

Cấu Trúc Máy Tính & ASM

Created by mercury

www.updatesofts.com
Ebooks Team

CẤU TRÚC MÁY TÍNH

LẬP TRÌNH HỢP NGỮ



MỤC TIÊU

:

Cấu trúc Máy tính & Lập trình Assembly

1. Khám phá bí mật bên trong máy tính.
2. Trang bị những kiến thức cơ bản về cấu trúc tổng quát của máy tính cũng như các thành phần cấu tạo nên máy
3. Nắm được cách hoạt động, cách giao tiếp của các thành phần cấu tạo nên máy tính.
4. Biết viết 1 chương trình bằng Assembly – dịch liên kết và thực thi chương trình này.
5. Biết lập trình xử lý đơn giản phần cứng, lập trình hệ thống
6. Các khái niệm cơ bản về virus TH - nghiên cứu các kỹ thuật lây lan của virus tin học

Tài liệu tham khảo

- Structured Computer Organization – Andrew Tanenbaum
- Assembly Language For the IBM-PC – Kip R Irvine
- Assembly Programming Language & IBM PC Ythayu – Charles Marut
- Giáo trình Cấu trúc máy tính - Tổng Văn On
- Lập trình Hợp ngữ - Nguyễn Ngọc Tấn - Vũ Thanh Hiên
- Cấu trúc Máy tính - Đại học Bách khoa

Tài liệu tham khảo

- Computer Virus Handbook
- Virus Writing guide Billy Belceb
- The macro virus writing guide
- The little black book of computer viruses
- Một số mẫu chương trình virus (virus file, virus macro)

**Giáo viên : Ngô Phước Nguyên
Email : nguyenktcn@yahoo.com
Mobile: 091-8-380-926**

Đề cương môn học

Chương 1 : Tổ chức tổng quát của hệ thống MT

Chương 2 : Tổ chức CPU

Chương 3 : Mức logic số

Chương 4 : Tổ chức bộ nhớ

Chương 5 : Xuất nhập

Chương 6 : Lập trình Assembly – Tập lệnh

Chương 7 : Cấu trúc điều khiển & Vòng lặp

Chương 8 : Macro & Procedure – nhúng CT Assembly vào ngôn ngữ cấp cao như C...

Chương 9 : Lập trình xử lý màn hình-bàn phím-mouse.

Chương 10 : Lập trình xử lý File

**Chương 11 : Các khái niệm cơ bản về Virus tin học –
phân tích các kỹ thuật lây lan chung của VR tin học và
lây lan trên mạng.**

Chương 1 :CẤU TRÚC TỔNG QUÁT CỦA MỘT HỆ THỐNG MÁY TÍNH

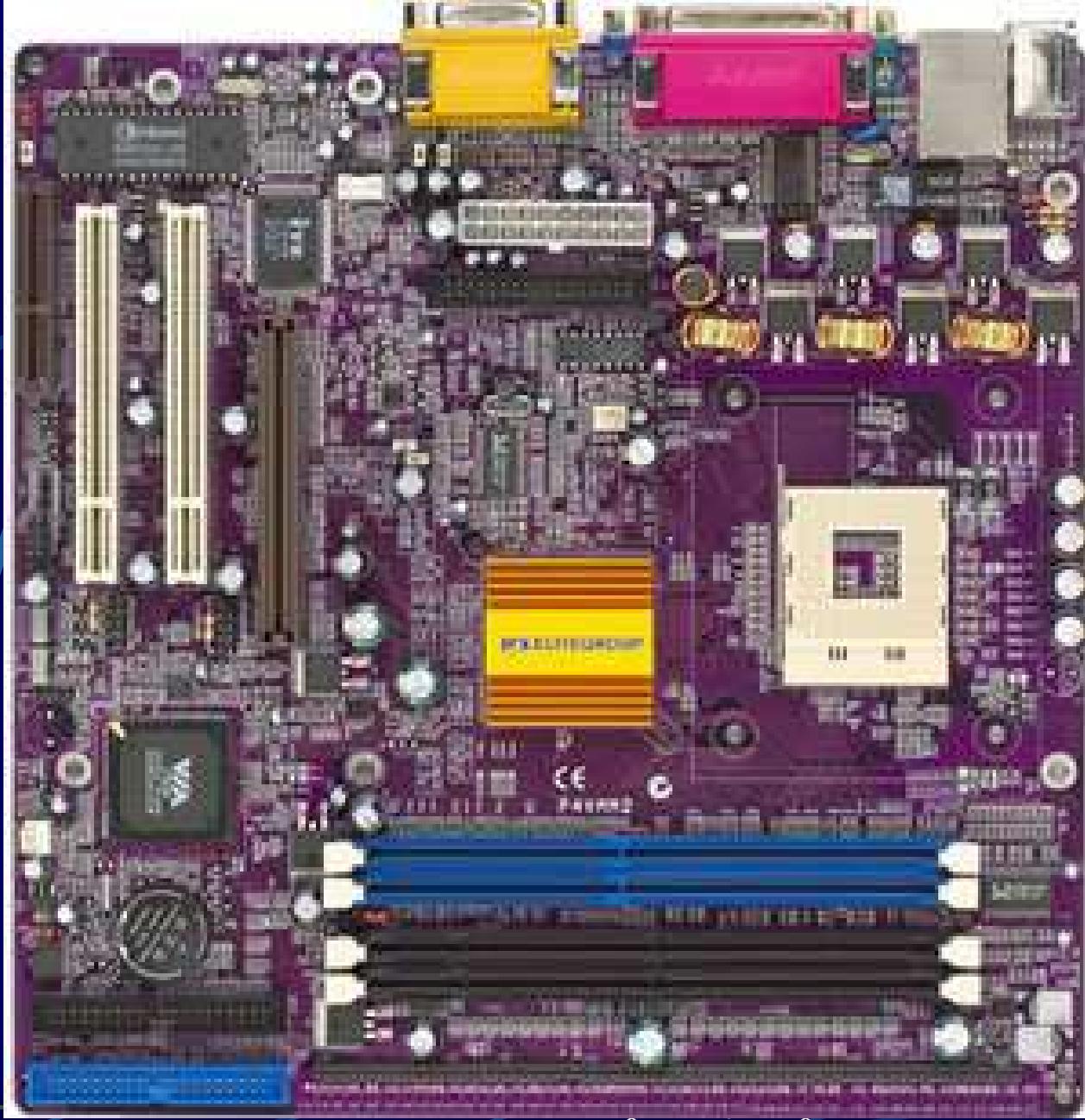
Mục tiêu :

- Nắm được tổng quan về cấu trúc máy tính.
- Hiểu về Máy Turing & Nguyên lý Von Neumann
- Biết sơ đồ khối chi tiết của máy tính
- Nắm nguyên lý hoạt động máy tính
- Biết các component của máy tính :
Processors,Memory,Input/Output devices,Bus

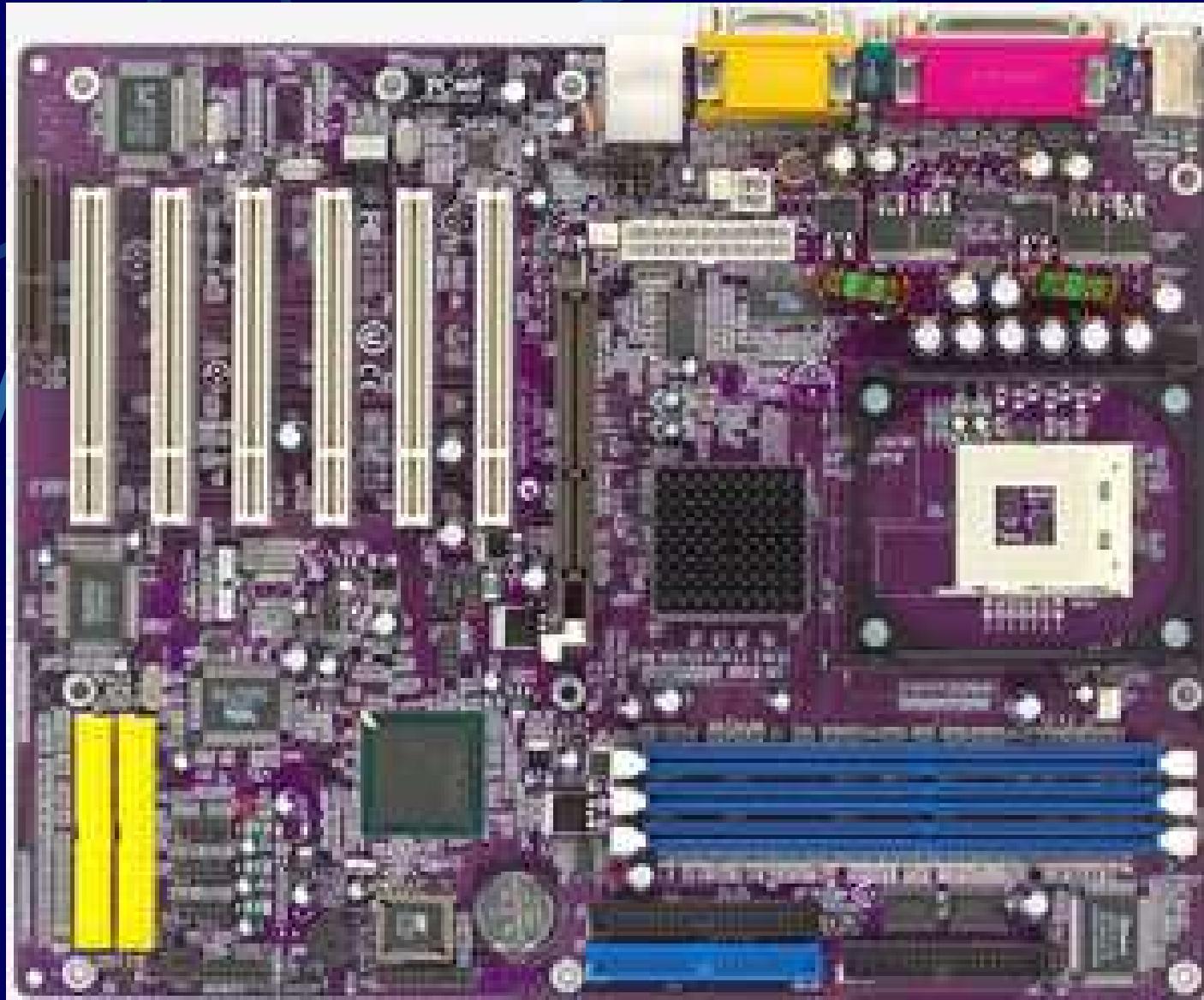
Chương 1

Nội dung

- Tổng quan về cấu trúc máy tính.
- Mô hình máy Turing
- Nguyên lý Von Neumann.
- Sơ đồ tổng quát của một máy tính.
- Nguyên lý hoạt động của máy tính
- Câu hỏi ôn tập



Chuong 1 CÂU TRUC TÔNG
QUÁT CỦA HTMT



Chuong 1 CẤU TRÚC TỔNG
QUÁT CỦA HTMT

Máy tính & Sự tính toán

Memory : chứa các chỉ thị & dữ liệu



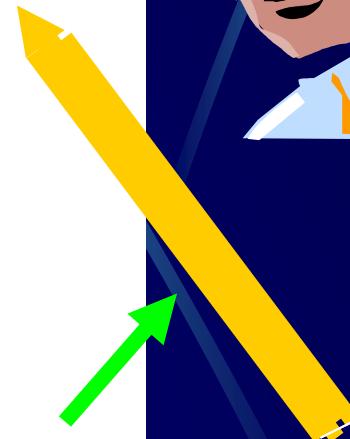
Bộ xử lý

$$2+3/4*3-5=?$$

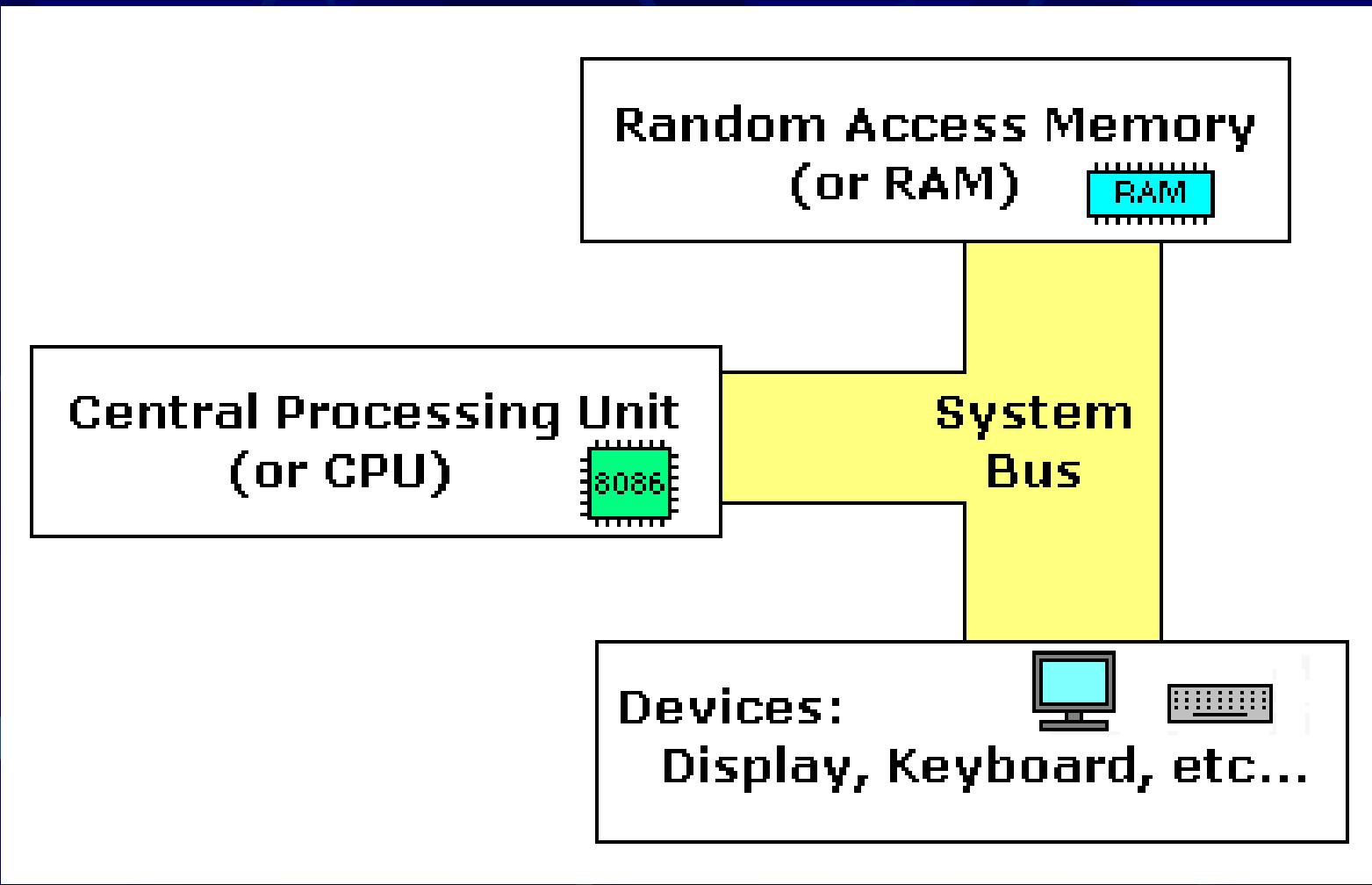
.....

.....

.....



Input device : thiết bị nhập



The system bus (shown in yellow) connects the various components of a computer.

The CPU is the heart of the computer, most of computations occur inside the CPU.

RAM is a place to where the programs are loaded in order to be executed.

Tổng quan về cấu trúc máy tính

Máy tính hiện đại ngày nay được thiết kế dựa trên mô hình Turing Church và mô hình Von Neumann.

Mô hình Turing :

Mô hình này rất đơn giản nhưng nó có tất cả các đặc trưng của 1 hệ thống máy tính sau này. Nguyên lý cấu tạo máy Turing :

đầu đọc ghi

khối xử lý

chứa tập hữu hạn các trạng thái

Băng dữ liệu vô hạn, dữ liệu kết thúc là b



Nguyên lý xây dựng MT

**MT điện tử làm việc theo hai nguyên lý cơ bản :
nguyên lý số và nguyên lý tương tự.**

Nguyên lý số sử dụng các trạng thái rời rạc của 1 đại lượng vật lý để biểu diễn số liệu → nguyên lý đếm.

Nguyên lý tương tự sử dụng 1 đại lượng vật lý biến đổi liên tục để biểu diễn số liệu → nguyên lý đo

Mạch điện trong MT

Trong MT có những loại mạch điện nào ?

Mạch tổ hợp : là mạch điện có trạng thái ngõ ra phụ thuộc tức thời vào tổ hợp của trạng thái ngõ vào.

Ex : Mạch giải mã địa chỉ

Mạch tuần tự : là mạch điện thực hiện 1 mục đích mà trạng thái ngõ ra phụ thuộc vào tổ hợp của trạng thái ngõ vào và trạng thái của quá khứ ngõ vào.

Ex : mạch cộng, trừ, nhân , chia

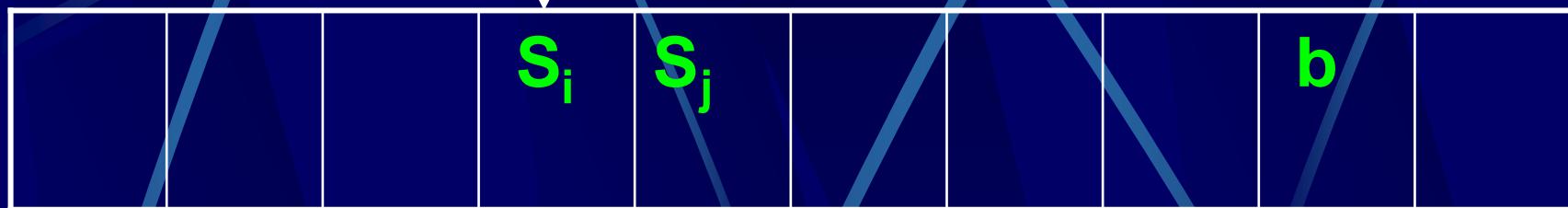
Nguyên lý Turing

khối xử lý

chứa tập hữu hạn các trạng thái

đầu đọc ghi

Băng dữ liệu vô hạn, dữ liệu kết thúc là b



Máy làm việc theo từng bước rời rạc. Một lệnh của máy như sau : $q_i S_i S_j X q_j$.

Nghĩa là : đầu đọc ghi đang ở ô S_i thì sẽ ghi đè S_j vào ô hiện tại và dịch chuyển hoặc đứng yên theo chỉ thị là X và trạng thái hiện hành của máy là q_j

Nguyên lý hoạt động máy Turing

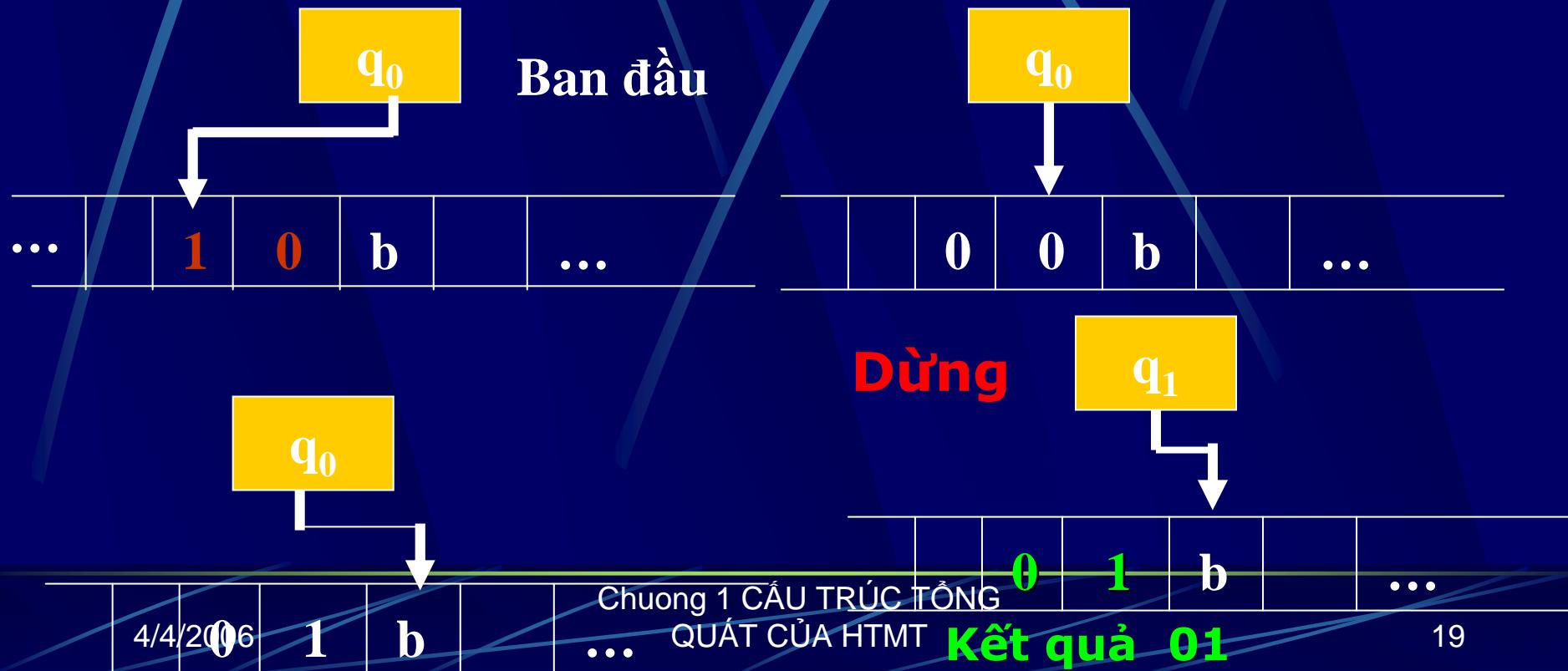
Dữ liệu của bài toán là 1 chuỗi các ký hiệu thuộc tập các ký hiệu của máy không kể ký hiệu rỗng b, được cắt vô băng.

- Trạng thái trong ban đầu của máy là q_0 .
- Đầu đọc/ghi ở ô chứa ký hiệu đầu tiên của chuỗi ký hiệu nhập. Trong quá trình hoạt động, sự thay đổi dữ liệu trên băng, sự dịch chuyển đầu đọc ghi và sự biến đổi trạng thái trong của máy sẽ diễn ra tuân theo các lệnh thuộc tập lệnh của máy tùy theo trạng thái hiện tại và ký hiệu ở ô hiện tại.
- Quá trình sẽ dừng lại khi trạng thái trong của máy là trạng thái kết thúc q_f .

Thí dụ máy Turing

Xét thí dụ máy Turing thực hiện phép toán NOT trên chuỗi các bit 0/1. Chuỗi dữ liệu nhập ban đầu là 10

- tập các ký hiệu của máy {0,1}
- tập các trạng thái trong { q_0, q_1 }
- tập lệnh gồm 3 lệnh : q_001Rq_0 , q_010Rq_0 , q_0bbNq_1

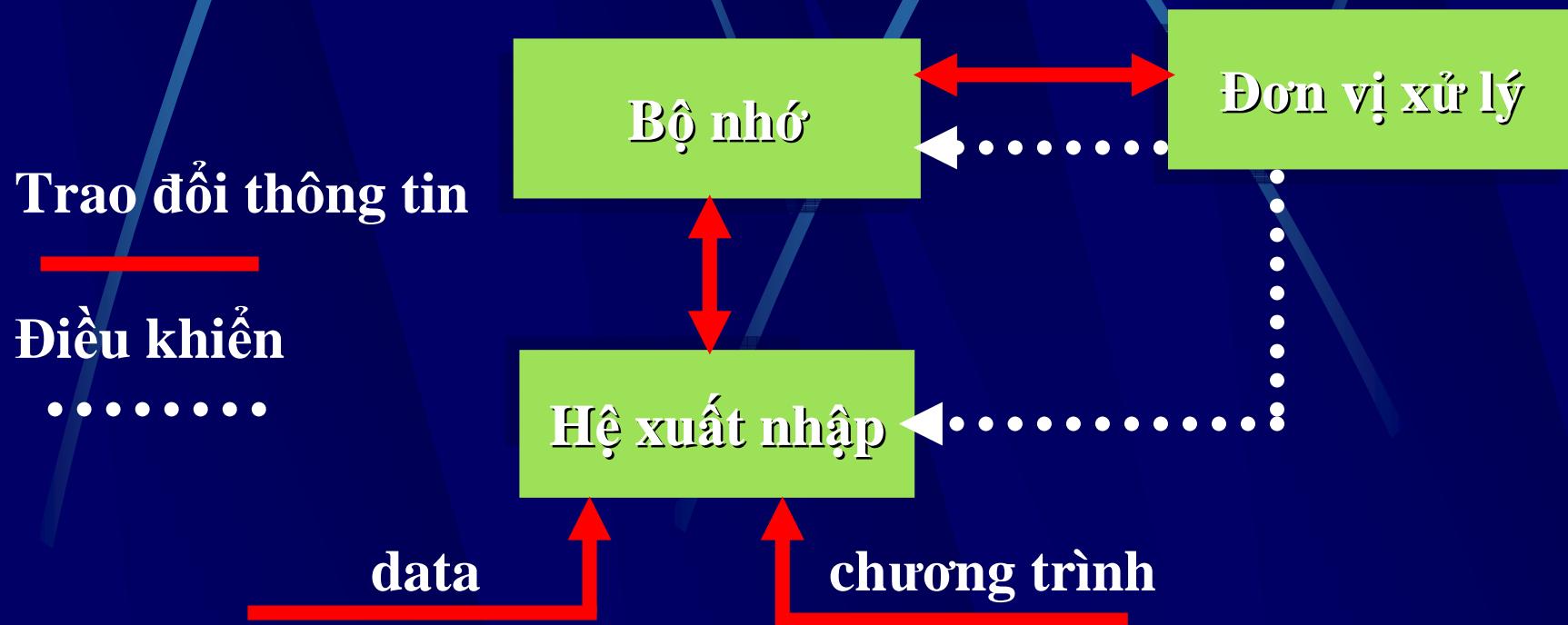


Nguyên lý VonNeumann

Máy Von Neumann là mô hình của các máy tính hiện đại.

Nguyên lý của nó như sau :

Về mặt logic (chức năng) , máy gồm 3 khối cơ bản : đơn vị xử lý, bộ nhớ và hệ thống xuất nhập.



Nguyên lý Von Neumann (cont)

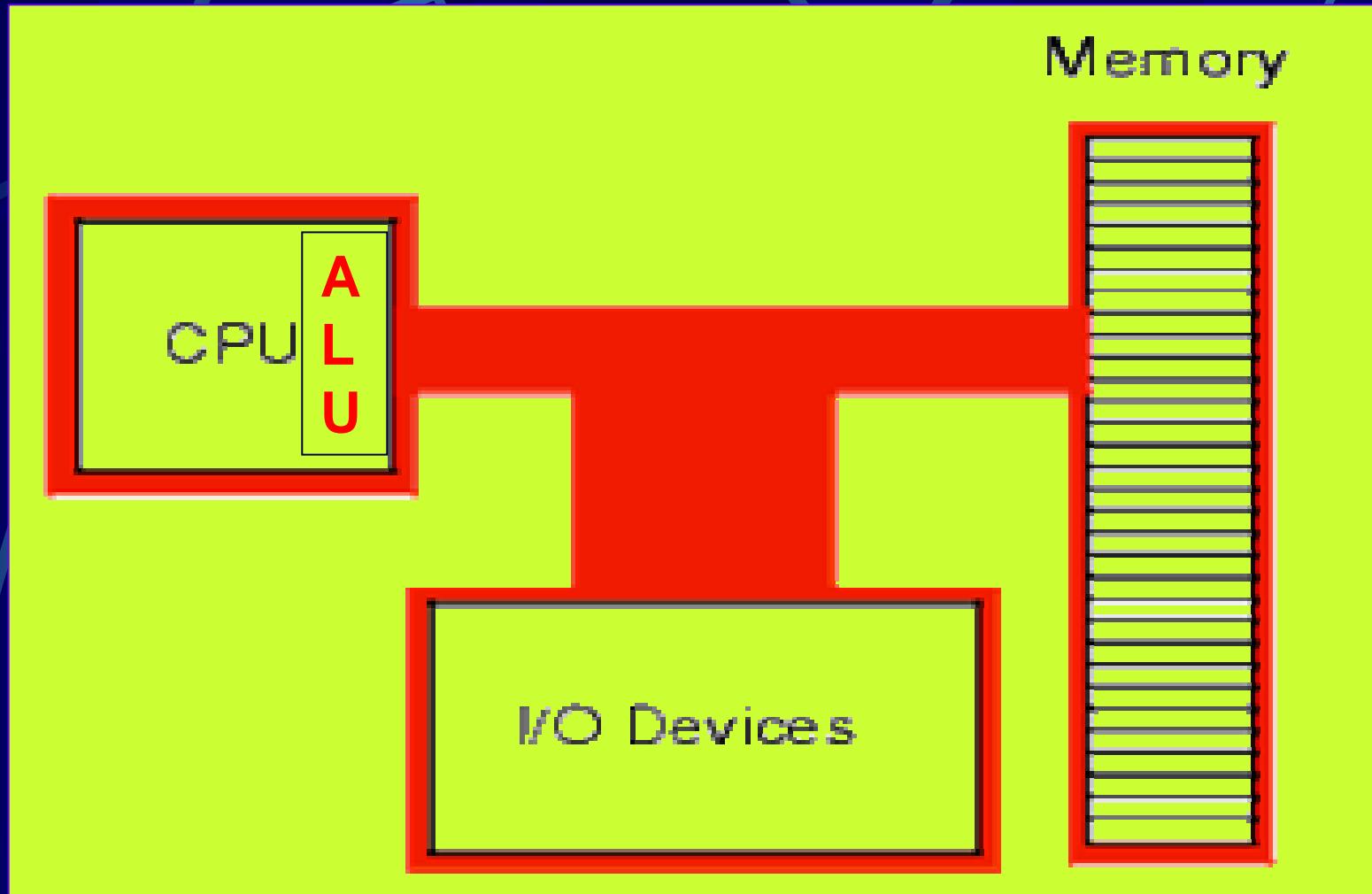
- Chương trình điều khiển xử lý dữ liệu cũng được xem là data và được lưu trữ trong bộ nhớ gọi là chương trình lưu trữ.
- Bộ nhớ chia làm nhiều ô, mỗi ô có 1 địa chỉ (đánh số thứ tự) để có thể chọn lựa ô nhớ trong quá trình đọc ghi dữ liệu. (nguyên lý định địa chỉ)

Nguyên lý Von Neumann (cont)

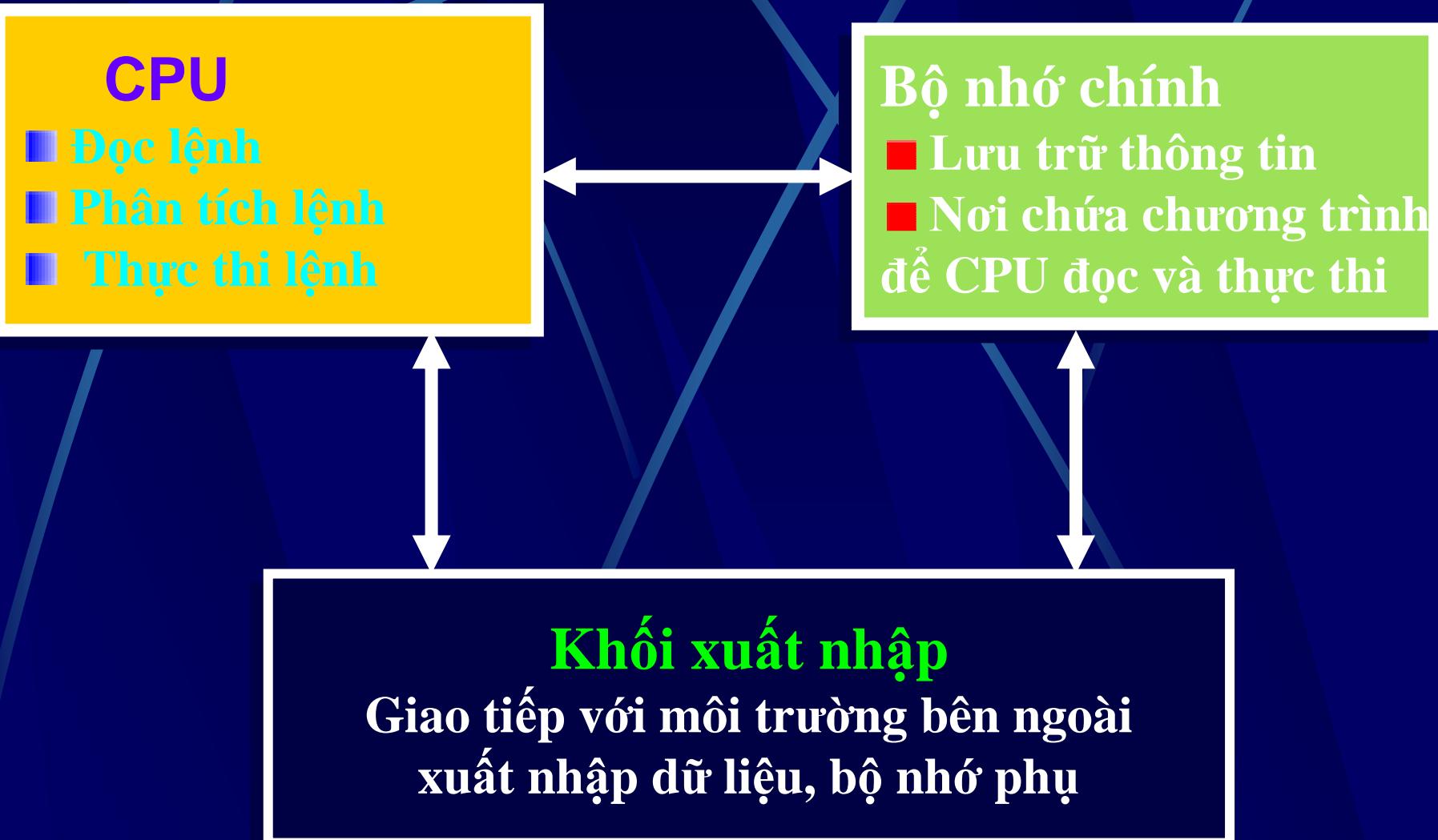
- Các lệnh được thực hiện tuần tự nhờ 1 bộ đếm chương trình (thanh ghi lệnh) nằm bên trong đơn vị xử lý.

Chương trình MT có thể biểu diễn dưới dạng số và đặt vào trong bộ nhớ của MT bên cạnh dữ liệu.

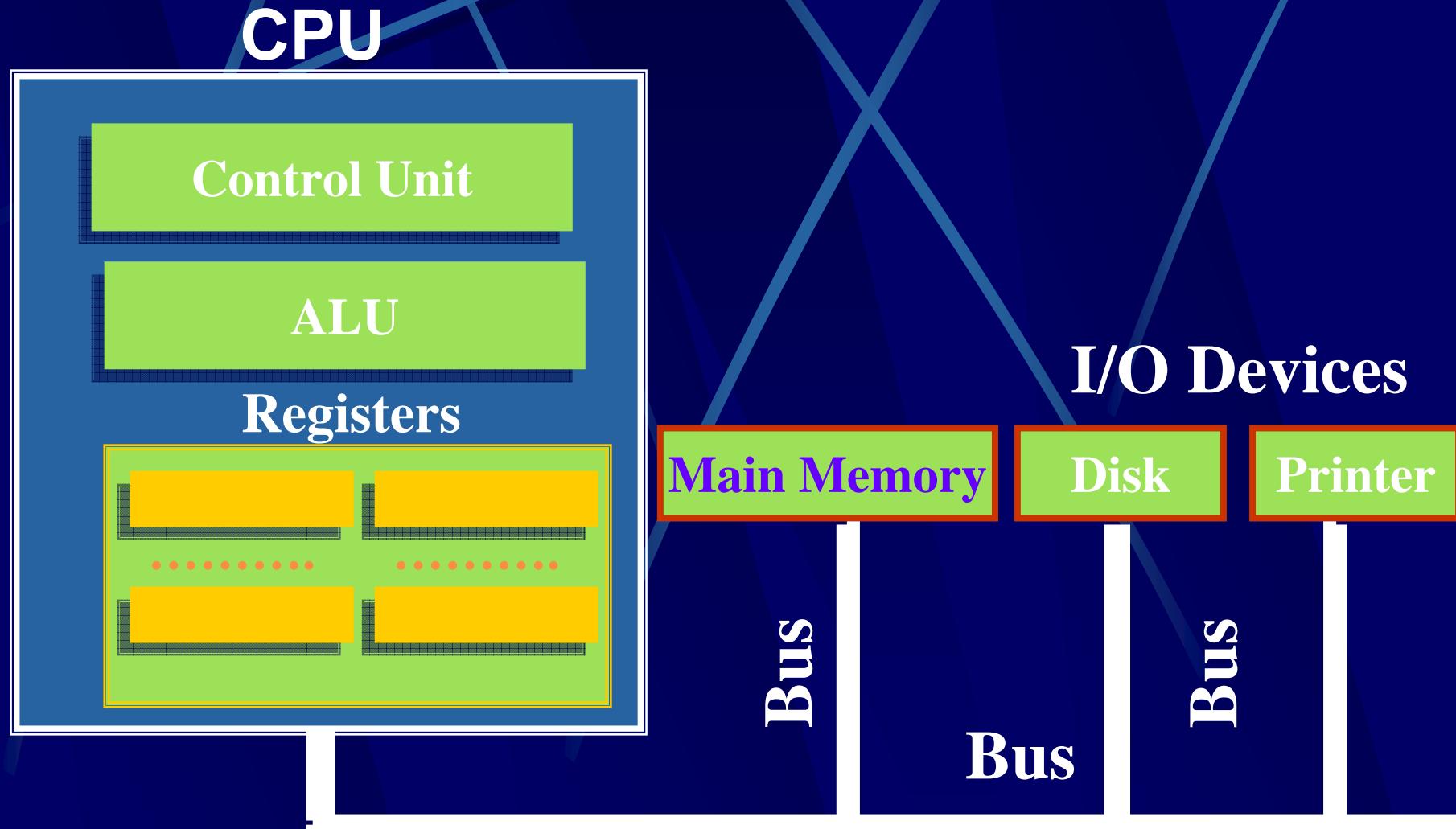
Typical Von Neumann Machine



Nguyên lý hoạt động MT



Tổ chức Máy tính 1 CPU & 2 I/O device



Sơ đồ khối chi tiết

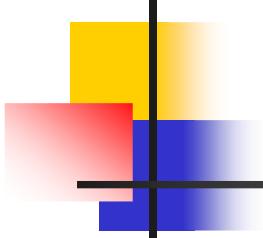


Tổng kết chương

- Máy tính được thiết kế trên ý tưởng của Máy Turing và nguyên lý Von Neumann.
- Về mặt chức năng máy tính gồm 3 phần : đơn vị xử lý, bộ nhớ chính và các thiết bị xuất nhập.

Câu hỏi

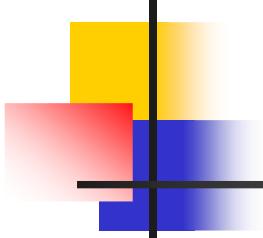
- Câu 1: Trình bày nguyên lý Von Neumann.
- Câu 2: Cho biết sự khác nhau giữa mô hình Turing và mô hình VonNeumann.
- Câu 3: Trình bày nguyên lý hoạt động của Máy Turing.
- Câu 4: Trước khi có nguyên lý Von Neumann, chương trình để máy tính thực hiện được để ở đâu?
- Câu 5 : Cho biết kết quả của $2+3$?



Chương 2 : Tổ chức CPU

Mục tiêu :

- **Nắm được chức năng của CPU**
- **Hiểu được các thành phần bên trong CPU.**
- **Nắm được cách CPU giao tiếp với thiết bị ngoại vi.**
- **Biết được các đặc tính của CPU họ Intel**

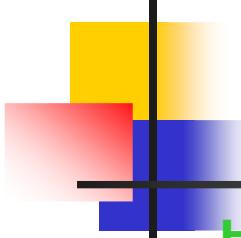


NỘI DUNG

- 2.1 Giới thiệu hệ thống số
- 2.2 Bộ xử lý trung tâm CPU
- 2.3 Hệ thống Bus
- 2.4 Bộ thanh ghi
- 2.5 Cơ chế định vị địa chỉ
- 2.6 Các đặc tính thiết kế liên quan đến hiệu suất CPU họ Intel
- 2.7 Các đặc trưng của CPU họ Intel
- 2.8 Câu hỏi ôn tập

2.1 Hệ thống số

Hệ đếm	Cơ số	số ký số	dạng ký số và ký tự biểu diễn số
nhi phân	2	2	0 1 Ex : 1010_b
bát phân	8	8	0 1 2 3 4 5 6 7 Ex : 24_o
thập phân	10	10	0 1 2 3 4 5 6 7 8 9 Ex : 12_d
thập lục phân	16	16	0 1 2 3 4 5 6 7 8 9 A B C D E F Ex : 3F8_h



Hệ thống số

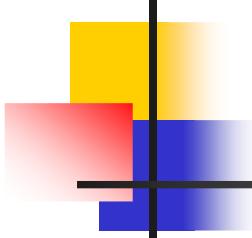
Hệ thống số là gì ?

**Vào thời điểm đó, việc dùng các que để đếm là 1 ý tưởng vĩ đại!!
Còn việc dùng các ký hiệu thay cho các que đếm còn vĩ đại hơn!!!!
Một trong các cách để biểu diễn 1 số hiện nay là sử dụng hệ thống
số đếm decimal.**

**Có nhiều cách để biểu diễn 1 giá trị số. Ngày xưa, con người dùng
các que để
đếm sau đó đã học vẽ các hình trên mặt đất và trên giấy.
Thí dụ số 5 lần đâu được biểu diễn bằng | | | | | (bằng 5
que).**

**Sau đó chữ số La Mã bắt đầu dùng các ký hiệu khác nhau để biểu
diễn nhiều số gọn hơn.**

**Thí dụ số 3 vẫn biểu diễn bởi 3 que | | | nhưng số 5 thì được thay
bằng V còn số 10 thì thay bằng X.**

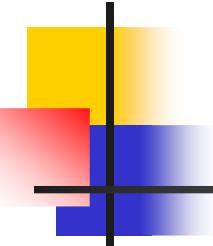


Hệ thống số

Sử dụng que để đếm là 1 ý nghĩa vĩ đại ở thời điểm này. Và việc dùng các ký hiệu để thay cho các que đếm càng vĩ đại hơn!!!.

Một trong những cách tốt nhất hiện nay là dùng hệ thống số thập phân (decimal system).

Decimal System



Con người ngày nay dùng hệ 10 để đếm. Trong hệ 10 có 10 digits

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

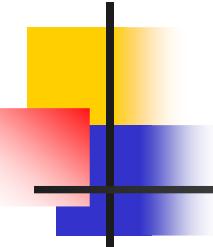
Những ký số này có thể biểu diễn bất kỳ 1 giá trị nào, thí dụ :

754

$$7 \cdot 10^2 + 5 \cdot 10^1 + 4 \cdot 10^0 = 700 + 50 + 4 = 754$$

base

digit
position



Vị trí của từng ký số rất quan trọng, thí dụ nếu ta đặt "7" ở cuối thì:

547

nó sẽ là 1 giá trị khác :

$$5 \cdot 10^2 + 4 \cdot 10^1 + 7 \cdot 10^0 = 500 + 40 + 7 = 547$$

base

digit
position

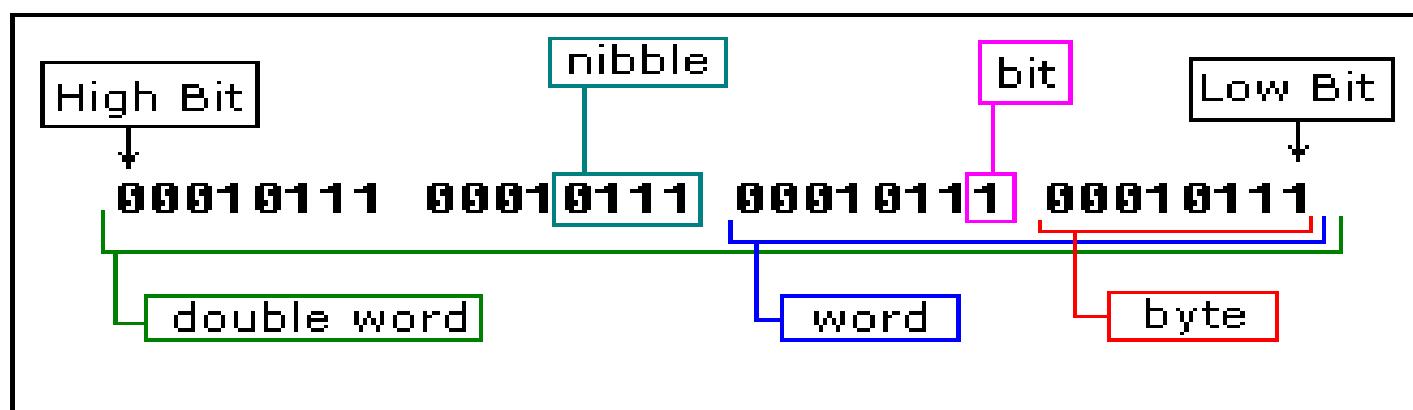
Binary System

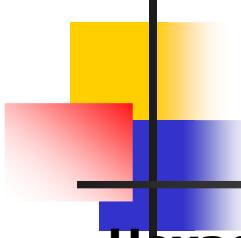
MT không thông minh như con người, nó dùng trạng thái của điện tử :
on and off, or 1 and 0.

MT dùng binary system, binary system có 2 digits:
0, 1

Như vậy cơ số (base) là 2.

Mỗi ký số (digit) trong hệ binary number được gọi là BIT, 4 bits nhóm thành 1 NIBBLE, 8 bits tạo thành 1 BYTE, 2 bytes tạo thành 1 WORD, 2 words tạo thành 1 DOUBLE WORD (ít dùng):





Hexadecimal System

Hexadecimal System

Hexadecimal System dùng 16 digits:

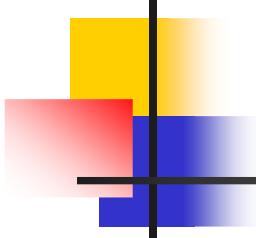
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

do đó cơ số (**base**) là **16**.

Hexadecimal numbers are compact and easy to read.

Ta dễ dàng biến đổi các số từ binary system sang hexadecimal system and và ngược lại, mỗi nibble (4 bits) có thể biến thành 1 hexadecimal digit :

Ex : **1234_h** = 4660_d



Các phép toán trong hệ nhị phân

cộng :

$$0 + 0 = 0 \quad 0 + 1 = 1 \quad 1 + 0 = 1 \quad 1 + 1 = 0 \text{ nhớ } 1$$

trừ : $0 - 0 = 0 \quad 0 - 1 = 1$ mượn 1 $1 - 0 = 1 \quad 1 - 1 = 0$

Nhân : có thể coi là phép cộng liên tiếp

Chia : có thể coi là phép trừ liên tiếp

Các phép toán trong hệ nhị phân ...

Bảng phép tính Logic cho các số nhị phân

A	B	A and B	A or B	A xor B	Not A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

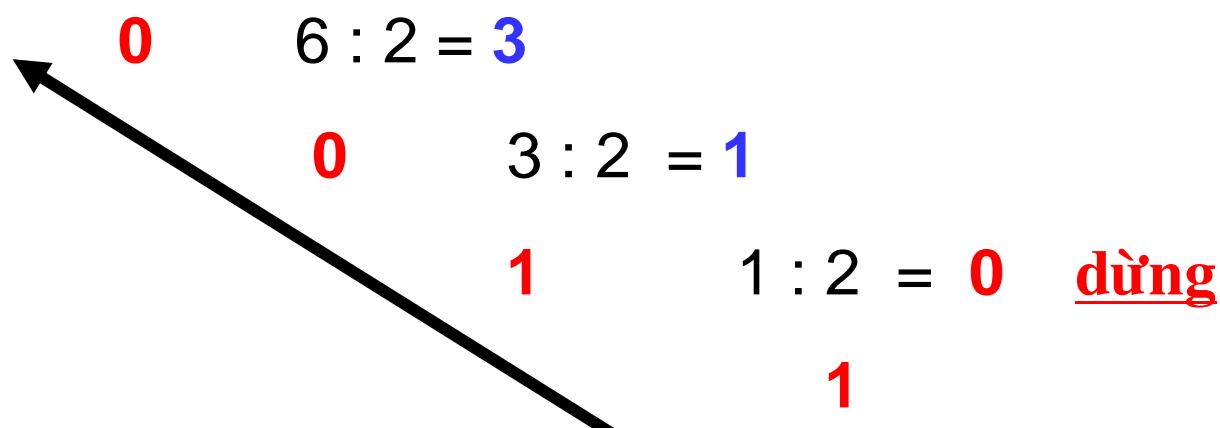
Chuyển hệ từ 10 → hệ 2

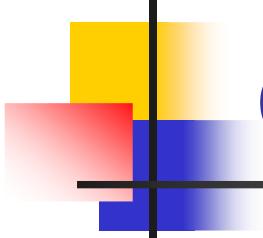
Đổi từ hệ 10 → hệ 2 :

$$\text{Ex : } 12_{\text{d}} = 1100_{\text{b}}$$

Cách đổi : lấy số cần đổi chia liên tiếp cho 2, dừng khi số bị chia bằng 0. Kết quả là các số dư lấy theo chiều ngược lại.

$$12 : 2 = 6$$





Chuyển hệ từ hệ 2 → hệ 10

Đổi từ hệ 2 → hệ 10 :

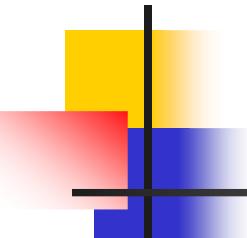
$$\text{Ex : } 1100_b = ?_d$$

Cách đổi : $\sum a_i * 2^i$ với $i \in 0...n$

a là ký số của số cần đổi.

$$1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 0 * 2^0 = 12_d$$

↓
a



Chuyển hệ từ hệ 10 → hệ 16

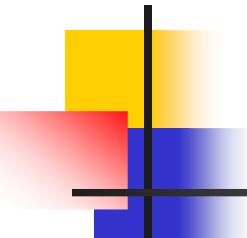
Đổi từ hệ 10 → hệ 16 :

Ex : $253_d = ?_h$

Cách đổi : lấy số cần đổi chia liên tiếp cho 16, dừng khi số bị chia = 0. Kết quả là chuỗi số dư lấy theo chiều ngược lại.

$$253_d = FD_h$$





Chuyển hệ từ hệ 2 → hệ 16

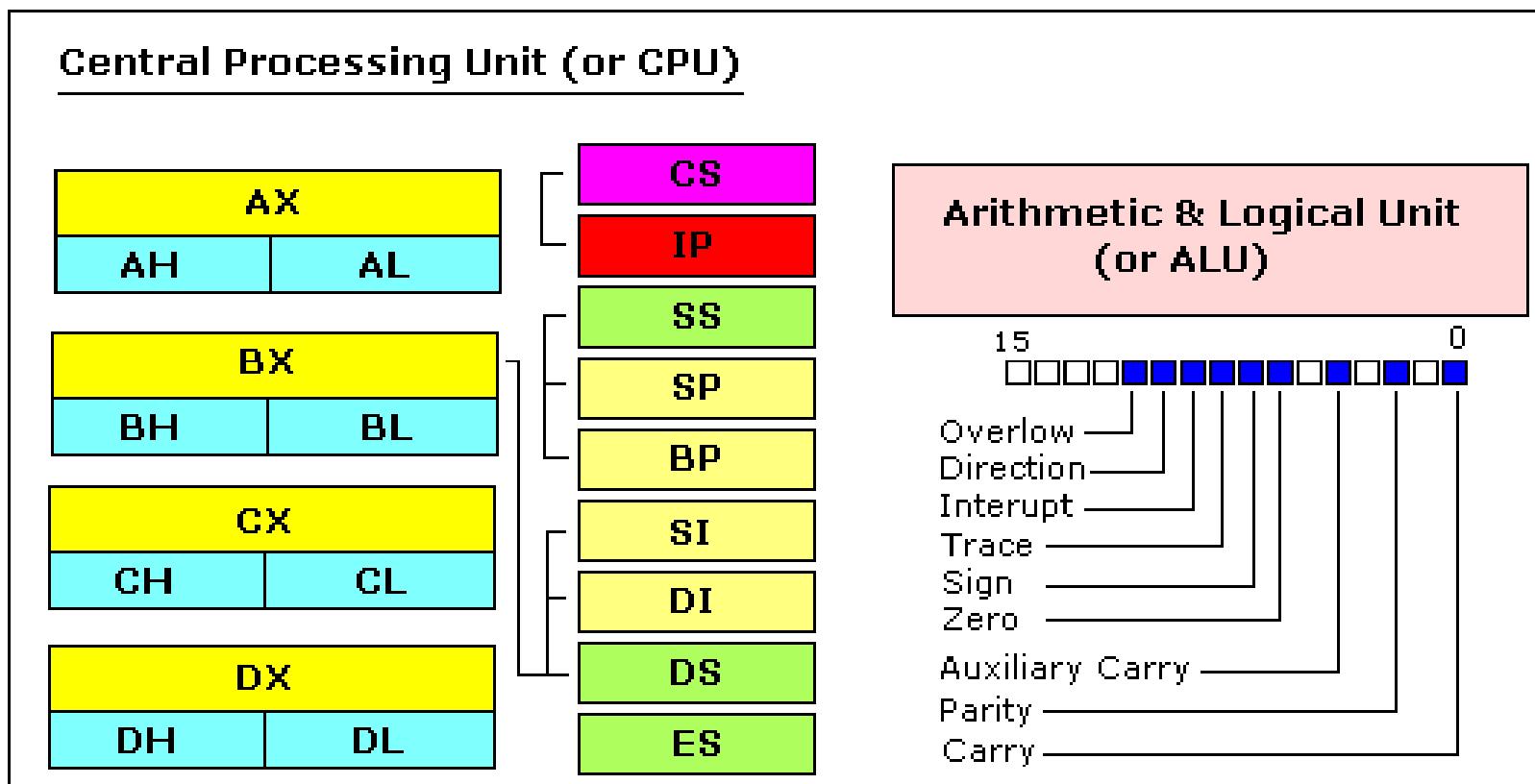
Đổi từ hệ 2 → hệ 16 :

$$\text{Ex : } 101011010_b = ?_h$$

Cách đổi : nhóm 4 chữ số nhị phân thành từng nhóm, rồi chuyển đổi từng nhóm sang số hệ thập lục phân.

$$\begin{array}{r} \underline{0001} \underline{0101} \underline{1010} \\ 1 \quad \quad \quad 5 \quad \quad \quad A \end{array}$$

2.2 Bộ xử lý trung tâm CPU



2.2 Bộ xử lý trung tâm CPU

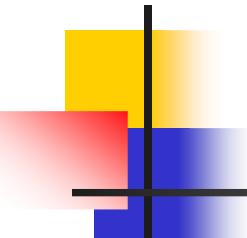
CPU (Central Processing Unit) Bộ xử lý trung tâm –

Chức năng : thực hiện chương trình lưu trong bộ nhớ chính bằng cách lấy lệnh ra - khảo sát - thực hiện lần lượt các lệnh.

Mỗi CPU có 1 tập lệnh riêng. Chương trình được thực thi ở CPU nào sẽ chỉ gồm các lệnh trong tập lệnh của CPU đó.

CPU gồm 1 số bộ phận tách biệt :

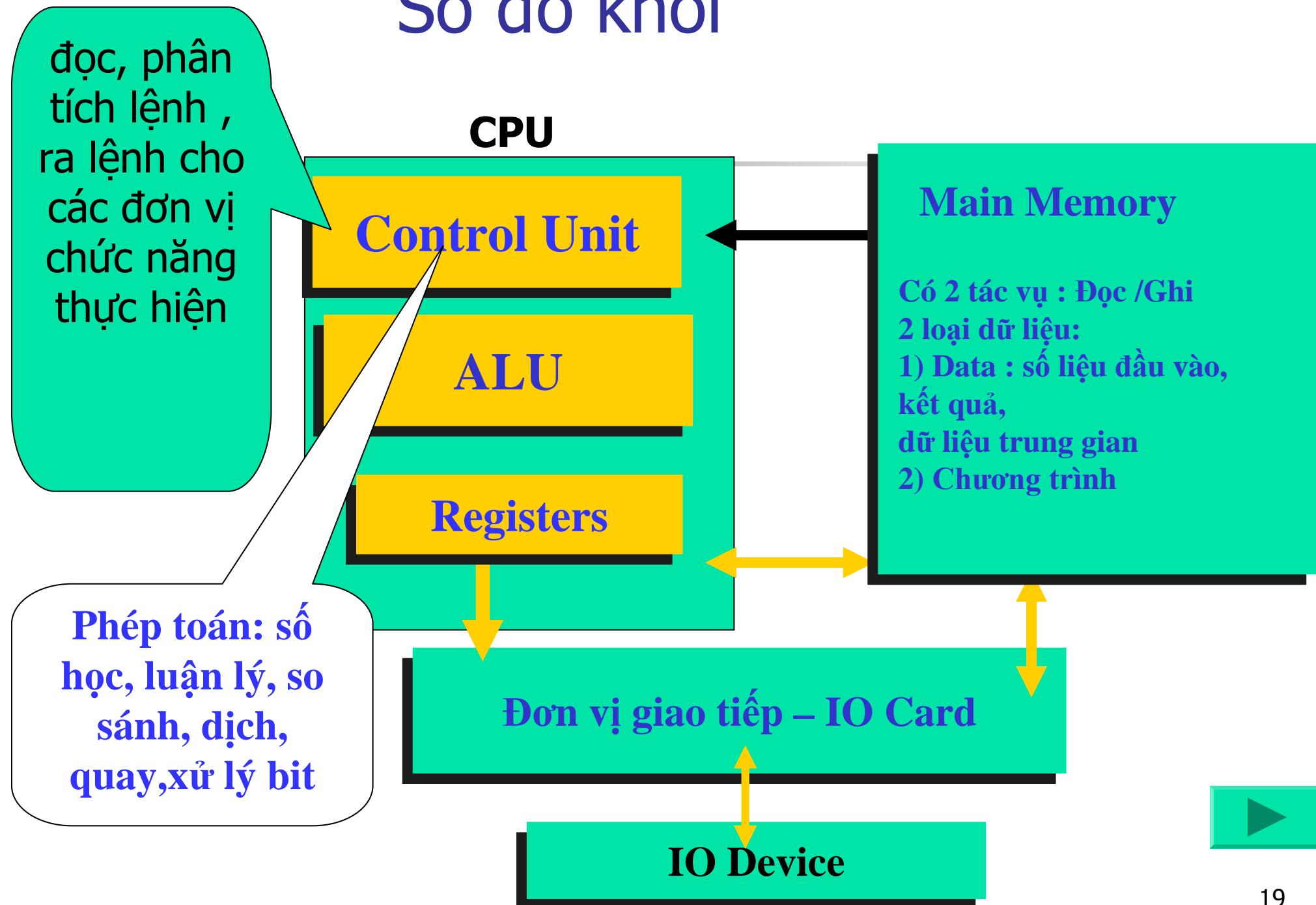
- **Bộ điều khiển lấy lệnh ra từ bộ nhớ và xác định kiểu lệnh.** 
- **Bộ luận lý và số học (ALU) thực hiện phép toán như cộng, and.**
- **Các thanh ghi (Registers) :** lưu kết quả tạm thời và các thông tin điều khiển. CPU giao tiếp với các bộ phận khác trong máy tính thông qua các tuyến gọi là Bus

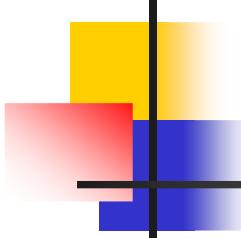


CPU (cont)

- Các nhà chế tạo CPU qui định tốc độ thực hiện của từng chip phù hợp với nhịp tim của chip đó (clock speed) tốc độ đồng hồ, nhịp đồng hồ.
- Đơn vị đo tốc độ của chip CPU là Mhz cho biết chip đập bao nhiêu nhịp trong 1 s.
Ex : CPU 500Mhz.

Sơ đồ khối



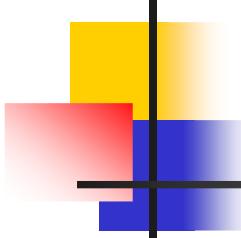


Chu kỳ lệnh

Một chu kỳ thực hiện lệnh máy gồm 3 giai đoạn chính sau :

1. **Lấy lệnh** : lệnh cất ở ô nhớ sẽ được lấy vào thanh ghi lệnh.
2. **Giải mã và thực hiện lệnh** : lệnh trong thanh ghi lệnh sẽ được giải mã và thực hiện theo mô tả của lệnh trong tập lệnh.
3. **Xác định địa chỉ của lệnh tiếp theo** : trong khi lệnh được thực hiện, giá trị của bộ đếm chương trình sẽ tự động tăng lên chỉ đến ô nhớ chứa lệnh sẽ được thực hiện tiếp theo.

Chu kỳ lệnh được xây dựng từ những đơn vị cơ bản là chu kỳ máy.

