CẤU TRÚC MÁY TÍNH

LẬP TRÌNH HỢP NGỮ

Cấu trúc Máy tính & Lập trình Assembly

MỤC TIÊU :

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 tính.
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 - Tống Văn On
- Lập trình Hợp ngữ - Nguyễn Ngọc Tấn -Vũ Thanh Hiền
- 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

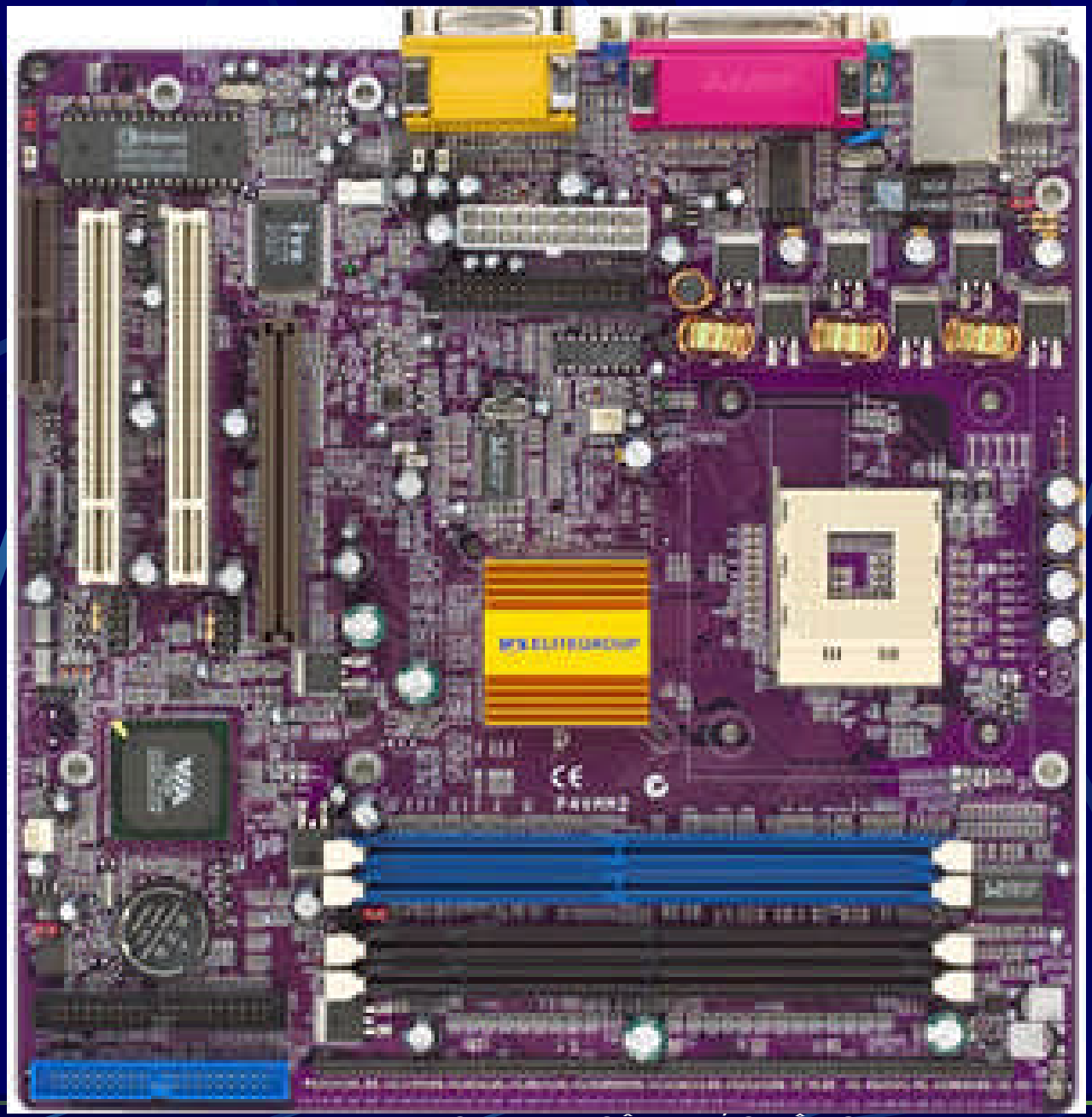
Mục tiêu :

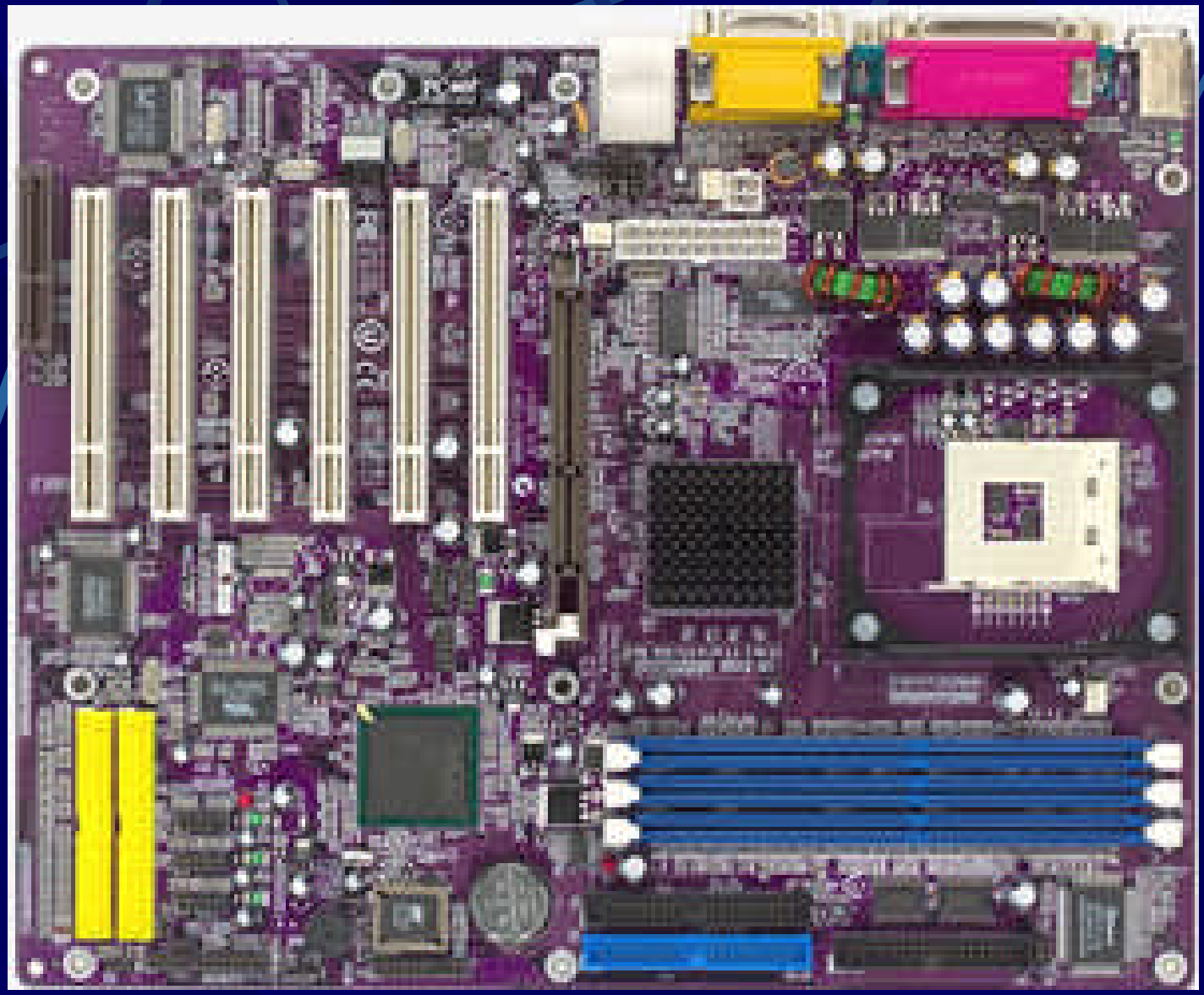
- Nắm được tổng quan về cấu trúc máy tính.
- Hiểu về Máy Turing & Nguyên lý Von Neumann
- Biết sơ đồ khối chi tiết của máy tính
- Nắm nguyên lý hoạt động máy tính
- Biết các component của máy tính : Processors,Memory,Input/Output devices,Bus

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





Máy tính & Sự tính toán

Bộ xử lý

Memory : chứa các chỉ thị & dữ liệu

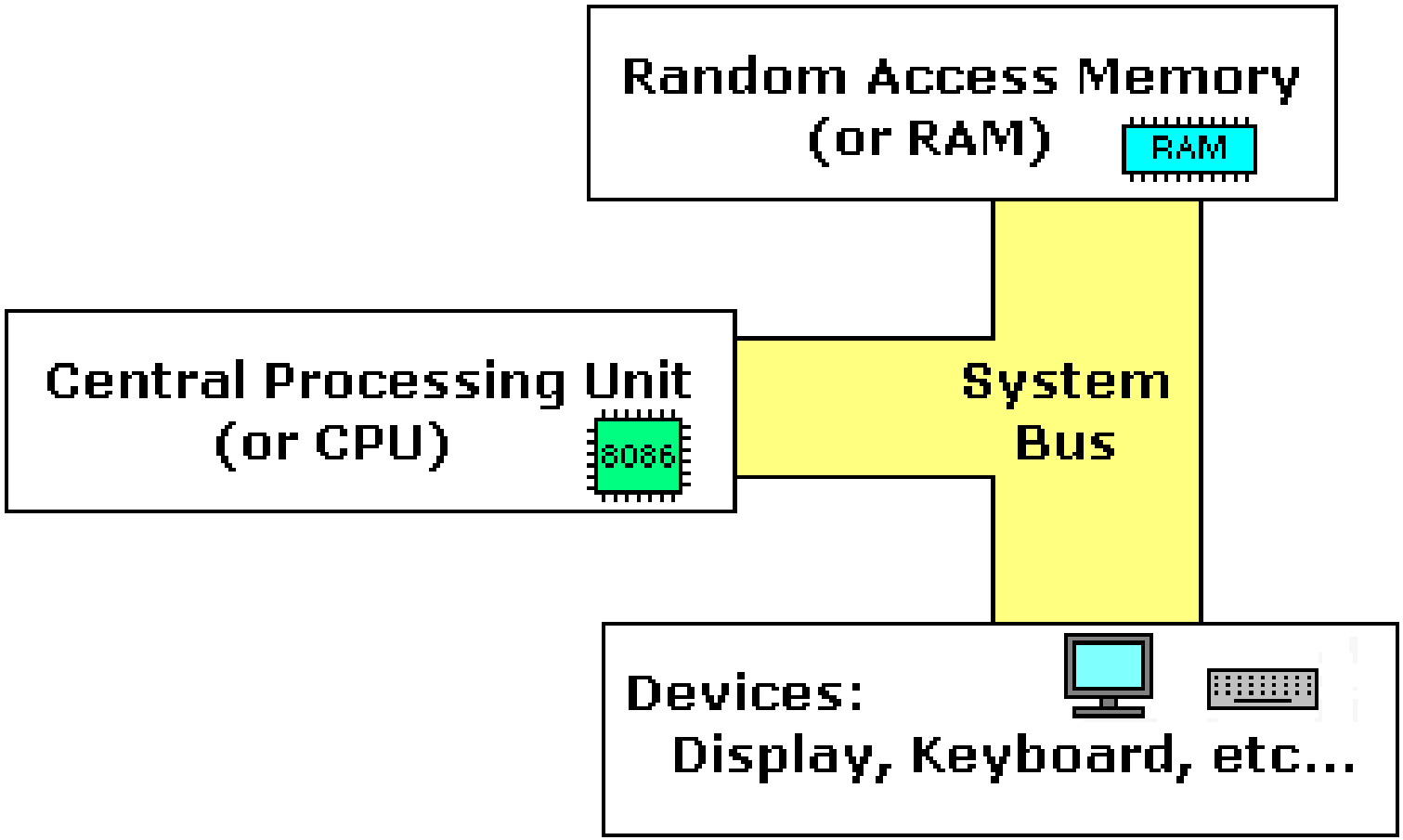
2+3/4*3-5=?

……

……..

……

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.

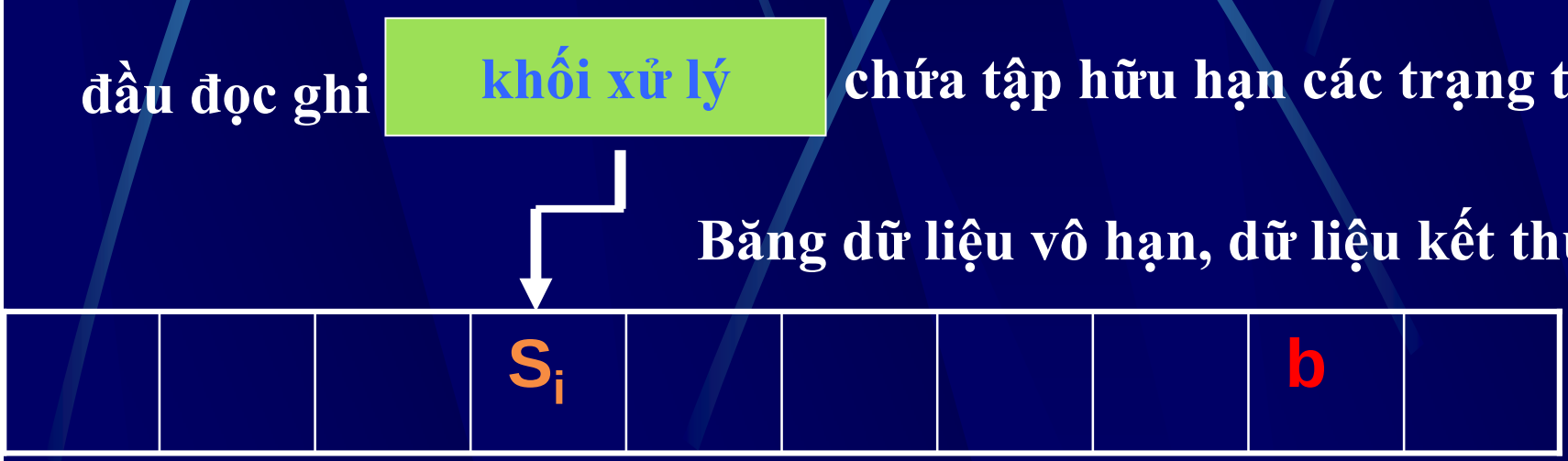
Tổng quan về cấu trúc máy tính

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



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 ?

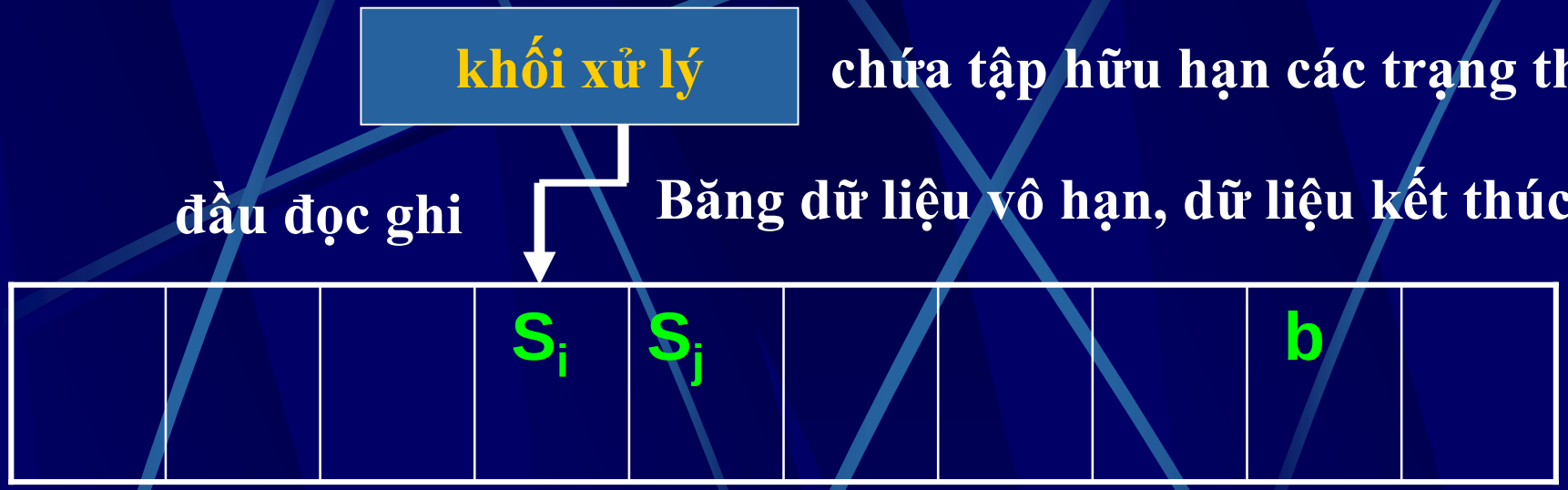
Mạch tổ hợp : là mạch điện có trạng thái ngõ ra phụ thuộc tức thời vào tổ hợp của trạng thái ngõ vào.

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



Máy làm việc theo từng bước rời rạc. Một lệnh của máy như sau : $q_iS_iS_jXq_j$.

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 hoat 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 cất vô băng.

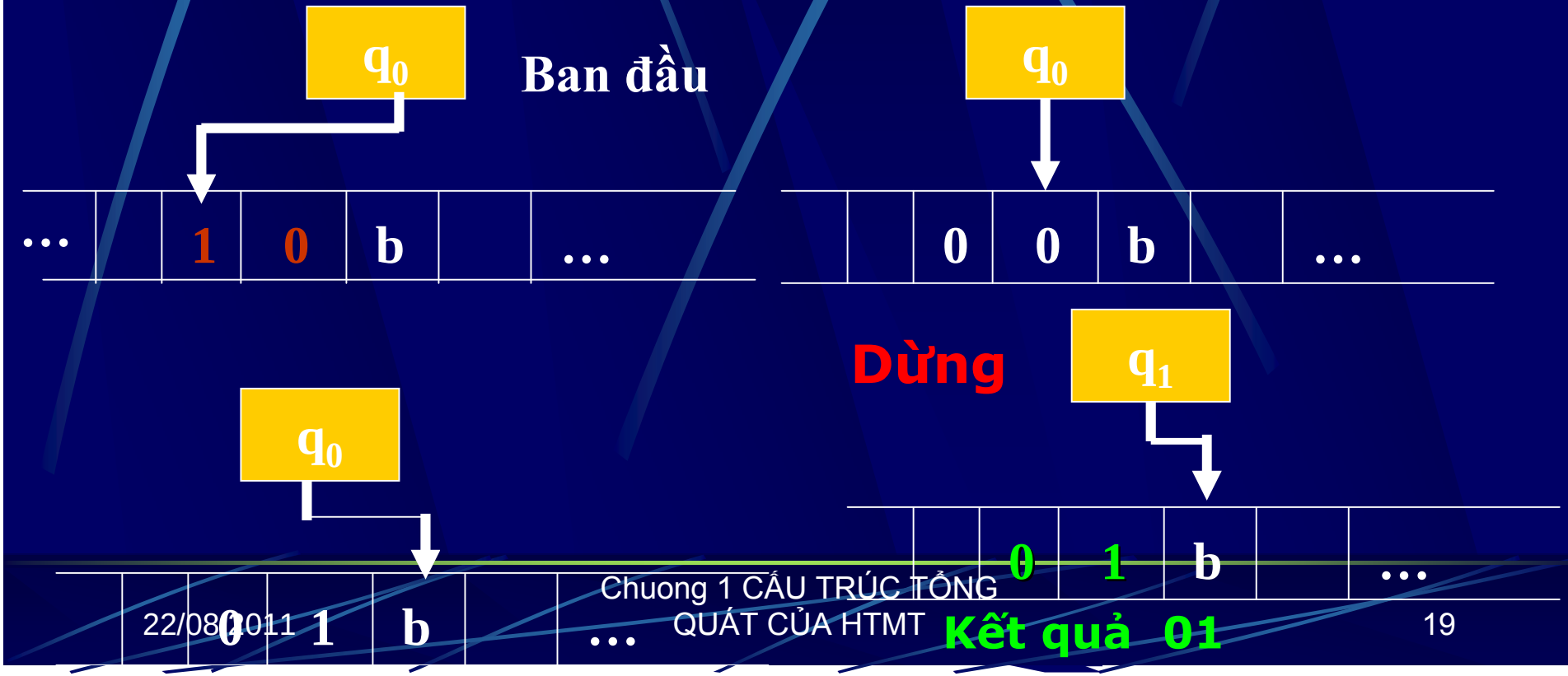
- Trạng thái trong ban đầu của máy là q_0 .
  - Đầu đọc/ghi ở ô chứa ký hiệu đầu tiên của chuỗi ký hiệu nhập. Trong quá trình hoạt động, sự thay đổi dữ liệu trên băng, sự dịch chuyển đầu đọc ghi và sự biến đổi trạng thái trong của máy sẽ diễn ra tuân theo các lệnh thuộc tập lệnh của máy tùy theo trạng 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 : $q_0 01 R q_0$, $q_0 10 R q_0$, $q_0 bb N q_1$**

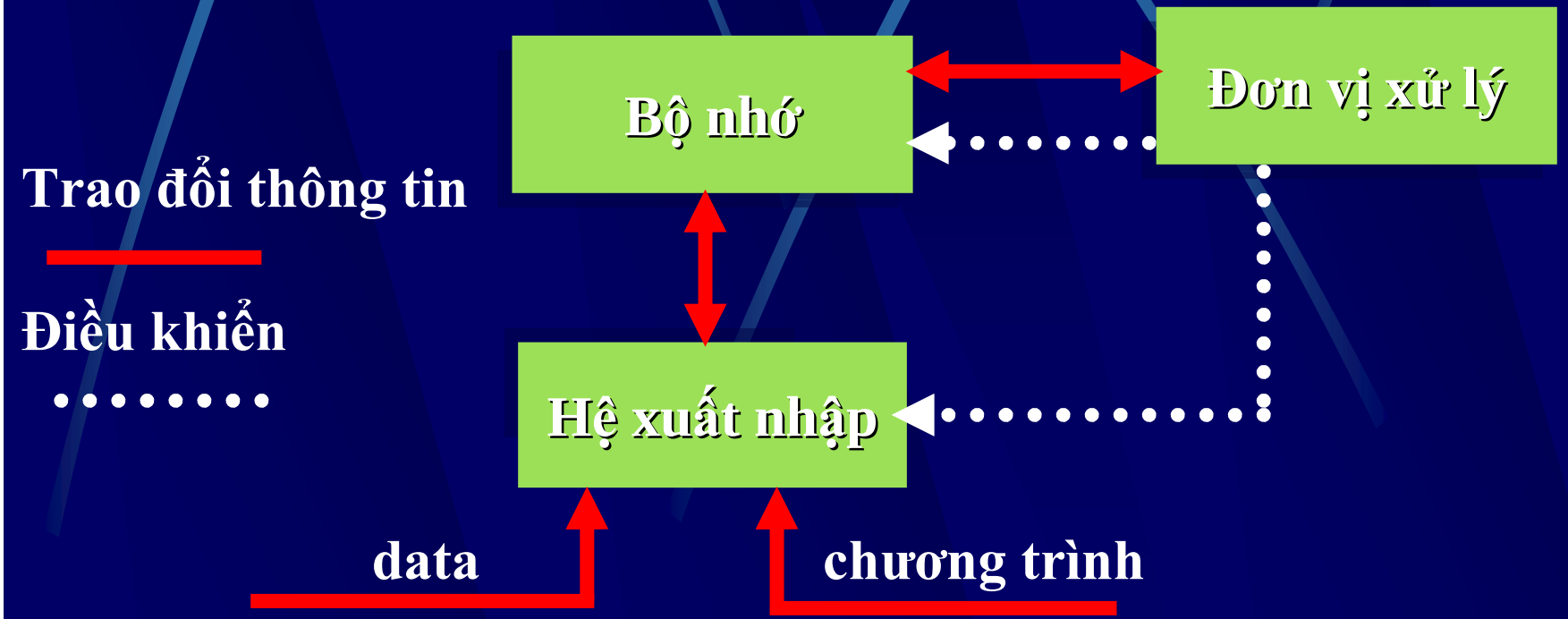


Nguyên lý VonNeumann

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.

