần túy**

## Vấn đề 2

Người viết ASM phải thêm dấu ‘\_’ vào trước các nhãn dùng chung với C/C++ và thêm ở mọi nơi mà nhãn đó xuất hiện. vì dùng C khi dịch các nhãn ở ngoài nó đều thêm ‘\_’ vào trước nhãn.

**Vấn đề 3** Tên hàm ASM mang giá trị quay về AX, DX:AX tương ứng 2,4 byte

## Phương pháp 1 (Hàm không đối)

Chúng ta phải chuyển giao tham số thông qua biến toàn cục

*Ví dụ* Tính giai thừa của n!

C: Nhập n; Gọi chương trình con tính n! do ASM tính; Hiện kết quả

ASM: Viết chương trình con tính n!

gtn1.c

#include <stdio.h>

#include <conio.h>

int n

extern int gt();

Void main(void)

{

clrscr();

printf("\n Nhập vào n = "); scanf("%d", &n);

printf("\n %d Giai thừa là : %d", n, gt():5);

getch();

}

// biến toàn cục sẽ cất trong Data, biến cục bộ cất trong Stack, static biến cục bộ cất trong Data

gtn2.asm

.model small/large // -ms/-ml

```

.data
    extrn _n: Word
    a      dw      ?
    b      dw      ?
.code
    public _gt
_gt proc
    mov     a,1
    mov     b,2
    mov     cx,_n
    cmp     cx
    jb      exit
    dec     cx
lap:
    mov     ax,a
    mul     b
    mov     a,ax
    inc     b
    loop    lap
exit:
    mov     ax,a
    ret
_gt endp
end

```

Dịch liên kết file

```
tcc ms/ml Ic:\tc\include -Lc:\tc\lib gtn1 gtn2.asm -> gtn1.exe.
```

## Phương pháp 2 (Hàm có đối)

Hàm có đối thì chương trình phải chuyển giao tham số thông qua Stack.

**Lý do:** Chúng ta biết chương trình con thuần túy ASM không có đối. tuy nhiên khi C/C++ liên kết với ASM thì nó giả thiết chương trình con ASM có đối. Số lượng đối, kiểu đối do C/C++ giả thiết và với những giả thiết đó thì chương trình con ASM. C/C++ đưa tham số thực vào Stack và người viết chương trình con ASM phải vào Stack lấy giá trị đó.

*Giải thích*

```

extern     int     test (int n1, int n2, int n3);
Void main (void)
{
    int     a,b,c
    -
    test (a,b,c);

```

```

-
}
có 5 bước :
.model    small
[.data]
.code
    public _test
_test     proc
    push  bp
    mov   bp,sp
    các lệnh ASM
    pop   bp
    ret
_test     endp
end

```

Bài tập 3      tính  $n!$  hàm có 1 đối

```

                                gtn1.c
#include <stdio.h>
#include <conio.h>
int n
extern    int    gt(int i);
Void      main(void)
{
    clrscr();
    printf("\n Nhập vào n = ");          scanf("%n", &n);
    printf("\n %d Giai thừa là : %d",n , gt(n):5);
    getch();
}

```

```

                                gtn2.asm
.model    small/large          //-ms/-ml
.data
    a      dw    ?
    b      dw    ?
.code
    public _gt
_gt proc
    push   bp
    mov    bp,sp
    mov    a,1

```

```

    mov    b,2
    mov    cx,_n
    cmp    cx
    jb     exit
    dec    cx
lap:
    mov    ax,a
    mul    b
    mov    a,ax
    inc    b
    loop   lap
exit:
    mov    ax,a
    pop    bp
    ret
    _gt    endp
end

```

#### **Bài tập 4** Tính trung bình cộng 2 số nguyên

Cách 1: Hàm không có đối

- S1,S2, flag là các biến toàn cục.
- Tên Hàm ASM -> trung bình cộng làm tròn dưới.

TBC1.C

```

#include <stdio.h>
#include <conio.h>
int s1,s2,flag = 0;
extern int tbc();
Void main(Void)
{
    Printf("\n nhap vao so thu 1 : ");    scanf("%d",&s1);
    Printf("\n nhap vao so thu 2 : ");    scanf("%d",&s2);
    printf( "\n Trung binh cong cua 2 so nguyen la: %d", tbc()+0.5*flag);
    getch();
}
// chú ý ngôn ngữ C phân biệt chữ hoa và chữ thường.

```

TBC2.ASM

```

.model    small
.data
    extrn _s: Word, _s2: Word, flag: Word
.code

```



```

    public _tbc
_tbc    proc
    mov  ax,_s1
    mov  bx,_s2
    add  ax,bx
    sar  ax,1
    jnc  exit
    mov  cx,1
    mov  _flag,cx
exit:
    ret
_tbc    endp
end

```

Cách 2:

- s1,s2 là biến cục bộ -> trong Stack
- flag là biến toàn cục
- hàm tính trung bình cộng là làm tròn dưới.

TBC1.C

```

#include <stdio.h>
#include <conio.h>
int flag = 0;
extern int tbc(int n1, int n2);
Void main(Void)
{
    int s1,s2;
    Printf("\n nhap vao so thu 1 : ");    scanf("%d",&s1);
    Printf("\n nhap vao so thu 2 : ");    scanf("%d",&s2);
    printf( "\n Trung binh cong cua 2 so nguyen la: %d", tbc()+0.5*flag);
    getch();
}

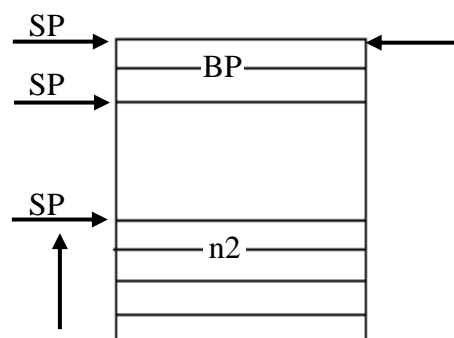
```

TBC2.ASM

```

.model    small
.data
    extrn flag: Word
.code
    public _tbc
_tbc    proc
    push  bp
    mov  bp,sp

```



```

mov ax,[bp+4]
mov bx,[pb+6]
add ax,bx
sar ax,1
jnc exit
mov cx,1
mov _flag,cx
exit:
pop bp
ret
_tbc endp
end

```

**Bài tập 5:** Sắp xếp dãy số theo chiều tăng dần.

C:

- Nhận số lượng thành phần
- Nhận các số đưa vào một mảng
- Hiện các số của mảng ra màn hình
- Gọi chương trình con sắp xếp do ASM viết
- Hiện các số đã sắp xếp

ASM: Viết chương trình con sắp xếp dãy số.