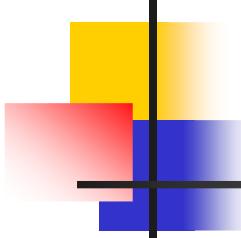


Chu kỳ máy

Chu kỳ máy là chu kỳ của 1 hoạt động cơ bản của máy tính như :

- Chu kỳ đọc bộ nhớ
- Chu kỳ ghi bộ nhớ
- Chu kỳ đọc toán hạng
- Chu kỳ ghi kết quả

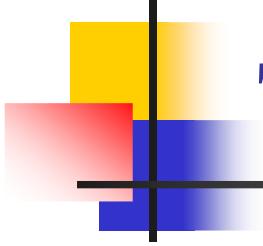
Clock : xung làm nhiệm vụ định thì cho mạch tuần tự.



Thực hiện lệnh

CPU thực hiện lệnh tuân tự theo chuỗi các bước :

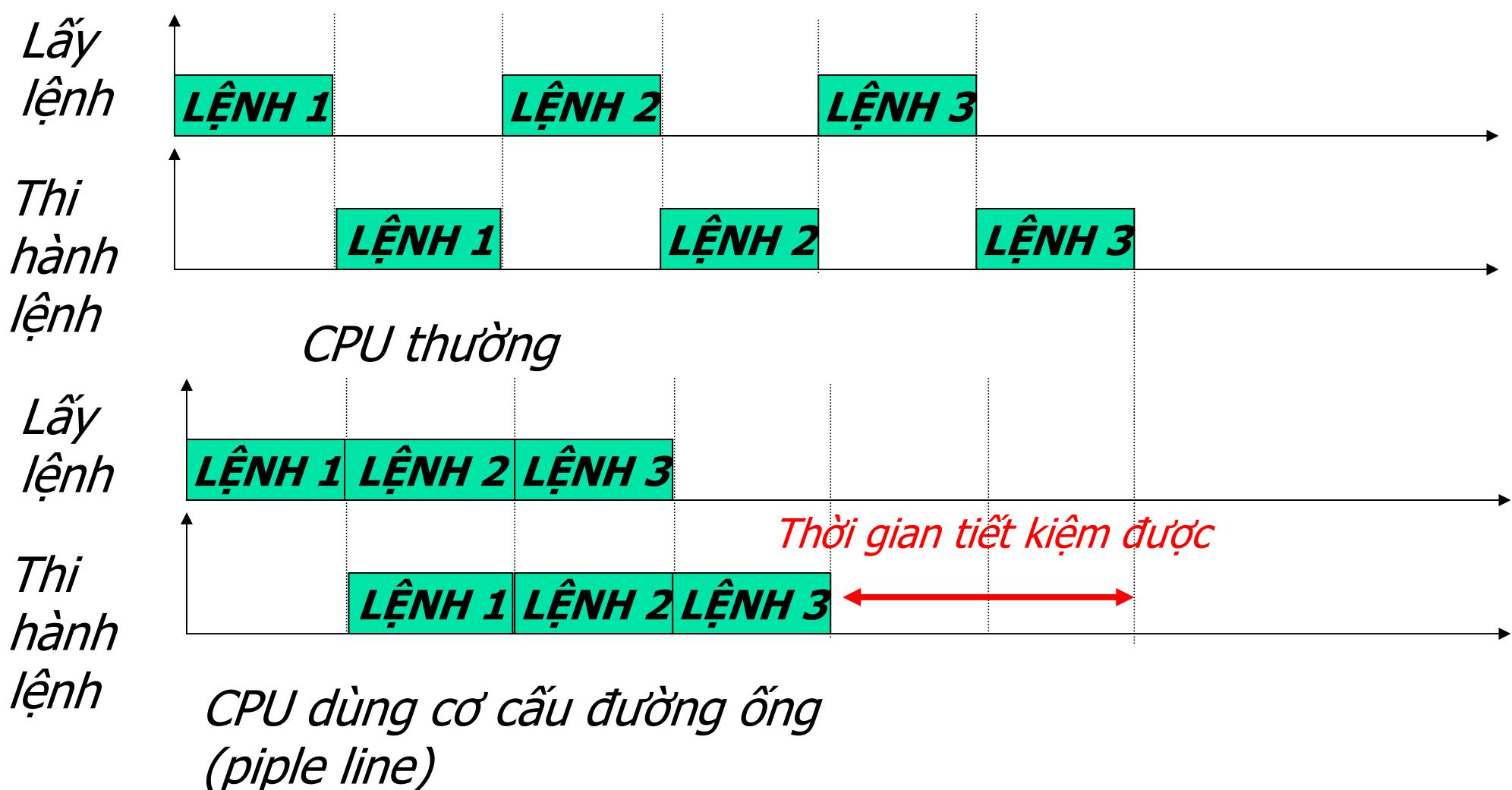
- Lấy lệnh kế từ bộ nhớ. → thanh ghi lệnh.
- Thay đổi PC để chỉ đến lệnh kế tiếp.
- Xác định kiểu lệnh vừa lấy ra.
- Xác định kiểu dữ liệu vừa yêu cầu và xác định vị trí dữ liệu trong bộ nhớ.
- Nếu lệnh cần dữ liệu trong bộ nhớ, nạp nó vào thanh ghi của CPU

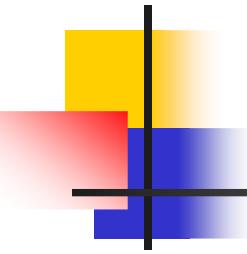


Thực hiện lệnh (cont)

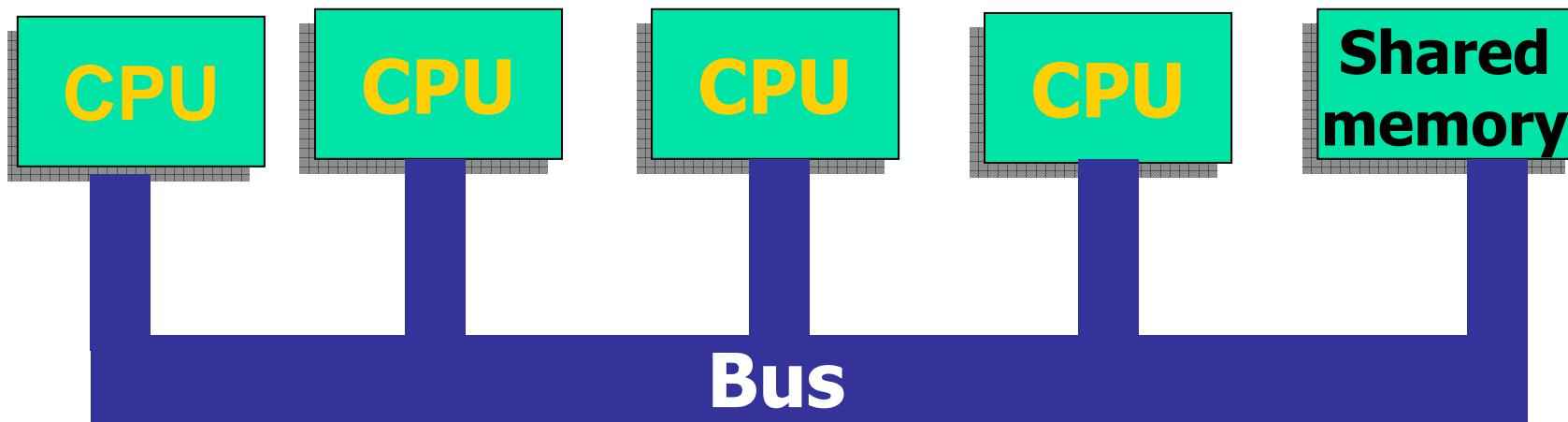
- Thực hiện lệnh..
- Lưu kết quả ở nơi thích hợp..
- Trở về bước 1 để thực hiện lệnh kế.

Sự phân phối thời gian cho 2 quá trình lấy lệnh và thi hành lệnh của CPU thường và CPU đường ống



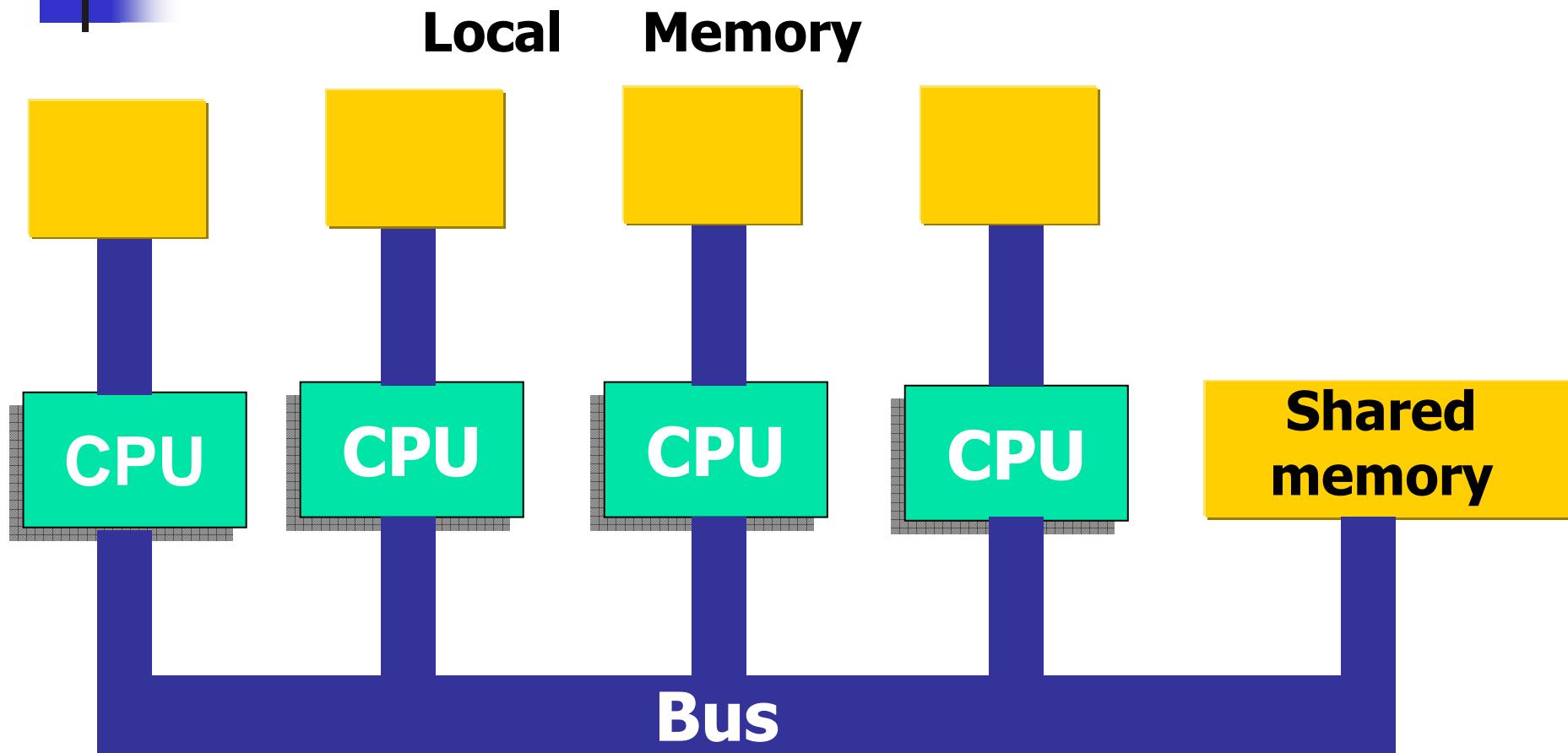


Hệ đa bộ xử lý (MultiProcessor)

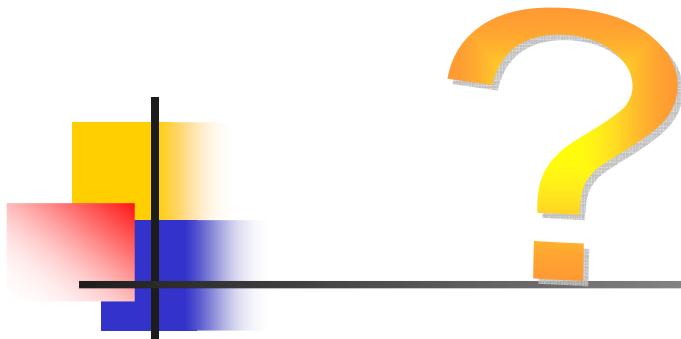


Hệ MultiProcessor sử dụng 1 đường Bus

Hệ đa bộ xử lý (MultiProcessor)

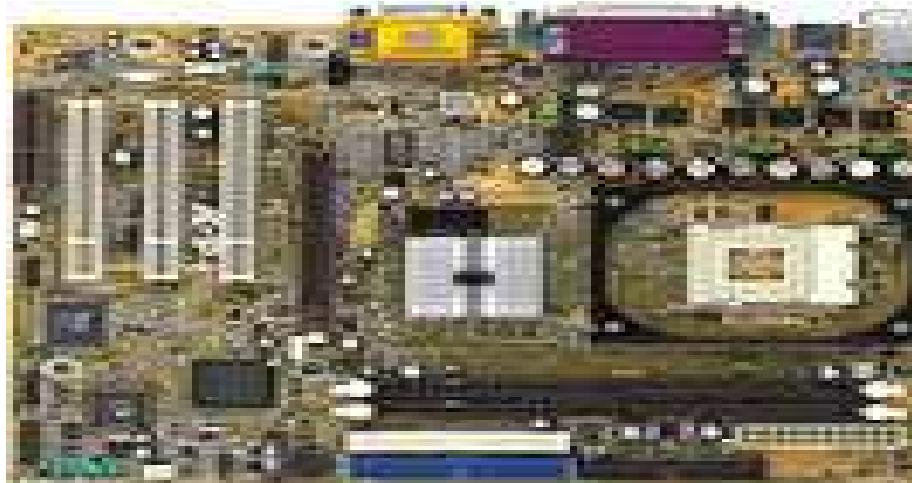


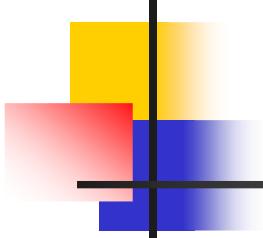
Hệ MultiProcessor sử dụng nhiều bộ nhớ cục bộ



Bus

Bus là các đường truyền. Thông tin sẽ được chuyển qua lại giữa các thành phần linh kiện thông qua mạng lưới gọi là các Bus.



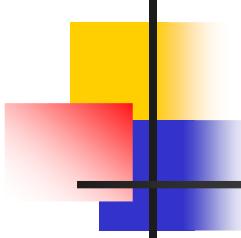


2.3 Hệ thống Bus

Các thiết bị ngoại vi kết nối với hệ thống nhờ các khe cắm mở rộng (expansion slot).

Bus hệ thống (Bus system) sẽ kết nối tất cả các thành phần lại với nhau.

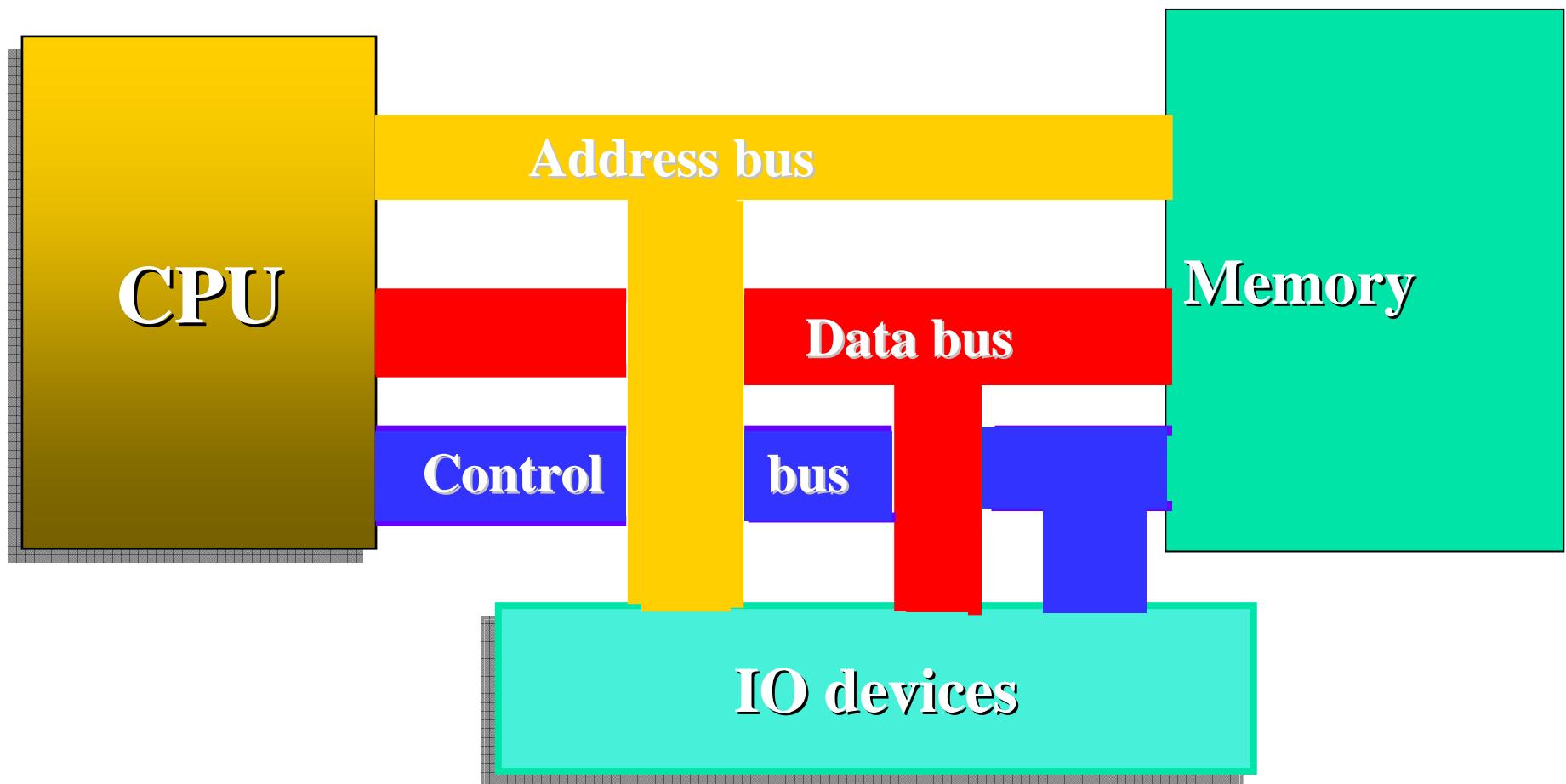
Có 3 loại bus :bus dữ liệu (data bus), bus địa chỉ (address bus) và bus điều khiển (control bus).



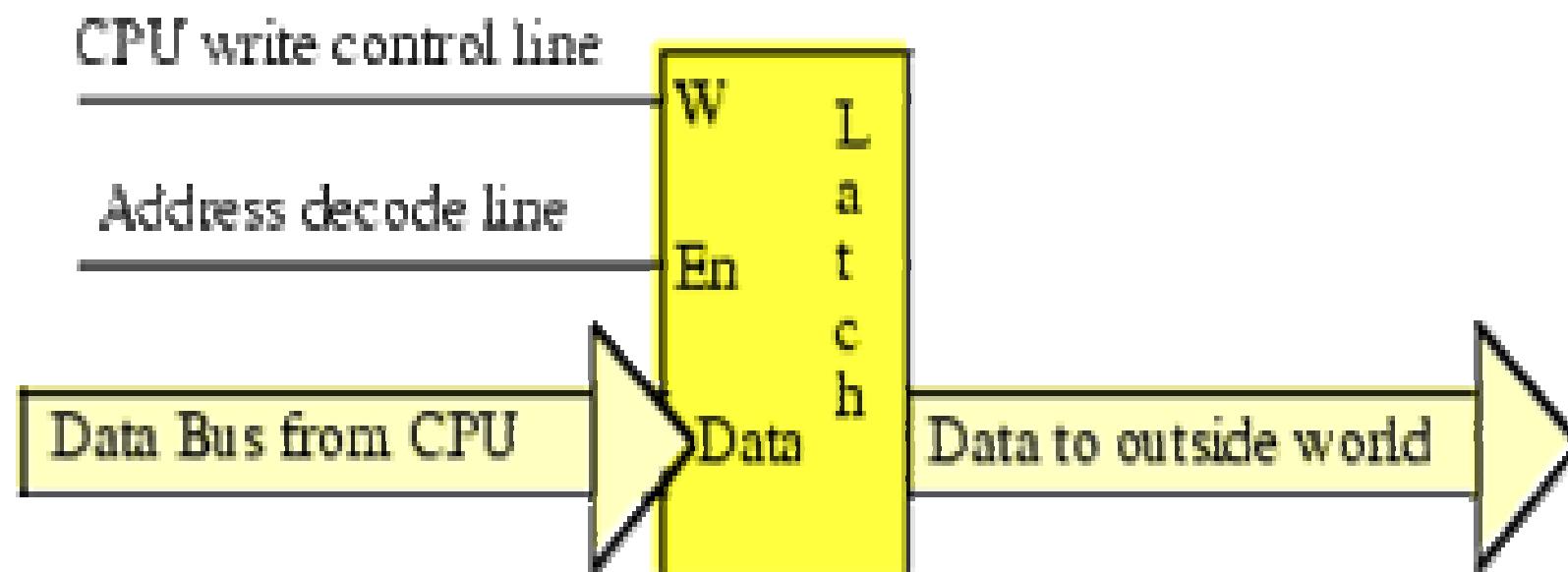
Các loại Bus

- **Address Bus** : nhóm đường truyền nhận diện vị trí truy xuất trong thiết bị đích : thông tin được đọc từ đâu hoặc ghi vào đâu.
- **Data Bus** : nhóm đường truyền để tải data thực sự giữa các thiết bị hệ thống do địa chỉ trên address bus đã xác định. Độ rộng của data bus (số đường dây dẫn) xác định data trong mỗi lần truyền là bao nhiêu.
- **Control Bus** : nhóm đường truyền cho các tín hiệu điều khiển như : tác vụ là đọc hay ghi, tác vụ thực thi trên bộ nhớ hay trên thiết bị ngoại vi, nhận dạng chu kỳ bus và khi nào thì hoàn tất tác vụ...

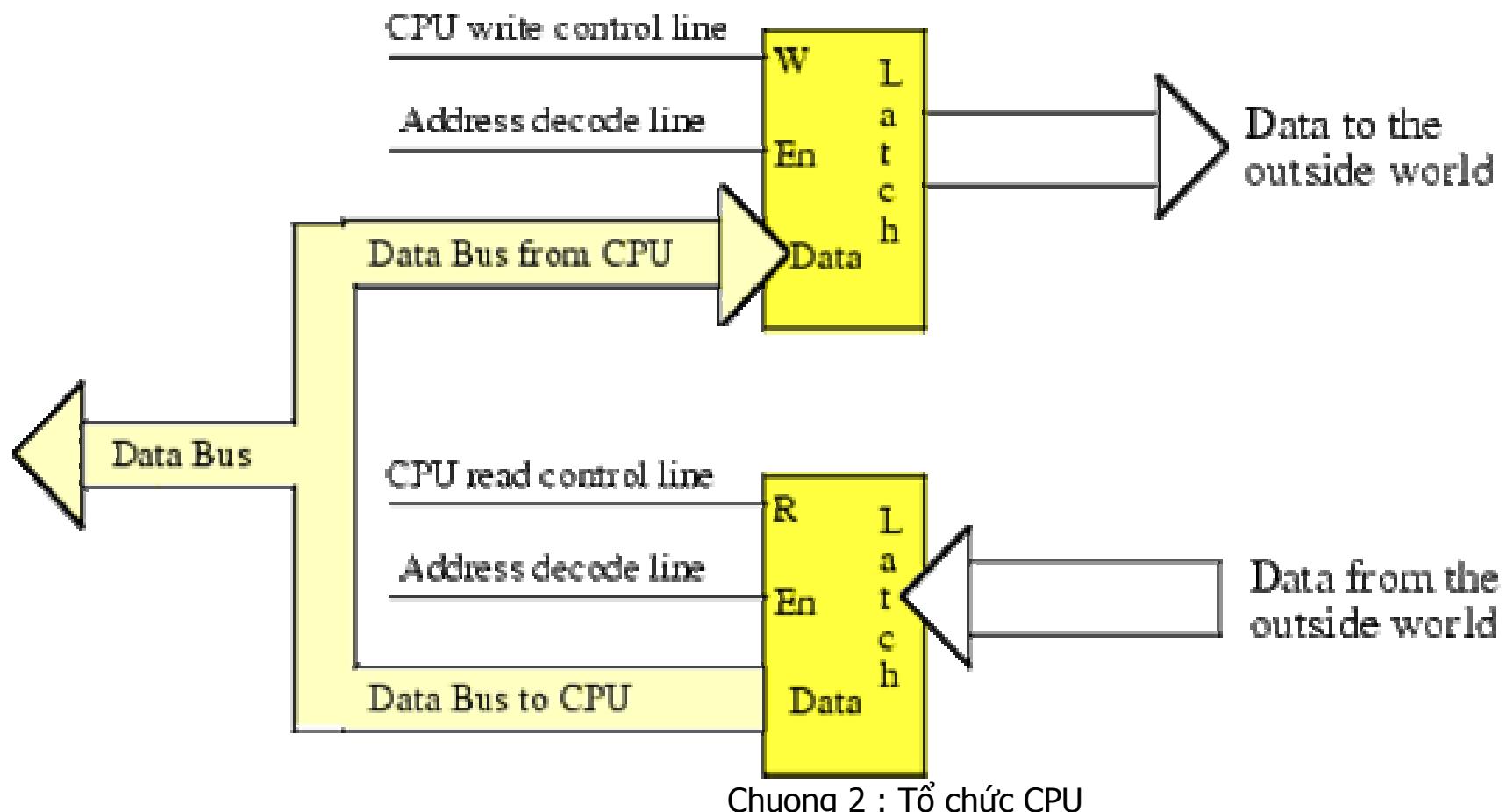
Minh họa hệ thống Bus



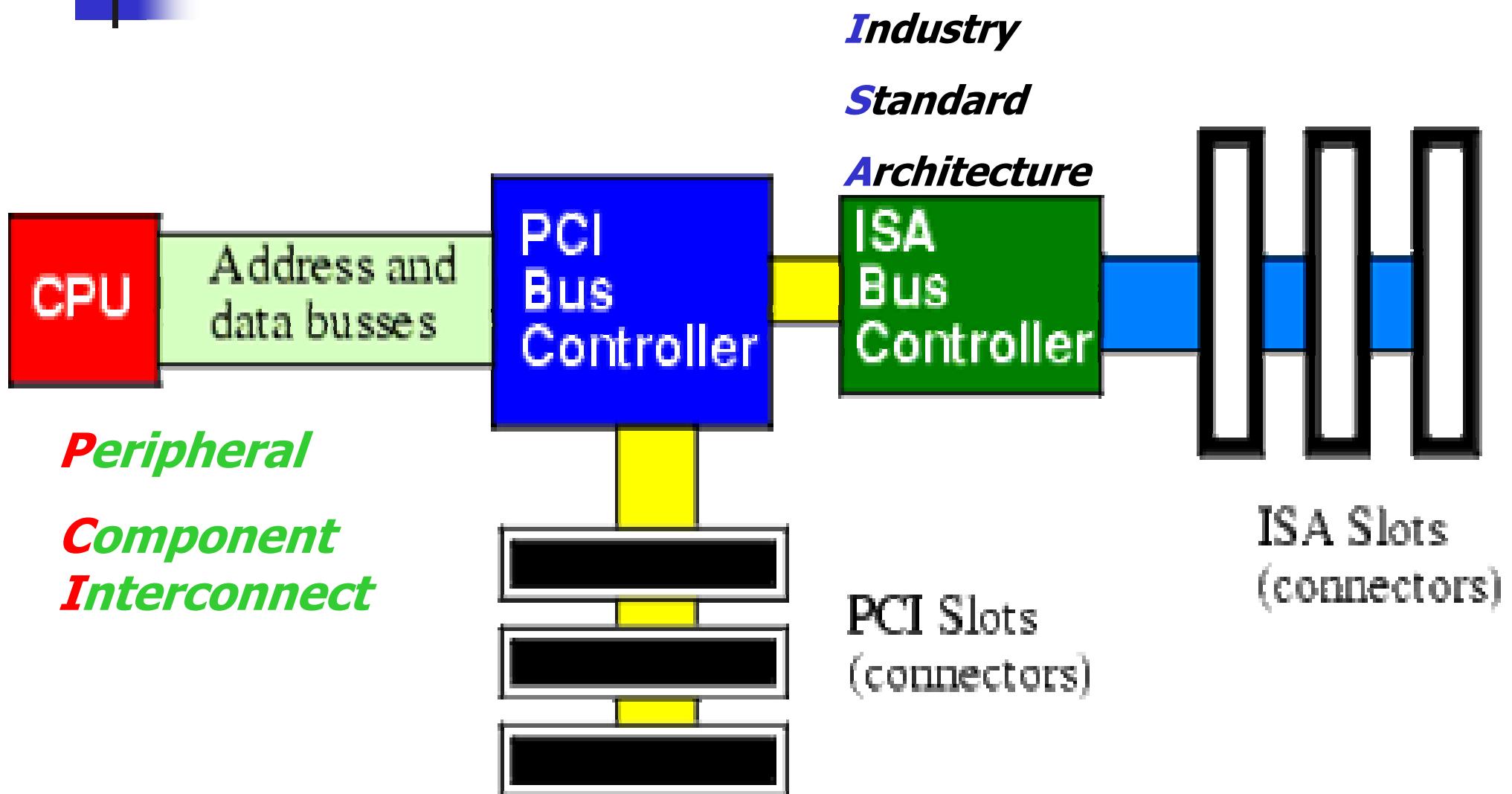
A Typical Output Port

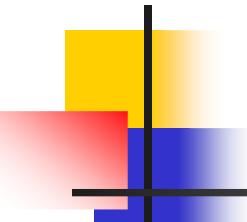


An Input and an Output Device That Share the Same Address (a Dual I/O Port)

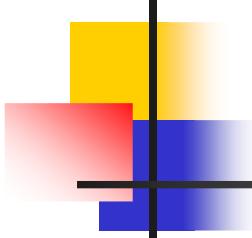


Connection of the PCI and ISA Busses in a Typical PC





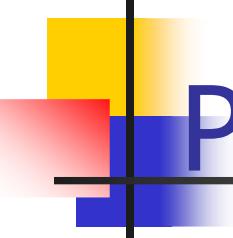
PCI local bus n. Short for Peripheral Component Interconnect local bus. A specification introduced by Intel Corporation that defines a local bus system that allows up to 10 PCI-compliant expansion cards to be installed in the computer. A PCI local bus system requires the presence of a PCI controller card, which must be installed in one of the PCI-compliant slots. Optionally, an expansion bus controller for the system's ISA, EISA, or Micro Channel Architecture slots can be installed as well, providing increased synchronization over all the system's bus-installed resources. The PCI controller can exchange data with the system's CPU either 32 bits or 64 bits at a time, depending on the implementation, and it allows intelligent, PCI-compliant adapters to perform tasks concurrently with the CPU using a technique called bus mastering. The PCI specification allows for multiplexing, a technique that permits more than one electrical signal to be present on the bus at one time.



Bus PCI

PCI chuẩn nối ghép các thiết bị ngoại vi với bộ VXL tốc độ cao của Intel như 486/Pentium

- *Tốc độ tối đa 33MHz*
- *Data bus 32 bits và 64 bits*
- *Hỗ trợ cho 10 thiết bị ngoại vi*
- *Plug and Play*



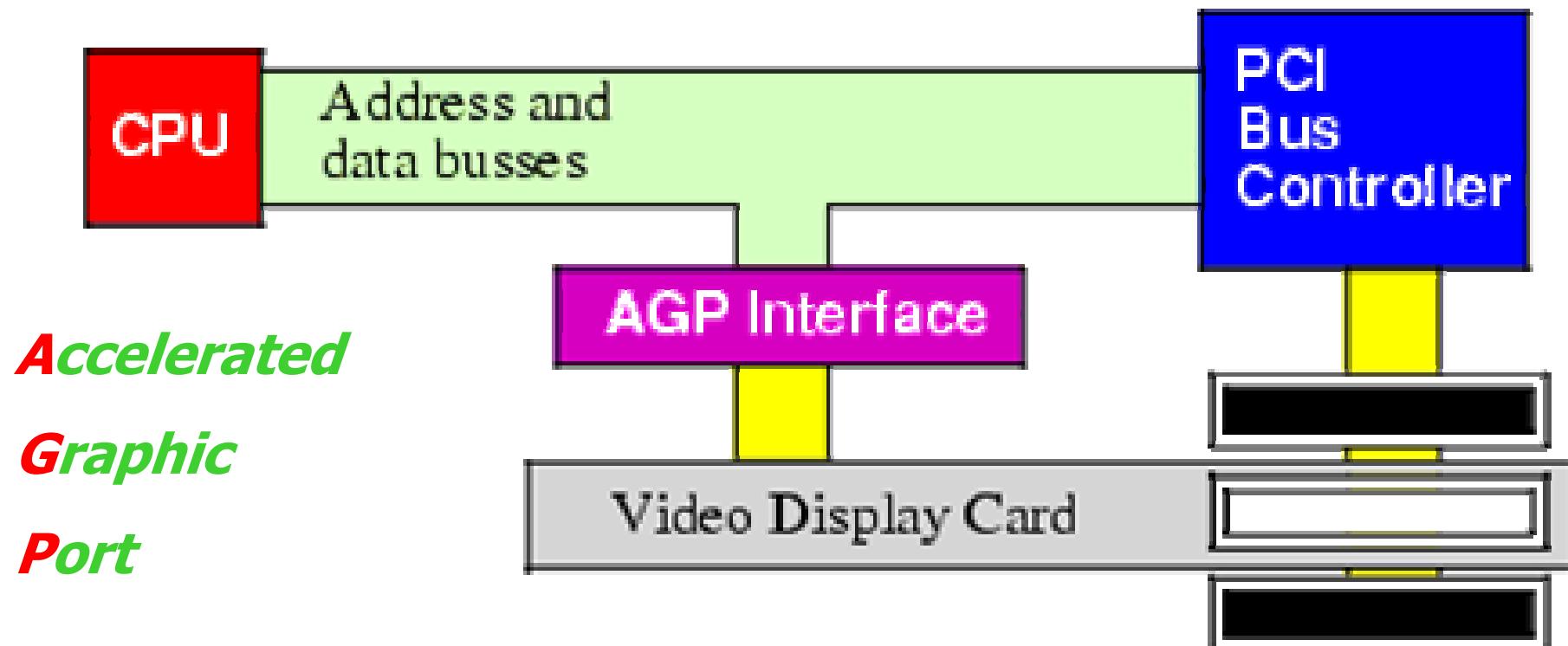
Plug and Play

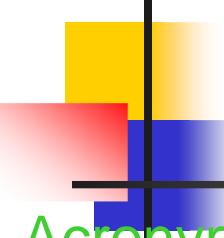
1. Cả BIOS trên mainboard và Card bổ sung đều không phải là Plug and Play.

2. BIOS trên mainboard Plug and Play nhưng Card bổ sung thì không → phần mềm cài đặt sẽ giúp sắp xếp địa chỉ I/O, IRQ và các kênh DMA.

3. BIOS trên mainboard và Card bổ sung là Plug and Play → cấu hình tự động thực hiện mọi công việc.

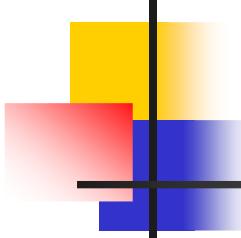
AGP Bus Interface





AGP (Accelerated Graphics Port)

Acronym for Accelerated Graphics Port. A high-performance bus specification designed for fast, high-quality display of 3-D and video images. Developed by Intel Corporation, AGP uses a dedicated point-to-point connection between the graphics controller and main system memory. This connection enables AGP-capable display adapters and compatible chip sets to transfer video data directly between system memory and adapter memory, to display images more quickly and smoothly than they can be displayed when the information must be transferred over the system's primary (PCI) bus. AGP also allows for storing complex image elements such as texture maps in system memory and thus reduces the need for large amounts of memory on the adapter itself. AGP runs at 66 MHz—twice as fast as the PCI bus—and can support data transfer speeds of up to 533 Mbps..

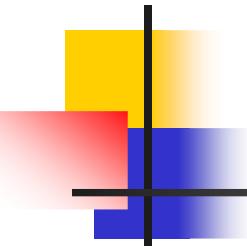


Độ rộng Bus

Độ rộng bus chính là số đường dây dẫn hợp thành bus.

Với address bus : trên mỗi đường dây chỉ có thể có 1 trong 2 trạng thái 0 hoặc 1 nên bus có độ rộng n thì có thể nhận biết được 2^n địa chỉ.

Với data bus : được thiết kế theo nguyên tắc là bội của 8 (8,16,32,64 bit) như thế mỗi lần truyền 1 byte/2 bytes/4 bytes tùy theo máy. Bề rộng Data bus càng lớn thì data truyền càng nhanh.



Bus PC/XT có khe cắm 62 chân bao gồm :

Data bus D0-D7

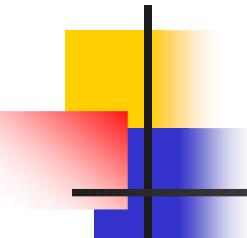
Address Bus A0-A19

Các tín hiệu điều khiển

*Bus PC/AT : bus XT + 36 chân nữa để làm việc
với data bus 16 bit, bus địa chỉ 24 bit.*

*36 chân bổ sung được dùng làm các đường dữ
liệu D8-D15, các đường địa chỉ A21-A23,...*

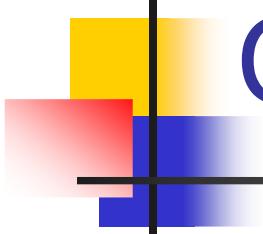
*D0-D7 : là bus dữ liệu 8 bit, 2 chiều nối giữa
bộ VXL với bộ nhớ, I/O.*



Nhược điểm của Bus ISA

Data bus bị hạn chế ở 16 bits → không thể phối hợp với data bus 32 bits của bộ VXL 386/486/Pentum.

Address bus địa chỉ 24 bits giới hạn khả năng truy cập bộ nhớ cực đại qua khe cắm mở rộng 16MB → không thể phối hợp được với bus địa chỉ 32 bit của 386/486/Pentium.



Chu kỳ Bus

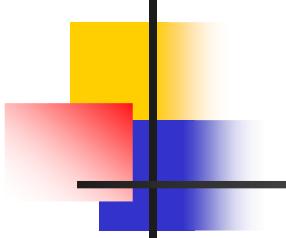
Mỗi chu kỳ bus là 1 tác vụ xảy ra trên bus để truyền tải data.

Mỗi lần CPU cần lệnh (hoặc data) từ bộ nhớ hoặc I/O, chúng phải thực thi 1 chu kỳ bus để có được thông tin hoặc ghi thông tin ra bộ nhớ hoặc ra I/O.

Mỗi chu kỳ bus gồm 2 bước :

bước 1 : gửi địa chỉ

bước 2 : truyền data từ địa chỉ đã được định vị.



4 chu kỳ bus cơ bản :

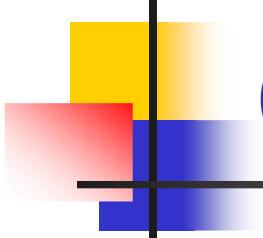
đọc bộ nhớ (memory Read)

ghi bộ nhớ (memory Write)

đọc I/O (I/O Read)

ghi I/O (I/O Write).

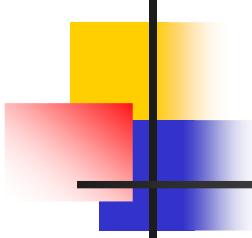
**Các tín hiệu cần thiết để thực hiện các chu kỳ bus
được sinh ra bởi CPU hoặc DMA Controller hoặc
bộ làm tươi bộ nhớ.**



Chu kỳ Bus

Mỗi chu kỳ Bus đòi hỏi tối thiểu trọn vẹn 2 xung đồng hồ hệ thống.

Đây là mốc tham chiếu theo thời gian để đồng bộ hóa tất cả các tác vụ bên trong máy tính. Xung đầu tiên gọi là Address time , **địa chỉ truy xuất sẽ được gửi đi cùng với tín hiệu xác định loại tác vụ sẽ được thực thi (đọc/ghi/đến mem/đến I/O).**

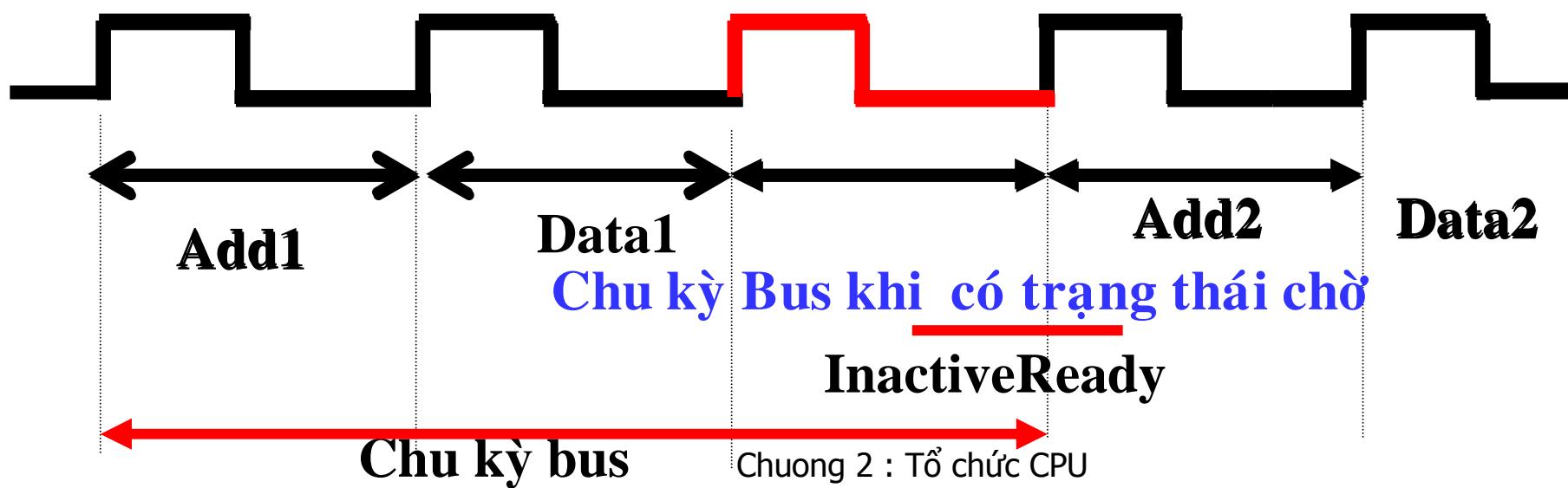
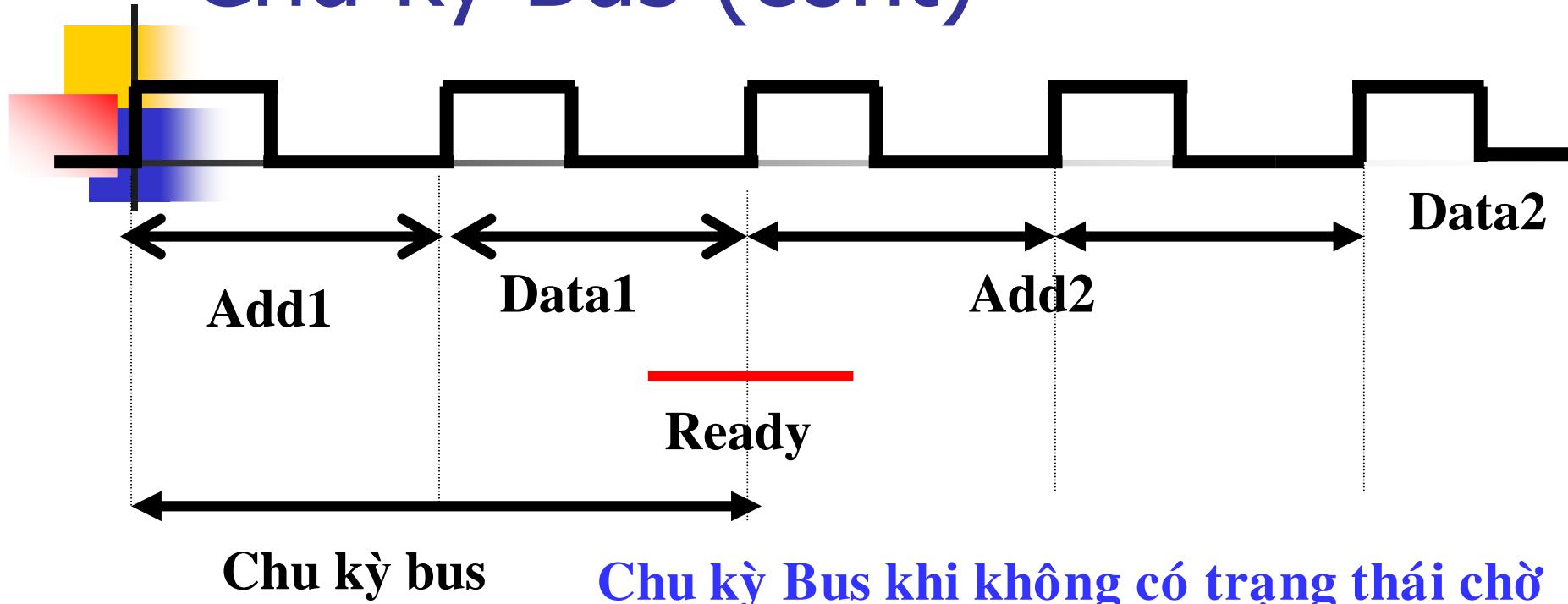


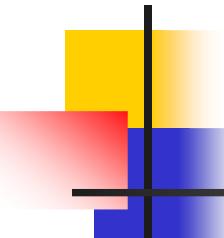
Chu kỳ Bus

Cuối xung thứ 2, CPU sẽ kiểm tra đường tín hiệu Ready. Nếu thiết bị cần truy xuất sẵn sàng đáp ứng tác vụ, thiết bị này sẽ kích 1 tín hiệu lên đường Ready để báo cho CPU biết và chu kỳ bus hoàn tất.

Khi 1 thiết bị không sẵn sàng, không có tín hiệu trên đường Ready, CPU phải chờ, có thể phải tiêu tốn thêm 1 hay nhiều xung clock.

Chu kỳ Bus (cont)





Chu kỳ Bus (cont)

Chú ý :

Trong 1 số hệ thống, cho phép ta Setup một số **wait states** trong phần Extend Setup của Bios.

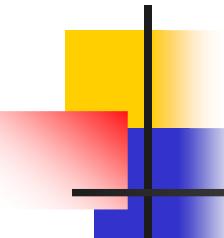
Nếu ta cho giá trị này nhỏ thì có thể ngoại vi không theo kịp CPU và hệ thống bị treo.

Còn nếu cho giá trị này lớn thì tốc độ chung của hệ thống bị chậm lại.

Wait states mặc định là 4 cho các vĩ mạch 8 bit và là 1 cho các vĩ mạch 16 bit.

tốc độ truyền tải tối đa :

tốc độ truyền tải = tốc độ bus (MHz) x số bytes trong 1 lần truyền / số chu kỳ xung clock cho mỗi lần truyền



2.4 Hệ thống thanh ghi

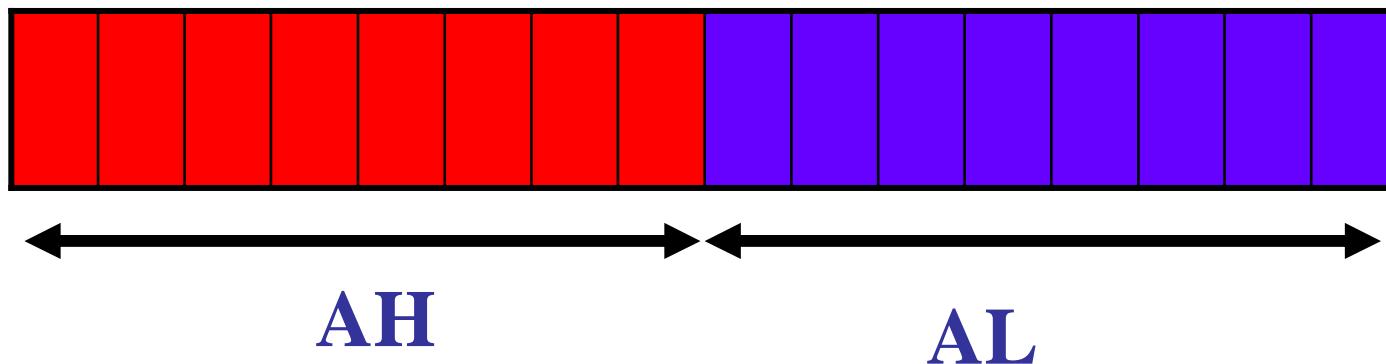
- Là các phần tử có khả năng lưu trữ thông tin với dung lượng 8, 16 , 32, 64 bit.
- Được xây dựng từ các FlipFlop nên có tốc độ truy xuất rất nhanh.

Phân loại thanh ghi :

- **Thanh ghi tổng quát** : chủ yếu dùng để lưu trữ dữ liệu trong quá trình thực thi CT, nhưng mỗi thanh ghi còn có 1 số chức năng riêng.
- **Thanh ghi điều khiển** : các bit của nó qui định tác vụ của các đơn vị chức năng của MT.
- **Thanh ghi trạng thái** : lưu trữ thông tin mô tả trạng thái.

AX Register

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

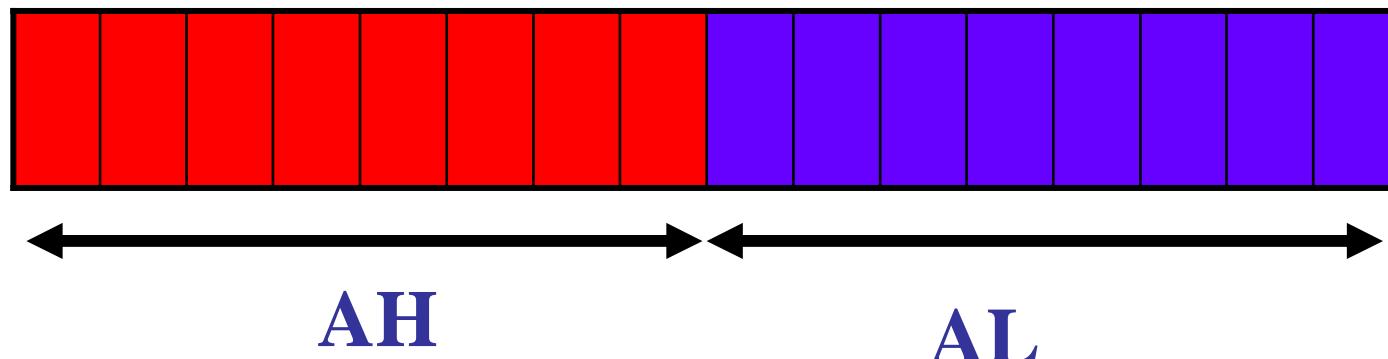


Thanh ghi AX (Accumulator register) : thanh ghi tích luỹ, dài 16 bit nhưng nó cũng có thể chia làm 2 thanh ghi 8 bit AH và AL

AX ngoài chức năng lưu trữ dữ liệu, nó còn được CPU dùng trong phép toán số học như nhân, chia.

AX Register

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

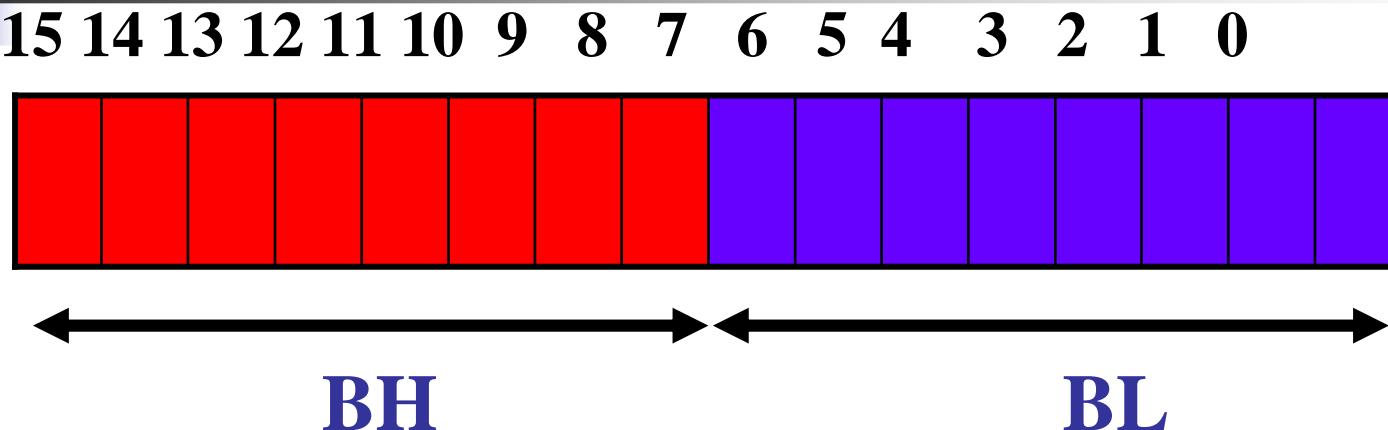


Thanh ghi AH là nửa cao của thanh ghi AX

Thanh ghi AL là nửa thấp của thanh ghi AX

Thí dụ nếu AX=1234h thì AH=12H AL=34h

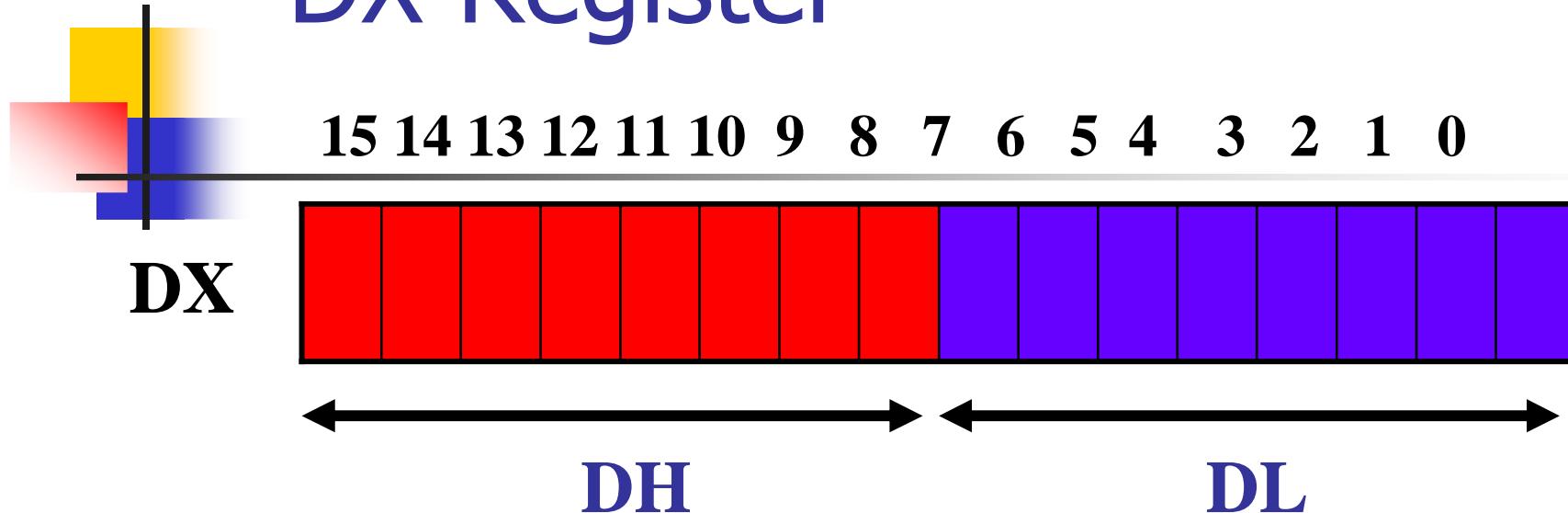
BX Register



Thanh ghi BX (Base register) : dài 16 bit nhưng nó cũng có thể chia làm 2 thanh ghi 8 bit BH và BL

BX lưu giữ địa chỉ của 1 thủ tục hay biến, nó cũng được dùng thực hiện các phép dời chuyển số học và dữ liệu.

DX Register

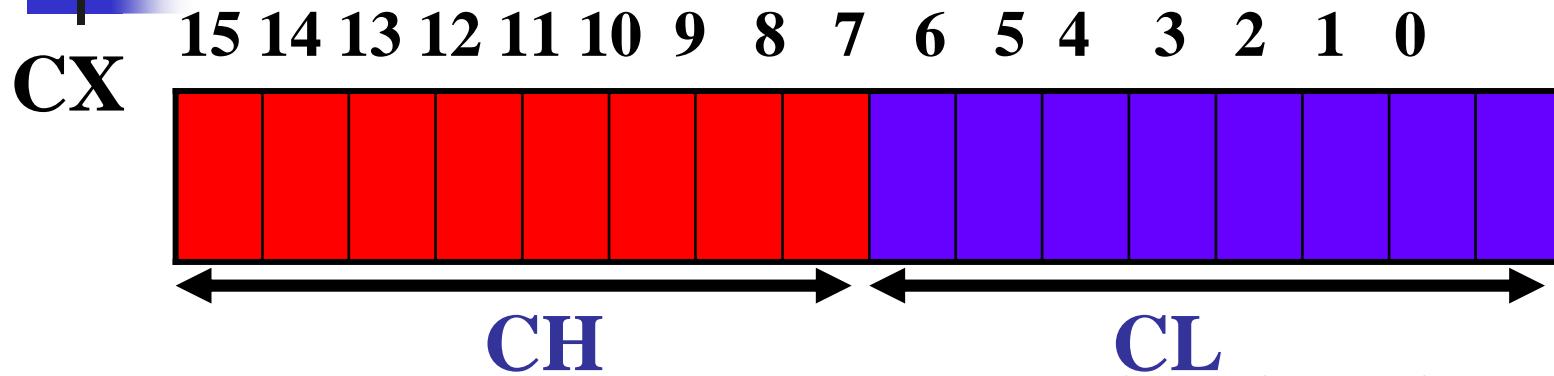


Thanh ghi DX (Data register) : dài 16 bit nhưng nó cũng có thể chia làm 2 thanh ghi 8 bit DH và DL

Thanh ghi DX : có vai trò đặc biệt trong phép nhân và phép chia ngoài chức năng lưu trữ dữ liệu.

Ex : khi nhân DX sẽ lưu giữ 16 bit cao của tích.

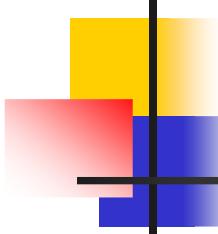
CX Register



CX (Counter register) : thanh ghi này dùng làm bộ đếm trong các vòng lặp. Các lệnh tự động lặp lại và sau mỗi lần lặp giá trị của CX tự động giảm đi 1.

CL thường chứa số lần dịch, quay trong các lệnh dịch, quay thanh ghi

CX dài 16 bit, nó cũng có thể chia làm 2 thanh ghi 8 bit là CH và CL



Các thanh ghi Segment

CPU có 4 thanh ghi segment dài 16 bit, các thanh ghi này không thể chia làm 2 thanh ghi 8 bit như 4 thanh ghi AX,BX,CX và DX.

Các thanh ghi đoạn được sử dụng như là địa chỉ cơ sở của các lệnh trong chương trình, stack và dữ liệu.

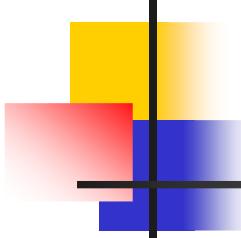
4 thanh ghi đoạn : CS (Code Segment), DS (Data Segment), SS (Stack Segment) và ES (Extra Segment).

CS : chứa địa chỉ bắt đầu của code trong chương trình.

DS : chứa địa chỉ của các biến khai báo trong chương trình.

SS : chứa địa chỉ của bộ nhớ Stack dùng trong chương trình

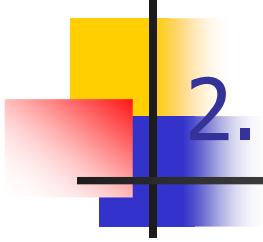
ES : chứa địa chỉ cơ sở bổ sung cho các biến bộ nhớ.



Thanh ghi 32 bit

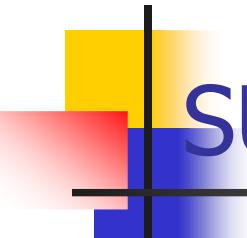
- Đối với một số CPU đời mới, có các thanh ghi dài 32, 64 bit. Ta ghi thêm E đứng trước tên các thanh ghi 16 bit...

EAX, EBX, ECX, EDX



2.5 Thanh ghi đoạn và sự hình thành địa chỉ

- 8088 sử dụng 20 bit để đánh địa chỉ bộ nhớ → quản lý trên 1Mb bộ nhớ. Nhưng 8088 lại không có thanh ghi nào 20 bit, tất cả là 16 bit do đó 1 thanh ghi chỉ có thể đánh địa chỉ tối đa là 64 kB bộ nhớ.
- Như vậy phải kết hợp 2 thanh ghi mới địa chỉ hoá toàn bộ bộ nhớ. 8088 sử 1 trong các thanh ghi dùng chung và 1 trong các thanh ghi đoạn (CS,DS,SS,ES) để tạo thành 1 địa chỉ 20 bit.