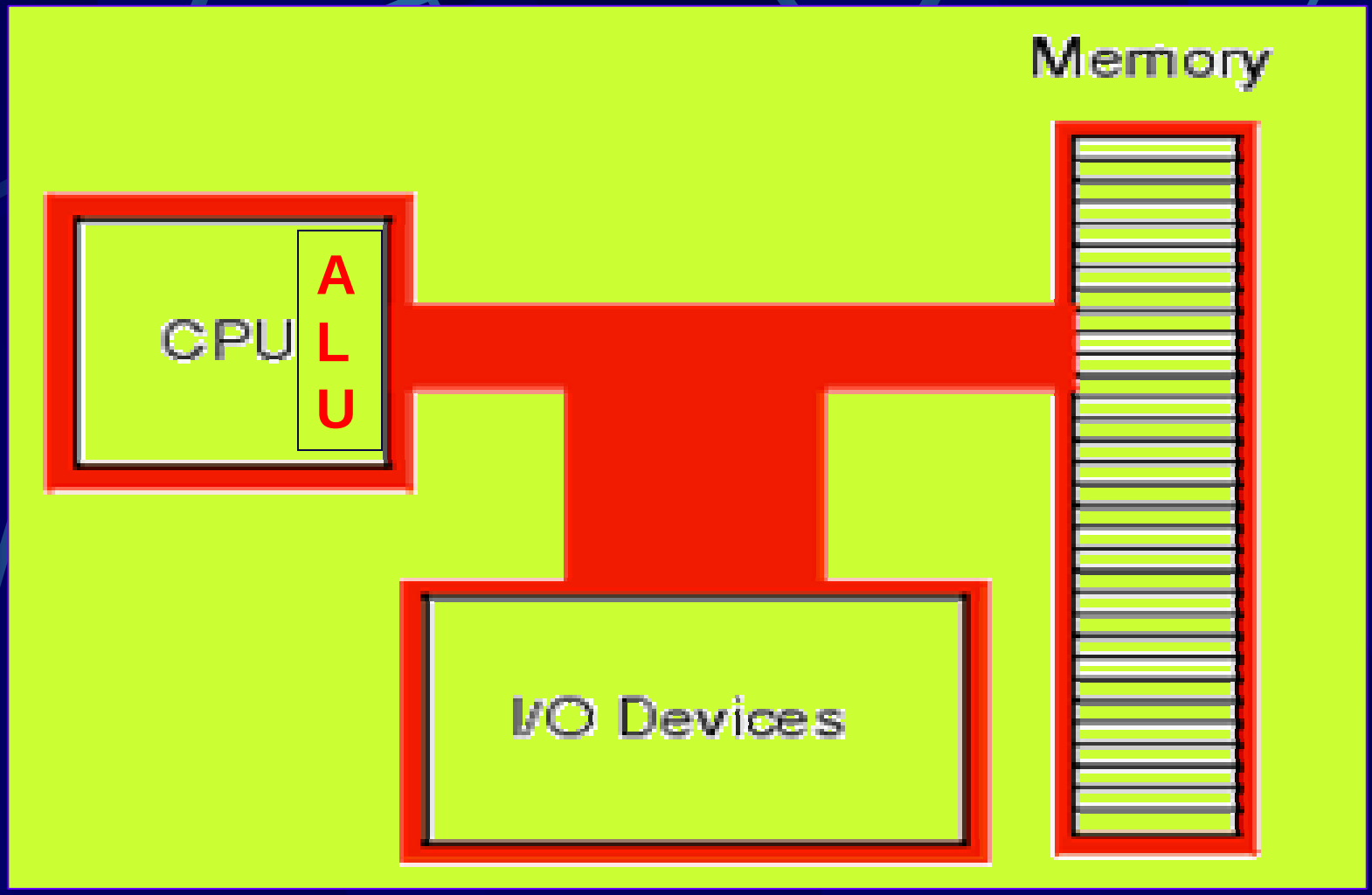
- Bộ nhớ chia làm nhiều ô, mỗi ô có 1 địa chỉ (đánh số thứ tự) để có thể chọn lựa ô nhớ trong quá trình đọc ghi dữ liệu. (nguyên lý định địa chỉ)

Nguyên lý Von Neumann (cont)

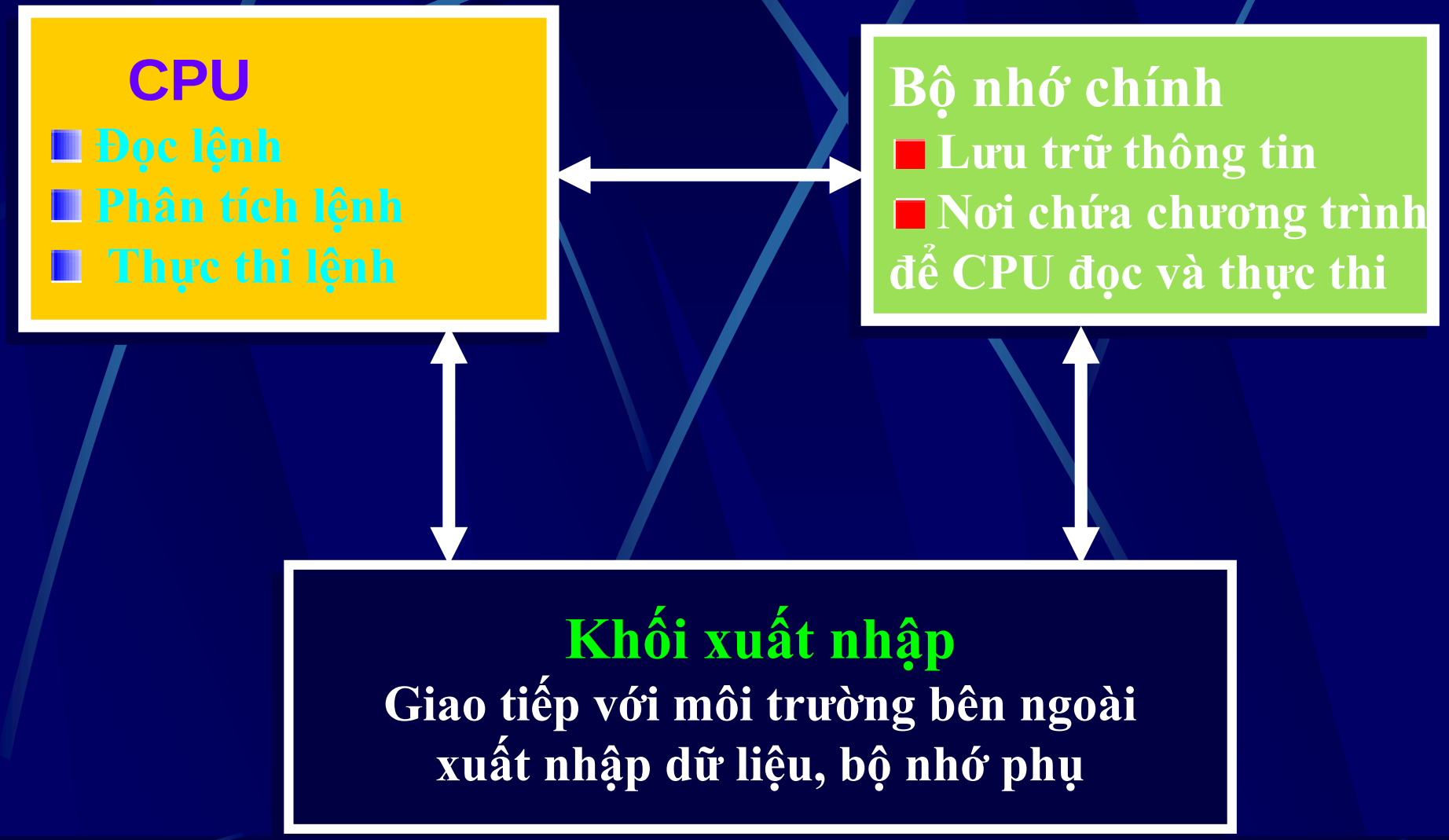
- Các lệnh được thực hiện tuần tự nhờ 1 bộ đếm chương trình (thanh ghi lệnh) nằm bên trong đơn vị xử lý.

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

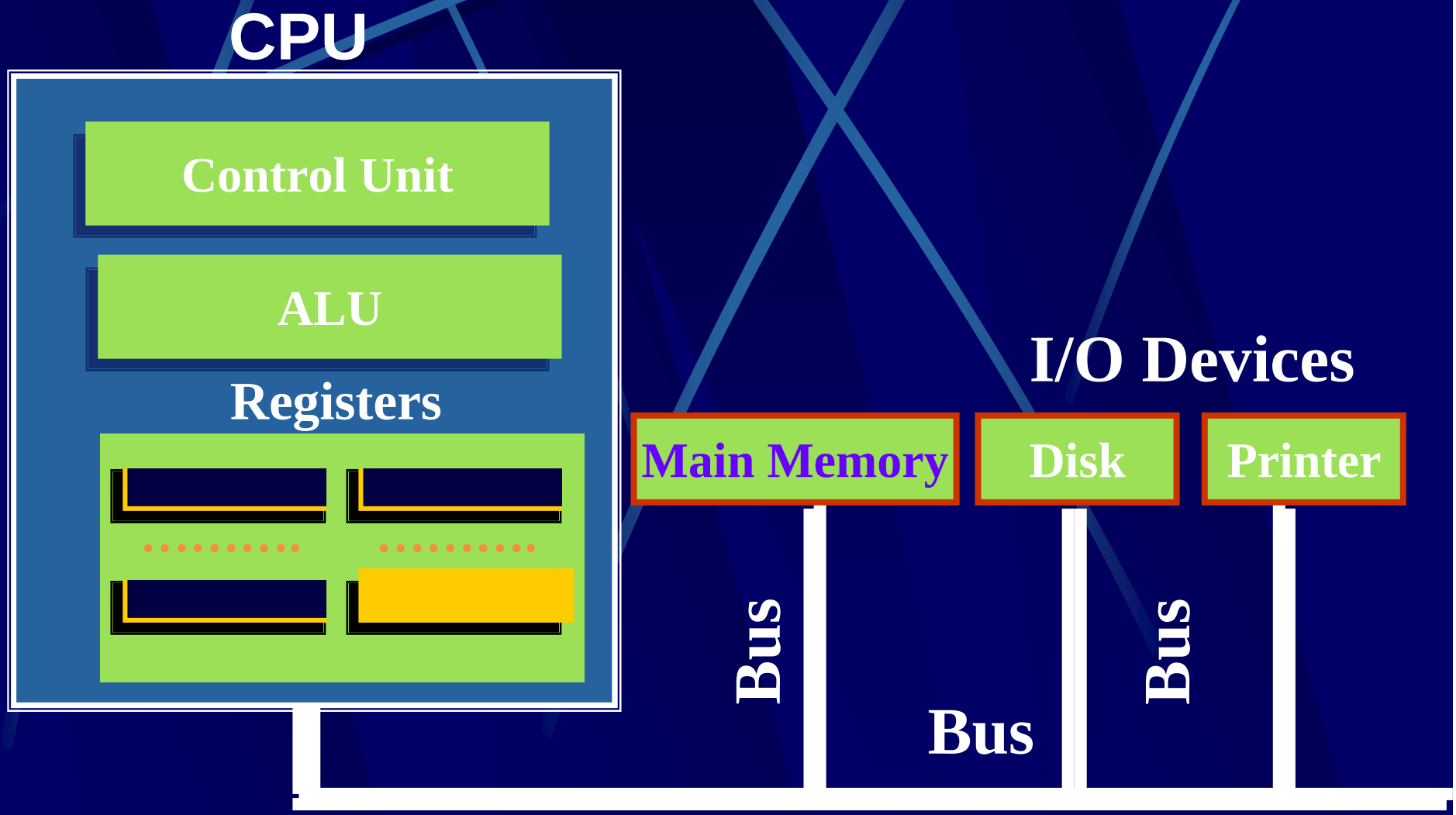
Typical Von Neumann Machine



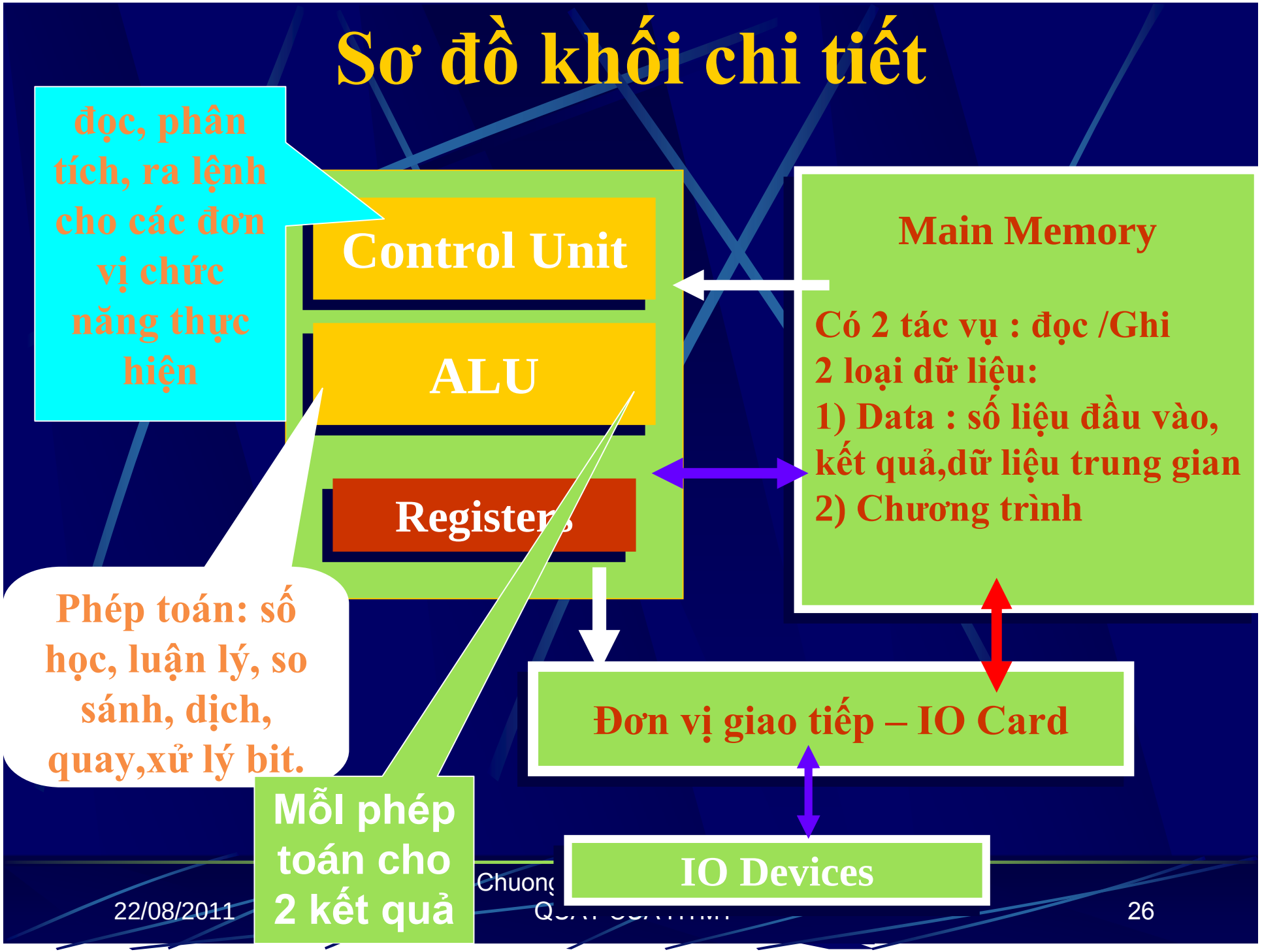
Nguyên lý hoạt động MT



Tổ chức Máy tính 1 CPU & 2 I/O device



Sơ đồ khối chi tiết

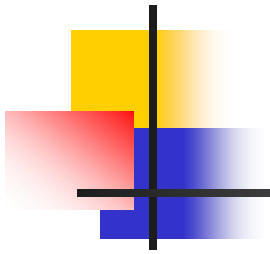


Tổng kết chương

- 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

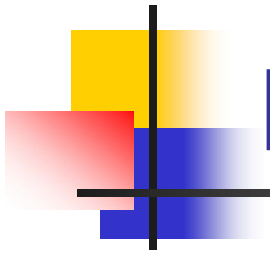
Hệ đếm	Cơ số	số ký số	dạng ký số và ký tự biểu diễn số
nhị phân	2	2	**0 1** **Ex : 1010_b**
bát phân	8	8	0 1 2 3 4 5 6 7 Ex : 24_o
thập phân	10	10	0 1 2 3 4 5 6 7 8 9 Ex : 12_d
thập lục phân	16	16	**0 1 2 3 4 5 6 7 8 9 A B C D E F** **Ex : $3F8_h$**

Hệ thống số

Hệ thống số là gì ?

Vào thờI điểm đó, việc dùng các que để đểm 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 ngườidùng các que để
đếm sau đó đã học vẽ các hình trên mặtđất và trên giấy.
thí dụ số 5 lần đầu 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.



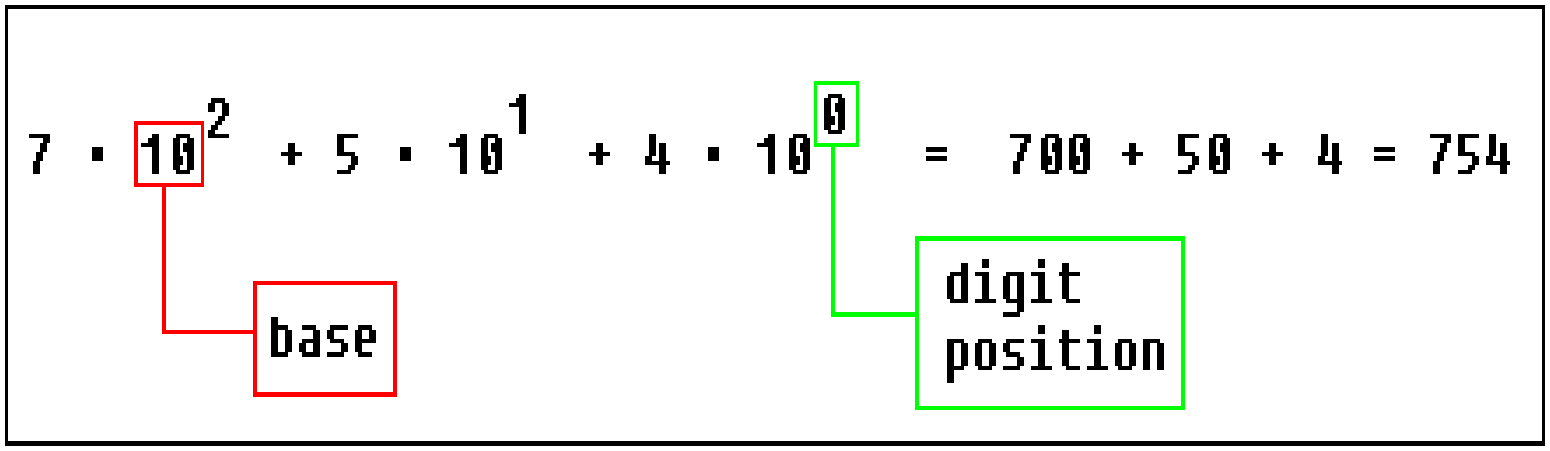
Hệ thống số

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!!!.

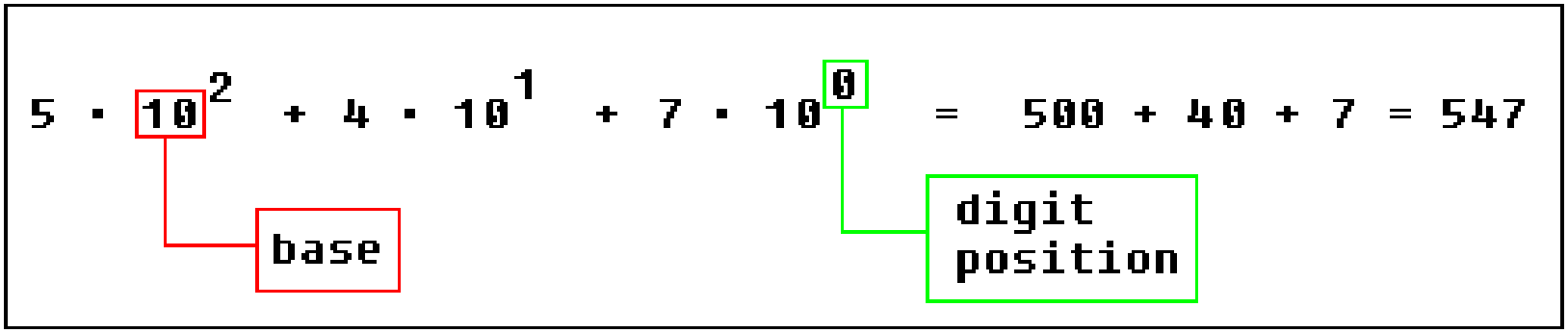
Một trong những cách tốt nhất hiện nay là dùng hệ thống số thập phân (decimal system).

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



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 :

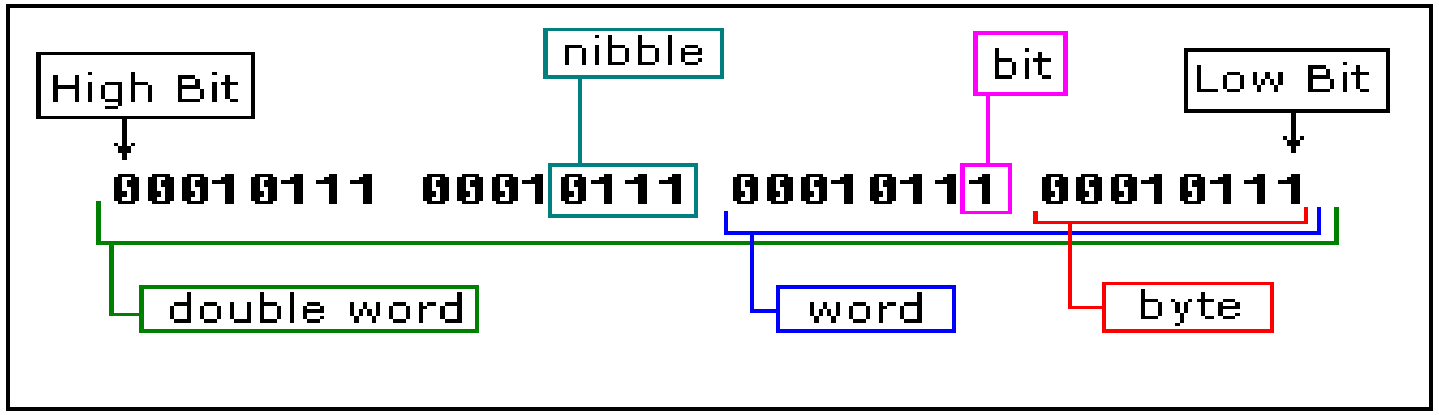


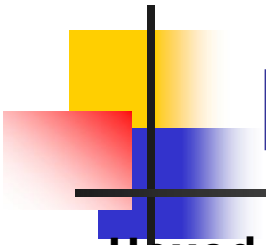
Binary System

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 : $\mathbf{1234_h} = 4660_d$

Các phép toán trong hệ nhị phân

cộng :

0 + 0 = 0 0 + 1 = 1 1+ 0 = 1 1 + 1 = 0 nhớ 1

trừ : 0 - 0 = 0 0 - 1 = 1 mượn 1 1 – 0 = 1 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 töø 10 ➔ heä 2

Ñoåi töø heä 10 ➔ heä 2 :

Ex : 12d = 1100b

Caùch ñoåi : laáy soá caàn ñoåi chia lieân tieáp cho 2, döøng khi soá bò chia baèng 0. Keát quaû laø caùc soá dö laáy theo chieàu ngöôïc laïi.

