# Giới thiệu môn học

# LẬP TRÌNH HỆ THỐNG

Xem xét các phần cơ bản của các phần mềm hệ thống phục vụ cho việc điều hành và ghép nối giữa các phần mềm ứng dụng và phần cứng của máy tính trên môi trường DOS và Windows để trên cơ sở đó các phần mềm ứng dụng có thể khai thác hiệu quả tối đa tài nguyên vật lý của máy tính.

# Tài liệu

**Tài liệu học tập:**

*Giáo trình Lập trình hệ thống* của nhóm tác giả Khoa Công nghệ Thông tin, Viện Đại học Mở Hà Nội

**Tài liệu tham khảo:**


1. *Turbo Assembler Ver.4.0 (User Guide).* Borland International, INC 1800 GREEN HILLS ROAD, 1993
2. *Turbo Assembler Ver.4.0 (Quick Reference Guide).* Borland International, INC 1800 GREEN HILLS ROAD, 1993
3. *Turbo Assembler và ứng dụng*, Đặng Thành Phu, NXB Khoa học và Kỹ thuật, 2007
4. *Lập trình trên môi trường Windows với MFC*, Dương Thăng Long, NXB Khoa học và Kỹ thuật, 2006
5. *PC System Programming – An in depth reference for DOS programmer*, Michael Tischer, Abacus, 1990.
6. *Công cụ phần mềm hỗ trợ THELP*

# Chương 1

## Ngôn ngữ Assembly và cách lập trình

*Mục đích*:

Giới thiệu các khái niệm, kiến thức, các thành phần cơ bản các bước chi tiết cần thiết khi tiến hành lập trình bằng ngôn ngữ Assembly cùng các ví dụ minh họa.

# 1.1 Mở đầu

Ngôn ngữ Assembly là ngôn ngữ bậc thấp.

Ưu điểm:

- Chạy nhanh và tiết kiệm bộ nhớ,
- Dễ dàng thâm nhập trực tiếp vào các thiết bị phần cứng như: vùng nhớ, các cổng, các thanh ghi,…

Nhược điểm:

- Khó viết vì phải am hiểu sâu về phần cứng,
- Khó khăn trong việc kiểm tra lỗi,
- Khó khăn trong việc chuyển giao chương trình lên các máy tính có cấu trúc khác nhau.

Ứng dụng:

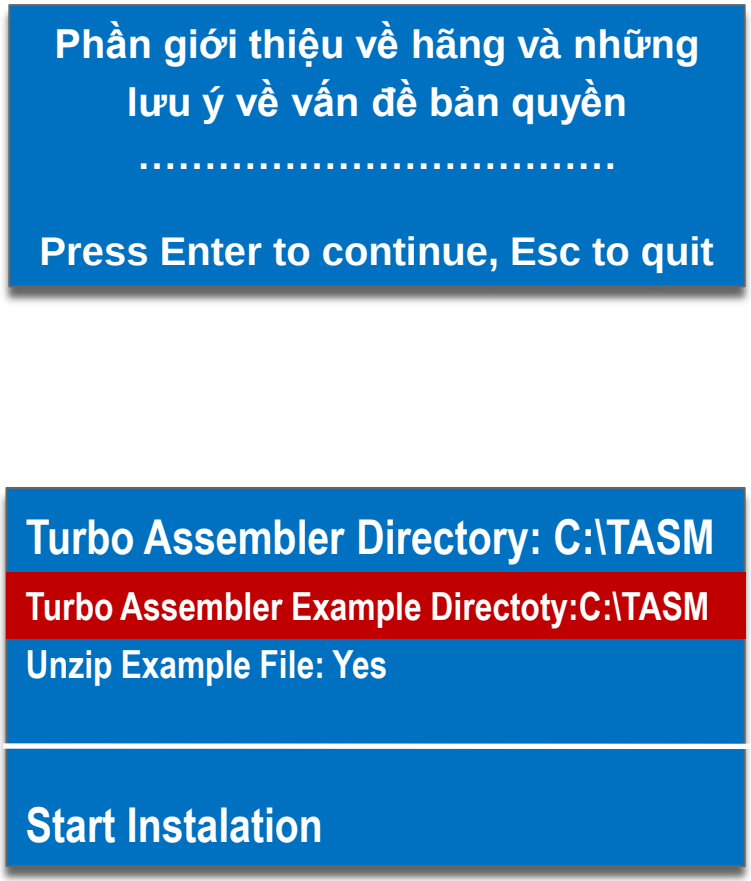
- Các chương trình của ROM BIOS,
- Các chương trình trong các hệ thống nhúng,
- Các chương trình tạo và diệt VIRUS.

# 1.2 Cài đặt chương trình dịch

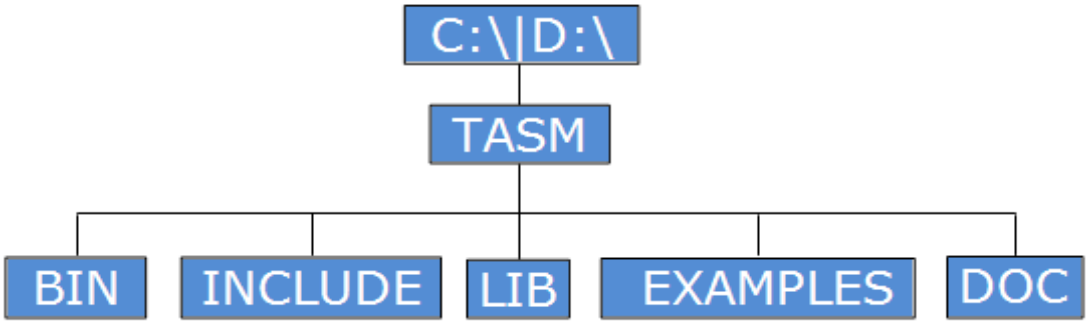
Có 2 CT dịch: MASM của Microsoft và TASM của Borland.