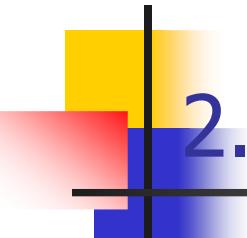
SỰ PHÂN ĐOẠN BỘ NHỚ

CPU 8086 dùng phương pháp phân đoạn bộ nhớ để quản lý bộ nhớ 1MB của nó.

Địa chỉ 20 bit của bộ nhớ 1MB không thể chứa đủ trong các thanh ghi 16 bit của CPU 8086 → bộ nhớ 1MB được chia ra thành các đoạn (segment) 64KB.

Địa chỉ trong các đoạn 64KB chỉ có 16 bit nên CPU 8086 dễ dàng xử lý bằng các thanh ghi của nó.

→ PHÂN ĐOẠN BỘ NHỚ : là cách dùng các thanh ghi 16 bit để biểu diễn cho địa chỉ 20 bit.



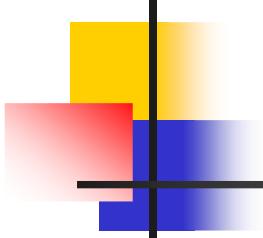
2.5 Địa chỉ vật lý & địa chỉ luận lý

Địa chỉ 20 bits được gọi là địa chỉ vật lý.

Địa chỉ vật lý dùng như thế nào ?

Dùng trong thiết kế các mạch giải mã địa chỉ cho bộ nhớ và xuất nhập.

Còn trong lập trình , địa chỉ vật lý không thể dùng được mà nó được thay thế bằng địa chỉ luận lý (logic).



Địa chỉ luận lý

Địa chỉ của 1 ô nhớ được xác định bởi 2 phần:

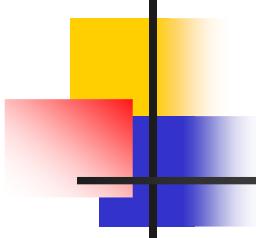
Segment : offset —→ *Địa chỉ trong
đoạn (độ dời)*

↑
Địa chỉ đoạn

Ex : B001:1234

*Mỗi địa chỉ thành phần là 1 số 16 bit và được viết
theo cách sau :*

Segment : offset



Sự hình thành địa chỉ

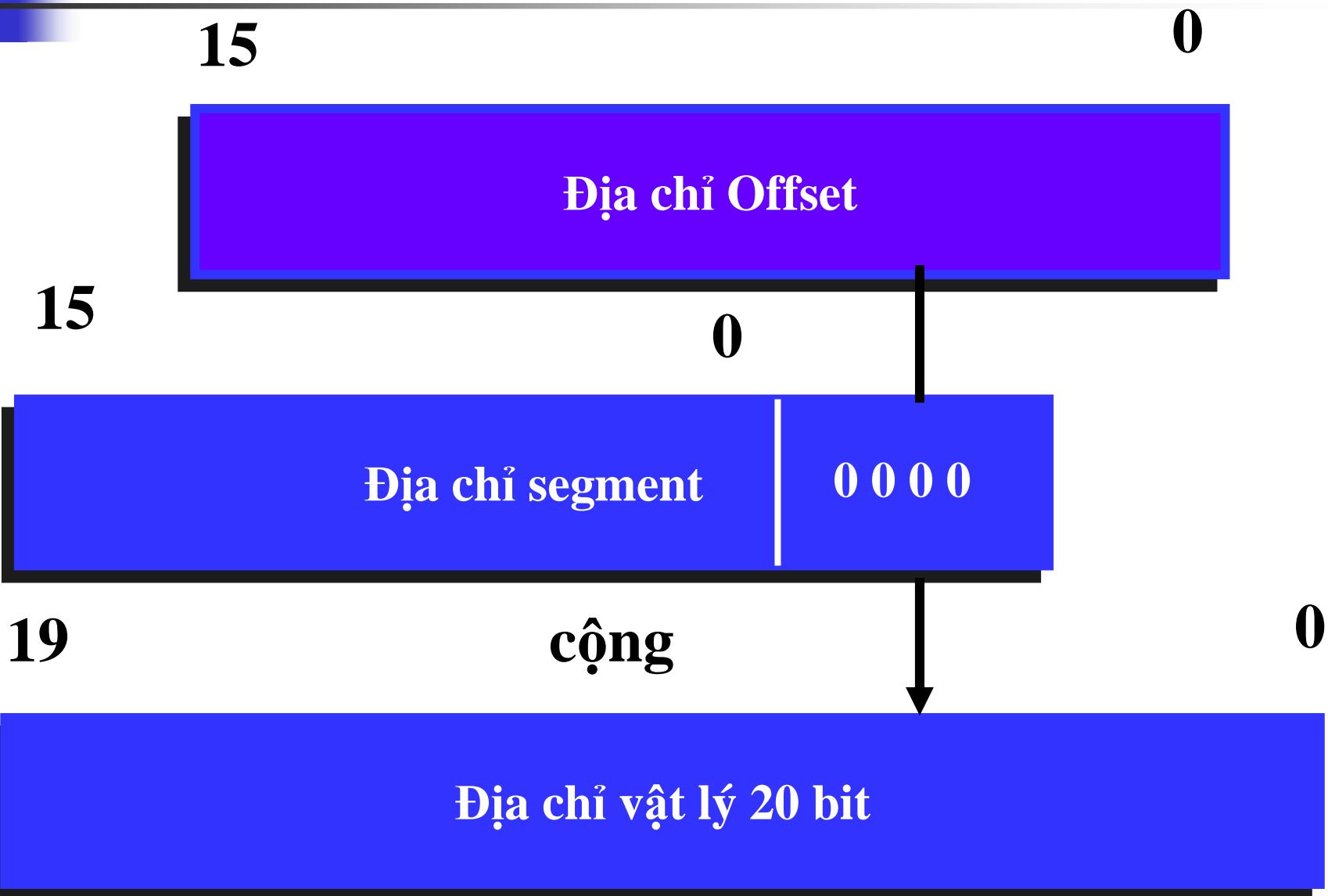
Hãng Intel đề xuất 1 phương pháp để hình thành địa chỉ.

Mỗi địa chỉ ô nhớ được hình thành từ 1 phép tính tổng 1 địa chỉ cơ sở và 1 địa chỉ offset.

Địa chỉ cơ sở lưu trong 1 thanh ghi segment, còn địa chỉ offset nằm trong 1 thanh ghi chỉ số hay thanh ghi con trỏ.

Phép cộng này sẽ tạo 1 địa chỉ 20 bit gọi là địa chỉ vật lý.

Thí dụ minh họa hình thành địa chỉ



Sự hình thành địa chỉ tuyệt đối

địa chỉ
segment

địa chỉ Offset

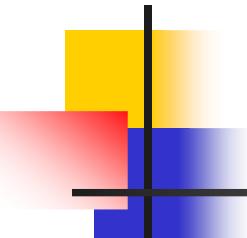
Gia sử ta có địa chỉ **08F1 : 0100**

địa chỉ tương đối

CPU tự động lấy địa chỉ segment x 10 (hệ 16) thành **08F10**

Sau đó nó cộng với địa chỉ Offset **0100**

→ địa chỉ tuyệt đối : **09010**



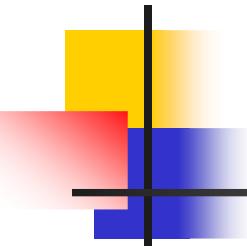
Cách tính địa chỉ vật lý từ địa chỉ luận lý

*Địa chỉ vật lý = (segment*16) + offset*

<i>Segment</i>	0
+	<i>offset</i>
<i>Địa chỉ vật lý</i>	

Ex : tính địa chỉ vật lý tương ứng địa chỉ luận lý B001:1234

Địa chỉ vật lý = B0010h + 1234h = B1244h



Sự chồng chất các đoạn

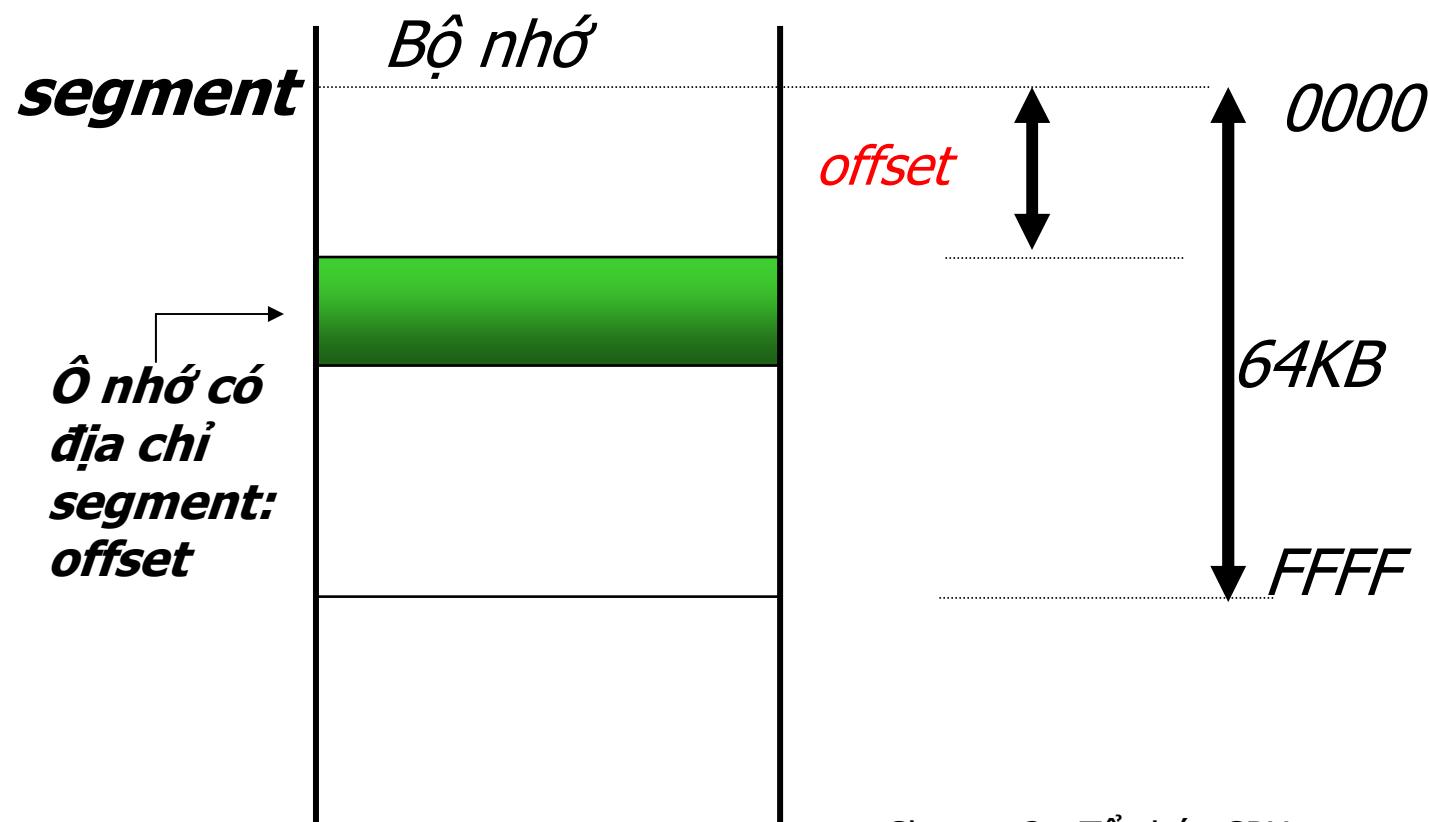
Địa chỉ segment hay còn gọi là địa chỉ nền của đoạn. Nó cho biết điểm bắt đầu của đoạn trong bộ nhớ.

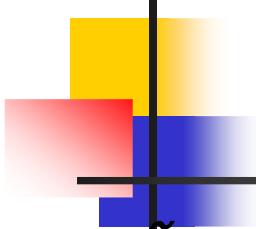
Địa chỉ offset thể hiện khoảng cách kể từ đầu đoạn của ô nhớ cần tham khảo.

Do offset dài 16 bit nên chiều dài tối đa của mỗi đoạn là 64K.

Sự chồng chất các đoạn

Trong mỗi đoạn, ô nhớ đầu tiên có offset là $0000h$ và ô nhớ cuối cùng là $FFFFh$.





*Mỗi ô nhớ chỉ có địa chỉ vật lý nhưng có thể có
nhiều địa chỉ luận lý.*

Ex : 1234:1234

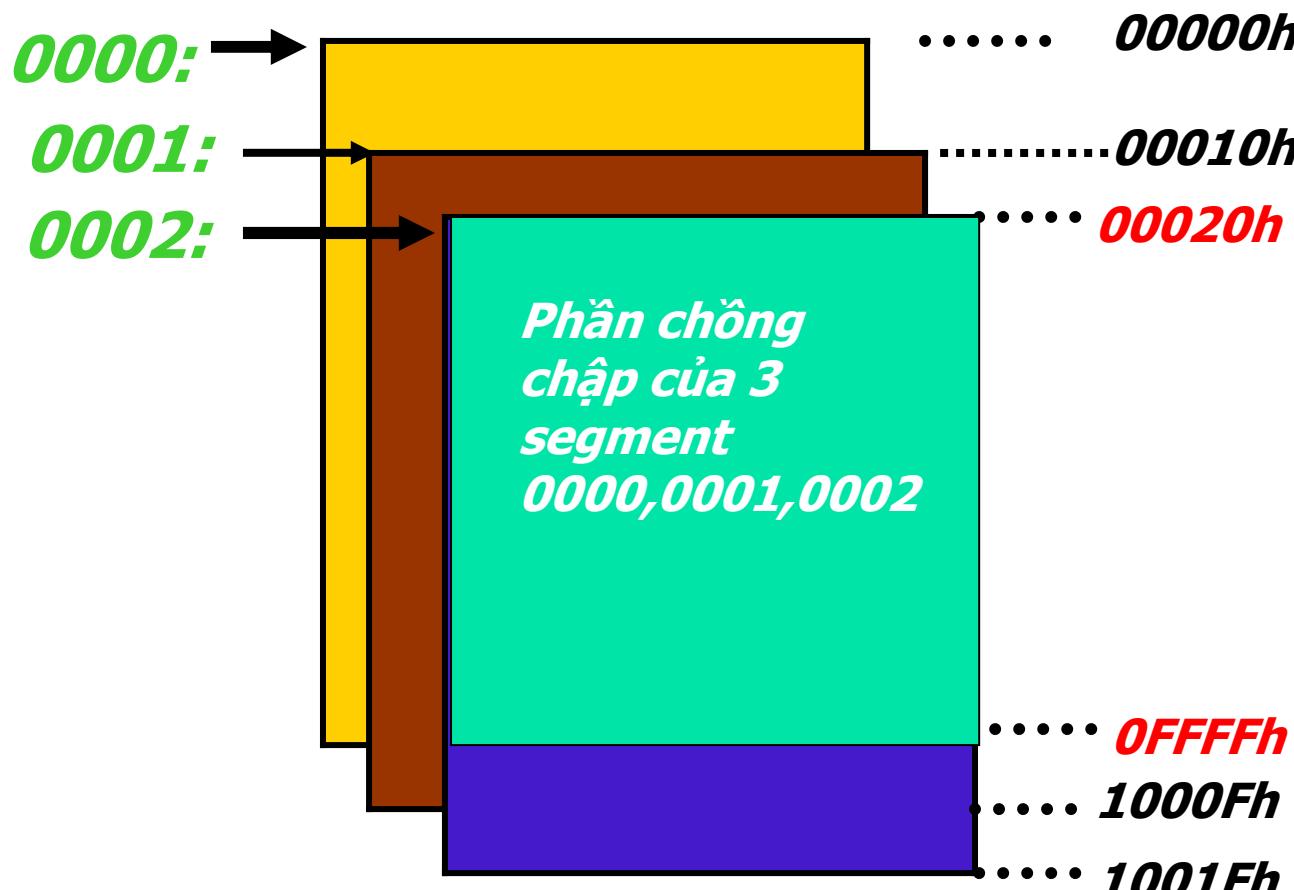
1334:0234

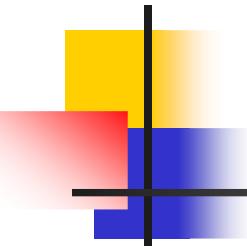
1304:0534

*Đều có chung địa chỉ
vật lý 13574h*

Tại sao ?

Để hiểu rõ tại sao ta hãy xét mối quan hệ giữa địa chỉ vật lý với segment và offset





Giải thích

$0000:0000 \rightarrow 00000h$

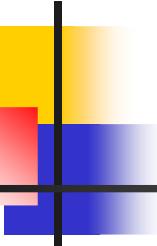
*Giữ nguyên phần segment, tăng phần offset
lên 1 thành ra địa chỉ luận lý là 0000:0001*

Địa chỉ vật lý tương ứng là 00001h

*Tương tự với địa chỉ luận lý là 0000:0002 ta có
địa chỉ vật lý là 00002h*

*Khi offset tăng 1 đơn vị thì địa chỉ vật lý
tăng 1 địa chỉ hoặc là tăng 1 byte.*

Như vậy có thể xem đơn vị của offset là byte



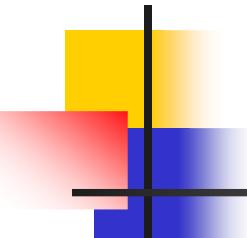
Làm lại quá trình trên nhưng giữ nguyên phần offset chỉ tăng phần segment.

0001:0000 → 00010h

0002:0000 → 00020h

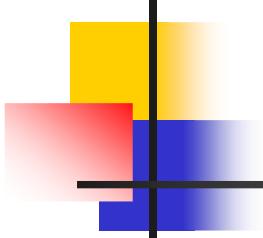
*Khi segment tăng 1 đơn vị thì địa chỉ vật lý
tăng 10h địa chỉ hoặc là tăng 16 bytes*

Đơn vị của segment là paragraph



**Ta thấy segment 0000 nằm ở đâu
vùng nhớ nhưng segment 0001 bắt
đâu cách đâu vùng nhớ chỉ có 16
bytes, segment 0002 bắt đâu cách
đâu vùng nhớ 32 bytes.....**

**Phân chia chập 3 segment
0000,0001,0002 trên hình vẽ là vùng bộ
nhớ mà bất kỳ ô nhớ nào nằm trong đó
(địa chỉ vật lý từ 00020h đến OFFFFh) đều
có thể có địa chỉ luận lý tương ứng trong
cả 3 segment.**

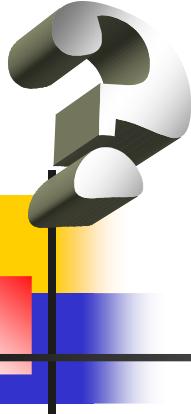


Ex : ô nhớ có địa chỉ 0002Dh sẽ có địa chỉ logic trong segment 0000 là 0000:002D

Trong segment 0001 là 0001:001D

Trong segment 0002 là 0002:000D

→ nếu vùng bộ nhớ nào càng có nhiều segment chồng chập lên nhau thì các ô nhớ trong đó càng có nhiều địa chỉ luận lý.



**Một ô nhớ có bao nhiêu
địa chỉ luận lý**

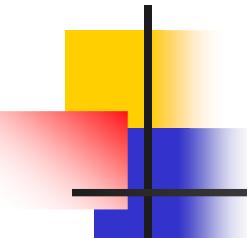
**Một ô nhớ có ít nhất 1 địa chỉ
luận lý và nhiều nhất là
 $65536/16 = 4096$ địa chỉ luận lý**

Các thanh ghi đoạn CS, DS, SS, ES

- 3 trong 4 thanh ghi đoạn được dùng trong các mục đích đặc biệt sau
- **CS** : xác định đoạn lệnh – nơi chứa chương trình được thi hành.
- **DS** : xác định đoạn dữ liệu – nơi chứa chương trình được thi hành.
- **SS** : xác định đoạn stack – vùng làm việc tạm thời dùng để theo dõi các tham số và các địa chỉ đang được chương trình hiện hành sử dụng.
- Còn thanh ghi ES : trả đến đoạn thêm, thường được dùng để bổ sung cho đoạn dữ liệu → có vùng nhớ >64k cho đoạn dữ liệu.

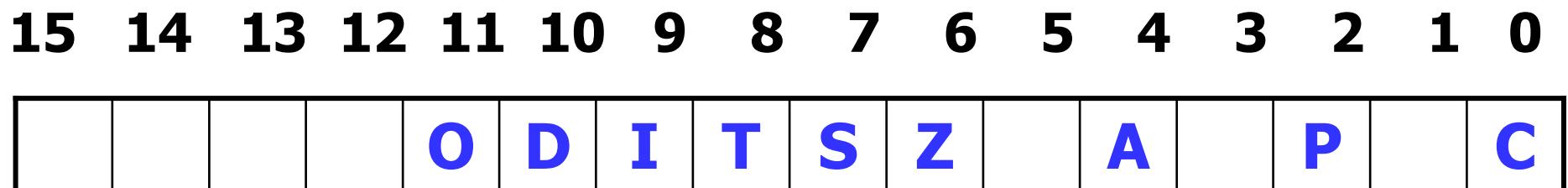
Các thanh ghi đoạn CS, DS, SS, ES

- 3 trong 4 thanh ghi đoạn được dùng trong các mục đích đặc biệt sau
- **CS** : xác định đoạn lệnh – nơi chứa chương trình được thi hành.
- **DS** : xác định đoạn dữ liệu – nơi chứa chương trình được thi hành.
- **SS** : xác định đoạn stack – vùng làm việc tạm thời dùng để theo dõi các tham số và các địa chỉ đang được chương trình hiện hành sử dụng.
- Còn thanh ghi ES : trả đến đoạn thêm, thường được dùng để bổ sung cho đoạn dữ liệu → có vùng nhớ >64k cho đoạn dữ liệu.



Thanh ghi trạng thái (thanh ghi cờ)

- **Thanh ghi cờ là thanh ghi 16 bit nằm bên trong EU (Execution Unit). Tuy nhiên chỉ có 9 trong 16 bit được sử dụng. 7 bit còn lại không dùng.**



O OverFlow flag

D : Direction flag

I : Interrupt flag

T : Trap flag

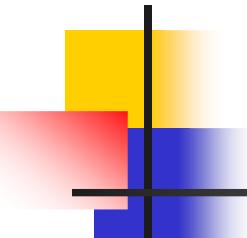
S : Sign flag

Z : Zero flag

A : Auxiliary flag

P : Parity flag

C : Carry flag



Thanh ghi trạng thái (thanh ghi cờ)

Giải thích :

Cờ CF : chỉ thị cộng có nhớ, trừ có mượn.

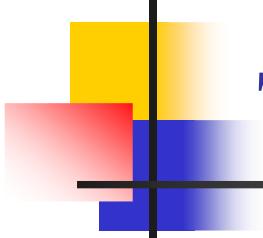
Cờ PF : On khi kết quả của tác vụ có số bit 1 là số chẵn.

Nếu số bit 1 là số lẻ thì PF là Off.

Cờ AF : có nhớ trong phép cộng hoặc có mượn trong phép trừ với 4 bit thấp sang 4 bit cao.

Cờ ZF : On khi tác vụ luận lý cho kết quả là 0.

Cờ SF : bit cao nhất của kết quả sẽ được copy sang SF. SF =1 kết quả là số âm. SF = 0 khi kết quả là số dương.



Thanh ghi trạng thái (thanh ghi cờ)

Giải thích :

Cờ OF : $OF=1$ khi kết quả bị tràn số (vượt quá khả năng lưu trữ).
Nếu kết quả không bị tràn thì $OF=0$.

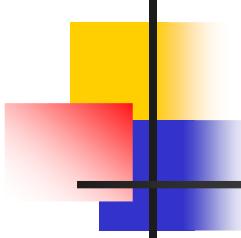
3 bit còn lại là 3 bit điều khiển :

Cờ TF : báo CPU thi hành từng bước. Cung cấp công cụ debug chương trình.

Cờ IF : $IF=1$ giúp 8086 nhận biết có yêu cầu ngắt quang có che.

Cờ DF : xác định hướng theo chiều tăng/giảm trong xử lý chuỗi.

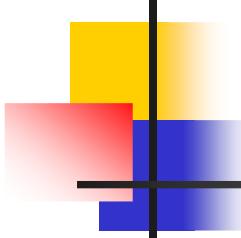
8086 cho phép User lập trình bật tắt các cờ CF,DF,IF,TF



Thanh ghi chỉ số (Index)

5 thanh ghi offset dùng để xác định chính xác 1 byte hay 1 word trong 1 đoạn 64K. Đó là :

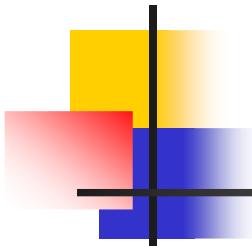
- IP : thanh ghi con trỏ lệnh, cho biết vị trí của lệnh hiện hành trong đoạn lệnh. Con trỏ lệnh IP còn được gọi là bộ đếm chương trình.
Thường được dùng kết hợp với CS để theo dõi vị trí chính xác của lệnh sẽ được thực hiện kế tiếp.



Thanh ghi chỉ số (Index)

- Các thanh ghi con trỏ Stack : SP và BP, mỗi thanh ghi dài 16 bit.
- SP (Stack pointer) cho biết vị trí hiện hành của đỉnh Stack.
- BP (Basic Pointer) dùng để truy cập dữ liệu trong Stack.
- SI (source index) : trỏ đến ô nhớ trong đoạn dữ liệu được định địa chỉ bởi thanh ghi DS.
- DI (destination) : chức năng tương tự SI.
Hai thanh ghi này thường dùng trong xử lý chuỗi.

ĐỊA CHỈ LUẬN LÝ VÀ THANH GHI

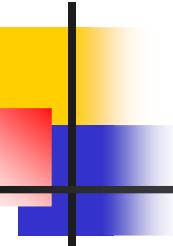


Để tham khảo đến bộ nhớ trong chương trình, VXL 8086 cho phép sử dụng các địa chỉ luận lý 1 cách trực tiếp hoặc thông qua các thanh ghi của nó.

Thanh ghi đoạn dùng để chứa segment

Thanh ghi tổng quát dùng để chứa địa chỉ trong đoạn offset

Để tham khảo đến địa chỉ luận lý có segment trong thanh ghi DS, offset trong thanh ghi BX, ta viết **DS:BX**

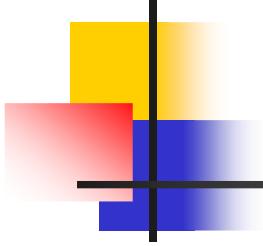


Ex : nếu lúc tham khảo

DS = 2000h BX = 12A9h thì địa chỉ luận lý

DS:BX chính là tham khảo đến ô nhớ

2000:12A9



Trong cách sử dụng địa chỉ luận lý thông qua các thanh ghi có 1 số cặp thanh ghi luôn phải dùng chung với nhau 1 cách bắt buộc :

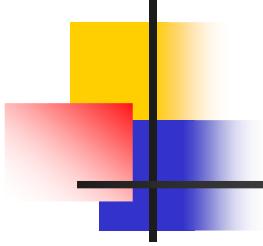
CS:IP lấy lệnh (địa chỉ lệnh sắp thi hành)

SS:SP địa chỉ đỉnh Stack

***SS:BP thông số trong Stack
(dùng trong chương trình con)***

DS:SI địa chỉ chuỗi nguồn

ES:DI địa chỉ chuỗi đích

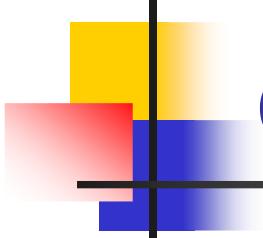


*Chương trình mà VXL 8086 thi hành thường
có 3 đoạn :*

Đoạn chương trình có địa chỉ trong thanh ghi CS.

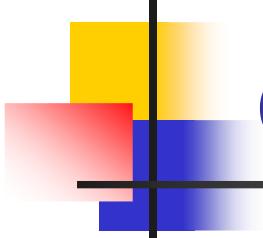
Đoạn dữ liệu có địa chỉ trong thanh ghi DS.

Đoạn stack có địa chỉ trong thanh ghi SS.



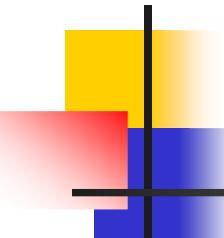
Các đặc tính của CPU Intel

- Hiệu quả của CPU thuộc họ Intel khi xử lý và chuyển giao thông tin được xác định bởi các yếu tố sau :
- Tần số mạch xung đồng hồ của CPU.
- Độ rộng của Data bus
- Độ rộng của Address bus



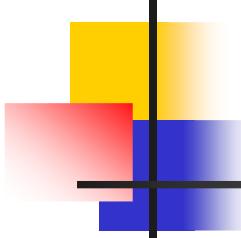
Các đặc tính của CPU Intel

- Tần số mạch xung đồng hồ của CPU càng nhanh thì tốc độ xử lý càng nhanh.
- Độ rộng của Data bus càng rộng thì càng nhiều data được chuyển giao trong 1 lần giao dịch.
- Độ rộng của Address bus càng rộng thì khả năng quản lý bộ nhớ càng lớn.



Các đặc tính của CPU Intel

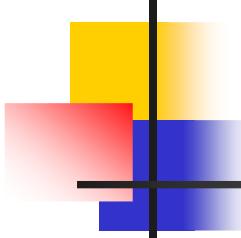
Loại CPU	Data Bus (bit)	Address bus (bit)	Khả năng quản lý bộ nhớ
8088	8	20	1 MB
8086	16	20	1MB
80286	16	24	16Mb
80386	32	32	4 GB
80486	32	32	4 GB
Pentium	64	32	4GB



Tóm tắt CPU họ Intel

- **CPU 80286 : Data bus 16 bit nên mỗi lần chuyển giao 2 bytes → quản lý 16MB bộ nhớ.**
Chỉ có khả năng thực hiện các phép toán đối với các số nguyên, có thể dùng tập lệnh 80286 để mô phỏng các phép toán số học dấu chấm động nhưng điều này sẽ làm giảm hiệu suất hệ thống.
- Nếu muốn có khả năng thực hiện các phép toán dấu chấm động phải gắn CoProcessor 8087.

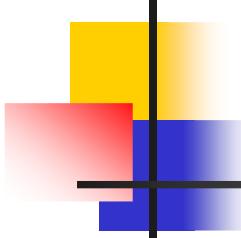
80286 làm việc theo 2 chế độ : chế độ thực và chế độ bảo vệ.



Tóm tắt CPU họ Intel

- **CPU 80386 : Data bus 32 bit nên có thể quản lý 4GB bộ nhớ.**
Các thanh ghi dài 32 bit → tăng độ chính xác của các phép toán.
Độ rộng Bus → tăng tốc độ thực thi.

CPU 80386 hoàn toàn tương thích với các CPU trước nó.



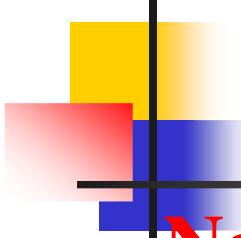
Tóm tắt CPU họ Intel

- **CPU 80486 : có bus 32 bit . 1 Coprocessor 387, bộ phận điều khiển Cache, 1 Cache 8K, dùng phối hợp tập lệnh rút gọn RISC và tập lệnh phức tạp CISC.**

CPU 80486 phần lớn các lệnh chỉ dùng 1 số ít xung.

Sử dụng cơ chế đường ống có khả năng xử lý 5 lệnh đồng thời :

- **Lấy lệnh trước Prefetch**
- **Giải mã lần 1 Decode 1**
- **Giải mã lần 2 Decode 2**
- **Thực thi lệnh Execution**
- **Ghi lại trạng thái. WriteBack**



RISC & CISC

■ Nguyên lý CISC :

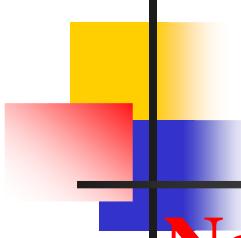
Complex Instruction Set Computer

- Tập lệnh khá lớn >300 lệnh
- Khả năng định vị phức tạp
- Một số lệnh cần phải vi lệnh hoá

quá nhiều lệnh → nạp lâu → làm chậm hệ thống

lệnh phức tạp → nên time giải mã lệnh nhiều khi lớn hơn time thực thi.

Chỉ có hơn 20% lệnh thường dùng tới



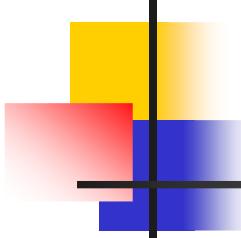
RISC & CISC

■ **Nguyên lý RISC** : tập lệnh thu gọn

Reduce **I**nstruction **S**et **C**omputer

tập lệnh nhỏ → thi hành ngay không cần giải mã.

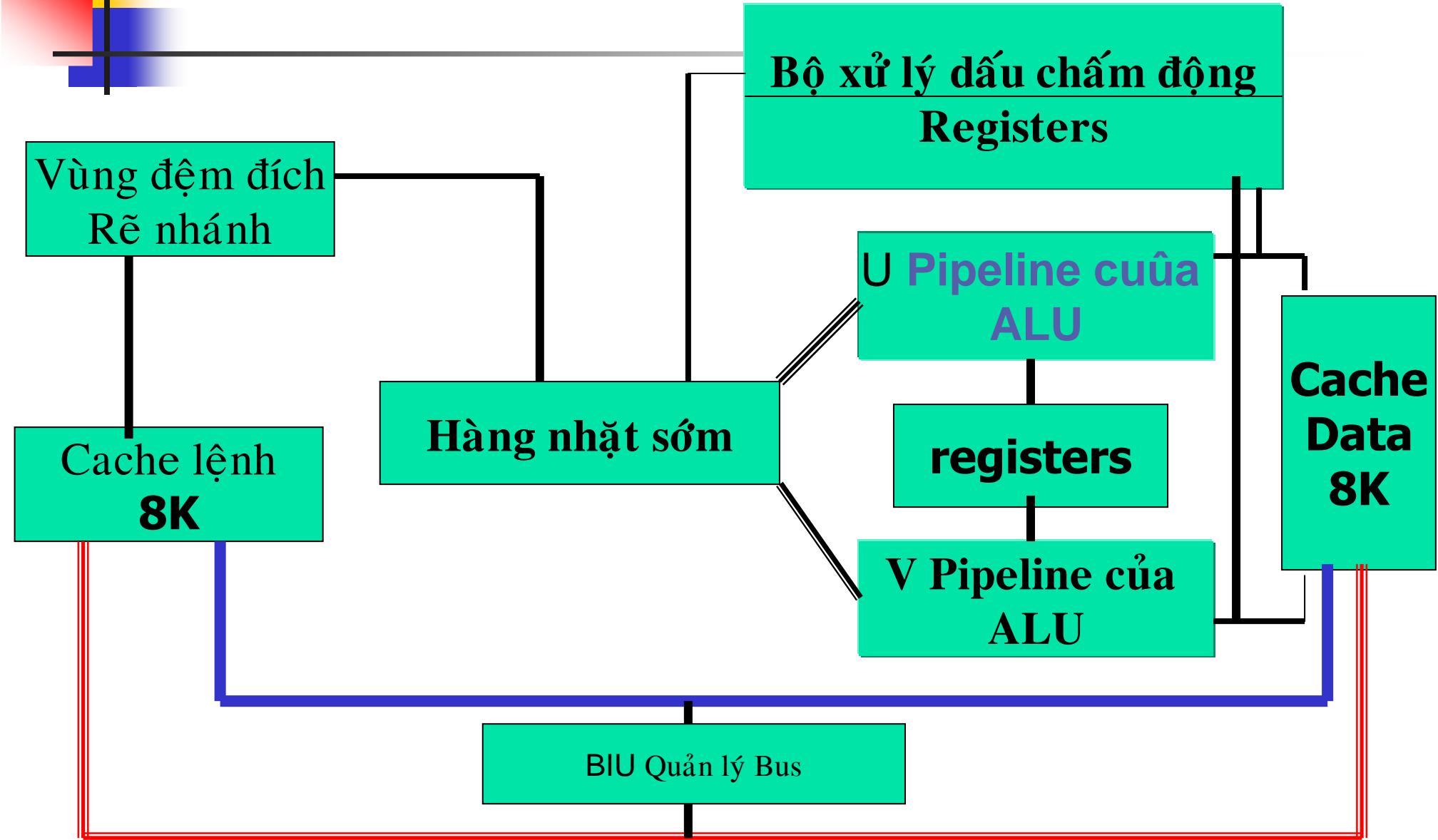
lệnh làm việc theo cơ chế đường ống (pipeline).

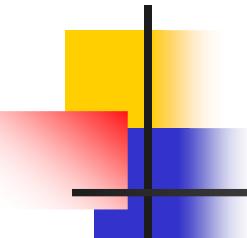


CPU Pentium

- 3 thành phần góp sức tăng tốc độ xử lý của Pentium :
 - **Đơn vị tính toán số nguyên supercalar**
 - **Bộ nhớ Cache cấp 1 ở bên trong CPU.**
 - **Đơn vị tính toán số chấm động supercalar**

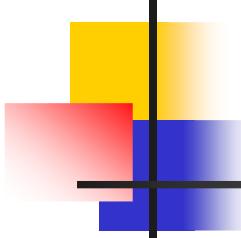
SƠ ĐỒ KHỐI PENTIUM





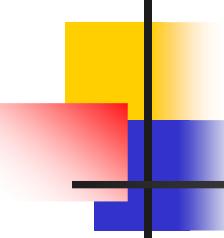
Câu hỏi ôn tập

- Bus là gì? Trong các loại Bus, Bus nào là Bus 2 chiều.
- Cho 1 ô nhớ có địa chỉ vật lý là 1256H, cho biết địa chỉ dạng segment:offset với các đoạn 1256H và 1240H.
- Ô nhớ có địa chỉ vật lý 80FD2H, ở trong đoạn nào thì nó có offset = BFD2H?
- Xác định địa chỉ vật lý của ô nhớ có địa chỉ logic 0A51H:CD90H



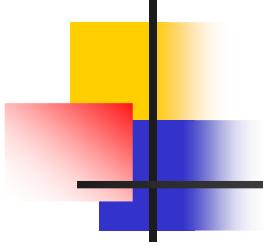
Câu hỏi ôn tập

- Thế nào là biên giới đoạn?
- Sự khác nhau cơ bản giữa bộ vi xử lý 8086 và 80286?
- Thuyết minh trình tự CPU thực hiện câu lệnh $\text{Mem}(b) \leftarrow \text{Not Mem}(a)$
- Chu kỳ lệnh, chu kỳ máy. Cho biết quan hệ giữa chu kỳ clock, chu kỳ máy và chu kỳ lệnh.
- Quan hệ giữa tập lệnh và kiến trúc của CPU



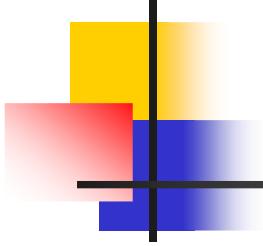
Câu hỏi ôn tập

- Giải thích tại sao khi tăng tần số xung clock, giảm chu kỳ wait state của bộ nhớ, thêm cache cho CPU lại làm cho hệ thống chạy với hiệu suất cao hơn. ?
- Trình bày chiến lược chính lưu trữ thông tin trong Cache?
- Tính tốc độ chuyển giao dữ liệu của máy tính có CPU 486DX-66MHz và máy Pentium 100MHz.
- Phân biệt RISC và CISC.
- Trình bày cơ chế đường ống trong thực thi của CPU



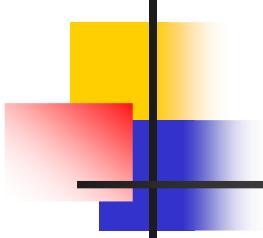
Bus ISA-8 bits :

- a. *chạy ở tốc độ đồng hồ là 8 MHz truyền tải dữ liệu tối đa 8 MB/s.*
- b. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 6MB/s.*
- c. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 1MB/s.*
- d. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 12MB/s.*



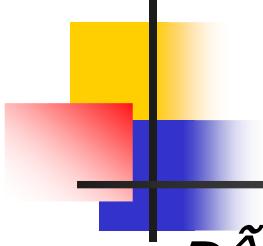
Bus ISA-16 bits :

- a. *chạy ở tốc độ đồng hồ là 8→12 MHz truyền tải dữ liệu tối đa 8 MB/s.*
- b. *chạy ở tốc độ đồng hồ là 32 MHz truyền tải dữ liệu tối đa 12MB/s.*
- c. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 12MB/s.*
- d. *chạy ở tốc độ đồng hồ là 16MHz truyền tải dữ liệu tối đa 12MB/s.*



Bus PCI :

- a. truyền tải dữ liệu tối đa 528 MB/s.
- b. truyền tải dữ liệu tối đa 128MB/s.
- c. truyền tải dữ liệu tối đa 512MB/s.
- d. truyền tải dữ liệu tối đa 64MB/s.



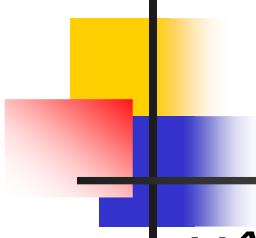
Dẫn đầu về Chipset hiện có trên thị trường là :

a.AMD

b.ALI

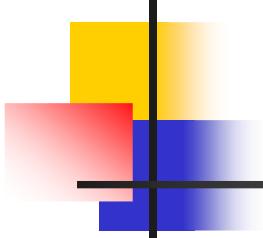
c.Intel

d.Mac



Hệ thống Bus là hệ thống xa lộ thông tin bên trong PC giúp trao đổi:

- a. thông tin giữa các máy tính*
- b. dữ liệu giữa các thiết bị ngoại vi*
- c. dữ liệu giữa bộ VXL và các thiết bị khác*
- d. tất cả đều đúng*



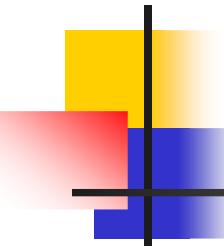
Mọi hoạt động của máy tính từ CPU đến bộ nhớ RAM và những thiết bị I/O đều phải thông qua sự nối kết được gọi chung là :

a. Chuẩn giao tiếp

b. Bus

c. BIOS

d. CMOS



BÀI TẬP

Bài 1 : Cho biết giá trị chuỗi 'XY' được lưu trữ trong MT dưới dạng số hex và dạng số bin?

Bài 2 : Cho biết giá trị ở hệ 10 của các số nguyên có dấu sau :

a. 10000000_b b. 01111111_b

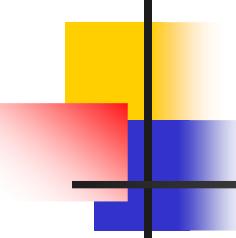
Bài 3 : Cho đoạn code sau :

MOV AH,7F INT 20H

MOV AX,1234 Hãy cho biết giá trị của

MOV BH,AL các thanh ghi AX,BX ?

MOV BL,AH



BÀI TẬP

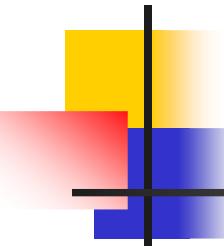
Bài 4: Cho đoạn code sau :

MOV AL, 81

ADD AL, 0FE

INT 20H

*Giả sử các số đều là số có dấu. Giải thích kết quả
chứa trong thanh ghi AL khi đoạn code trên được
thực thi. Sử dụng giá trị ở hệ 10 để giải thích.*

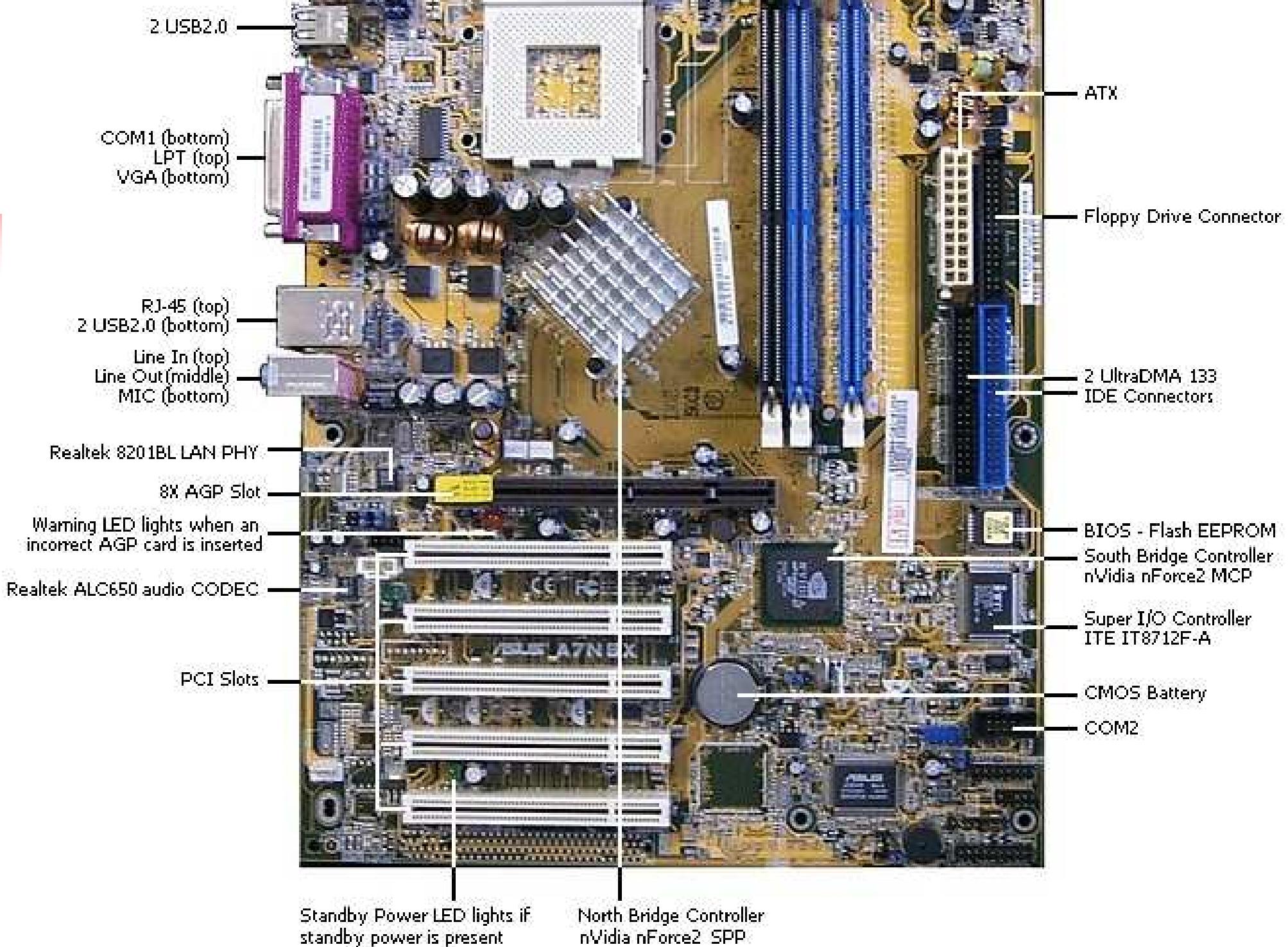


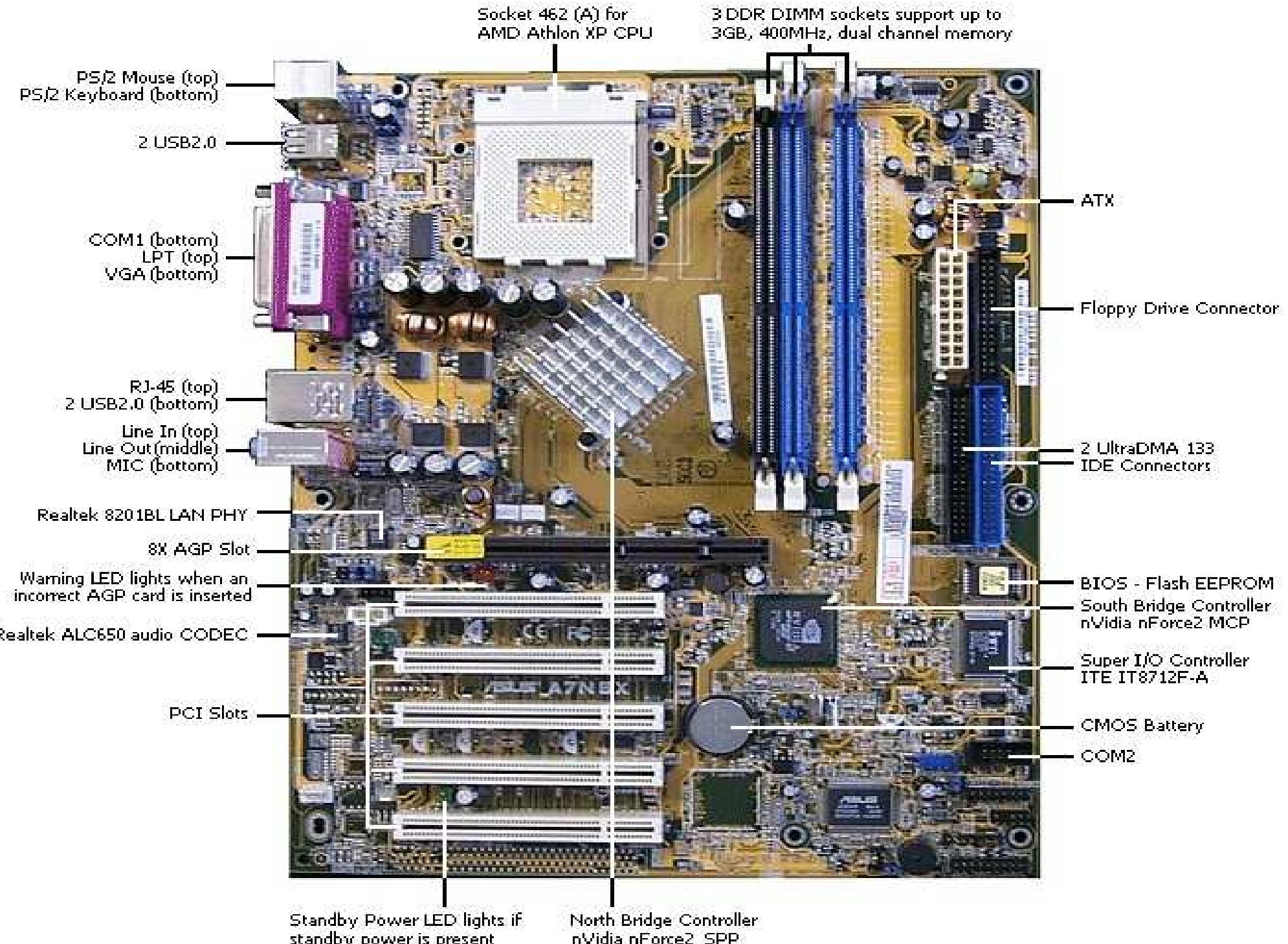
BÀI TẬP

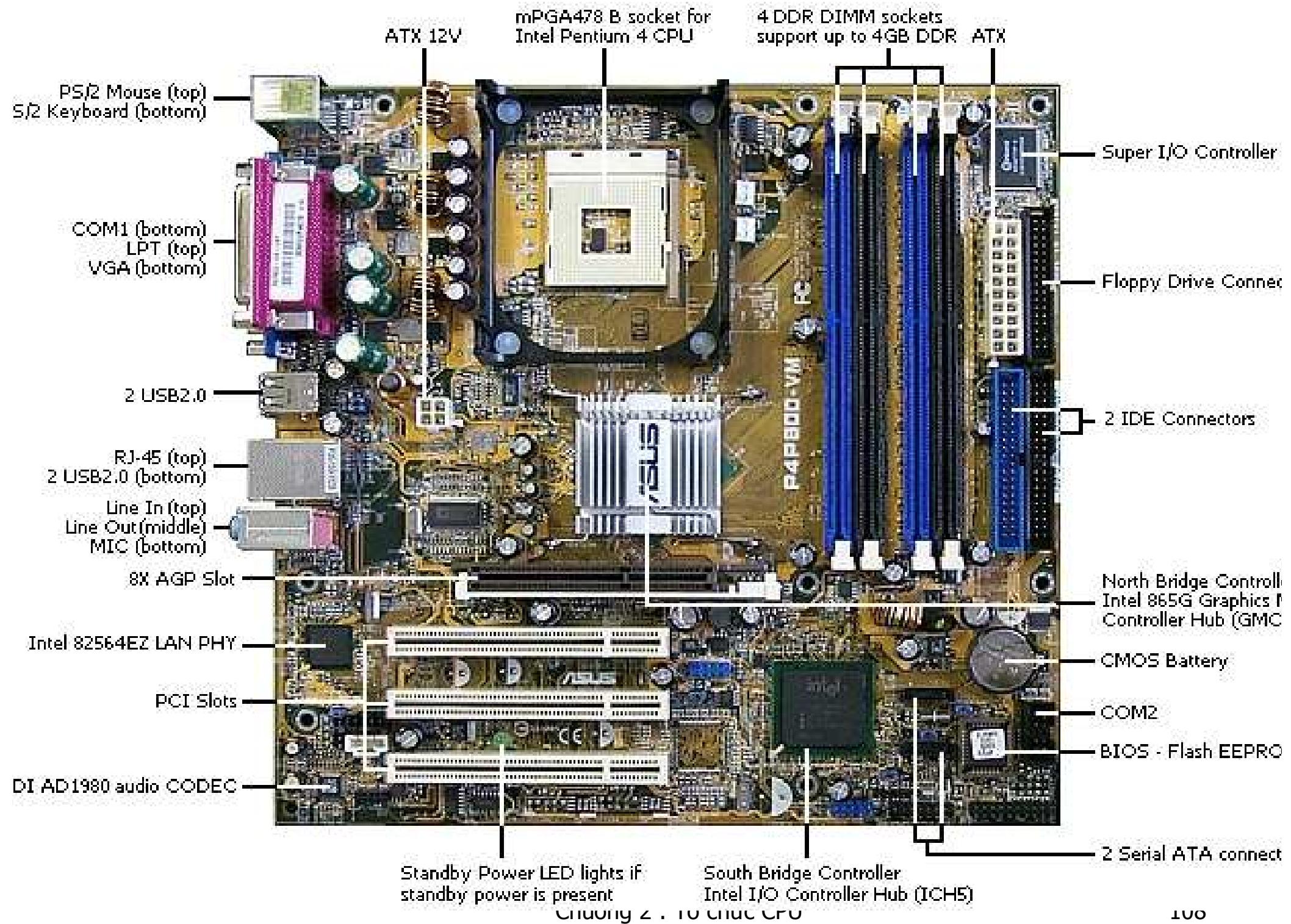
Bài 5: Giả sử thanh ghi trong MT của bạn dài 24 bits, cho biết giá trị của số dương lớn nhất mà thanh ghi này có thể chứa ở 2 hệ 2 và hệ 16?