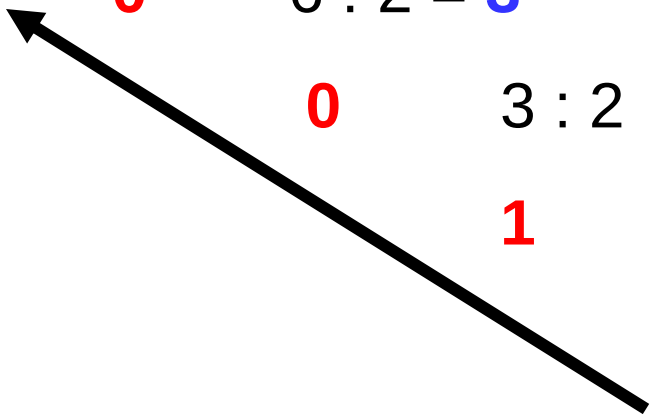
12 : 2 = **6**

0 6 : 2 = **3**

0 3 : 2 = **1**

1 1 : 2 = **0** **döøng**

1



Chuyển hệ tö hei 2 ➔ hei 10

Ñoi tö hei 2 ➔ hei 10 :

Ex : $1100_b = ?_d$

Cach ñoi : $\sum a_i * 2^i$ **vôi i** $\in$ **0...n**

a la ky so cua so can ñoi.

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

↓

a

Chuyển hệ tö hei 10 ➔ hei 16

Ñoi tö hei 10 ➔ hei 16 :

Ex : $253_d = ?_h$

Cach ñoi : lay soi can ñoi chia lien tiep cho 16, döng khi soi bò chia = 0. Ket quai lai chuoi soi dö lay theo chieu ngöôc lai.

$$\mathbf{253_d = FD_h}$$

↓

Chuyển hệ tö hei 2 ➔ hei 16

Ñoi tö hei 2 ➔ hei 16 :

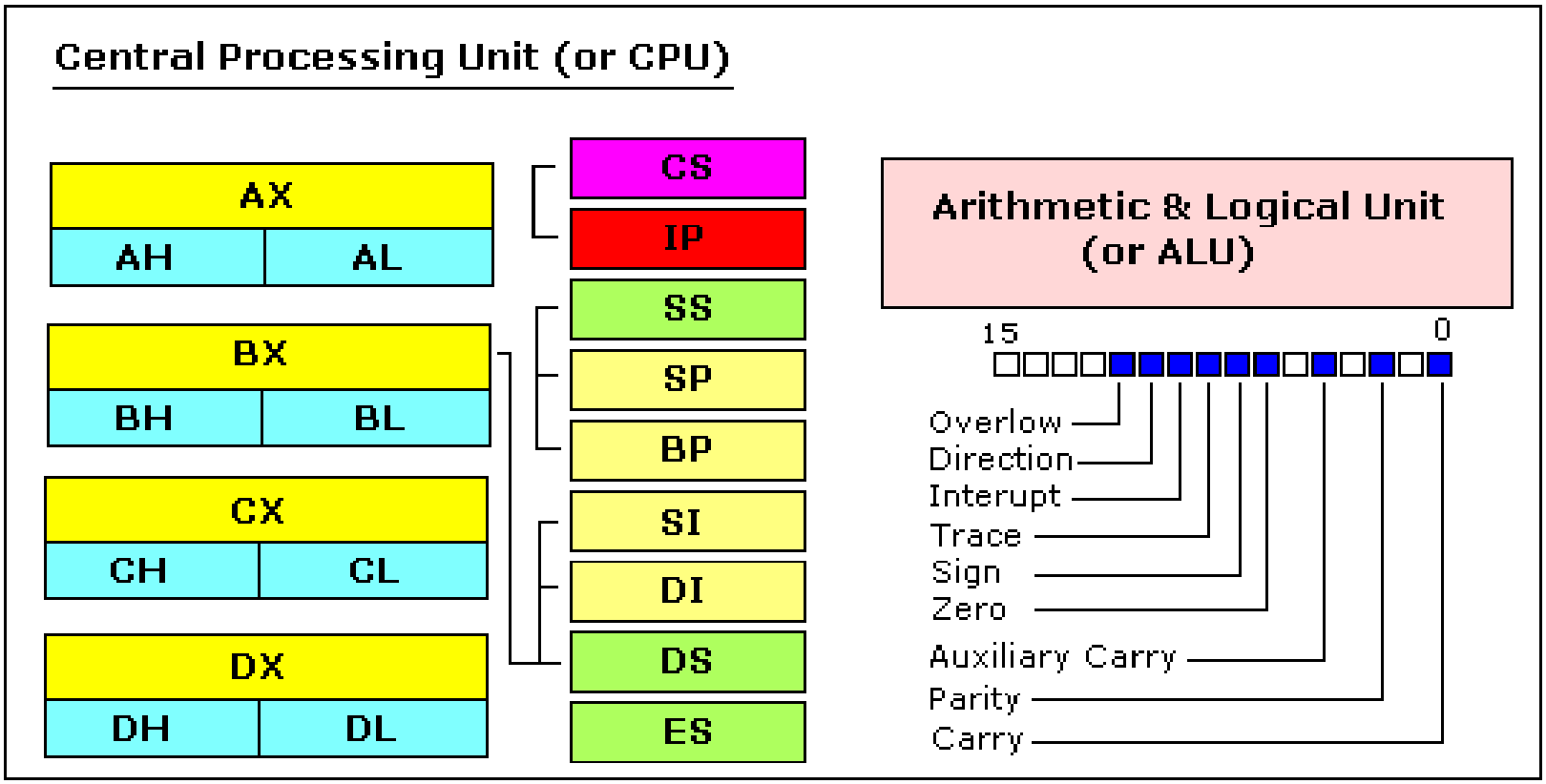
Ex : $101011010_b = ?_h$

Cach ñoi : nhom 4 chöi soi nhò phan thanh töng nhom, roi chuyen ñoi töng nhom sang soi hei thap luc phan.

$$0001\,0101\,1010_d = 15A_h$$

1 5 A

2.2 Bộ xử lý trung tâm CPU



2.2 Bộ xử lý trung tâm CPU

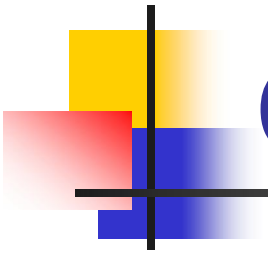
CPU (Central Processing Unit) Bộ xử lý trung tm –

Chức năng : thực hiện chương trình lưu trong bộ nhớ chính bằng cach lấy lệnh ra - khảo sat - thực hiện lần lượt cac lệnh.

Moi CPU co 1 tap lenh rieng. Chöông trình ñöôc thöc thi ô CPU nao se chø gom cac lenh trong tap lenh cua CPU ño.

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**

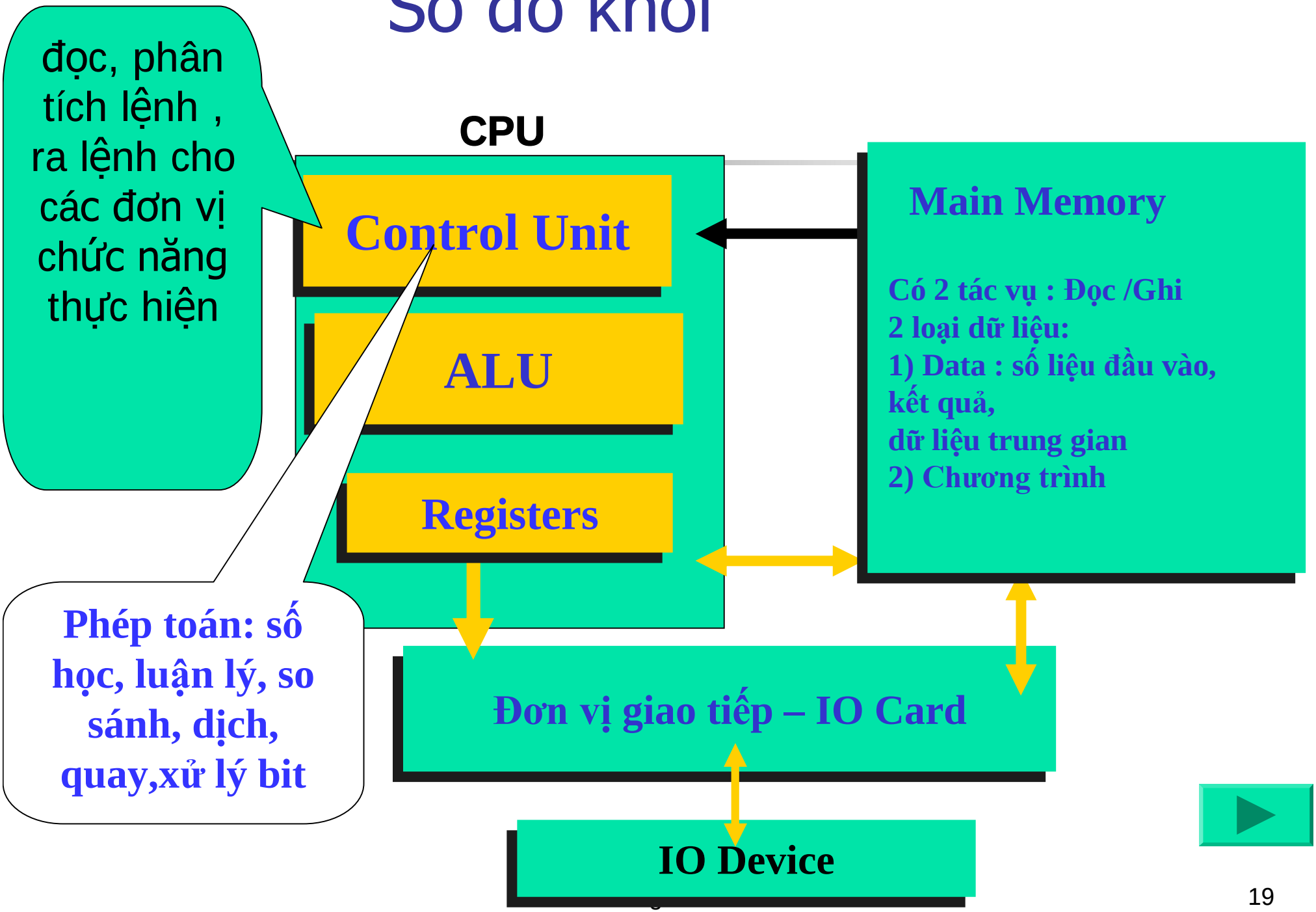


CPU (cont)

- Caùc nhaø cheá taïo CPU qui ñònh toác ñoä thöïc hieän cuûa töøng chip phuø hôïp vôùi nhòp tim cuûa chip ñoù (clock speed) toác ñoä ñoàng hoà, nhòp ñoàng hoà.
- Ñôn vò ño toác ñoä cuûa chip CPU laø Mhz cho bieát chip ñaïp bao nhieâu nhòp trong 1 s.
  Ex : CPU 500Mhz.

Sơ đồ khối

đọc, phân tích lệnh , ra lệnh cho các đơn vị chức năng thực hiện



Chu ky lenh

Mot chu ky thöc hien lenh may gom 3 giai ñoan chính sau :

1. **Lay lenh : lenh cat ô o nhô se ñöôc lay vao thanh ghi lenh.**
2. **Giai ma va thöc hien lenh : lenh trong thanh ghi lenh se ñöôc giai ma va thöc hien theo mo ta cua lenh trong tap lenh.**
3. **Xac ñònh ñòa chæ cua lenh tiep theo : trong khi lenh ñöôc thöc hien, gia trò cua bo ñem chöông trình se tö ñong tang len chæ ñen o nhô chöa lenh se ñöôc thöc hien tiep theo.**

Chu ky lenh ñöôc xay döng tö nhöng ñôn vò cô ban la chu ky may.

Chu kyø maùy

Chu kyø maùy laø chu kyø cuûa 1 hoaït ñoäng cô baûn cuûa maùy tính nhö :

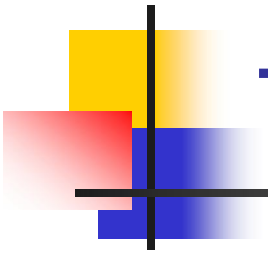
- Chu kyø ñoïc boä nhôù
- Chu kyø ghi boä nhôù
- Chu kyø ñoïc toaùn haïng
- Chu kyø ghi keát quaû

Clock : xung laøm nhieäm vuï ñònh thì cho maïch tuaàn töï.

Thöc hien lenh

CPU thöc hien lenh tuan tö theo chuoi cac böôc :

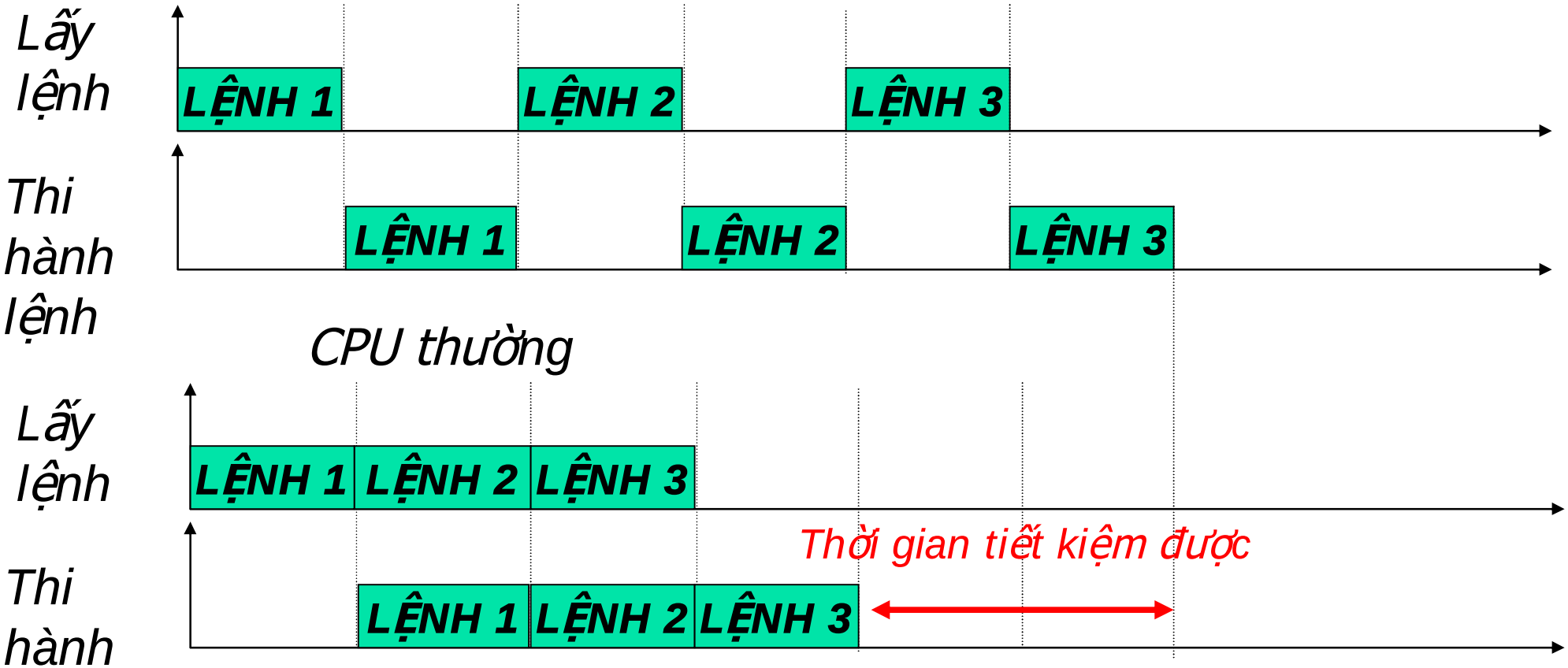
- Lay lenh ke tö bo nhô.→ thanh ghi lenh.
- Thay ñoi PC ñe chæ ñen lenh ke tiep.
- Xac ñònh kieu lenh vöa lay ra.
- Xac ñònh kieu dö lieu vöa yeu cau va xac ñònh vò trí dö lieu trong bo nhô.
- Neu lenh can dö lieu trong bo nhô, nap no vao thanh ghi cua CPU



Thöc hien lenh (cont)

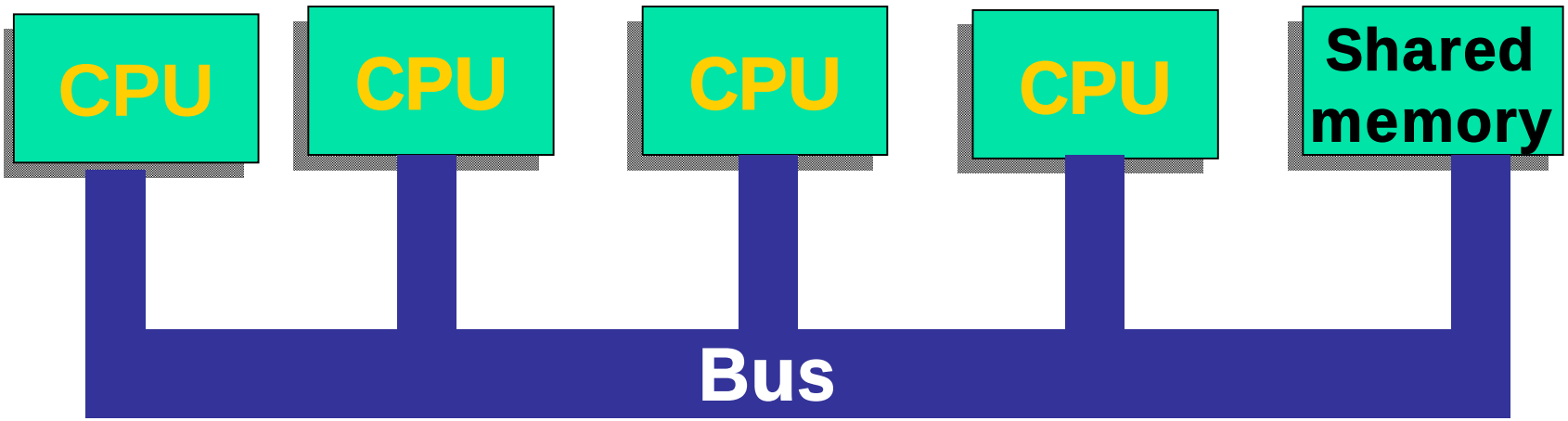
- Thöc hien lenh..
- Löu ket qua ô nôi thích hôp. .
- Trô về böôc 1 ñe thöc hien lenh ke.

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



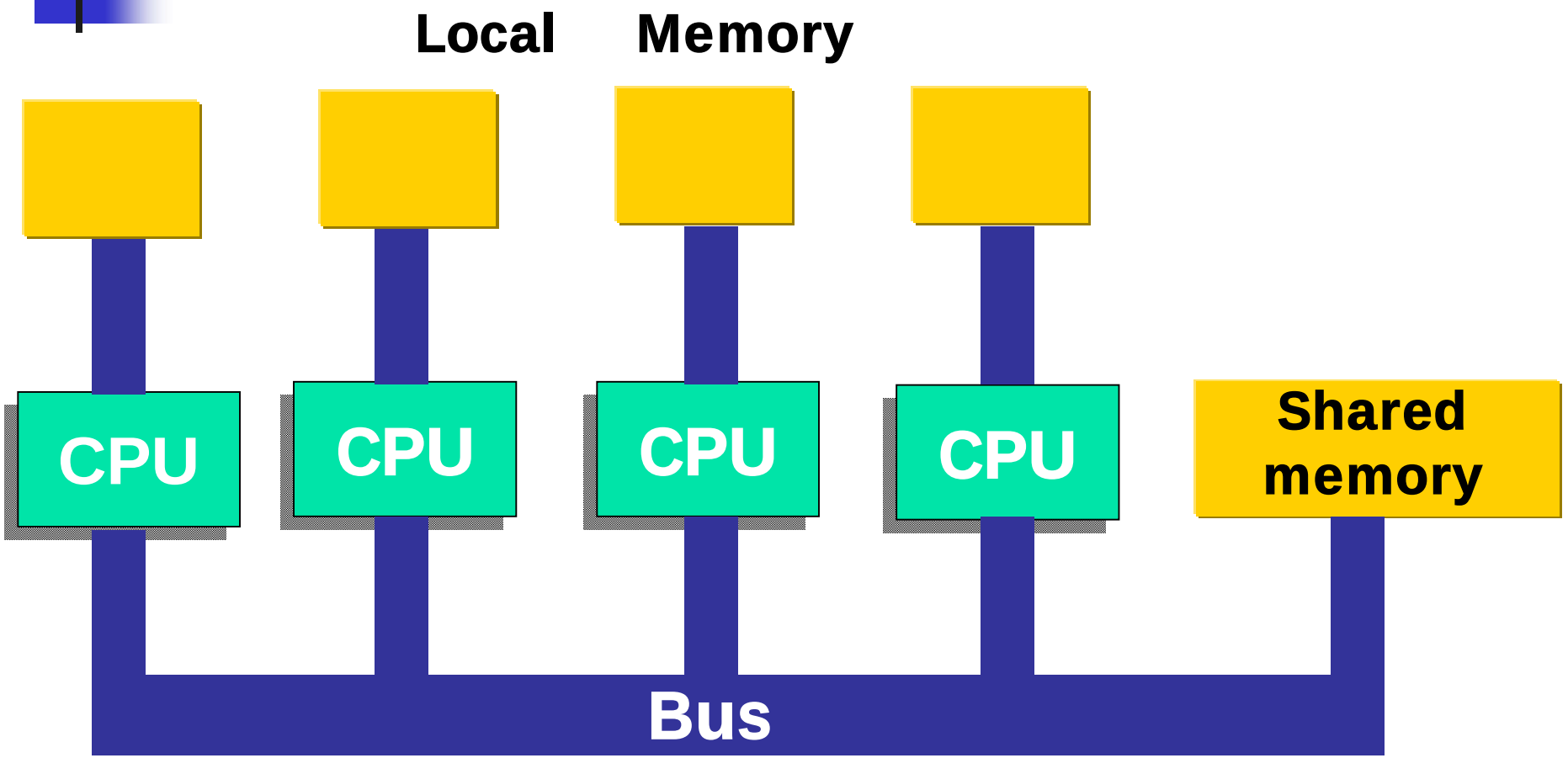
CPU dùng cơ cấu đường ống (piple line)

Heä ña boä xöû lyù (MultiProccessor)



Heä MultiProccessor söû duïng 1 ñöôøng Bus

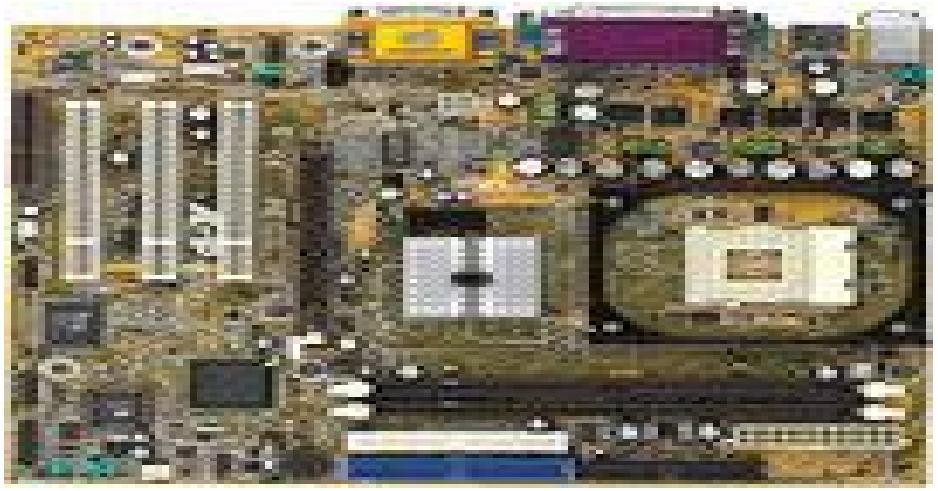
He ña bo xö ly (MultiProccessor)



He MultiProccessor sö dung nhieu bo nhô cuc bo

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).