Giải

```

#include<stdio.h>
#include<conio.h>
extern sx(int n, int far* mang);
Void main(void)
{
    int sltp, a[100];
    clrscr();
    printf("\n Nhap vao sltp = "); scanf("%d",&sltp);
    printf("\n Nhap vao cac so cua mang");
    for (I=0, I<sltp, I++)
    {
        printf("\n a[%d]=", i); scanf("%d",&a[i]);
    }
    printf("\ day so vua nhap vao la");
    for(I=0, I<sltp, I++) printf("%d ", a[i]);
    sx(sltp,a);
    printf("\ day so da sap xep la");
    for(I=0, I<sltp, I++) printf("%d ", a[i]);
}

```

```

    getch();
}

sx2. asm
.model small
.code
    public _sx
_sx proc
    push    bp
    mov     bp,sp
    mov     si,[bp+4]
    dec     si
l1:
    mov     cx,[bp+4]
    les     bx,[bp+6]
    dec     cx
l2:
    mov     ax,es:[bx]
    mov     dx,es:[bx+2]
    cmp     ax,dx
    jl      l3
    mov     es:[bx+2],ax
    mov     es:[bx],dx
l3:
    add     bx,2
    loop    l2
    dec     si
    jne     l1
    pop     bp
    ret
_sx endp
end

```

**chú ý:**                      **Directive**      **ARG**

Cú pháp	ARG	tên đối	Kiểu
//	C ngược với P là từ trái qua phải		

### Liên kết C++ với ASM

Giống C liên kết với chương trình con trừ một vấn đề tên chương trình con ASM.

```

C:
    .code
        public _tên chương trình con

```

```

    _tên chương trình con      proc
        các câu lệnh của ASM
    ret
    _tên chương trình con      endp
End

```

C++:

```

.code
    public @tên chương trình con $...
    @tên chương trình con $...
        các câu lệnh ASM
    ret
    @tên chương trình con $... endp

```

Các bước:

b1: Viết modul C++ .cpp

b2: Dịch từ đuôi .cpp ra .asm

**tcc -S tên\_tệp.cpp ->tên\_tệp.asm**

b3: Hiện lên màn hình tên tệp .asm ( ở dòng cuối cùng có @tên chương trình con \$...)

**Bài tập 6:** So sánh 2 số và hiện số bé

SS1.CPP

```

#include<iostream.h>
#include<conio.h>
extern int ss(int n1, int n2);
Void      main(void)
{
    int    s1,s2
    clrscr();
    cout<<"\n nhap so thu nhât: ";   cin>>s1;
    cout<<"\n nhap so thu hai : ";   cin>>s2;
    cout<<"\n So be la: "; << ss(s1,s2);
    getch();
}

```

Sau khi viết xong ta dịch

**tcc -S ss1.cpp -> ss1.asm**

Hiện ss1.asm lên màn hình: -> @ss\$...

```

.model    small
.code
    public @ss$...
    @ss$...    proc

```

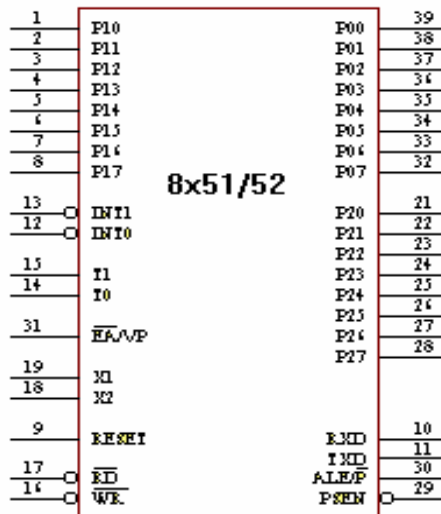
```
    push    bp
    mov     bp,sp
    mov     ax,[bp+4]
    mov     bx,[bp+6]
    cmp     ax,bx
    jl      ll
    xchg    ax,bx
ll:
    pop     bp
    ret
@ss$...   endp
End.
```