Bài 6 : Biến đổi địa chỉ sau thành địa chỉ tuyệt đối

- a. 0950:0100
- b. 08F1:0200

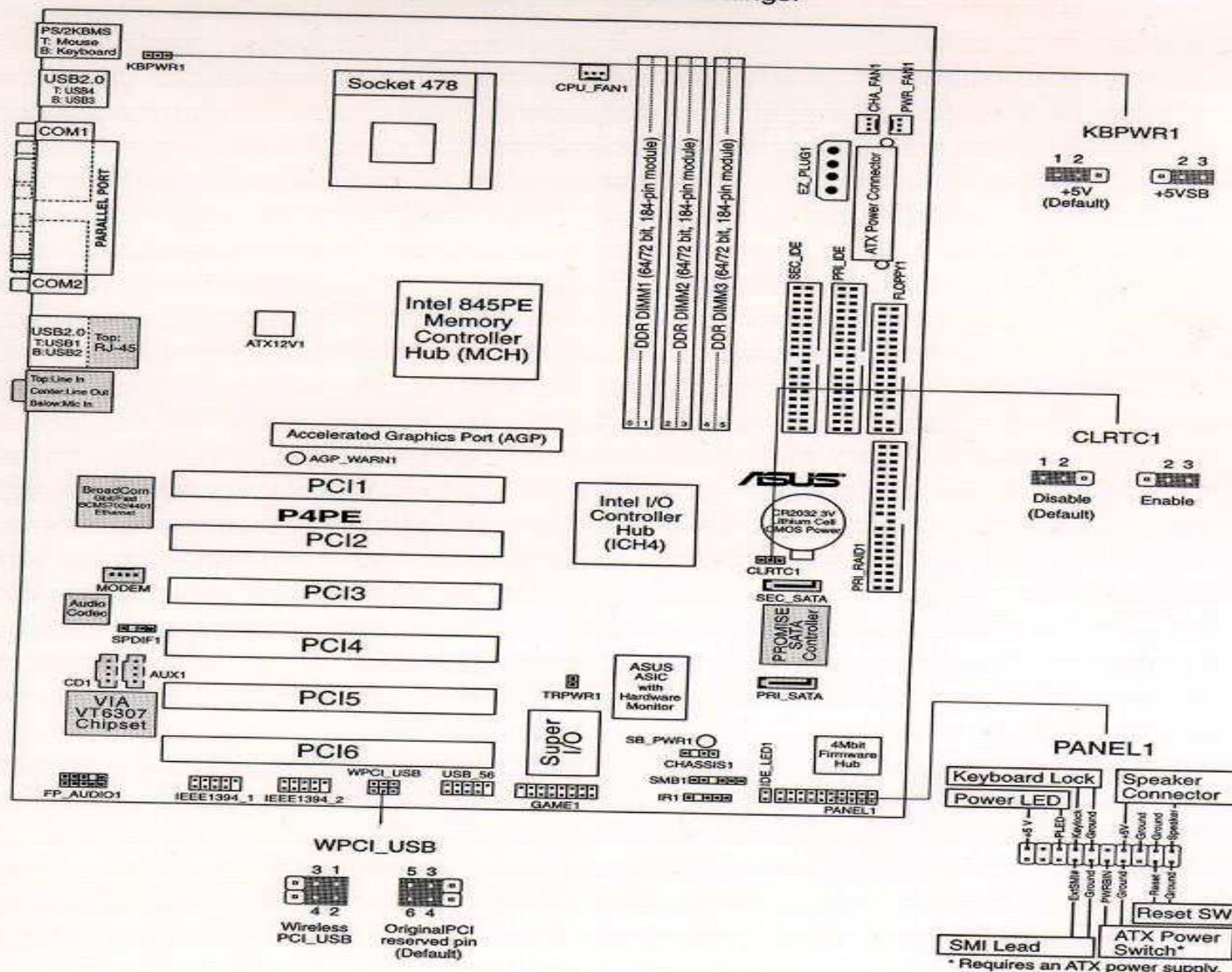


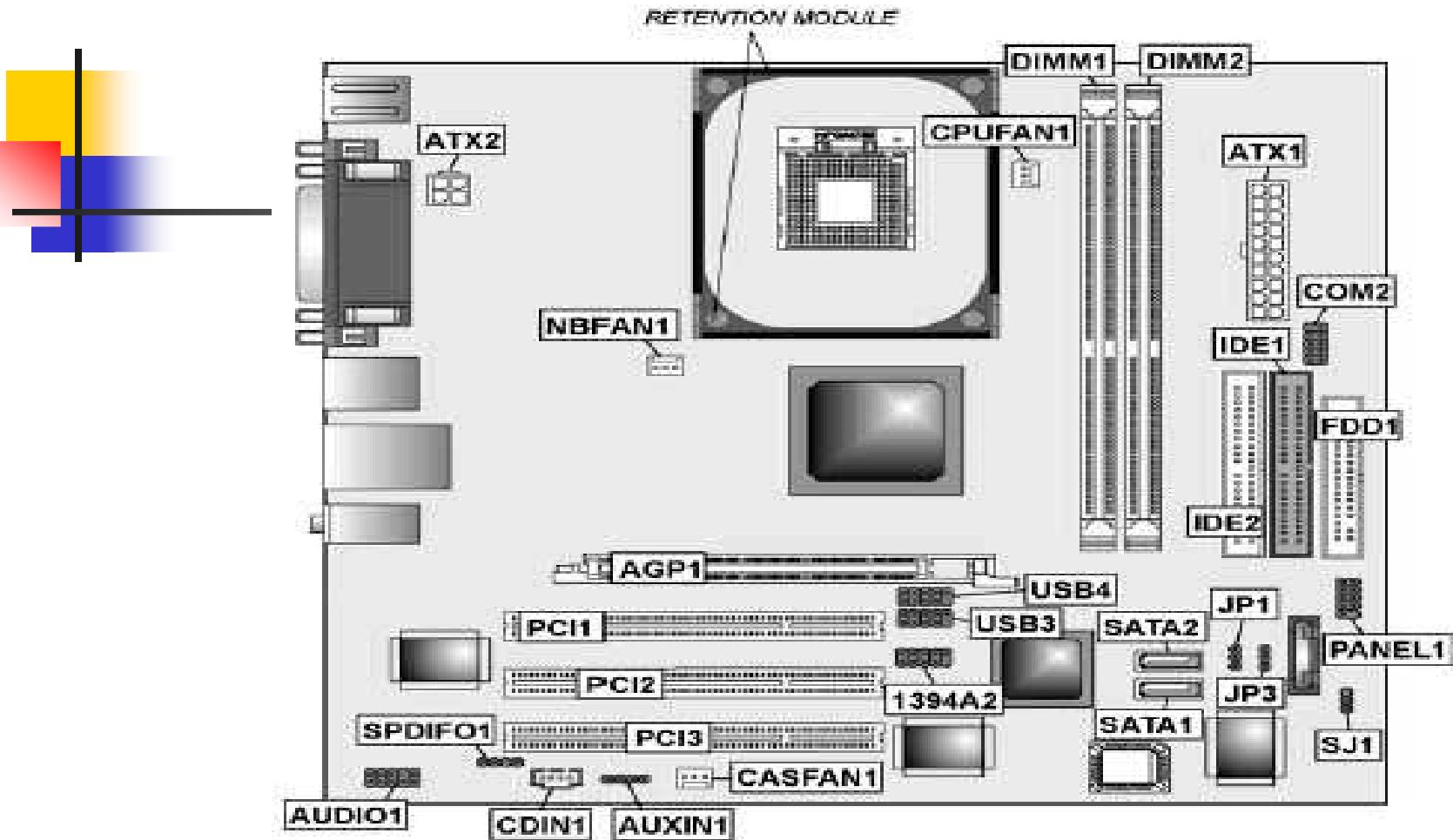




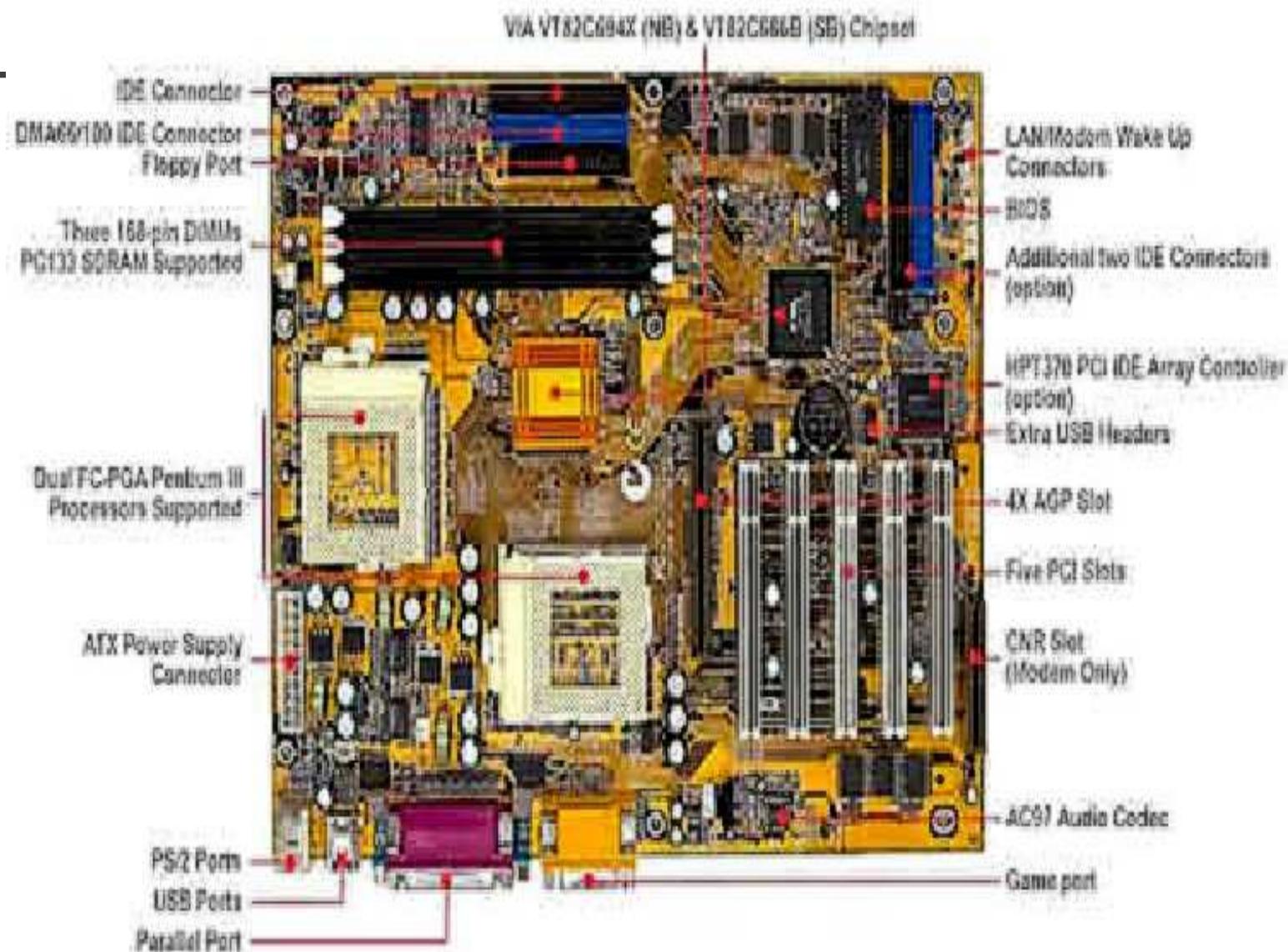
P4PE Motherboard Settings

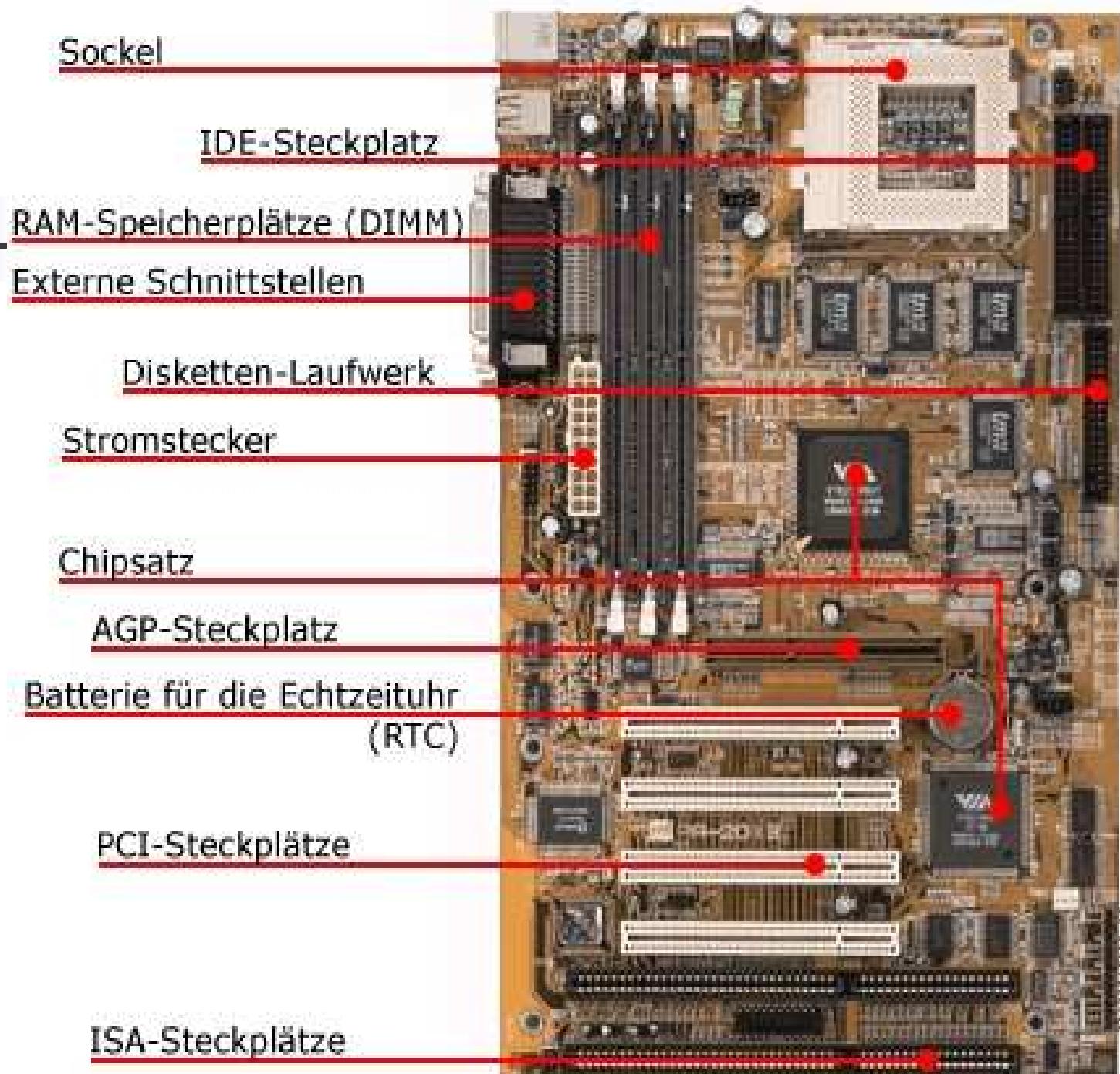
IMPORTANT! This diagram presents the general layout of your motherboard, and a summary of the switch and jumper settings. Before changing any setting, refer to the motherboard user guide for detailed descriptions and important notes on the settings.





MAINBOARD





CHƯƠNG TRÌNH GỠ RỐI DEBUG

Mục tiêu

- Dịch được 1 chương trình ngắn
- Xem các thanh ghi và cờ của CPU
- Xem sự thay đổi nội dung của các biến
- Dò tìm trị ở dạng nhị phân hoặc ASCII trong bộ nhớ
- Hỗ trợ luyện tập viết chương trình bằng Assembly

Dạng lệnh của Debug

■ <mã lệnh> <thông số>

Trong đó mã lệnh là 1 trong các chữ A,B,C,D,E, ... còn thông số thì thay đổi tùy theo lệnh.

Các thông số có thể là :

Địa chỉ : là 1 bộ địa chỉ đầy đủ segment : offset hay chỉ cần offset là đủ. Segment có thể dùng tên thanh ghi.

Ex : F000:0100

DS: 200

0AF5

Dạng lệnh của Debug

Tập tin : là 1 tham khảo tên tập tin đầy đủ, ít nhất phải có tên tập tin.

Danh sách :

Là 1 hay nhiều trị byte hoặc chuỗi cách nhau bằng dấu phẩy.

Khoảng : là 1 tham khảo đến vùng bộ nhớ

Trị : là 1 số hệ 16 có tối đa có 4 chữ số

Tập lệnh của Debug

Thí dụ minh họa lệnh A

- Phải nhập lệnh vào theo từng dòng một và kết thúc bằng Enter.
- Kết thúc nhập nhấn Enter ở dòng trống.

■ Ex : - A 100

5514:0100

MOV AH, 2

User gõ vào

5514:0102

MOV DL, 41

5514:0104

INT 21H

SEGMENT

OFFSET

C (Compare)

- So sánh 2 vùng bộ nhớ và liệt kê các ô nhớ có nội dung khác nhau.

Cú pháp : C <khoảng> , <địa chỉ>

Ex : - C 100, 200, 3000 : 1000

So sánh ô nhớ DS:100h với ô nhớ 3000:1000h, ô nhớ DS:101h với ô nhớ 3000:1001h.... Cho đến ô nhớ DS :200h với ô nhớ 3000:1100h.

→ So sánh 101 bytes

D (Dump)

- Hiện nội dung bộ nhớ theo dạng hệ 16 và ASCII.

Caùch goïi : D <khoảng>

Ex : - D F000 : 0

- D ES : 100

- D 100

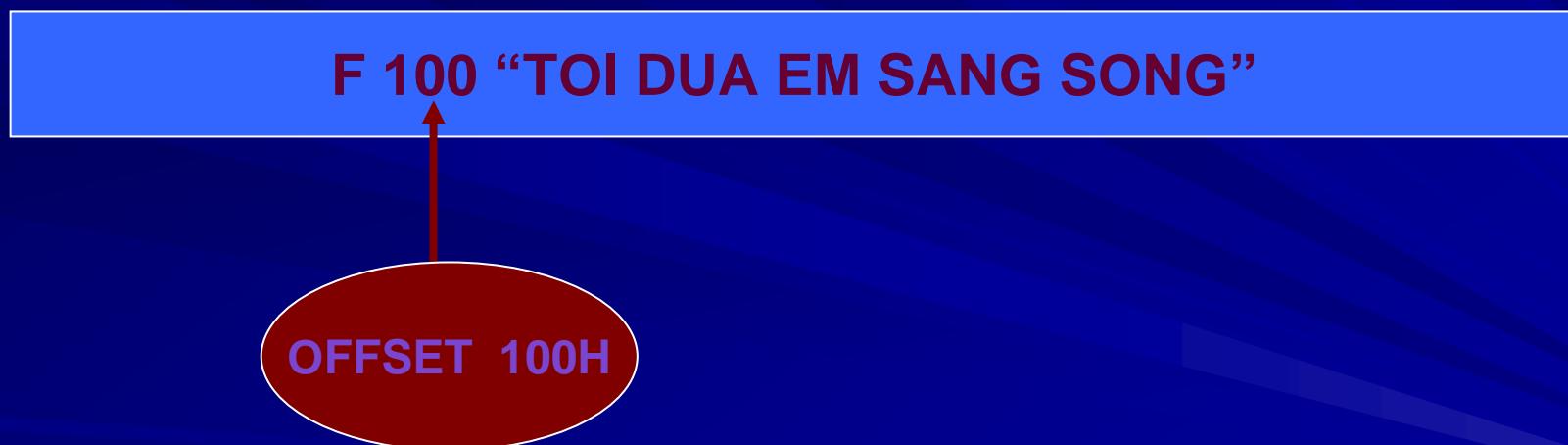
Lệnh F (Fill)

- Cú pháp : F <khoảng> <danh sách>
- Công dụng : lấp đầy trị vào vùng nhớ ngay tại địa chỉ mong muốn.

Trị nhập vào từng byte một theo hệ 16
Đấu trừ (-) dùng để lùi lại 1 địa chỉ.
SPACE BAR dùng để tới 1 địa chỉ.
ENTER để kết thúc.

Minh họa lệnh F

- Lắp đầy vùng nhớ tại địa chỉ offset 100h chuỗi “ Toi dua em sang song”.



KẾT QUẢ

-F 100 "TOI DUA EM SANG SONG"

-D 100

0ADD:0100	54 4F 49 20 44 55 41 20-45 4D 20 53 41 4E 47 20	TOI DUA EM SANG
0ADD:0110	53 4F 4E 47 54 4F 49 20-44 55 41 20 45 4D 20 53	SONGTOI DUA EM S
0ADD:0120	41 4E 47 20 53 4F 4E 47-54 4F 49 20 44 55 41 20	ANG SONGTOI DUA
0ADD:0130	45 4D 20 53 41 4E 47 20-53 4F 4E 47 54 4F 49 20	EM SANG SONGTOI
0ADD:0140	44 55 41 20 45 4D 20 53-41 4E 47 20 53 4F 4E 47	DUA EM SANG SONG
0ADD:0150	54 4F 49 20 44 55 41 20-45 4D 20 53 41 4E 47 20	TOI DUA EM SANG
0ADD:0160	53 4F 4E 47 54 4F 49 20-44 55 41 20 45 4D 20 53	SONGTOI DUA EM S
0ADD:0170	41 4E 47 20 53 4F 4E 47-54 4F 49 20 44 55 41 20	ANG SONGTOI DUA

D (DUMP)

Mục đích : in nội dung bộ nhớ trong MT ra màn hình dưới dạng số hex.

Cú pháp : D [address]

D [range]

Ex : in nội dung vùng nhớ đã lấp đầy ở ví dụ trước ở địa chỉ 100h

Ex2 : xem nội dung vùng nhớ 16 bytes bắt đầu ở địa chỉ F000:100

- D F000:100 L10

Thí dụ minh họa lệnh D

- đánh vào lệnh D để xem nội dung vùng nhớ của 30h bytes bộ nhớ từ địa chỉ 0000:0040 đến 0000:006F

- D 0000:0040 006F

Địa chỉ bắt đầu

- D 0000:0040 L 30

Số bytes

E (ENTER)

- Dùng để đưa dữ liệu byte vào bộ nhớ ngay tại địa chỉ mong muốn.

Caùch goïi :

- E <địa chỉ> <danh sách>

Trị nhập vào theo dạng số 16 từng byte một
Dấu - dùng để lùi lại 1 địa chỉ
Space Bar dùng để tới 1 địa chỉ
Enter dùng để kết thúc

Minh họa lệnh E

■ Mục đích : thay đổi nội dung bộ nhớ.

Cú pháp : - E [address] [list]

Ex : thay đổi 6 bytes bắt đầu ở địa chỉ 100 thành “ABCDE”

- E 100 “ABCDE”

Debug lấy đoạn chỉ bởi DS
Nếu ta không qui định địa chỉ
đoạn

Leānh U (Unassemble)

- công dụng : in ra 32 bytes mã máy của chương trình trong bộ nhớ ra màn hình dưới lệnh gợi nhớ.
- cú pháp : U [address]
U [range]

Ex : U 100 119

In ra màn hình các lệnh mã máy từ địa chỉ CS:100 đến CS:119

Lệnh R (Register)

- Công dụng : xem và sửa nội dung thanh ghi.
 - Cú pháp : - R enter (xem tất cả thanh ghi)
xem thanh ghi AX : - R AX
xem thanh ghi cờ : R F
- Ex : muốn bật thanh ghi cờ CF và ZF ta nhập CY và ZR.

Lệnh N (Name)

- Công dụng : tạo tập tin cần đọc hay ghi trước khi dùng lệnh L hay W.
- Cú pháp : - N <tên file> [thông số] L [địa chỉ]

Thí dụ minh họa lệnh N

Ex : tạo tập tin Love.txt .

- Dùng lệnh R để xác định vùng địa chỉ dành cho User.
- Dùng lệnh để đưa câu thông báo “ I love you more than I can say’ ở địa chỉ 2000:100.
- Dùng lệnh D để kiểm tra vùng nhớ tại địa chỉ 2000:100.
- Dùng lệnh N để đặt tên tập tin trên đĩa.
 - N Love.txt
- Dùng lệnh R để định số byte cần thiết ghi lên đĩa trong 2 thanh ghi BX và CX. Cụ thể trong trường hợp này số byte cần ghi là 1Eh byte.
 $BX = 0000$ $CX = 1E$
- Dùng lệnh W 2000:100 để ghi dữ liệu đã nhập vào tập tin ở địa chỉ bộ nhớ 2000:100.

- Thoát khỏi Debug và go đến lai tiếp tin theo cách sau :
C :\> Debug Love.txt
tìm xem Debug nãõ naip tiếp tin Love.txt
vào chỗ nào trong bộ nhớ.

Lệnh W (Write)

■ Cú pháp : W [address]

Thöôøng ñöôïc söû duïng chung vòùi leanh N

Ex : taïo taäp tin cou têân Love.txt

Bước 1 : dùng lệnh E để đưa câu ‘I love you more than I can say” vào ô nhớ ở địa chỉ 100.

Bước 2 : dùng lệnh D để kiểm tra lại địa chỉ 100

Bước 3 : dùng lệnh N để đặt tên tập tin : - N Love.txt

Bước 4 : dùng lệnh R để định số byte cần ghi lên đĩa trong 2 thanh ghi BX và CX. (BX chứa 16 bit cao, CX chứa 16 bit thấp).

Ở đây số byte cần ghi là 1Eh.

Bước 5 : dùng lệnh W để ghi câu trên đã nhập vào vùng nhớ có địa chỉ bắt đầu là 100.

Leähn T (Trace)vaø P

- cuù phaüp : - T [= <ñiaïi chæ>][soá laàn]

Muïc ñích : duøng ñeå chaïy 1 hay nhieàu laàn caùc leähn trong boä nhôù

Ex : - T = 3000:1000

Ex : - T = 3000:1000 <soá laàn>

Leānh L (Load)

- naïp taäp tin hoaëc naïp sector luaän lyù töø ñóa vaøo boä nhôù.

Cuù phaùp : - L <ñòa chæ> [<ñóa> <sector><soá>]

Daïng 1 : neáu chæ coù ñòa chæ duøng ñeå naïp taäp tin.
Teân taäp tin phaûi ñööïc gaùn tröôùc baèng leanh N.

Taäp tin luoân luoân ñööïc gaùn ôû ñòa chæ offset 100h
Daïng 2 : neáu coù ñaày ñuû caùc thoâng soá , duøng ñeå ñoïc sector
luaän lyù treân ñóa vaøo boä nhôù.

Ñóa : = 0 oå ñóa A, =1 oå ñóa B, =2 oå ñóa C

Leähn H (Hex Arithmetic)

- thöïc hieän pheùp coäng vaø tröø heä 16

Cuù phaùp : - H <trò 1> <trò 2>

Keát quaû : hieän ra toång vaø hieäu cuûa trò 1 vaø trò 2

Lệnh S (Search)

- Công dụng : tìm kiếm trị trong 1 vùng bộ nhớ.
- Cú pháp : - S <khoảng> <danh sách>
- Giải thích : tìm kiếm trị có hiện diện trong vùng bộ nhớ đã chỉ định hay không? Nếu có Debug hiện các địa chỉ đầu của những nơi có chứa danh sách.
Ex : - S 100 L 1000 ‘DOS’

18AF : 0154

18AF : 0823

Ex2 : - S 2000 2200 13,15,8A, 8

Lệnh M (Move)

- Công dụng : chép nội dung vùng nhớ đến 1 địa chỉ khác.
- Cú pháp : - M <khoảng>
- Ex : - M 100 105 200

Chép 5 bytes từ DS:100 đến DS:200

Ex2 : - M CS:100 L 50 ES:300

Chép 50 bytes từ CS:100 đến ES:300

Lệnh I (Input)

- Công dụng : nhập 1 byte từ cổng xuất nhập và hiện ra màn hình.
- Cú pháp : - I <địa chỉ cổng>
địa chỉ cổng là số hệ 16 tối đa 4 chữ số.
- Ex : - I 37E
EC

Lệnh O (Output)

- Công dụng : xuất 1 byte ra cổng xuất nhập.
- Cú pháp :- O<địa chỉ cổng> <trị>
địa chỉ cổng là số hệ 16 tối đa 4 chữ số.
- Ex : - O 378 5E

Summary

- Dùng lệnh D để xem nội dung vùng nhớ tại địa chỉ của ROM BIOS F000:0000.
- Tương tự xem nội dung vùng nhớ RAM màn hình ở địa chỉ B800:0000; bảng vector ngắt quãng 0000:0000
- Gõ vào máy bằng lệnh A, đoạn chương trình sau ở địa chỉ 2000:0100

Summary

2000:0100	MOV AL,32
2000:0102	MOV AH, 4F
2000:0104	MOV CX, [200]
2000:0108	MOV WORD PTR [1800], 1
2000:010E	MOV BYTE PTR [1800], 1
2000:0113	

Xem lại đoạn chương trình vừa đánh trên bằng lệnh U. Chú ý quan sát phần mã máy. Tìm xem các toán hạng tức thời và các địa chỉ xuất hiện ở đâu trong phần mã máy của lệnh.

Phần mã máy của 2 câu lệnh cuối có gì khác nhau khi dùng các toán tử WORD PTR và BYTE PTR.

Summary

- Dùng lệnh E nhập vào đoạn văn bản sau vào bộ nhớ tại địa chỉ DS:0100

8086/8088/80286 Assembly language.

Copyright 1988, 1886 by Brady Books, a division of Simon, Inc.

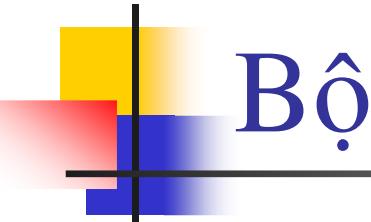
All right reserved, including the of reproduction in whole or in part, in any form.

(chú ý ký tự đầu dòng xuống dòng có mã ASCII là 0D và 0A).

BỘ NHỚ (Memory)

Mục tiêu :

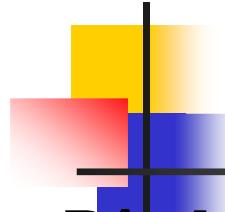
1. Hiểu được cấu tạo của bộ nhớ, chức năng và hoạt động của bộ nhớ.
2. Nắm được quá trình đọc bộ nhớ & ghi bộ nhớ.
3. Vai trò của bộ nhớ Cache trong máy tính.



Bộ nhớ (Memory)

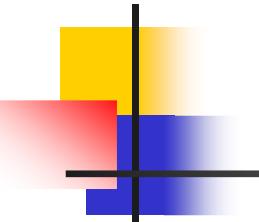
Nội dung :

1. Tổ chức bộ nhớ của máy tính IBM PC
2. Phân loại bộ nhớ : Primary Memory và Secondary Memory.
3. Quá trình CPU đọc bộ nhớ.
4. Quá trình CPU ghi bộ nhớ.
5. Bộ nhớ Cache.



Memory

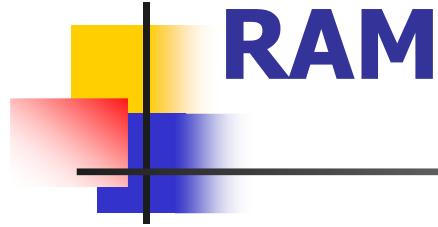
- **Bộ nhớ (Memory) là nơi chứa chương trình và dữ liệu.**
- **Đơn vị đo bộ nhớ :**
- **Bit : đơn vị bộ nhớ nhỏ nhất là bit. Mỗi bit có thể lưu trữ 1 trong 2 trạng thái là 0 và 1.**
- **Byte = 8 bits, được đánh chỉ số từ 0 đến 7 bắt đầu từ phải sang trái.**
- **Kbyte = 1024bytes = 2^{10} bytes.**
- **Mbyte = 1024Kbytes = 2^{10} Kbytes.**
- **Gbyte = 1024Mbytes = 2^{10} Mbytes.**



Primary Memory

Còn được gọi là bộ nhớ chính hay bộ nhớ trung tâm.

Chia làm 2 loại : RAM và ROM



RAM

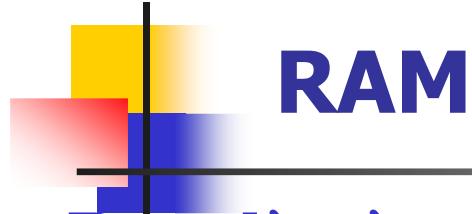
RAM (Random Access Memory) bộ nhớ truy xuất ngẫu nhiên. Là nơi lưu giữ các chương trình và dữ liệu khi chạy chương trình. Đặc điểm của RAM :

Cho phép đọc/ ghi dữ liệu.

Dữ liệu bị mất khi mất nguồn.



Khi máy tính khởi động, Ram rỗng. Người lập trình chủ yếu là làm việc với Ram – vùng nhớ tạm để dữ liệu và chương trình.



RAM

Ram là vùng nhớ làm việc → nếu vùng nhớ này trở nên nhỏ so với nhu cầu sử dụng thì ta tăng thêm Ram (gắn thêm Ram).

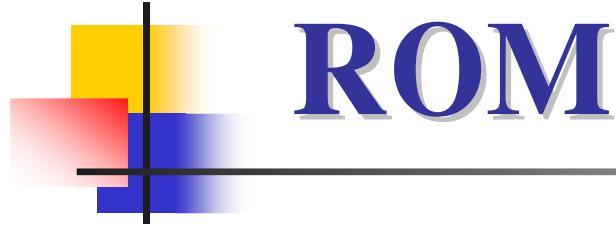
RAM có thể chia làm 2 loại : Dynamic và Static RAM

- **Dynamic RAM : phải được làm tươi trong vòng dưới 1 ms nếu không sẽ bị mất nội dung.**
- **Static RAM : giữ được giá trị không cần phải làm tươi.**
- **RAM tĩnh có tốc độ cao, có tên là bộ nhớ CACHE nằm trong CPU.**



RAM

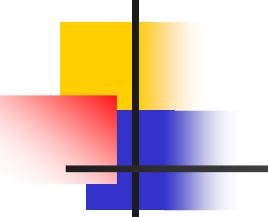




ROM (Read Only Memory) : bộ nhớ chỉ đọc.

ROM BIOS chứa phần mềm cấu hình và chẩn đoán hệ thống, các chương trình con nhập/xuất cấp thấp mà DOS sử dụng. Các chương trình này được mã hoá trong ROM và được gọi là phần mềm (firmware).

Một tính năng quan trọng của ROM BIOS là khả năng phát hiện sự hiện diện của phần cứng mới trong MT và cấu hình lại hệ điều hành theo Driver thiết bị.



ROM(cont)

Đặc điểm của ROM:

- Chỉ cho phép đọc không cho phép ghi.
- Dữ liệu vẫn tồn tại khi không có nguồn.



Các loại Rom

PROM (Programmable Read Only Memory) :

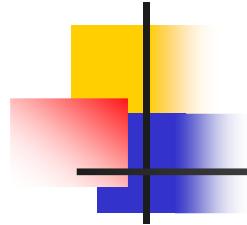
Cho phép user có thể lập trình và ghi vào ROM bằng cách đốt.

EPROM (Erasable Programmable Read Only Memmory)

Cho phép user viết ghi chương trình và xóa ghi lại. Việc xóa bằng cách dùng tia cực tím.

EEPROM (Electrically Erasable Programmable Read Only Memory)

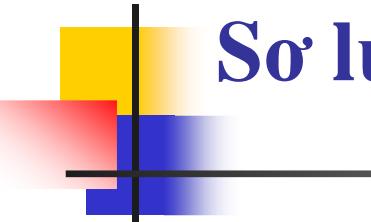
bộ nhớ có thể lập trình bằng xung điện đặc biệt



Secondary Memory

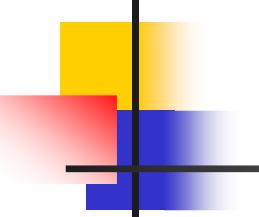
Là bộ nhớ phụ nằm ngoài hộp CPU.

Floppy disk, Tapes, Compact discs ... là secondary Memory.



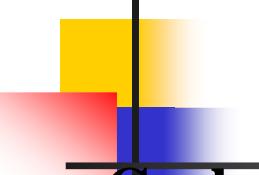
Sơ lược về Cache

- **Cache cấp 1 (Level 1-cache)** : nằm trong CPU, tốc độ truy xuất rất nhanh, theo tốc độ của CPU.
- **Cache cấp 2 (Level 2-cache)** : thường có dung lượng 128K, 256K là cache nằm giữa CPU và Ram, thường cấu tạo bằng Ram tĩnh (Static Ram), tốc độ truy xuất nhanh vì không cần thời gian làm tươi dữ liệu.
- **Cache cấp 3 (Level 3-cache)** : chính là vùng nhớ DRAM dùng làm vùng đệm truy xuất cho đĩa cứng và các thiết bị ngoại vi.
Tốc độ truy xuất cache cấp 3 chính là tốc độ truy xuất DRAM.



Cache (cont)

- **Tổ chức của Cache :** liên quan đến chiến lược trữ đệm và cách thức lưu thông tin trong Cache.
- **Loại lệnh phải thi hành :** Cache chứa cả chương trình và dữ liệu, khi CPU truy xuất mà chúng có sẵn thì truy xuất nhanh. Khi CPU cần truy xuất bộ nhớ, cache sẽ kiểm tra xem cái mà CPU cần đã có trong cache chưa.
- **Dung lượng cache :** như vậy nếu 1 tập lệnh nằm gọn trong cache (vòng lặp chẵng hạn) thì thực thi rất nhanh.



Cấu trúc Cache

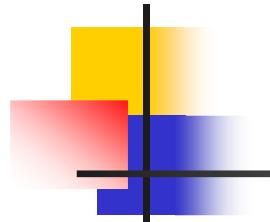
Cache được cấu tạo thành từng hàng (cache lines) , 32 bit/hàng cho 386, 128 bit/hàng cho 486, 256 bit/hàng cho Pentium.

Mỗi hàng có kèm theo 1 tag để lưu trữ địa chỉ bắt đầu của đoạn bộ nhớ mà thông tin được đưa vào cache. Nếu là cache cấp 2 (SRAM), địa chỉ bắt đầu của đoạn bộ nhớ đã chuyển data vào cache còn được lưu trong 1 vùng nhớ riêng.

Một bộ điều khiển cache (cache controller) sẽ điều khiển hoạt động của cache với CPU và data vào/ra cache. Chính Cache controller phản ánh chiến lược trữ đệm của cache.

Với cache cấp 1, cache controller là 1 thành phần của CPU.

Với cache cấp 2, cache controller nằm trên Mainboard.



Hiệu suất của Cache

Cache dùng làm vùng đệm truy xuất nên nếu CPU truy xuất data mà có sẵn trong cache thì thời gian truy xuất nhanh hơn nhiều. Hiệu quả của cache ngoài việc cho tốc độ truy xuất nhanh còn phụ thuộc vào Cache hit hoặc Cache miss.

Cache Hit : tức data có sẵn trong Cache.

Cache Miss : tức data chưa có sẵn trong cache.

tỉ lệ cache hit và cache miss phụ thuộc vào 3 yếu tố :

tổ chức cache , loại lệnh phải thi hành và dung lượng của cache.

Hiệu suất của Cache

Tính toán hiệu suất thực thi của Cache :

Gọi c thời gian truy xuất của Cache

M là thời gian truy xuất bộ nhớ

h là tỉ lệ thành công (hit ratio), là tỉ số giữa số lần tham chiếu cache với tổng số lần tham chiếu. $h = (k-1)/k$

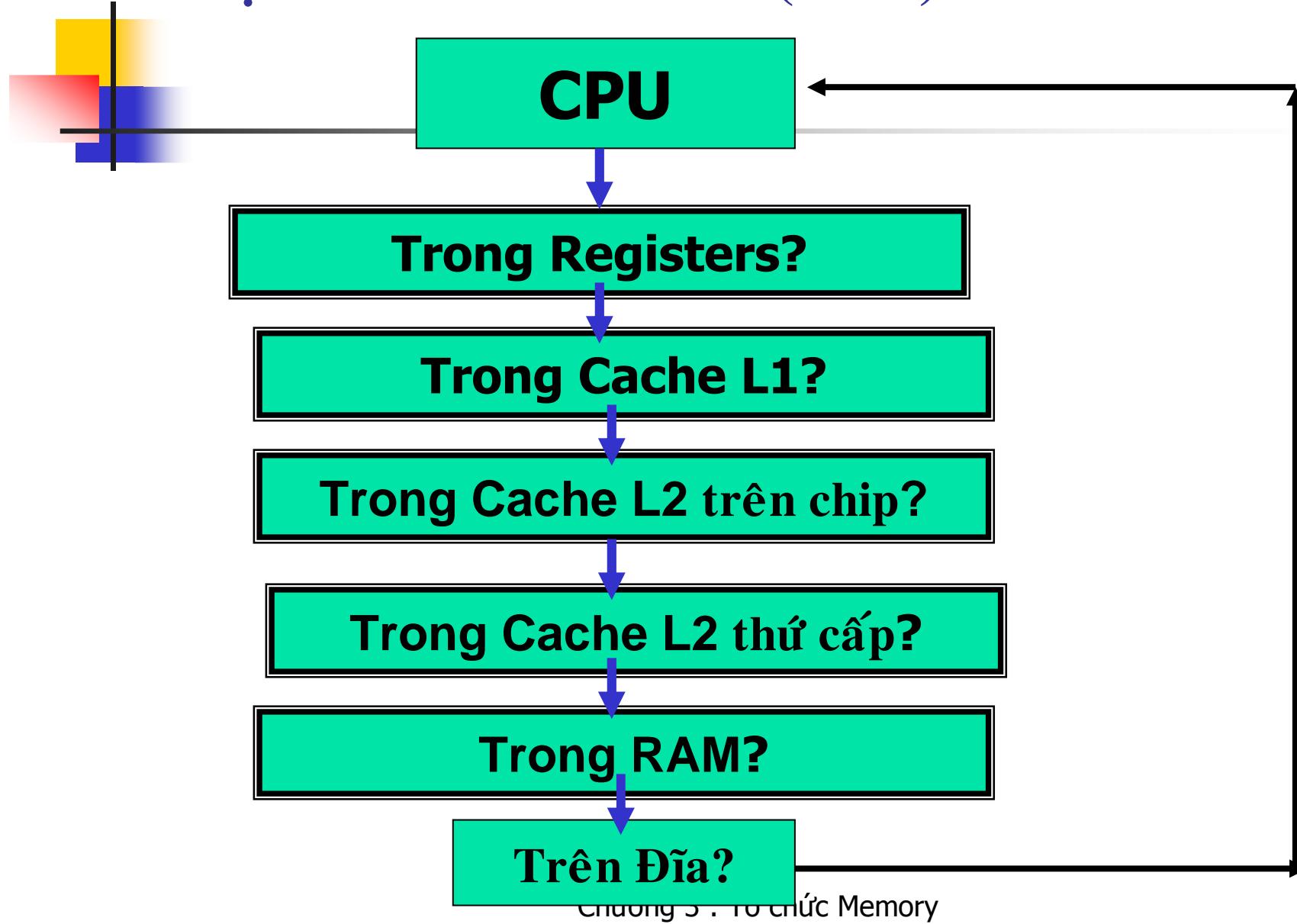
Tỉ lệ thất bại (miss ratio) $(1-h)$

Thời gian truy xuất trung bình $= c + (1-h)m$

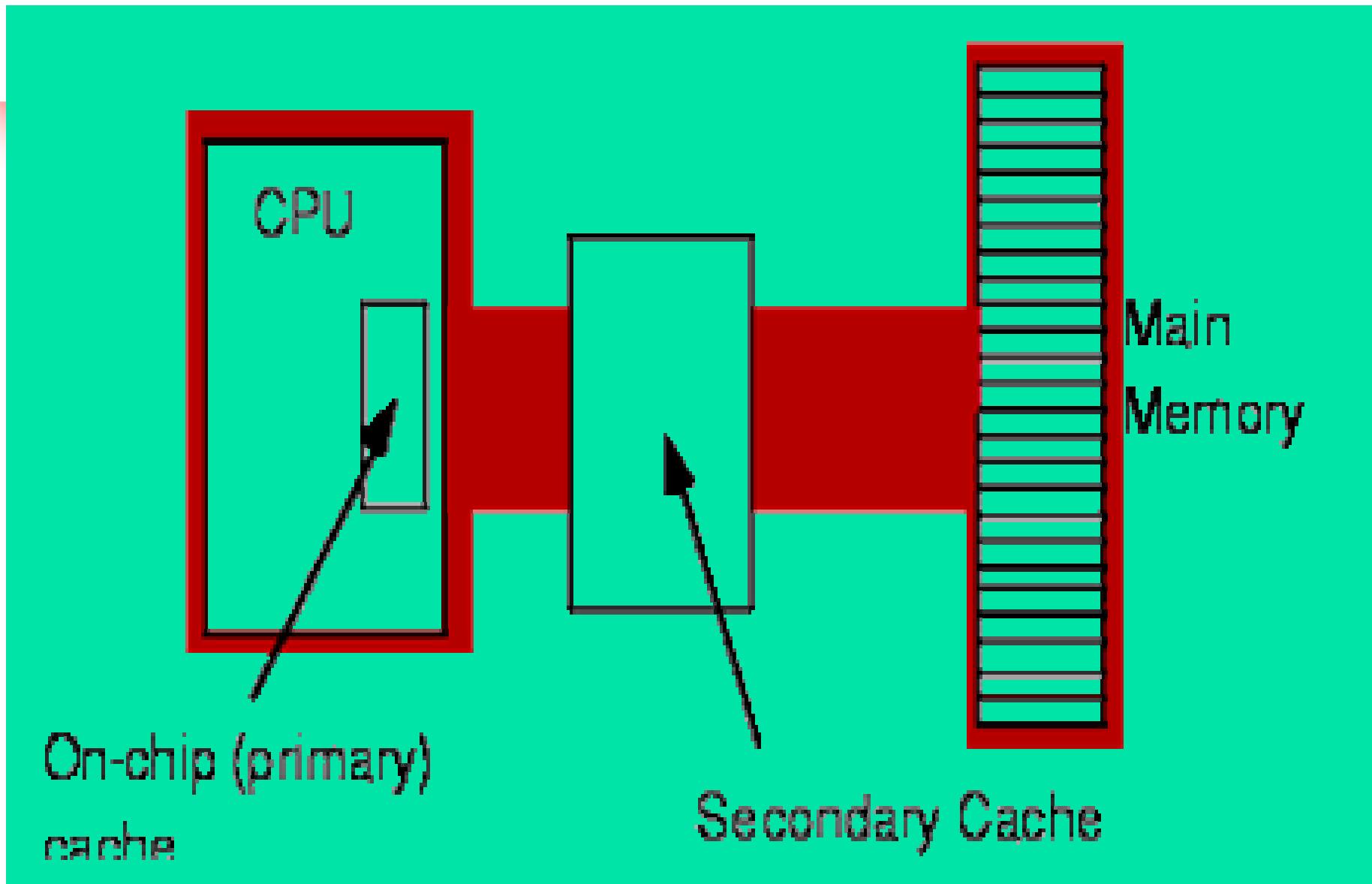
Khi $h \rightarrow 1$, tất cả truy xuất đều tham chiếu tới Cache, thời gian truy xuất trung bình $\rightarrow c$.

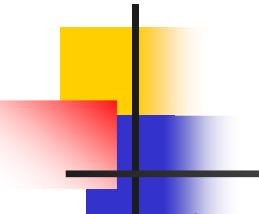
Khi $h \rightarrow 0$, cần phải tham chiếu bộ nhớ chính mọi lúc, thời gian truy xuất trung bình $\rightarrow c+m$.

Hiệu suất của Cache (cont)



A Two Level Caching System

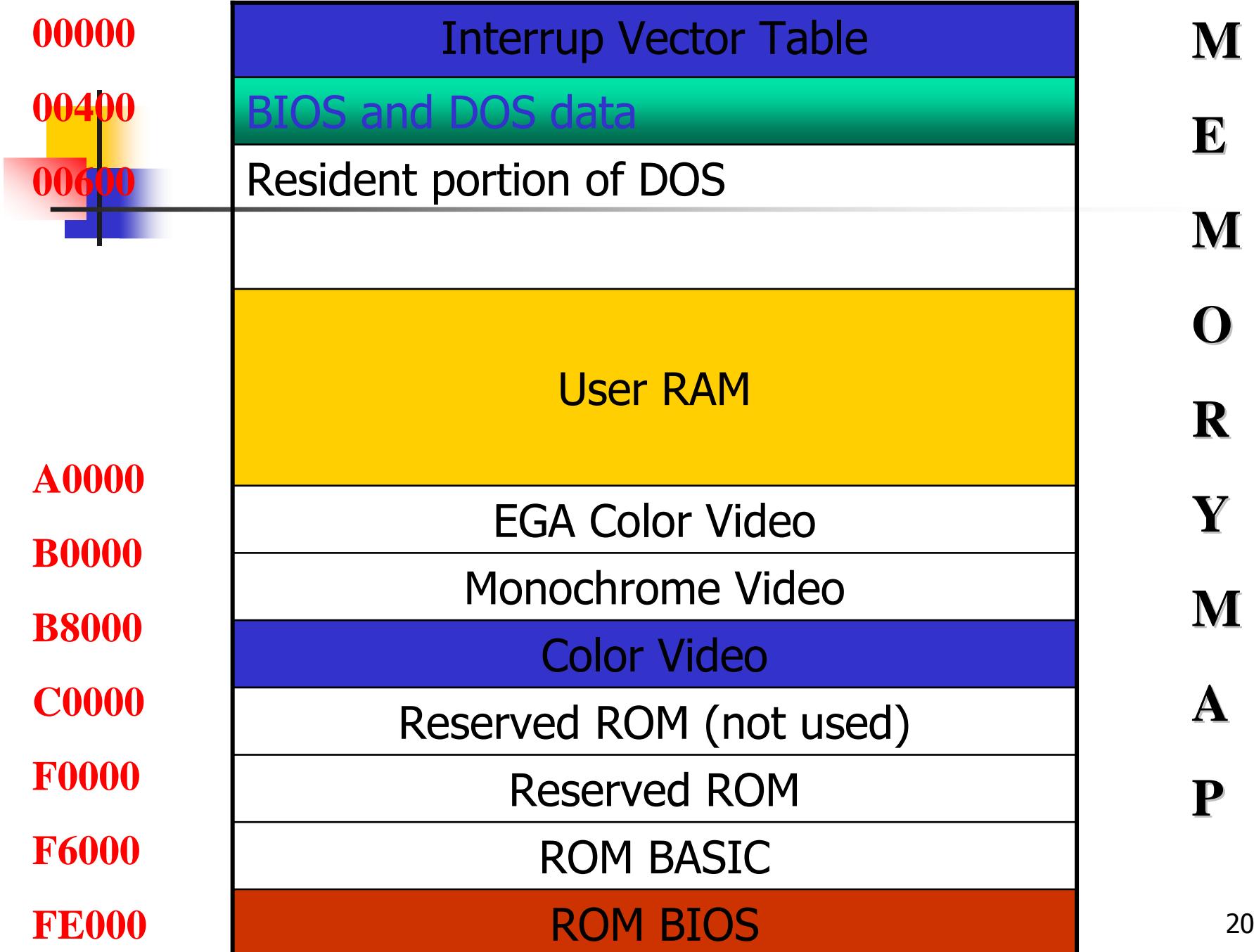




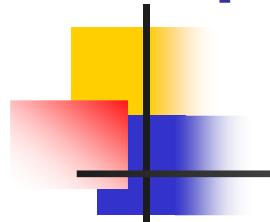
Các chiến lược trữ đệm trong Cache

Các chiến lược trữ đệm liên quan đến tác vụ đọc ghi từ CPU. Có 2 loại : Writethrough Cache (WTC) và Writeback cache (WBC).

- Khi CPU đọc từ bộ nhớ qui ước thì WTC và WBC đều như nhau : sẽ đọc 1 đoạn nội dung trong bộ nhớ vào cache.
- Khi CPU ghi ra bộ nhớ qui ước :
 - WTC : CPU ghi data ra vùng đệm ghi (write buffer) rồi bỏ đó tiếp tục việc khác, cache sẽ lấy nội dung trong buffer rồi chịu trách nhiệm ghi ra bộ nhớ qui ước khi bus rãnh.
 - WBC : CPU ghi data vào cache, khi cache đầy thì đẩy thông tin ra bộ đệm (đệm castoff) rồi từ castoof, data chuyển sang bộ nhớ qui ước.



Memory Map

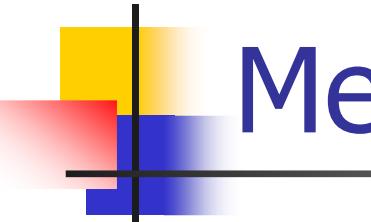


1024 bytes thấp nhất chứa bảng vector

interrupt

Dos data Area chứa các biến được DOS sử dụng như :

- Keyboard buffer : các phím nhấn được lưu cho đến khi được xử lý.
- Cờ chỉ tình trạng keyboard : cho biết phím nào đang được nhấn.
- Địa chỉ cổng printer.
- Địa chỉ cổng tuần tự
- Mô tả các thiết bị đang có trong hệ thống : tổng dung lượng bộ nhớ, số ổ đĩa, kiểu màn hình...



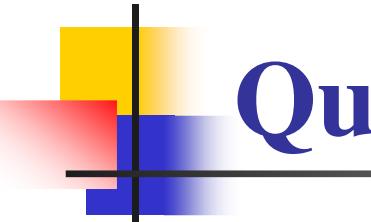
Memory Map

User Ram : vị trí thường trú của DOS ở địa chỉ 0600H.

Vùng nhớ trống nằm ngay dưới vùng nhớ Dos.

Rom Area : từ C000H – FFFFH được IBM dành riêng cho Rom sử dụng chứa hard disk controller, Rom Basic.

Rom BIOS : từ F000H – FFFFH vùng nhớ cao nhất của bộ nhớ chứa các chương trình con cấp thấp của Dos dùng cho việc xuất nhập và các chức năng khác..

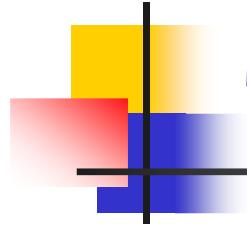


Quá trình Boot máy

■ Xãy ra khi ta power on hay nhấn nút Reset.

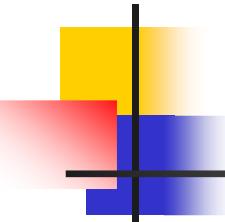
Bộ VXL xóa tất cả ô nhớ của bộ nhớ trở về 0, kiểm tra chẵn lẻ bộ nhớ, thiết lập thanh ghi CS trở đến segment FFFFh và con trỏ lệnh IP trở tới địa chỉ offset bằng 0.

→ Chỉ thị đầu tiên được MT thực thi ở địa chỉ ấn định bởi nội dung cặp thanh ghi CS:IP, đó chính là FFFF0H , điểm nhập tới BIOS trong ROM.



Trình tự tác vụ đọc ô nhớ

- ✓ CPU **đưa địa chỉ ô nhớ cần đọc vào thanh ghi địa chỉ.**
- ✓ Mạch giải mã xác định **địa chỉ ô nhớ.**
- ✓ CPU **gửi tín hiệu điều khiển đọc → bộ nhớ.** Nội dung ô nhớ cần đọc được **được đưa ra thanh ghi dữ liệu.**
- ✓ CPU **đọc nội dung của thanh ghi dữ liệu.**



Mạch giải mã địa chỉ ô nhớ

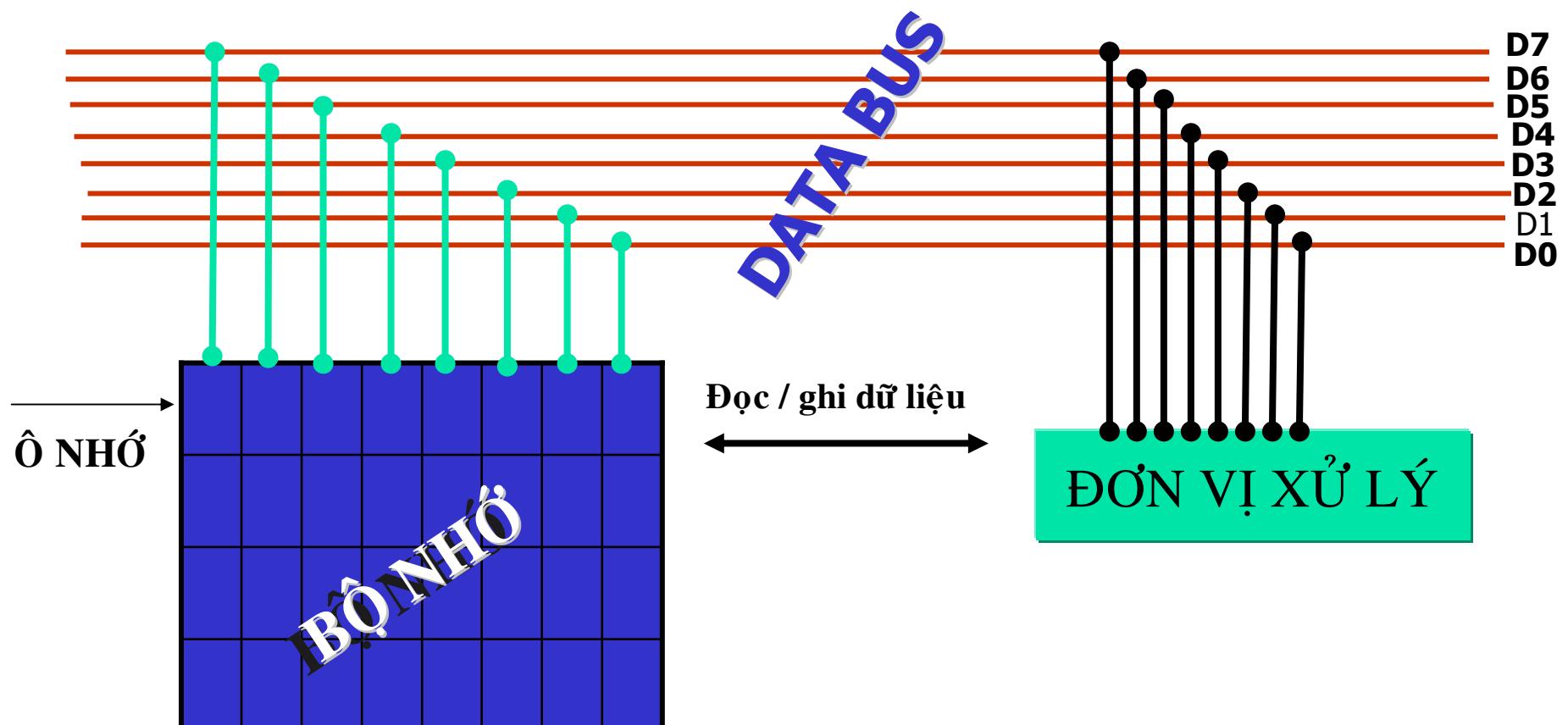
Mạch điện có nhiệm vụ xác định đúng ô nhớ cần truy xuất đang có địa chỉ lưu trong thanh ghi địa chỉ.

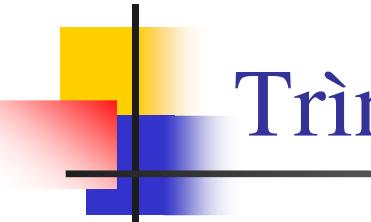
Bộ nhớ làm việc được chia thành nhiều ô nhớ.

Kích thước mỗi ô nhớ thay đổi tùy theo máy, thường là 8 hay 16 bit tức 1 byte hay 1 word.

Nếu kích thước mỗi ô nhớ là 1 byte thì sẽ có 8 đường dữ liệu song song nối bộ nhớ làm việc với bộ VXL. Mỗi đường 1 bit , tất cả 8 đường tạo thành một tuyến dữ liệu (data bus)

Truy xuất bộ nhớ (cont)

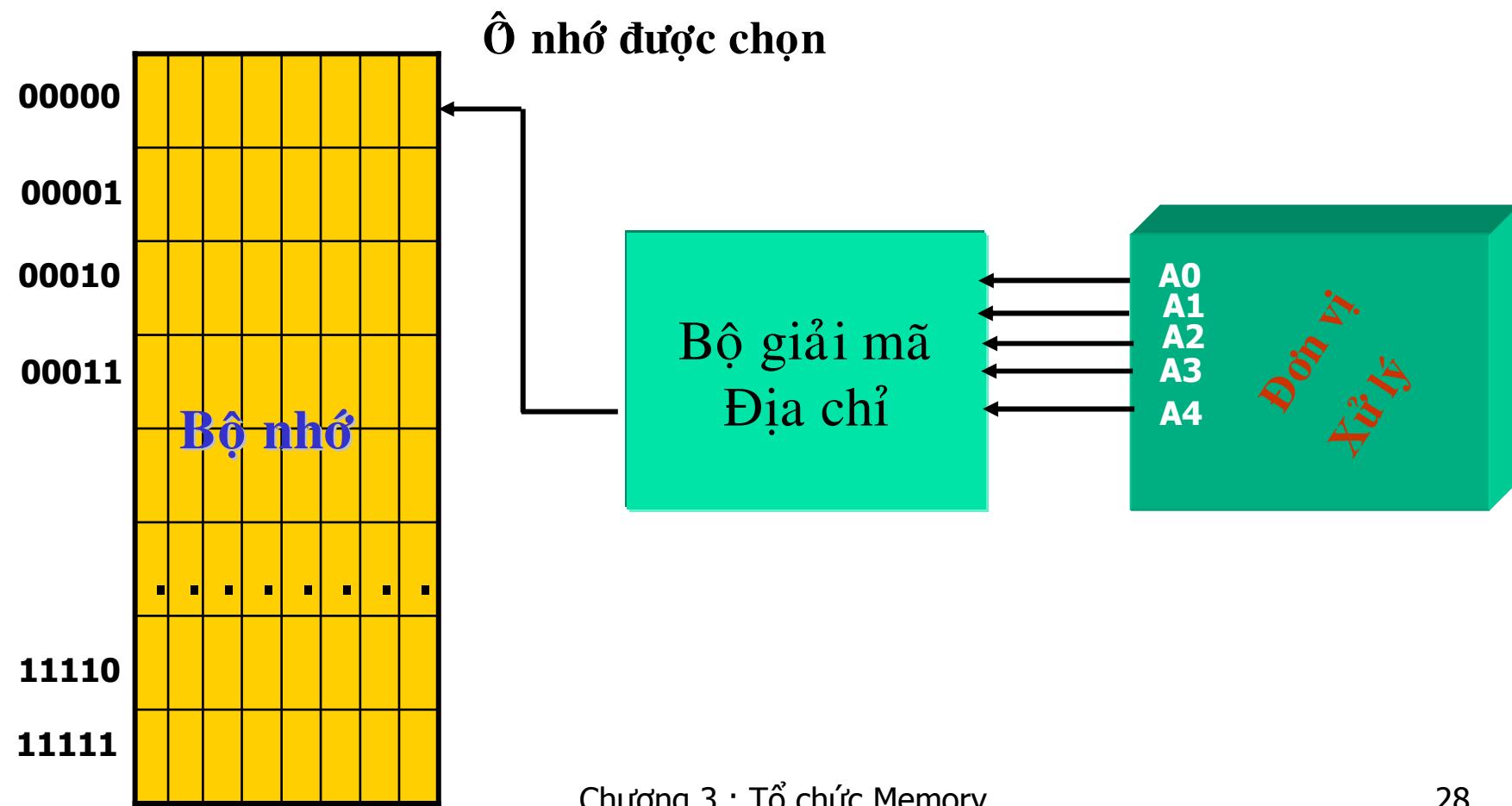


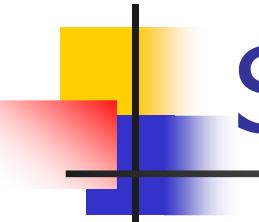


Trình tự tác vụ ghi ô nhớ

- CPU **đưa địa chỉ ô nhớ cần ghi vào thanh ghi địa chỉ của bộ nhớ.**
- Mạch giải mã xác định **địa chỉ ô nhớ.**
- CPU **đưa dữ liệu cần ghi vào thanh ghi dữ liệu của bộ nhớ.**
- CPU **gửi tín hiệu điều khiển ghi → bộ nhớ.** Nội dung trong thanh ghi dữ liệu **được ghi vào ô nhớ có địa chỉ xác định.**

Truy xuất bộ nhớ : ghi ô nhớ





Stack

- Stack là vùng nhớ đặc biệt dùng để lưu trữ địa chỉ và dữ liệu.

Stack thường trú trong stack segment. Mỗi vùng 16 bit trên stack được trỏ đến bởi thanh ghi SP, gọi là stack pointer.

Stack pointer lưu trữ địa chỉ của phần tử dữ liệu cuối mới được thêm vào (pushed lên stack.)



Stack

phần tử dữ liệu cuối mới được thêm vào này lại là phần tử sẽ được lấy ra (popped trước tiên).

→ Stack làm việc theo cơ chế LIFO (Last In First Out).

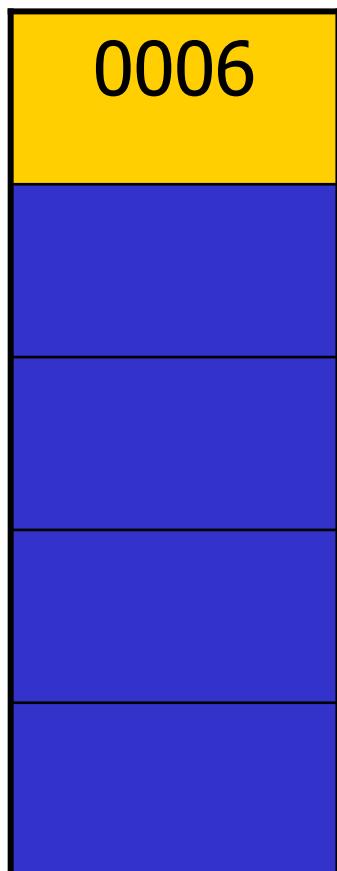
Xét ví dụ sau : giả sử stack đang chứa 1 giá trị 0006



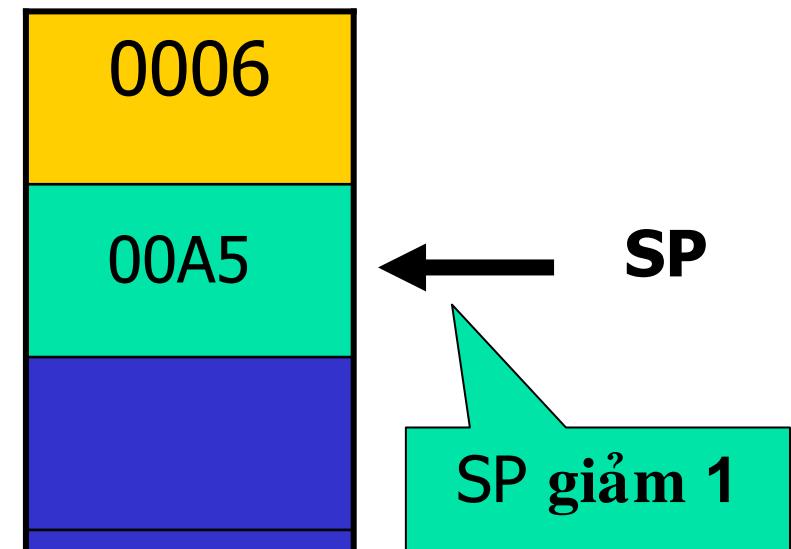
Sau đó ta đưa 00A5 vào stack

Stack

BEFORE **HIGH MEM**



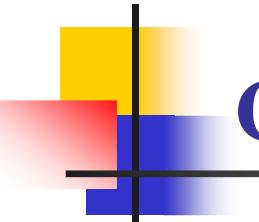
AFTER **HIGH MEM**



LOW MEM

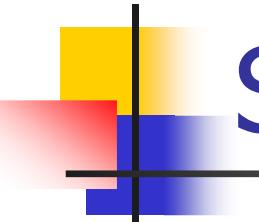
Chương 3 : Tổ chức Memory





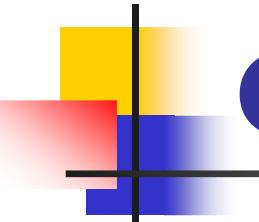
Công dụng của Stack

- Dùng để lưu trữ dữ liệu tạm cho thanh ghi nếu ta cần sử dụng các dữ liệu này.
- Khi 1 chương trình con được gọi, stack sẽ lưu trữ địa chỉ trả về ngay sau khi chương trình con thực hiện xong.
- Các ngôn ngữ cấp cao thường tạo ra 1 vùng nhớ bên trong chương trình con gọi là stack frame để chứa các biến cục bộ.