*Cách 1*:

- Đưa đĩa có chương trình cài đặt vào ổ CD, chạy setup.exe thì hiện màn hình giới thiệu.
- Ấn Enter thì để tiếp tục quá trình cài đặt, trên màn hình sẽ hiện lên với các thư mục mặc định. Muốn thay đổi thì đưa thanh sáng đến phần cần thay đổi và ấn Enter, các dòng hướng dẫn sẽ hiện ra cho phép vào tên ổ đĩa và thư mục yêu cầu. Sau khi thay đổi xong ấn Start Instalation thì quá trình cài đặt sẽ được thực hiện.



- Khi cài đặt hoàn tất, các thư mục trong đó chứa các tệp CT dịch được tạo ra, ví dụ với sự lựa chọn mặc định C: hoặc thay đổi D:




- Trong thư mục BIN có rất nhiều tệp của chương trình dịch TASM, trong đó có các tệp chính (các tệp lõi của CT dịch) gồm :

TASM.EXE … CT dịch
TLINK.EXE … CT liên kết
và hai tệp hỗ trợ là RTM.EXE
và DPMI16BI.OVL

*Cách 2*:

Để tiết kiệm bộ nhớ chỉ cần copy 4 tệp lõi của chương trình dịch: tasm.exe, tlink.exe, rtm.exe và dpmi16bi.ovl từ một máy đã được cài theo cách 1 về máy mình.

# 1.3 Các bước thực hiện một CT ASM trên PC

Để thực hiện 1 chương trình Assembly trên máy PC có 4 bước:

- *Bước 1*: Dùng 1 editor bất kỳ để soạn thảo CT, sau đó cất vào một tệp phải có đuôi .ASM,
- *Bước 2*: Dịch CT (chuyển tệp .asm sang tệp .obj) với cú pháp:

  TASM [option] SOURCEfile [,OBJfile][,LSTfile][,XRFfile]

  trong đó:

  SOURCEfile … tệp nguồn (có phần mở rộng .asm),
  OBJflie … tệp đích (có phần mở rộng .obj),
  LSTfile … tệp phục vụ in (có phần mở rộng .lst)
  XRFfile … giống tệp LST (thêm phần qui chiếu nhãn)

  *Chú ý*: Bước 2 chỉ tạo ra các tệp có đuôi **.obj** khi dịch không sai.
- *Bước 3*: Liên kết (chuyển tệp .obj sang tệp .exe) với cú pháp:

  TLINK [option] OBJfile [,EXEfile] [,MAPfile] [,LIBfile]

  *Chú ý*: Bước 3 chỉ tạo ra tệp **.exe** khi liên kết không có sai.
- *Bước 4*: Chạy thử chương trình (đánh tên tệp)

# 1.4 Tổng quan về môi trường lập trình

## 1.4.1 Các thanh ghi

Các thanh ghi là một vùng nhớ đặc biệt dạng RAM nằm trên CPU. Việc thâm nhập vào các thanh ghi thông qua tên thanh ghi chứ không phải thông qua địa chỉ như khai báo biến. Người lập trình ASM rất hay dùng các thanh ghi làm toán hạng sau các lệnh thay vì các biến. Có thể chia thanh ghi làm 4 nhóm.

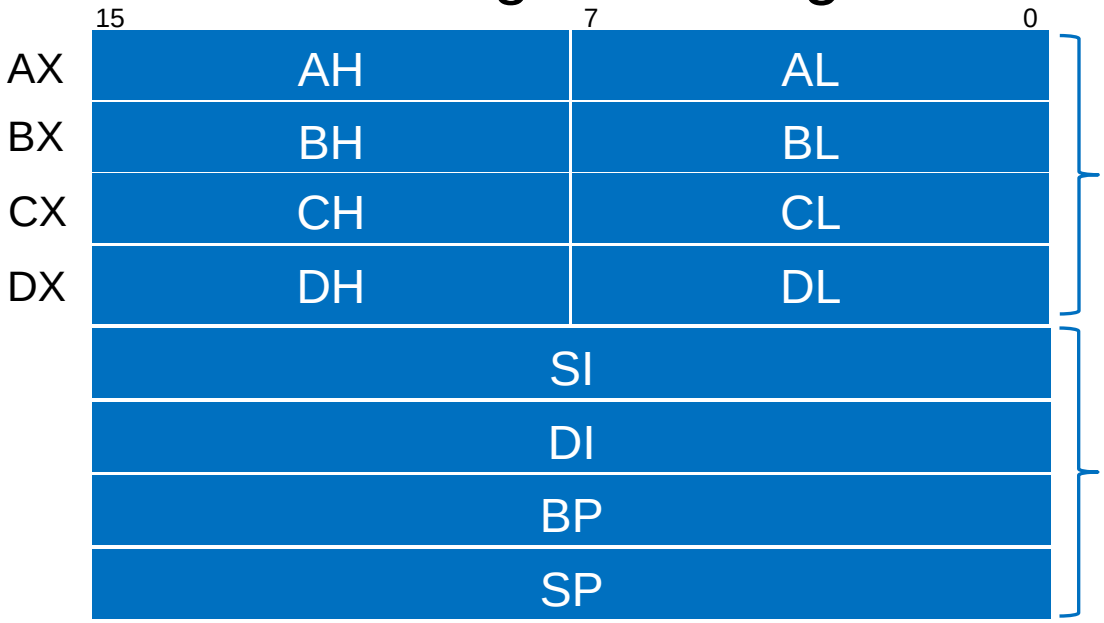
a. *Với máy tính 16 bit*: Có 14 thanh ghi

*Nhóm 1*: Một thanh ghi cờ 16 bit


trong đó:

O… cờ tràn (Overflow)
D… cờ hướng (Direction)
I… cờ ngắt (Interruption)
T… cờ bẫy (Trap)
S… cờ dấu (Sign)
Z… cờ zero (Zero)
A… cờ phụ (Auxiliary)
P… cờ chẵn/lẻ (Parity)
C… cờ carry (Carry)

## *Nhóm 2*: 8 thanh ghi đa năng 16 bit




*Có 3 mode truy nhập*:
Thanh ghi byte thấp (ví dụ AL)
Thanh ghi byte cao (ví dụ AH)
Thanh ghi 16 bit (ví dụ AX)

Chỉ có một mode truy nhập duy nhất 16 bit

## *Nhóm 3*: Một thanh ghi con trỏ lệnh 16 bit (IP)


Chứa phần địa chỉ offset của vùng nhớ chứa mã lệnh

## *Nhóm 4*: 4 thanh ghi segment 16 bit


Chứa phần địa chỉ segment của vùng nhớ chứa mã lệnh (CS), vùng nhớ chứa dữ liệu (DS và ES) và vùng nhớ dành cho ngăn xếp (SS)

*b.Với máy tính 32 bit*: Có 16 thanh ghi

- Các thanh ghi thuộc nhóm 1, nhóm 2 và nhóm 3 có độ dài 32 bit với tên có chữ E đứng trước (ví dụ AX→EAX, IP→ EIP,.. .)
- Các thanh ghi thuộc nhóm 4 (các thanh ghi segment) vẫn là 16 bit và có thêm 2 thanh ghi FS và GS hỗ trợ cho phần dữ liệu.