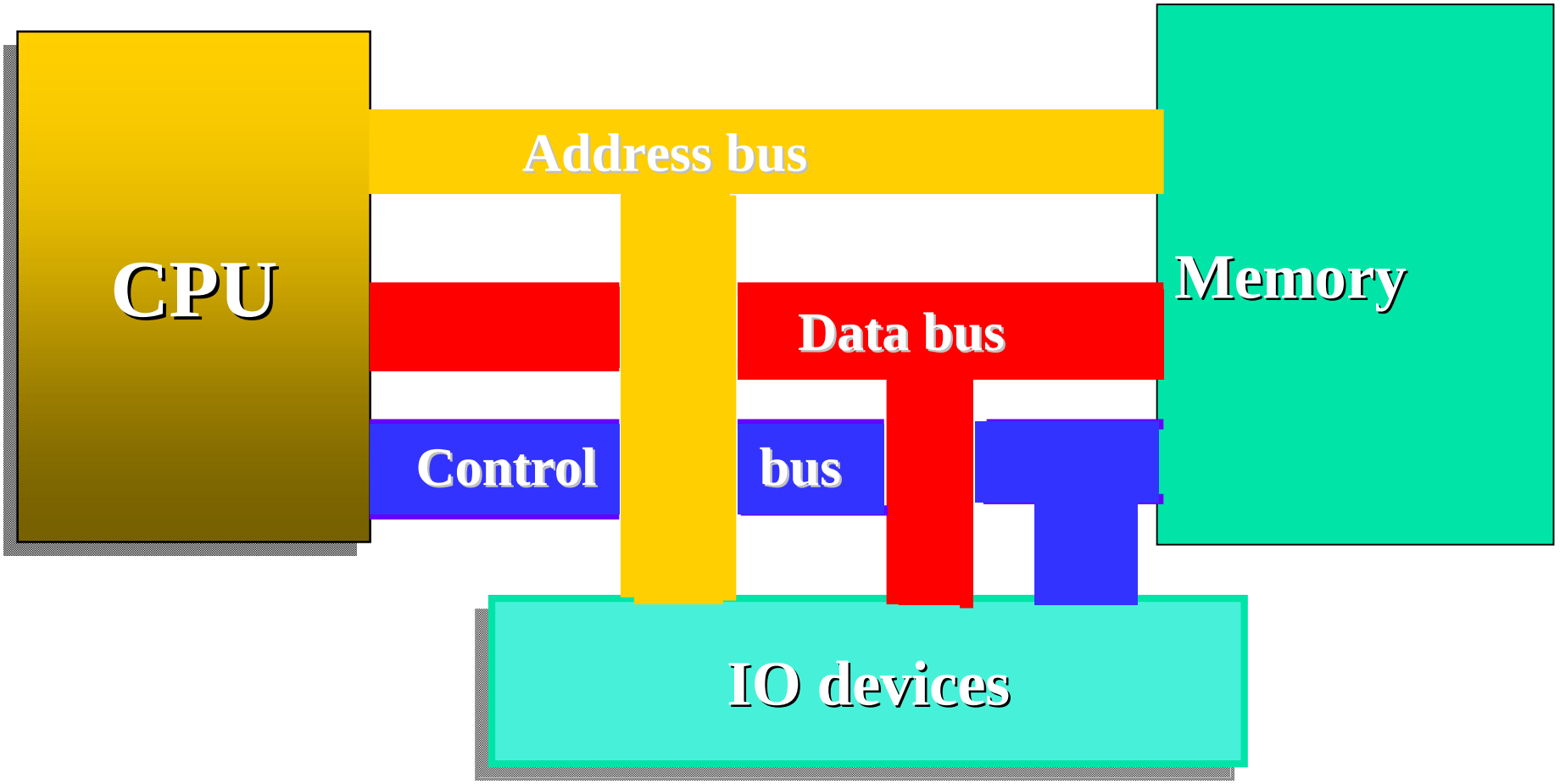
Các loại Bus

- **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.**

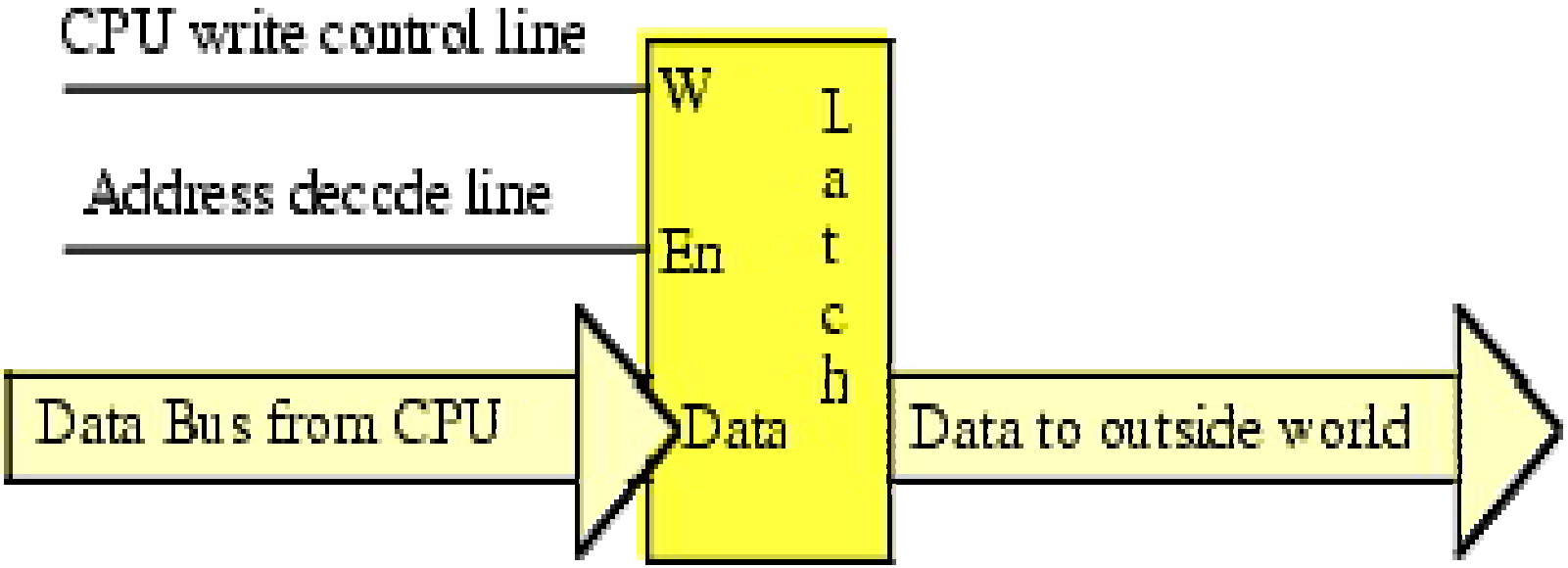
- **Data Bus : nhóm đường truyền để tải data thực sự giữa các thiết bị hệ thống do địa chỉ trên address bus đã xác định. Độ rộng của data bus (số đường dây dẫn) xác định data trong mỗi lần truyền là bao nhiêu.**

- **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…**

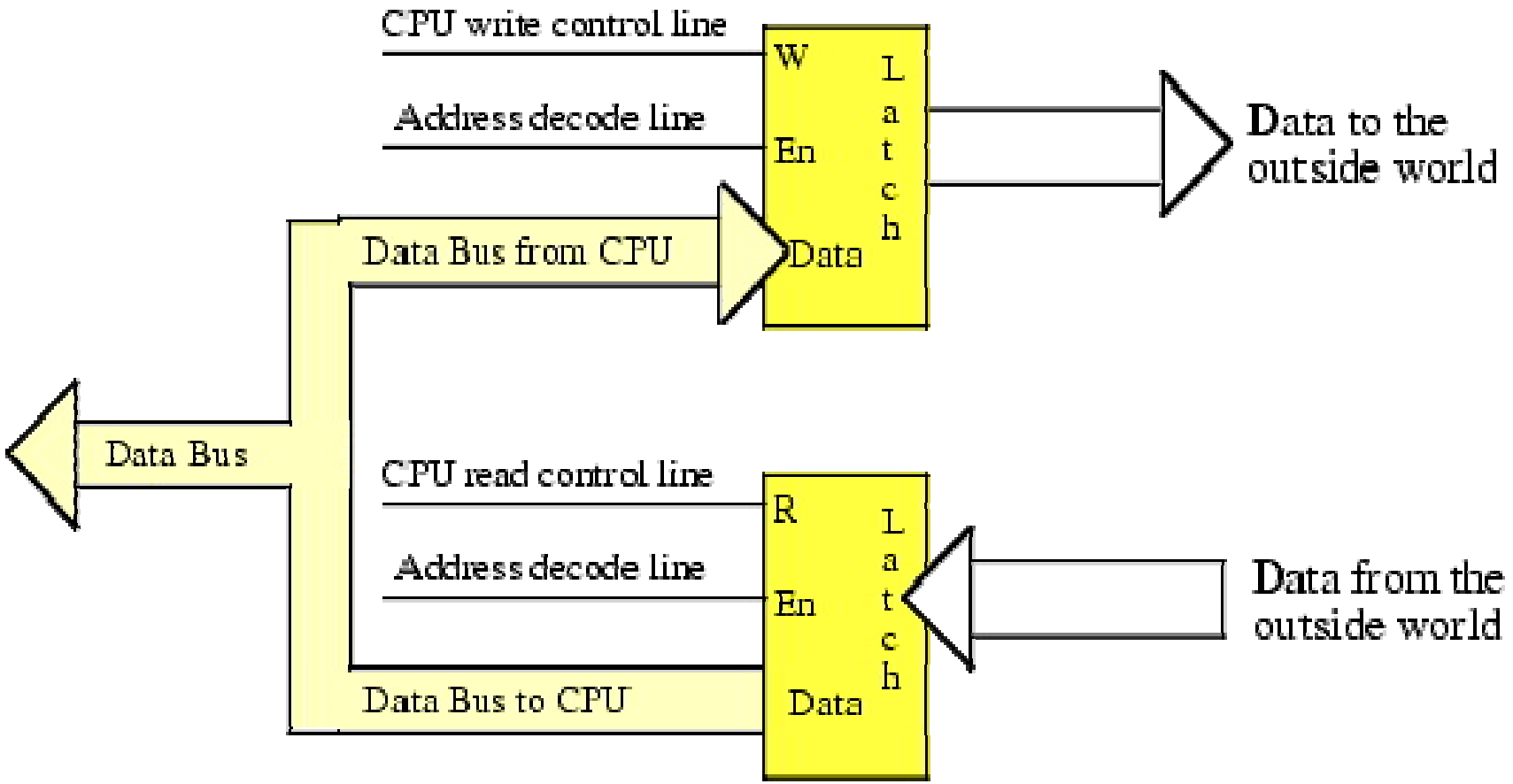
Minh họa hệ thống Bus



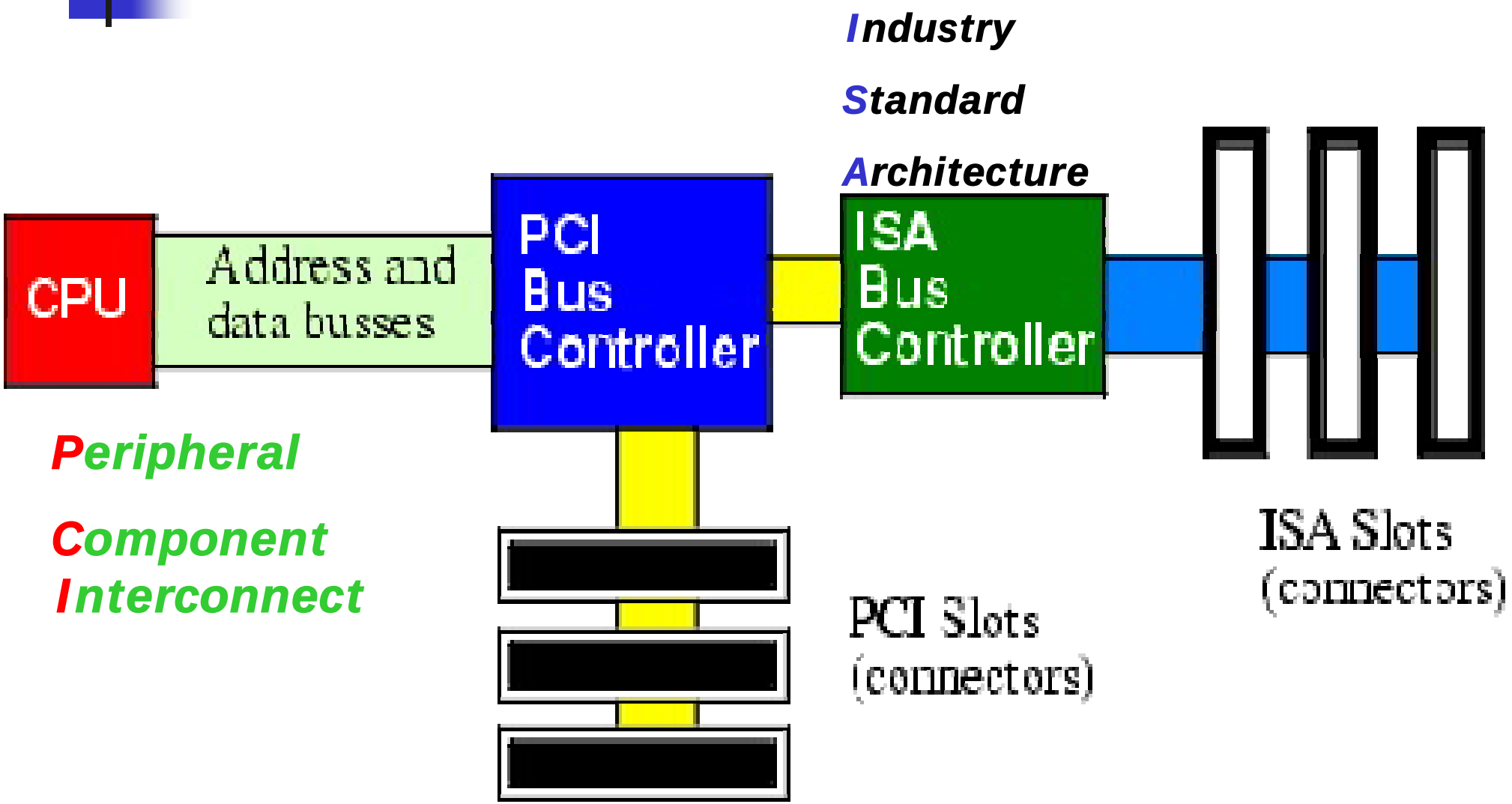
A Typical Output Port

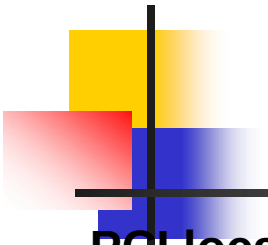


An Input and an Output Device That Share the Same Address (a Dual I/O Port)

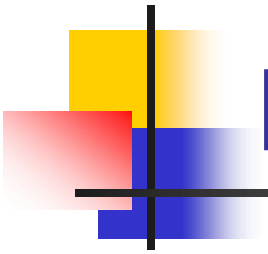


Connection of the PCI and ISA Busses in a Typical PC





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*

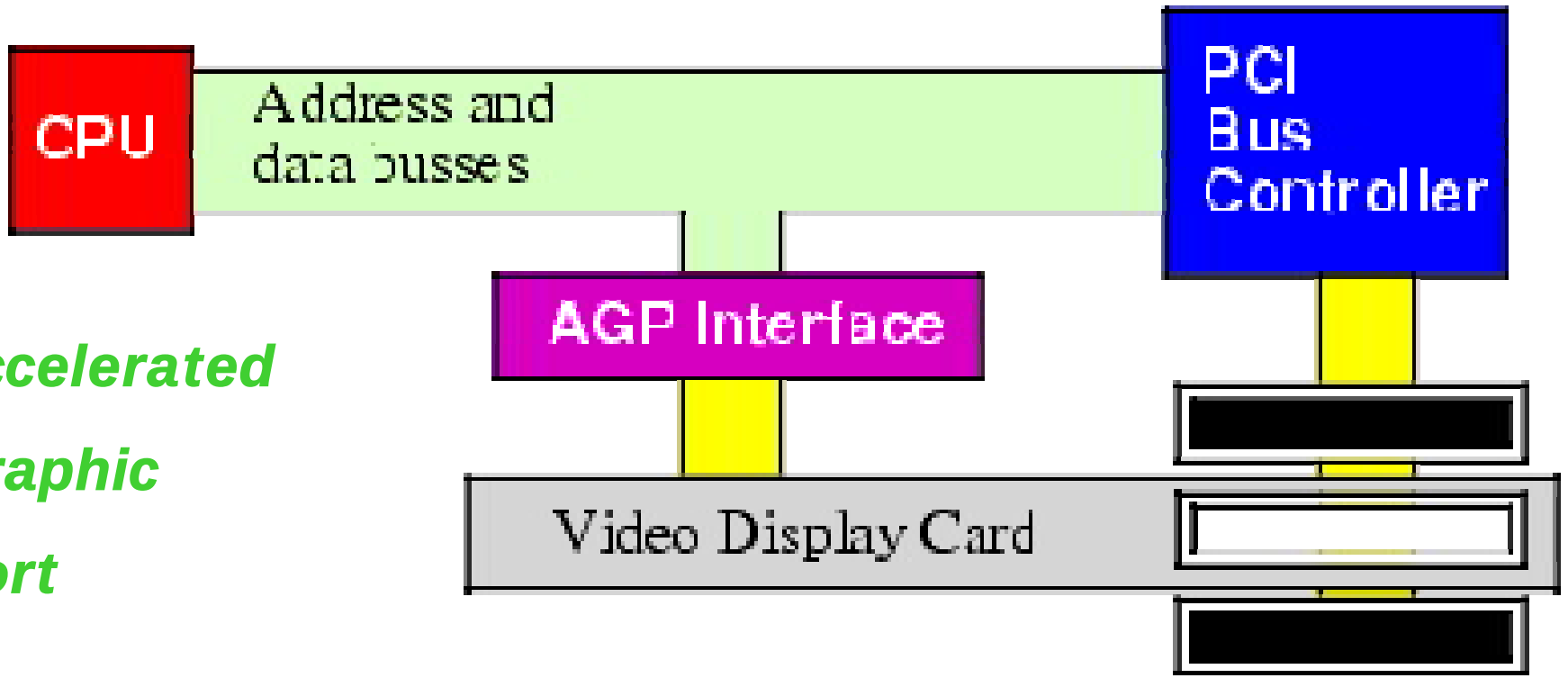
Plug and Play

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



Accelerated

Graphic

Port

AGP (**Accelerated Graphics Port**)

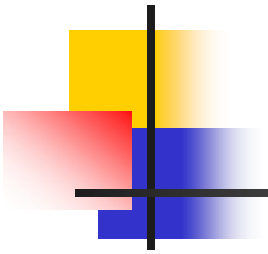
Acronym for Accelerated Graphics Port. A high-performance bus specification designed for fast, high-quality display of 3-D and video images. Developed by Intel Corporation, AGP uses a dedicated point-to-point connection between the graphics controller and main system memory. This connection enables AGP-capable 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

Adrress 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 l 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

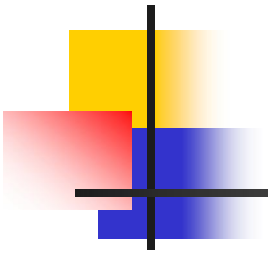
Mỗi chu kỳ bus là 1 tác vụ xãy ra trên bus để truyền tải data.

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.

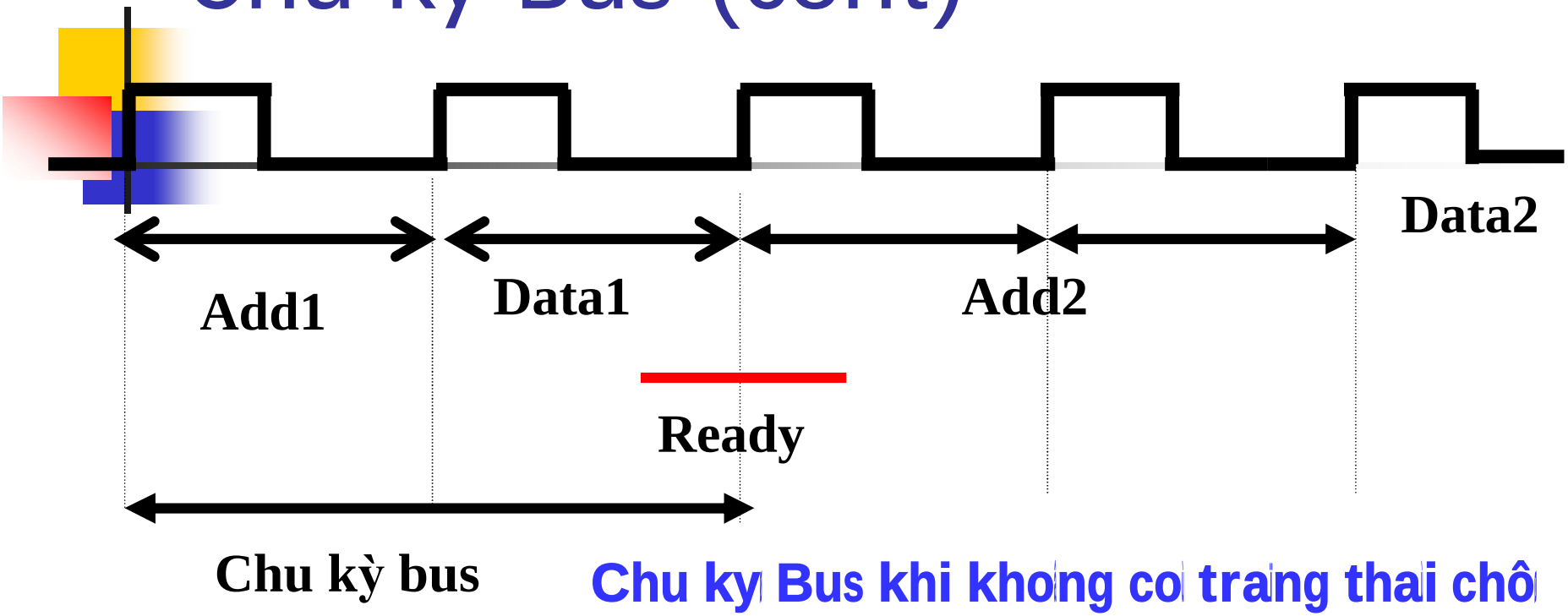
Đây là mốc tham chiếu theo thời gian để đồng bộ hoá tất cả các tác vụ bên trong máy tính. Xung đầu tiên gọi là Address time , địa chỉ truy xuất sẽ được gửi đi cùng với tín hiệu xác định loại tác vụ sẽ được thực thi (đọc/ghi/đến mem/đến I/O).

Chu kỳ Bus

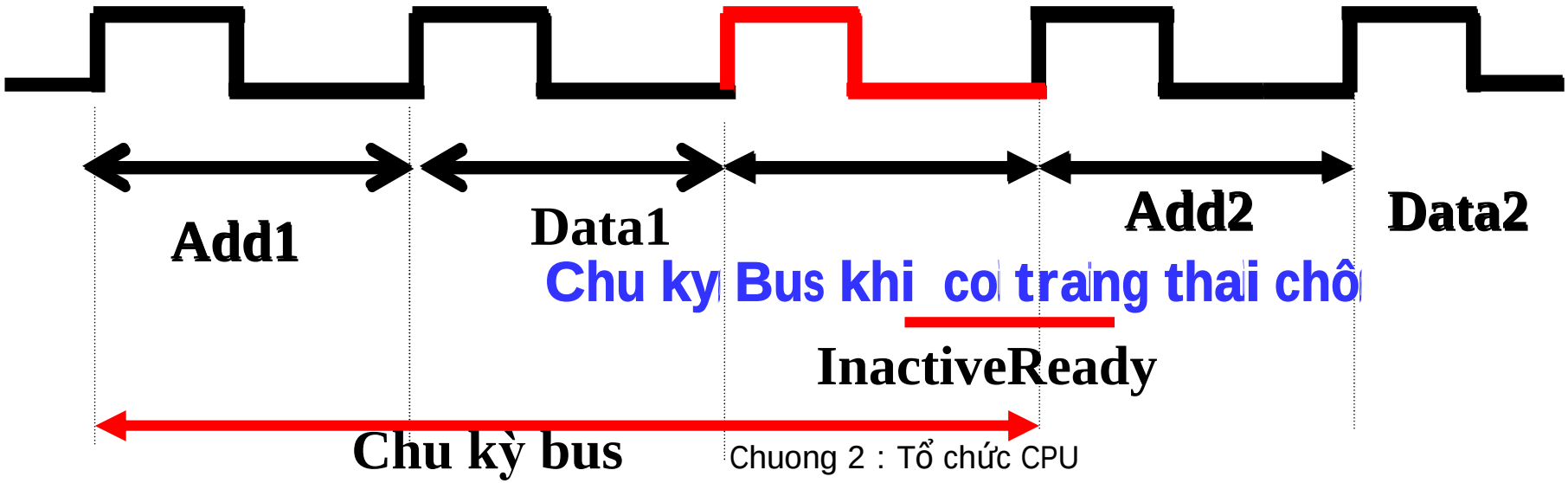
Cuối xung thứ 2, CPU sẽ kiểm tra đường tín hiệu Ready. Nếu thiết bị cần truy xuất sẵn sàng đáp ứng tác vụ, thiết bị này sẽ kích 1 tín hiệu lên đường Ready để báo cho CPU biết và chu kỳ bus hoàn tất.