## Chương 3: LẬP TRÌNH HỢP NGỮ CHO 8051/55

### 3.1. Giới thiệu chung về họ vi điều khiển 8051

Phân biệt chip vi xử lý và chip vi điều khiển

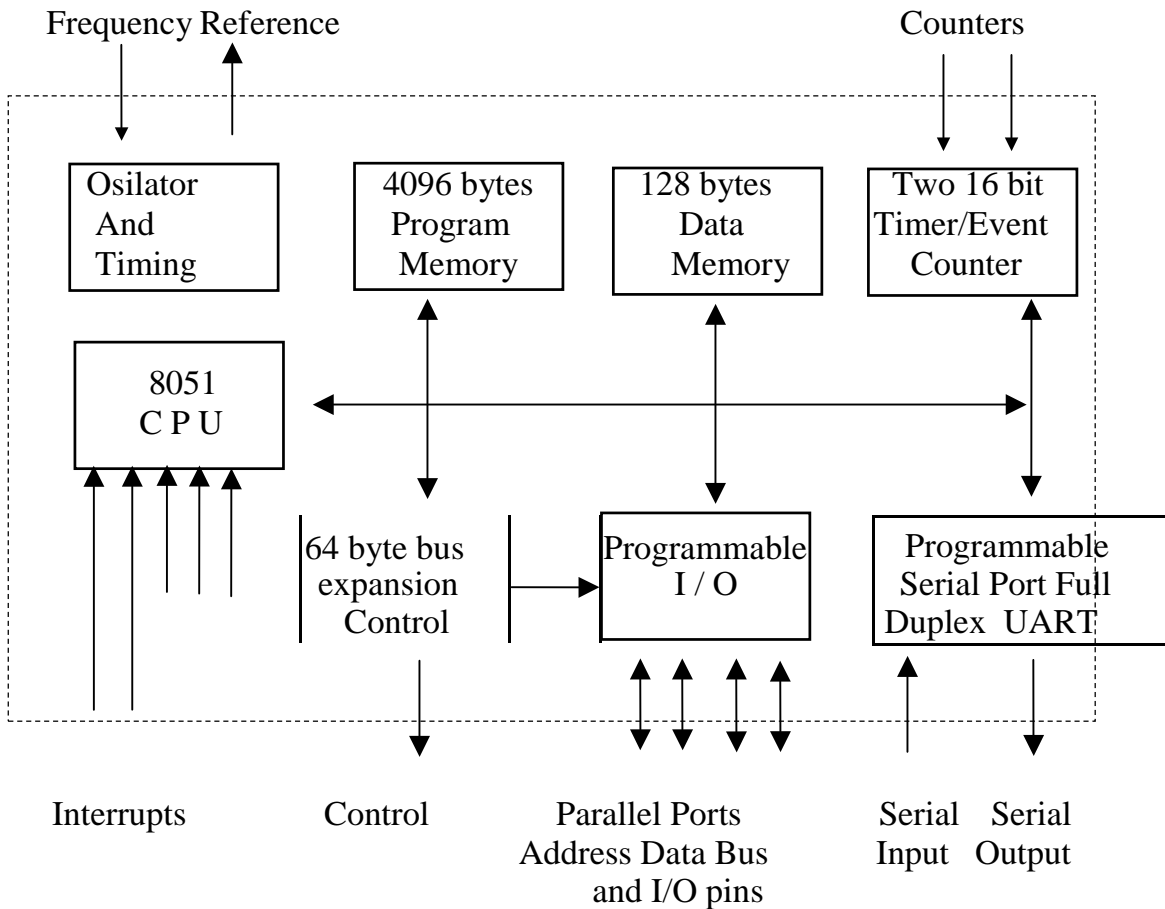
- Bộ vi xử lý: Làm chức năng xử lý và tính toán. Chỉ riêng một chip vi xử lý thì không thể tạo nên một hệ thống có khả năng tính toán và xử lý (hệ vi xử lý) mà còn cần thêm các chip nhớ, chip ngoại vi, truyền tin, ... Chip vi xử lý ví dụ, 8086, Pentium I, II, III, IV...
- Chip vi điều khiển: Khác với các chip vi xử lý chip vi điều khiển đã là một hệ thống tính toán trong một chip (System on chip). Trong một chip vi điều khiển ngoài CPU ra đã có sẵn bộ nhớ ROM, RAM, các cổng vào ra song song, nối tiếp, cổng truyền tin, các bộ đếm - định thời gian... giúp cho người sử dụng thuận tiện và dễ dàng khi ứng dụng vi điều khiển giải quyết các bài toán điều khiển trong thực tế. Một loại vi điều khiển rất thông dụng hiện nay là vi điều khiển họ 8x51/52 như 8031, 8032, 8051, 8052, 8951, 8952... và các vi điều khiển thế hệ sau như PIC16F84. Các vi điều khiển thế hệ mới AVR như 90S8515, 90S8535... với nhiều tính năng ưu việt hơn, tốc độ xử lý nhanh hơn 8051 là vi điều khiển đầu tiên của họ vi điều khiển được Intel chế tạo. Đặc tính kỹ thuật cơ bản như sau :



- Là vi điều khiển 8 bit
- Khả năng địa chỉ hoá:
  - +64K bộ nhớ chương trình
  - +64K bộ nhớ dữ liệu
- Có 128 bytes nhớ RAM trong
- Có 2 Time/Counters
- 1 Cổng nối tiếp, 4 cổng vào ra song song
- Có bộ điều khiển ngắt logic với 5 nguồn ngắt
- 22 thanh ghi có chức năng đặc biệt SFR (Special function)

registers) 8051 có thể đánh địa chỉ 64K bộ nhớ dữ liệu ngoài và 64K bộ nhớ chương trình ngoài.

### Sơ đồ khối của vi điều khiển 8051



Vi điều khiển 8051 có cổng nối tiếp nằm trên chip. Chức năng quan trọng của cổng nối tiếp là biến đổi dữ liệu từ song song thành nối tiếp để đẩy lên đường truyền và biến đổi dữ liệu vào từ nối tiếp thành song song.

Việc truy nhập phần cứng của cổng nối tiếp thông qua các chân TxD và RxD của 8051 và đó cũng là 2 bit của Port 3:

P3.1 (TxD) là chân 11

P3.0 (RxD) là chân 10

Cổng nối tiếp của 8051 có thể truyền 2 chiều đồng thời (*full duplex*) và ký tự có thể được nhận và lưu trữ trong bộ đệm trong khi ký tự thứ 2 đã được nhận và nếu CPU đọc ký tự thứ nhất trước khi ký tự thứ hai được nhận thì dữ liệu không bị mất.

Có 2 thanh ghi chức năng đặc biệt (Special Function Register) để phần mềm qua đó truy nhập cổng nối tiếp là SBUF và SCON. SBUF (Serial port buffer) có địa chỉ 99h được xem như 2 buffer. Khi ghi dữ liệu vào SBUF là truyền dữ liệu còn khi đọc dữ liệu từ SBUF là nhận dữ liệu từ đường truyền. SCON (Serial port Control register) có địa chỉ 98h là thanh ghi có thể đánh địa chỉ theo từng bit bao gồm bit trạng thái và bit điều

hiển. Bit điều khiển xác lập chế độ điều khiển cho cổng nối tiếp và bit trạng thái cho biết ký tự được truyền hay là được nhận Bit trạng thái được kiểm tra bằng phần mềm hoặc lập trình để gây ra ngắt.