### 1.4.2 Cách thể hiện địa chỉ 1 ô nhớ (RAM hoặc ROM)

Địa chỉ 1 ô nhớ có 2 cách thể hiện:

*a. Dạng lôgic*:

địa chỉ 1 ô nhớ = seg : offset

trong đó:

seg … phần địa chỉ segment cho biết ô nhớ đó nằm 64 k (segment) nào và

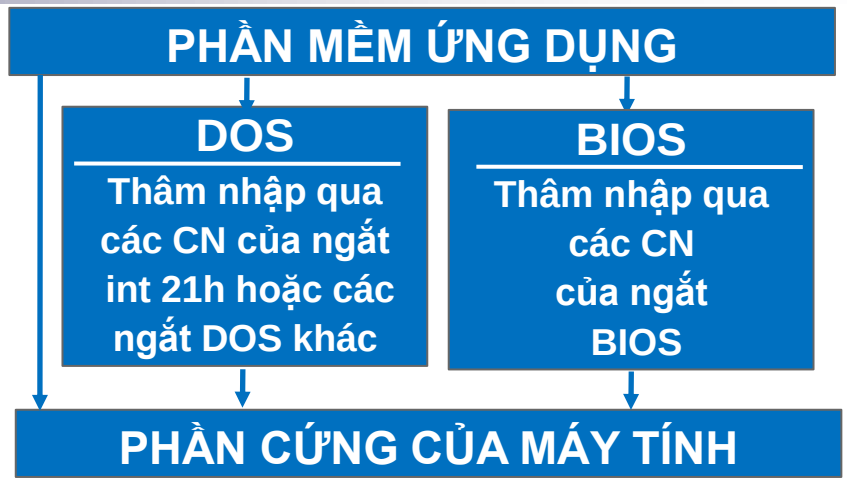
offset … phần địa chỉ offset cho biết khoảng cách ô nhớ đó so với đầu segment

*b. Dạng vật lý*:

địa chỉ 1 ô nhớ = seg*16 + offset

## 1.4.3 Các phần mềm hệ thống

Các ngắt của DOS và BIOS là 2 phần mềm phục vụ cho điều hành và ghép nối giữa phần mềm ứng dụng với phần cứng của máy tính.




Các hàm VÀO/RA thường dùng thông qua ngắt của DOS và BIOS:

**Hàm số 1: Chờ nhận 1 ký tự từ bàn phím**

```
mov   AH,1
int   21h
```

KQ: mã ASCII -> AL

**Hàm số 2: Hiện 1 ký tự lên màn hình**

*Cách 1*:
```
mov   AL, mã ASCII của ký tự
mov   AH,0eh
int   10h
```

*Cách 2*:
```
mov   DL, mã ASCII của ký tụ
mov   AH,2
int   21h
```

**Hàm số 3: Hiện 1 xâu (kết thúc '$')**

```
lea   DX, tên biến xâu
mov   AH,9
int   21h
```

**Hàm số 4: Trở về DOS**

```
mov   AH,4Ch
int   21h
```

# 1.5 Hệ lệnh

## 1.5.1 Cú pháp 1 dòng lệnh Assembly

Mỗi 1 dòng chỉ được viết 1 lệnh Assembly với cú pháp sau:

[label] [directive/instruction] [operands] [; comment]

trong đó:

- label (nhãn): là 1 định danh để qui chiếu đến các số, các xâu, các biến, tên CT con hoặc nhãn nhảy,
- directive/instruction (lệnh điều khiển khi dịch CT - hỗ trợ khi dịch CT/lệnh dạng mnemonic-sinh ra mã máy để chạy chương trình),
- operands (toán hạng): chỉ cho CT dịch biết thanh ghi nào, tham số nào, nhãn nào (tên biến, tên CT con, tên hằng, tên nhãn nhảy, …) liên quan đến lệnh điều khiển khi dịch CT (directive) hoặc lệnh (instruction),
- comment (ghi chú): từ dấu ; (chấm phẩy) đến hết dòng là chú thích và dấu ; (ghi chú) chỉ có hiệu lực trên 1 dòng,

## 1.5.2 Tập lệnh mnemonic (Instruction Set)

Có thể chia tập lệnh của ngôn ngữ Assembly thành 6 nhóm:

- Nhóm các lệnh di chuyển dữ liệu,
- Nhóm các lệnh số học,
- Nhóm các lệnh thao tác bit,
- Nhóm các lệnh làm việc với xâu ký tự,
- Nhóm các lệnh rẽ nhánh và
- Nhóm các xác lập trạng thái các bit cờ.

Một số qui ước về toán hạng:

| | |
|---|---|
| SRC | toán hạng nguồn |
| DST | toán hạng đích |
| reg | toán hạng là thanh ghi |
| reg8/reg16 | toán hạng là thanh ghi 8/16 bit |
| mem | toán hạng là biến nhớ |
| mem8/mem16 | toán hạng là biến 1 byte hoặc 2 byte |
| data | toán hạng là hằng số |
| segreg | toán hạng là thanh ghi segment |

# *Nhóm 1*: Một số lệnh hay dùng thuộc nhóm di chuyển dữ liệu

- Lệnh MOV

*Chức năng*: Gán giá trị toán hạng SRC cho toán hạng DST.