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 ky Bus khi khong co trang thai chô



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 vỉ 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

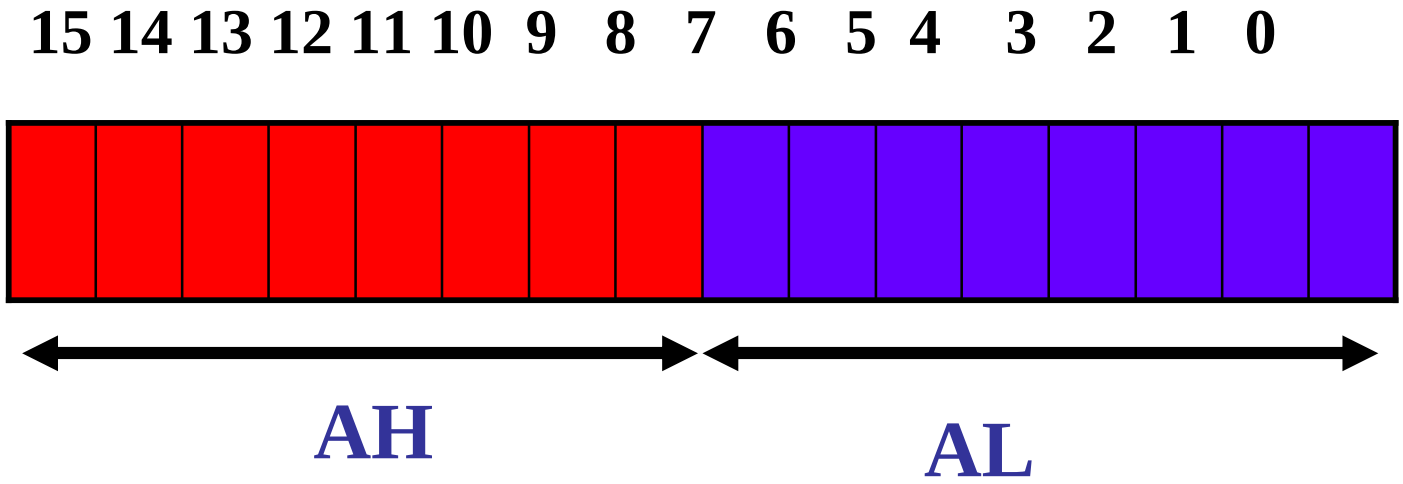
2.4 Hệ thống thanh ghi

- **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.**

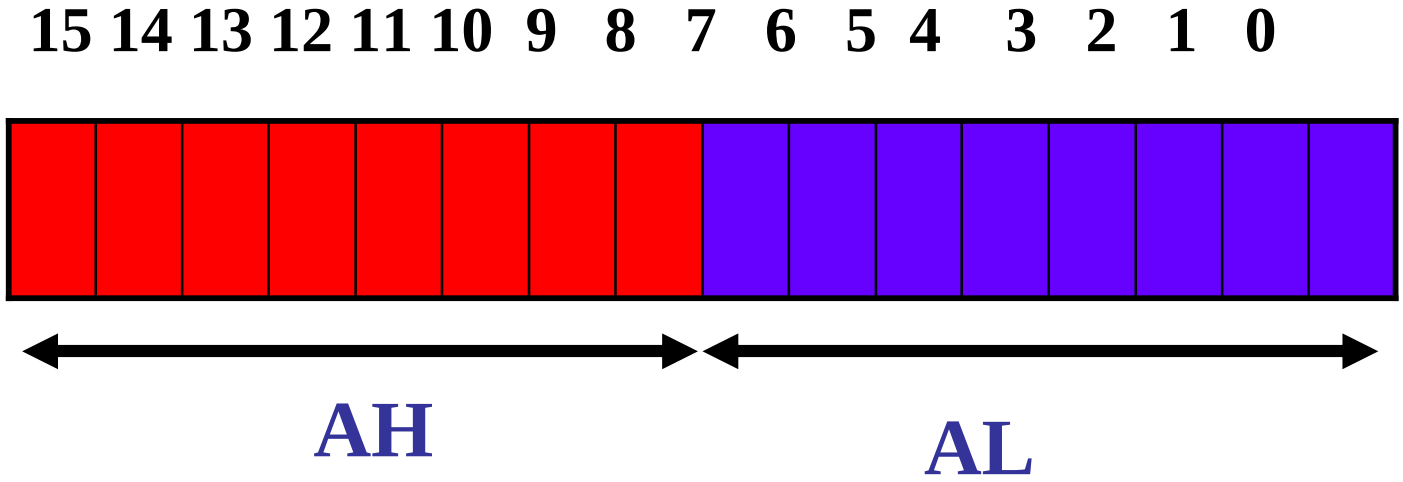
AX Register



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

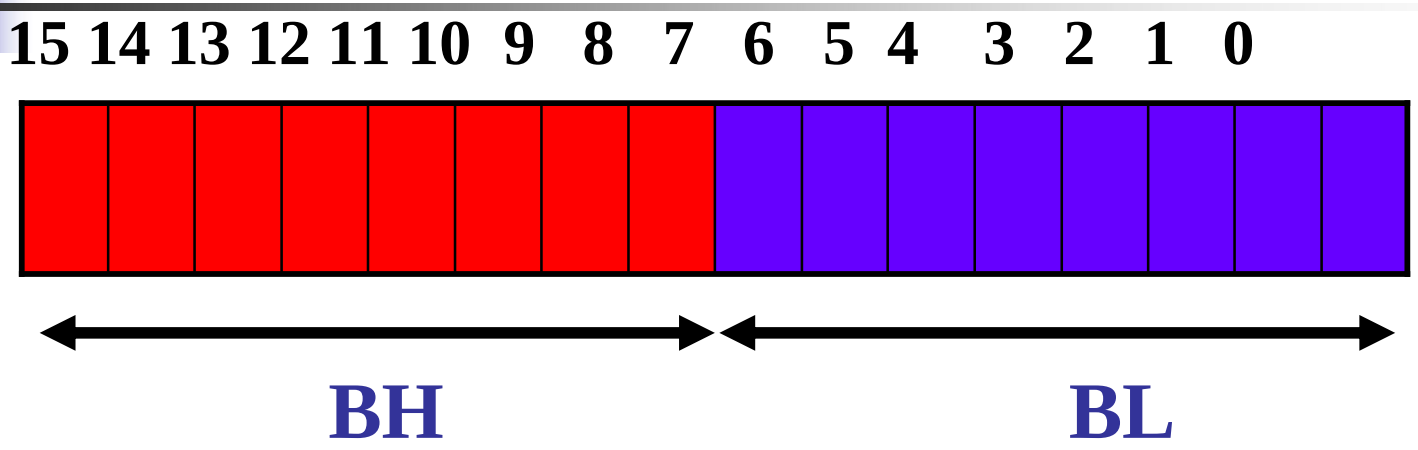


Thanh ghi AH là nửa cao của thanh ghi AX

Thanh ghi AL là nửa thấp của thanh ghi AX

Thí dụ nếu AX=1234h thì AH=12H AL=34h

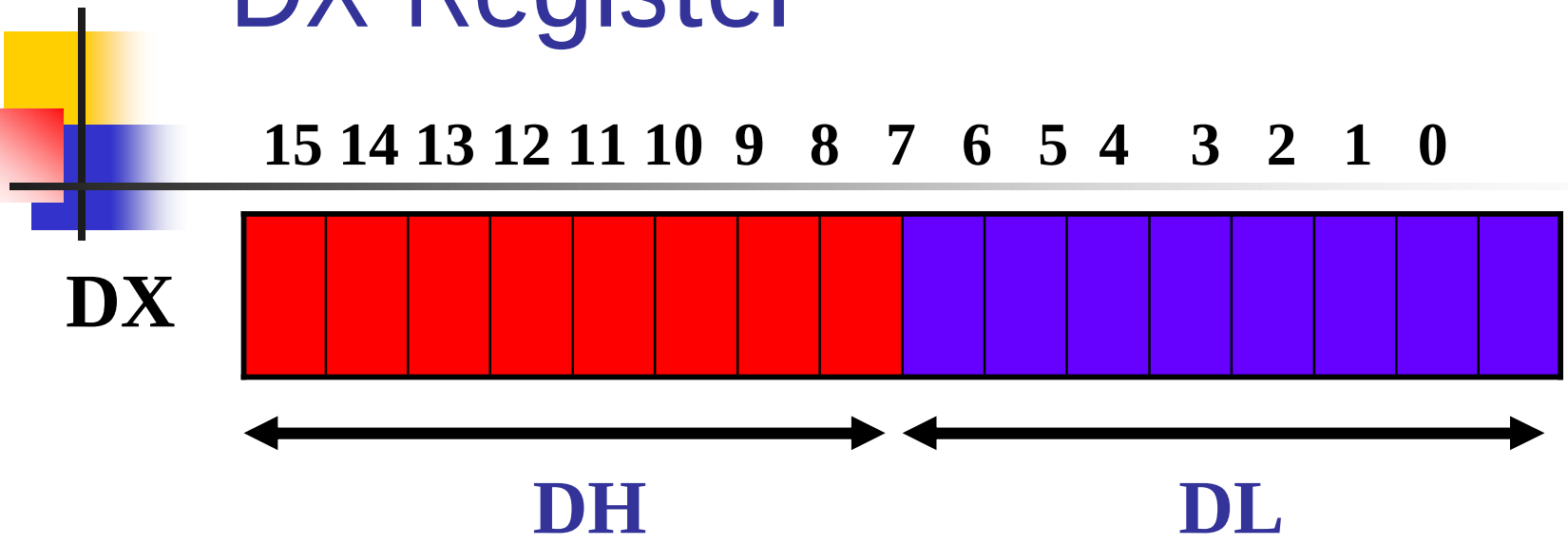
BX Register



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

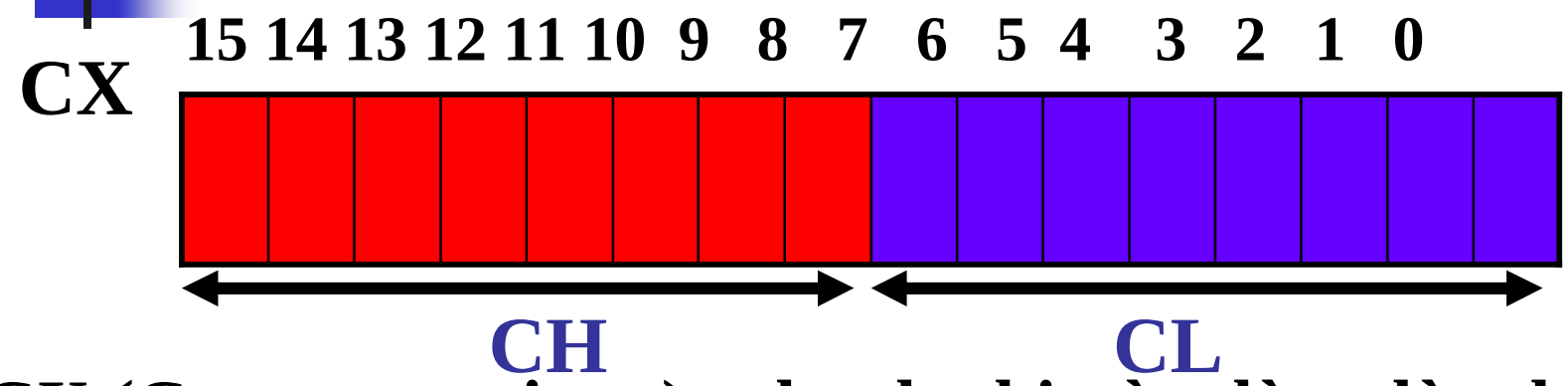


Thanh ghi DX (Data register) : dài 16 bit nhưng nó cũng có thể chia làm 2 thanh ghi 8 bit DH và DL

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

- **Ñoi vôi mot so CPU ñôi môi, co cac thanh ghi dai 32, 64 bit. Ta ghi them E ñöng tröôc ten cac 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.**

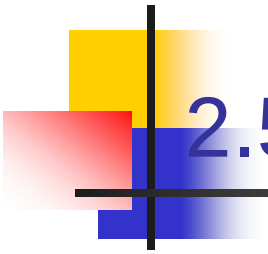
SỰ PHÂN ĐOẠN BỘ NHỚ

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:

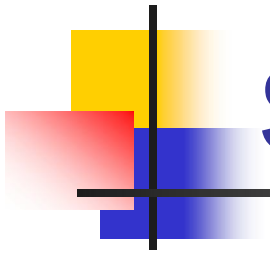
Segment : offset ⟶ *Địa chỉ trong đoạn (độ dời)*

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

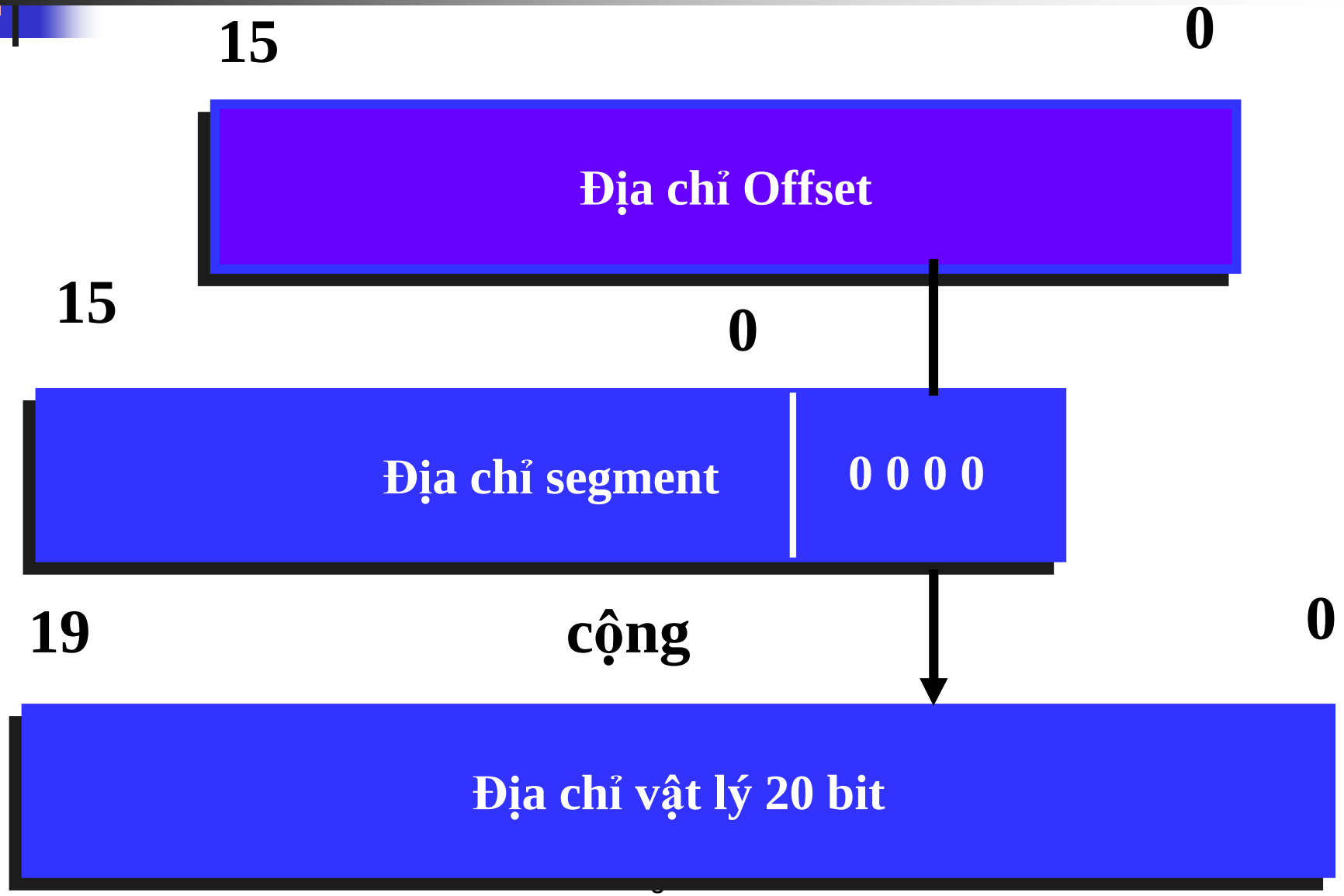
Hãng Intel đề xuất 1 phương pháp để hình thành địa chỉ.

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 segemnt, 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 hoạ hình thành địa chỉ



Sự hình thành địa chỉ tuyệt đối

địa chỉ segment

địa chỉ Offset

Gỉa sử ta có địa chỉ **08F1 : 0100**

địa chỉ tương đối

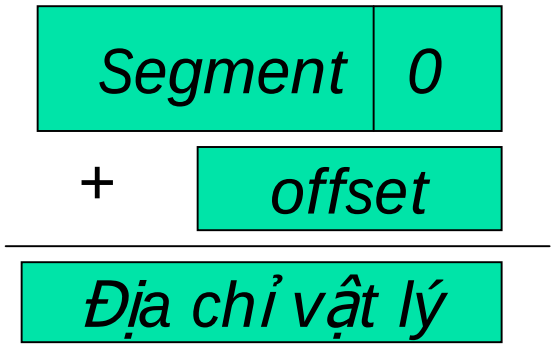
CPU tự động lấy địa chỉ segment x 10 (hệ 16) thành 08F10

Sau đó nó cộng với địa chỉ Offset 0100

➔ **địa chỉ tuyệt đối : 09010**

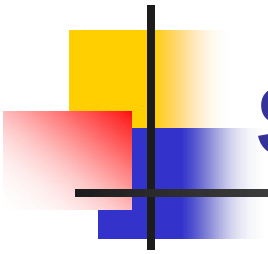
Cách tính địa chỉ vật lý từ địa chỉ luận lý

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



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

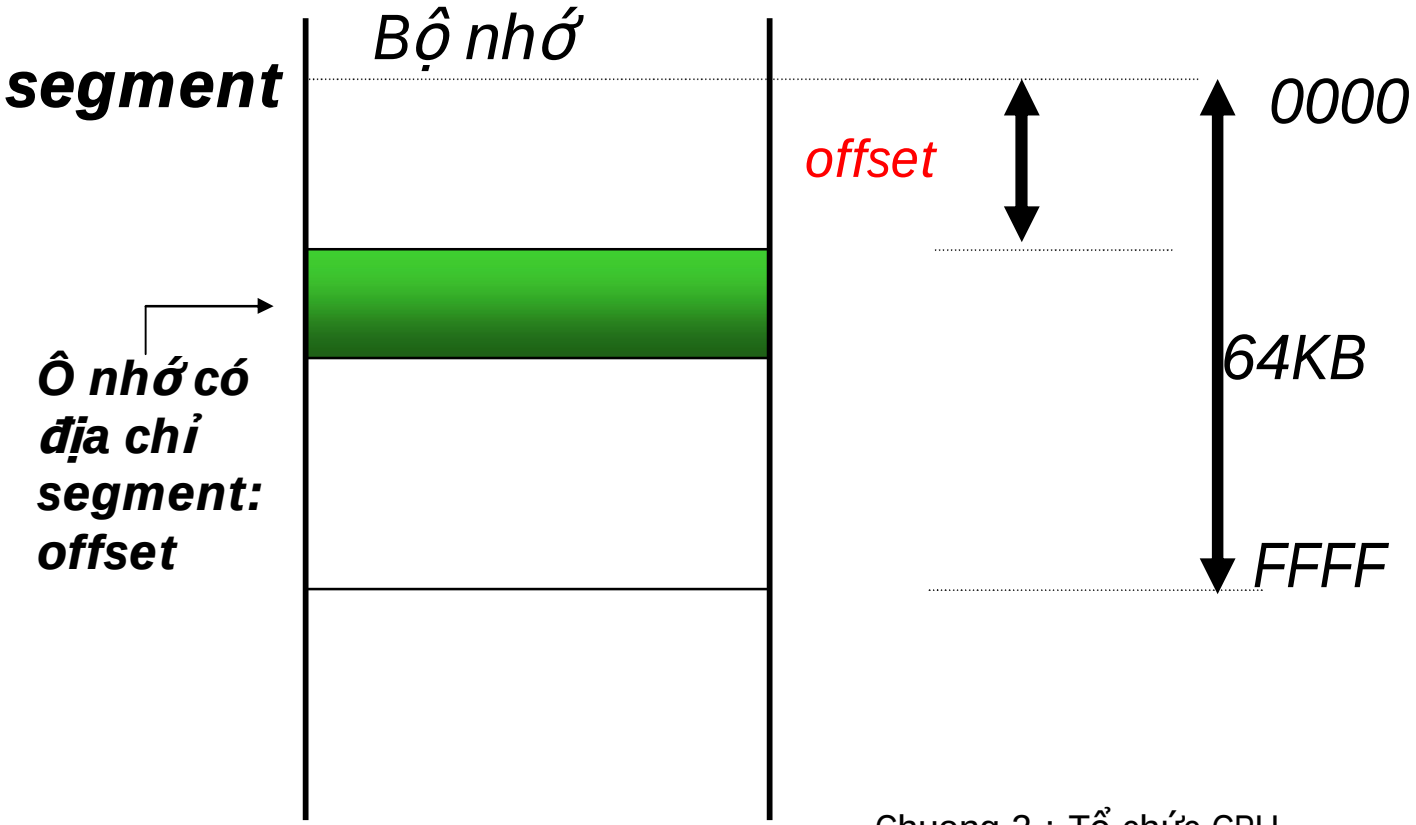
Địa chỉ segment hay còn gọi là địa chỉ nền của đoạn. Nó cho biết điểm bắt đầu của đoạn trong bộ nhớ.

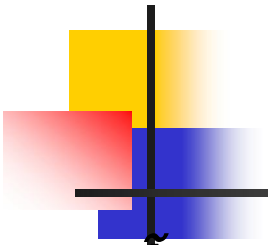
Địa chỉ offset thể hiện khoảng cách kể từ đầu đọan của ô nhớ cần tham khảo.