8051 có 4 chế độ hoạt động, việc chọn chế độ hoạt động bằng cách xác lập các bit SM0 và SM1 của thanh ghi SCON. Có 3 chế độ hoạt động truyền không đồng bộ, mỗi ký tự truyền hoặc nhận theo từng khung với bit start và stop tương tự RS232 của máy vi tính. Còn chế độ thứ 4 hoạt động như 1 thanh ghi dịch đơn giản.

a. Thanh ghi dịch 8 bit (Mode 0).

Mode 0 được chọn bằng cách ghi 0 vào bit SM0 và SM1 của thanh ghi SCON, xác lập cổng nối tiếp hoạt động như thanh ghi dịch 8 bit. Dữ liệu vào và ra nối tiếp qua RxD và TxD theo nhịp đồng hồ. 8 bit được truyền và nhận với bit thấp nhất (least significant :LSB) được truyền đầu tiên. Tốc độ baud được đặt bằng 1/12 tốc độ đồng hồ. Trong chế độ này ta không nói đến RxD và TxD. Đường RxD được dùng cho cả truyền và nhận dữ liệu còn đường TxD được dùng cho tín hiệu clock

Việc truyền dữ liệu được khởi đầu bằng cách ghi dữ liệu vào thanh ghi SBUF, dữ liệu được dịch ra từng bit ra ngoài qua chân RxD (P3.0) cùng với xung đồng hồ được gửi ra ngoài qua đường TxD (P3.1). Mỗi bit được truyền qua RxD trong 1 chu kỳ máy.

Việc nhận dữ liệu được khởi đầu khi bit cho phép nhận (REN) được xác lập lên bit 1 và bit RI phải xoá về 0. Nguyên tắc chung là xác lập bit REN lúc bắt đầu chương trình để khởi tạo các tham số của cổng nối tiếp và xoá bit RI để bắt đầu công việc nhận dữ liệu. Khi RI được xoá, xung đồng hồ được ghi ra chân TxD, bắt đầu chu kỳ máy tiếp theo và dữ liệu được đưa vào chân RxD.

Một khả năng của chế độ thanh ghi dịch là có thể mở rộng đường ra của 8051. Thanh ghi dịch chuyển đổi nối tiếp thành song song có thể kết nối với các đường TxD và RxD cung cấp thêm 8 đường ra.

b. 8 bit UART với tốc độ baud có thể thay đổi được (mode1).

ở mode 1 cổng nối tiếp của 8051 hoạt động như là UART 8 bit với tốc độ baud có thể thay đổi được. UART (Universal Asynchronous Receiver/Transmitter) là 1 thiết bị nhận và truyền dữ liệu nối tiếp, mỗi ký tự được truyền bắt đầu bằng bit start(trạng thái thấp ) sau đó là các bit dữ liệu của ký tự được truyền, parity bit để kiểm tra lỗi đường truyền và cuối cùng là bit stop (trạng thái cao).

Một chức năng quan trọng của UART là chuyển đổi dữ liệu song song thành nối tiếp để truyền và chuyển đổi nối tiếp thành song song để nhận.

Trong mode này, 10 bit dữ liệu được truyền qua TxD và nhận vào qua RxD và việc truyền cũng tương tự như mode 0, start bit luôn bằng 0 sau đó đến 8 bit dữ liệu (LSB đầu tiên ) và cuối cùng là stop bit. Bit TI của SCON được lập bằng 1 khi bit stop ở chân TxD. Trong quá trình nhận stop bit được đưa vào bit RB8 của thanh ghi SCON và tốc độ truyền được đặt bởi timer1. Việc đồng bộ thanh ghi dịch của cổng nối tiếp ở mode 1, 2, 3 được điều khiển bởi Counter với đầu ra counter là nhịp đồng hồ điều khiển tốc độ baud



còn đầu vào của counter được chọn bằng phần mềm.

c.9 bit UART với tốc độ baud cố định (mode 2).

Mode được chọn bằng cách đặt bit SM1=1 và SM0=0, cổng nối tiếp của 8051 sẽ hoạt động như UART 9 bit có tốc độ baud cố định. 11bit sẽ được truyền và nhận qua TxD và RxD: 1 start bit, 9bits dữ liệu và stop bit. Trong khi truyền bit thứ 9 sẽ được đặt vào TB8 của thanh ghi SCON còn trong khi nhận bit thứ 9 sẽ được đặt vào bit RB8. Tốc độ baud của mode 2 có thể là 1/32 hoặc 1/64 xung nhịp đồng hồ. d.9

bit UART với tốc độ baud có thể thay đổi được (mode 3).

Mode 3 tương tự như mode 2 nhưng tốc độ baud được lập trình và được cung cấp bằng timer. Trong thực tế cả 3 mode 1, 2, 3 là tương đương nhau chỉ khác nhau ở chỗ tốc độ baud ở mode 2 là cố định còn mode 1 và mode 3 có thể thay đổi được và số lượng bit dữ liệu của mode 1 là 8 bit còn mode 2 và mode 3 là 9 bit.

- Cho phép nhận (Receiver enable)

Bit cho phép nhận (REN) của thanh ghi SCON phải được lập bằng phần mềm để cho phép nhận dữ liệu. Việc này được tiến hành lúc bắt đầu chương trình khi cổng nối tiếp, timer/counter,... được khởi tạo. Có 2 cách xác lập bit này:

SETB REN hoặc MOV SCON,#xxx1xxxxb

(Trong đó x có thể là 0 hoặc 1 tùy theo yêu cầu của chương trình).