*Cú pháp*:

**mov DST,SRC**

| | | |
|---|---|---|
| reg1,reg2 | reg16, segreg | mem16,segreg |
| reg,data | mem,reg | segreg,reg16 |
| reg,mem | mem,data | segreg,mem16 |

*Chú ý*: 1.~~mov mem1,mem2~~ ⟹ **mov reg,mem2 mov mem1,reg**

2.~~mov segreg,data~~ ⟹ **mov reg16,data mov segreg,reg16**

3.**mov AX,BX** ⇦ **mov AX,[BX]** vì thanh ghi đứng trong dấu [ ] là con trỏ offset của 1 ô nhớ

- Lệnh PUSH

*Chức năng*: Cất giá trị toán hạng SRC vào đỉnh ngăn xếp.

*Cú pháp*:

**push SRC**

**reg16** hoặc **mem16** hoặc **segreg**

- Lệnh POP

*Chức năng*: Đưa giá trị 2 byte của đỉnh ngăn xếp vào DST.

*Cú pháp*:

```
pop   DST
      reg16  hoặc mem16  hoặc   segreg
```

- Lệnh PUSHF

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

*Cú pháp*:

```
pushf
```

- Lệnh POPF

*Chức năng*: Đưa giá trị 2 byte của đỉnh ngăn xếp vào thanh ghi cờ.

*Cú pháp*:

```
popf
```

- Lệnh XCHG

*Chức năng*: Đổi chéo cho nhau giá trị toán hạng nguồn và đích.

*Cú pháp*:

```
xchg   DST,SRC
       reg1,reg2
       reg,mem
       mem,reg
```

- Lệnh LEA

*Chức năng*: Đưa phần địa chỉ offset toán hạng nguồn (phần địa chỉ ô nhớ cấp phát cho biến nhớ) vào thanh ghi 16 bit.

*Cú pháp*:

**lea reg16,mem**

*Chú ý*: **reg16** chỉ có thể là các thanh ghi: **BX, SI, DI** hoặc **BP**

- Lệnh LDS

*Chức năng*: Chuyển giá trị 2 byte của biến nhớ vào thanh ghi đích và giá trị 2 byte tiếp theo vào thanh ghi DS.

*Cú pháp*:

**lds reg16,mem**

*Chú ý*: **reg16** chỉ có thể là các thanh ghi: **BX, SI, DI** hoặc **BP**

- Lệnh LES

*Chức năng*: Giống lệnh trên chỉ thay thanh ghi DS bằng ES.

*Cú pháp*:

**les reg16,mem**

*Chú ý:* **reg16** chỉ có thể là các thanh ghi: **BX, SI, DI** hoặc **BP**

## *Nhóm 2*: Một số lệnh hay dùng thuộc nhóm số học

*Chú ý*: Giá trị các bit cờ có thể bị thay đổi giá khi thực hiện lệnh.

- Lệnh ADD

  *Chức năng*: DST = DST+SRC

  *Cú pháp*:

  **add DST,SRC**
  **reg1,reg2** **reg,data**
  **reg,mem** **mem,reg** **mem,data** ★
  *Cờ*: **C, P, A, Z, S** và **O**

- Lệnh ADC

  *Chức năng*: DST = DST+SRC+C

  *Cú pháp*:

  **adc DST,SRC**
  ★
  *Cờ*: **C, P, A, Z, S** và **O**

- Lệnh INC

  *Chức năng*: DST = DST+1

  *Cú pháp*:

  **inc DST**
  **reg** hoặc **mem**
  *Cờ*: **P, A, Z, S** và **O**

- Lệnh SUB

*Chức năng*: DST = DST-SRC

*Cú pháp*:

**sub DST,SRC**

*Cờ*: **C, P, A, Z, S** và **O**

- Lệnh SBB

*Chức năng*: DST = DST-SRC-C

*Cú pháp*:

**sbb DST,SRC**

*Cờ*: **C, P, A, Z, S** và **O**

- Lệnh DEC

*Chức năng*: DST = DST-1

*Cú pháp*:

**dec DST**

**reg** hoặc **mem**

*Cờ*: **P, A, Z, S** và **O**

- Lệnh NEG

*Chức năng*: Đổi dấu giá trị toán hạng (DST=-DST - bù 2).

*Cú pháp*:

**neg DST**

**reg hoặc mem**

*Cờ*: **C, P, A, Z, S** và **O**

- Lệnh CMP

*Chức năng*: So sánh nội dung 2 toán hạng và dựng cờ (phục vụ cho các lệnh nhảy có điều kiện).

*Chú ý*: Sau khi thực hiện giá trị 2 toán hạng không thay đổi.

*Cú pháp*:

**cmp DST,SRC**

*Cờ*: **C, P, A, Z, S** và **O**

- Lệnh MUL (với số không dấu)/IMUL (với số có dấu)

*Chức năng*: Nhân nội dung AL hoặc AX với nội dung của toán hạng nguồn :

- Với phép nhân 2 toán hạng 8 bit:
  AL*SRC và tích sẽ đặt trong thanh ghi AX
- Với phép nhân 2 toán hạng 16 bit:
  AX*SRC và tích sẽ đặt trong 2 thanh ghi DX:AX

*Cú pháp*:

**mul SRC**
**reg**
**mem**

*Cờ*: **C, P, A, Z, S** và **O**

hoặc

**imul SRC**
**reg**
**mem**

*Cờ*: **C, P, A, Z, S** và **O**

- Lệnh DIV (với số không dấu)/IDIV (với số có dấu)

*Chức năng*: Chia nội dung AX hoặc DX:AX cho nội dung của toán hạng nguồn SRC:

- Với phép chia 16 bit cho toán hạng 8 bit:

$\frac{AX}{SRC}$ và kết quả: AL chứa thương và AH chứa phần dư

- Với phép chia 32 bit cho toán hạng 16 bit:

$\frac{DX:AX}{SRC}$ và kết quả AX chứa thương và DX chứa phần dư

*Cú pháp*:

| | | | | |
|---|---|---|---|---|
| **div** | **SRC**<br>**reg**<br>**mem** | hoặc | **idiv** | **SRC**<br>**reg**<br>**mem** |

*Chú ý*: Trong các lệnh MUL/IMUL và DIV/IDIV thì sau lệnh chỉ có 1 toán hạng và tùy thuộc vào toán hạng đứng sau lệnh có kích cỡ như thế nào sẽ suy ra toán hạng kia (ẩn) nằm ở đâu. Phép nhân và chia luôn sử dụng thanh ghi AX/AL và DX là ẩn để chứa số hạng còn lại và chứa kết quả.