Do offset dài 16 bit nên chiều dài tối đa của mỗi đọan 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

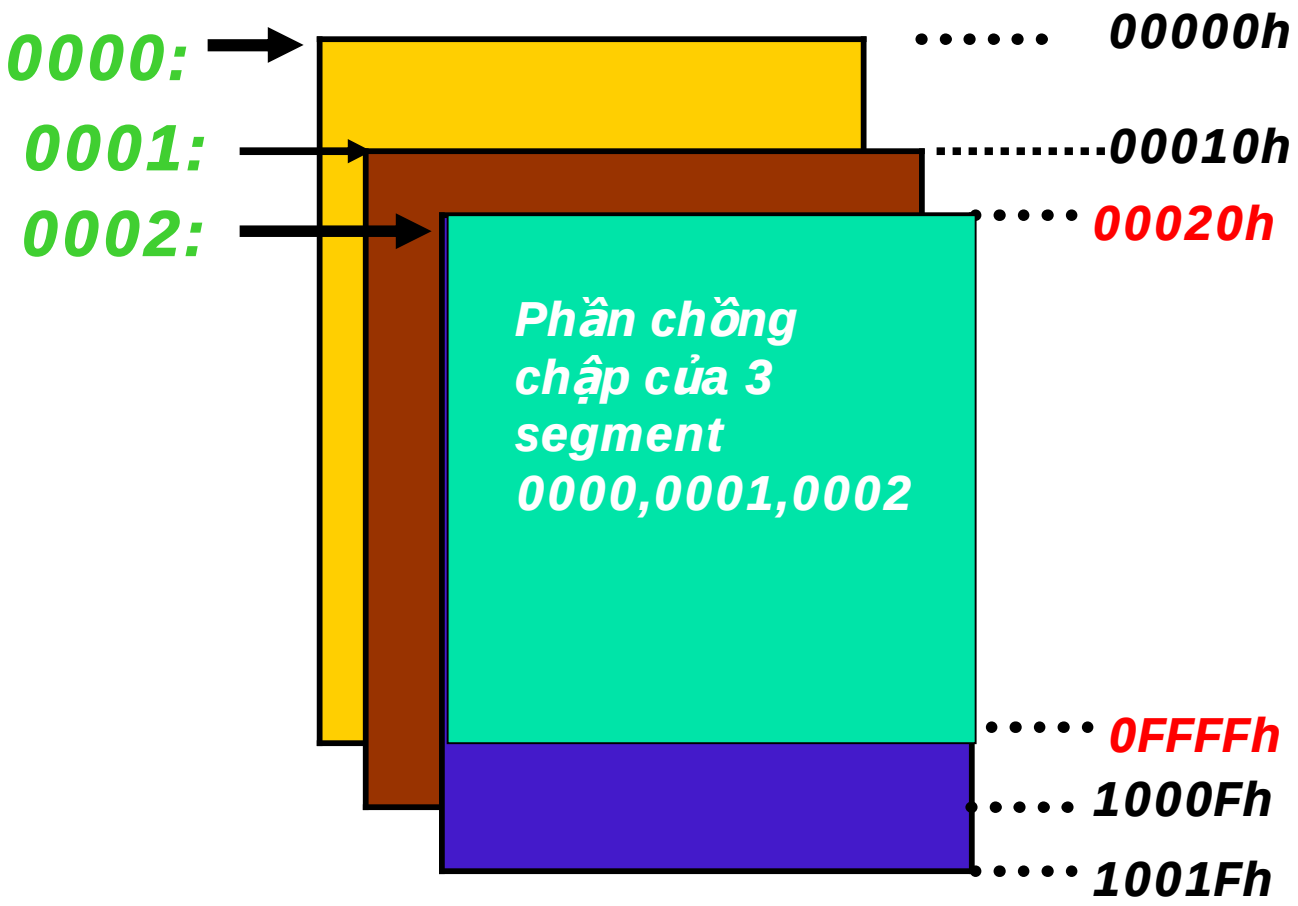
1334:0234

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

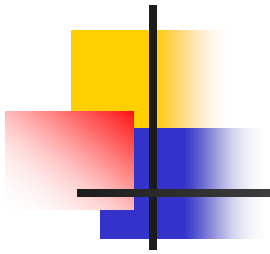
0000:0000 ➔ 00000h

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



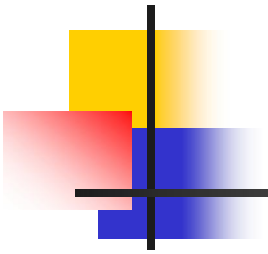
Làm lại quá trình trên nhưng giữ nguyên phần offset chỉ tăng phần segment.

0001:0000 ➔ 00010h

0002:0000 ➔ 00020h

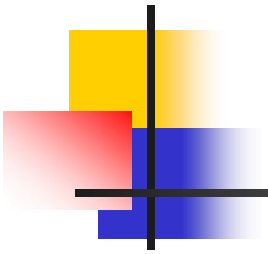
Khi segment tăng 1 đơn vị thì địa chỉ vật lý tăng 10h địa chỉ hoặc là tăng 16 bytes

Đơn vị của segemnt là paragraph



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

Phần chồng 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 0FFFFh) đều có thể có địa chỉ luận lý tương ứng trong cả 3 segment.

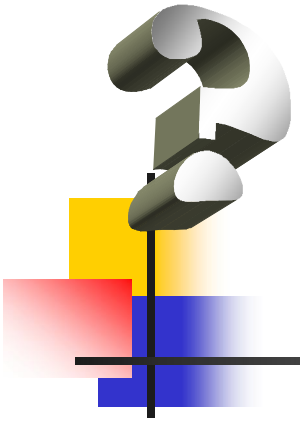


Ex : ô nhớ có địa chỉ 0002Dh sẽ có địa chỉ logic trong segment 0000 là 0000:002D

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 ñoan CS, DS, SS, ES

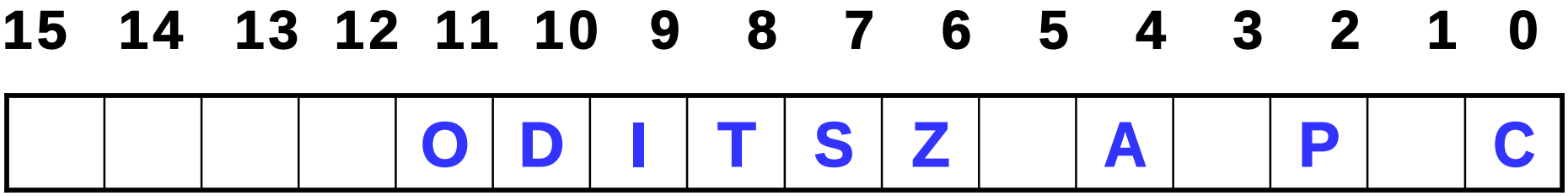
- 3 trong 4 thanh ghi ñoan ñöôc dung trong cac muc ñích ñac biet sau
- **CS** : xac ñònh ñoan lenh – nôi chöa chöông trình ñöôc thi hanh.
- **DS** : xac ñònh ñoan dö lieu – nôi chöa chöông trình ñöôc thi hanh.
- **SS** : xac ñònh ñoan stack – vung lam viec tam thôi dung ñe theo doi cac tham so va cac ñòa chæ ñang ñöôc chöông trình hien hanh sö dung.
- Con thanh ghi ES : tro ñen ñoan them, thöông ñöôc dung ñe bo sung cho ñoan dö lieu → co vung nhô >64k cho ñoan dö lieu.

Các thanh ghi ñoan CS, DS, SS, ES

- 3 trong 4 thanh ghi ñoan ñöôc dung trong cac muc ñích ñac biet sau
- **CS** : xac ñònh ñoan lenh – nôi chöa chöông trình ñöôc thi hanh.
- **DS** : xac ñònh ñoan dö lieu – nôi chöa chöông trình ñöôc thi hanh.
- **SS** : xac ñònh ñoan stack – vung lam viec tam thôi dung ñe theo doi cac tham so va cac ñòa chæ ñang ñöôc chöông trình hien hanh sö dung.
- Con thanh ghi ES : tro ñen ñoan them, thöông ñöôc dung ñe bo sung cho ñoan dö lieu → co vung nhô >64k cho ñoan dö lieu.

Thanh ghi traïng thaùi (thanh ghi cô)

- **Thanh ghi cô laø thanh ghi 16 bit naèm beân trong EU (Excution Unit). Tuy nhieân chæ coù 9 trong 16 bit ñöôïc söû duïng.7 bit con laïi khoâng duïng.**



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 trang thai (thanh ghi cô)

Giai thích :

Cô CF : chæ thò cong co nhô, trö co möôn.

Cô PF : On khi ket qua cua tac vu co so bit 1 la so chan.

Neu so bit 1 la so le thì PF la Off.

Cô AF : co nhô trong phep cong hoac co möôn trong phep trö vôi 4 bit thap sang 4 bit cao.

Cô ZF : On khi tac vu luan ly cho ket qua la 0.

Cô SF : bit cao nhat cua ket qua se ñöôc copy sang SF. SF =1 ket qua la so am. SF = 0 khi ket qua la so döong.

Thanh ghi trang thai (thanh ghi cô)

Giai thích :

Cô OF : OF=1 khi ket qua bò tran so (vöôt qua kha nang löu trö). Neu ket qua khong bò tran thì OF=0.

3 bit con lai la 3 bit ñieu khien :

Cô TF : bao CPU thi hanh töng böôc. Cung cap cong cu debug chöông trình.

Cô IF : IF=1 giup 8086 nhan biet co yeu cau ngat quang co che.

Cô DF : xac ñònh höông theo chieu tang/giam trong xö ly chuoi.

8086 cho phep User lap trình bat tat cac cô CF,DF,IF,TF

Thanh ghi chæ soá (Index)

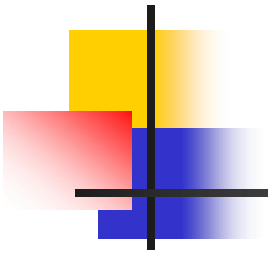
5 thanh ghi offset dung ñeå xaùc ñònh chích xaùc 1 byte hay 1 word trong 1 ñoaïn 64K.Ño laø :

- IP : thanh ghi con troû leänh, cho bieát vò trí cuûa leänh hieän haønh trong ñoaïn leänh. Con troû leänh IP con ñöôïc goïi laø boä ñeám chöông trình.
  Thöông ñöôïc dung keát hôïp vôùi CS ñeå theo doõi vò trí chính xaùc cuûa leänh seõ ñöôïc thöïc hieän keá tieáp.

Thanh ghi chæ soá (Index)

- Caùc thanh ghi con troû Stack : SP vaø BP, moãi thanh ghi daøi 16 bit.
- SP (Stack pointer) cho bieát vò trí hieän haønh cuûa ñænh Stack.
- BP (Basic Pointer) duøng ñeå truy caäp döõ lieäu trong Stack.
- SI (source index) : troû ñeán oâ nhôù trong ñoaïn döõ lieäu ñöôïc ñònh ñòa chæ bôûi thanh ghi DS.
- DI (destination) : chöùc naêng töông töï SI.
  Hai thanh ghi naày thöôøng duøng trong xöû lyù chuoãi.

ĐỊA CHỈ LUẬN LÝ VÀ THANH GHI

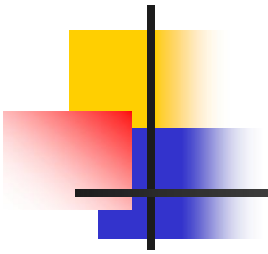


Để tham khảo đến bộ nhớ trong chương trình, VXL 8086 cho phép sử dụng các địa chỉ luận lý 1 cách trực tiếp hoặc thông qua các thanh ghi của nó.

Thanh ghi đoạn dùng để chứa segment

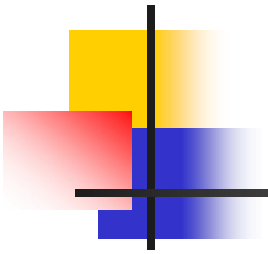
Thanh ghi tổng quát dùng để chứa địa chỉ trong đoạn offset

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



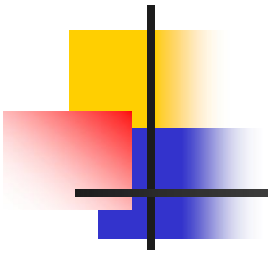
Ex : nếu lúc tham khảo

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 chuổi nguồn
ES:DI địa chỉ chuổi đích



Chương trình mà VXL 8086 thi hành thường có 3 đọan :

Đọan chương trình có địa chỉ trong thanh ghi CS.

Đọan dữ liệu có địa chỉ trong thanh ghi DS.

Đọan stack có địa chỉ trong thanh ghi SS.

Caùc ñaëc tính cuûa CPU Intel

- Hieäu quaû cuûa CPU thuoäc hoï Intel khi xöû lyù vaø chuyeån giao thoâng tin ñöôïc xaùc ñònh bôûi caùc yeáu toá sau :
- Taàn soá maïch xung ñoàng hoà cuûa CPU.
- Ñoä roäng cuûa Data bus
- Ñoä roäng cuûa Address bus

Caùc ñaëc tính cuûa CPU Intel

- Taàn soá maïch xung ñoàng hoà cuûa CPU.caøng nhanh thì toác ñoä xöû lyù caøng nhanh.
- Ñoä roäng cuûa Data bus caøng roäng thì caøng nhieàu data ñöôïc chuyeån giao trong 1 laàn giao dòch.
- Ñoä roäng cuûa Address bus caøng roäng thì khaû naêng quaûn lyù boä nhôù caøng lôùn.

Caùc ñaëc tính cuûa CPU Intel

Loaïi CPU	Data Bus (bit)	Address bus (bit)	Khaû naêng quaûn lyù boä nhôù
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**

Tom tat CPU ho Intel

- **CPU 80286 : Data bus 16 bit nen moi lan chuyen giao 2 bytes ➔ quan ly 16MB bo nhô.**
  Chæ co kha nang thöc hien cac phep toan ñoi vôi cac so nguyen, co the dung tap lenh 80286 ñe mo phong cac phep toan so hoc dau cham ñong nhöng ñieu nay se lam giam hieu suat he thong.

- **Neu muon co kha nang thöc hien cac phep toan dau cham ñong phai gan CoProccessor 8087.**

80286 lam viec theo 2 che ño : che ño thöc va che ño bao ve.

Tom tat CPU ho Intel

- CPU 80386 : Data bus 32 bit nen co the quan ly 4GB bo nhô. Cac thanh ghi dai 32 bit ➔ tang ño chính xac cua cac phep toan. Ño rong Bus ➔ tang toc ño thöc thi.

CPU 80386 hoan toan töông thích vôi cac CPU tröôc no.

Tom tat CPU ho Intel

- **CPU 80486 : co bus 32 bit . 1 Coprocessor 387, bo phan ñieu khien Cache, 1 Cache 8K, dung phoi hôp tap lenh rut gon RISC va tap lenh phöc tap CISC.**

CPU 80486 phan lôn cac lenh chæ dung 1 so ít xung.

Sö dung cô che ñöông ong co kha nang xö ly 5 lenh ñong thôi :

- **Lay lenh tröôc PreFetch**
- **Giai ma lan 1 Decode 1**
- **Giai ma lan 2 Decode 2**
- **Thöc thi lenh Execution**
- **Ghi lai trang thai. WriteBack**

RISC & CISC

- **Nguyen ly CISC** :

Complex Intruction Set Computer

- **Tap lenh kha lôn >300 lenh**
- **Kha nang ñònh vò phöc tap**
- **Mot so lenh can phai vi lenh hoa**

qua nhieu lenh → nap lau → lam cham he thong

lenh phöc tap→ nen time giai ma lenh nhieu khi lôn hôn time thöc thi.

Chæ co hôn 20% lenh thöông dung tôi

RISC & CISC

- **Nguyen ly RISC : tap lenh thu gon**

Reduce Intruction Set Computer

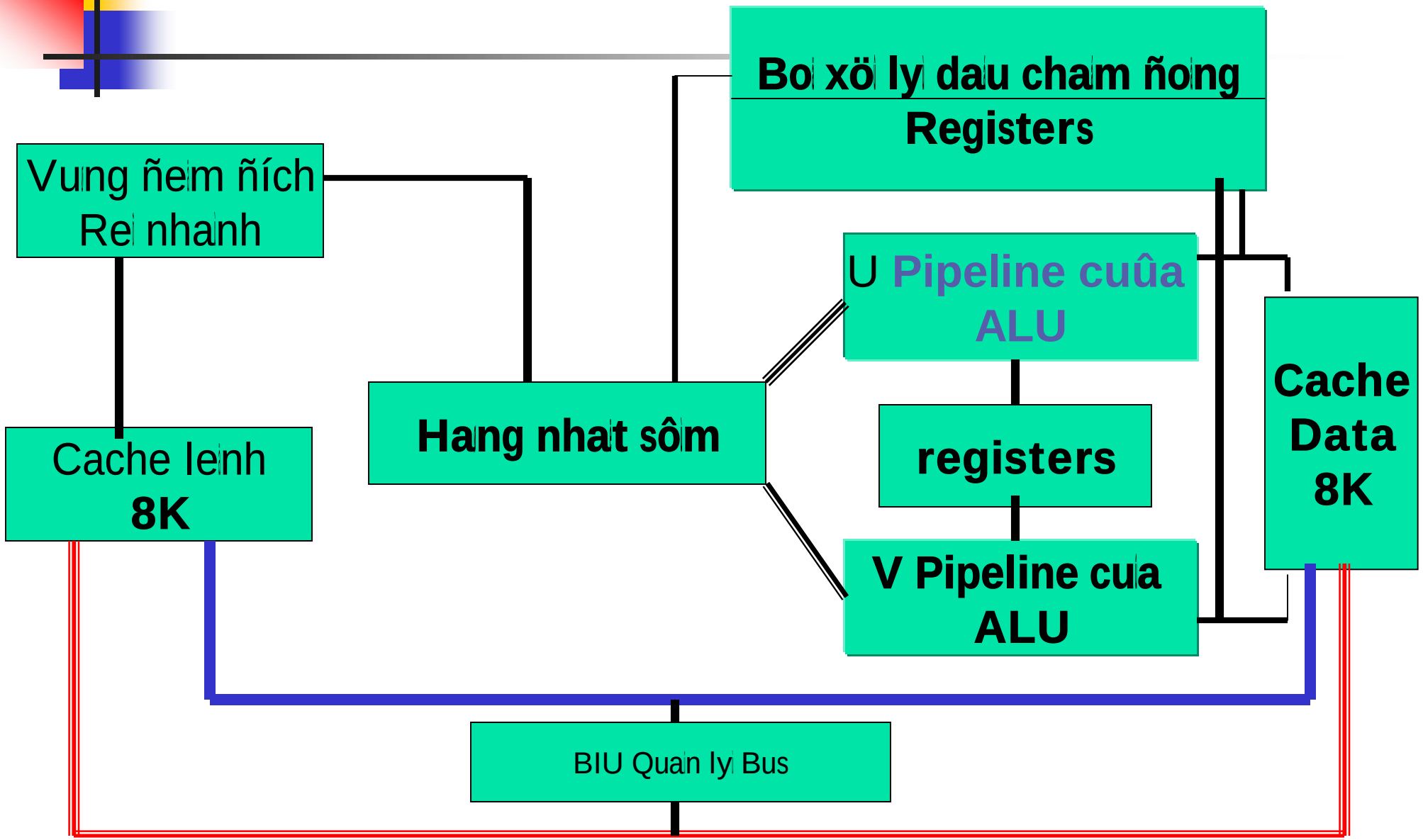
tap lenh nho ➔ thi hanh ngay khong can giai ma.

lenh lam viec theo cô che ñöông ong (pipeline).

CPU Pentium

- 3 thanh phan gop söc tang toc ño xö ly cua Pentium :
  - **Ñôn vò tính toan so nguyen supercallar**
  - **Bo nhô Cache cap 1 ô ben trong CPU.**
  - **Ñôn vò tính toan so cham ñong supercallar**

SÔ ÑOÀ KHOÁI PENTIUM



Cau hoi on tap

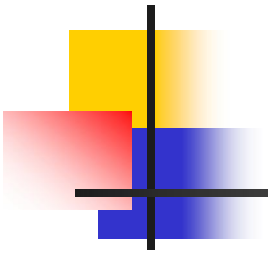
- Bus la gì? Trong cac loai Bus, Bus nao la Bus 2 chieu.
- Cho 1 o nhô co ñòa chæ vat ly la 1256H, cho biet ñòa chæ dang segment:offset vôi cac ñoan 1256H va 1240H.
- O nhô co ñòa chæ vat ly 80FD2H, ô trong ñoan nao thì no co offset = BFD2H?
- Xac ñònh ñòa chæ vat ly cua o nhô co ñòa chæ logic 0A51H:CD90H

Cau hoi on tap

- The nao la bien giôi ñoan?
- Sö khac nhau cô ban giöa bo vi xö ly 8086 va 80286?
- Thuyet minh trình tö CPU thöc hien cau lenh Mem(b) ← Not Mem(a)
- Chu ky lenh, chu ky may. Cho biet quan he giöa chu ky clock, chu ky may va chu ky lenh.
- Quan he giöa tap lenh va kien truc cua CPU

Cau hoi on tap

- Giai thích tai sao khi tang tan so xung clock, giam chu ky wait state cua bo nhô, them cache cho CPU lai lam cho he thong chay vôi hieu suat cao hôn. ?
- Trình bay chien löôc chính löu trö thong tin trong Cache?
- Tính toc ño chuyen giao dö lieu cua may tính co CPU 486DX-66MHz va may Pentium 100MHz.
- Phan biet RISC va CISC.
- Trình bay cô che ñöông ong trong thöc thi cua 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.*

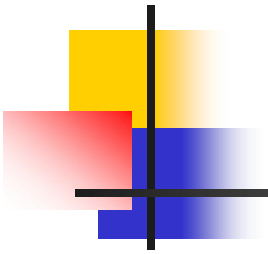
Bus ISA-16 bits :

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.*



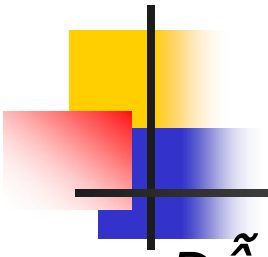
Bus PCI :

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

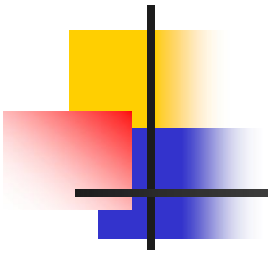
Hệ thống Bus là hệ thống xa lộ thông tin bên trong PC giúp trao đổi:

a.thông tin giữa các máy tính

b.dữ liệu giữa các thiết bị ngọai 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 chuổi '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.