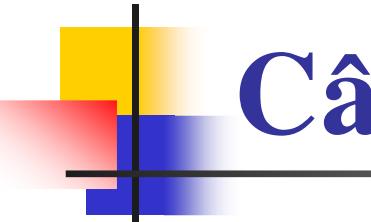
Summary Slide

- Cờ nào được thiết lập khi 1 phép tính số học không dấu quá rộng không vừa với đích?
- Hai thanh ghi nào được tổ hợp thành địa chỉ của lệnh sẽ được thực kế tiếp?
- Nếu quá trình đọc bộ nhớ. Tại sao quá trình đọc bộ nhớ lại chiếm nhiều chu kỳ máy hơn so với truy cập thanh ghi?
- Thanh ghi AH bị sửa đổi, tại sao thanh ghi AX cũng thay đổi theo.
- Nội dung nào chiếm 1024 bytes thấp nhất của bộ nhớ?



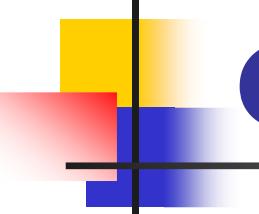
Câu hỏi ôn tập

- Vai trò của Cache trong máy tính.
- Trình bày chiến lược trữ đệm của Cache.
- Phân biệt bộ nhớ RAM và ROM.
- Nêu trình tự quá trình thực hiện khi khởi động máy tính.



Câu hỏi ôn tập

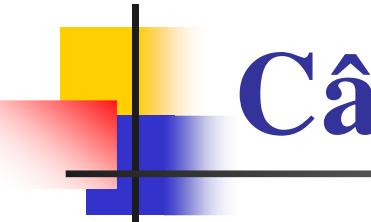
- Một bộ nhớ có dung lượng $4K \times 8$.
 - a) Có bao nhiêu đầu vào dữ liệu, đầu ra dữ liệu.
 - b) Có bao nhiêu đường địa chỉ.
 - c) Dung lượng của nó tính theo byte.



Câu hỏi ôn tập

Bộ nhớ Cache nằm giữa :

- a) Mainboard và CPU
- b) ROM và CPU
- c) CPU và bộ nhớ chính.
- d) Bộ nhớ chính và bộ nhớ ngoài



Câu hỏi ôn tập

Theo quy ước, người ta chia bộ nhớ thành từng vùng có những địa chỉ được mô tả bằng :

- a) số thập phân
- b) số thập lục phân
- c) số nhị phân
- d) số bát phân

Chương 5 : Nhập môn Assembly

Mục tiêu

- Hiểu ngôn ngữ máy và ngôn ngữ Assembly.
- Trình hợp dịch Assembler.
- Lý do nghiên cứu Assembly.
- Hiểu các thành phần cơ bản của Assembly
- Nắm được cấu trúc của 1 CT Assembly.
- Biết viết 1 chương trình Assembly.
- Biết cách dịch, liên kết và thực thi 1 chương trình Assembly.

Slide 1

h1 shjsahjsa
 huh, 13/10/2004

h2 ssasasasas
 huh, 13/10/2004

Giới thiệu ngôn ngữ Assembly

- Giúp khám phá bí mật phần cứng cũng như phần mềm máy tính.
- Nắm được cách phần cứng MT làm việc với hệ điều hành và hiểu được bằng cách nào 1 trình ứng dụng giao tiếp với hệ điều hành.
- Một MT hay một họ MT sử dụng 1 tập lệnh máy riêng cũng như 1 ngôn ngữ Assembly riêng.

Assembler

- Một chương trình viết bằng ngôn ngữ Assembly muốn MT thực hiện được ta phải chuyển thành ngôn ngữ máy.
- Chương trình dùng để dịch 1 file viết bằng Assembly → ngôn ngữ máy , gọi là Assembler.

Có 2 chương trình dịch:

MASM và TASM

Lý do nghiên cứu Assembly

- Đó là cách tốt nhất để học phần cứng MT và hệ điều hành.
- Vì các tiện ích của nó .
- Có thể nhúng các chương trình con viết bằng ASM vào trong các chương trình viết bằng ngôn ngữ cấp cao .

Lệnh máy

- Là 1 chuỗi nhị phân có ý nghĩa đặc biệt – nó ra lệnh cho CPU thực hiện tác vụ.
 - Tác vụ đó có thể là :
di chuyển 1 số từ vị trí nhớ này sang vị trí nhớ khác.
Cộng 2 số hay so sánh 2 số.

0 0 0 0 0 1 0 0

Add a number to the AL register

1 0 0 0 0 1 0 1

Add a number to a variable

1 0 1 0 0 0 1 1

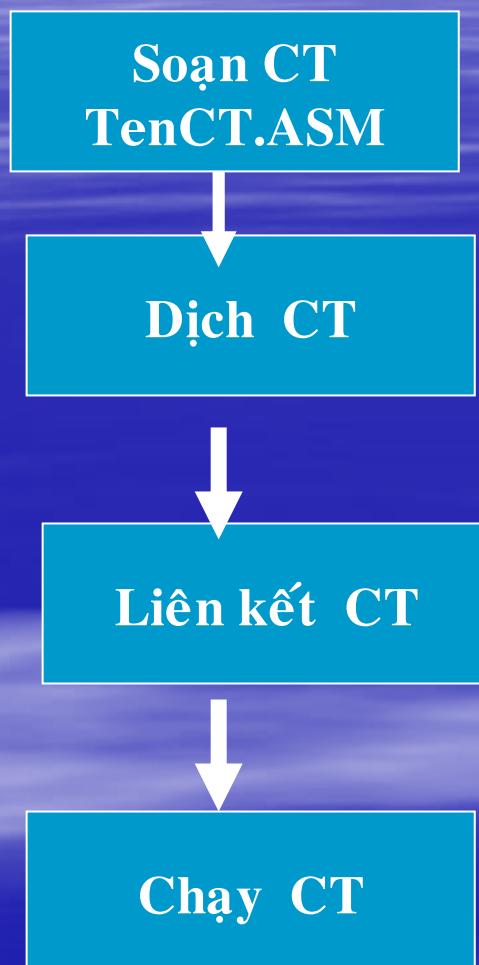
Move the AX reg to another reg

Lệnh máy (cont)

- Tập lệnh máy được định nghĩa trước, khi CPU được sản xuất và nó đặc trưng cho kiểu CPU .
 - Ex : B5 05 là 1 lệnh máy viết dạng số hex, dài 2 byte.
 - Byte đầu B5 gọi là Opcode
 - Byte sau 05 gọi là toán hạng Operand

Ý nghĩa của lệnh B5 05 : chép giá trị 5 vào reg AL

Cách viết 1 chương trình Assembly



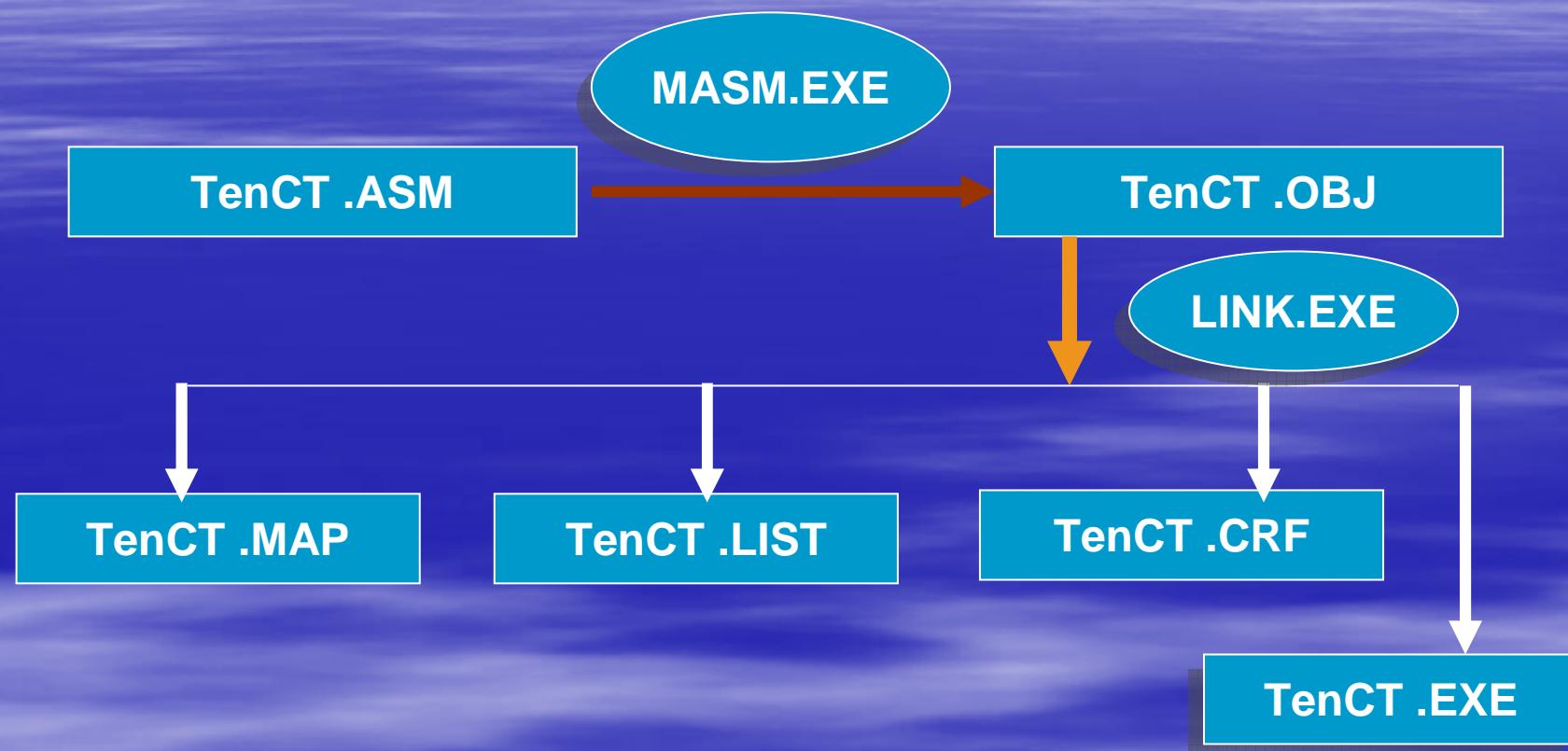
Dùng 1 phần mềm soạn thảo VB bất kỳ để soạn CT Assembly như : NotePad, NC, màn hình C, Pascal ...

CT có phần mở rộng là .ASM
dùng MASM để dịch chương trình nguồn .ASM
→ File Object.

dùng LINK để liên kết Object tạo tập tin thực hiện .EXE

Gõ tên tập tin thực hiện .EXE từ dấu nhắc DOS để chạy

Dịch và nối kết chương trình



Một chương trình minh họa

DOSSEG

.MODEL SMALL

.STACK 100h

.DATA

MES DB "HELLO WORD",'\$'

.CODE

MAIN PROC

MOV AX, @DATA

MOV DS, AX

MOV DX, OFFSET MES

MOV AH, 9

INT 21

MOV AH,4CH

INT 21

MAIN ENDP

END MAIN

Các file được tạo

- Sau khi dịch thành công file nguồn.ASM, ta có các file :
- File listing : file VB , các dòng có đánh số thứ tự mã.
- File Cross reference
- File Map
- File Obj
- File EXE

File Listing

- Microsoft (R) Macro Assembler Version 5.10 10/11/4
- Page 1-1

```
■ 1      DOSSEG
■ 2      .MODEL SMALL
■ 3      .STACK 100H
■ 4      .DATA
■ 5 0000 48 45 4C 4C 4F 20      MES DB "HELLO WORD$"
■ 6      57 4F 52 44 24
■ 7      .CODE
■ 8 0000      MAIN PROC
■ 9 0000 B8 ---- R      MOV AX,@DATA
■ 10 0003 8E D8      MOV DS, AX
■ 11 0005 B4 09      MOV AH,9
■ 12 0007 BA 0000 R      MOV DX, OFFSET MES
■ 13 000A CD 21      INT 21H
■ 14 000C B4 4C      MOV AH,4CH
■ 15 000E CD 21      INT 21H
■ 16 0010      MAIN ENDP
■ 17      END MAIN
■ ♀ Microsoft (R) Macro Assembler Version 5.10 10/11/4
```

Map File

- Start Stop Length Name Class
 - 00000H 0001FH 00020H _TEXT CODE
 - 00020H 0002AH 0000BH _DATA DATA
 - 00030H 0012FH 00100H STACK STACK
- Origin Group
 - 0002:0 DGROUP
- Program entry point at 0000:0010

Giải thích

- .model small : dùng kiểu cấu trúc <= 64 K bộ nhớ cho mã , 64K cho dữ liệu.
- .Stack 100h : dành 256 bytes cho stack của chương trình .
- .Data : đánh dấu phân đoạn dữ liệu ở đó các biến được lưu trữ.
- .Code : đánh dấu phân đoạn mã chứa các lệnh phải thi hành.
- Proc : khai báo đầu 1 thủ tục, trong Ex này ta chỉ có 1 thủ tục Main.

Giải thích (cont)

- Chép địa chỉ đoạn dữ liệu vào thanh ghi AX.
- Sau đó chép vào thanh ghi DS
- Gọi hàm số 9 của Int 21h của Dos để xuất chuỗi ký tự ra màn hình.
- Thoát khỏi CT .
- Main endp : đánh dấu kết thúc thủ tục
- End main : chấm dứt chương trình

Các chế độ bộ nhớ

Kiểu	Mô tả
SMALL	Mã lệnh trong 1 đoạn. Dữ liệu trong 1 đoạn
MEDIUM	Mã lệnh nhiều hơn 1 đoạn. Dữ liệu trong 1 đoạn
COMPACT	Mã lệnh trong 1 đoạn. Dữ liệu nhiều hơn 1 đoạn
LARGE	Mã lệnh nhiều hơn 1 đoạn Dữ liệu nhiều hơn 1 đoạn, không có mảng nào > 64K
HUGE	Mã lệnh nhiều hơn 1 đoạn Dữ liệu nhiều hơn 1 đoạn, mảng có thể > 64K

Dạng lệnh

- [name] [operator] [operand] [comment]

Nhãn, tên biến
Tên thủ tục

Mã lệnh dạng
gọi nhở

Register, ô nhớ
Trị, hằng

Ex : MOV CX , 0

LAP : MOV CX, 4

LIST DB 1,2,3,4

Mỗi dòng chỉ chứa 1 lệnh và mỗi lệnh
phải nằm trên 1 dòng

INT 21H

- Lệnh INT số hiệu ngắn được dùng để gọi chương trình ngắn của DOS và BIOS.

Ngắn 21h

Muốn sử dụng hàm nào của INT 21h ta đặt function_number vào thanh ghi AH, sau đó gọi INT 21h

Function_number

1

2

9

chức năng

nhập 1 ký tự từ bàn phím

Xuất 1 ký tự ra màn hình.

Xuất 1 chuỗi ký tự ra màn

hình

Chuong 5 Nhap mon ASM

INT 21h (cont)

Hàm 1 : Nhập 1 ký tự

Input : AH =1

**Output : AL = mã ASCII của phím ấn
= 0 nếu 1 phím điều khiển được ấn**

Hàm 2 : Hiển thị 1 ký tự ra màn hình

Input : AH =2

DL = Mã ASCII của ký tự hiển thị hay ký tự điều khiển

Thí dụ minh họa

```
DOSSEG  
.MODEL SMALL  
.STACK 100H  
.CODE  
MAIN PROC  
    MOV AH , 2  
    MOV DL , '?'  
    INT 21H  
    MOV AH ,1  
    INT 21H  
    MOV BL,AL
```

```
MOV AH,2  
MOV DL, 0DH  
INT 21H  
MOV DL , 0AH  
INT 21H  
MOV DL , BL  
INT 21H  
MOV AX , 4C00H  
INT 21H  
MAIN ENDP  
END MAIN
```

KẾT QUẢ

? N
N

Thí dụ minh họa các hàm của INT 21

- In dấu ? ra màn hình :

MOV AH, 2

MOV DL, ‘?’

INT 21H

- Nhập 1 ký tự từ bàn phím :

MOV AH, 1

INT 21H

Biến

- Cú pháp : **[tên biến] DB | DW |... [trị khởi tạo]**
- Là một tên ký hiệu dành riêng cho 1 vị trí trong bộ nhớ nơi lưu trữ dữ liệu.
- Offset của biến là khoảng cách từ đầu phân đoạn đến biến đó.
- Ex : khai báo 1 danh sách aList ở địa chỉ 100 với nội dung sau :
`.data
aList db "ABCD"`

Biến (cont)

Lúc đó :

Offset	Biến
0000	A
0001	B
0002	C
0003	D

Khai báo biến

Từ gợi nhớ	Mô tả	Số byte	Thuộc tính
DB	Định nghĩa byte	1	Byte
DW	Từ	2	Word
DD	Từ kép	4	Doubleword
DQ	Từ tứ	8	Quardword
DT	10 bytes	10	tenbyte

Minh họa khai báo biến

KIỂU BYTE

- **Char db ‘A’**
- **Num db 41h**
- **Mes db “Hello Word”,'\$'**
- **Array_1 db 10, 32, 41h, 00100101b**
- **Array_2 db 2,3,4,6,9**
- **Myvar db ? ; biến không khởi tạo**
- **Btable db 1,2,3,4,5**
db 6,7,8,9,10

Minh họa khai báo biến

KIỂU WORD

DW 3 DUP (?)

DW 1000h, 'AB', 1024

DW ?

DW 5 DUP (1000h)

DW 256*2

DẠNG LUU TRỮ DỮ LIỆU KIỂU WORD :

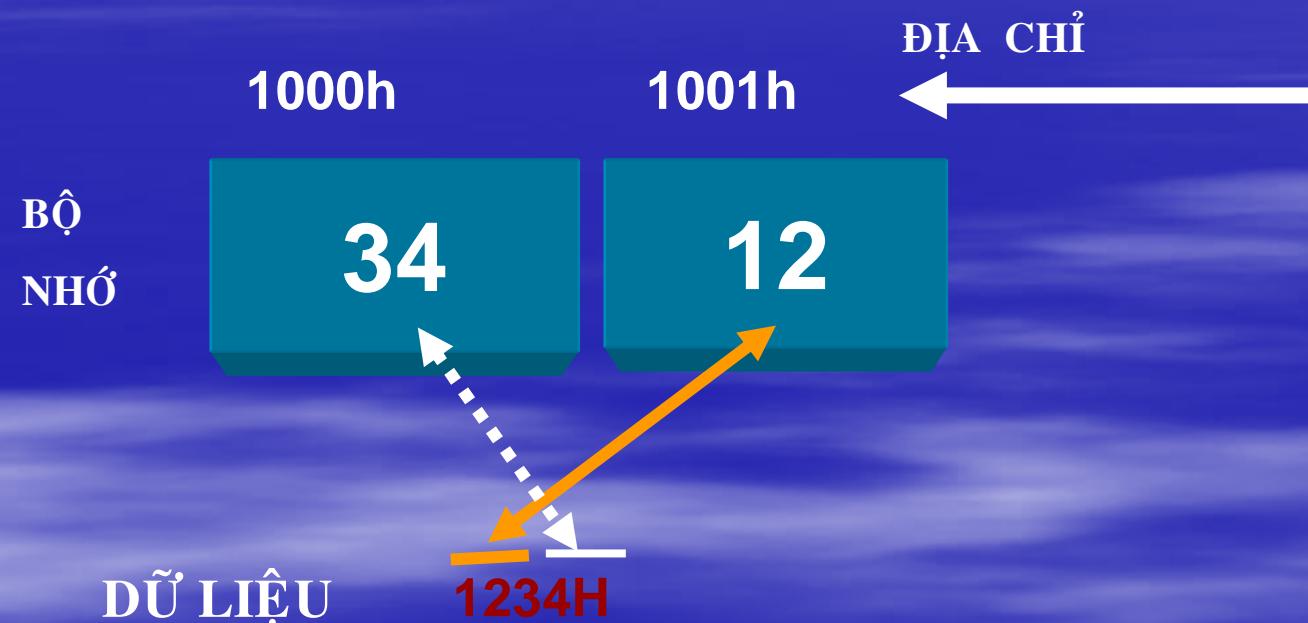
Trình hợp dịch đảo ngược các byte trong 1 giá trị kiểu WORD
khi lưu trữ trong bộ nhớ :

Byte thấp lưu ở địa chỉ thấp Byte cao lưu ở địa chỉ cao

Minh họa khai báo biến

KIẾU WORD

Ex : 1234h được lưu trữ trong bộ nhớ như sau :



Toán tử DUP

- Lặp lại 1 hay nhiều giá trị khởi tạo.
- Ex :

Bmem DB 50 Dup(?)

; khai báo vùng nhớ gồm 50 bytes.

db 4 dup (“ABC”)

; 12 bytes “ABCABCABCABC”

db 4096 dup (0)

; Vùng đệm 4096 bytes tất cả bằng 0

Khởi tạo biến

- Lưu ý :

Khi khởi tạo trị là 1 số hex thì giá trị số luôn luôn bắt đầu bằng 1 ký số từ 0 đến 9. Nếu ký số bắt đầu là A.. F thì phải thêm số 0 ở đầu.

- Ex :

Db A6H ; sai

Db 0A6h ; đúng

Toán tử DUP (cont)

Amtrix dw 3 dup (4 dup (0))

Tạo 1 ma trận 3x4

Atable db 4 dup (3 dup (0), 2 dup ('X'))

Tạo 1 vùng nhớ chứa 000XX 000XX 000XX 000XX

Toán tử DUP

- Chỉ xuất hiện sau 1 chỉ thị DB hay DW
- Với DUP ta có thể lặp lại 1 hay nhiều trị cho vùng nhớ.
- Rất có ích khi làm việc với mảng hay chuỗi.

Toán tử ?

- Muốn khai báo 1 biến hay 1 mảng mà không cần khởi tạo trị ta dùng toán tử ?

Ex : **MEM8 DB ? ; khai báo 1 byte trống trong bộ nhớ**

MEM16 DW ? ; khai báo 2 byte trống trong bộ nhớ

BMEM DB 50 DUP(?)

; khai báo 50 byte trống trong bộ nhớ

Chương trình dạng .COM

CODE SEGMENT

```
ASSUME CS:CODE , DS:CODE, SS:CODE  
; toàn bộ chương trình chỉ nằm trong 1 segment  
Org 100h ;; chỉ thị nạp thanh ghi lệnh IP=100h khi CT được nạp  
Main proc  
    mov ax,bx  
    .....  
Main endp  
Count db 10  
.....  
Code ends  
End main
```

SUMMARY

- chương trình Assembly gồm nhiều dòng lệnh.
- Mỗi lệnh phải viết trên 1 dòng
- Lệnh có thể gồm [tên] [toán tử] [toán hạng]
- Các ký tự phải đặt trong dấu ‘ ‘ hay “ “
- DB dùng để định nghĩa biến kiểu BYTE
- DW dùng để định nghĩa biến kiểu WORD.
- Có 2 cách xuất nhập dữ liệu : liên lạc trực tiếp qua cổng hay dùng các phục vụ ngắt của DOS và BIOS.

Câu hỏi ôn tập

- Trong mã máy dưới đây được lấy từ tập tin liệt kê, hãy nêu ý nghĩa của R

5B 0021 R ADD BX, VAL1

- Nêu ý nghĩa của ký hiệu địa chỉ của biến dưới đây trong 1 tập tin liệt kê.

5B 0021 R ADD BX, VAL1

Câu hỏi ôn tập

- Chương trình sau có lỗi. Hãy tìm câu lệnh nào gây ra lỗi, giải thích và sửa lại cho đúng.

```
.MODEL SMALL  
.STACK 100H  
.DATA  
    MOV AX, VALUE1  
    MOV BX, VALUE2  
    INC BX, 1  
    INT 21H  
    MOV 4C00H, AX  
MAIN ENDP  
  
    VALUE1    0AH  
  
    VALUE2    1000H  
  
END MAIN
```

- Chương trình sau có lỗi. Hãy tìm câu lệnh nào gây ra lỗi, giải thích và sửa lại cho đúng.

.MODEL SMALL

.STACK 100H

.CODE

MAIN PROC

MOV AX, @DATA

MOV DS , AX

MOV AX, VALUE1

MOV AX, VALUE2

MOV AX, 4C00H

INT 21H

MAIN ENDP

VALUE1 DB 0AH

VALUE2 DB 1000H

END MAIN

4/4/2006

Câu hỏi ôn tập

Bài tập lập trình

Bài 1 : Viết chương trình nhập 1 ký tự thường , in ra ký tự hoa tương ứng.

Bài 2 : Viết chương trình hoán vị 2 biến kiểu byte được gán sẵn trị.

Bài 3 : Viết chương trình tạo 1 array có các phần tử 31h,32h,33h,34h.

Nạp từng phần tử vào thanh ghi DL và xuất nó ra màn hình. Giải thích tại sao kết xuất trên màn hình là 1234.

Toán tử số học

Toán tử	Cú pháp	Công dụng
+	+ expression	Dương
-	- expression	Âm
*	exp1*exp2	Nhân
/	exp1/exp2	Chia
MOD	exp1 mod exp2	Phần dư
+	exp1 + exp2	Cộng
-	exp1 - exp2	Trừ
SHL	exp shl n	Dịch exp sang trái n bit
SHR	exp shr n	Dịch exp sang phải n bit

Toán tử logic

Not	Not expression
And	Exp1 and exp2
Or	Exp1 or exp2
Xor	Exp1 xor exp2

Ex : MOV AH , 8 OR 4 AND 2

MOV AL, NOT (20 XOR 0011100B)

Toán Tử Quan Hệ

- So sánh 2 biểu thức và cho trị là true (-1) nếu điều kiện của toán tử thỏa, ngược lại là false.

EQ	Exp1 EQ exp2	True nếu Exp1 = exp2
NE	Exp1 NE exp2	True nếu Exp1 <> exp2
LT	Exp1 LT exp2	True nếu Exp1 < exp2
LE	Exp1 LE exp2	True nếu Exp1 <= exp2
GT	Exp1 GT exp2	True nếu Exp1 > exp2
GE	Exp1 GE exp2	True nếu Exp1 >= exp2

ĐỘ ƯU TIÊN TOÁN TỬ

Độ ưu tiên
giảm dần

TOÁN TỬ	MÔ TẢ
()	Dấu ngoặc
+ , -	Dấu dương , âm
* / MOD	Nhân , chia, Modulus
+ , -	Cộng, trừ

Toán tử SEG

- Cú pháp :
SEG expression
- Cho địa chỉ đoạn của biểu thức expression.
- Expression có thể là biến | nhãn | tên segment
hay toán hạng bộ nhớ khác.

Toán tử OFFSET

- Cú pháp :
OFFSET **expression**
- Cho địa chỉ OFFSET của biểu thức expression.
- Expression có thể là biến | nhãn | tên segment
hay toán hạng trực tiếp bộ nhớ khác.

Ex : nạp địa chỉ segment và offset của biến table vào DS :AX
TABLE DB ?

MOV AX, SEG TABLE

MOV DS, AX

MOV DX, OFFSET Table

TOÁN TỬ \$

- Cho địa chỉ của OFFSET của phát biểu chứa toán tử \$.
- Thường được dùng để tính chiều dài chuỗi.

TOÁN TỬ PTR

Cú pháp : **type PTR expression**

■ Cho phép thay đổi dạng của expression

■ nếu expr là 1 biến | toán hạng bộ nhớ thì type có thể là byte , word hay dword.

■ Nếu expr là 1 nhãn thì type có thể là near hay far.

Ex : **mov ax, word ptr var1 ; var1 là toán hạng kiểu Word**

mov bl , byte ptr var2 ; var2 là toán hạng kiểu byte

Toán hạng (Operand)

Các toán hạng chỉ ra nơi chứa dữ liệu cho 1 lệnh , chỉ thị.

Hầu hết các lệnh Assembly đều có đối số là 1 hoặc 2 toán hạng
Có 1 số lệnh chỉ có 1 toán hạng như RET, CLC.

Với các lệnh 2 toán hạng thì toán hạng thứ 2 là toán hạng
nguồn (source) – chứa dữ liệu hoặc địa chỉ của dữ liệu.

Toán hạng (Operand)

- Toán hạng đích giữ kết quả (nếu có yêu cầu) sau khi thi hành lệnh.
- Toán hạng đích có thể là thanh ghi hay Bộ nhớ.

Toán hạng nguồn có thể là thanh ghi, bộ nhớ hay 1 giá trị tức thời .

Toán hạng số tức thời có thể là số trong các hệ đếm khác nhau và được viết theo qui định sau :

Số hệ 2 : xxxxxxxxB (x là bit nhị phân)

Số hệ 10 : xxxxxD hay xxxx (x là 1 số hệ 10)

Số hệ 16 : xxxxH và bắt đầu bằng số (x là 1 số hệ 16)



ĐỊNH VỊ THANH GHI

Giá trị của toán hạng được truy xuất nằm
ngay trong thanh ghi của CPU.

Ex : MOV AX,BX ; chuyển nội dung của
thanh ghi BX vào thanh ghi AX



Định vị gián tiếp thanh ghi :

EX1 : MOV AX, [SI]

Nạp nội dung của ô nhớ mà địa chỉ Offset lưu trong SI và địa chỉ đoạn lưu trong DS vào AX.

EX2 : MOV AX, [BP]

Nạp nội dung của ô nhớ mà địa chỉ Offset lưu trong BP và địa chỉ đoạn lưu trong ES vào AX.

ĐỊNH VỊ TRỰC TIẾP

Địa chỉ Offset của ô nhớ chứa dữ liệu toán hạng nằm trực tiếp trong câu lệnh còn địa chỉ segment ngầm định chứa trong DS.

Ex : MOV BX, [1234]

Nạp nội dung ô nhớ có địa chỉ DS:1234 → BX

ĐỊNH VỊ CƠ SỞ

Địa chỉ Offset của toán hạng được tính là tổng của nội dung thanh ghi BX hoặc BP và 1 độ dịch.

Độ dịch là 1 số nguyên âm hoặc dương. Địa chỉ đoạn là đoạn hiện tại.

ĐỊA CHỈ HIỆU DỤNG

Toán hạng bộ nhớ dùng trong tập lệnh vi xử lý 86 sử dụng phương pháp định địa chỉ tổng hợp được gọi là địa chỉ hiệu dụng.

Địa chỉ hiệu dụng là tổ hợp của 3 nhóm sau đặt trong dấu [].

Nhóm thanh ghi chỉ số : SI , DI

Nhóm thanh ghi nền : BX, BP

Địa chỉ trực tiếp : số 16 bit

Các thanh ghi trong cùng 1 nhóm không được xuất hiện trong cùng 1 địa chỉ hiệu dụng.

ĐỊA CHỈ HIỆU DỤNG

Một số thí dụ

Địa chỉ hiệu dụng hợp lệ :

[1000h] [SI], [DI] , [BX] , [BP]

[SI+BX], [SI+BP] , [DI+BX] , [DI+BP] , [SI+1000h], [DI+100h]

[SI] [BX] [1000h], [SI+BP+1000h] , [DI+BX][1000h],
[DI+1000h]+[BP]

Địa chỉ hiệu dụng không hợp lệ :

[70000], [AX] , [SI+DI+1000h], [BX] [BP]

Địa chỉ hiệu dụng (tt)

Qui ước

Để thuận tiện trong vấn đề giải thích lệnh, ta qui ước sau :

Dữ liệu 8 bit bộ nhớ : [địa chỉ]

Dữ liệu 16 bit bộ nhớ : [địa chỉ +1, địa chỉ]

Để xác định rõ hoạt động của bộ nhớ, ta phải dùng thêm toán tử PTR như sau :

8 bit : BYTE PTR [1000H]

Tham khảo 1 byte bộ nhớ ở địa chỉ 1000h

16 bit : WORD PTR [1000H]

Tham khảo 2 byte bộ nhớ liên tiếp ở địa chỉ 1000h và 1001h

Ex : Tính tổng 1 array có 5 phần tử

```
MOV BX, OFFSET LIST  
    MOV AX, 0  
    MOV AL, [BX]  
    ADD AL , [BX+1]  
    ADD AL , [BX+2]  
    ADD AL , [BX+3]  
    ADD AL , [BX+4]  
    MOV SUM , AX
```

.....

```
LIST DB 10h, 20h, 40h, 2h, 5h  
SUM DW 0
```

Cách thực hiện :

Lấy địa chỉ của List vào BX

Dựa vào BX để xác định các phần tử của array.

Khi tính tổng xong, đưa tổng vào biến SUM.

CHẠY CT này bằng DEBUG

Ex : Tính tổng 1 array có 5 phần tử

```
-A 100  
MOV BX, 0120  
MOV AX, 0  
MOV AL, [BX]  
ADD AL , [BX+1]  
ADD AL , [BX+2]  
ADD AL , [BX+3]  
ADD AL , [BX+4]  
MOV [0125], AX  
-A 120  
DB 10, 20, 40, 2, 5  
DW 0
```

Tập lệnh

Lệnh **MOV** :

Ý nghĩa : copy giá trị từ toán hạng nguồn → toán hạng đích

Cú pháp : **MOV dest , source**

Yêu cầu : Dest và source cùng kiểu

Dạng lệnh :

MOV reg , reg

MOV mem , reg

MOV reg, mem

MOV reg16, segreg

MOV segreg, reg16

MOV reg, immed

MOV mem, immed

MOV mem16, segreg

MOV segreg, mem16

Minh họa lệnh MOV

MOV AX, CX

MOV DL, BH

MOV [SI+1000h], BP ; [SI+1000h, SI+1001h] ← BP

MOV DX, [1000h] ; DX ← [1000h, 1001h]

MOV DI, 12h

MOV AL, 12h

MOV BYTE PTR [1000h], 12h

MOV WORD PTR [2000h] , 1200h

MOV [BX] , DS

MOV SS, [2000h]

Chú ý

- Lệnh MOV không làm ảnh hưởng đến cờ.
- Không thể chuyển dữ liệu trực tiếp giữa 2 toán hạng bộ nhớ với nhau, muốn chuyển phải dùng thanh ghi trung gian.
- Không thể chuyển 1 giá trị tức thời vào thanh ghi đoạn, muốn chuyển phải dùng thanh ghi trung gian.
- Không thể chuyển trực tiếp giữa 2 thanh ghi đoạn

Minh họa lệnh MOV

Ex1 : Cho table là 1 mảng gồm 10 phần tử dạng byte

Table DB 3,5,6,9,10, 29,30,46,45,90

Truy xuất phần tử đầu , phần tử thứ 2 và thứ 5 của mảng:

MOV AL, TABLE hay MOV AL, TABLE[0]

MOV AL, TABLE+1 hay MOV AL, TABLE[1]

MOV AL, TABLE+4 hay MOV AL, TABLE[4]

Minh họa lệnh MOV

Ex2 : MOV AX, DS : [100h]

; chép nội dung 16 bit tại địa chỉ
100h trong đoạn chỉ bởi DS vào Reg AX.

Ex3 : MOV AX, [100h]
; chuyển NỘI DUNG Ở NHỚ 100h vào Reg AX

Áp dụng

Viết chương trình chuyển nội dung vùng nhớ bắt đầu tại địa chỉ 70 sang vùng nhớ có địa chỉ bắt đầu là 1000h. Biết chiều mỗi vùng nhớ là 9 bytes và dữ liệu đang khảo sát trong đoạn được chỉ bởi D

Cho vùng nhớ MEM có chiều dài 9 bytes gồm các ký tự ‘abcdefghi’ trong đoạn chỉ bởi DS.

Viết chương trình đảo ngược vùng nhớ MEM.

Lệnh LEA (Load Effective Address)

Cú pháp : LEA REG | MEM

ý nghĩa : nạp địa chỉ Offset vào thanh ghi để khởi động Reg.

Ex : MOV DX, OFFSET MES

Tương đương với LEA DX, MES

Ex : LEA BX, [1000h] ; BX \leftarrow 1000h

LEA SI, [DI][BX][2000h] ; SI \leftarrow DI + BX + 2000h

Lệnh XCHG (XCHANGE)

Cú pháp : XCHG DEST , SOURCE
ý nghĩa : hoán chuyển nội dung 2 Reg, Reg và ô nhớ
Yêu cầu :
 2 toán hạng phải cùng kiểu
 2 toán hạng không thể là 2 biến bộ nhớ. Muốn hoán đổi trị của 2 biến phải dùng Reg trung gian.

Ex : XCHG AH, BL

MOV VAR1, VAR2 ; không hợp lệ, phải dùng Reg tạm

Lệnh PUSH

Cú pháp : PUSH REG16

PUSH MEM16

PUSH SEGREG

Đẩy toán hạng nguồn 16 bit vào STACK

Ex : PUSH DI ; [SS :SP+1, SS :SP] ← DI

Ex : PUSH CS ; [SS :SP+1, SS :SP] ← CS

Lệnh POP

Cú pháp : POP REG16

POP MEM16

POP SEGREG

Lấy dữ liệu từ đỉnh STACK vào toán hạng đích.

Ex : POP AX ; AX ← [SS :SP+1, SS :SP]

Ex : POP [BX+1] ; [BX+2, BX+1] ← [SS :SP+1, SS :SP]

Lệnh IN

Cú pháp : IN ACCUM, IMMED8
IN ACCUM, DX

nhập dữ liệu từ cổng xuất nhập vào thanh ghi tích luỹ AL hay AX. Trường hợp AX sẽ nhập byte thấp trước, byte cao sau.

Ex : IN AL ,61h

IN AX, 40h

Ex : MOV DX, 378H

IN AL, DX

Dạng lệnh có Reg DX dùng
Để cho cổng có địa chỉ 16 bit

SUMMARY

- Dùng DEBUG để hợp dịch và chạy chương trình sau : Chép 3 số nguyên kiểu Word ở địa chỉ 0120h vào địa chỉ 0130h.
- Cho biết giá trị của AX sau khi các lệnh sau được thực thi :

```
MOV AX, ARRAY1  
INC AX  
ADD AH, 1  
SUB AX, ARRAY1  
.....  
ARRAY1 DW 10h, 20h
```

SUMMARY

- Giả sử biến VAL1 ở địa chỉ offset 0120h và PTR1 ở địa chỉ 0122h. Cho biết giá trị của các thanh ghi AX, BX khi mỗi lệnh sau được thực thi :

```
.CODE
    MOV AX, @DATA
    MOV DS, AX
    MOV AX, 0
    MOV AL, BYTE PTR VAL1 ; AX = ?
    MOV BX, PTR1           ; BX = ?
    XCHG AX, BX            ; BX = ?
    SUB AL,2               ; AX = ?
    MOV AX, PTR2           ; AX = ?

.DATA
    VAL1 DW 3Ah
    PTR1 DW VAL1
    PTR2 DW PTR1
```

Cho biết giá trị của các thanh ghi ở bên phải, khi mỗi lệnh của đoạn chương trình sau được thực thi. Giả sử FIRST ở offset 0H

MOV AL, BYTE PTR FIRST+1 ; AL =

MOV BX, WORD PTR SECOND+2 ; BX =

MOV DX, OFFSET FIRST + 2 ; DX =

MOV AX, 4C00H

INT 21H

.....

FIRST DW 1234h

SECOND DW 16385

THIRD DB 10,20,30,40

Bài tập Lập trình

Bài 1 : Viết chương trình nhập 1 ký tự.

Hiển thị ký tự đứng trước và ký tự đứng sau ký tự đã nhập theo thứ tự mã ASCII.

Kết quả có dạng :

Nhập một ký tự : B

Ký tự đứng trước : A

Ký tự đứng sau : C

Bài 2 : Viết chương trình nhập 2 ký tự và hiển thị ký tự thứ 3 có mã ASCII là tổng của mã 2 ký tự đã nhập.

Kết quả có dạng :

Chương 8 : Cấu trúc điều khiển và Vòng lặp

Mục tiêu

- Biết cách mô phỏng cấu trúc điều khiển và vòng lặp như ở ngôn ngữ lập trình cấp cao.
- Nắm được các lệnh nhảy trong lập trình Assembly.
- Trên cơ sở đó, vận dụng để lập trình giải quyết 1 số bài toán.

Nội dung

- ✓ Sự cần thiết của lệnh nhảy trong lập trình ASM.
- ✓ Lệnh JMP (Jump) : nhảy không điều kiện.
- ✓ Lệnh LOOP : cho phép lặp 1 công việc với 1 số lần nào đó.
- ✓ Các lệnh so sánh và luận lý.
- ✓ Lệnh lặp có điều kiện.
- ✓ Lệnh nhảy có điều kiện.
- ✓ Biểu diễn mô phỏng cấu trúc luận lý mức cao.
- ✓ Chương trình con.
- ✓ Một số chương trình minh họa.

Sự cần thiết của lệnh nhảy

- Ở các chương trình viết bằng ngôn ngữ cấp cao thì việc nhảy (lệnh GoTo) là điều nên tránh nhưng ở lập trình hệ thống thì đây là việc cần thiết và là điểm mạnh của 1 chương trình viết bằng Assembly.
- Một lệnh nhảy → CPU phải thực thi 1 đoạn lệnh ở 1 chỗ khác với nơi mà các lệnh đang được thực thi.
- Trong lập trình, có những nhóm phát biểu cần phải lặp đi lặp lại nhiều lần trong 1 điều kiện nào đó. Để đáp ứng điều kiện này ASM cung cấp 2 lệnh JMP và LOOP.

Lệnh JMP (Jump)

■ Công dụng : Chuyển điều khiển không điều kiện
Cú pháp : JMP đích

Nhảy gần (NEAR) : 1 tác vụ nhảy trong cùng 1 segment.

Nhảy xa (FAR) : 1 tác vụ nhảy sang segment khác.

Các lệnh chuyển điều khiển

Chuyển điều khiển vô điều kiện

JMP [SORT | NEAR PTR | FAR PTR] DEST

Chuyển điều khiển có điều kiện

JConditional destination

Ex : JNZ nhãn đích ;

LỆNH LOOP

Công dụng : cho phép lặp 1 công việc với 1 số lần nào đó.

Mỗi lần lặp CX giảm đi 1 đơn vị. Vòng lặp chấm dứt khi CX =0.

Ex 1 : xuất ra màn hình 12 dòng gồm các ký tự A.

MOV CX, 12 * 80

MOV DL, ‘A’

NEXT :

MOV AH, 2

INT 21H

LOOP NEXT

LOOP (tt)

Ex : có 1 Array A gồm 6 bytes, chép A sang array B – dùng SI và DI để lấy Offset

```
MOV SI, OFFSET A
MOV DI, OFFSET B
MOV CX, 6
MOVE_BYTE :
    MOV AL, [SI]
    MOV [DI], AL
    INC SI
    INC DI
LOOP MOVE_BYTE
A DB 10H,20H,30H,40H,50H,60H
B DB 6 DUP (?)
```

CÁC LỆNH LUẬN LÝ

Lưu ý về các toán tử LOGIC :

AND 2 Bit : kết quả là 1 khi và chỉ khi 2 bit là 1

OR 2 Bit : kết quả là 1 khi 2 Bit có bit là 1

XOR 2 Bit : kết quả là 1 chỉ khi 2 bit khác nhau

NOT 1 Bit : lấy đảo của Bit này

Lưu ý về thanh ghi cờ :

Cờ ZERO được lập khi tác vụ cho kết quả là 0.

Cờ CARRY được lập khi cộng kết quả bị tràn hay trừ phải mượn.

Cờ SIGN được lập khi bit dấu của kết quả là 1, tức kết quả là số âm.

Lệnh AND

Cú pháp : AND Destination , Source

Công dụng :

Lệnh này thực hiện phép AND giữa 2 toán hạng, kết quả cuối cùng chứa trong toán hạng đích.

Dùng để xóa các bit nhất định của toán hạng đích giữ nguyên các bit còn lại.

Muốn vậy ta dùng 1 mẫu bit gọi là mặt nạ bit (MASK), các bit mặt nạ được chọn để sao cho các bit tương ứng của đích được thay đổi như mong muốn.

Lệnh AND

Ex1 : xoá bit dấu của AL, giữ nguyên các bit còn lại :
dùng AND với **0111111b** làm mặt nạ
AND AL, 7FH

Ex2 :
MOV AL, ‘5’ ; Đổi mã ASCII của số
AND AL, 0FH ; thành số tương ứng.

Ex3 :
MOV DL, ‘a’ ; Đổi chữ thường thành chữ hoa.
AND DL, 0DFH ; thành số tương ứng.



LỆNH OR

Công dụng : dùng để bật lên 1 số bit và giữ nguyên các bit khác.

Cú pháp : OR destination, source

Ex1 :

OR AL , 10000001b ; bật bit cao nhất và bit thấp nhất trong thanh ghi AL lên 1

Ex 2:

MOV AL , 5 ; đổi 0..9 thành ký số

OR AL , 30h ; ASCII tương ứng.

Ex 3:

OR AL , AL ; kiểm tra một thanh ghi có = 0.

Nếu : cờ ZF được lập \rightarrow AL =0

cờ SIGN được lập \rightarrow AL <0

cờ ZR và cờ SIGN không được lập \rightarrow AL >0

LỆNH XOR

Công dụng : dùng để tạo đồ họa màu tốc độ cao.

Cú pháp : XOR destination, source

Ex : lật bit cao của AL 2 lần

MOV AL , 00111011b ;

XOR AL, 11111111b ; AL = **1**1000100b

XOR AL, 11111111b ; AL = **0**0111011b

LỆNH TEST

Cú pháp : TEST destination, source

Công dụng : dùng để khảo sát trị của từng bit hay nhóm bit.

Test thực hiện giống lệnh AND nhưng không làm thay đổi toán hạng đích.

Ex : kiểm tra bit 13 trong DX là 0 hay 1

TEST DX, 2000h

JZ BitIs0

BitIs1 : bit 13 is 1

BitIs0 : bit 13 is 0

Để kiểm tra 1 bit nào đó chỉ cần đặt bit 1 vào đúng vị trí bit cần kiểm tra và khảo sát cờ ZF.
(nếu bit kiểm là 1 thì ZF sẽ xoá, ngược lại ZF được lập.)

MINH HỌA LỆNH TEST

Ex : kiểm tra trạng thái máy in. Interrupt 17H trong BIOS sẽ kiểm tra trạng thái máy in, sau khi kiểm tra AL sẽ chứa trạng thái máy in. Khi bit 5 của AL là 1 thì máy in hết giấy.

MOV AH, 2

INT 17h

TEST AL , 00100000b ; Test bit 5, nếu bit 5 = 1 ➔ máy in hết giấy.

Lệnh TEST cho phép test nhiều bit 1 lượt.

MINH HỌA LỆNH TEST(tt)

Ex :viết đoạn lệnh thực hiện lệnh nhảy đến nhãn A1 nếu AL chứa số chẵn.

TEST AL, 1 ; AL chứa số chẵn ?

JZ A1 ; nếu đúng nhảy đến A1.

Lệnh CMP

Cú pháp : CMP destination , source

Công dụng : so sánh toán hạng đích với toán hạng nguồn
bằng cách lấy toán hạng đích – toán hạng nguồn.

Hoạt động : dùng phép trừ nhưng không có toán hạng
đích nào bị thay đổi.

Các toán hạng của lệnh CMP không thể cùng là các ô nhớ.

lệnh CMP giống hệt lệnh SUB trừ việc toán hạng đích không thay đổi.

LỆNH NHẢY CÓ ĐIỀU KIỆN

Cú pháp : Jconditional destination

Công dụng : nhờ các lệnh nhảy có điều kiện, ta mới mô phỏng được các phát biểu có cấu trúc của ngôn ngữ cấp cao bằng Assembly.

Phạm vi

- Chỉ nhảy đến nhãn có khoảng cách từ -128 đến +127 byte so với vị trí hiện hành.
- Dùng các trạng thái cờ để quyết định có nhảy hay không?

LỆNH NHẢY CÓ ĐIỀU KIỆN

Hoạt động

- để thực hiện 1 lệnh nhảy CPU nhìn vào các thanh ghi cờ.
- nếu điều kiện của lệnh nhảy thỏa, CPU sẽ điều chỉnh IP trả đến nhãn đích các lệnh sau nhãn này sẽ được thực hiện.

.....

MOV AH, 2	PRINT_LOOP :
MOV CX, 26	INT 21H
MOV DL, 41H	INC DL
	DEC CX
	JNZ PRINT_LOOP
	MOV AX, 4C00H
	INT 21H

LỆNH NHẢY DỰA TRÊN KẾT QUẢ SO SÁNH CÁC TOÁN HẠNG KHÔNG DẤU.

Thường dùng lệnh CMP Opt1 , Opt2 để xét điều kiện nhảy hoặc dựa trên các cờ.

JZ	Nhảy nếu kết quả so sánh = 0
JE	Nhảy nếu 2 toán hạng bằng nhau
JNZ	Nhảy nếu kết quả so sánh là khác nhau.
JNE	Nhảy nếu 2 toán hạng khác nhau.
JA	Nhảy nếu Opt1 > Opt2
JNBE	Nhảy nếu Opt1 <= Opt2
JAE	Nhảy nếu Opt1 >= Opt2
EQU	Xác định biến bằng giá trị (0, 1, 0, 0)

LỆNH NHẢY DỰA TRÊN KẾT QUẢ SO SÁNH CÁC TOÁN HẠNG KHÔNG DẤU (ctn) .

JNC	Nhảy nếu không có Carry.
JB	Nhảy nếu Opt1 < Opt2
JNAE	Nhảy nếu Not(Opt1 >= Opt2)
JC	Nhảy nếu có Carry
JBE	Nhảy nếu Opt1<=Opt2
JNA	Nhảy nếu Not (Opt1 > Opt2)

LỆNH NHẢY DỰA TRÊN KẾT QUẢ SO SÁNH CÁC TOÁN HẠNG CÓ DẤU .

JG	Nhảy nếu Opt1>Opt2
JNLE	Nhảy nếu Not(Opt1 <= Opt2)
JGE	Nhảy nếu Opt1>=Opt2
JNL	Nhảy nếu Not (Opt1 < Opt2)
JL	Nhảy nếu Opt1 < Opt2
JNGE	Nhảy nếu Not (Opt1 >= Opt2)
JLE	Nhảy nếu Opt1 <= Opt2
JNG	Nhảy nếu Not (Opt1 > Opt2)

LỆNH NHẢY DỰA TRÊN CÁC CỜ .

JCXZ	Nhảy nếu CX=0
JS	Nhảy nếu SF=1
JNS	Nhảy nếu SF =0
JO	Nhảy nếu đã tràn trị
JL	Nhảy nếu Opt1 < Opt2
JNGE	Nhảy nếu Not (Opt1 >= Opt2)
JLE	Nhảy nếu Opt1 <= Opt2
JNO	Nhảy nếu tràn trị
JP	Nhảy nếu parity chẵn
JNP	Nhảy nếu PF =0

CÁC VỊ DỤ MINH HỌA LỆNH NHẢY CÓ ĐK

Ex1 : tìm số lớn hơn trong 2 số
chứa trong thanh ghi AX và BX .

Kết quả để trong DX

```
MOV DX, AX          ; giả sử AX là số lớn hơn.  
CMP DX, BX          ; IF AX >=BX then  
JAE QUIT            ; nhảy đến QUIT  
MOV DX, BX          ; ngược lại chép BX vào DX  
QUIT :  
    MOV AH,4CH  
    INT 21H  
.....
```

CÁC VÍ DỤ MINH HỌA LỆNH NHẢY CÓ ĐK

Ex1 : tìm số nhỏ nhất trong 3 số chứa trong thanh ghi AL BL và CL . Kết quả để trong biến SMALL

```
MOV SMALL, AL  
CMP SMALL, BL  
JBE L1  
MOV SMALL, BL  
L1 :  
    CMP SMALL, CL  
    JBE L2  
    MOV SMALL, CL  
L2 : ...
```

; giả sử AL nhỏ nhất
; nếu SMALL <= BL thì
Nhảy đến L1
; nếu SMALL <= CL thì
; Nhảy đến L2
; CL là số nhỏ nhất

Các lệnh dịch và quay bit

**SHL (Shift Left) : dịch các bit của toán hạng
dịch sang trái**

Cú pháp : SHL toán hạng dịch ,1

Dịch 1 vị trí.

Cú pháp : SHL toán hạng dịch ,CL

Dịch n vị trí trong đó CL chứa số bit cần dịch.

Hoạt động : một giá trị 0 sẽ được đưa vào vị trí
bên phải nhất của toán hạng đích, còn bit msb
của nó được đưa vào cờ CF

Các lệnh dịch và quay bit

Ex : DH chứa 8Ah, CL chứa 3.

SHL DH, CL ; 01010000b

? Cho biết kết quả của :

SHL 1111b, 3

**MT thực hiện phép nhân bằng
dịch trái**

lệnh dịch phải SHR

Công dụng : dịch các bit của toán hạng đích sang bên phải.

Cú pháp : **SHR toán hạng đích , 1**

SHR toán hạng đích , CL ; dịch phải n bit trong đó CL chứa n

Hoạt động : 1 giá trị 0 sẽ được đưa vào bit msb của toán hạng đích, còn bit bên phải nhất sẽ được đưa vào cờ CF.

**MT thực hiện phép chia bằng
dịch phải**

lệnh dịch phải SHR

Ex : shr 0100b, 1 ; 0010b = 2

Đối với các số lẻ, dịch phải sẽ chia đôi nó và làm tròn xuống số nguyên gần nhất.

Ex : shr 0101b, 1 ; 0010b = 2

Chương trình con

Có vai trò giống như chương trình con ở ngôn ngữ cấp cao.

ASM có 2 dạng chương trình con : dạng FAR và dạng NEAR.

Lệnh gọi CTC
nằm cùng đoạn
bộ nhớ với CTC
được gọi

Lệnh gọi CTC
nằm khác đoạn
bộ nhớ với CTC
được gọi

BIỂU DIỄN CẤU TRÚC LOGIC MỨC CAO

Dù Assembly không có phát biểu IF, ELSE, WHILE, REPEAT, UNTIL, FOR, CASE nhưng ta vẫn có thể tổ hợp các lệnh của Assembly để hiện thực cấu trúc logic của ngôn ngữ cấp cao.

Cấu trúc IF Đơn giản

Phát biểu IF sẽ kiểm tra 1 điều kiện và theo sau đó là 1 số các phát biểu được thực thi khi điều kiện kiểm tra có giá trị true.

Cấu trúc logic

```
IF (OP1=OP2)
    <STATEMENT1>
    <STATEMENT2>
ENDIF
```

HIỆN THỰC BẰNG ASM

```
CMP OP1,OP2
JNE CONTINUE
<STATEMENT1>
<STATEMENT2>
CONTINUE : ....
```

Cấu trúc IF với OR

Phát biểu IF có kèm toán tử OR

Cấu trúc logic

```
IF(A1>OP1) OR  
(A1>=OP2) OR  
(A1=OP3) OR  
(A1<OP4)  
<STATEMENT>  
ENDIF
```

HIỆN THỰC BẰNG ASM

```
CMP A1,OP1  
JG EXECUTE  
CMP A1,OP2  
JGE EXECUTE  
CMP A1,OP3  
JE EXECUTE  
CMP A1,OP4  
JL EXECUTE  
JMP CONTINUE  
EXECUTE : <STATEMENT>  
CONTINUE : .....
```

Cấu trúc IF với AND

Phát biểu IF có kèm toán tử AND

Cấu trúc logic

```
IF (A1>OP1) AND  
(A1>=OP2) AND  
(A1=OP3) AND  
(A1<OP4)  
<STATEMENT>  
ENDIF
```

HIỆN THỰC BẰNG ASM

```
CMP A1,OP1  
JNG CONTINUE  
CMP A1,OP2  
JL CONTINUE  
CMP A1,OP3  
JNE CONTINUE  
CMP A1,OP4  
JNL CONTINUE  
<STATEMENT>  
JMP CONTINUE  
CONTINUE : .....
```

CHÚ Ý : khi điều kiện có toán tử AND, cách hay nhất là dùng nhảy với điều kiện ngược lại đến nhãn, bỏ qua phát biểu trong cấu trúc Logic.

Cấu trúc WHILE

VÒNG LẶP WHILE

Cấu trúc logic

DO WHILE (OP1<OP2)
<STATEMENT1>
<STATEMENT2>
ENDDO

HIỆN THỰC BẰNG ASM
DO_WHILE :
 CMP OP1, OP2
 JNL ENDDO
 <STATEMENT1>
 <STATEMENT2>
 JMP DO_WHILE
ENDDO :

Cấu trúc WHILE có lồng IF

Cấu trúc logic

```
DO WHILE (OP1<OP2)
<STATEMENT>
IF (OP2=OP3) THEN
<STATEMENT2>
<STATEMENT3>
ENDIF
ENDDO
```

VÒNG LẶP WHILE CÓ LỒNG IF

HIỆN THỰC BẰNG ASM

_WHILE :

```
CMP OP1, OP2
JNL WHILE_EXIT
<STATEMENT1>
CMP OP2,OP3 ; phần If
JNE ELSE ; không thỏa If
<STATEMENT2> ; thỏa If
<STATEMENT3>
JMP ENDIF; thỏa If nên
            bỏ qua Else
ELSE : <STATEMENT4>
ENDIF : JMP _WHILE
WHILE_EXIT : .....
```

Cấu trúc REPEAT UNTIL

Cấu trúc logic

REPEAT

<STATEMENT1>

<STATEMENT2>

<STATEMENT3>

**UNTIL (OP1=OP2) OR
(OP1>OP3)**

Bằng nhau
thoát
Repeat

VÒNG LẶP REPEAT UNTIL

HIỆN THỰC BẰNG ASM
REPEAT :

<STATEMENT1>

<STATEMENT2>

<STATEMENT3>

TESTOP12:

CMP OP1, OP2

JE ENDREPEAT

TESTOP13 :

CMP OP1, OP3

JNG REPEAT

ENDREPEAT :

Cấu trúc CASE

Cấu trúc logic
CASE INPUT OF
‘A’ : Proc_A
‘B’ : Proc_B
‘C’ : Proc_C
‘D’ : Proc_D
End ;

HIỆN THỰC BẰNG ASM

CASE : MOV AL, INPUT

CMP AL, ‘A’

JNE TESTB

CALL PROC_A

JMP ENDCASE

TESTB :

CMP AL, ‘B’

JNE TESTC

CALL PROC_B

JMP ENDCASE

TESTC :

CMP AL, ‘C’

JNE TESTD

CALL PROC_C

JMP ENDCASE

TESTD : CMP AL, ‘D’

JNE ENDCASE

CALL PROC_D

ENDCASE :

LooKup Table

Rất hiệu quả khi xử lý phát biểu CASE là dùng bảng OFFSET chứa địa chỉ của nhãn hoặc của hàm sẽ nhảy đến tùy vào điều kiện.

Bảng Offset này được gọi Lookup Table rất hiệu quả khi dùng phát biểu Case có nhiều trị lựa chọn.

LooKup Table

Case_table db ‘A’

; giá trị tìm kiếm
Địa chỉ các procedure
giả sử ở địa chỉ 0120

Db ‘B’

giả sử ở địa chỉ 0130

Db ‘C’

giả sử ở địa chỉ 0140

Dw Proc_C

giả sử ở địa chỉ 0150

Db ‘D’

Dw Proc_D

‘A’	0120	‘B’	0130	‘C’	0140	‘D’	0150
-----	------	-----	------	-----	------	-----	------

Cấu trúc lưu trữ của CaseTable như sau

LooKup Table

Case :

MOV AL, INPUT

MOV BX, OFFSET CASE_TABLE

MOV CX, 4 ; lặp 4 lần số entry của table

TEST :

CMP AL, [BX] ; kiểm tra Input

JNE TESTAGAIN ; không thỏa kiểm tra tiếp

CALL WORD PTR [BX+1] ; gọi thủ tục tương ứng

JMP ENDCASE

TESTAGAIN : ADD BX , 3 ; sang entry sau của CaseTable

LOOP TEST

ENDCASE :

Chương trình con

Cấu trúc CTC :

```
TênCTC PROC <Type>
    ; các lệnh
    RET
TênCTC ENDP
```

CTC có thể gọi 1 CTC khác hoặc gọi chính nó.

CTC được gọi bằng lệnh CALL <TenCTC>.

CTC gần (near) là chương trình con nằm chung segment với nơi gọi nó.

CTC xa (far) là chương trình con không nằm chung segment với nơi gọi nó.

Kỹ thuật lập trình

- Hãy tổ chức chương trình → các chương trình con
→ đơn giản hóa cấu trúc luận lý của CT làm cho CT
dễ đọc, dễ hiểu , dễ kiểm tra sai sót..
- Đầu CTC hãy cất trị thanh ghi vào Stack bằng
lệnh PUSH để lưu trạng thái hiện hành.
- Sau khi hoàn tất công việc của CTC nên phục hồi
lại trị các thanh ghi lúc trước đã Push bằng lệnh
POP .
 - Nhớ trình tự là ngược nhau để trị của thanh ghi
nào trả cho thanh ghi này.
- Đừng tối ưu quá CT vì có thể làm cho CT kém
thông minh, khó đọc.