## *Nhóm 3:* Một số lệnh hay dùng thuộc nhóm thao tác bit

*Chú ý:* Giá trị các bit cờ có thể bị thay đổi trong hầu hết các lệnh.

- Lệnh AND

*Chức năng*: Thực hiện phép VÀ LÔGIC các bit của 2 toán hạng. Kết quả đặt ở toán hạng đích.

*Cú pháp*:

```
and   DST,SRC
        ★
Cờ: C=O=0   ;   P, Z, S
```

Trong lập trình Assembly hay sử dụng lệnh AND cho:

- *Tách bit*: Muốn tách bit nào đó (giữ trạng thái bit đó) của 1 toán hạng thì hãy AND bit đó với 1 và các bit khác với 0.
  *Ví dụ*: AL=xxxxxxxx ; Nếu AND AL,00110000b thì AL=00xx0000b
- *Dựng cờ*: Thực hiện lệnh AND toán hạng với chính nó sẽ dựng các cờ cho biết trạng thái giá trị của toán hạng đó và trên cơ sở các cờ đó thực hiện các lệnh nhảy có điều kiện.
  *Ví dụ*: AND AX,AX giá trị AX không đổi song dựng các cờ cho biết giá trị AX (S=1…âm, S=0…dương, Z=1…0, Z=0… #0).

- Lệnh OR

*Chức năng*: Thực hiện phép HOẶC LÔGIC các bit của 2 toán hạng (bit của toán hạng kết quả bằng 1 khi chỉ cần 1 trong 2 bit tương ứng của 2 toán hạng bằng 1). Kết quả đặt ở toán hạng đích.

*Cú pháp*:

```
or  DST,SRC
      ★
Cờ: C=O=0    ;    P, Z, S
```

- Lệnh XOR

*Chức năng*: Thực hiện phép EXCLUSIVE OR các bit của 2 toán hạng. Kết quả đặt ở toán hạng đích.

*Cú pháp*:

```
xor  DST, SRC
       ★
Cờ: C=O=0    ;    P, Z, S
```

Trong lập trình Assembly thường sử dụng lệnh xor một toán hạng với chính nó để đưa giá trị toán hạng đó về 0.

*Ví dụ*: xor AX,AX thì AX=0

- Lệnh SHL (Shift Left)

*Chức năng*: Dịch trái các bit của toán hạng đi Count lần.


*Cú pháp*:

```
shl   DST,Count
      reg  hoặc  mem
Cờ: C, P, Z, S, O
```

*Ý nghĩa dịch trái 1 lần*: Dịch trái toán hạng 1 lần có nghĩa nhân đôi giá trị của toán hạng nếu toán hạng là nguyên dương.

- Lệnh SHR (Shift right)

*Chức năng*: Dịch phải các bit của toán hạng đi Count lần.


*Cú pháp*:

```
shr   DST,Count
      reg  hoặc  mem
Cờ: C, P, Z, S, O
```

*Ý nghĩa dịch phải 1 lần*: Dịch phải toán hạng 1 lần có nghĩa chia đôi làm tròn dưới nếu toán hạng là nguyên dương.

- Lệnh SAR (Shift Arithmetic Right)

*Chức năng*: Dịch phải các bit của toán hạng đi Count lần như sau:


*Cú pháp*:

```
sar   DST,Count
      reg  hoặc  mem
Cờ: C, P, Z, S, O
```

*Ý nghĩa dịch phải 1 lần*: Dịch phải toán hạng 1 lần có nghĩa chia đôi làm tròn dưới nếu toán hạng là nguyên.

## *Nhóm 4*: Một số lệnh hay dùng thuộc nhóm làm việc với xâu

- Lệnh MOVSB/MOVSW

*Chức năng*: Chuyển xâu ký tự theo từng byte (movsb) hoặc theo từng word (movsw) từ vùng nhớ trỏ bởi DS:SI sang vùng nhớ trỏ bởi ES:DI. Sau mỗi lần chuyển 1 (hoặc 2) byte thì giá trị SI và DI tự động tăng 1 (hoặc 2) khi cờ D=0 (chuyển theo chiều tăng của địa chỉ) hoặc giảm đi 1 (hoặc 2) khi cờ D=1 (chuyển theo chiều giảm của địa chỉ).

*Cú pháp*: **movsb** hoặc **movsw**

- Lệnh LODSB/LODSW

*Chức năng*: Chuyển ký tự của xâu theo từng byte (lodsb) hoặc 2 byte (lodsw) từ vùng nhớ trỏ bởi DS:SI vào thanh ghi AL hoặc AX. Sau mỗi lần chuyển 1 (hoặc 2) byte thì giá trị SI tự động tăng 1 (hoặc 2) khi cờ D=0 (chuyển theo chiều tăng địa chỉ) hoặc giảm đi 1 (hoặc 2) khi cờ D=1 (chuyển theo chiều giảm địa chỉ).

*Cú pháp*: **lodsb** hoặc **lodsw**

- Lệnh STOSB/STOSW

*Chức năng*: Chuyển ký tự của xâu theo từng byte có trong AL (stosb) hoặc 2 byte có trong AX (stosw) đến vùng nhớ trỏ bởi ES:DI. Sau mỗi lần chuyển 1 (hoặc 2) byte thì giá trị DI tự động tăng 1 (hoặc 2) khi cờ D=0 (chuyển theo chiều tăng địa chỉ) hoặc giảm đi 1 (hoặc 2) khi cờ D=1 (chuyển theo chiều giảm địa chỉ).