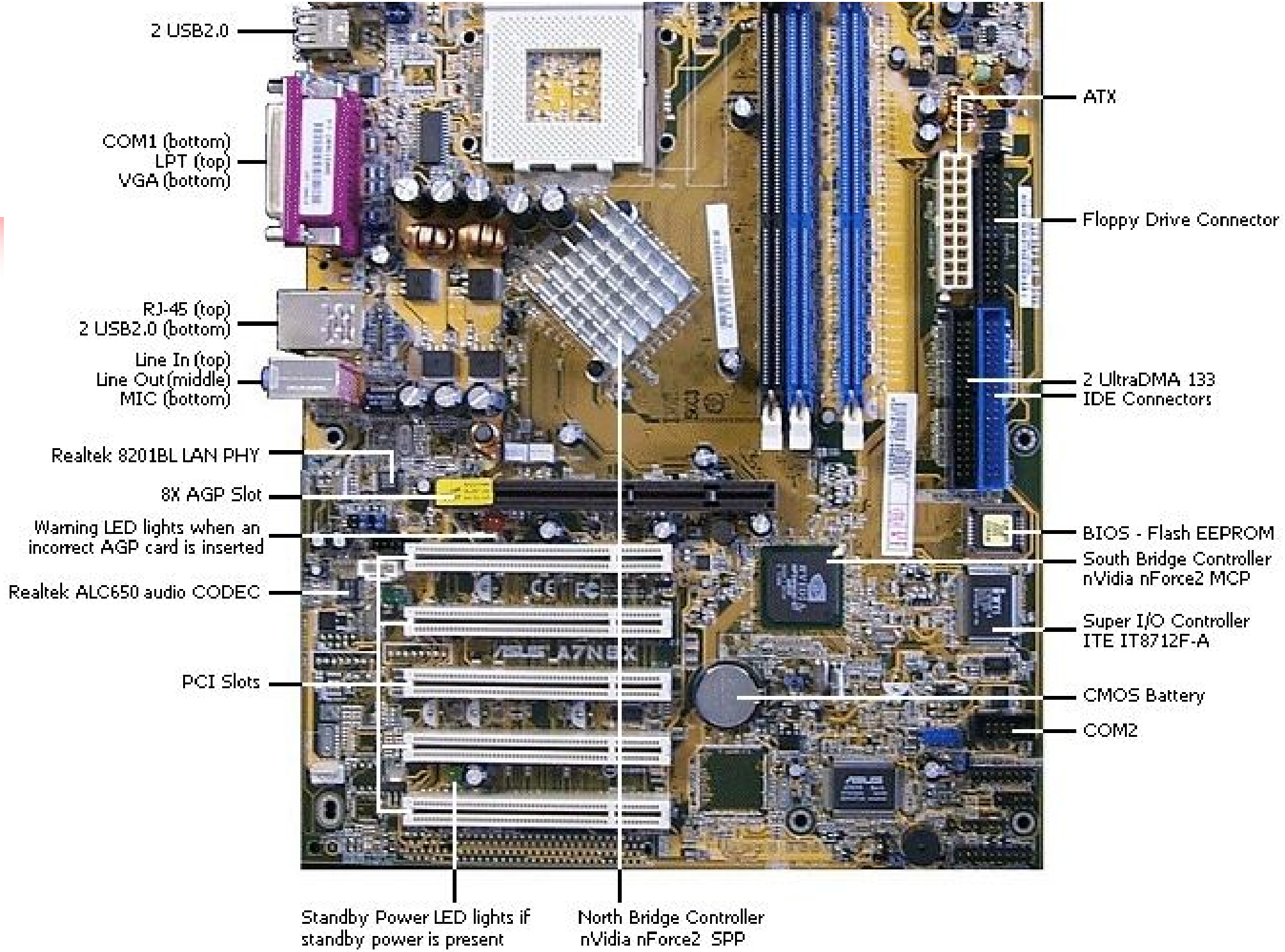
BÀI TẬP

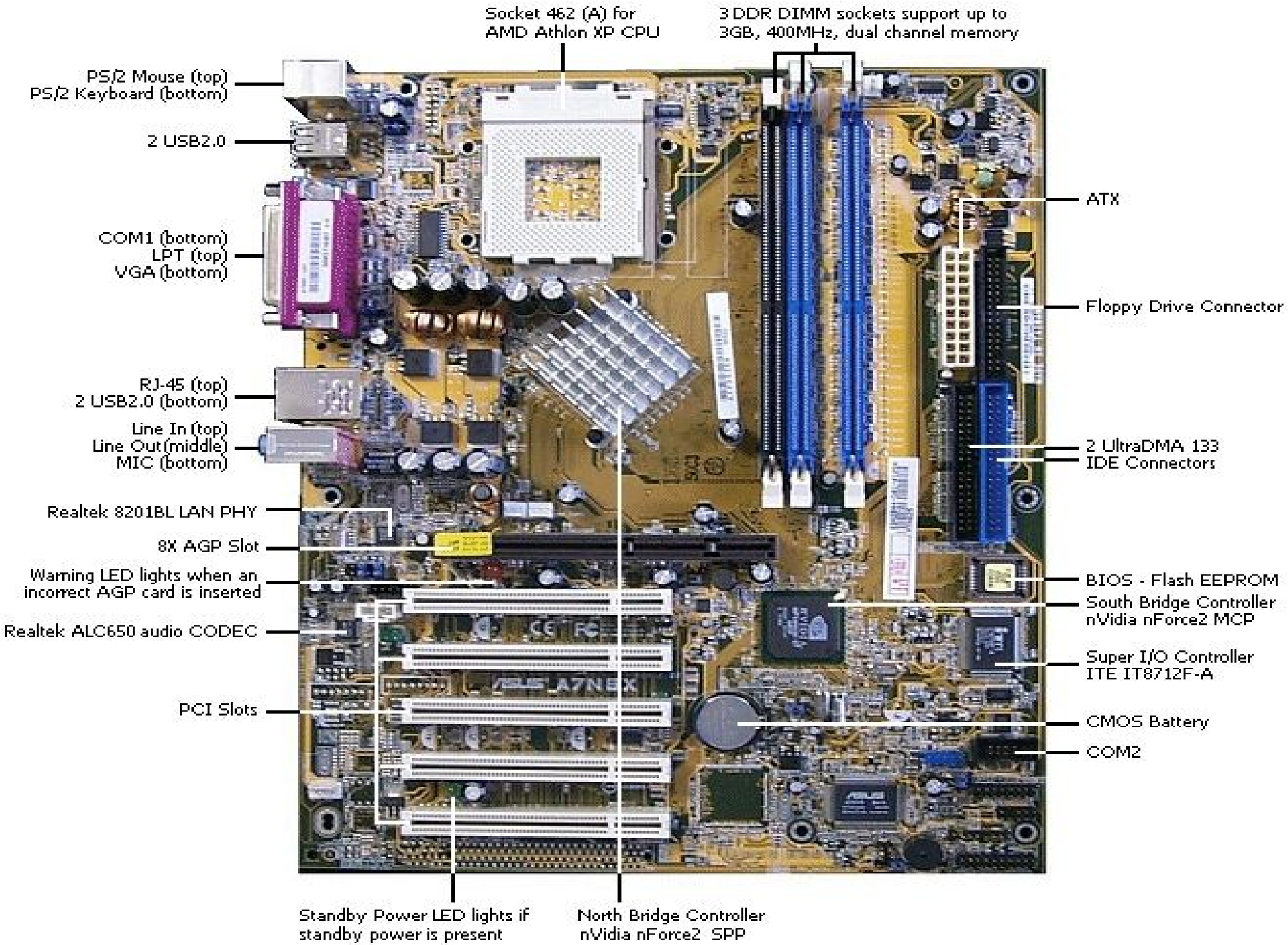
Bài 5: Giả sử thanh ghi trong MT của bạn dài 24 bits, cho biết giá trị của số dương lớn nhất mà thanh ghi này có thể chứa ở 2 hệ 2 và hệ 16?

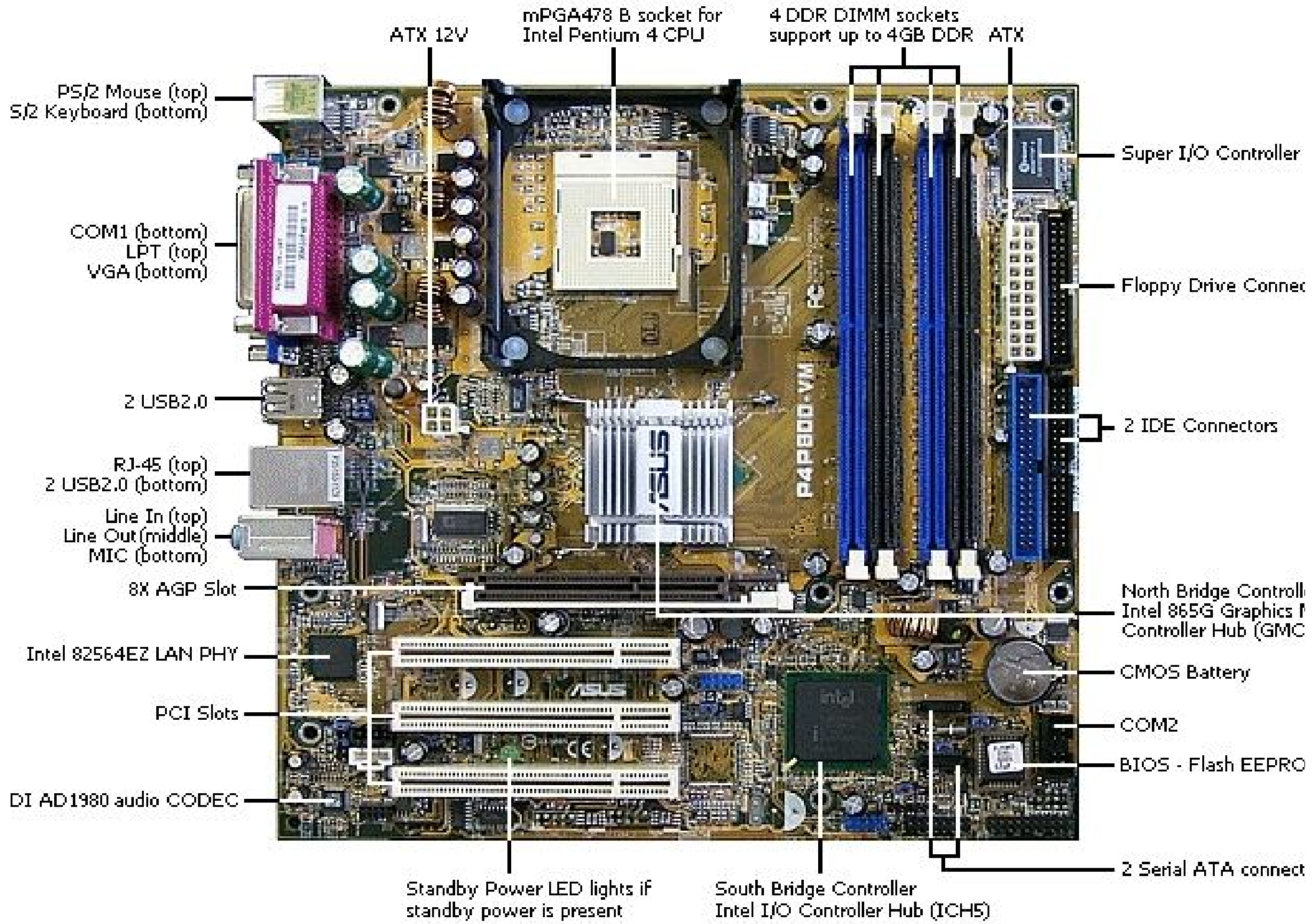
Bài 6 : Biến đổI địa chỉ sau thành địa chỉ tuyệt đối

a. *0950:0100*

b. *08F1:0200*

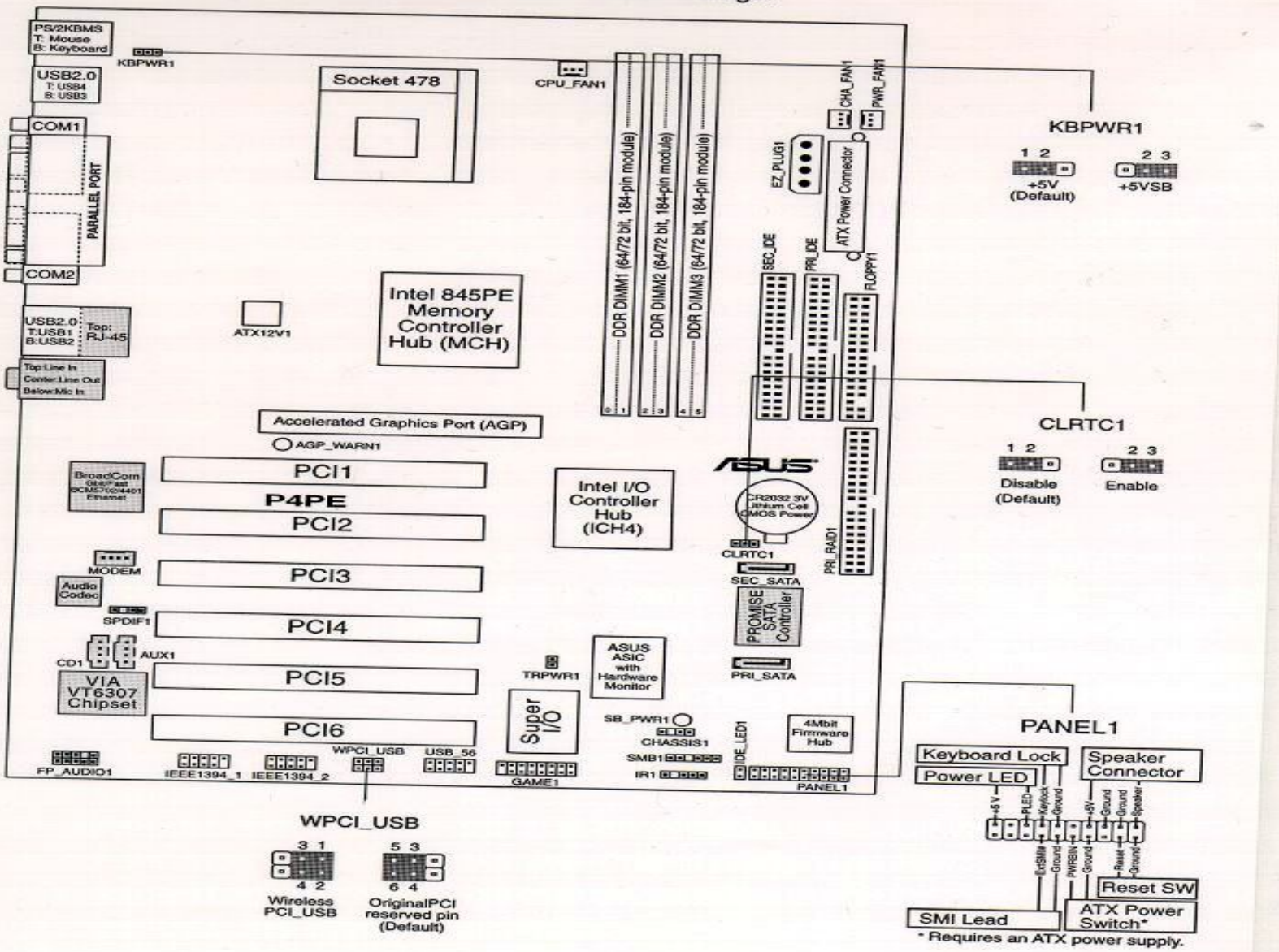

2 USB2.0
COM1 (bottom)
LPT (top)
VGA (bottom)
RJ-45 (top)
2 USB2.0 (bottom)
Line In (top)
Line Out (middle)
MIC (bottom)
Realtek 8201BL LAN PHY
8X AGP Slot
Warning LED lights when an incorrect AGP card is inserted
Realtek ALC650 audio CODEC
PCI Slots
Standby Power LED lights if standby power is present
North Bridge Controller nVidia nForce2 SPP
ATX
Floppy Drive Connector
2 UltraDMA 133 IDE Connectors
BIOS - Flash EEPROM
South Bridge Controller nVidia nForce2 MCP
Super I/O Controller ITE IT8712F-A
CMOS Battery
COM2

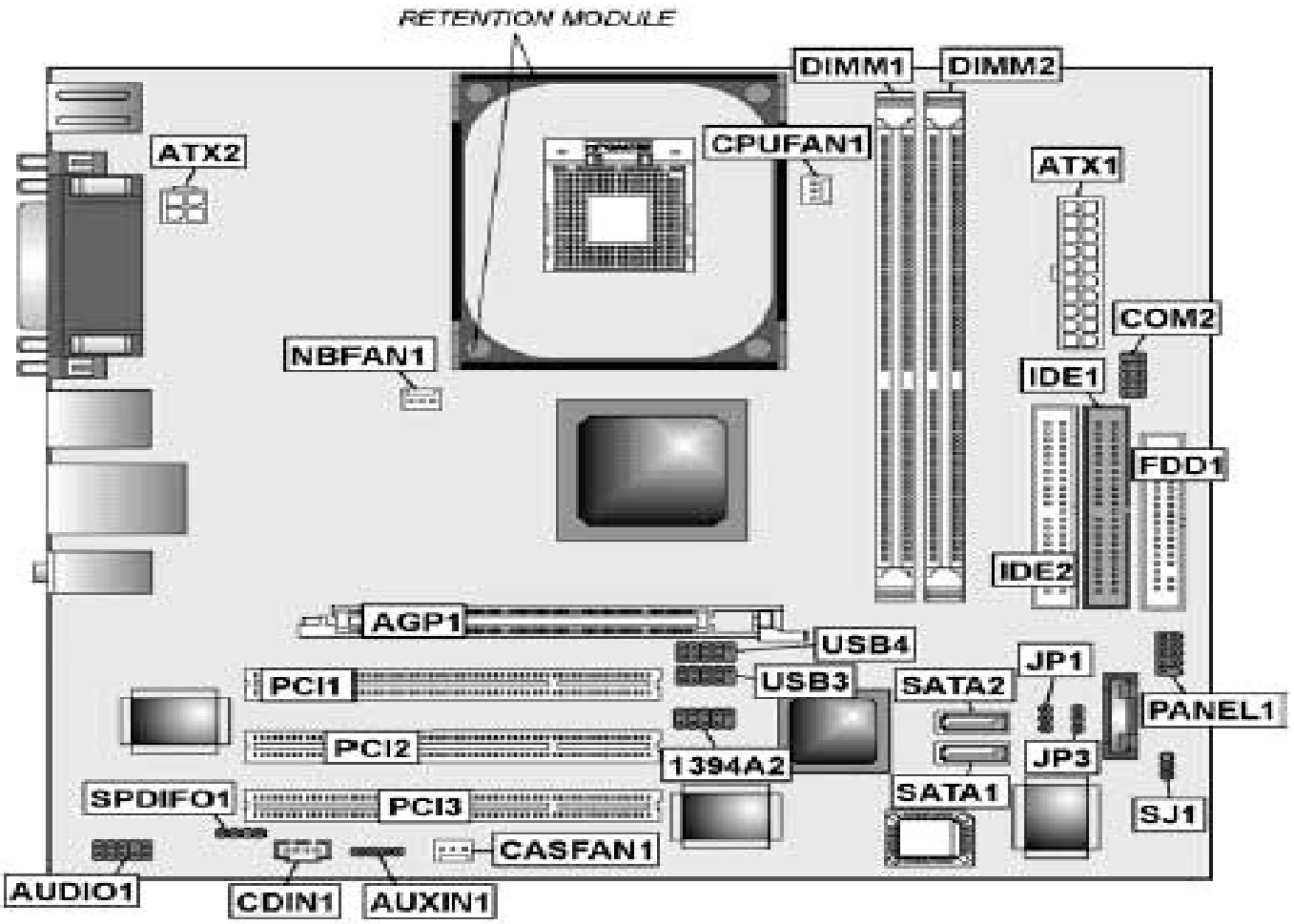

Socket 462 (A) for
AMD Athlon XP CPU
3 DDR DIMM sockets support up to
3GB, 400MHz, dual channel memory
PS/2 Mouse (top)
PS/2 Keyboard (bottom)
2 USB2.0
COM1 (bottom)
LPT (top)
VGA (bottom)
RJ-45 (top)
2 USB2.0 (bottom)
Line In (top)
Line Out (middle)
MIC (bottom)
Realtek 8201BL LAN PHY
8X AGP Slot
Warning LED lights when an
incorrect AGP card is inserted
Realtek ALC650 audio CODEC
PCI Slots
Standby Power LED lights if
standby power is present
North Bridge Controller
nVidia nForce2 SPP
ATX
Floppy Drive Connector
2 UltraDMA 133
IDE Connectors
BIOS - Flash EEPROM
South Bridge Controller
nVidia nForce2 MCP
Super I/O Controller
ITE IT8712F-A
CMOS Battery
COM2


ATX 12V
mPGA478 B socket for
Intel Pentium 4 CPU
4 DDR DIMM sockets
support up to 4GB DDR
ATX
PS/2 Mouse (top)
S/2 Keyboard (bottom)
Super I/O Controller
COM1 (bottom)
LPT (top)
VGA (bottom)
Floppy Drive Connec
2 USB2.0
2 IDE Connectors
RJ-45 (top)
2 USB2.0 (bottom)
Line In (top)
Line Out (middle)
MIC (bottom)
8X AGP Slot
North Bridge Controll
Intel 865G Graphics
Controller Hub (GMC
Intel 82564EZ LAN PHY
CMOS Battery
PCI Slots
COM2
BIOS - Flash EEPRO
DI AD1980 audio CODEC
2 Serial ATA connect
Standby Power LED lights if
standby power is present
South Bridge Controller
Intel I/O Controller Hub (ICH5)

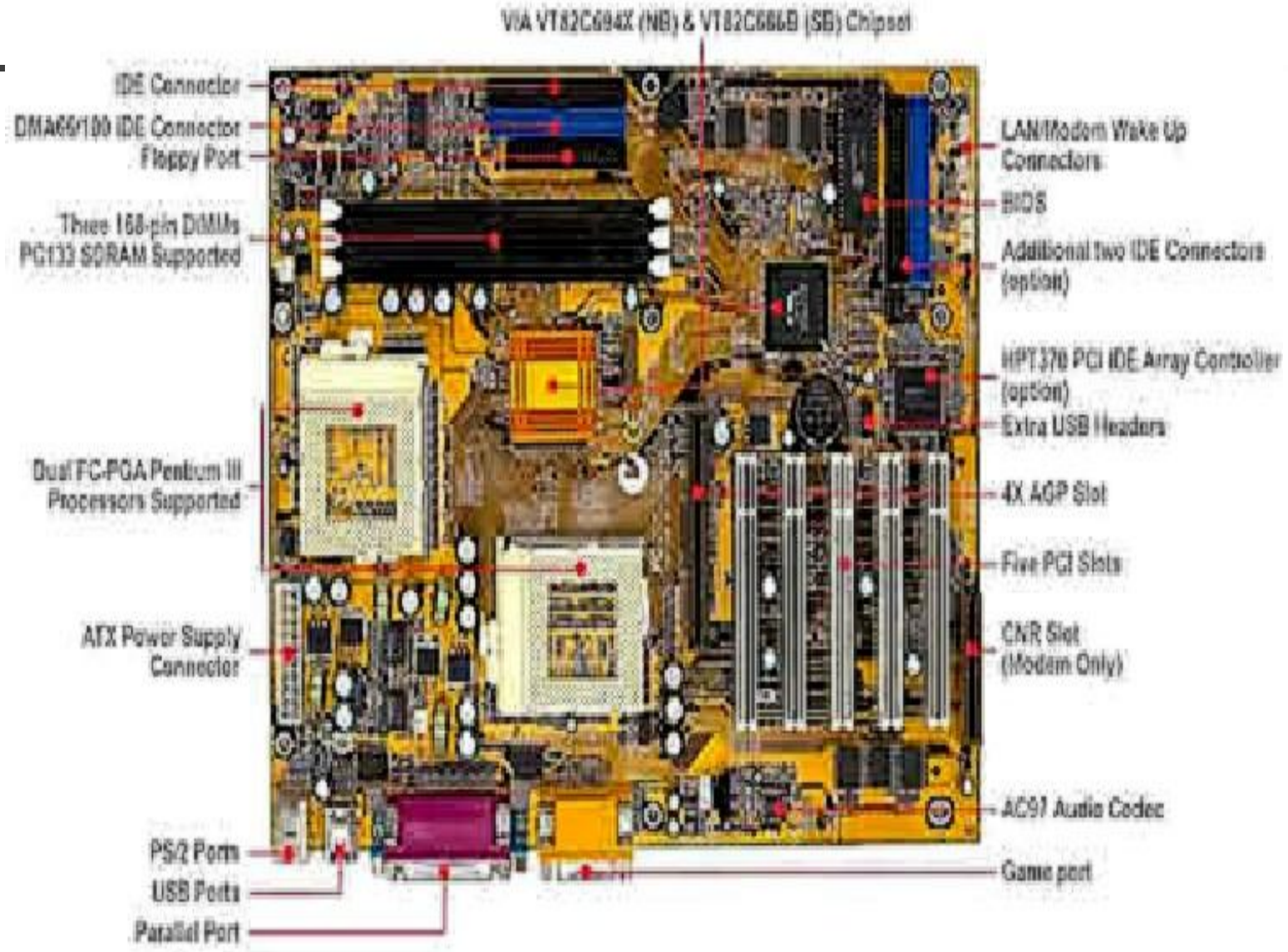
P4PE Motherboard Settings Affix this sticker to the chassis.