Kỹ thuật lập trình (tt)

- Cố gắng tổ chức chương trình cho tốt → phải thiết kế được các bước chương trình sẽ phải thực hiện.
- Kinh nghiệm : khi vấn đề càng lớn thì càng phải tổ chức logic chương trình càng chặt chẽ.
- Bằng sự tổ hợp của lệnh nhảy ta hoàn toàn có thể mô phỏng cấu trúc điều khiển và vòng lặp.

SUMMARY

- ✓ Có thể mô phỏng cấu trúc logic như ngôn ngữ cấp cao trong Assembly bằng lệnh JMP và LOOP.
- ✓ các lệnh nhảy : có điều kiện và vô điều kiện.
- ✓ Khi gặp lệnh nhảy, CPU sẽ quyết định nhảy hay không bằng cách dựa vào giá trị thanh ghi cờ.
- ✓ các lệnh luận lý dùng để làm điều kiện nhảy là AND, OR, XOR, CMP ...
- ✓ Bất cứ khi nào có thể, hãy tổ chức chương trình thành các chương trình con ➔ đơn giản được cấu trúc luận lý của chương trình.

Câu hỏi

1. Giả sử DI = 2000H, [DS:2000] = 0200H. Cho biết địa chỉ ô nhớ toán hạng nguồn và kết quả lưu trong toán hạng đích khi thực hiện lệnh MOV DI, [DI]
2. Giả sử SI = 1500H, DI=2000H, [DS:2000]=0150H . Cho biết địa chỉ ô nhớ toán hạng nguồn và kết quả lưu trong toán hạng đích sau khi thực hiện lệnh ADD AX, [DI]
3. Có khai báo A DB 1,2,3
Cho biết trị của toán hạng đích sau khi thi hành lệnh MOV AH, BYTE PTR A.
4. Có khai báo B DB 4,5,6
Cho biết trị của toán hạng đích sau khi thi hành lệnh MOV AX, WORD PTR B.

Bài tập LẬP TRÌNH

Bài 1 : Có vùng nhớ VAR1 dài 200 bytes trong đoạn được chỉ bởi DS.

Viết chương trình đếm số chữ ‘S’ trong vùng nhớ này.

Bài 2 : Có vùng nhớ VAR2 dài 1000 bytes. Viết chương trình chuyển đổi các chữ thường trong vùng nhớ này thành các ký tự hoa, các ký tự còn lại không đổi.

Bài 3 : Viết chương trình nhập 2 số nhỏ hơn 10.

In ra tổng của 2 số đó.

Bài tập LẬP TRÌNH

Bài 4 : Viết chương trình nhập 2 số bất kỳ.

In ra tổng và tích của 2 số đó. Chương trình có dạng sau :

Nhập số 1 : 12

Nhập số 2 : 28

Tổng là : 40

Tích là : 336

Bài 5 : Viết chương trình nhập 1 ký tự. Hiển thị 5 ký tự kế tiếp trong bộ mã ASCII.

Ex : nhập ký tự : a

5 ký tự kế tiếp : b c d e f

Bài tập LẬP TRÌNH

Bài 6 : Viết chương trình nhập 1 ký tự. Hiển thị 5 ký tự đứng trước trong bộ mã ASCII.

Ex : nhập ký tự : f

5 ký tự kế tiếp : a b c d e

Bài 7 : Viết chương trình nhập 1 chuỗi ký tự.

In chuỗi đã nhập theo thứ tự ngược.

Ex : nhập ký tự : abcdef

5 ký tự kế tiếp : fedcba

MACRO

- **Định nghĩa Macro và gọi Macro**
- **Vấn đề truyền thông số trong Macro.**
- **Macro lồng nhau.**
- **Sử dụng Macro để gọi chương trình con.**
- **Các toán tử Macro.**
- **Thư viện Macro**
- **So sánh việc dùng Macro với Procedure**
- **Một số Macro mẫu.**

DỊNH NGHĨA MACRO

- Macro là 1 ký hiệu được gán cho 1 nhóm lệnh ASM – Macro là tên thay thế cho 1 nhóm lệnh.



Tại sao cần có Macro :

- Trong lập trình nhiều lúc ta cần phải viết những lệnh na ná nhau nhiều lần mà ta không muốn viết dưới dạng hàm vì dùng hàm tốn thời gian thực thi, thay vì ta phải viết đầy đủ nhóm lệnh này vào CT, ta chỉ cần viết Macro mà ta đã gán cho chúng.

LÀM QUEN VỚI MACRO

Khi ta có nhiều đoạn code giống nhau, chúng ta có thể dùng macro để thay thế, giống như ta dùng define trong C. Thí dụ chúng ta thay thế đoạn lệnh sau bằng macro để in dấu xuống dòng.

MOV DL,13 ; về đầu dòng

MOV AH,2

INT 21H

MOV DL,10 ; xuống dòng mới

MOV AH,2

INT 21H

Thay vì phải viết lại 6 dòng lệnh trên, ta có thể tạo 1 macro có tên @Newline để thay thế đoạn code này :

@NewLine Macro

MOV DL,13

MOV AH,2

INT 21H

MOV DL,10

MOV AH,2

INT 21H

ENDM

Sau đó, bất kỳ chỗ nào cần xuống dòng, ta chỉ cần gọi macro @NewLine.

@NewLine



MACRO (tt)

- Khi hợp dịch nội dung nhóm lệnh này mà ta đã gán cho macro sẽ được thay thế vào những nơi có tên macro trước khi CT được hợp dịch thành file OBI.
 - Ex1 : nhiều khi ta phải viết lại nhiều lần đoạn lệnh xuất ký tự trong DL ra màn hình.

■ MOV AH, 2

■ INT 21H

- Thay vì phải viết cả 1 cặp lệnh trên mỗi khi cần xuất ký tự trong DL, ta có thể viết Macro **PUTCHAR** như sau :

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

ENDM

CHUONG 9 MACRO



- MỞ RỘNG CỦA MACRO CÓ THỂ XEM TRONG FILE.LIST.
- 3 DIRECTIVE BIÊN DỊCH SAU SẼ QUYẾT ĐỊNH MỞ RỘNG MACRO NHƯ THẾ NÀO.
- .SALL (SUPPRESS ALL) PHẦN MỞ RỘNG MACRO KHÔNG ĐƯỢC IN. SỬ DỤNG KHI MACRO LỚN HAY MACRO ĐƯỢC THAM CHIỀU NHIỀU LẦN TRONG CT.
- .XALL CHỈ NHỮNG DÒNG MACRO TẠO MÃ NGUỒN MỚI ĐƯỢC IN RA. THÍ ĐỰ CÁC DÒNG CHÚ THÍCH ĐƯỢC BỎ QUA. ĐÂY LÀ TUỲ CHỌN DEFAULT.
- .LALL (LIST ALL) TOÀN BỘ CÁC DÒNG TRONG MACRO ĐƯỢC IN RA TRỪ NHỮNG CHÚ THÍCH BẮT ĐẦU BẰNG 2 DẤU ;;

ĐỊNH NGHĨA MACRO

■ CÚ PHÁP KHAI BÁO MACRO :

**MACRO_NAME MACRO [<THÔNG SỐ HÌNH THỨC>]
STATEMENTS**

ENDM

■ GỌI MACRO :

MACRO_NAME [<THÔNG SỐ THỰC>, ...]

THÔNG SỐ HÌNH THỨC CHỈ CÓ TÁC DỤNG ĐÁNH DẤU VỊ TRÍ
CỦA THÔNG SỐ TRONG MACRO. QUAN TRỌNG NHẤT LÀ VỊ TRÍ
CÁC THÔNG SỐ.

MACRO TRUYỀN THAM SỐ

.MODEL SMALL

.STACK 100H

PUTCHAR MACRO KT

MOV DL,KT

MOV AH,2

INT 21H

ENDM

.CODE

MAIN PROC

MOV DL, 'A'

PUTCHAR

MOV DL, "*"

PUTCHAR

MOV AH,4CH

INT 21H

MAIN ENDP

END MAIN



SWAP MACRO BIẾN1, BIẾN2

MOV AX, BIEN1

XCHG AX, BIEN2

MOV BIEN1, AX

ENDM

GOI : SWAP TRI1, TRI2

TRAO ĐỔI THAM SỐ CỦA MACRO

MỘT MACRO CÓ THỂ CÓ THÔNG SỐ HOẶC KHÔNG CÓ THÔNG SỐ.

MACRO CÓ THÔNG SỐ

SỬ DỤNG MACRO

PUTCHAR MACRO CHAR

MOV AH, 2

MOV DL, CHAR

INT 21H

ENDM

.CODE

... ...

PUTCHAR 'A'

PUTCHAR 'B'

PUTCHAR 'C'

...

MACRO TRUYỀN THÔNG SỐ

Thí dụ : macro @Printstr

Viết chương trình in 2 chuỗi ‘Hello’ và ‘Hi’.

.DATA

MSG1 DB ‘Hello’,13,10

MSG2 DB‘Hi’,13,10

.CODE

.....

MOV DX, OFFSET MSG1 ;1

MOV AH,9 ;1

INT 21H ;1

MOV DX, OFFSET MSG2 ;2

MOV AH,9 ;2

INT 21H ;2

.....

Ta thấy đoạn 1
và đoạn 2 gần
giống nhau →
có thể tạo macro
có tham số như
sau :

THÍ DỤ VỀ MACRO



DISPLAY MACRO STRING

PUSH AX

PUSH DX

LEA DX, STRING

MOV AH,9

INT 21H

POP DX

POP AX

ENDM

GỌI: DISPLAY CHUOI

TRAO ĐỔI THAM SỐ CỦA MACRO

MACRO LOCATE : ĐỊNH VỊ CURSOR MÀN HÌNH

LOCATE MACRO ROW, COLUMN

```
PUSH AX  
PUSH BX  
PUSH DX  
MOV BX, 0  
MOV AH, 2  
MOV DH, ROW  
MOV DL, COLUMN  
INT 10H  
POP DX  
POP BX  
POP AX  
ENDM
```

SỬ DỤNG MACRO

TA CÓ CÁC DẠNG SỬ DỤNG
SAU :

LOCATE 10,20
LOCATE ROW, COL
LOCATE CH, CL

**CHÚ Ý : KHÔNG DÙNG CÁC
THANH GHI AH, AL, BH, BL VÌ
SẼ ĐỘNG ĐỘ VỚI CÁC
THANH GHI ĐÃ SỬ DỤNG
TRONG MACRO**

MACRO LỒNG NHAU

MỘT CÁCH ĐƠN GIẢN ĐỂ XÂY DỰNG MACRO LÀ XÂY DỰNG 1 MACRO MỚI TỪ MACRO ĐÃ CÓ.

EX : HIỂN THỊ 1 CHUỖI TẠI 1 TOẠ ĐỘ CHO TRƯỚC

DISPLAY_AT MACRO ROW, COL, STRING

LOCATE ROW, COL ; Gọi macro định vị cursor

DISPLAY STRING ; Gọi Macro xuất string

ENDM

**MỘT MACRO CÓ THỂ THAM CHIẾU ĐẾN CHÍNH NÓ,
NHỮNG MACRO NHƯ VẬY GỌI LÀ MACRO ĐỆ QUI.**

ĐỊNH NGHĨA NHÃN BÊN TRONG MACRO

TRONG MACRO CÓ THỂ CÓ NHÃN.
GỌI MACRO NHIỀU LẦN → NHIỀU NHÃN ĐƯỢC TẠO RA
→ LÀM SAO GIẢI QUYẾT VẤN ĐỀ NHẨY ĐIỀU KHIỂN?

ASSEMBLY GIẢI QUYẾT VẤN ĐỀ NÀY BẰNG CHỈ THỊ LOCAL
ÔNG BỨC MASM TẠO RA 1 TÊN DUY NHẤT CHO MỖI MỘT
LABEL KHI MACRO ĐƯỢC GỌI.

CÚ PHÁP : LOCAL LABEL_NAME

Một số Macro yêu cầu user định nghĩa các thành phần dữ liệu và các nhãn bên trong định nghĩa của Macro.

Nếu sử dụng Macro này nhiều hơn 1 lần trong cùng một chương trình, trình ASM định nghĩa thành phần dữ liệu hoặc nhãn cho mỗi lần sử dụng → các tên giống nhau lặp lại khiến cho ASM báo lỗi.

Để đảm bảo tên nhãn chỉ được tạo ra 1 lần, ta dùng chỉ thị LOCAL ngay sau phát biểu Macro

- Khi ASM thấy 1 biến được định nghĩa là LOCAL nó sẽ thay thế biến này bằng 1 ký hiệu có dạng ??n, trong đó n là 1 số có 4 chữ số. Nếu có nhiều nhãn có thể là ??0000, ??0001, ??0002 ...

Ta cần biết điều này để trong CT chính ta không sử dụng các biến hay nhãn dưới cùng 1 dạng.

Thí dụ minh họa chỉ thị Local

Xây dựng Macro REPEAT có nhiệm vụ xuất count lần số ký tự char ra màn hình.

REPEAT MACRO CHAR, COUNT

LOCAL L1

MOV CX, COUNT

L1: MOV AH,2

MOV DL, CHAR

INT 21H

LOOP L1

ENDM

GIẢ SỬ GỌI :
REPEAT ‘A’, 10
REPEAT ‘*’, 20

**ASM SẼ DÙNG CƠ CHẾ
ĐÁNH SỐ CÁC NHÃN (TỪ
0000H ĐẾN FFFFH) ĐỂ
ĐÁNH DẤU CÁC NHÃN CÓ
CHỈ ĐỊNH LOCAL.**

SẼ ĐƯỢC DỊCH RA →

Thí dụ minh họa chỉ thị Local

```
MOV CX, 10
??0000 : MOV AH,2
MOV DL, 'A'
INT 21H
LOOP ??0000
MOV CX, 20
??0001 : MOV AH,2
MOV DL, '*'
INT 21H
LOOP ??0001
```

GIẢ SỬ GỌI:
REPEAT 'A', 10
REPEAT '*', 20

Thí dụ minh họa

Viết 1 macro đưa từ lớn hơn trong 2 từ vào
AX

GETMAX MACRO WORD1, WORD2

LOCAL EXIT

MOV AX, WORD1

CMP AX, WORD2

JG EXIT

MOV AX, WORD2

EXIT :

ENDM

GIẢ SỬ FIRST,SECOND, THIRD LÀ
CÁC BIẾN WORD.

SỰ THAM CHIẾU MACRO ĐƯỢC
MỞ RỘNG NHƯ SAU :

MOV AX, FIRST

CMP AX, SECOND

JG ??0000

MOV AX, SECOND

??0000:

Thí dụ minh họa

Viết 1 macro đưa từ lớn hơn trong 2 vào AX

LỜI GỌI MACRO TIẾP THEO :

GETMAX SECOND, THIRD

ĐƯỢC MỞ RỘNG NHƯ SAU :

MOV AX, SECOND

CMP AX, THIRD

JG ??0001

??0001 :

**SỰ THAM CHIẾU LIÊN TIẾP
MACRO NÀY HAY ĐẾN MACRO
KHÁC KHIẾN TRÌNH BIÊN DỊCH
CHÈN CÁC NHÃN ??0002, ??0003
VÀ CỨ NHƯ VẬY TRONG CHƯƠNG
TRÌNH CÁC NHÃN NÀY LÀ DUY
NHẤT.**

THƯ VIỆN MACRO

CÁC MACRO MÀ CHƯƠNG TRÌNH THAM CHIẾU CÓ THỂ ĐẶT Ở FILE RIÊNG → TA CÓ THỂ TẠO 1 FILE THƯ VIỆN CÁC MACRO.

- DÙNG 1 EDITOR ĐỂ SOẠN THẢO MACRO
- LƯU TRỮ TÊN FILE MACRO.LIB
- KHI CẦN THAM CHIẾU ĐẾN MACRO TA DÙNG CHỈ THỊ INCLUDE TÊN FILE THƯ VIỆN

MỘT CÔNG DỤNG QUAN TRỌNG CỦA MACRO LÀ TẠO RA CÁC LỆNH MỚI.

SO SÁNH GIỮA MACRO & THỦ TỤC

■ THỜI GIAN BIÊN DỊCH.

MACRO ÍT TỐN THỜI GIAN BIÊN DỊCH HƠN PROCEDURE

■ THỜI GIAN THỰC HIỆN : NHANH HƠN PROCEDURE VÌ KHÔNG TỐN THỜI GIAN KHÔI PHỤC TRẠNG THÁI THÔNG TIN KHI ĐƯỢC GỌI → TỐC ĐỘ NHANH HƠN.

■ KÍCH THƯỚC : KÍCH THƯỚC CT DÀI HƠN

CÁC LỆNH LẶP TRONG MACRO

■ REP <BIỂU THỨC> :

...

ENDM

■ TÁC DỤNG : LẶP LẠI CÁC KHỐI LỆNH TRONG MACRO VỚI SỐ LẦN LÀ <BIỂU THỨC>

EX : MSHL MACRO OPER, BITS

REPT BITS

SHL DEST, 1

ENDM

ENDM

GỌI MSHL BX, 3

SẼ ĐƯỢC THAY THẾ BẰNG :

SHL BX, 1

SHL BX, 1

SHL BX, 1

CÁC LỆNH LẶP TRONG MACRO

■ IRP <THÔNG SỐ>, <DANH SÁCH CÁC TRỊ TRONG NGOẶC NHỌN> :

...
ENDM

TÁC DỤNG :

- LẶP LẠI KHỐI LỆNH TÙY THEO DANH SÁCH TRỊ.
- SỐ LẦN LẶP CHÍNH LÀ SỐ TRỊ TRONG DANH SÁCH
- MỖI LẦN LẶP LẠI SẼ THAY <THÔNG SỐ> BẰNG 1 TRỊ TRONG DANH SÁCH VÀ SẼ LẦN LUỘT LẤY HẾT CÁC TRỊ TRONG DANH SÁCH.

EX : PROCTABLE LABEL WORD

IRP PROCNAME, <MOVEUP, MOVDOWN,MOVLEFT,MOVRIGHT>

DW PROCNAME

ENDM

CÁC LỆNH LẶP TRONG MACRO

■ TUY NHIÊN CÁCH KHAI BÁO NÀY RƯỜM RÀ HƠN LÀ DÙNG :

**PROCTABLE DW MOVUP,
MOVEDOWN,MOVLEFT,MOVRIGHT**

➔ VIỆC SỬ DỤNG CÁC MACRO LẶP VÒNG NÀY CHO CÓ HIỆU QUẢ LÀ ĐIỀU KHÓ, ĐÒI HỎI PHẢI CÓ NHIỀU KINH NGHIỆM

BÀI TẬP MACRO

Bài 1 : 1. Viết một MACRO tính USCLN của 2 biến số M và N. Thuật toán USCLN như sau :

```
WHILE N <> 0 DO
    M = M MOD N
    Hoán vị M và N
END WHILE
```

Bài 2 : MACRO doi tu so chua trong ax sang chuoi tro den boi DI

```
; in : DI =offset chuoi
;      AX =so can doi
; out: khong co(chuoi van do di tro toi)
```

Bài 3 :Viết macro chuyển từ chuỗi thành số chưa trong ax

; in : DI =offset chuỗi
; out : AX =so đã đổi

*

Bài 4 : Viết MACRO xuất số hexa chứa trong AL ra màn hình

; INPUT : AL chứa số cần xuất; OUTPUT: nothing

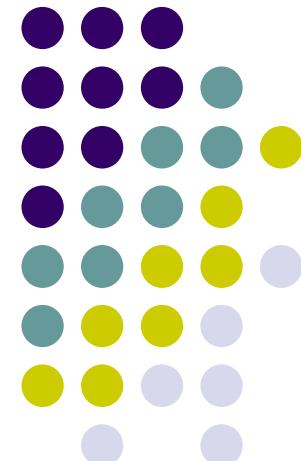
Bài 5 : Viết Macro in số hexa chứa trong BL ra dạng binary

;Input: BL chứa số cần in

;Output: Nothing

Chương 9 STACK & CHƯƠNG TRÌNH CON

- Giới thiệu STACK
- Một số ứng dụng của STACK
 - Cấu trúc của 1 CTC
 - Cơ chế làm việc của 1 CTC
 - Vấn đề truyền tham số
- Chương trình gồm nhiều MODULE





GIỚI THIỆU STACK

STACK : là một cấu trúc dữ liệu một chiều. Các phần tử cất vào và lấy ra theo phương thức LIFO (Last In First Out). Mỗi chương trình phải dành ra một khối bộ nhớ để làm stack bằng khai báo **STACK**. Ví dụ :

.STACK 100H ; Xin cấp phát 256 bytes làm stack

- Là 1 phần của bộ nhớ, được tổ chức lưu trữ dữ liệu theo cơ chế vào sau ra trước (LIFO).



LẬP TRÌNH VỚI STACK

- Trong lập trình có khi cần truy xuất đến các phần tử trong STACK nhưng không được thay đổi trật tự của STACK. Để thực hiện điều này ta dùng thêm thanh ghi con trỏ BP :
trỏ BP về đỉnh Stack : MOV BP,SP
thay đổi giá trị của BP để truy xuất đến các phần tử trong Stack : [BP+2]



- Phần tử được đưa vào STACK lần đầu tiên gọi là đáy STACK, phần tử cuối cùng được đưa vào STACK được gọi là đỉnh STACK.

- Khi thêm một phần tử vào STACK ta thêm từ đỉnh, khi lấy một phần tử ra khỏi STACK ta cũng lấy ra từ đỉnh → địa chỉ của ô nhớ đỉnh STACK luôn luôn bị thay đổi.

SS dùng để lưu địa chỉ segment của đoạn bộ nhớ dùng làm STACK
SP để lưu địa chỉ của ô nhớ đỉnh STACK (trỏ tới đỉnh STACK)



THÍ DỤ

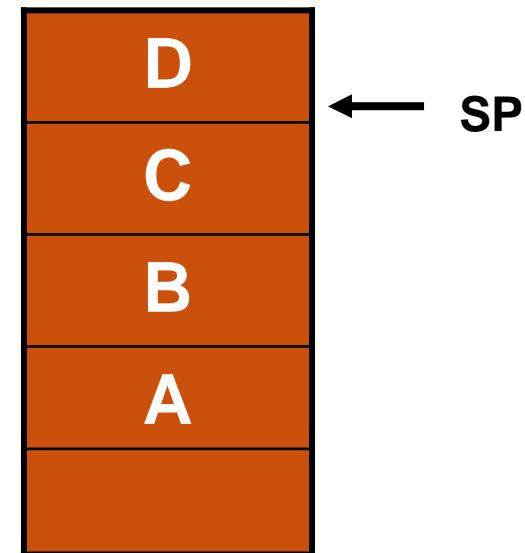
A,B,C là các Word
MOV BP,SP

MOV AX,[BP] ;AX = D

MOV AX,[BP+2] ;AX = C

MOV AX,[BP+6] ;AX = A

STACK





Để lưu 1 phần tử vào Stack ta dùng lệnh PUSH

Để lấy 1 phần tử ra từ Stack ta dùng lệnh POP

PUSH nguồn : đưa nguồn vào đỉnh STACK

PUSHF : cắt nội dung thanh ghi cờ vào STACK

- **nguồn là một thanh ghi 16 bit hay một từ nhớ**



POP và POPF : dùng để lấy một phần tử ra khỏi STACK.

Cú pháp : POP đích : đưa nguồn vào đỉnh STACK

 POPF : cắt nội dung ở đỉnh STACK

vào thanh ghi cờ

Chú ý : - Ở đây đích là một thanh ghi 16 bit (trừ thanh ghi IP) hay một từ nhớ

Các lệnh PUSH, PUSHF, POP và POPF không ảnh hưởng
tới các cờ



MỘT SỐ ỨNG DỤNG CỦA STACK

- Khắc phục các hạn chế của lệnh MOV
Ex : MOV CS,DS ; sai
PUSH DS
POP CS ; đúng
- Truyền tham số cho các chương trình con
- Lưu tạm thời giá trị thanh ghi hay biến.



THÍ DỤ 2

- Nhập vào 1 chuỗi, in chuỗi đảo ngược
Ex : nhập : Cong nghe thong tin
xuất : int gnoht ehgn gnoC



Ví dụ minh họa : dùng STACK trong thuật toán đảo ngược thứ tự như sau :

- ; Nhập chuỗi kí tự
- Khởi động bộ đếm
- Đọc một kí tự
- WHILE kí tự <> 13 DO
- Cắt kí tự vào STACK
- Tăng biến đếm
- Đọc một kí tự
- END WHILE
- ; Hiển thị đảo ngược
- FOR biến đếm lần DO
- Lấy một kí tự từ STACK
- Hiển thị nó
- END FOR



GIỚI THIỆU CHƯƠNG TRÌNH CON



- CTC là 1 nhóm các lệnh được gộp lại dưới 1 cái tên mà ta có thể gọi từ nhiều nơi khác nhau trong chương trình thay vì phải viết lại các nhóm lệnh này tại nơi cần đến chúng.



Lợi ích

CTC làm cho cấu trúc logic của chương trình dễ kiểm soát hơn, dễ tìm sai sót hơn và có thể tái sử dụng mã → tiết kiệm được công sức và thời gian lập trình.



CẤU TRÚC CỦA CTC CON

TÊN CTC PROC [NEAR|FAR]

CÁC LỆNH CỦA CTC

RET

TÊN CTC ENDP



MINH HỌA

- Viết chương trình nhập 1 số n (n nguyên dương và $n < 9$). Tính giải thừa của n và xuất ra màn hình dưới dạng số hex (giới hạn kết quả 16 bit). 
- Viết chương trình tìm số hoàn thiện (giới hạn 2 chữ số) và in nó ra màn hình. 

THÍ DỤ



```
.DATA  
EXTRN MemVar : WORD, Array1 : BYTE , ArrLength :ABS  
...  
.CODE  
EXTRN NearProc : NEAR , FarProc : FAR  
....  
MOV AX,MemVar  
MOV BX, OFFSET Array1  
MOV CX, ArrLength  
...  
CALL NearProc  
....  
CALL FarProc  
.....
```



CƠ CHẾ LÀM VIỆC CỦA CTC

- Cơ chế gọi và thực hiện CTC trong ASM cũng giống như ngôn ngữ cấp cao.

→ Khi gấp lệnh gọi CTC thì :

- . Địa chỉ của lệnh ngay sau lệnh gọi CTC sẽ được đưa vào STACK.
- . Địa chỉ của CTC được gọi sẽ được nạp vào thanh ghi IP.
- . Quyền điều khiển của CT sẽ được chuyển giao cho CTC.
- . CTC sẽ thực hiện các lệnh của nó và khi gấp RET, nó sẽ lấy địa chỉ cất trên STACK ra và nạp lại thanh ghi IP để thực thi lệnh kế tiếp.



PUBLIC EXTRN GLOBAL

Để thuận lợi trong việc dịch, liên kết chương trình đa file, Assembler cung cấp các điều khiển Public, Extern và Global.

PUBLIC



Chỉ cho Assembler biết nhãn (label) nào nằm trong module này được phép sử dụng ở các module khác.

Cú pháp : PUBLIC tên nhãn
khai báo nhãn

TÊN CTC

TÊN BIẾN

TÊN ĐI TRƯỚC NHÃN



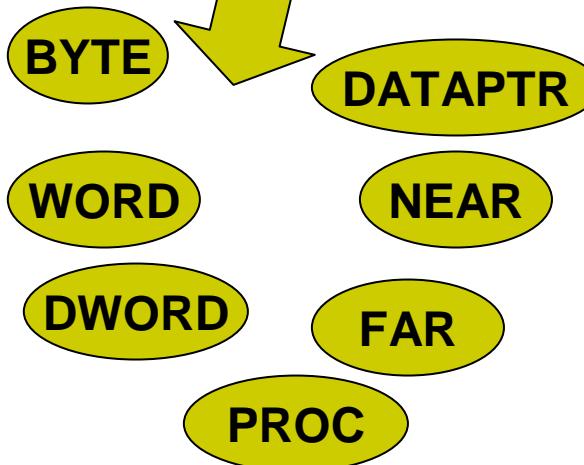


EXTRN



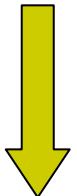
Báo cho Assembler biết những nhãn đã
được khai báo PUBLIC ở các module
khác được sử dụng trong module này mà
không cần phải khai báo lại.

Cú pháp : EXTRN Tên nhãn : Kiểu





GLOBAL

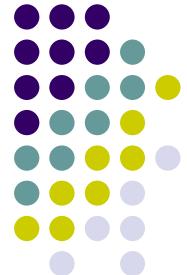


THAY THẾ PUBLIC VÀ EXTRN.

Viết chương trình nằm trên 2 file (2 module) với sự phân công như sau :

Module của chương trình chính (Main.ASM) có nhiệm vụ xác định Offset của 2 chuỗi ký tự và gọi CTC nối 2 chuỗi này và cho hiện kết quả ra màn hình.

Module CTC (Sub.ASM) làm nhiệm vụ nối 2 chuỗi và đưa vào bộ nhớ.



Ví dụ minh họa về STACK, CALL/RET : chương trình in một số nguyên (16 bit) ra màn hình

```
PrintNum10 PROC
; số nguyên N nằm trong AX
    PUSH BX CX DX
    MOV CX, 0      ; so lan push (so ky tu)

laysodu:
    XOR DX, DX    ; cho DX = 0 trước khi chia
    MOV BX, 10
    DIV BX        ; số dư trong DX, phần nguyên
trong AX
    PUSH DX        ; lưu phần dư vào stack
    INC CX
    CMP AX, 0     ; đã hết chưa?
    JNZ laysodu ; chưa hết, lấy số dư tiếp
    MOV AH, 2

INSO:
    POP DX
    ADD DL, '0'
    INT 21H
    LOOP inso
    POP DX CX BX
    RET

ENDP PrintNum10
```



CHƯƠNG TRÌNH ĐA FILE

- Cho phép nhiều user cùng tham gia giải quyết 1 chương trình lớn.
- Sửa module nào thì chỉ cần dịch lại module đó.
- Mỗi module chỉ giải quyết 1 vấn đề → dễ tìm sai sót.



VẤN ĐỀ TRUYỀN THAM SỐ

- CHUYỂN GIÁ TRỊ CỦA THAM SỐ TỪ
CT GỌI → CT ĐƯỢC GỌI

Có 3 cách truyền tham số

Thông qua thanh ghi

Thông qua biến toàn cục

Thông qua STACK



TRUYỀN THAM SỐ THÔNG QUA THANH GHI

- DỄ
- ĐƠN GIẢN
- THƯỜNG ĐƯỢC SỬ DỤNG ĐỐI VỚI
NHỮNG CT THUẦN TÚY ASM

**ĐẶT 1 GIÁ TRỊ NÀO ĐÓ VÀO THANH GHI
Ở CTCHÍNH VÀ SAU ĐÓ CTC SẼ SỬ
DỤNG GIÁ TRỊ NÀY TRONG THANH GHI.**

TRUYỀN THAM SỐ THÔNG QUA BIẾN GLOBAL



- KHAI BÁO BIẾN TOÀN CỤC.
- DÙNG NÓ ĐỂ CHUYỂN CÁC GIÁ TRỊ GIỮA CT GỌI VÀ CT ĐƯỢC GỌI.

CÁCH NÀY THƯỜNG ĐƯỢC DÙNG :

TRONG 1 CT VIẾT THUẦN TÚY BẰNG ASM

VIẾT HỒN HỌP GIỮA ASM VÀ 1 NGÔN NGỮ
CẤP CAO



TRUYỀN THAM SỐ QUA STACK

- PHỨC TẠP HƠN.
- DÙNG RẤT NHIỀU KHI VIẾT CHƯƠNG TRÌNH HỖN HỢP GIỮA ASM VÀ NGÔN NGỮ CẤP CAO.

CHUYỂN GIÁ TRỊ TỪ CTC CON LÊN CT CHÍNH.



- CŨNG THÔNG QUA CÁC THANH GHI, BỘ NHỚ VÀ STACK.

NẾU GIÁ TRỊ TRẢ VỀ LÀ 8 BIT HOẶC 16 BIT (CHO KHAI BÁO CHAR, INT, CON TRỎ GẦN) THÌ GIÁ TRỊ ĐÓ PHẢI ĐƯỢC ĐẶT TRONG THANH GHI AX CỦA HÀM TRƯỚC KHI QUAY VỀ CTCHÍNH.



CHUYỂN GIÁ TRỊ TỪ CTC CON LÊN CT CHÍNH.

- NẾU GIÁ TRỊ QUAY LẠI LÀ 32 BIT (CHO KHAI BÁO LONG, CON TRỎ XA) THÌ GIÁ TRỊ ĐÓ PHẢI ĐƯỢC ĐẶT TRONG THANH GHI DX, AX CỦA HÀM TRƯỚC KHI QUAY VỀ CT CHÍNH.



NEAR | FAR báo cho lệnh RET lấy địa chỉ quay về chương trình gọi nó trong STACK.

- NEAR : Lấy địa chỉ OFFSET (16BIT) trong STACK và gán vào thanh ghi IP.
- FAR : Lấy địa chỉ OFFSET và SEGMENT trong STACK nạp vào thanh ghi CS:IP.



VÂN ĐỀ BẢO VỆ CÁC THANH GHI

- CẦN ĐƯỢC QUAN TÂM TRONG QUÁ TRÌNH LẬP TRÌNH ASM.
- RẤT DỄ XÂY RA CÁC TRƯỜNG HỢP LÀM MẤT GIÁ TRỊ CỦA MÀ CT CHÍNH ĐÃ ĐẶT VÀO THANH GHI ĐỂ SỬ DỤNG SAU NAY KHI TA GỌI CTCON.



CÁC VÍ DỤ MINH HỌA

**NHẬP VÀO 1 SỐ HỆ HEX. IN RA SỐ ĐÃ
NHẬP VỚI YÊU CẦU SAU :**

VIẾT CTC CON NHẬP SỐ

VIẾT CTC CON XUẤT SỐ

CTC CHÍNH GỌI 2 CTC CON TRÊN.



LUYỆN TẬP LẬP TRÌNH C10

Bài 1 : Viết chương trình nhập 1 số nguyên n ($n < 9$). Tính giai thừa của n và xuất kết quả ra màn hình dưới dạng số Hex (giới hạn 16 bits).

Bài 2 :Viết chương trình nhap vao 1 chuỗi ky tu. Hay in ra man hinh chuỗi ky tu vua nhap theo thứ tự đảo (trong mỗi từ đảo từng ký tự).

**Bài 3 :Viết chương trình kiểm tra một biểu thức đại số có chứa các dấu ngoặc (như (), [] và {}) là hợp lệ hay không hợp lệ .
Ví dụ : (a + [b – { c * (d – e) }] + f) là hợp lệ nhưng (a + [b – { c * (d – e)] } + f) không hợp lệ.
HD : dùng ngăn xếp để PUSH các dấu ngoặc trái (‘(‘, ’{‘, ‘[‘) vào Stack**



Bài 4 : Viết chương trình nhập vào 1 ký tự, cho biết ký tự vừa nhập thuộc loại gì ? – ký tự, ký số , toán tử toán học hay ký tự khác. Nếu ký tự là phím Escape thì thoát chương trình.

```
C:\WINDOWS\System32\cmd.exe
D:\HUFLIT\BTASM>LOAIKYTU

Nhap vao mot ki tu :Q
Ky tu vua nhap vao la chu in hoa
Nhap vao mot ki tu :d
Ky tu vua nhap vao la chu in thuong
Nhap vao mot ki tu :3
Ky tu vua nhap vao la chu so
Nhap vao mot ki tu :+
Ky tu vua nhap vao la toan tu toan hoc
Nhap vao mot ki tu :@
Ky tu vua nhap vao la ky tu loai khac
Nhap vao mot ki tu :-
D:\HUFLIT\BTASM>
```



**Bài 6 :Viết chương trình nhập 1 chuỗi ký tự.
Xuất ký tự dưới dạng viết hoa ký tự đầu của từng từ,
các ký tự còn lại là chữ thường**

Ex :

Nhập : ngo phuoc nguyen

Xuất : Ngo Phuoc Nguyen

Nhập : VU tHanh hIEn

Xuất : Vu Thanh Hien

Bài 7 : Viết chương trình tìm số hoàn thiện (giới hạn 2 chữ số). Xuất các số hoàn thiện từ số lớn nhất đến số nhỏ.

LẬP TRÌNH XỬ LÝ MÀN HÌNH & BÀN PHÍM

- Giới thiệu màn hình & việc quản lý màn hình
- Hiểu được tổ chức của màn hình.
- So sánh chức năng điều khiển màn hình của INT 10h của ROM BIOS với chức năng của INT 21h.
- Biết cách lập trình quản lý màn hình trong ASM.
- Biết cách lập trình xử lý phím và 1 số ứng dụng của nó.

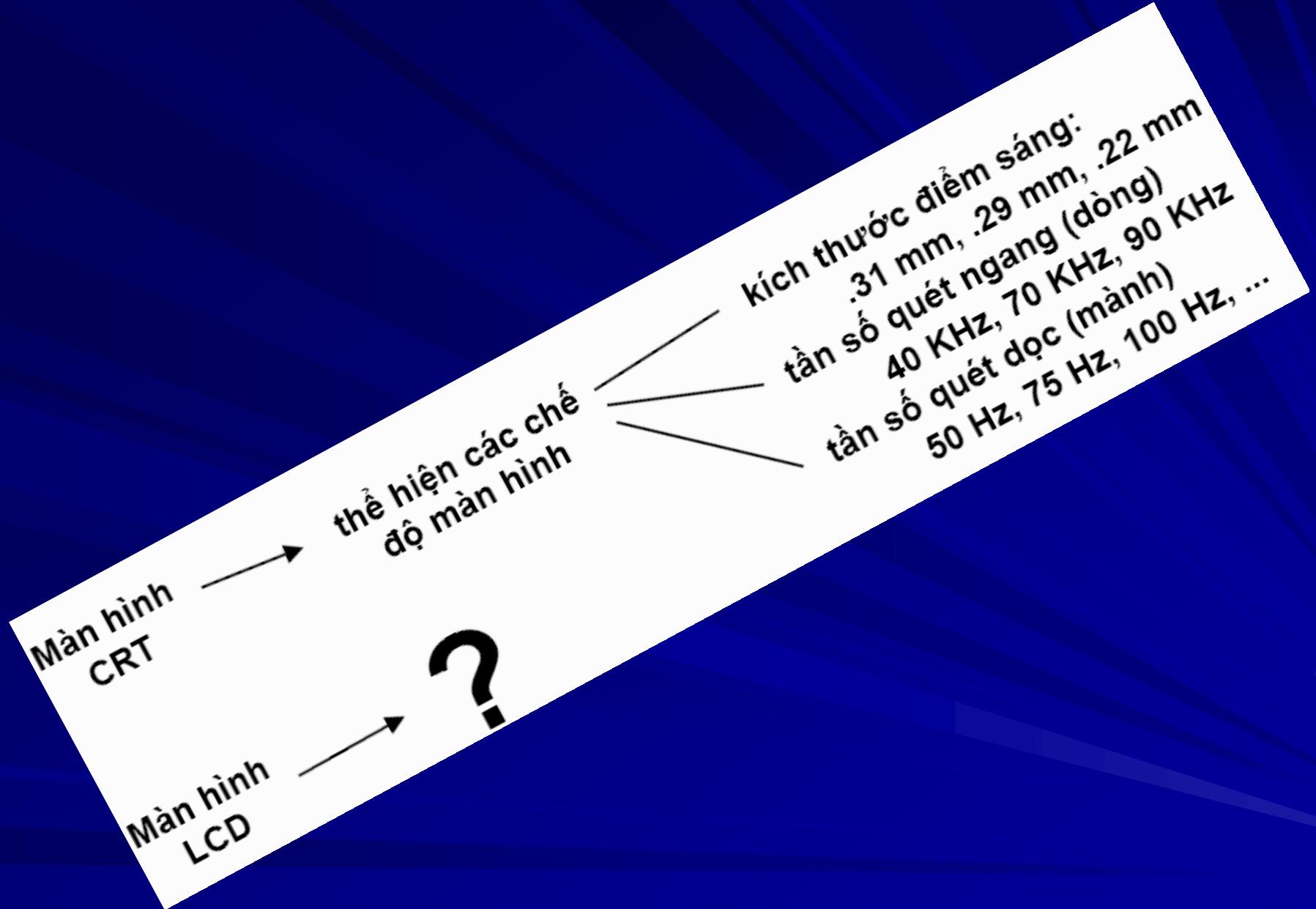
MÀN HÌNH

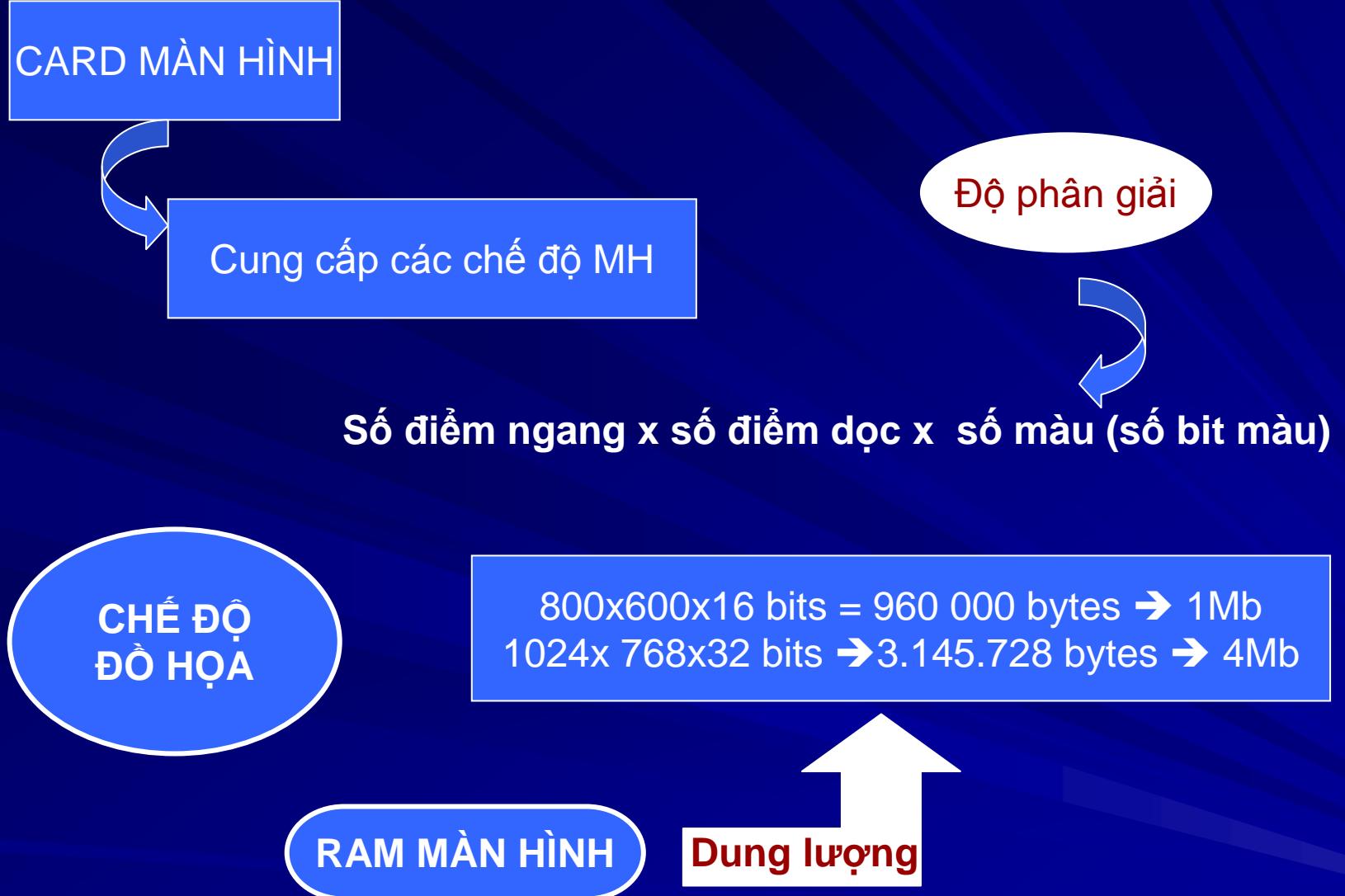
ĐẶC TRƯNG CỦA MÀN HÌNH

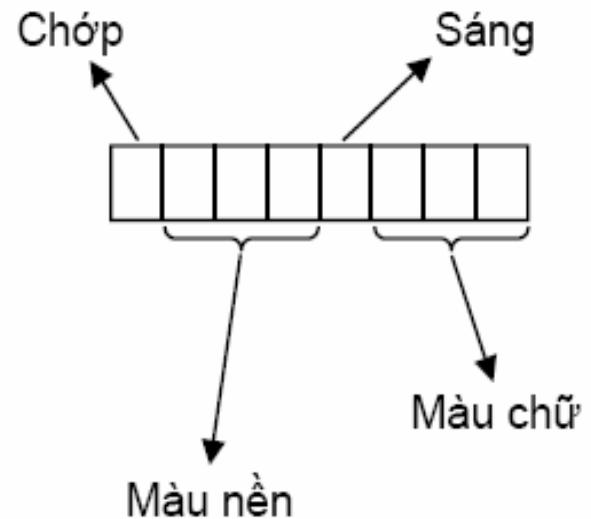
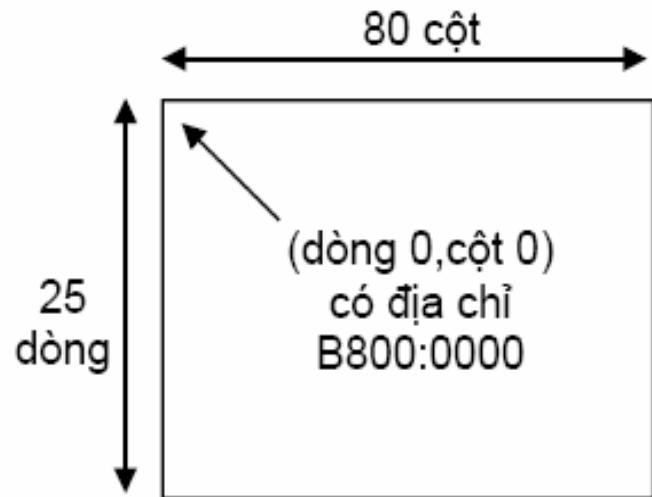


ĐỘ PHÂN GIẢI

■ Độ phân giải : số điểm trong màn hình.Hình ảnh ma trận gồm 1 lưới hình chữ nhật các điểm (thí dụ $640*480$). Độ phân giải thường cho dưới dạng $h \times v$ trong đó h là số lượng pixel theo dòng và v là số lượng pixel theo cột.







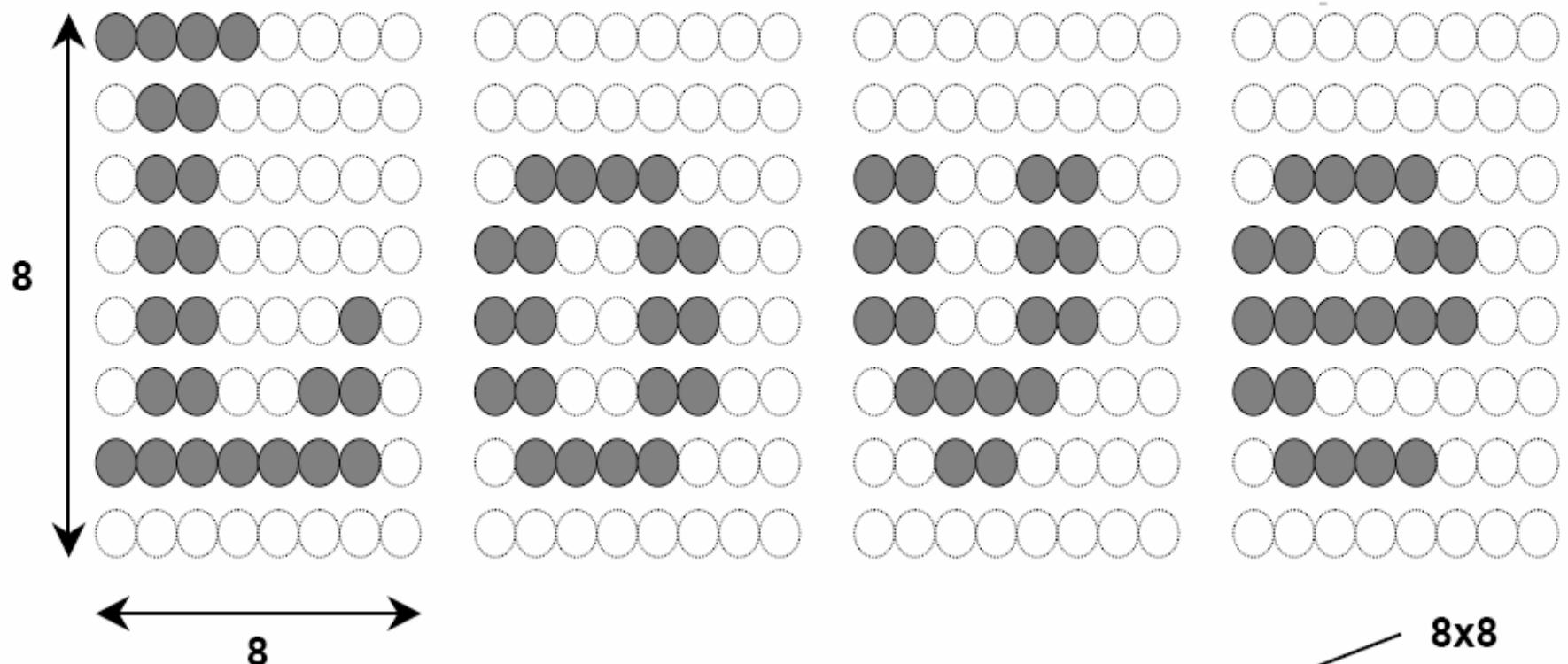
- Mỗi ký tự lưu bằng 2 byte.
- Byte địa chỉ thấp chứa mã ASCII.
- Byte địa chỉ cao chứa mã màu

$$\text{Địa chỉ } (i,j) = \text{B800:0000} + (i * 160 + j * 2)$$

Mã trận kí tự trên màn hình

Thiết bị
ngoại vi

Chế độ văn bản



QUẢN LÝ MÀN HÌNH

- Màn hình được điều khiển hiệu quả nhờ các chức năng của INT 10H trong Rom Bios. Các chức năng này quản lý màn hình tốt hơn các chức năng của INT 21h của Dos.
- Bên cạnh 1 số chức năng do INT 21h của Dos cung cấp, 1 số tác vụ được thực hiện trên màn hình nhờ các chức năng trong INT 10h như xoá màn hình, định vị con trỏ, thiết lập màn hình ...
- IBM PC hỗ trợ 3 loại màn hình cơ bản có tên tùy thuộc vào loại Card màn hình cắm trên Bus mở rộng trên Mainboard như : Monochrome chỉ hiển thị text đơn sắc; CGA (Color Graphic Adaptor) cho phép hiển thị text và đồ họa; EGA (Enhanced Graphics Adaptor) hiển thị text và đồ họa với độ phân giải cao hơn. Ngoài ra còn có card VGA (Video Graphics Array), SVGA ...

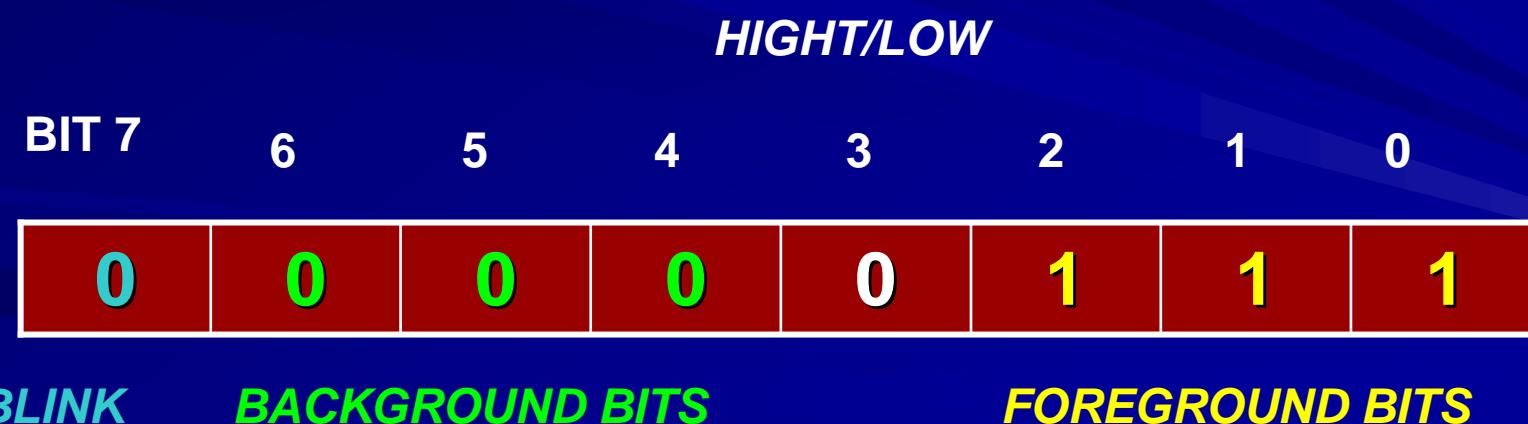
THUỘC TÍNH MÀN HÌNH

- Mỗi ký tự hiển thị (văn bản) chiếm 2 byte trong RAM.
- Bai 1: mã ASCII của ký tự
- Bai 2: byte thuộc tính xác định màu ký tự



THUỘC TÍNH MÀN HÌNH

- Mỗi vị trí trên màn hình có thể lưu 1 ký tự đơn cùng với thuộc tính riêng của ký tự này chẳng hạn như đảo màu, nhấp nháy, chiếu sáng, gạch dưới ...
- Thuộc tính của ký tự được lưu trong 1 byte gọi là byte thuộc tính.



THUỘC TÍNH MÀN HÌNH

■ Ex : các ký tự màu vàng chanh nhấp nháy trên nền màu nâu

BLINK = 10000000B

RED = 100B

MOV BH, (RED SHL 4) +YELLOW+BLINK

ĐỂ TẠO 1 BYTE THUỘC TÍNH VIDEO TỪ 2 MÀU, TA DÙNG SHL CHUYỂN CÁC BIT MÀU NỀN SANG TRÁI 4 VI TRÍ.

VÙNG HIỂN THỊ MÀN HÌNH

- Vùng hiển thị của màn hình đơn sắc ở địa chỉ B000h trong Bios.
- Vùng hiển thị video đồ họa màu cơ bản bắt đầu từ vị trí B800h của Bios.

THUỘC TÍNH MÀN HÌNH

■ Các thuộc tính chuẩn của màn hình Monochrome :

HEX VALUE	ATTRIBUTE
07H	Normal – thường
87H	Blinking – nhấp nháy
0FH	Bright – sáng
70H	Reverse – đảo thuộc tính
01H	Underline
09H	Bright Underline

THUỘC TÍNH MÀN HÌNH (tt)

- Bất kỳ 1 thuộc tính nào cũng có thể thêm thuộc tính nhấp nháy bằng cách cho bit 7 có trị là 1. Thí dụ normal blinking 87H, bright linking 8Fh.

Card màn hình CGA và EGA không hỗ trợ thuộc tính Underline nhưng cho phép sử dụng màu trong text mode. Các màu được chia làm 2 loại : màu chữ (Foreground) và màu nền (Background).

Bit 6,5,4 : màu nền

Bit 2,1,0 : màu chữ

Bit 3 : độ sáng

BẢNG MÀU (COLOR PALETTE)

BACKGROUND COLOR

foreground color only

000	BLACK	1000	GRAY
001	BLUE	1001	LIGHT BLUE
010	GREEN	1010	LIGHT GREEN
011	CYAN	1011	LIGHT CYAN
100	RED	1100	LIGHT RED
101	MAGENTA	1101	LIGHT MAGENTA
110	BROWN	1110	YELLOW
111	WHITE	1111	BRIGHT WHITE

EX : 01101110 : 06EH nền Brown, chữ Yellow, không nhấp nháy.

EX : 11010010 : 0D2H nền Magenta, chữ Green, nhấp nháy.

CÁC MODE MÀN HÌNH

- Các Card màn hình CGA,EGA,VGA cho phép chuyển đổi Video mode nhờ INT 10h.
- Các trình ứng dụng thường dùng INT 10h để tìm Video mode hiện hành.

Ex: 1 ứng dụng thường muốn thể hiện đồ họa với độ phân giải cao (640x200) phải kiểm tra chắc chắn rằng MT hiện đang sử dụng đang dùng Card màn hình CGA,VGA hoặc EGA.

CÁC MODE MÀN HÌNH

- Có 2 chế độ làm việc của màn hình : text và đồ họa.

Màn hình là hình ảnh của Video Ram.

- Chế độ màn hình :
25 dòng và 80 cột
25 dòng và 40 cột.



*Ở chế độ text một trang màn hình
cần tối thiểu bao nhiêu byte của
VIDEO Ram*



$$25 \times 80 \times 2 = 4000 \text{ BYTES RAM VIDEO}$$

VÙNG NHỚ NÀY NẰM TRÊN CARD MH

CÁC MODE MÀN HÌNH

Các Video mode thông dụng :

Mode	Mô tả
<i>02h</i>	<i>80x25 black and white text</i>
<i>03h</i>	<i>80x25 color text</i>
<i>04h</i>	<i>320x400 4 color graphics</i>
<i>06h</i>	<i>640x200 2 color graphics</i>
<i>07h</i>	<i>80x25 black and white text, monochrome adaptor only</i>
<i>0Dh</i>	<i>320x200 16 color graphics</i>
<i>0Eh</i>	<i>640x200 16 colors graphics, EGA, VGA only</i>
<i>0Fh</i>	<i>640x350 monochrome graphics, EGA, VGA only</i>
<i>10h</i>	<i>640x350 16 colors graphics, EGA, VGA only</i>

TRANG MÀN HÌNH (VIDEO PAGE)

Tất cả các Card CGA đều có khả năng lưu trữ nhiều màn hình text gọi là các trang màn hình (video page) trong bộ nhớ. Riêng card mono chỉ hiển thị 1 trang – trang 0. Số trang phụ thuộc vào mode màn hình.

Trong card màn hình màu, ta có thể ghi vào 1 trang này trong khi hiển thị trang khác hoặc chuyển đổi qua lại vị trí giữa các trang. Các trang được đánh số từ 0 đến 7.

TRANG MÀN HÌNH (VIDEO PAGE)

số trang	mode	adaptor
0	07h	monochrome
0-7	00h – 01h	CGA
0-3	02h-03h	CGA
0-7	02h-03h	EGA
0-7	0Dh	EGA
0-3	0Eh	EGA
0-1	0Fh, 10h	EGA

THÍ DỤ VỀ TRANG MH

**ĐỂ HIỂN THỊ 1 KÝ TỰ VỚI THUỘC TÍNH CỦA
NÓ TẠI 1 VỊ TRÍ BẤT KỲ → CHỨA KÝ TỰ
VÀ THUỘC TÍNH VÀO TỪ TƯƠNG ỨNG
TRONG TRANG HIỂN THỊ HOẠT ĐỘNG.**

EX : Lấp đầy màn hình bằng chữ ‘A’ màu đỏ trên nền xanh

CHẾ ĐỘ ĐỒ HỌA

- MOV AH, 0
MOV AL, CheDo
INT 10h
- CheDo= 4 320 x 200 4 màu CGA
 6 640 x 200 2 màu CGA
 D 320 x 200 16 màu EGA
 E 640 x 200 16 màu EGA
 10 640 x 350 16 màu EGA
 11 640 x 480 2 màu VGA
 12 640 x 480 16 màu VGA
 13 640 x 480 256 màu VGA

Truy xuất thiết bị xuất chuẩn (màn hình)

. 1. Chọn chế độ hiển thị :

Chức năng AH = 0, ngắt 10H

Vào : AH = 0, AL = kiểu

Ví dụ : thiết lập chế độ văn bản màu

XOR AH, AH

MOV AL, 3 ; chế độ văn bản màu 80 x 25

INT 10H

THAY ĐỔI SIZE CON TRỎ MÀN HÌNH

Chức năng AH = 1, ngắt 10H

Vào : AH = 1,
CH = dòng quét đầu, CL = dòng quét cuối

Ví dụ : thiết lập con trỏ với kích thước lớn nhất

```
MOV AH, 1
MOV CH, 0 ; dòng bắt đầu
MOV CL, 13 ; dòng kết thúc
INT 10H
```

DỊCH CHUYÊN CON TRỎ

Chức năng AH = 2, ngắt 10H
Vào : AH = 2,
DH = dòng mới (0-24),
DL = cột mới (0-79)
BH = số hiệu trang

Ví dụ : Di chuyển con trỏ đến giữa màn
hình 80 x 25 của trang 0

```
MOV AH, 2
XOR BH, BH      ; trang 0
MOV DX, 0C27H    ; dòng 12 cột 39
INT 10H
```

LẤY VỊ TRÍ KÍCH THUỐC CON TRỎ HIỆN HÀNH

Chức năng AH = 3, ngắt 10H

Vào : AH = 3, BH = số hiệu trang

Ra : DH = dòng, DL = cột,

CH = dòng quét đầu, CL = dòng quét cuối

**Ví dụ : Di chuyển con trỏ lên một dòng nếu nó không ở dòng
trên cùng**

```
MOV  AH, 3
XOR  BH, BH      ; trang 0
INT  10H
OR   DH, DH ; dòng trên cùng DH = 0 ?
JZ   exit
MOV  AH, 2 ; chức năng dịch con trỏ
DEC  DH    ; giảm một dòng
INT  10H
```

exit :

CUỘN MÀN HÌNH

Chức năng AH = 6, ngắt 10H

Vào : AH = 6,
AL = số dòng cuộn (= 0 là toàn màn hình)
Ra : BH = thuộc tính các dòng trống ,
CH, CL = dòng, cột góc trái trên
DH, DL = dòng, cột góc phải dưới của cửa sổ

Ví dụ : Xoá đen màn hình 80 x 25

```
MOV AH, 6
XOR AL, AL
XOR CX, CX
MOV DX,
184FH ; góc phải dưới
MOV BH, 7
INT 10H
```

Ví dụ tổng hợp : Viết chương trình thực hiện như sau:

- . Lập chế độ hiển thị màu 80 x 25
- . Xoá cửa sổ tại góc trái trên : cột 26 dòng 8 và góc phải dưới tại cột 52 dòng 16 thành màu đỏ. .
- Sau đó hiển thị kí tự A màu cam tại vị trí con trỏ.