*Cú pháp*: **stosb** hoặc **stosw**

- Lệnh CMPSB/CMPSW

*Chức năng*: So sánh xâu ký tự theo từng byte (cmpsb) hoặc theo từng word (cmpsw) có trong 2 vùng nhớ trỏ bởi DS:SI và ES:DI. Sau mỗi lần so sánh 1 (hoặc 2) byte thì giá trị SI và DI tự động tăng 1 (hoặc 2) khi cờ D=0 (so sánh theo chiều tăng địa chỉ) hoặc giảm đi 1 (hoặc 2) khi cờ D=1 (so sánh theo chiều giảm địa chỉ).

*Cú pháp*: **cmpsb** hoặc **cmpsw**

- Tiền tố REP đứng trước các lệnh làm việc với xâu

*Chức năng*: Thực hiện lệnh làm việc với xâu (nhóm 4) đứng sau nó một số lần có trong CX cho đến khi CX=0. Sau mỗi lần thực hiện nội dung CX tự động giảm đi 1.

*Cú pháp*:

```
rep   lệnh làm việc với xâu ký tự
```

## *Nhóm 5*: Các lệnh rẽ nhánh

- Lệnh CALL

*Chức năng*: Gọi một chương trình con.

*Cú pháp*:

```
call   tên chương trình con/ địa chỉ 1 ô nhớ
```

- Lệnh RET

*Chức năng*: Quay về chương trình đã gọi nó từ chương trình con.

*Cú pháp*:

```
ret
```

- Lệnh INT

*Chức năng*: Kích hoạt (thực hiện) một ngắt mềm.

*Cú pháp*:

```
int   n        ; (n là số ngắt)
Cờ:  I=T=0
```

- Lệnh IRET

*Chức năng*: Trở về chương trình đã kích hoạt nó sau khi thực hiện chương trình con phục vụ ngắt.

*Cú pháp*:

**iret**
*Cờ*: **C, P, A, Z, S ,O**

- Lệnh JMP

*Chức năng*: Nhảy không điều kiện.

*Cú pháp*: **jmp label/addr/tên CT con/reg** (chứa 1 địa chỉ)**/mem** (chứa 1 địa chỉ)

*Chú ý*: Bước nhảy của lệnh jmp không được quá 64k (1 segment)

- Các lệnh nhảy có điều kiện

Xem xét một số lệnh nhảy có điều kiện:

- Nhảy khi so sánh các số nguyên dương,
- Nhảy khi so sánh các số nguyên và
- Nhảy theo trạng thái các bit cờ.

- *Với số nguyên dương (không dấu)*

cmp DST,SRC

- **jb/jnae label/addr** (khi giá trị **DST** dưới **SRC**)
- **jbe/jna label/addr** (khi **DST** dưới/bằng **SRC**)
- **je label/addr** (khi **DST = SRC**)
- **jne label/addr** (khi **DST ≠ SRC**)
- **ja/jnbe label/addr** (khi giá trị **DST** trên **SRC**)
- **jae/jnb label/addr** (khi **DST** trên/bằng **SRC**)

- *Với số nguyên (có dấu)*

cmp DST,SRC

- **jl/jnge label/addr** (khi giá trị **DST<SRC**)
- **jle/jng label/addr** (khi giá trị **DST≤SRC**)
- **je label/addr** (khi **DST= SRC**)
- **jne label/addr** (khi **DST≠ SRC**)
- **jg/jnle label/addr** (khi giá trị **DST>SRC**)
- **jge/jnl label/addr** (khi giá trị **DST≥RC**)

- *Nhảy theo trạng thái các bit cờ:*

Nhảy theo trạng thái bit cờ Carry:

- **jc label/addr** (khi bit cờ C=1)
- **jnc label/addr** (khi bit cờ C=0)

Nhảy theo trạng thái bit cờ Zero:

- **jz label/addr** (khi bit cờ Z=0)
- **jnz label/addr** (khi bit cờ Z=1)

Nhảy theo trạng thái bit cờ Sign:

- **js label/addr** (khi bit cờ S=1-số âm)
- **jns label/addr** (khi bit cờ S=0–số dương)

- *Chú ý*: Bước nhảy của các lệnh nhảy có điều kiện ≤ 128 byte.

- Lệnh lặp LOOP

*Chức năng*: Lặp việc thực hiện khối lệnh nằm giữa nhãn và loop nhãn cho đến khi CX=0. Sau mỗi lần lặp CX tự động giảm đi 1.

*Cú pháp*:

```
        mov  CX, số lần lặp
nhãn:
        khối lệnh ASM
        loop  nhãn
```

***Nhóm 6*: Các lệnh xác lập trạng thái các bit cờ**

*Chức năng*: Xác lập trạng thái các bit cờ.

*Cú pháp*:

Bit cờ Carry:

**clc** (clear C - C=0) **stc** (set C - C=1) **cmc** (đảo giá trị bit cờ C)

Bit cờ Interrupt:

**cli** (clear I - I=0, cấm ngắt) **sti** (set I - I=1, cho phép ngắt)

Bit cờ Direction:

**cld** (clear D - D=0) **std** (set D - D=1)

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

Các directive không sinh ra mã máy để chạy chương trình mà chỉ hỗ trợ cho CT dịch. Có rất nhiều directive trong đó các directive điều khiển segment là quan trọng hơn cả.

### 1.5.3.1 Các lệnh điều khiển segment (segment directives)

Có 2 dạng directive điều khiển segment: đơn giản và chuẩn.

*A. Các lệnh điều khiển segment dạng đơn giản*

Có 4 directive hay dùng : .MODEL, .STACK, .DATA, .CODE

a. Lệnh điều khiển .MODEL:

*Chức năng*: Báo cho chương trình dịch biết để xác lập bộ nhớ thích hợp cho chương trình.

*Cú pháp*:

| .MODEL kiểu mô hình bộ nhớ | |
|---|---|
| tiny | code+data ≤ 64k |
| small | code ≤ 64k ; data ≤ 64k |
| compact | code ≤ 64k ; data ≥ 64k |
| medium | code ≥ 64k ; data ≤ 64k |
| large | code ≥ 64k ; data ≥ 64k song 1 array ≤ 64k |
| huge | code ≥ 64k ; data ≥ 64k song 1 array ≥ 64k |

b. Lệnh điều khiển .STACK:

*Chức năng*: Báo cho chương trình dịch biết để xác lập 1 vùng nhớ cho ngăn xếp.