- Cờ ngắt (Interrupt flag)

Hai bit RI và TI trong thanh ghi SCON được lập lên 1 bằng phần cứng và phải được xóa về 0 bằng phần mềm. RI được xác lập khi bit cuối cùng của dữ liệu được nhận và nó cho biết rằng đã kết thúc truyền 1 byte dữ liệu (receiver buffer full) nó được test bằng chương trình để gây ra ngắt. Nếu chương trình muốn nhận dữ liệu từ 1

thiết bị nối với cổng nối tiếp nó phải chờ cho đến khi RI được lập sau đó xoá RI và đọc dữ liệu từ SBUF .

Ví dụ:

WAIT : JNB RI, WAIT ;test RI và chờ cho đến khi được lập

CLR RI ;xoá RI

MOV A, SBUF ; Đọc dữ liệu từ SBUF

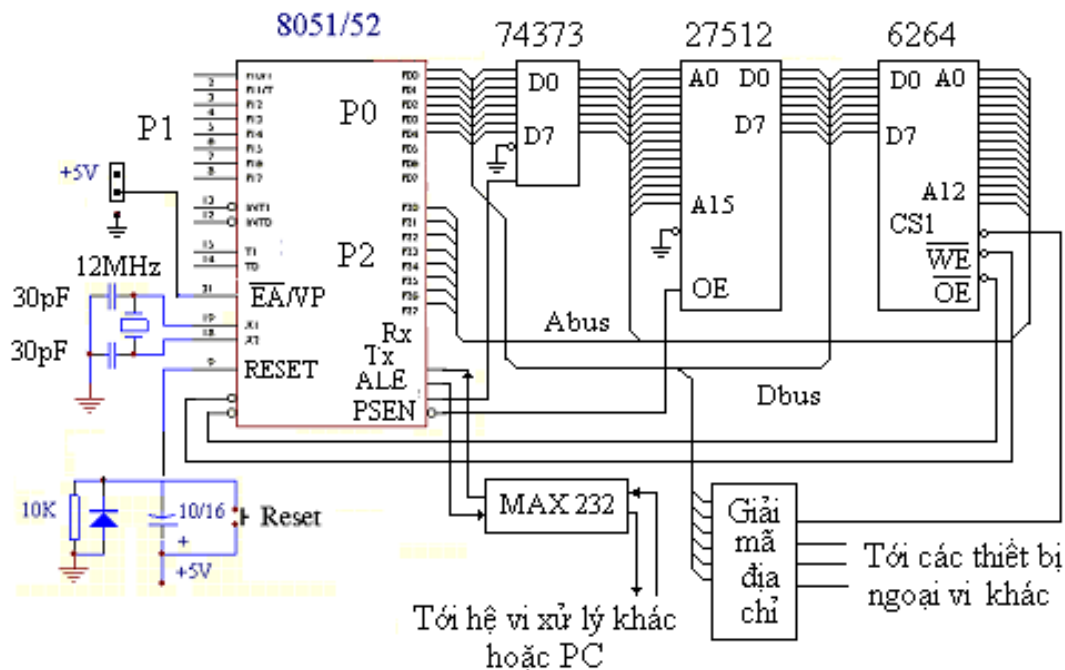
Tương tự bit TI được lập khi bit cuối cùng của dữ liệu được truyền và nó cho biết rằng đã truyền xong (Transmit buffer empty). Nếu chương trình muốn gửi dữ liệu đến thiết bị nối qua cổng nối tiếp nó phải kiểm tra xem ký tự trước đó đã gửi chưa, nếu chưa gửi nó phải chờ đến khi gửi xong mới được gửi .

Ví dụ:

WAIT: JNB TI, WAIT ; Kiểm tra và chờ đến khi TI set

CLR TI ; Nếu set thì xoá TI

MOV SBUF, A ; Gửi dữ liệu vào SBUF để truyền.



Hệ vi xử lý sử dụng 8051 có bộ nhớ ngoài

### 3.2 Hệ lệnh của 8051

Tập lệnh của 8051 được chia thành 5 nhóm:

- Số học.
- Luận lý.
- Chuyển dữ liệu.

- Chuyển điều khiển.
- Rẽ nhánh.

Bảng 15. Các chi tiết thiết lập lệnh

Rn	Thanh ghi R0 đến R7 của bank thanh ghi được chọn.
Data	8 bit địa chỉ vùng dữ liệu bên trong. Nó có thể là vùng RAM dữ liệu trong (0-127) hoặc các thanh ghi chức năng đặc biệt.
@Ri	8 bit vùng RAM dữ liệu trong (0-125) được đánh giá địa chỉ gián tiếp qua thanh ghi R0 hoặc R1.
#data	Hằng 8 bit chức trong câu lệnh.
#data 16	Hằng 16 bit chứa trong câu lệnh.
Addr16	16 bit địa chỉ đích được dùng trong lệnh LCALL và LJMP.
Addr11	11 bit địa chỉ đích được dùng trong lệnh LCALL và AJMP.
Rel	Byte offset 8 bit có dấu được dùng trong lệnh SJMP và những lệnh nhảy có điều kiện.
Bit	Bit được định địa chỉ trực tiếp trong RAM dữ liệu nội hoặc các thanh ghi chức năng đặc biệt.

### Nhóm lệnh xử lý số học

Bảng 16. Lệnh xử lý số học

ADD A,Rn	(1byte 1 chu kỳ máy) : cộng nội dung thanh ghi Rn vào thanh ghi A.
ADD A,data	(21): Cộng trực tiếp 1 byte vào thanh ghi A.
ADD A,@Ri	(11): Cộng gián tiếp nội dung RAM chứa tại địa chỉ được khai báo trong Ri vào thanh ghi A.
ADD A,#data	(21): Cộng dữ liệu tức thời vào A.
ADD A,Rn	(11): Cộng thanh ghi và cờ nhớ vào A.