CÁC HÀM XỬ LÝ MÀN HÌNH

Các chức năng xử lý màn hình nằm trong INT 10h

<i>Chức năng (để trong AH)</i>	<i>nhiệm vụ</i>
0	<i>set video mode</i> chọn mono, text, graphic hoặc color mode
1	<i>Set cursor line</i> thiết lập 1 dòng quét tạo dạng cho cursor.
2	<i>Set cursor position</i> định vị cursor
3	<i>get cursor position</i> lấy vị trí cursor
4	<i>đọc vị trí và trạng thái của bút vẽ light pen.</i>
5	<i>chọn trang muốn hiển thị.</i>
6	<i>cuộn của sổ hiện hành lên, thế các dòng cuộn bằng ktrống.</i>
7	<i>cuộn của sổ hiện hành xuống.</i>
8	<i>đọc ký tự và thuộc tính ký tự tại vị trí con trỏ hiện hành.</i>
9	<i>ghi ký tự và thuộc tính ký tự tại vị trí con trỏ hiện hành.</i>

CÁC HÀM XỬ LÝ MÀN HÌNH

Các chức năng xử lý màn hình nằm trong INT 10h

Chức năng (để trong AH)

nhiệm vụ

0Ah Ghi ký tự bỏ qua thuộc tính ký tự vào vị trí con trỏ hiện hành.

0Bh Chọn palette màu

0Ch Ghi 1 điểm graphic trong graphics mode.

0Dh Đọc giá trị màu của 1 pixel có vị trí đã biết.

0Eh Ghi ký tự ra màn hình và cập nhật con trỏ sang phải 1 vtrí.

0Fh Lấy mode màn hình hiện hành để xem đang ở chế độ text hay graphics.

HÀM 0H INT 10H

- Thiết lập video mode.

AH = 0

AL = mode.

Nếu bit cao của AL = 0 sẽ tự động xoá màn hình.

Nếu bit cao của AL = 1 không xoá màn hình.

- Ex : thiết lập 80x25 color text mode

MOV AH, 0

MOV AL, 3 ; mode 3 , có xoá màn hình

INT 10h

LUU Y : Không muốn xoá màn hình thì AL = 83H

HÀM OH INT 10H

- Ex : đoạn chương trình sau sẽ thiết lập video mode là high resolution graphics, đợi gõ 1 phím sau đó thiết lập video mode là color text mode.

MOV AH, 0 ; set video mode

MOV AL, 6 ; 640x200 color graphics mode

INT 10h

MOV AH, 1 ; đợi gõ 1 phím

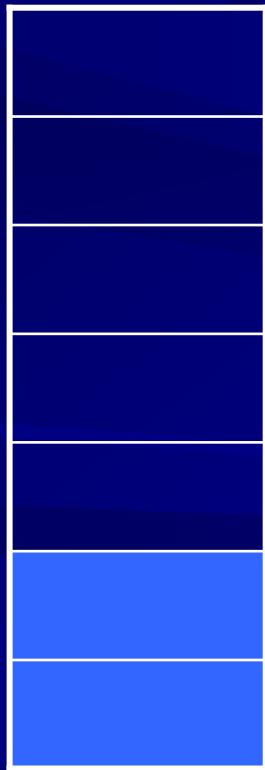
INT 21H

MOV AH, 0 ; set video mode

MOV AL, 3 ; color text mode

INT 10H

- Dạng con trỏ màn hình được tạo ra bằng cách chỉ định số dòng quét.
Việc thay đổi dạng con trỏ chính là thay đổi số lượng và vị trí dòng quét này.
- Màn hình monochrome dùng 13 dòng (từ 0 – 0Ch)
- Màn hình CGA,VGA dùng 8 dòng (từ 0-7).



CGA/EGA

7



MONOCHROME

Ex: Minh họa

Thiết lập con trỏ có hình khối đặc.

MOV AH , 1

MOV CH, 0

MOV CL,0CH

INT 10H

Để thay đổi dạng con trỏ :

AH = 1

CH = TOP (dòng đầu)

CL = BOTTOM (dòng cuối)

Ex2: trả kích thước con trỏ về dạng mặc định trước khi thoát.

```
MOV AH , 1  
MOV CX, 0607H  
INT 10H  
MOV AX, 4C00H  
INT 21H
```

```
MOV AH, 3  
MOV BH, 0  
INT 10H  
MOV SAVECURSOR, CX  
OR CH, 00100000  
INT 10H  
.....  
MOV AH, 1  
MOV CX, SAVECURSOR  
INT 10H
```

Ex2: lưu kích thước con trỏ hiện
hành vào 1 biến trước khi thay đổi
kích
thước con trỏ để sau này phục hồi
lại.

HÀM 02H
INT 10H

Thiết lập vị trí hiện hành
SET CURSOR POSITION

AH = 2 ; DH = CHỈ SỐ DÒNG ; DL = CHỈ SỐ CỘT ;
BH= TRANG MÀN HÌNH CHỨA CURSOR

Ex : THIẾT LẬP CURSOR TẠI TỌA ĐỘ (DÒNG 10, CỘT 20) CỦA TRANG 0

MOV AH, 2	; Chức năng set cursor
MOV DH, 10	; dòng 10
MOV DL, 20	; cột 20
MOV BH,0	; trang 0
INT 10H	; gọi BIOS

HÀM 03H
INT 10H

Lấy vị trí cursor hiện hành
GET CURSOR POSITION

AH = 3 ;
BH = TRANG MÀN HÌNH MUỐN LẤY CURSOR
Giá trị trả về :
CH = Dòng quét đầu của cursor
CL = Dòng quét cuối của cursor
DH = vị trí dòng màn hình
DL = vị trí cột màn hình

**HÀM 03H
INT 10H**

*Lấy vị trí cursor hiện hành
GET CURSOR POSITION*

Ex : lấy vị trí của cursor lưu vào biến.

Thường dùng trong các tác vụ menu.

```
MOV AH, 3  
MOV BH, 0  
INT 10H  
MOV SAVECURSOR, CX  
MOV CURRENT_ROW , DH  
MOV CURRENT_COL , DL
```

HÀM 05H
INT 10H

*THIẾT LẬP TRANG MÀN HÌNH
SET VIDEO PAGE*

AH = 5 ; AL = TRANG MÀN HÌNH SẼ LÀ TRANG HIỆN HÀNH

EX : THAY ĐỔI TRANG MH KHI GÓP 1 PHÍM BẤT KỲ

```
DOSSEG  
.MODEL SMALL  
.STACK 100H  
.CODE  
MAIN PROC  
MOV AX, @DATA  
MOV DS,AX
```

```
MOV DX, OFFSET TRANG0  
INT 21H  
MOV AH, 1  
INT 21H  
SANG_TRANG_1 :  
MOV AH, 5  
MOV AL, 1
```

HÀM 05H INT 10H

```
INT 10H  
MOV AH, 9  
MOV DX, OFFSET  
TRANG1  
  
INT 21H  
MOV AH, 1  
INT 21H  
  
SANG_TRANG_0 :  
MOV AH, 5  
MOV AL, 0
```

THIẾT LẬP TRANG MÀN HÌNH SET VIDEO PAGE

```
INT 10H  
MOV AX, 4C00H  
INT 21H  
MAIN ENDP  
.DATA  
TRANG0 DB ‘ DAY LA TRANG 0,$’  
TRANG1 DB ‘ DAY LA TRANG 1,$’  
END MAIN
```

HÀM 06H, 07H
INT 10H

*CUỘN MÀN HÌNH
SCROLL WINDOW UP AND DOWN*

CUỘN MÀN HÌNH LÀ TÁC VỤ LÀM CHO DỮ LIỆU TRƯỢT LÊN HOẶC XUỐNG.

CÁC DÒNG DỮ LIỆU BỊ CUỐN SẼ ĐƯỢC THAY THẾ BẰNG CÁC DÒNG TRỐNG TA ĐỊNH NGHĨA WINDOWS NHỜ HỆ TỌA ĐỘ HÀNG CỘT VỚI GỐC TỌA ĐỘ LÀ GÓC TRÊN TRÁI CỦA MÀN HÌNH.

HÀNG SẼ THAY ĐỔI TỪ 0 ĐẾN 24 TỪ TRÊN XUỐNG.

CỘT SẼ THAY ĐỔI TỪ 0 ĐẾN 79 TỪ TRÁI SANG PHẢI.

TẠO CÓ THỂ CUỘN 1 VÀI DÒNG HOẶC CẢ WINDOWS.

TOÀN BỘ WINDOWS BỊ CUỘN ➔ MÀN HÌNH BỊ XÓÁ.

HÀM 06H, 07H
INT 10H

*CUỘN MÀN HÌNH
SCROLL WINDOW UP AND DOWN*

CÁC THÔNG SỐ :

CUỘN LÊN AH =6 ; CUỘN XUỐNG AH =7

AL = SỐ DÒNG SẼ CUỘN (= 0 NẾU CUỘN TOÀN BỘ MÀN HÌNH)

CH, CL = TỌA ĐỘ HÀNG, CỘT CỦA GÓC TRÊN TRÁI CỦA WINDOWS

DH, DL = TỌA ĐỘ HÀNG, CỘT CỦA GÓC DƯỚI PHẢI CỦA WINDOWS

**BH = THUỘC TÍNH MÀN HÌNH CỦA CÁC DÒNG TRỐNG
KHI MÀN HÌNH ĐÃ CUỘN.**

HÀM 06H, 07H
INT 10H

CUỘN MÀN HÌNH
SCROLL WINDOW UP AND DOWN

EX : XÓA MÀN HÌNH BẰNG CÁCH CUỘN LÊN TOÀN BỘ MÀN HÌNH
VỚI THUỘC TÍNH NORMAL

```
MOV AH, 6  
MOV AL, 0  
MOV CH, 0  
MOV CL, 0  
MOV DL, 24  
MOV DH, 79  
MOV BH, 7  
INT 10H
```



```
MOV AX, 0600H  
MOV CX, 0000H  
MOV DX, 184FH  
MOV BH, 7  
INT 10H
```

HÀM 06H, 07H
INT 10H

CUỘN MÀN HÌNH
SCROLL WINDOW UP AND DOWN

EX : CUỘN WINDOWS TỪ (10,20) TỚI (15,60), CUỘN XUỐNG 2 DÒNG, 2 DÒNG CUỘN SẼ CÓ THUỘC TÍNH VIDEO ĐẢO.

MOV AX, 0702H

MOV CX,0A14H

MOV DX, 0F3CH

MOV BH, 70H

INT 10H

HÀM 08H
INT 10H

*ĐỌC 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.
READ CHARACTER AND ATTRIBUTE*

AH = 8 ; BH = TRANG MÀN HÌNH
TRI TRẢ VỀ :
AL = KÝ TỰ ĐÃ ĐỌC ĐƯỢC ; AH = THUỘC TÍNH CỦA KÝ TỰ

EX : THIẾT LẬP CURSOR TẠI HÀNG 5 CỘT 1 SAU ĐÓ NHẬN 1 KÝ TỰ
NHẬP.LƯU KÝ TỰ ĐÃ ĐỌC ĐƯỢC VÀ THUỘC TÍNH CỦA KÝ TỰ NÀY.

LOCATE :

MOV AH, 2

MOV BH, 0

MOV DX, 0501H

INT 10H

GETCHAR :

MOV AH, 8

MOV BH, 0

INT 10H

MOV CHAR, AL

MOV ATTRIB , AH

HÀM 09H
INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.
WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 09H INT 10H :

XUẤT (GHI) 1 HOẶC NHIỀU KÝ TỰ CÙNG VỚI THUỘC TÍNH CỦA CHÚNG LÊN MÀN HÌNH. CHỨC NĂNG NÀY CÓ THỂ XUẤT MỌI MÃ ASCII KỂ CẢ KÝ TỰ ĐỒ HỌA ĐẶC BIỆT CÓ MÃ TỪ 1 ĐẾN 31

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;

BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT

CX = HỆ SỐ LẶP

HÀM 0AH INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.
WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 0AH INT 10H :

XUẤT (GHI) 1 HOẶC NHIỀU KÝ TỰ CÙNG VỚI THUỘC TÍNH CỦA CHÚNG LÊN MÀN HÌNH. CHỨC NĂNG NÀY CÓ THỂ XUẤT MỌI MÃ ASCII KỂ CẢ KÝ TỰ ĐỒ HỌA ĐẶC BIỆT CÓ MÃ TỪ 1 ĐẾN 31

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;

BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT

CX = HỆ SỐ LẶP

HÀM 0AH INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.
WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 0AH INT 10H :

XUẤT (GHI) 1 HOẶC NHIỀU KÝ TỰ CÙNG VỚI THUỘC TÍNH CỦA CHÚNG LÊN MÀN HÌNH. CHỨC NĂNG NÀY CÓ THỂ XUẤT MỌI MÃ ASCII KỂ CẢ KÝ TỰ ĐỒ HỌA ĐẶC BIỆT CÓ MÃ TỪ 1 ĐẾN 31

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;

BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT

CX = HỆ SỐ LẶP

HÀM 0FH
INT 10H

LẤY VIDEO MODE
GET VIDEO MODE

CHỨC NĂNG 0FH INT 10H : LẤY VIDEO MODE

AH = 0F ;
BH = TRANG HIỆN HÀNH

AH = SỐ CỘT MÀN HÌNH ;
AL = MODE MÀN HÌNH HIỆN HÀNH

EX : MOV AH,0FH ; Get Video Mode Function

INT 10H ; gọi BIOS

MOVE VIDEO_MODE, AL ; lưu Video Mode vào biến bộ nhớ

MOV PAGE, BH ; lưu trang hiện hành.

LẬP TRÌNH XỬ LÝ PHÍM

1. Đọc phím nhấn :

Chức năng AH = 0, ngắt 16H

Vào : AH = 0

Ra : AL = mã ASCII nếu một phím ASCII được nhấn
= 0 nếu phím điều khiển được nhấn

AH = mã scan của phím nhấn

LẬP TRÌNH XỬ LÝ PHÍM

BÀN PHÍM

- Gồm 2 nhóm phím:
 1. ASCII: chữ (a..z), số (0..9), dấu (+-*/*...), Esc, Enter (\downarrow), BackSpace (\leftarrow), Tab ($\leftarrow\leftarrow$).
 2. ASCII mở rộng: Shift, CTrl, Alt, Caps Lock, Num Lock, Scroll Lock (thường dùng với phím khác); phím hàm (F1..F12), hướng (\leftarrow , \uparrow , \rightarrow , \downarrow , Home, End, Page Up, Page Down), Insert, Delete và các phím còn lại.

BÀN PHÍM

INT 21h, AH = 1 (DOS)

- Nhập ký tự bàn phím → AL (hiển thị)
- AL = ASCII (nhóm 1), AL = 0 (nhóm 2)
- INT 21h → AL = mã quét

INT 21h, AH = 8 (DOS)

- Như INT 21h, AH = 1 nhưng không hiển thị.
- Ví dụ sau chờ nhập “secret” và Enter. Chỉ kết thúc khi nhập đúng. Chuỗi nhập không hiển thị.
- → PASSWORD.ASM

BÀN PHÍM

INT 16h, AH = 0 (BIOS)

- ASCII / 0 → AL , mã quét → AH
- Phím nhập không hiển thị
- → INT16-00.ASM



BÀN PHÍM

INT 16h, AH = 1 (BIOS)

- Kiểm tra vùng đệm bàn phím
- ZF = 1, rỗng
- ZF = 0, không rỗng
- Xoá vùng đệm: + INT 16h, AH = 0

BÀN PHÍM

INT 16h, AH = 2 (BIOS)

- Xác định trạng thái các phím điều khiển
- AL(b) = 1: đã nhấn bit b
- AL(b) = 0: chưa nhấn bit b

LẬP TRÌNH XỬ LÝ PHÍM

Kiểm tra trạng thái các phím Ctrl, Alt, Shift :

Chúng ta có thể đọc trực tiếp từ địa chỉ 0:0417 hoặc lấy trong AL thông qua hàm AH = 2 ngắt 16H.

Cách đọc trực tiếp

XOR AX, AX

MOV ES, AX

MOV AL, ES:[417H]

TEST AL, 01H

; kiểm tra phím Shift

JNZ SHIFT_DANGNHAN

TEST AL, 04H

; kiểm tra phím Ctrl

JNZ CTRL_DANGNHAN

TEST AL, 08H

; Kiểm tra phím Alt

JNZ ALT_DANGNHAN

3. Kiểm tra và thiết lập trạng thái các phím Caps/Num/Scroll Lock tương tự nhưng với mã scan khác Scroll = 10H, Num = 20H, Cap = 40H.

4. Đặt lại các trạng thái đèn Caps/Num/Scroll Lock, ta chỉ cần đặt lại giá trị ở địa chỉ 0:0417.
Vd, để bật đèn Caps Lock và đổi trạng thái đèn Num Lock ta sẽ làm như sau:

```
XOR    AX, AX
MOV    ES, AX      ; ES = 0
MOV    AL, ES:[417H] ; đọc trạng thái đèn
OR     AL, 40H      ; bật đèn Caps Lock
XOR    AL, 20H      ; đảo đèn Nums Lock
MOV    ES:[417H], AL ;
MOV    AH, 2H
INT    16H
```

BÀI TẬP LẬP TRÌNH

BÀI 1 : VIẾT ĐOẠN CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

- CUỘN WINDOW TỪ HÀNG 5, CỘT 10 TỚI HÀNG 20 CỘT 70 VỚI THUỘC TÍNH MÀN HÌNH ĐẢO.
- ĐỊNH VỊ CURSOR TẠI HÀNG 10, CỘT 20
- HIỂN THỊ DÒNG TEXT “ DAY LA 1 DONG TEXT TRONG WINDOW ”
- SAU KHI XUẤT TEXT ĐỢI NHẤN 1 PHÍM.
- CUỘN WINDOW TỪ HÀNG 5, CỘT 15 TỚI HÀNG 18 CỘT 68 VỚI THUỘC TÍNH THƯỜNG.
- XUẤT KÝ TỰ A VỚI THUỘC TÍNH NHẤP NHÁY TẠI GIỮA WINDOW.
- ĐỢI GỎ 1 PHÍM, XÓA TOÀN BỘ MÀN HÌNH..

BÀI TẬP LẬP TRÌNH

BÀI 2 : VIẾT CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

- XUẤT CHUỖI “GO VAO 1 KÝ TU THUONG : ‘.
- KHI USER GỎ 1 KÝ TỰ (KHI GỎ KHÔNG HIỂN THI KÝ TỰ GỎ RA MÀN HÌNH) , ĐỔI KÝ TỰ NÀY THÀNH CHỮ HOA RỒI XUẤT RA MÀN HÌNH.GIẢ SỬ CHỈ NHẬP CÁC KÝ TỰ HỢP LỆ.
- KHI GỎ KÝ TỰ MỞ RỘNG SẼ THOÁT VỀ DOS, NHƯNG CÓ LẼ BẠN CÒN NHÌN THẤY 1 KÝ TỰ XUẤT THÊM TRÊN MÀN HÌNH. GIẢI THÍCH.

1. Viết chương trình để :
 - a. Xoá màn hình, tạo kích thước to nhất cho con trỏ và di chuyển nó đến góc trái trên
 - b. Nếu nhấn phím Home : chuyển con trỏ đến góc trái trên, End : chuyển đến góc trái dưới, Page Dn : chuyển con trỏ đến góc phải dưới, Esc : kết thúc chương trình.
2. Dịch chuyển con trỏ đến góc trái trên màn hình nếu phím F1 được nhấn, góc trái dưới nếu phím F2 được nhấn. Chương trình sẽ bỏ qua các kí tự thông thường.
3. Viết chương trình soạn thảo văn bản như sau :
 - a. Xoá màn hình, định vị con trỏ tại đầu dòng 12
 - b. Để người sử dụng đánh vào các kí tự. Con trỏ dịch chuyển đi sau khi hiển thị kí tự nếu nó không ở tại lề phải của màn hình
 - c. Phím mũi tên trái , phải, lên , xuống dịch con trỏ tương ứng
 - d. Phím Insert : chèn kí tự, Delete : Xoá một kí tự , Esc : kết thúc chương trình.

BÀI TẬP LẬP TRÌNH

BÀI 3 : VIẾT CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

CHO PHÉP VẼ ĐƠN GIẢN NHỜ CÁC PHÍM MŨI TÊN TRÊN BÀN PHÍM ĐỂ DI CHUYỂN THEO HƯỚNG MONG MUỐN.

PHẢI BẢO ĐẢM XUẤT CÁC KÝ TỰ GÓC THÍCH HỢP.

BIẾT RẰNG MÃ ASCII CỦA 1 SỐ KÝ TỰ : xem bảng mã ASCII

MÃ SCAN CODE CỦA CÁC PHÍM MŨI TÊN :

TRÁI 4BH PHẢI 4DH LÊN 48H XUỐNG 50H

LẬP TRÌNH XỬ LÝ ĐĨA&FILE

- CƠ BẢN VỀ LƯU TRỮ TRÊN ĐĨA TÙ.
- MỘT ỨNG DỤNG HIỂN THỊ SECTOR
- MỘT ỨNG DỤNG HIỂN THỊ CLUSTER.
- CÁC CHỨC NĂNG VỀ FILE Ở MỨC HỆ THỐNG.
- QUẢN LÝ ĐĨA VÀ THƯ MỤC.
- TRUY XUẤT ĐĨA VỚI INT 13H CỦA ROMBIOS
- BÀI TẬP
- GIỚI THIỆU FILE VÀ LẬP TRÌNH XỬ LÝ FILE

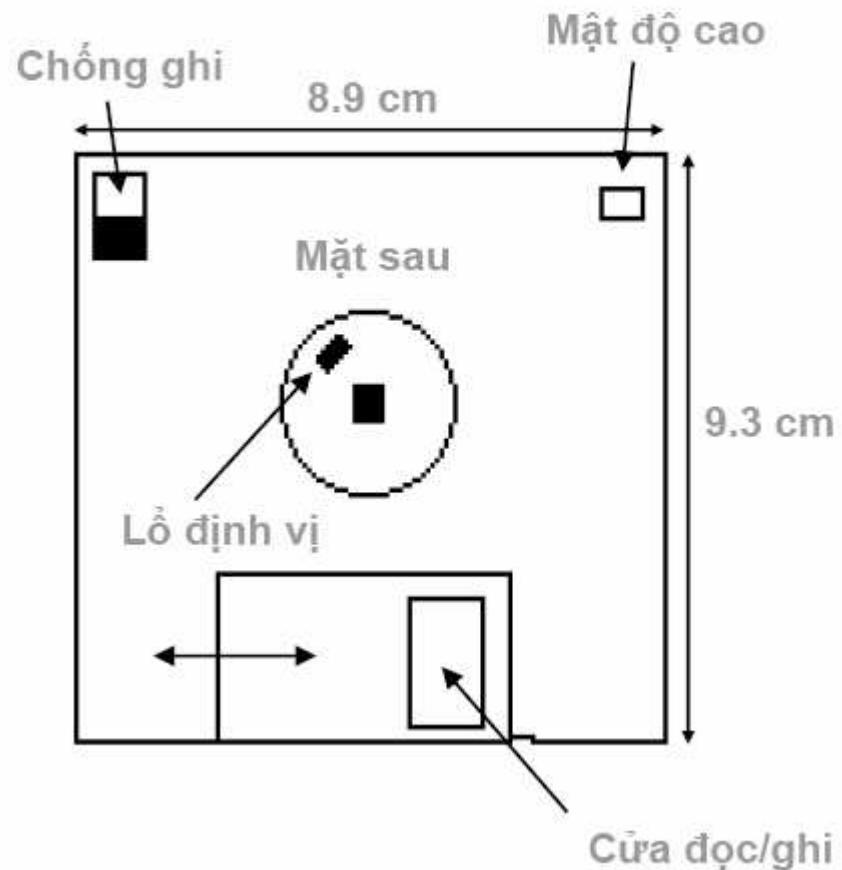
CƠ BẢN VỀ LƯU TRỮ TRÊN ĐĨA TỪ

Ngôn ngữ ASM vượt trội hơn các ngôn ngữ khác về khả năng xử lý đĩa.

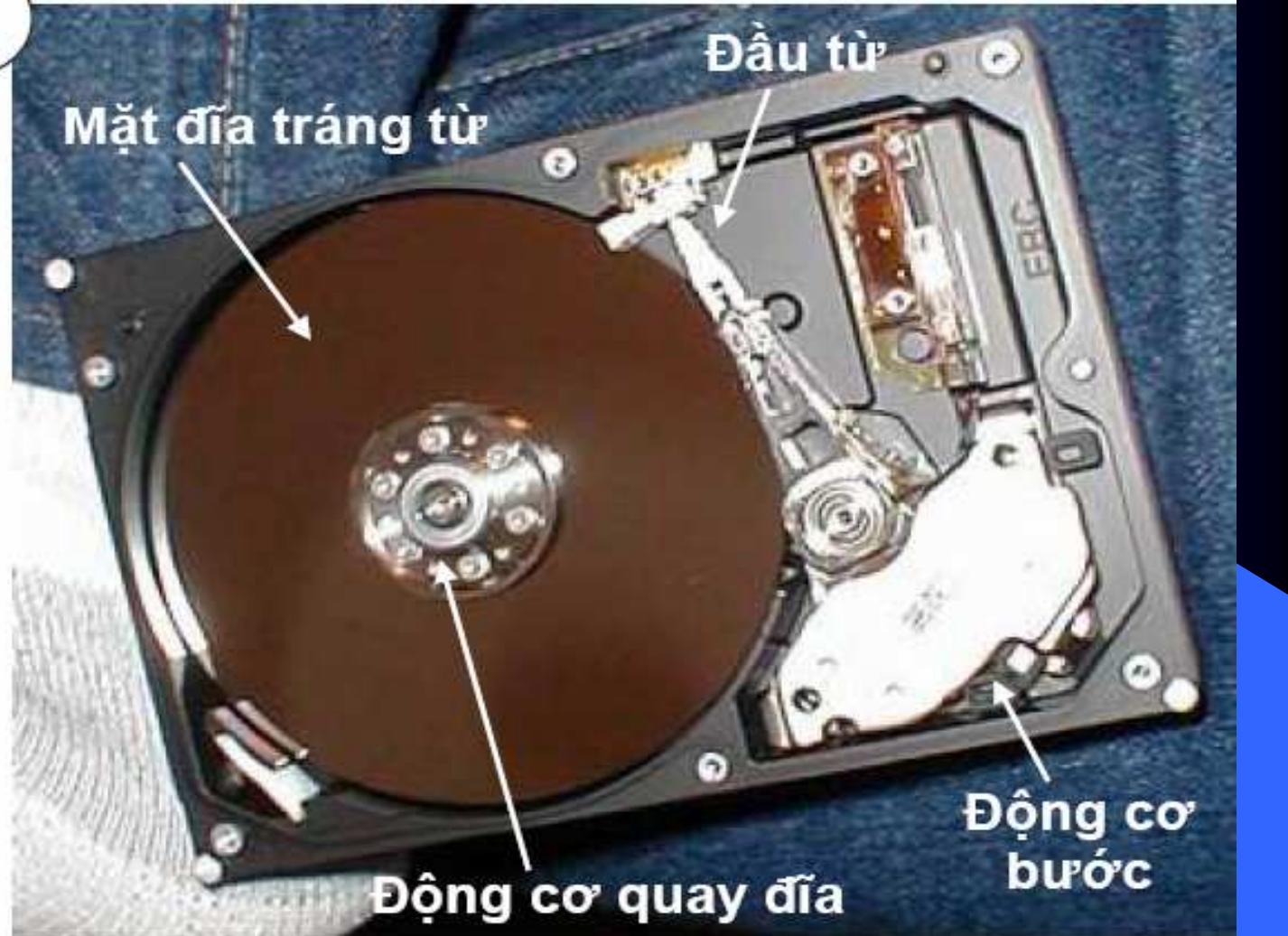
Ta xem xét việc lưu trữ thông tin trên đĩa theo 2 mức độ : mức phần cứng/BIOS và mức phần mềm/DOS.

- **mức phần cứng :**lưu trữ thông tin liên quan đến cách dữ liệu được lưu trữ 1 cách vật lý như thế nào trên đĩa từ?
- **mức phần mềm :** việc lưu trữ được quản lý bởi tiện ích quản lý File của HĐH DOS.

Đĩa mềm 3.5"



Đĩa cứng



CÁC ĐẶC TÍNH LUẬN LÝ & VẬT LÝ CỦA ĐĨA TÙ

Ở mức vật lý : đĩa được tổ chức thành các Tracks, Cylinders, Sectors.

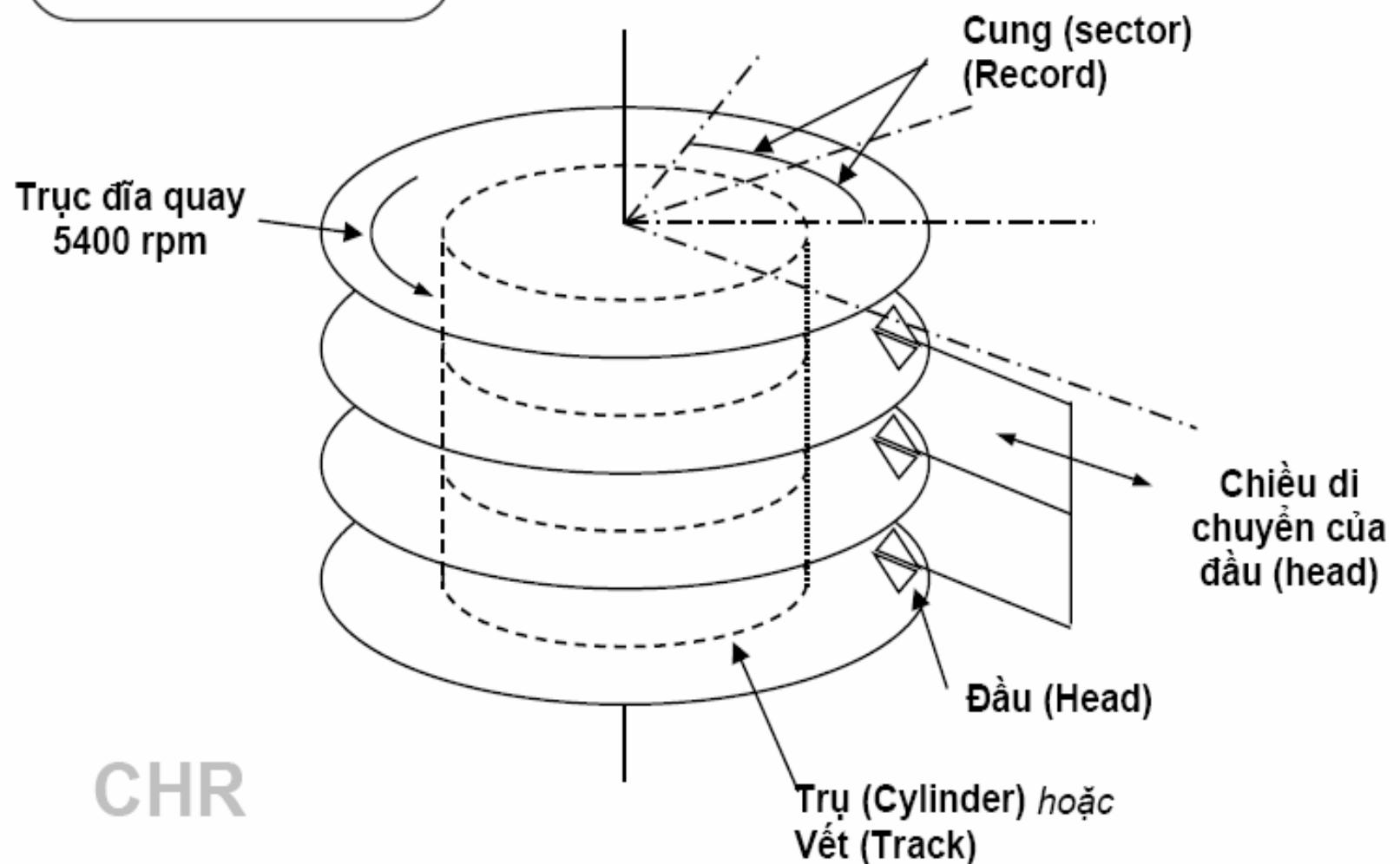
→ **Khả năng lưu trữ của đĩa được mô tả bằng 3 thông số :**

C (cylinder number)

H (Head side)

R (sector number)

Phân chia địa vật lý



CHR

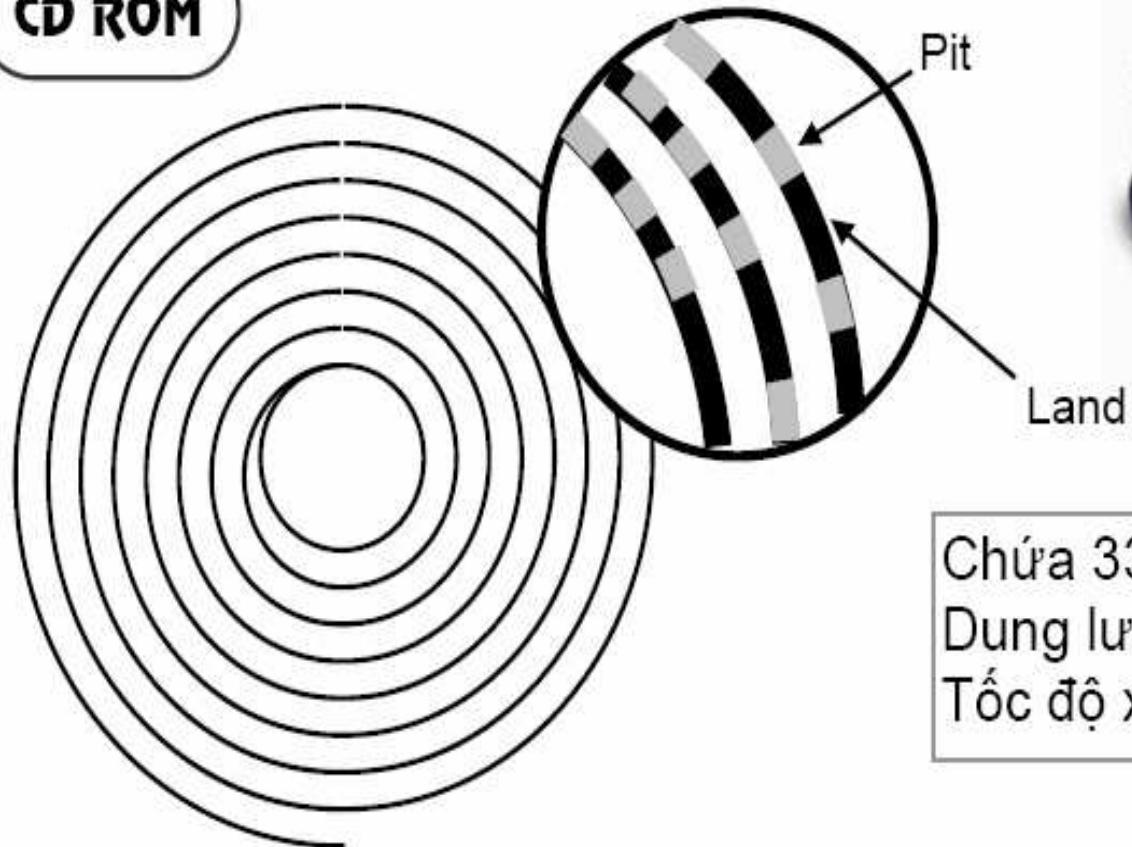
CÁC KHÁI NIỆM TRACK, CYLINDER, SECTOR

Tracks : là các vòng tròn đồng tâm được tạo ra trên bề mặt đĩa.

Cylinder : tập các tracks cùng bán kính trên 1 ch่อง đĩa. Mặt đĩa có bao nhiêu track thì sẽ có bấy nhiêu Cylinder.

Sector : là 1 đoạn của track (cung từ) có khả năng lưu trữ 512 bytes dữ liệu.
Các sector được đánh số bắt đầu từ 1 trên mỗi track → trên 1 đĩa tồn tại nhiều sector cùng số hiệu.

CD ROM



Chứa 330.000 khối dữ liệu.
Dung lượng 650 MB / 74 min
Tốc độ x1 = 153.60 KByte/s

Thông tin ghi theo rãnh (track) hình xoắn ốc
Dùng tia laser đục lỗ 1 µm trên rãnh gọi là Pit.
Phần không bị đục lỗ trên rãnh gọi là Land.

Ở mức luận lý : đĩa được tổ chức thành các Clusters, các files mà DOS sẽ dùng để cấp phát vùng lưu trữ cho dữ liệu cần lưu trữ.

Cluster : là 1 nhóm gồm 2,4,6 các sector kề nhau. Đó chính là đơn vị cấp phát vùng lưu trữ cho dữ liệu (file). Các cluster được đánh số bắt đầu từ 0.

**Nếu dữ liệu cần lưu trữ chỉ 1 byte thì hệ điều hành cũng cấp phát 1 cluster.
số bytes/cluster hay sector/cluster tùy thuộc vào từng loại đĩa.**

TƯƠNG QUAN GIỮA SECTOR VẬT LÝ VÀ SECTOR LOGIC TRÊN ĐĨA MỀM

MẶT ĐĨA	TRACK	SECTOR	SECTOR LOGIC	THÔNG TIN
0	0	1	0	BOOT RECORD
0	0	2-5	1-4	FAT
0	0	6-9	5-8	Thư mục gốc
1	0	1-3	9-11	Thư mục gốc
1	0	4-9	12-17	Dữ liệu
0	1	1-9	18-26	Dữ liệu

BAD SECTOR

Trên bề mặt đĩa có thể tồn tại các sector mà HĐH không thể ghi dữ liệu vào đó hoặc không thể đọc dữ liệu từ đó. Các sector này gọi là Bad Sector.



Làm sao biết sector nào là bad sector

Kiểm tra giá trị của các phần tử (entry) trong bảng FAT, phần tử nào chứa giá trị (F)FF7H thì cluster tương ứng bị Bad

BẢNG FAT FILE ALLOCATION TABLE

DOS quản lý các File nhờ vào 1 bảng gọi là bảng FAT.

Trong bảng FAT có ghi cluster bắt đầu của File này ở đâu ? Và đĩa còn bao nhiêu Clusters trống chưa cấp phát.

tổ chức luận lý của đĩa được mô tả như hình sau :



Thí dụ về bảng FAT

Đĩa mềm 3.5"" 360K thì :

Sector 0 : boot sector

Sector 1-4 : bảng FAT

Sector 5 – 11 : thư mục gốc

Sector 12-719 : vùng chứa data

BOOT RECORD

- Còn được gọi là Boot Sector. Ổ đĩa cứng gọi là Master boot, là Sector đầu tiên khi đĩa được format.
- chứa 1 chương trình nhỏ cho biết dạng lưu trữ trên đĩa và tên hệ thống MT, kiểm tra xem có các file hệ thống IO.SYS, MSDOS.SYS, COMMAND.COM hay không ?
- nếu có thì nạp chúng vào bộ nhớ (gọi là chương trình mồi của HĐH)

BOOT RECORD (tt)

- **Tọa độ vật lý :**
C=0, H=0, R =1 (COH0R1) tức ở tại sector đầu tiên của track đầu tiên, mặt trên của đĩa đầu tiên trong ổ đĩa cứng.
- Trong Master boot có chứa bảng PARTITION TABLE cho biết tầm địa chỉ vật lý (dung lượng) của ổ đĩa luận lý.

Master boot không thuộc Partition nào

BOOT RECORD (tt)

- **BOOT RECORD** được ROM BIOS nạp vào địa chỉ 0000:7C00H.
- Nếu máy không bị Virus thì lệnh đầu tiên của chương trình **BOOT** là JMP 7C3EH, nghĩa là nhảy đến chương trình nạp mồi.
- **chương trình nạp mồi (Bootstrap Loader)** nạp thành phần cốt lõi của DOS lên RAM trong quá trình khởi động MT.

THÔNG TIN TRONG MASTER BOOT

BYTE ĐẦU	SỐ BYTES	THÔNG TIN
00H	3	chỉ thị nhảy về nói chứa CT nạp mỗi
03H	8	Tên nhà sản xuất và hệ điều hành
0BH	2	Bytes/sector
0DH	1	Sector/block (mỗi block ≥ 1 sector)
0EH	2	Số lượng Sectors không dùng đến kể từ sector 0.
10h	1	Số lượng bảng FAT

THÔNG TIN TRONG MASTER BOOT

BYTE BĐẦU SỐ BYTES

THÔNG TIN

11H	2	Số Entry của thư mục gốc ổ đĩa.
13H	2	Tổng số sector của ổ đĩa logic này.
15H	1	Byte mô tả
16H	2	Số sector cho 1 bảng FAT
18H	2	Số Sectors trong 1 track.
1AH	2	Số lượng đầu đọc
1CH	4	Số lượng sector ẩn
20H	4	Tổng số sectors

THÔNG TIN TRONG MASTER BOOT

BYTE ĐẦU SỐ BYTES

THÔNG TIN

3EH		Bootstrap
....		
1BEH	64	PARTITION TABLE
.....		
1FEH	1	Giá trị 55H
1FFH	1	Giá trị 0AAH

THÔNG TIN TRONG MASTER BOOT

Từ thông tin trong bảng FORMAT, ta tính được địa chỉ của bảng FAT1, FAT2, Thư mục gốc ổ đĩa, địa chỉ bắt đầu của vùng dữ liệu.

BẢNG FAT

- Bảng chứa các danh sách liên kết các clusters. Mỗi danh sách trong bảng cho DOS biết rằng các clusters nào đã cấp phát, các clusters nào chưa dùng.
- tùy theo ổ đĩa có thể có 1 hay 2 bảng FAT, bảng FAT2 để dự phòng.
- có 2 loại bảng FAT :
 - bảng có Entry 12 bit cho đĩa mềm.
 - bảng có Entry 16 bit cho đĩa cứng.

PARTITION TABLE

64 Bytes của Partition table được chia làm 4, mỗi phần 16 bytes mô tả cho 1 partition các thông tin sau :

Bytes	Mô tả
00H	active flag (=0 Non bootable =80H Bootable)
01H	starting head – Nơi bắt đầu Partition
02H	starting cylinder

PARTION TABLE

Bằng FDISK của HĐH ta có thể chia không gian lưu trữ của đĩa cứng thành các phần khác nhau gọi là Partition.

DOS cho phép tạo ra 3 loại Partition :

Primary Dos, Extended Dos và None Dos

Ta có thể cài đặt các HĐH khác nhau lên các Partition khác nhau.

PARTITION TABLE

03H	starting sector
04H	partition type :
	0 Non Dos
	1 cho đĩa nhỏ 12 bit FAT Entry
	4 cho đĩa lớn 16 bit FAT Entry
	5 Extended Dos
05H	Ending nơi kết thúc Partition
06H	Ending Cylinder
07H	Ending Sector
08H, 0BH	Starting sector for partition
0Ch,0FH	Partition length in sectors

Một số thí dụ

kiểm tra Partition Active

- đọc sector đầu tiên của đĩa cứng lưu vào biến.
- kiểm tra offset 00 của 4 phần tử Partition trong Partition Table

MOV CX, 4

MOV SI, 1BEH

PACTIVE :

MOV AL, MBOOT [SI]

CMP AL, 80H

JE ACTIVE

ADD SI, 16

LOOP PACTIVE

NO_ACTIVE :

.....

ACTIVE :

Một số thí dụ

Đọc nội dung của BootSector ghi vào biến dem

- đọc sector đầu tiên của đĩa cứng lưu vào buffer.
- tìm partition active (phần tử trong bảng partition có offset 80h)
- đọc byte tại offset 01h và word tại offset 02h của phần tử partition tương ứng ở trên (head, sector, cylinder) để xác định số hiệu bắt đầu của partition active → boot sector của đĩa cứng.
- đọc nội dung của sector đọc được ở trên lưu vào buffer.

Một số thí dụ

ACTIVE :

MOV AX, 0201H ; đọc 1 sector

MOV CX, WORD PTR MBOOT [SI+2] ; sector cylinder

MOV DH, BYTE PTR MBOOT[SI+1] ; head

MOV DL, 80H ; đĩa cứng

MOV ES, CS ; trỏ về đầu vùng buffer lưu

LEA BX, BUFFER

INT 13H

THƯ MỤC GỐC (ROOT DIRECTORY)

- Là danh sách tất cả các Files đã có trên đĩa, các thư mục cấp 1 đã có.
- Mỗi phần tử (32 bytes) trong bảng thư mục sẽ chứa thông tin về tên file hoặc là thư mục, kích thước, thuộc tính, cluster bắt đầu của file này hoặc cluster bắt đầu của thư mục thứ cấp (thư mục con).

mỗi bảng thư mục chứa tối đa 112 entry, mỗi entry là 32 bytes.

THƯ MỤC GỐC (ROOT DIRECTORY)

Offset	Nội dung	Kích thước
00H	tên chính của File	8 bytes
08H	phần mở rộng của tên file	3 bytes
0BH	thuộc tính của File	1 byte
0CH	dự trữ	10 bytes
16H	giờ thay đổi thông tin cuối cùng	2 bytes
18H	ngày thay đổi thông tin cuối cùng	2 bytes
1Ah	cluster đầu tiên của File	2 bytes
1CH	Kích thước File	4bytes

BYTE THUỘC TÍNH

x	x	a	d	v	s	h	r
---	---	---	---	---	---	---	---

x : không sử dụng

a : thuộc tính lưu trữ (Archive)

d : thuộc tính thư mục con (Sub - Directory)

v : thuộc tính nhãn đĩa (Volume)

s : thuộc tính hệ thống (System)

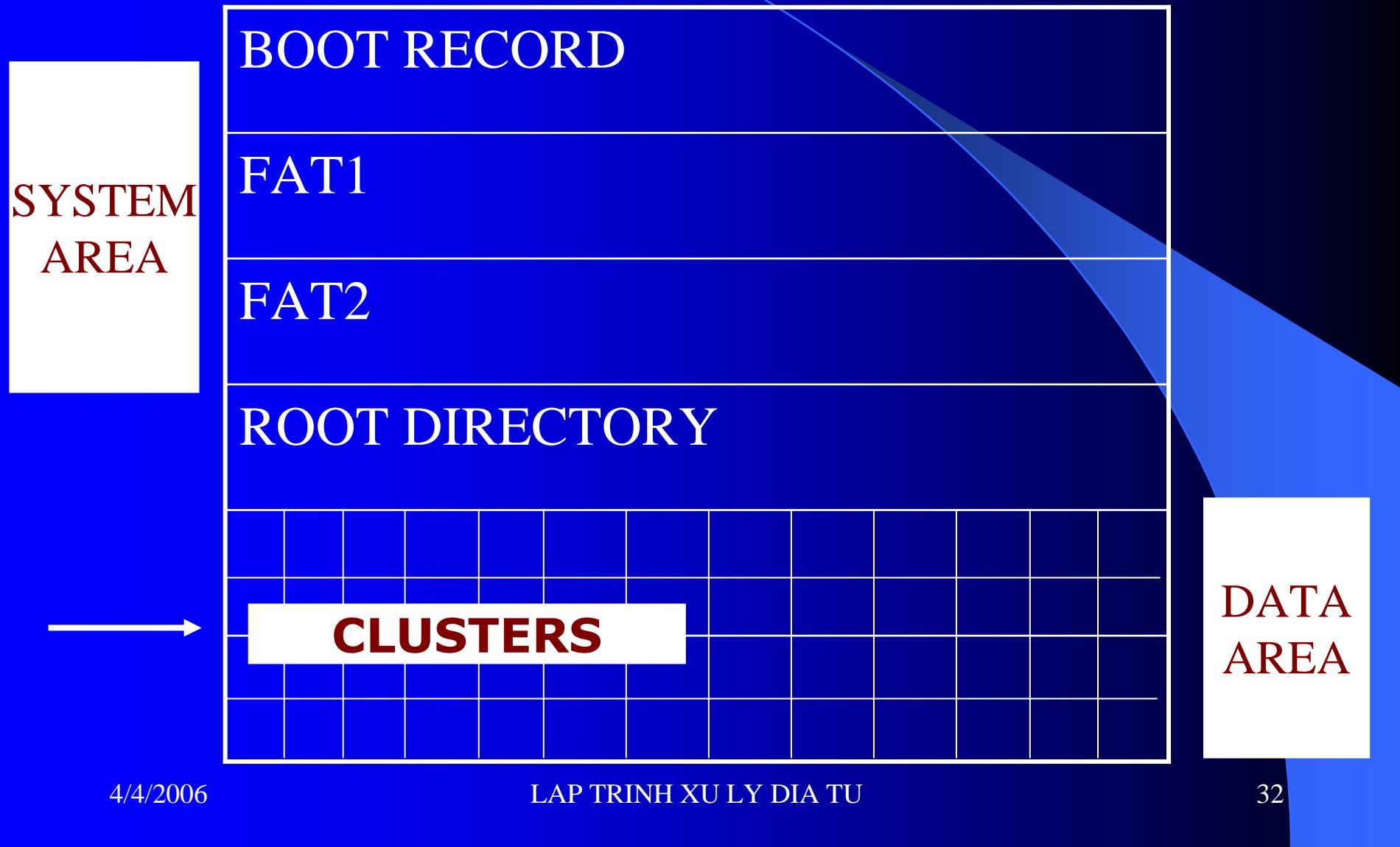
h : thuộc tính ẩn (Hidden)

r : thuộc tính chỉ đọc (Read Only)

VÙNG LƯU TRỮ

- là vùng dành cho việc lưu trữ dữ liệu.
- như vậy việc lưu trữ dữ liệu trên đĩa có cấu trúc là 1 danh sách liên kết mà bảng thư mục gốc là đầu của danh sách liên kết.
- đầu mỗi cluster luôn luôn chứa địa chỉ của cluster sau nó cho biết phần còn lại của file là cluster nào. Nếu giá trị này là 0 thì cluster này là cluster cuối cùng.

SỰ PHÂN VÙNG TRÊN ĐĨA



CÁC LOẠI ĐĨA

Disk Type	sides	track per side	sectors per track	total sector	cluster size	total bytes
		side	track			
360K	2	40	9	720	1,024	368,640
720K	2	80	9	1,440	512	737,280
1.2MB	2	80	15	2,400	512	1,228,800
1.4MB	2	80	18	2,880	512	1,474,560
32MB	6	614	17	62,610	2,048	32,056,832

TÍNH DUNG LƯỢNG ĐĨA

Công thức tính dung lượng đĩa :

Dung lượng đĩa (bytes) = số byte/1 sector
* số sector/1 track * số track/ 1 mặt đĩa *
số mặt đĩa.

MỘT SỐ HÀM THAO TÁC VỚI FILE VÀ ĐĨA INT 21H

HÀM 36H INT 21H :

Lấy số bytes còn trống trên đĩa

Input :

AH = 36H DL = 063 đĩa (0 : mặc định, 1 ổ A

Output :

Có lỗi AX = 0FFFFH

Không lỗi : AX = số sector / cluster

BX = số cluster còn trống

DX = tổng số cluster trên đĩa

CX = số bytes/cluster



BÀI TẬP

Viết chương trình tạo thư mục với yêu cầu tên thư mục (có thể bao gồm tên ổ đĩa, đường dẫn và tên thư mục) được nhập từ bàn phím, cho phép sửa sai khi gõ nhầm tên thư mục.

Viết chương trình ghi dữ liệu vào file với yêu cầu :

- Tên file nhập từ bàn phím
- Dữ liệu ghi vào file cũng gõ từ bàn phím và kết thúc việc nhập bằng phím **CTRL+Z**

Viết chương trình gộp nội dung 1 file vào cuối 1 file khác.

LẬP TRÌNH XỬ LÝ FILE

GiỚI THIỆU FILE
CÁC HÀM CHỨC NĂNG XỬ LÝ FILE
CỦA INT 21H CỦA DOS

GIỚI THIỆU FILE

- Trong quản lý File, Dos vay mượn khái niệm Handle trong HĐH Unix để truy xuất File và thiết bị.

HANDLE



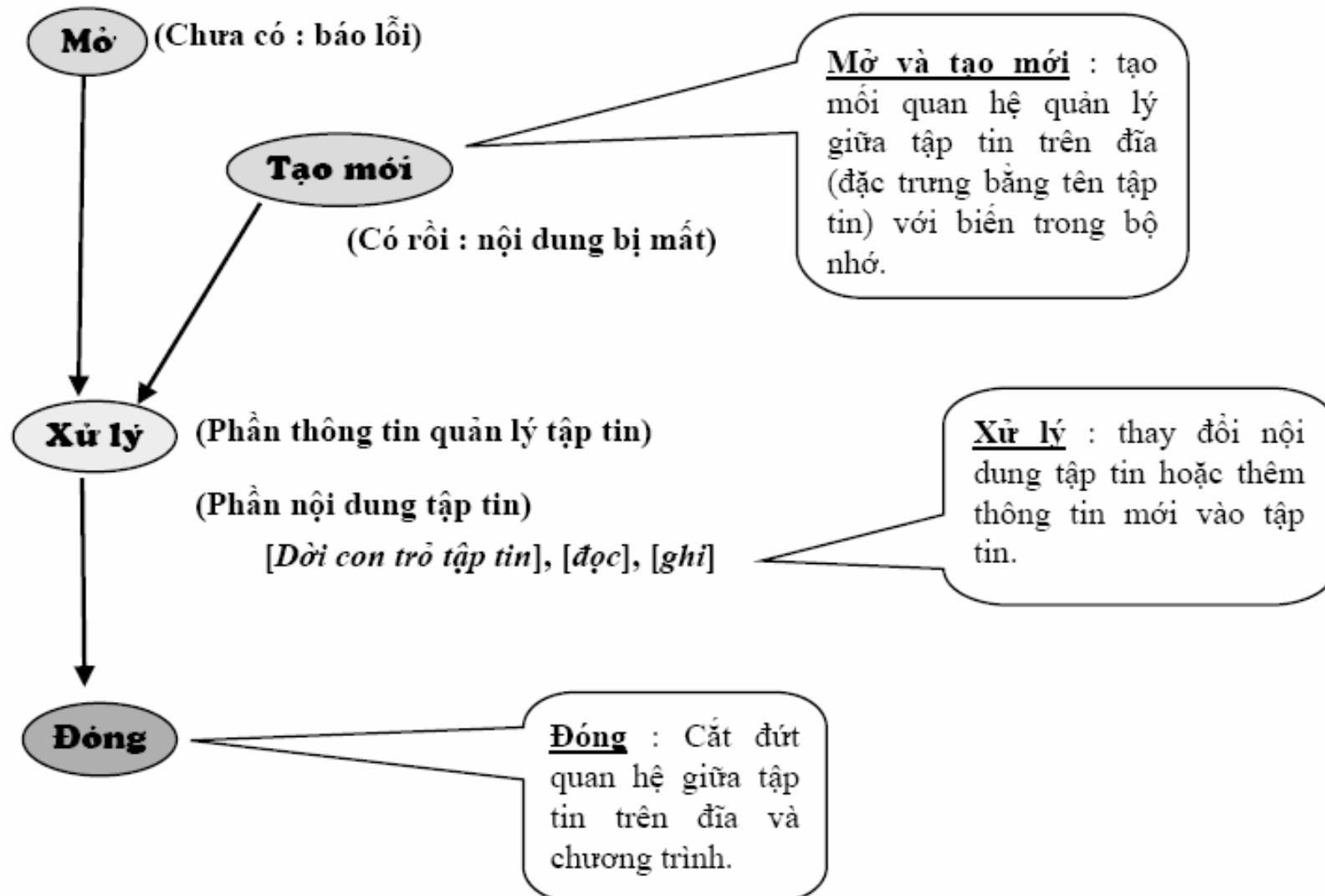
- Handle là 1 số 16 bits được Dos sử dụng để nhận biết File đã mở hoặc 1 thiết bị trong hệ thống.

GIỚI THIỆU FILE

- Có 5 Handle thiết bị chuẩn được Dos nhận dạng.

Handle	Thiết bị
0	Keyboard, standard input
1	Console, standard output
2	Error output thiết bị xuất lỗi – màn hình
3	Auxiliary device asynchronous
4	Printer

CÁC THAO TÁC XỬ LÝ FILE



CÁC CHỨC NĂNG CƠ BẢN VỀ XỬ LÝ FILE CỦA INT 21H

Chức năng

CÁC CHỨC NĂNG
NÀY PHẢI ĐƯA
VÀO AH

Tác vụ

- | | |
|-----|--|
| 3Ch | Tạo File mới |
| 3Dh | Mở File đã có để xuất/nhập/vừa nhập vừa xuất |
| 3Eh | Đóng thẻ File |
| 3Fh | Đọc từ File hay đọc từ thiết bị 1 số bytes định trước |
| 40h | Ghi vào File hay đọc từ thiết bị 1 số bytes định trước |
| 42h | di chuyển con trỏ File trước khi đọc/ ghi |

CHỨC NĂNG TẠO FILE 3Ch

CREATE FILE FUNCTION

3Ch

Chức năng : Mở 1 File mới để đọc ghi. Nếu file đã có thì file cũ sẽ bị xóa.

AH = 3Ch

DS:DX địa chỉ của tên File muốn mở (ASCII String)

CX = thuộc tính File

(0 normal 1 ReadOnly 2 Hidden 4 System)

Xuất : không lỗi CF =0 AX = File Handle Có lỗi CF =1.

Mã lỗi trong AX (3,4,5).

CHỨC NĂNG TẠO FILE 3Ch CREATE FILE FUNCTION 3Ch

Ex :

CREATE_FILE :

MOV AH, 3CH

MOV DX, OFFSET NEWFILE

MOV CX, 0

INT 21H

JC DISPLAY_ERROR

MOV NEWFILEHANDLE, AX

...

NEWFILE DB ‘ FILE1.DOC ’,0

NEWFILEHANDLE DW ?

CHỨC NĂNG TẠO FILE 3Ch CREATE FILE FUNCTION 3Ch

Ex :

CHỨC NĂNG 3Ch CÓ 1 KHUYẾT ĐIỂM LÀ NẾU CÓ 1 FILE CÙNG TÊN(CÙNG ĐƯỜNG DẪN) ĐÃ TỒN TẠI THÌ FILE CŨ SẼ BỊ XÓA.

ĐỂ BẢO VỆ FILE, CÓ 2 CÁCH :

C1 : MỞ FILE BẰNG CHỨC NĂNG 3Dh, NẾU FILE CHƯA CÓ THÌ TRẢ VỀ LỖI SỐ 2 (FILE NOT FOUND) → YÊN TÂM MỞ FILE MỚI.

C2 : DÙNG CHỨC NĂNG 5Bh MỞ FILE CÓ KIỂM TRA TÊN FILE NÀY ĐÃ CÓ CHƯA.

CHỨC NĂNG 5Bh TẠO FILE MỚI CÓ KIỂM TRA

ĐIỀU KIỆN : GIỐNG CHỨC NĂNG 3Ch

NẾU FILE NÀY ĐÃ CÓ THÌ KHÔNG MỞ FILE MỚI MÀ TRẢ VỀ LỖI 50h

CREATE_FILE :

MOV AH,5BH

MOV DX, OFFSET FILENAME

MOV CX, 0

INT 21H

JC ERROR

....

FILENAME DB 'FILE1.DOC' , 0

CÁC LỖI KHI MỞ FILE

MÃ LỖI

DIỄN GIẢI

- 2 FILE NOT FOUND KHÔNG TÌM THẤY FILE, CÓ THỂ ĐƯỜNG DẪN KHÔNG ĐÚNG HOẶC TÊN FILE MÔ TẢ KHÔNG HỢP LỆ.
- 3 PATH NOT FOUND ĐƯỜNG DẪN KHÔNG CÓ.
- 4 TOO MANY OPEN FILES CÓ THỂ DO LỆNH PATH XX TRONG CONFIG.SYS QUÁ NHỎ KHÔNG CHO PHÉP MỞ NHIỀU FILE.
- 5 ACCESS DENIED TỪ CHỐI TRUY XUẤT. CÓ THỂ TA MUỐN XÓA FILE ĐANG MỞ, HAY FILE NÀY CÓ THUỘC TÍNH CHỈ ĐỌC.

CH Mã truy nhập không hợp lệ.