*Cú pháp*:

| **.STACK kích cỡ ngăn xếp** (tính theo đơn vị byte) |
|---|
| *Ví dụ* **: .STACK 100h** (xác lập 1 vùng nhớ 256 byte cho ngăn xếp) |

c. Lệnh điều khiển .DATA:

*Chức năng*: Báo cho chương trình dịch biết để xác lập 1 vùng nhớ cho phần dữ liệu (cấp phát cho biến)

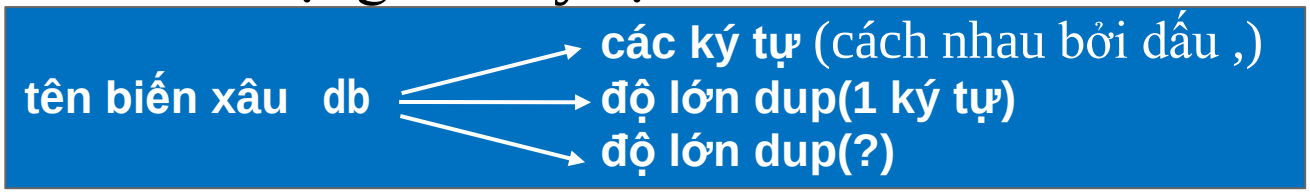
*Cú pháp*:

| **.DATA** |
|---|
| **Phần khai báo biến** |

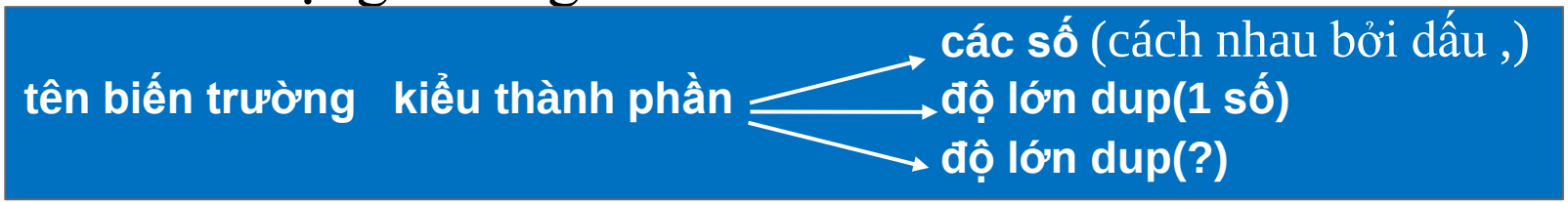
- Khai báo biến dạng số:

| tên biến | kiểu biến | gán giá trị ban đầu hoặc ? |
|---|---|---|
| | db | |
| | dw | |
| | dd | |
| | df/dp | |
| | dq | |
| | dt | |

- Khai báo biến dạng xâu ký tự:




- Khai báo biến dạng trường số:




*Chú ý quan trọng*: Để hoàn tất việc xác định vùng nhớ dữ liệu, người lập trình phải đưa giá trị @data vào DS nhờ 2 lệnh sau:

```
mov   reg16,@data
mov   DS,reg16
```

Thường là:

```
mov   AX,@data
mov   DS,AX
```

d. Lệnh điều khiển .CODE:

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

*Cú pháp*:

```
.CODE
 nhãn CT:
        các lệnh thân chương trình
END   nhãn CT
```

# Một chương trình Assembly đơn giản (sử dụng directive điều khiển segment dạng đơn giản) ví dụ có dạng sau:

```
.MODEL small            ; Mô hình bộ nhớ cho chương trình dạng small
[.STACK  100h]          ; Dành 1 vùng nhớ 256 byte cho ngăn xếp
[.DATA                  ; Dành 1 vùng nhớ để cấp phát cho biến (chỉ có khi
   khai báo biến]       ; chương trình có khai báo biến)
.CODE
nhãn CT:
    [ mov  AX,@data     ; Đưa phần địa chỉ segment vùng nhớ dành cho dữ
      mov    DS,AX ]    ; liệu vào DS (chỉ có khi có .DATA, có khai báo biến)
        các lệnh  thân
        chương trình
    mov   AH,4Ch
    int     21h         ; Trở về DOS
    END  nhãn CT
```

*Ví dụ*: Hãy viết chương trình hiện 1 xâu ký tự lên màn hình

*Cách 1*: Sử dụng hàm hiện 1 xâu ký tự kết thúc bằng '$' (chức năng thứ 9 của ngắt int 21h của DOS)

```
.MODEL  small                    ; Mô hình bộ nhớ small
.STACK  100h                     ; Dành 1 vùng nhớ 256 byte cho ngăn xếp
.DATA
  m   db  ' Hello World!$'       ; Khai báo biến xâu ký tự cần hiện
.CODE
  PS:
        mov     AX,@data         ; Đưa phần địa chỉ segment của vùng nhớ dữ
        mov     DS,AX            ; liệu vào DS
        lea     DX,m             ; Chức năng hiện 1 xâu ký tự  (kết thúc bằng
        mov     AH,9             ; '$') lên màn hình)
        int     21h
        mov     AH,4Ch           ; Chức năng về DOS
        int     21h
        END     PS
```

# *Cách 2*: Khai báo 1 xâu kết thúc \0 (không dùng hàm trên).

```
.MODEL  small
.STACK  100h
.DATA
   m  db  'Hello World !',0          ; Xâu m kết thúc bằng \0
.CODE
  PS:
        mov     AX,@data            ; Đưa phần địa chỉ segment của vùng nhớ dữ
        mov     DS,AX               ; liệu vào DS
        lea     SI,m                ; SI con trỏ offset đầu biến xâu cần hiện
    L1:
        mov     AL,[SI]             ; Đưa 1 byte trỏ bởi SI vào thanh ghi AL
        and     AL,AL               ; Liệu AL = 0  (kết thúc xâu)?
        jz      Exit                ; AL=0 (kết thúc xâu) thì nhảy đến nhãn Exit,
        mov     AH,0Eh              ; còn AL≠ 0 thì hiện ký tự ở AL lên màn hình
        int     10h
        inc     SI                  ; SI trỏ đến byte chứa ký tự tiếp theo của xâu
        jmp     L1                  ; Nhảy về nhãn L1
    Exit:
        mov     AH,4Ch              ; Chức năng về DOS
        int     21h
        END     PS
```

## B. *Các lệnh điều khiển segment dạng chuẩn*

Có 2 directive hay dùng : SEGMENT và ASSUME

a. Lệnh điều khiển SEGMENT:

*Chức năng*: Báo cho chương trình dịch biết để xác lập các segment cho chương trình.