ADD A,data	(21): Cộng trực tiếp byte dữ liệu và cờ nhớ vào A.
ADDC A,@Ri	(11): Cộng gián tiếp nội dung RAM và cờ nhớ vào A.
ADDC A,#data	(21): Cộng dữ liệu tức thời và cờ nhớ vào A.
SUBB A,Rn	(11): Trừ nội dung thanh ghi A cho nội dung thanh ghi Rn và cờ nhớ.
SUBB A,data	(21): Trừ trực tiếp A cho một số và cờ nhớ.
SUBB A,@Ri	(11): Trừ gián tiếp A cho một số và cờ nhớ.
SUBB A,#data	(21): Trừ nội dung A cho một số tức thời và cờ nhớ.
INC A	(11): Tăng nội dung thanh ghi A lên 1.
INC Rn	(11): Tăng nội dung thanh ghi Rn lên 1.
INC data	(21): Tăng dữ liệu trực tiếp lên 1.
INC @Ri	(11): Tăng gián tiếp nội dung vùng RAM lên 1.
DEC A	(11): Giảm nội dung thanh ghi A xuống 1.
DEC Rn	(11): Giảm nội dung thanh ghi Rn xuống 1.
DEC data	(21): Giảm dữ liệu trực tiếp xuống 1
DEC @Ri	(11): Giảm gián tiếp nội dung vùng RAM xuống 1.
INC DPTR	(12): Tăng nội dung con trỏ dữ liệu lên 1.
MUL AB	(14): Nhân nội dung thanh ghi A với nội dung thanh ghi B.
DIV AB	(14): Chia nội dung thanh ghi A cho nội dung thanh ghi B.
DA A	(11): hiệu chỉnh thập phân thanh ghi A.

### Nhóm lệnh luận lý

Bảng 17. Lệnh luận lý

ANL A,Rn	(11): AND nội dung thanh ghi A với nội dung thanh ghi Rn.
ANL A,data	(21): AND nội dung thanh ghi A với dữ liệu trực tiếp.
ANL A,@Ri	(11): AND nội dung thanh ghi A với dữ liệu gián tiếp trong RAM.
ANL A,#data	(21): AND nội dung thanh ghi với dữ liệu tức thời.
ANL data,A	(21): AND một dữ liệu trực tiếp với A.
ANL data,#data	(32): AND một dữ liệu trực tiếp với A một dữ liệu tức thời.

ANL C,bit	(22): AND cờ nhớ với 1 bit trực tiếp.
ANL C,/bit	(22): AND cờ nhớ với bự 1 bit trực tiếp.
ORL A,Rn	(11): OR thanh ghi A với thanh ghi Rn.
ORL A,data	(21): OR thanh ghi A với một dữ liệu trực tiếp.
ORL A,@Ri	(11): OR thanh ghi A với một dữ liệu gián tiếp.
ORL A,#data	(21): OR thanh ghi A với một dữ liệu tức thời.
ORL data,A	(21): OR một dữ liệu trực tiếp với thanh ghi A.
ORL data,#data	(31) :OR một dữ liệu trực tiếp với một dữ liệu tức thời.
ORL C,bit	(22): OR cờ nhớ với một bit trực tiếp.
ORL C,/bit	(22): OR cờ nhớ với bự của một bit trực tiếp.
XRL A,Rn	(11): XOR thanh ghi A với thanh ghi Rn.
XRL A,data	(21): XOR thanh ghi A với một dữ liệu trực tiếp.
XRL A,@Ri	(11): XOR thanh ghi A với một dữ liệu gián tiếp.
XRL A,#data	(21): XOR thanh ghi A với một dữ liệu tức thời.
XRL data,A	(21): XOR một dữ liệu trực tiếp với thanh ghi A.
XRL data,#data	(31): XOR một dữ liệu trực tiếp với một dữ liệu tức thời.
SETB C	(11): Đặt cờ nhớ.
SETB bit	(21): Đặt một bit trực tiếp.
CLR A	(11): Xóa thanh ghi A.
CLR C	(11): Xóa cờ nhớ.
CPL A	(11): Bù nội dung thanh ghi A.
CPL C	(11): Bù cờ nhớ.
CPL bit	(21): Bù một bit trực tiếp.
RL A	(11): Quay trái nội dung thanh ghi A.
RLC A	(11): Quay trái nội dung thanh ghi A qua cờ nhớ.
RR A	(11): Quay phải nội dung thanh ghi A.
RRC A	(11): Quay phải nội dung thanh ghi A qua cờ nhớ.
SWAP	(11): Quay trái nội dung thanh ghi A 1 nibble (1/2byte).

### Nhóm lệnh chuyển dữ liệu

Bảng 18. Lệnh chuyển dữ liệu

MOV A,Rn	(11):Chuyển nội dung thanh ghi Rn vào thanh ghi A.
----------	--

MOV A,data	(21): Chuyển dữ liệu trực tiếp vào thanh ghi A.
MOV A,@Ri	(11): Chuyển dữ liệu gián tiếp vào thanh ghi A.
MOV A,#data	(21): Chuyển dữ liệu tức thời vào thanh ghi A.
MOV Rn,data	(22): Chuyển dữ liệu trực tiếp vào thanh ghi Rn.
MOV Rn,#data	(21): Chuyển dữ liệu tức thời vào thanh ghi Rn.
MOV data,A	(21): Chuyển nội dung thanh ghi A vào một dữ liệu trực tiếp.
MOV data,Rn	(22): Chuyển nội dung thanh ghi Rn vào một dữ liệu trực tiếp.
MOV data,data	(32): Chuyển một dữ liệu trực tiếp vào một dữ liệu trực tiếp.
MOV data,@Ri	(22): Chuyển một dữ liệu gián tiếp vào một dữ liệu gián tiếp.
MOV data,#data	(32): Chuyển một dữ liệu tức thời vào một dữ liệu trực tiếp.
MOV @Ri,A	(11): Chuyển nội dung thanh ghi A vào một dữ liệu gián tiếp.
MOV @Ri,data	(22): Chuyển một dữ liệu trực tiếp vào một dữ liệu gián tiếp.
MOV @Ri,#data	(21): Chuyển dữ liệu tức thời vào dữ liệu gián tiếp.
MOV DPTR,#data	(32): Chuyển một hằng 16 bit vào thanh ghi con trỏ dữ liệu.
MOV C,bit	(21): Chuyển một bit trực tiếp vào cờ nhớ.
MOV bit,C	(22): Chuyển cờ nhớ vào một bit trực tiếp.
MOV A,@A+DPTR	(12): Chuyển byte bộ nhớ chương trình có địa chỉ là @A+DPTR vào thanh ghi A.
MOVC A,@A+PC	(12): Chuyển byte bộ nhớ chương trình có địa chỉ là @A+PC vào thanh ghi A.
MOVX A,@Ri	(12): Chuyển dữ liệu ngoài (8 bit địa chỉ) vào thanh ghi A.
MOVX A,@DPTR	(12): Chuyển dữ liệu ngoài (16 bit địa chỉ) vào thanh ghi A.
MOVX @Ri,A	(12): Chuyển nội dung A ra dữ liệu ngoài (8 bit địa chỉ).