FH Ở ổ đĩa không hợp lệ

10h Đang tìm cách xóa thư mục hiện thời

CÁC LỖI KHI MỞ FILE

MÃ LỖI

DIỄN GIẢI

11H Không cùng thiết bị

12H Không tìm được thêm File nào

CHỨC NĂNG MỞ FILE ĐÃ CÓ 3Dh Int 21h OPEN FILE

ĐIỀU KIỆN :

AH = 3DH DS:DX ĐỊA CHỈ TÊN FILE

AL = MODE

0: INPUT (MỞ CHỈ ĐỌC)

1 : OUTPUT (MỞ ĐỂ GHI)

2 : INPUT OUTPUT (MỞ VÙA ĐỌC VÙA GHI)

XUẤT :

KHÔNG LỖI CF = 0 AX = FILE HANDLE

CÓ LỖI CF = 1 AX ← mã lỗi (2,4,512)

MỞ FILE HÀM 3CH INT 21H

- Trước khi sử dụng 1 file, ta phải mở nó.
- Để tạo 1 file mới hay ghi lại 1 file cũ, ta sử dụng tên file và thuộc tính của File.
- → DOS trả về thẻ file

MỞ FILE HÀM 3CH INT 21H

AH = 3CH

DS:DX địa chỉ của chuỗi ASCII
(chuỗi tên File kết thúc bằng byte 0)

CL = thuộc tính File

Nếu thành công, AX = thẻ File

Nếu CF được set thì có lỗi, mã lỗi chứa trong AX
(lỗi 3,4,5)

**Viết code mở 1 File mới với thuộc tính chỉ đọc,
tên File là FILE1**

Fname DB ‘FILE1’,0

FHANDLE DW ?

MOV AX,@DATA

MOV DS,AX

MOV AH,3CH

MOV CL,1

LEA DX,FNAME

INT 21H

MOV FHANDLE, AX

JC OPEN_ERROR

.....

CHỨC NĂNG MỞ FILE ĐÃ CÓ SẴN

HÀM 3Dh INT 21H

OPEN FILE

AH = 3DH

DS:DX = địa chỉ của chuỗi ASCII
(chuỗi tên File kết thúc bằng byte 0)

AL = mã truy cập

0 : mở để đọc

1 : mở để ghi

2 : mở để đọc và ghi

→ Thành công, AX = Fhandle

→ Có lỗi. Mã lỗi chứa trong AX (2,4,5,12)

CHỨC NĂNG MỞ FILE ĐÃ CÓ SẴN

HÀM 3Dh INT 21H

OPEN FILE

```
MOV AH, 3DH  
MOV AL, 0  
MOV DX, OFFSET FILENAME  
INT 21H  
JC DISPLAY_ERROR  
MOV INFILEHANDLE, AX  
.....  
INFILE DB 'D:\FILE1.DOC', 0  
INFILEHANDLE DW ?
```

CHỨC NĂNG 3EH ĐÓNG FILE

ĐIỀU KIỆN :

AH = 3EH BX = FILE HANDLE CÂN ĐÓNG

XUẤT :

KHÔNG LỖI CF = 0 CÓ LỖI CF = 1

EX :

MOV AH, 3EH

MOV BX, INFILEHANDLE

INT 21H

JC DISPLAY_ERROR

.....

INFILE DB 'D:\FIEL1.DOC', 0

INFILEHANDLE DW ?

LỖI SỐ 6 : INVALID HANDLE

FILE HANDLE TRONG BX

**KHÔNG PHẢI LÀ THẺ FILE CỦA
FILE ĐÃ MỞ.**

CHỨC NĂNG 3FH ĐỌC FILE

ĐỌC 1 SỐ BYTES TỪ FILE LUU VÀO BỘ NHỚ

ĐIỀU KIỆN :

AH = 3FH BX = FILE HANDLE , CX = SỐ BYTES CẦN ĐỌC

DS:DX : ĐỊA CHỈ BỘ ĐỆM.

XUẤT :

AX = SỐ BYTES ĐỌC ĐƯỢC, NẾU AX = 0 HAY AX < CX FILE ĐÃ KẾT THÚC.

NẾU CỜ CF ĐƯỢC LẬP \rightarrow CÓ LỖI, MÃ LỖI CHỦA TRONG AX(5,6)

CHỨC NĂNG 3FH ĐỌC FILE

EX : ĐỌC 1 SECTOR 512 BYTES TỪ FILE

.DATA

HANDLE DW ?

BUFFER DB 512

DUP(?)

MOV AX, @DATA

MOV DS, AX

MOV AH, 3FH

MOV CX, 512

MOV BX, HANDLE

MOV CX, 512

INT 21H

JC READ_ERROR

NẾU CẦN ĐỌC HẾT CÁC SECTOR CHO
ĐẾN HẾT FILE → EOF

CMP AX, CX

JL EXIT

JMP READ_LOOP

CHỨC NĂNG 40H GHI FILE

GHI 1 SỐ BYTES LÊN FILE HAY THIẾT BỊ

INPUT :

AH =**40H** BX = THẺ FILE CX = SỐ BYTES CẦN GHI
DS:DX : ĐỊA CHỈ VÙNG ĐỆM.

OUTPUT :

AX : SỐ BYTES GHI ĐƯỢC, NẾU AX<CX, CÓ LỖI (ĐĨA
ĐẦY).NẾU CF ĐƯỢC LẬP → CÓ LỖI, MÃ LỖI TRONG AX
(5,6).

HÀM 40H CŨNG CÓ THẺ DÙNG ĐỂ ĐƯA DỮ LIỆU RA MÀN HÌNH

CON TRỎ FILE

- DÙNG ĐỂ ĐỊNH VỊ TRONG FILE.
- KHI FILE ĐƯỢC MỞ, CON TRỎ FILE NẰM Ở ĐẦU FILE.
- SAU MỖI THAO TÁC ĐỌC, CON TRỎ FILE SẼ DI CHUYỂN ĐẾN BYTE KẾ.
- SAU KHI GHI 1 FILE MỚI CON TRỎ CHỈ ĐẾN CUỐI FILE (EOF).
- ĐỂ DI CHUYỂN CON TRỎ FILE HÀM 42H

MINH HỌA LẬP TRÌNH FILE

Viết chương trình cho phép User gõ vào tên File (có thể có kèm theo tên ổ đĩa, thư mục chứa file), chương trình sẽ đọc và hiển thị nội dung File ra màn hình.



DỊCH CHUYỂN CON TRỎ FILE HÀM 42H INT 21H

AH = 42H AL = PHƯƠNG THỨC TRUY NHẬP

0 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI ĐẦU FILE.

1 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI VỊ TRÍ HIỆN THỜI CỦA CON TRỎ.

2 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI CUỐI FILE.

BX = THẺ FILE.

CX : DX SỐ BYTES CẦN DỊCH CHUYỂN.

OUTPUT :

DX:AX : VỊ TRÍ MỚI CỦA CON TRỎ FILE TÍNH BẰNG BYTE TỪ ĐẦU FILE.

NẾU CF =1 MÃ LỖI TRONG AX (1, 6).

DỊCH CHUYỂN CON TRỎ FILE HÀM 42H INT 21H

CX : DX CHỨA SỐ BYTES ĐỂ DI CHUYỂN CON TRỎ. NẾU LÀ SỐ
DƯƠNG → CHUYỂN VỀ CUỐI FILE.

NẾU LÀ SỐ ÂM → CHUYỂN VỀ ĐẦU FILE.

DI CHUYỂN CON TRỎ FILE ĐẾN CUỐI FILE VÀ XÁC ĐỊNH KÍCH THƯỚC
FILE

MOV AH, 42H ; DI CHUYỂN CON TRỎ FILE

MOV BX, HANDLE ; LẤY THẺ FILE

XOR DX, DX

XOR CX, CX ; DỊCH CHUYỂN 0 BYTE

MOV AL, 2 ; TÍNH TỪ CUỐI FILE

INT 21H ; CHUYỂN CON TRỎ ĐẾN CUỐI FILE, DX:AX KÍCH THƯỚC FILE

JC MOVE_ERROR

THAY ĐỔI THUỘC TÍNH FILE HÀM 43H INT 21H

INPUT :

AH = 43H DS :DX = ĐỊA CHỈ CHUỖI ASCII STRING

AL = 0 ĐỂ LẤY THUỘC TÍNH FILE AL =1 ĐỂ THAY ĐỔI
THUỘC TÍNH FILE, CX = THUỘC TÍNH FILE MỚI (NẾU
AL =1)

OUTPUT :

NẾU THÀNH CÔNG, CX = THUỘC TÍNH HIỆN THỜI
NẾU CF ĐƯỢC LẬP \rightarrow CÓ LỖI, MÃ LỖI TRONG AX (2,3,5).

Ex : thay đổi thuộc tính File thành hidden file

```
MOV AH, 43H          ; Hàm lấy / đổi thuộc tính File  
MOV AL, 1            ; tùy chọn thay đổi thuộc tính  
LEA DX, FILENAME    ; lấy tên file kế cả đường dẫn.  
MOV CX, 1            ; I; thuộc tính Hideen  
INT 21H              ; đổi thuộc tính  
JC ATT_ERROR         ; thoát nếu có lỗi, mã lỗi trong AX
```

LẬP TRÌNH FILE

1. **Viết chương trình chép một file nguồn đến một file đích trong đó thay chữ thường bằng chữ hoa.**
2. **Viết chương trình đọc 2 file và hiển thị chúng bên cạnh nhau trên màn hình. Chú ý có chức năng dừng từng trang màn hình nếu file quá dài.**
3. **Viết chương trình ghép nội dung 1 file vào cuối 1 file khác đã có.**
4. **Viết chương trình tạo 1 thư mục, tên thư mục được gõ từ bàn phím (tên thư mục có thể bao gồm tên ổ đĩa, đường dẫn).**

Chương 13 :LẬP TRÌNH XỬ LÝ MẢNG & CHUỖI

- GIỚI THIỆU
- CỜ HƯỚNG DF
- CÁC LỆNH THIẾT LẬP VÀ XÓA CỜ HƯỚNG
- CÁC LỆNH THAO TÁC TRÊN CHUỖI
- MỘT SỐ THÍ DỤ MINH HỌA
- THƯ VIỆN LIÊN QUAN ĐẾN CHUỖI

GIỚI THIỆU CHUỖI

Trong ASM 8086 khái niệm chuỗi bộ nhớ hay chuỗi là 1 mảng các byte hay word.

→ Các lệnh thao tác với chuỗi cũng được thiết kế cho các thao tác với mảng.

Cờ hướng DF

Cờ định hướng (Direction Flag) : xác định hướng cho các thao tác chuỗi.

**DF=0 chuỗi được xử lý theo chiều tăng tức địa chỉ vùng nhớ chứa chuỗi tăng dần.
(chuỗi được xử lý từ trái qua phải).**

**DF=1 chuỗi được xử lý theo chiều tăng tức địa chỉ vùng nhớ chứa chuỗi giảm dần.
(chuỗi được xử lý từ phải qua trái).**

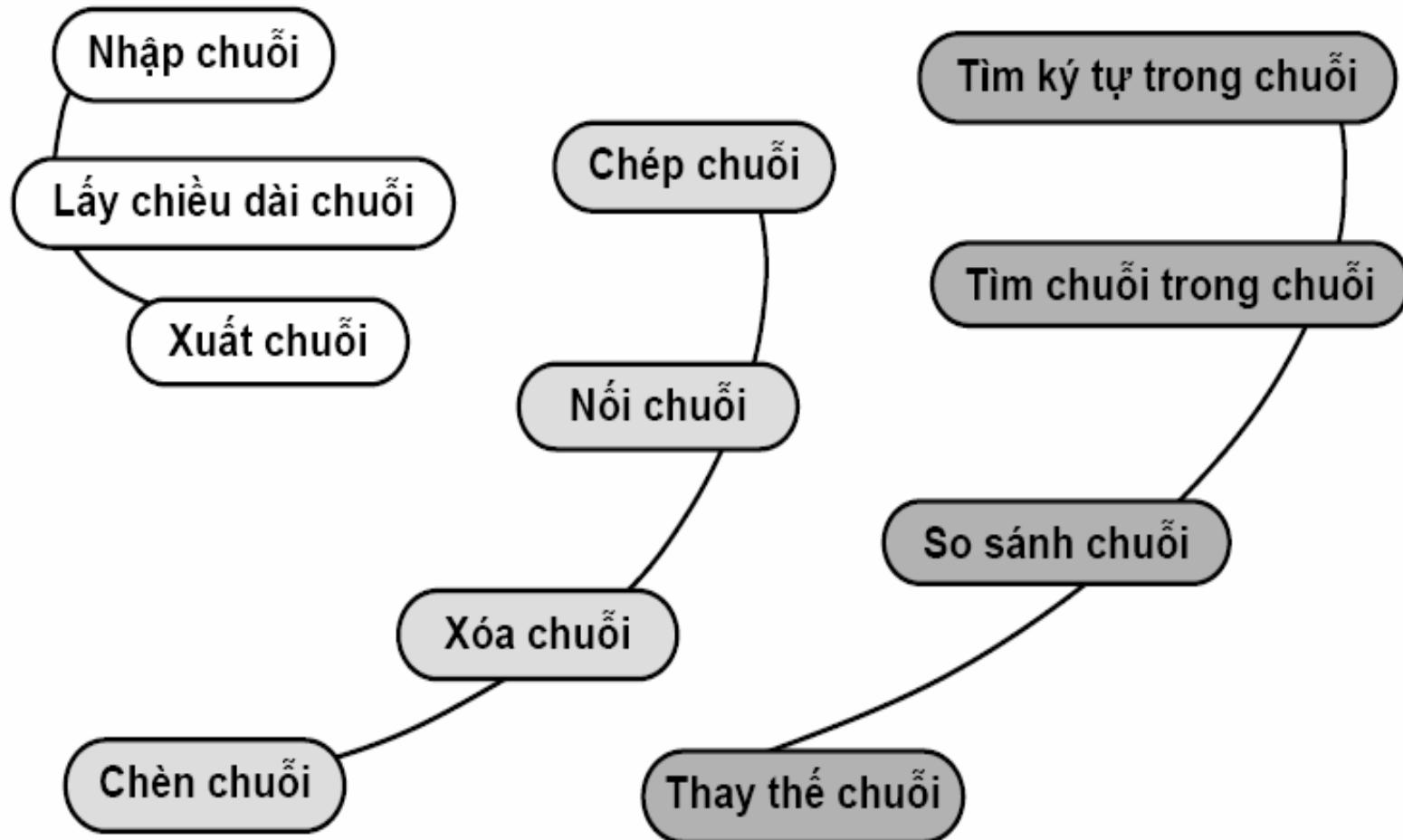
Trong DEBUG DF=0 ký hiệu là UP DF=1 ký hiệu là DN

LỆNH LIÊN QUAN ĐẾN CỜ HƯỚNG

CLD (CLEAR DIRECTION FLAG)
XÓA CỜ HƯỚNG DF =0

STD (SET DIRECTION FLAG)
THIẾT LẬP CỜ HƯỚNG DF=1

Các thao tác trên chuỗi

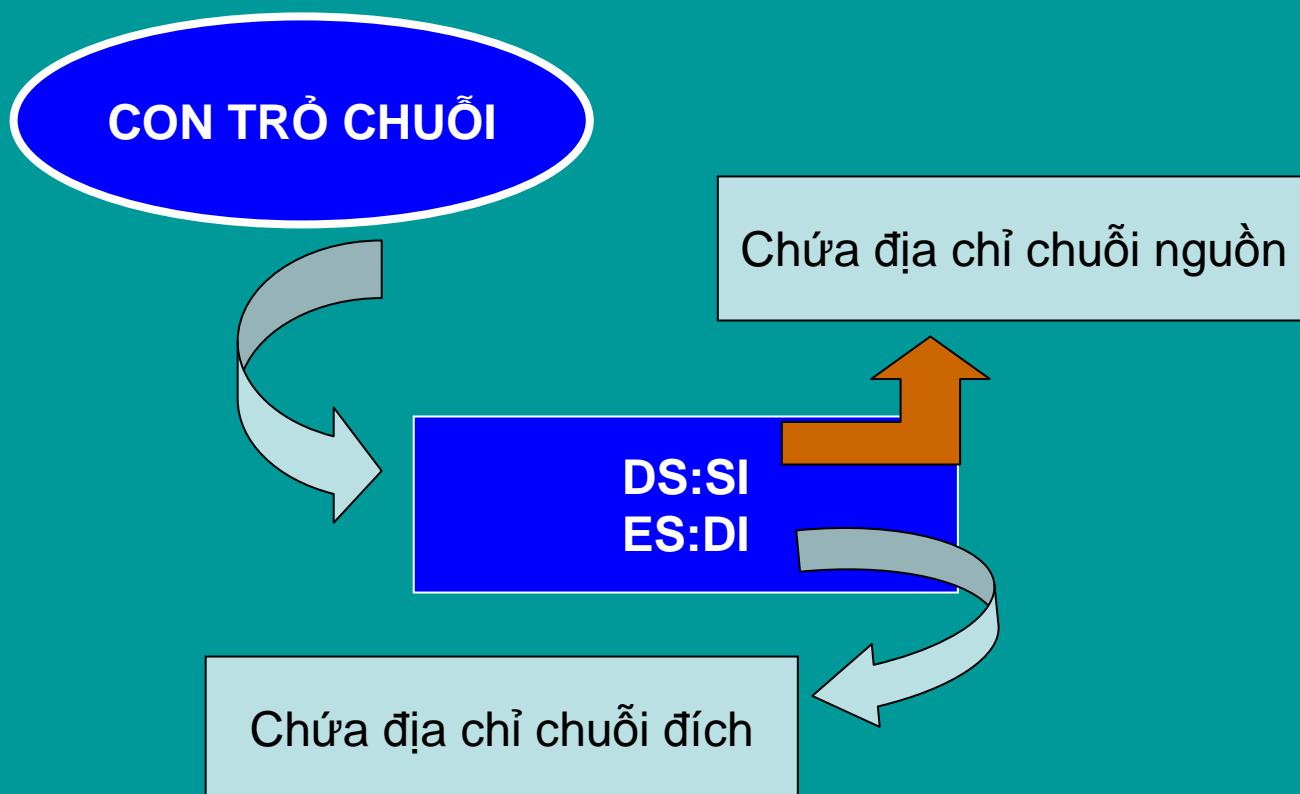


→ Trước khi sử dụng các lệnh xử lý chuỗi, ta phải xác định hướng xử lý chuỗi bằng cách set hay clear cờ hướng.

Lệnh đặt cờ hướng :

CLD : xóa cờ hướng, chuỗi được xử lý từ trái → phải

STD : đặt cờ hướng, chuỗi được xử lý từ phải → trái



CÁC THAO TÁC XỬ LÝ CHUỖI

NHẬP CHUỖI

Input : AH = 0AH, ngắt 21H

DS:DX = địa chỉ của buffer, trong đó buffer[0] là kích thước tối đa của chuỗi,

buffer[1] sẽ là kích thước dữ liệu nhập.

Output : Chuỗi buffer chứa nội dung nhập vào từ buffer[2] trở đi

Yêu cầu xem thêm các chức năng AH = 3FH và AH = 40H của ngắt 21H.

Tech Help! 4.0

F1 Help F10 Exit =

DOS Fn 3fH: Read from File via Handle

Expects	AH BX DS:DX CX	3fH file handle address of buffer to receive data number of bytes to read
Returns	AX AX	error code if CF is set to CY number of bytes actually read

Description: CX bytes of data are read from the file or device with handle number BX. The data is read from the current position of the file's read/write pointer and is placed into the caller's buffer pointed to by DS:DX.

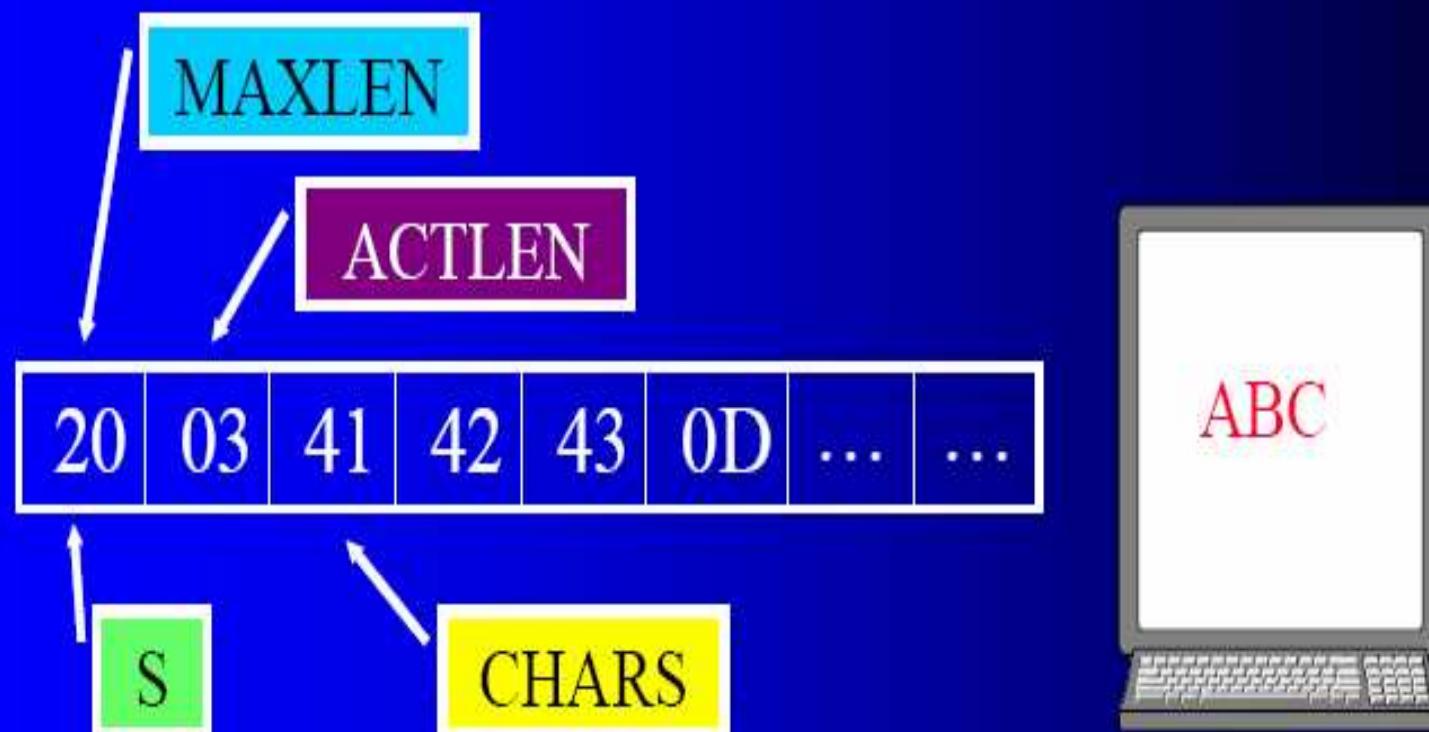
Use Fn 42H (Lseek) to position the file pointer before calling if necessary (OPEN sets the read/write pointer to 0).

This updates the file's read/write pointer to set up for a subsequent sequential-access read or write.

More ↓

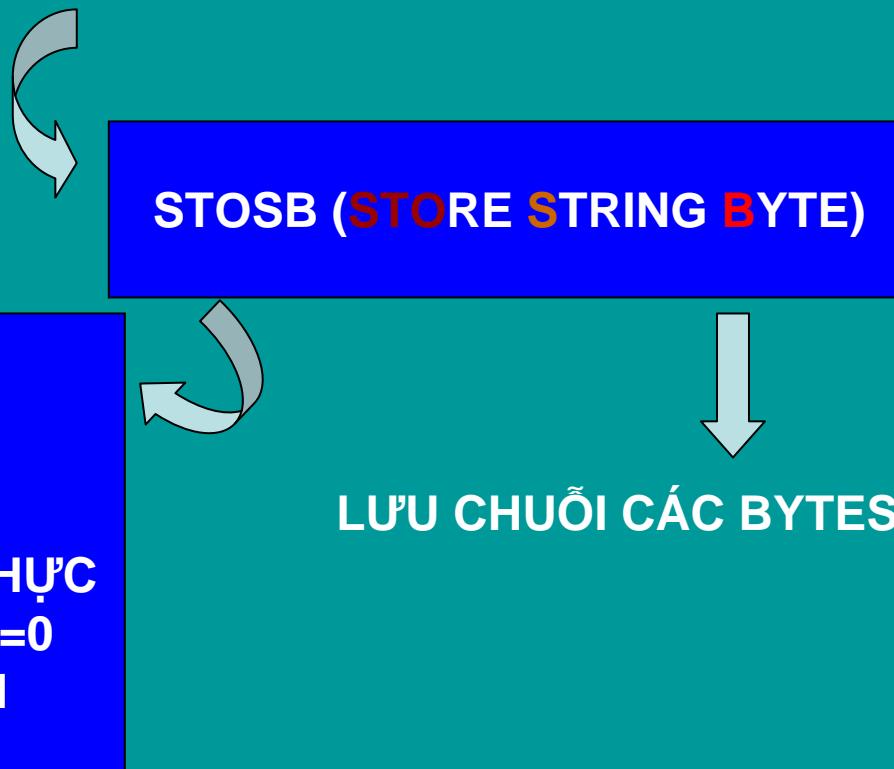
NHẬP CHUỖI

- Hàm 0Ah, INT 21h: Nhập chuỗi từ bàn phím, kết thúc Enter



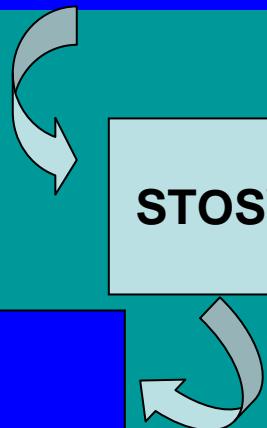
NHẬP CHUỖI

Ta cũng có thể dùng hàm 1 INT 21h đọc 1 ký tự từ bàn phím để nhập 1 chuỗi bằng cách dùng vòng lặp và lưu chuỗi bằng lệnh STOSB.



NHẬP CHUỖI

Ta cũng có thể dùng hàm 1 Int 21h đọc 1 ký tự từ bàn phím để nhập 1 chuỗi bằng cách dùng vòng lặp và lưu chuỗi bằng lệnh STOSW.



THÍ DỤ

```
.MODEL SMALL
.STACK 100H
.DATA
    STRING1 DB 'HELLO'
.CODE
MAIN PROC
    MOV AX,@DATA
    MOV ES,AX
    LEA DI, STRING1      ; khởi tạo ES
    CLD
    MOV AL,'A'            ; xử lý từ trái → phải
    STOSB                ; AL chứa ký tự cần lưu
    STOSB                ; lưu ký tự 'A'
    MOV AH,4CH             ; lưu ký tự thứ 2
    INT 21H
MAIN ENDP
END MAIN
```

THÍ DỤ

READSTR PROC

PUSH AX

PUSH DI

CLD

XOR BX,BX

MOV AH,1

INT 21H

LAP:

CMP AL,0DH

JE ENDLAP

CMP AL,8H

JNE ELSE1

DEC DI

DEC BX

JMP READ

ELSE1 :

STOSB

INC BX

READ :

INT 21H

JMP LAP

ENDLAP :

POP DI

POP AX

RET

READSTR ENDP

Giải thích :

DI chưa offset của chuỗi
BX chưa số ký tự nhập
8H mã ASCII của Backspace
không → lưu nó vào chuỗi
tăng số ký tự lên 1
Đúng → lùi con trỏ DI
giảm số ký tự nhập được

NHẬP XUẤT CHUỖI

HiỂN THỊ CHUỖI

AH = 09, ngắt 21H

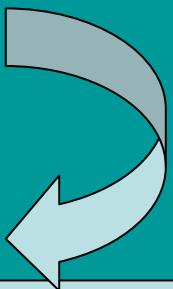
Vào : DX = địa chỉ offset của chuỗi.

Chuỗi phải kết thúc bằng kí tự '\$'.

Chú ý : thay vì dùng lệnh MOV
OFFSET ta có thể dùng lệnh LEA.

CÁC THAO TÁC XỬ LÝ CHUỖI

HiỂN THỊ CHUỖI



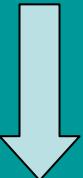
Nạp 1 chuỗi

For counter Do

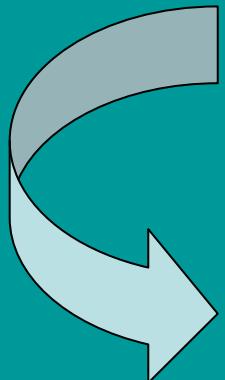
Nạp chuỗi cần hiển thị
vào AL
Chuyển vào DL
Hiển thị ký tự
EndFor



LODSB (LOAD STRING BYTE)



NẠP 1 CHUỖI CÁC BYTES



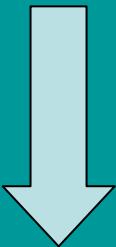
**CHUYỂN BYTE TẠI ĐỊA CHỈ DS:SI → AL
SI TĂNG 1 NẾU DF=0
SI GIẢM 1 NẾU DF =1**

THÍ DỤ

```
STRING1 DB 'ABC'  
MOV AX,@DATA  
MOV DS,AX  
LEA SI, STRING1  
CLD  
LODSB  
LODSB  
.....
```

NẠP BYTE THỨ 1 VÀ THỨ 2 → AL

LODSW (LOAD STRING WORD)



NẠP 1 CHUỖI CÁC WORD



**CHUYỂN WORD TẠI ĐỊA CHỈ DS:SI → AX
SI TĂNG HAY GIẢM TÙY TRẠNG THÁI DF**

THÍ DỤ

Hiển thị chuỗi nhập

```
DISPSTR PROC  
PUSH AX  
PUSH BX  
PUSH CX  
PUSH DX  
PUSH SI  
MOV CX, BX  
JCXZ EXIT  
CLD  
MOV AH,2  
LAP :  
LODSB  
MOV DL, AL  
INT 21H  
LOOP LAP
```

```
EXIT :  
POP SI  
POP DX  
POP CX  
POP BX  
POP AX  
RET  
DISPSTR ENDP
```

CHƯƠNG TRÌNH HÒAN CHỈNH

Viết chương trình nhập 1 chuỗi ký tự tối đa 80 ký tự, hiển thị 15 ký tự của chuỗi đã nhập ở dòng kế.

```
.MODEL SMALL
.STACK 100H
.DATA
STRING1 DB 80 DUP(0)
XDONG DB 0DH,0AH,'$'
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX

MOV ES,AX
LEA DI, STRING1
CALL READSTR
LEA DX,XDONG
MOV AH,9
INT 21H
```

```
LEA SI, STRING1
MOV BX, 15
CALL DISPSTR
MOV AX,4C00H
INT 21H
MAIN ENDP
; READSTR PROC
-----
; DISPSTR PROC
-----
END MAIN
```

CÁC THAO TÁC XỬ LÝ CHUỖI

Chuyển một BYTE : MOVS B

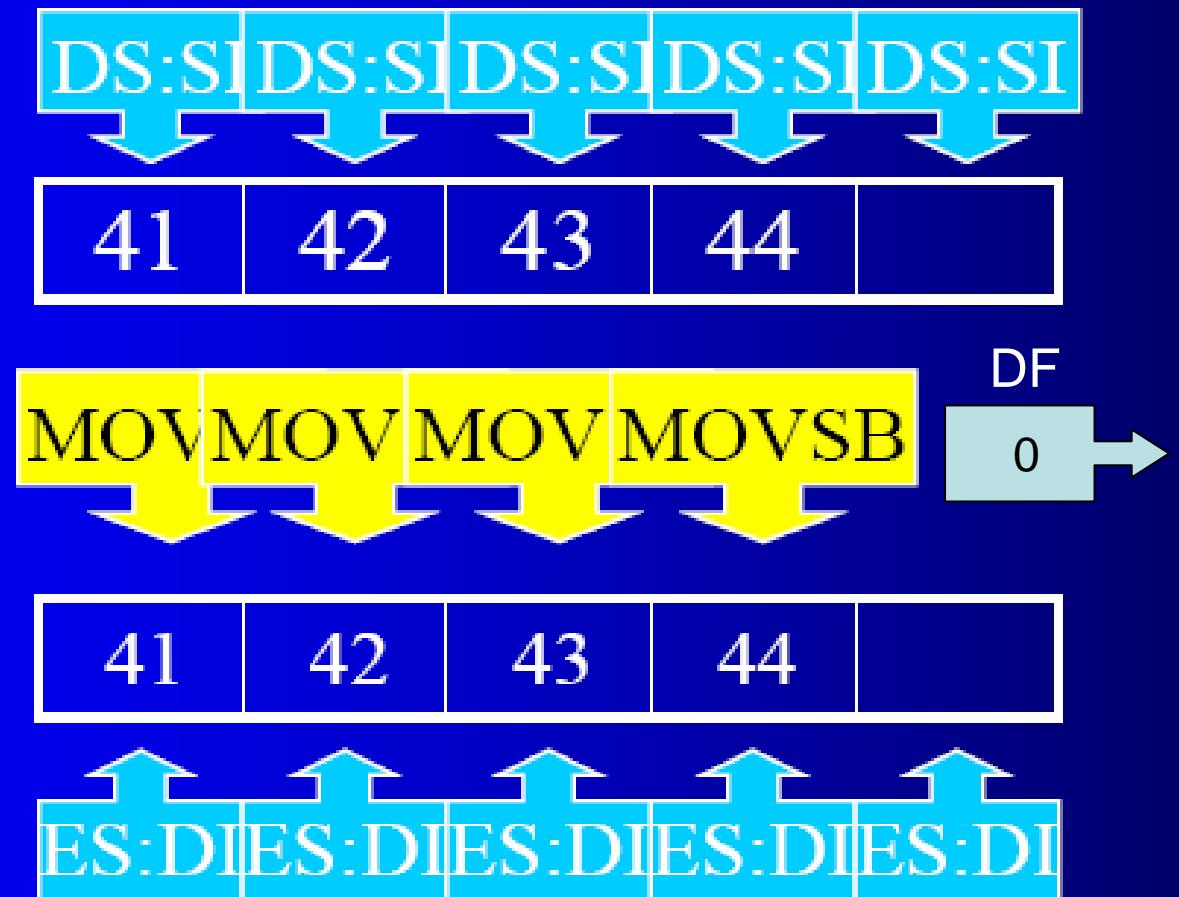
chuyển nội dung của byte được định bởi DS:SI đến byte
được chỉ bởi ES: DI.

Sau đó SI và DI tự động tăng lên 1 nếu cờ DF = 0
hay giảm 1 nếu DF = 1.

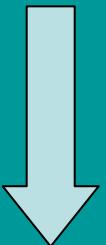


MOVSB chỉ chuyển 1 byte. Vậy cả chuỗi
ta làm thế nào ?

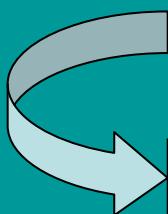
• MOVSB



MOVSW



Chuyển một chuỗi các word (2 bytes)

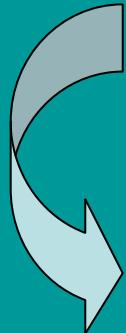


DS:SI trở đến chuỗi nguồn
ES:DI trở đến chuỗi đích

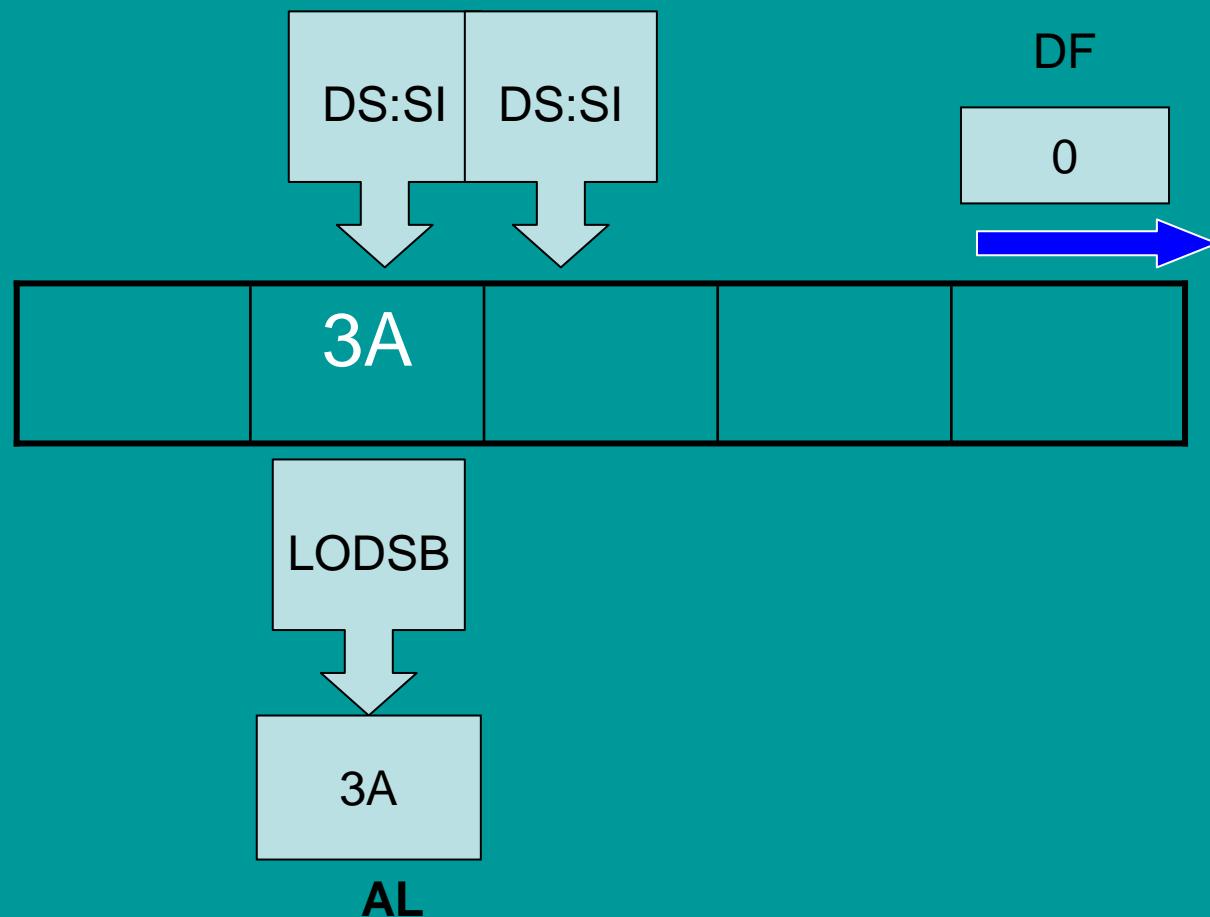
Sau khi đã chuyển 1 word của
chuỗi cả SI và DI cùng tăng
lên 2 nếu DF=0 hoặc cùng giảm
đi 2 nếu DF=1



LODSB (Load String Byte)



**Chuyển byte chỉ bởi DS:SI → AL
tăng SI lên 1 nếu DF=0
giảm SI xuống 1 nếu DF=1**

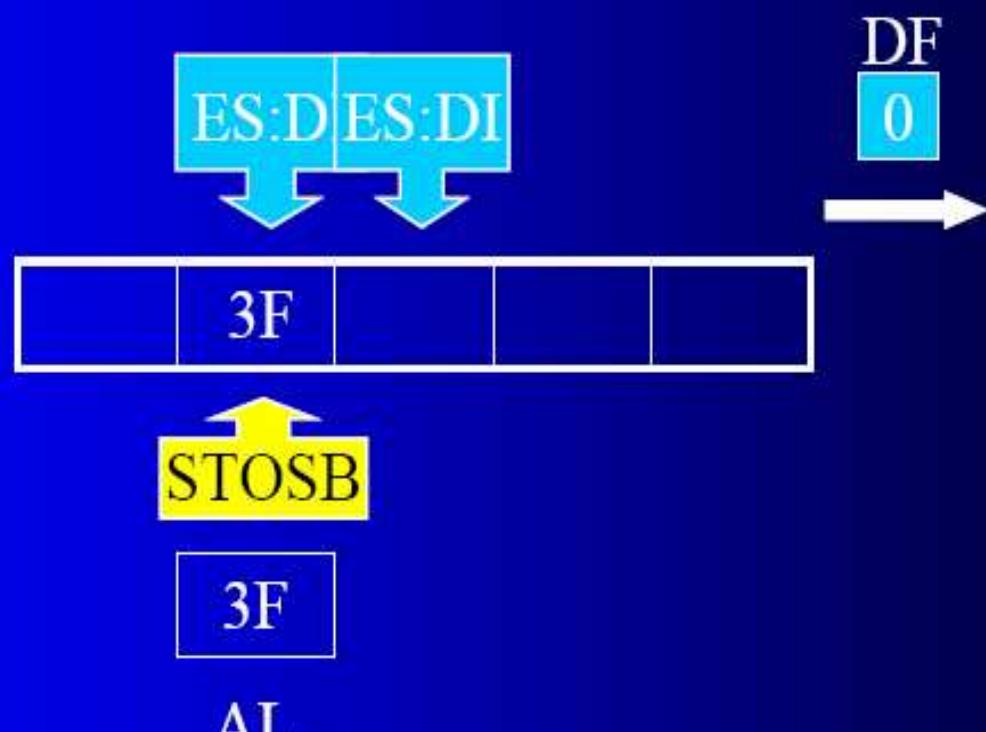


• MOVSW



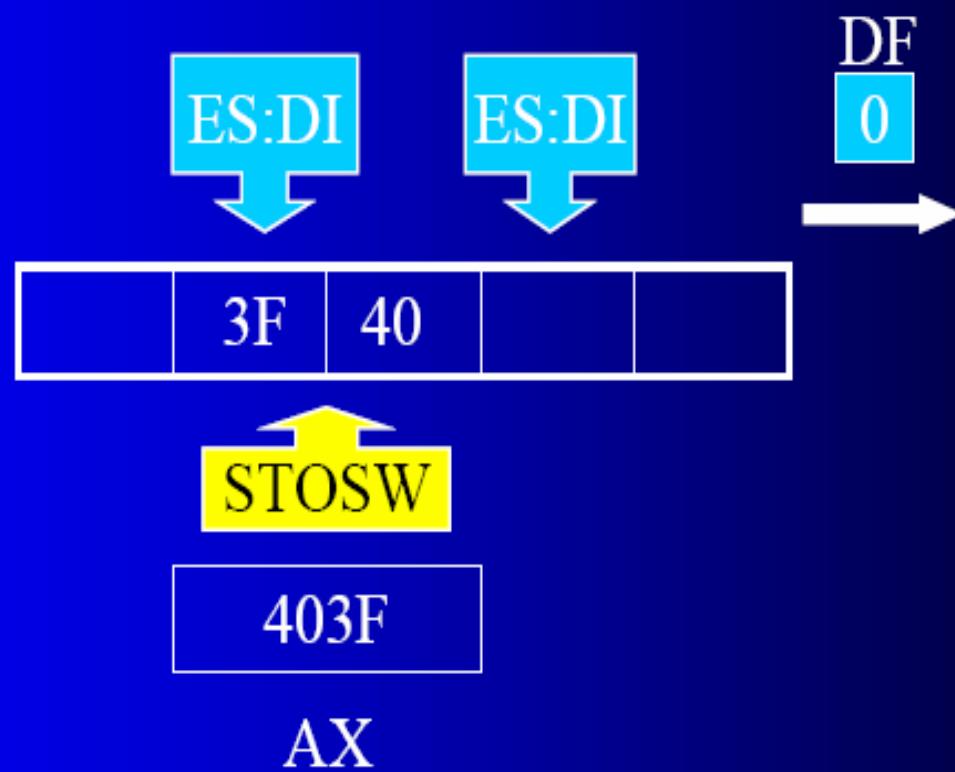
STOSB (LƯU CHUỖI BYTE)

- STOSB

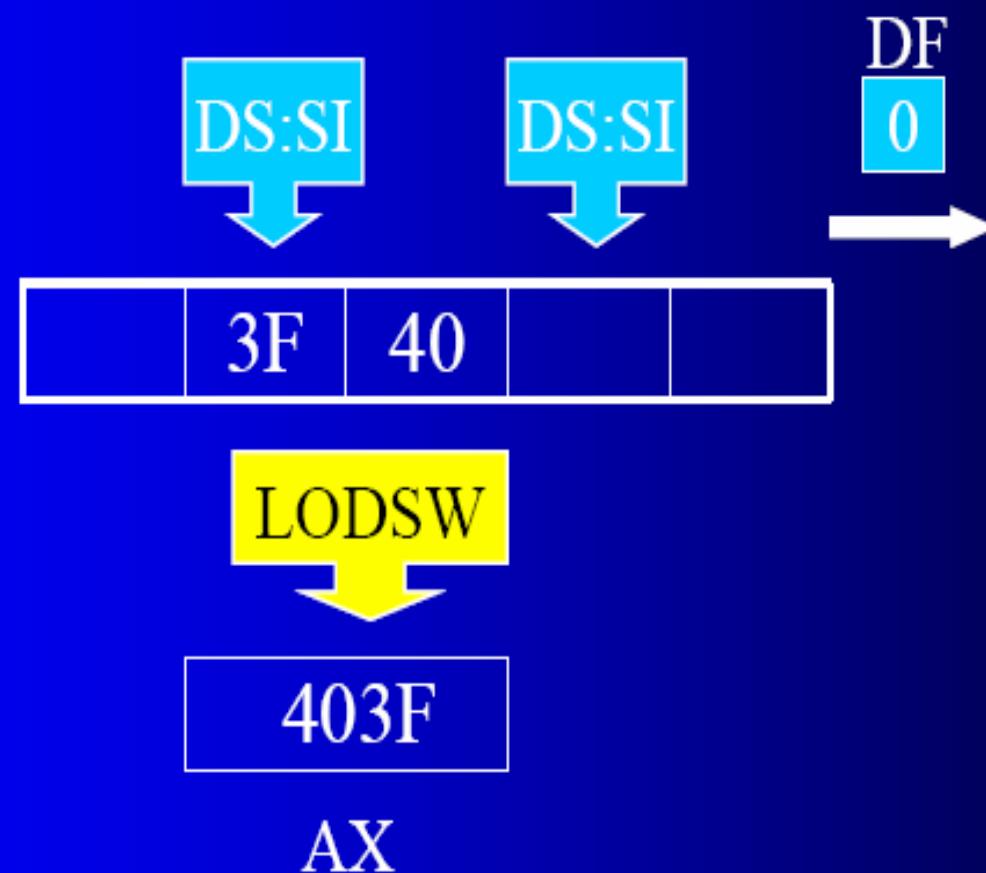


STOSW (LƯU CHUỖI WORD)

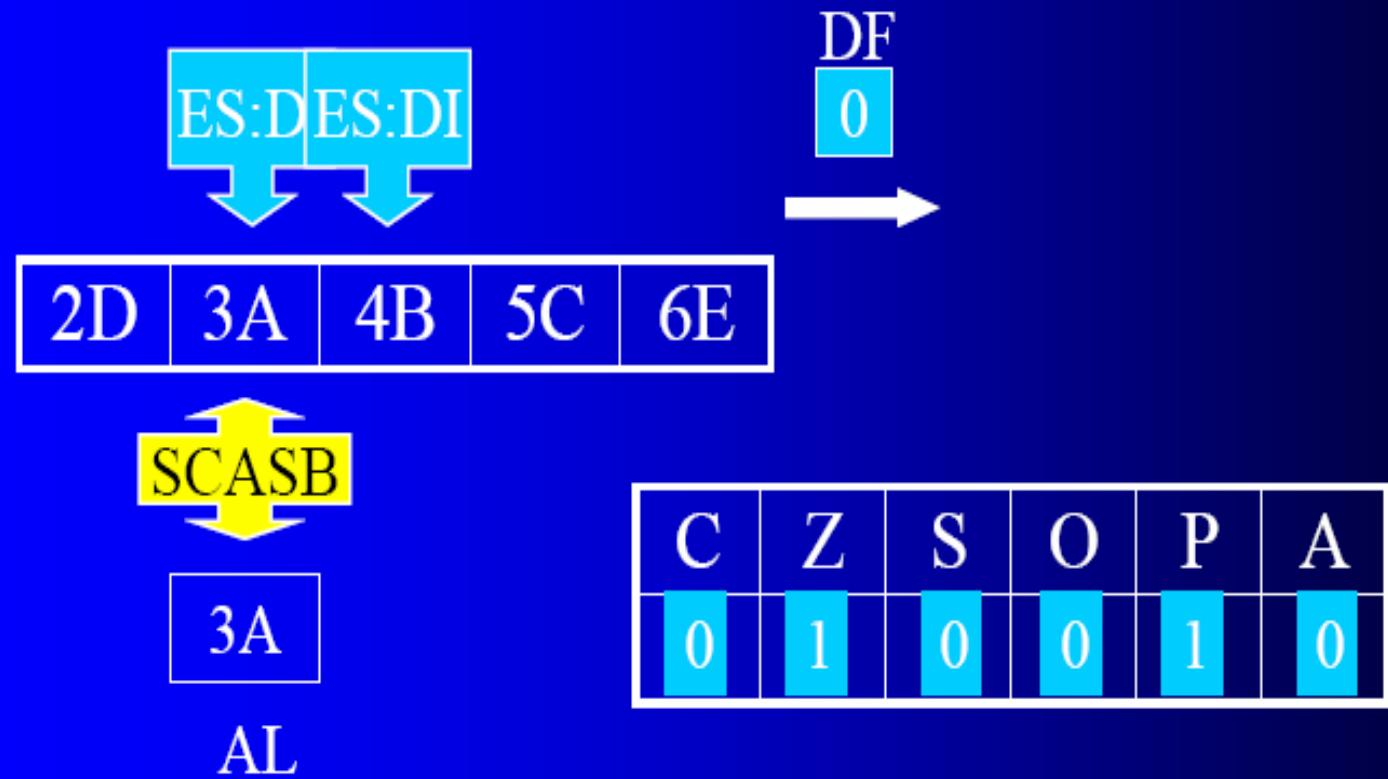
• STOSW



● LODSW



- SCASB



● CMPSB



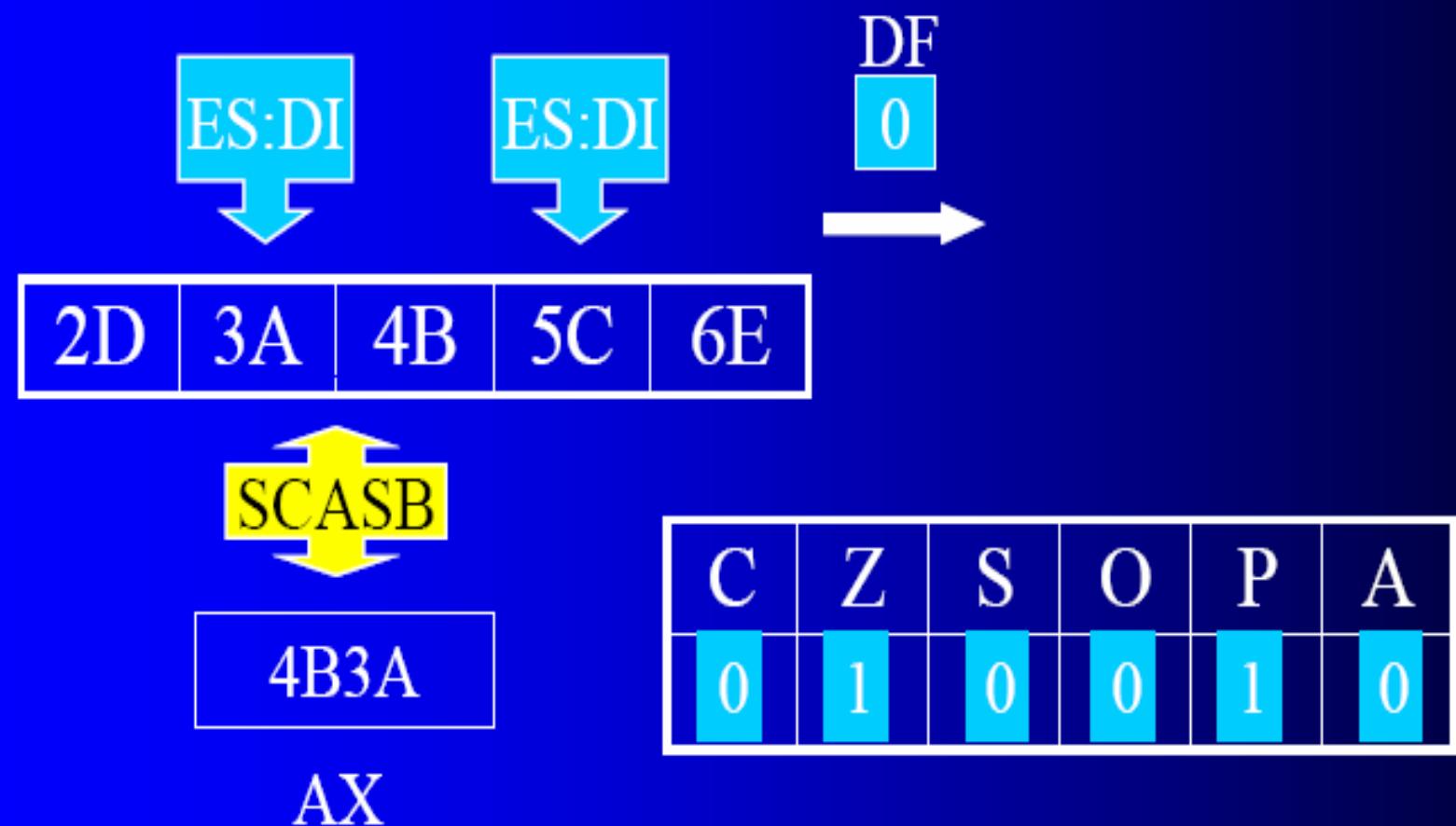
C	Z	S	O	P	A
0	1	0	0	1	0



DF
0



• SCASW



- CMPSW

ES:DI ES:DI

3A	30	4B	5C	6E
----	----	----	----	----

SCASW

3A	30	42	53	6E
----	----	----	----	----

DS:SI DS:SI

C	Z	S	O	P	A
0	1	0	0	1	0

DF
0



REP



Khởi tạo CX với số byte cần chuyển



Sau đó thực hiện lệnh
REP MOVSB



Sau mỗi lệnh **MOVSB**, CX giảm 1 cho đến
khi nó =0 → hết chuỗi.

THÍ DỤ MINH HỌA

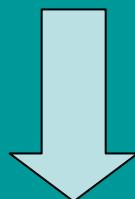
```
.DATA  
STRING1 DB 'HELLO'  
STRING2 DB 5 DUP(?)  
.....  
CLD  
LEA SI, STRING1  
LEA DI, STRING2  
MOV CX, 5  
REP MOVSB  
.....
```

Bài tập :
Viết đoạn chương trình chép chuỗi
STRING1 ở thí dụ trước vào
chuỗi STRING2 nhưng theo thứ
tự ngược lại.

THÍ DỤ MINH HỌA

Cho mảng sau
ARR DW 10,20,40,50,60,?

Viết các lệnh để chèn 30 vào giữa 20 và 40 (giả sử rằng DS và ES
đã chứa địa chỉ đoạn dữ liệu)



10,20, ,40,50,60

Dời 40,50,60 ra sau 1 vị trí

30

Sau đó chèn 30 vào

STD
LEA SI, ARR+8H
LEA DI, ARR+AH
MOV CX, 3
REP MOVSW
MOV WORD PTR[DI],30

MẢNG 1 CHIỀU

Một dãy các phần tử có cùng kiểu dữ liệu, có cùng 1 tên gọi.

Khai báo

MKT DB ‘abcdef’ ; mảng ký tự

MNB Dw 10h,20h,30h,40h,50h,60h ; mảng số

ArrA DB 100 DUP(0) ; khai báo mảng có 100 phần tử có giá trị khởi tạo bằng 0.

Các chương trình con

READSTRING - (Đọc tập tin bàn phím)

Vào : DS:DX = địa chỉ đệm nhận;

CX=số ký tự nhận tối đa.

Ra : DS:DX = địa chỉ chuỗi ASCIIIZ.

STRLEN - Lấy chiều dài chuỗi ASCIIIZ.

Vào : ES:DI = địa chỉ chuỗi ASCIIIZ.

Ra : AX = chiều dài chuỗi không kể số 0.

WRITESTRING - Xuất một chuỗi ra màn hình.

Vào : DS:DX= địa chỉ chuỗi ASCIIIZ.

Ra : Không.

STRCOPY - Chép chuỗi nguồn sang chuỗi đích.

Vào : DS:SI = địa chỉ chuỗi nguồn.

ES:DI = địa chỉ chuỗi đích.

Ra : Không.

Các chương trình con (tt)

STRCOMP - So sánh chuỗi và lập cờ.

Vào : DS:SI = địa chỉ chuỗi nguồn.

ES:DI = địa chỉ chuỗi đích.

Ra : thay đổi cờ.

CY = 1 : chuỗi nguồn < chuỗi đích.

ZF = 1 : chuỗi nguồn = chuỗi đích.

STRCHR - tìm ký tự trong chuỗi.

Vào : ES:DI = địa chỉ chuỗi.

AL = ký tự cần tìm.

Ra : CY = 0 : tìm thấy ký tự.

ES:DI = địa chỉ vị trí xuất hiện đầu tiên.

CY = 1 : không tìm thấy ký tự trong chuỗi.

STRSTR - tìm chuỗi trong chuỗi.

Vào : DS:SI = địa chỉ chuỗi nguồn.

DX = chiều dài chuỗi nguồn.

ES:DI = địa chỉ chuỗi đích.

BX = chiều dài chuỗi đích.

Ra : CY = 0 : tìm thấy

ES:DI = địa chỉ vị trí xuất hiện đầu tiên.

CY = 1 : không tìm thấy.

BÀI TẬP

Bài 1 : Viết chương trình nhập 1 số từ 1-12, in ra tên tháng tương ứng.

Bài 2 : Viết chương trình nhập 1 số từ 1-7, in ra tên thứ tương ứng.

MỘT SỐ BÀI TẬP MINH HỌA LẬP TRÌNH XỬ LÝ CHUỖI

Nhập 1 chuỗi dài tối đa 255 ký tự từ bàn phím. Cho phép dùng phím BackSpace để sửa khi nhập sai và kết thúc nhập khi gõ phím Enter.

Hướng dẫn :

Dùng hàm 0AH INT 21H để nhập chuỗi
DS:DX địa chỉ của buffer đệm lưu chuỗi.
Byte 0 : số byte tối đa có thể nhập.
Byte 1 : chứa giá trị 0
Byte 2 trở đi : để trống (lưu các ký tự sẽ nhập)

Để nhập 1 chuỗi ký tự vào Buffer
đêm ta khai báo như sau :
.DATA
BUFFERN DB 80,0,80 DUP(?)

B1. Viết chương trình nhập vào 1 từ, sau đó in từng ký tự trong từ theo chiều dọc.

Thí dụ Nhập CONG

Xuất : C

O

N

G

B2. Viết chương trình nhập vào 1 chuỗi, sau đó đổi tất cả chuỗi thành chữ hoa và in chuỗi ra màn hình ở dòng kế.

B3. Viết chương trình nhập hai chuỗi ký tự , kiểm tra xem chuỗi thứ hai có xuất hiện trong chuỗi thứ nhất hay không.

Ví dụ : Nhập chuỗi thứ nhất : computer information

Nhập chuỗi thứ hai : compute

Xuất: Chuỗi thứ hai có xuất hiện trong chuỗi thứ nhất.

B4. Viết chương trình nhập 1 chuỗi ký tự viết hoa các ký tự nguyên âm, viết thường các ký tự phụ âm.

Ví dụ : Nhập chuỗi : “aBcdE”

Xuất chuỗi: “AbCdE”

B5. Viết chương trình nhập vào 2 chuỗi ký tự s1, s2 và 1 số nguyên dương n. Chèn chuỗi s2 vào chuỗi s1 ở vị trí ký tự thứ n trong chuỗi s1 .

Ví dụ : Nhập chuỗi s1 : “abcde”

Nhập chuỗi s2 : “fgh”

Nhập n = 3

Xuất kết quả : “abcfghde”

B6. Viết chương trình nhập vào từ bàn phím 1 chuỗi và tính số lần xuất hiện của các nguyên âm (a,e,i,o,u, y), các phu am, các khoang trang, trong chuỗi tương ứng.

Ví dụ : Nhập chuỗi : “dai hoc khoa hoc tu nhien thanh pho ho chi minh”

Xuất : Số lần xuất hiện của các nguyên âm là : 14 , phu am la: 24, khoang trang la: 9

B7. Viết chương trình nhập vào từ bàn phím 1 chuỗi gồm các ký tự trong bảng chữ cái. Đếm xem trong chuỗi có bao nhiêu từ.

Ví dụ : Nhập chuỗi : “ hO Chi mINh ”

Xuất : chuỗi gồm có 3 từ

B8. Viết chương trình nhập vào từ bàn phím 4 số . Xuất ra màn hình 4 số đó theo thứ tự tăng dần .

Ví dụ : Nhập : 14 7 26 11

Xuất : 7 11 14 26

B9. Viết chương trình nhập vào từ bàn phím 4 số và sau đó xuất số lớn nhất và nhỏ nhất ra màn hình.

Ví dụ : Nhập : 13 21 1 49

Xuất : Số lớn nhất : 49

Số nhỏ nhất : 1

Viết chương trình nhập vào từ bàn phím chuỗi 1 (chuỗi dài), chuỗi 2 (chuỗi ngắn) và một ký tự. Sau đó, làm các công việc sau :

- Tìm chuỗi 2 trong chuỗi 1 và in ra vị trí xuất hiện đầu tiên của chuỗi 2 trong chuỗi 1 nếu tìm thấy. Ngược lại in ra không tìm thấy.
- Tìm ký tự đã nhập trong chuỗi 1 và in ra vị trí xuất hiện đầu tiên của ký tự nếu tìm thấy.Ngược lại in ra không tìm thấy.
- Thay chuỗi 2 trong chuỗi 1 bằng ký tự (nếu được).