*Cú pháp*:

```
tên segment   SEGMENT   [align  combine use 'class']
                                  options
                thân segment
tên segment   ENDS
```

trong đó:

- tên segment: là 1 định danh bất kỳ,
- options (các tùy chọn): cho phép xác lập và kiểm tra toàn bộ các đặc tính của segment (align: xác định ranh giới segment bắt đầu so với segment khai báo trước nó, combine: cho phép đặt segment vào vùng nhớ yêu cầu, phương thức gộp các segment cùng tên trong CT đa tệp, use: quản lý segment dạng 64 k hay 4 GB và 'class': điều khiển thứ tự sắp xếp các segment khi liên kết).

Dùng directive SEGMENT để xác lập 3 segment của chương trình.

| *Dạng đơn giản* | *Dạng chuẩn* |
|---|---|
| [.STACK 100h] | [_stack segment<br>db 100h dup(?)<br>_stack ends] |
| [.DATA<br>khai báo biến]<br>*Chú ý*: mov ax,@data<br>mov ds,ax | [data segment<br>khai báo biến<br>data ends]<br>*Chú ý:* mov ax,data<br>mov ds,ax |
| .CODE<br>nhãn CT:<br>các lệnh thân chương trình<br>END nhãn CT | code segment<br>nhãn CT:<br>các lệnh thân chương trình<br>code ends<br>END nhãn CT |

b. Lệnh điều khiển ASSUME:

*Chức năng*: Báo cho chương trình dịch biết segment khai báo thuộc loại segment nào.

*Cú pháp*: assume tên thanh ghi segment: tên segment hay nhóm segment

# Một chương trình Assembly đơn giản (sử dụng directive điều khiển segment dạng chuẩn), ví dụ có dạng:

```
_stack  segment                ; Dành vùng nhớ 256 byte cho ngăn xếp
    db 100h dup(?)
_stack  ends
[data  segment                 ; Dành 1 vùng nhớ để cấp phát cho biến (chỉ có khi
   khai báo biến               ; chương trình có khai báo biến)
data  ends]
code  segment
assume CS:code, DS:data, SS:_stack ; Segment khai báo thuộc loại segment nào
nhãn CT:
        [mov  AX,data          ; Đưa phần địa chỉ segment vùng nhớ  dành cho dữ
         mov  DS,AX]           ; liệu vào DS (chỉ có khi có khai báo data segment)
        các lệnh  thân
        chương trình
        mov  AH,4Ch            ; Trở về DOS
        int    21h
code    ends
        END  nhãn CT
```

*Ví dụ*: Viết chương trình tính 5!

*Cách 1*: Chỉ sử dụng thanh ghi

```
_stack   segment                    ; Dành vùng nhớ 256 byte cho ngăn xếp
    db 100h dup(?)
_stack   ends
code     segment
  assume CS:code, SS:_stack
PS:
         mov    CX,5                ; Gán thanh ghi CX=5
         mov    AX,1                ; Gán AX=1
    L1:
         mul    CX                  ; AX*CX → DX:AX (song DX=0)
         loop   L1
         mov    AH,4Ch              ; Về DOS
         int    21h
code     ends
 END     PS
```

*Cách 2*: Khai báo biến

```
_stack   segment                    ; Dành vùng nhớ 256 byte cho ngăn xếp
    db 100h dup(?)
_stack   ends
```

```
data    segment
  FV    dw    ?
  Fac   dw    ?
data    ends
code    segment
  assume CS:code,  DS:data, SS:_stack
PS:
        mov   AX,data       ; Đưa phần địa chỉ segment của vùng nhớ dữ
        mov   DS,AX         ; liệu vào DS
        mov   FV,1          ; Gán biến FV=1
        mov   Fac,2         ; Gán biến Fac=2
        mov   CX,4          ; Gán thanh ghi CX=4 (chỉ số vòng lặp)
    L1:
        mov   AX,FV         ; Gán AX=FV
        mul   Fac           ; AX*Fac=DX:AX (song DX=0)
        mov   FV,AX
        inc   Fac           ; Tăng Fac lên 1
        loop  L1            ; CX=CX-1 và liệu  CX=0?
        mov   AH,4Ch        ; Về DOS
        int   21h
code    ends
 END    PS
```

## *B. Các lệnh điều khiển khác hay dùng*

a. Lệnh điều khiển SEG:

*Chức năng*: Cho phép lấy phần địa chỉ segment của ô nhớ được cấp phát cho biến nhớ.

*Cú pháp*:

**SEG mem**

phần địa chỉ segment của ô nhớ được cấp phát cho biến.

a. Lệnh điều khiển OFFSET:

*Chức năng*: Cho phép lấy phần địa chỉ offset của ô nhớ được cấp phát cho biến nhớ.

*Cú pháp*:

**OFFSET mem**

phần địa chỉ offset của ô nhớ được cấp phát cho biến.

# Tóm lược bài học

Như vậy bài học đầu đã trình bày phần đầu của Ngôn ngữ Assembly và cách lập trình thông qua việc nêu lên:

- Các đặc tính của ngôn ngữ Assembly,
- Cách cài đặt chương trình dịch,
- Qui trình 4 bước để thực hiện một chương trình Assembly,
- Các khái niệm và sự hỗ trợ của hệ thống cho lập trình Assembly
- Lướt qua một số lệnh hay dùng trong 6 nhóm lệnh mnemonic (instruction set) của ngôn ngữ Assembly, những lệnh sinh ra mã máy để chạy chương trình và
- Các lệnh điều khiển hỗ trợ khi dịch chương trình (directive), đặc biệt là 2 dạng lệnh điều khiển segment dạng đơn giản và chuẩn.