MOVX @DPTR,A	(12): Chuyển nội dung A ra dữ liệu bên ngoài (16 bit địa chỉ).
PUSH data	(22): Chuyển dữ liệu trực tiếp vào ngăn xếp và tăng SP.
POP data	(22): Chuyển dữ liệu trực tiếp vào ngăn xếp và giảm SP.

XCH A,Rn	(11): Trao đổi dữ liệu giữa thanh ghi Rn và thanh ghi A.
XCH A,data	(21): Trao đổi giữa thanh ghi A và một dữ liệu trực tiếp.
XCH A,@Ri	(11): Trao đổi giữa thanh ghi A và một dữ liệu gián tiếp.
XCHD A,@R	(11): Trao đổi giữa nibble thấp (LSN) của thanh ghi A và LSN của dữ liệu gián tiếp.

### Nhóm lệnh chuyên điều khiển

Bảng 19. Lệnh chuyên điều khiển

ACALL addr11	(22): Gọi chương trình con dùng địa chỉ tuyệt đối.
LCALL addr16	(32): Gọi chương trình con dùng địa chỉ dài.
RET	(12): Trở về từ lệnh gọi chương trình con.
RETI	(12): Trở về từ lệnh gọi ngắt.
AJMP addr11	(22): Nhảy tuyệt đối.
LJMP addr16	(32): Nhảy dài.
SJMP rel	(22): Nhảy ngắn.
JMP @A+DPTR	(12): Nhảy gián tiếp từ con trỏ dữ liệu.
JZ rel	(22): Nhảy nếu A=0.
JNZ rel	(22): Nhảy nếu A không bằng 0.
JC rel	(22): Nhảy nếu cờ nhớ được đặt.
JNC rel	(22): Nhảy nếu cờ nhớ không được đặt.
JB bit,rel	(32): Nhảy tương đối nếu bit trực tiếp được đặt.
JNB bit,rel	(32): Nhảy tương đối nếu bit trực tiếp không được đặt.
JBC bit,rel	(32): Nhảy tương đối nếu bit trực tiếp được đặt, rồi xoá bit.
CJNE A,data,rel	(32): So sánh dữ liệu trực tiếp với A và nhảy nếu không bằng.
CJNE A,#data,rel	(32): So sánh dữ liệu tức thời với A và nhảy nếu không bằng.
CJNE	(32): So sánh dữ liệu tức thời với nội dung thanh ghi Rn và

Rn,#data,rel	nhảy nếu không bằng.
CJNE @Ri,#data,rel	(32): So sánh dữ liệu tức thời với dữ liệu gián tiếp và nhảy nếu không bằng.
DJNZ Rn,rel	(22): Giảm thanh ghi Rn và nhảy nếu không bằng.
DJNZ data	(32): Giảm dữ liệu trực tiếp và nhảy nếu không bằng.

### Các lệnh rẽ nhánh

Có nhiều lệnh để điều khiển lên chương trình bao gồm việc gọi hoặc trả lại từ chương trình con hoặc chia nhánh có điều kiện hay không có điều kiện. Tất cả các lệnh rẽ nhánh đều không ảnh hưởng đến cờ. Ta có thể định nhảy đến nơi cần nhảy mà không cần đưa rõ địa chỉ, trình biên dịch sẽ đặt địa chỉ nơi cần nhảy tới vào đúng khẩu lệnh đó đưa ra.

Bảng 20. Lệnh nhảy có điều kiện

<condition>	Jump_if_not<condition>	Jump_if <condition>
C=1	JNC rel	JC rel
Bit=1	JNB bit,rel	JB bit,rel/JNC bit,rel
A=0	JNZ rel	JZ rel
Rn=0	DJNZ Rn,rel	
Direct=0	DJNZ direct,rel	
A direct	CJNE A,direct,rel	
A#data	CJNE A,#data,rel	
Rn#data	CJNE Rn,#data,rel	
@Ri#data	CJNE @Ri,#data,rel	

#### Nhảy không có điều kiện

##### a. Cấu trúc “repeat... until”

Repeat

<action>

Until <condition>

*Ngôn ngữ Assembly:*

LOOP:

<action>

JUMP\_if\_not\_<condition>,LOOP

##### b. Cấu trúc “while... do”

while

<condition>

do <action>

*Ngôn ngữ Assembly:* LOOP:

JUMP\_if\_not\_<condition>,DO

SJMP STOP

DO: <action>



SJMP LOOP

STOP: ...

**c. Cấu trúc “if... then... else”**

if

<condition>

then <action 1>

else <action 2>

*Ngôn ngữ Assembly*

JUMP\_if\_not\_<condition>, ELSE

<action 1>

SJMP DONE

ELSE: <action 2>

DONE: ...

**d. Cấu trúc “case... of...”**

case P1 of

#11111110b: P2.0 = 1

#11111101b: P2.1 = 1

#11111011b: P2.2 = 1

else P2 = 0

end

*Ngôn ngữ Assembly*

CJNE P1,#11111110b,

SKIP1 SETB P2.0

SJMP EXIT

SKIP1: CJNE P1,#11111101b,SKIP2

SETB P2.1

SJMP EXIT

SKIP2: CJNE P1,#11111011b,SKIP3

SETB P2.2

SJMP EXIT

SKIP3: MOV P2,#0

EXIT: ...

Sau đây là sự tóm tắt từng hoạt động của lệnh nhảy.

JC	rel	: Nhảy đến “rel” nếu cờ Carry C = 1.
JNC	rel	: Nhảy đến “rel” nếu cờ Carry C = 0. JB
	bit, rel	: Nhảy đến “rel” nếu (bit) = 1.

JNB	bit, rel	: Nhảy đến “rel” nếu (bit) = 0.
JBC	bit, rel	: Nhảy đến “rel” nếu bit = 1 và xóa bit.
ACALL	addr11	: Lệnh gọi tuyệt đối trong page 2K. (PC) (PC) + 2 (SP) (SP) + 1 ((SP)) (PC7PC0) (SP) (SP) + 1 ((SP)) (PC15PC8) (PC10PC0) page Address.
LCALL	addr16	: Lệnh gọi dài chương trình con trong 64K. (PC) (PC) + 3 (SP) (SP) + 1  ((SP)) (PC7PC0) (SP) (SP) + 1 ((SP)) (PC15PC8) (PC) Addr15Addr0.
RET		: Kết thúc chương trình con trở về chương trình chính. (PC15PC8) (SP)  (SP) (SP) - 1 (PC7PC0) ((SP)) (SP) (SP) -1.
RETI		: Kết thúc thủ tục phục vụ ngắt quay về chương trình chính hoạt động tương tự như RET.
AJMP	Addr11	: Nhảy tuyệt đối không điều kiện trong 2K. (PC) (PC) + 2 (PC10PC0) page Address.
LJMP	Addr16	: Nhảy dài không điều kiện trong 64K Hoạt động tương tự lệnh LCALL.
SJMP	rel	: Nhảy ngắn không điều kiện trong (-128 127) byte (PC) (PC) + 2 (PC) (PC) + byte 2
JMP	@ A + DPTR	: Nhảy không điều kiện đến địa chỉ (A) + (DPTR) (PC) (A) + (DPTR)
JZ	rel	: Nhảy đến A = 0. Thực hành lệnh kế nếu A = 0. (PC) (PC) + 2 (A) = 0 (PC) (PC) + byte 2
JNZ	rel	: Nhảy đến A ≠ 0. Thực hành lệnh kế nếu A ≠ 0. (PC) (PC) + 2 < > 0 (PC) (PC) + byte 2

CJNE A, direct, rel	: So sánh và nhảy đến A direct (PC) (PC) + 3 (A) < > (direct) (PC) (PC) + Relative Address. (A) < (direct) C = 1 (A) > (direct) C = 0 (A) = (direct). Thực hành lệnh kế tiếp
CJNE A, # data, rel	: Tương tự lệnh CJNE A, direct, rel.
CJNE Rn, # data, rel	: Tương tự lệnh CJNE A, direct, rel.
CJNE @ Ri, # data, rel	: Tương tự lệnh CJNE A, direct, rel.
DJNE Rn, rel	: Giảm Rn và nhảy nếu Rn 0. (PC) (PC) + 2 (Rn) (Rn) - 1 (Rn) < > 0 (PC) (PC) + byte 2.
DJNZ direct, rel	: Tương tự lệnh DJNZ Rn, rel.

### Các lệnh dịch chuyển dữ liệu

Các lệnh dịch chuyển dữ liệu trong những vùng nhớ nội thực thi 1 hoặc 2 chu kỳ máy. Mẫu lệnh MOV <destination>, <source> cho phép di chuyển dữ liệu bất kỳ 2 vùng nhớ nào của RAM nội hoặc các vùng nhớ của các thanh ghi chức năng đặc biệt mà không thông qua thanh ghi A.

Vùng Ngăn xếp của 8951 chỉ chứa 128 byte RAM nội, nếu con trỏ ngăn xếp SP được tăng quá địa chỉ 7FH thì các byte được PUSH vào sẽ mất đi và các byte POP ra thì không biết rõ.

Các lệnh dịch chuyển bộ nhớ nội và bộ nhớ ngoại dùng sự định vị gián tiếp. Địa chỉ gián tiếp có thể dùng địa chỉ 1 byte (@ Ri) hoặc địa chỉ 2 byte (@ DPTR). Tất cả các lệnh dịch chuyển hoạt động trên toàn bộ nhớ ngoài thực thi trong 2 chu kỳ máy và dùng thanh ghi A làm toán hạng DESTINATION.

Việc đọc và ghi RAM ngoài (RD và WR) chỉ tích cực trong suốt quá trình thực thi của lệnh MOVX, còn bình thường RD và WR không tích cực (mức 1). Tất cả các lệnh dịch chuyển đều không ảnh hưởng đến cờ. Hoạt động của từng lệnh được tóm tắt như sau:

PUSH direct	: Cất dữ liệu vào ngăn xếp (SP) (SP) + 1 (SP) (Direct)
POP direct	: Lấy từ ngăn xếp ra direct (direct) ((SP)) (SP) (SP) - 1
XCH A, Rn	: Đổi chỗ nội dung của A với Rn (A) (Rn)
XCH A, direct	: (A) (direct)

XCH A, @ Ri : (A) ((Ri))

XCHD A, @ Ri : Đổi chỗ 4 bit thấp của (A) với ((Ri))  
(A3A0) ((Ri3Ri0))

**Các lệnh xen vào (MiCSellamous Intstruction):**

NOP : Không hoạt động gì cả, chỉ tốn 1 byte và 1 chu kỳ máy. Ta dùng để delay những khoảng thời gian nhỏ.