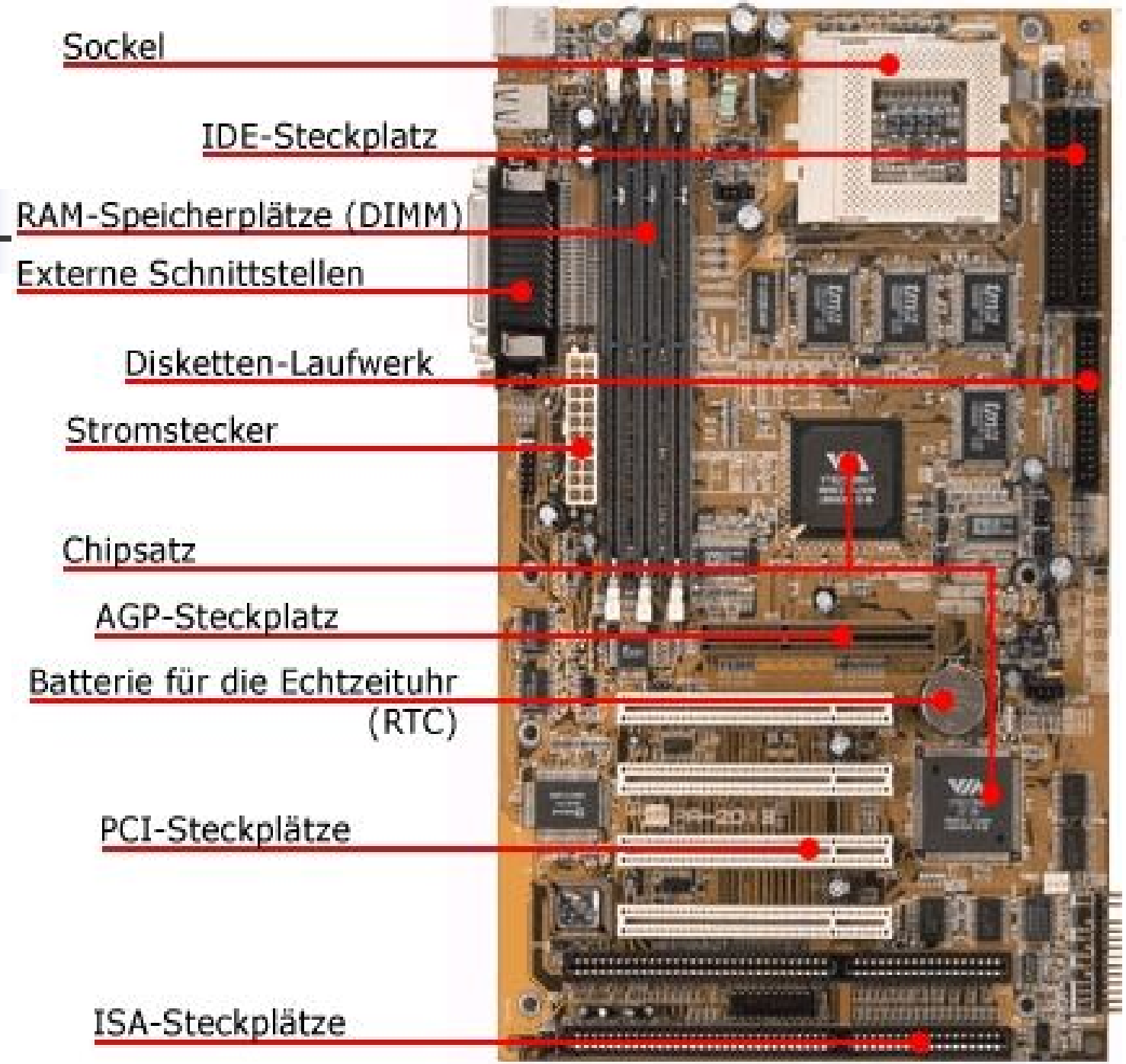
IMPORTANT! This diagram presents the general layout of your motherboard, and a summary of the switch and jumper settings. Before changing any setting, refer to the motherboard user guide for detailed descriptions and important notes on the settings.




RETENTION MODULE
DIMM1
DIMM2
ATX2
CPUFAN1
ATX1
NBFAN1
COM2
IDE1
FDD1
IDE2
AGP1
USB4
JP1
PCI1
USB3
SATA2
PANEL1
PCI2
1394A2
JP3
SPDIFO1
PCI3
SATA1
SJ1
CASFAN1
AUDIO1
CDIN1
AUXIN1

MAINBOARD


VIA VT82C694X (NB) & VT82C686B (SB) Chipset
IDE Connector
DMA66/100 IDE Connector
Floppy Port
Three 168-pin DIMMs
PC133 SDRAM Supported
Dual FC-PGA Pentium III
Processors Supported
ATX Power Supply
Connector
PS/2 Ports
USB Ports
Parallel Port
LAN/Modem Wake Up
Connectors
BIOS
Additional two IDE Connectors
(option)
HPT370 PCI IDE Array Controller
(option)
Extra USB Headers
4X AGP Slot
Five PCI Slots
CNR Slot
(Modem Only)
AC97 Audio Codec
Game port


Sockel
IDE-Steckplatz
RAM-Speicherplätze (DIMM)
Externe Schnittstellen
Disketten-Laufwerk
Stromstecker
Chipsatz
AGP-Steckplatz
Batterie für die Echtzeituhr (RTC)
PCI-Steckplätze
ISA-Steckplätze

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

- A <Assemble> :
  cho phép viết từ bàn phím các lệnh mã máy dưới dạng gợi nhớ.
- A [<địa chỉ>]
  Ex : - A 100 dịch ở địa chỉ CS:100h
  - A dịch ở địa chỉ hiện tại
  (Debug lấy địa chỉ đoạn CS)
  - A DS:2000h
  dịch ở địa chỉ DS:2000h

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
5514:0102       MOV  DL, 41
5514:0104         INT  21H
```

User gỏ vào

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.

Cách gọi : D <khoảng>

Ex : - D F000 : 0

- D ES : 100

- D 100

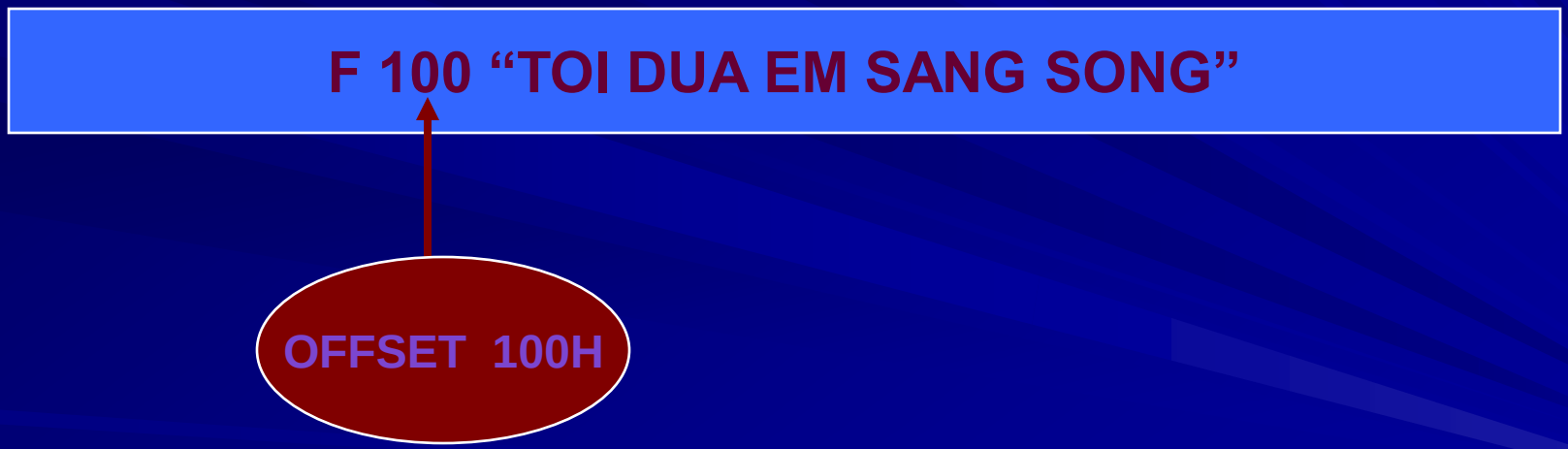
Lệnh F (Fill)

- 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
Dấu 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 lắp 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

- ñaùnh vaøo leänh D ñeå xem noäi dung vuøng nhôù cuûa 30h bytes boä nhôù töø ñòa chæ 0000:0040 ñeán 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.

Cách gọi :

- 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 dích : 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

Lệnh 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 gọi lại tập tin theo cách sau :
  C :\> Debug Love.txt
  tìm xem Debug đã nạp tập tin Love.txt vào chỗ nào trong bộ nhớ.

Lệnh W (Write)

Cú pháp : W [address]

Thường được sử dụng chung với lệnh N

Ex : tạo tập tin có tên 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).

Ơû 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.

Lệnh T (Trace)và P

- **cú pháp : - T [= <địa chỉ>][số lần]**

Mục đích : dùng để chạy 1 hay nhiều lần các lệnh trong bộ nhớ

Ex : - T = 3000:1000

Ex : - T = 3000:1000 <số lần>

Lệnh L (Load)

- **nạp tập tin hoặc nạp sector luận lý từ đĩa vào bộ nhớ.**

Cú pháp : - L <địa chỉ> [<đĩa> <sector><số>]

Dạng 1 : nếu chỉ có địa chỉ dùng để nạp tập tin. Tên tập tin phải được gán trước bằng lệnh N.

Tập tin luôn luôn được gán ở địa chỉ offset 100h

Dạng 2 : nếu có đầy đủ các thông số , dùng để đọc sector luận lý trên đĩa vào bộ nhớ.

Đĩa : = 0 ổ đĩa A, =1 ổ đĩa B, =2 ổ đĩa C ….

Lệnh H (Hex Arithmethic)

- thực hiện phép cộng và trừ hệ 16

Cú pháp : - H <trị 1> <trị 2>

Kết quả : hiện ra tổng và hiệu của trị 1 và trị 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
- Gỏ 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 nhaukhi 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.

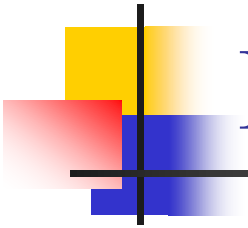
All right reserved, including the of reproduction in whole or in part, in any form.

(chú ý ký tự đầu dòng xuống dòng có mã ASCCI 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.



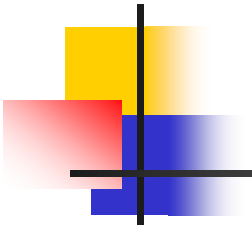
Bộ nhớ (Memory)

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

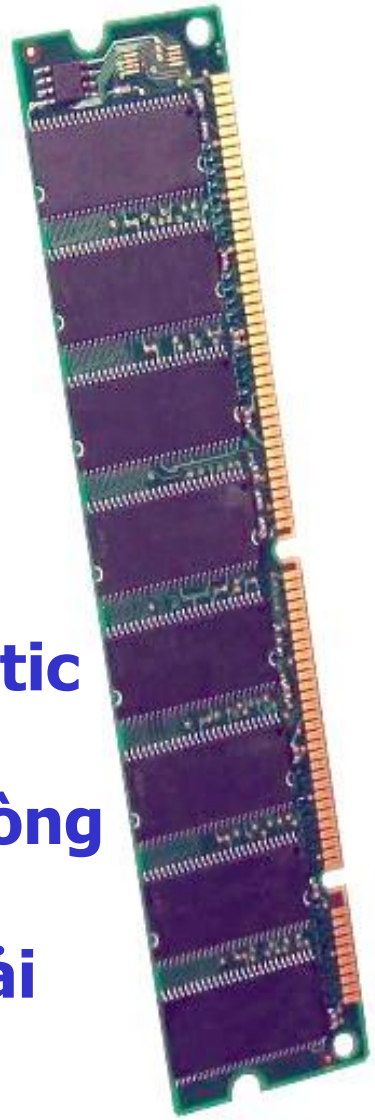
RAM

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

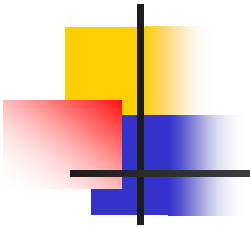

CORSAIR
CMX512-3500C2
XMS3500v1.1
024002
EXTREME MEMORY SPEED
XMS
ultra-performance DDR memory
www.corsairmemory.com
CORSAIR
CMX512-3500C2
XMS3502v1.1
024510
EXTREME MEMORY SPEED
XMS
ultra-performance DDR memory
www.corsairmemory.com

ROM

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 dẽo (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) :

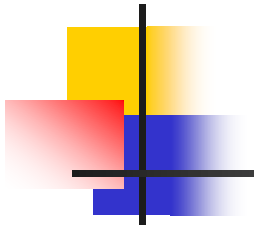
Cho phép user có thể lập trình và ghi vào ROM bằng cách đốt.

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 chẳng 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

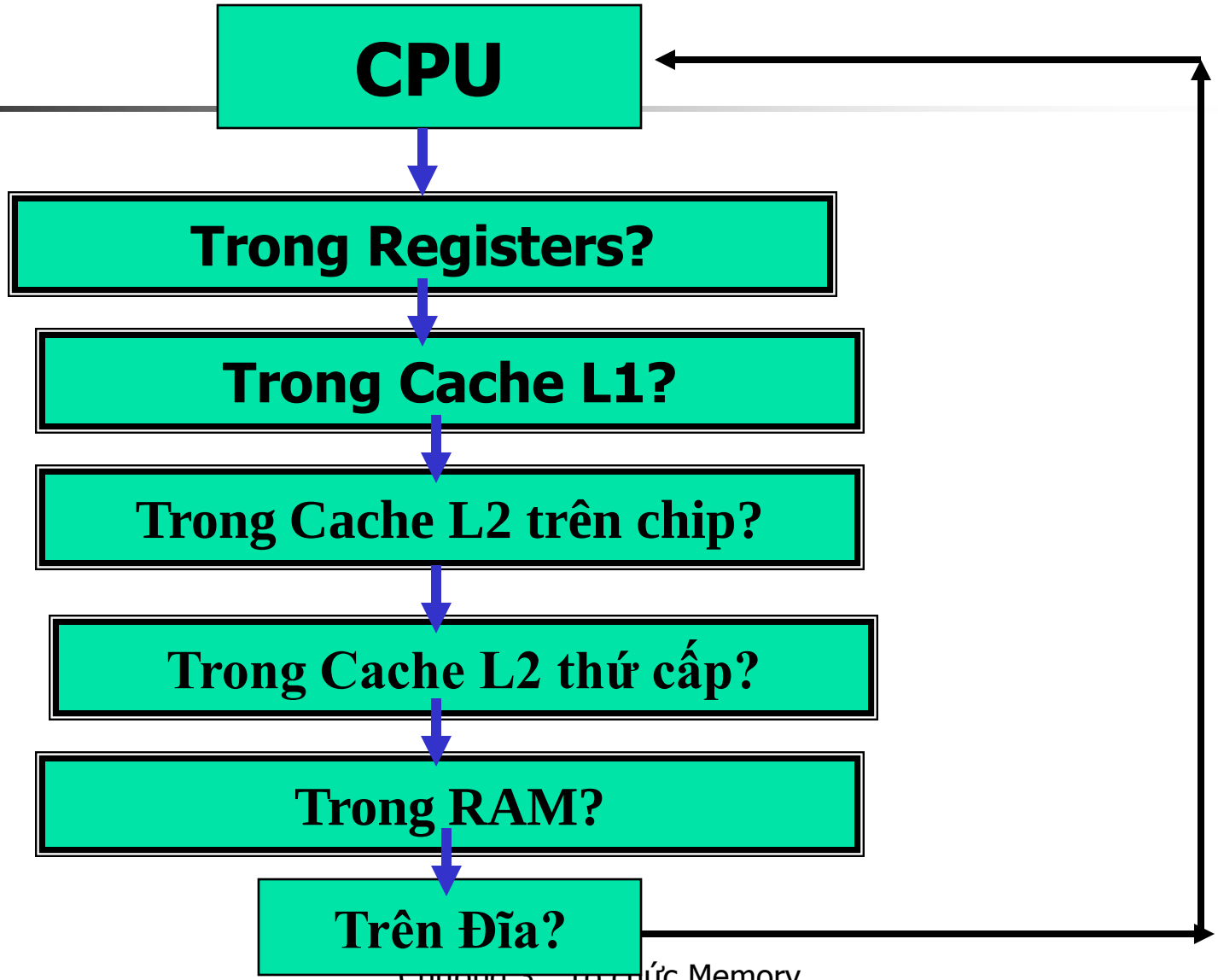
Tỉ lệ thất bại (miss ratio) (1-h)

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

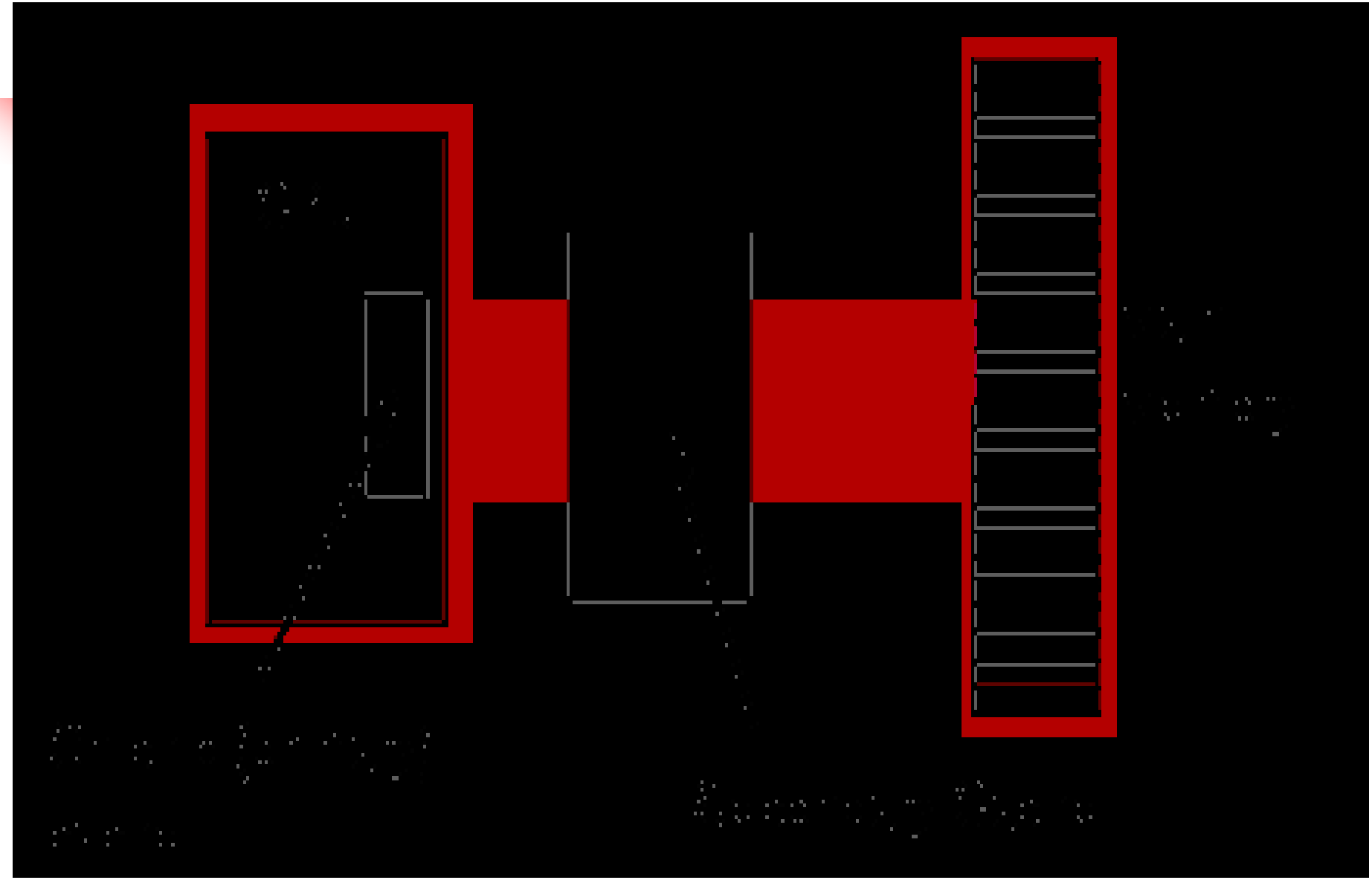
Khi h→ 1, tất cả truy xuất đều tham chiếu tới Cache, thời gian truy xuất trung bình → c.

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

Hiệu suất của Cache (cont)



A Two Level Caching System



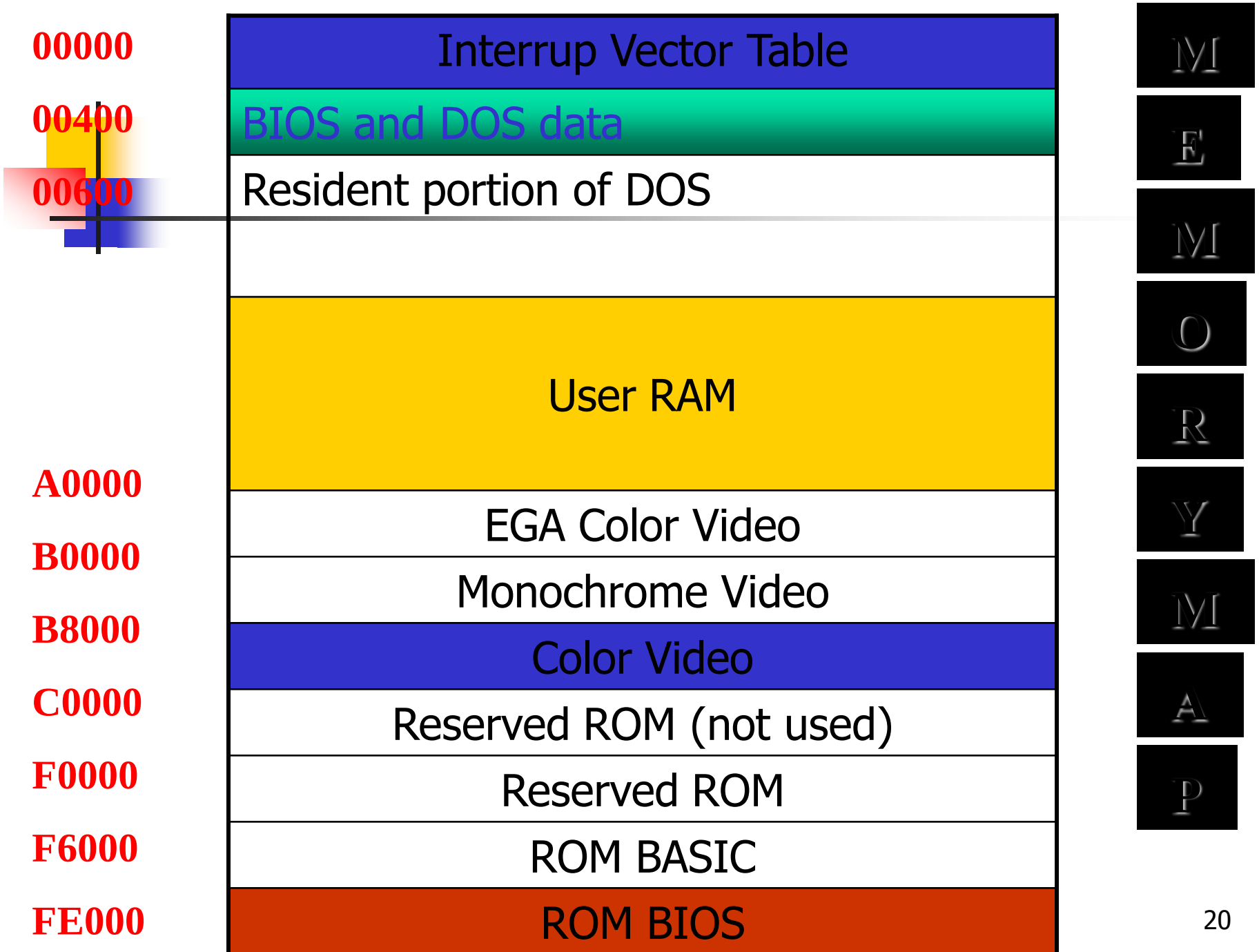
Các chiến lược trữ đệm trong Cache

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.


00000
00400
00600
A0000
B0000
B8000
C0000
F0000
F6000
FE000
Interrup Vector Table
BIOS and DOS data
Resident portion of DOS
User RAM
EGA Color Video
Monochrome Video
Color Video
Reserved ROM (not used)
Reserved ROM
ROM BASIC
ROM BIOS
M
E
M
O
R
Y
M
A
P

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 – FFFFHđược IBM dành riêng cho Rom sử dụng chứa hard disk controller, Rom Basic.

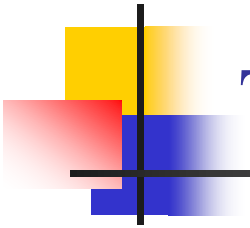
Rom BIOS : từ F000H – FFFFH vùng nhớ cao nhất của bộ nhớ chứa các chương trình con cấp thấp của Dos dùng cho việc xuất nhập và các chức năng khác..

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 chẳn lẻ bộ nhớ, thiết lập thanh ghi CS trỏ đến segment FFFFh và con trỏ lệnh IP trỏ 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 đưa ra thanh ghi dữ liệu.**

✓ **CPU đọc nội dung của thanh ghi dữ liệu.**

Mạch giải mã địa chỉ ô nhớ

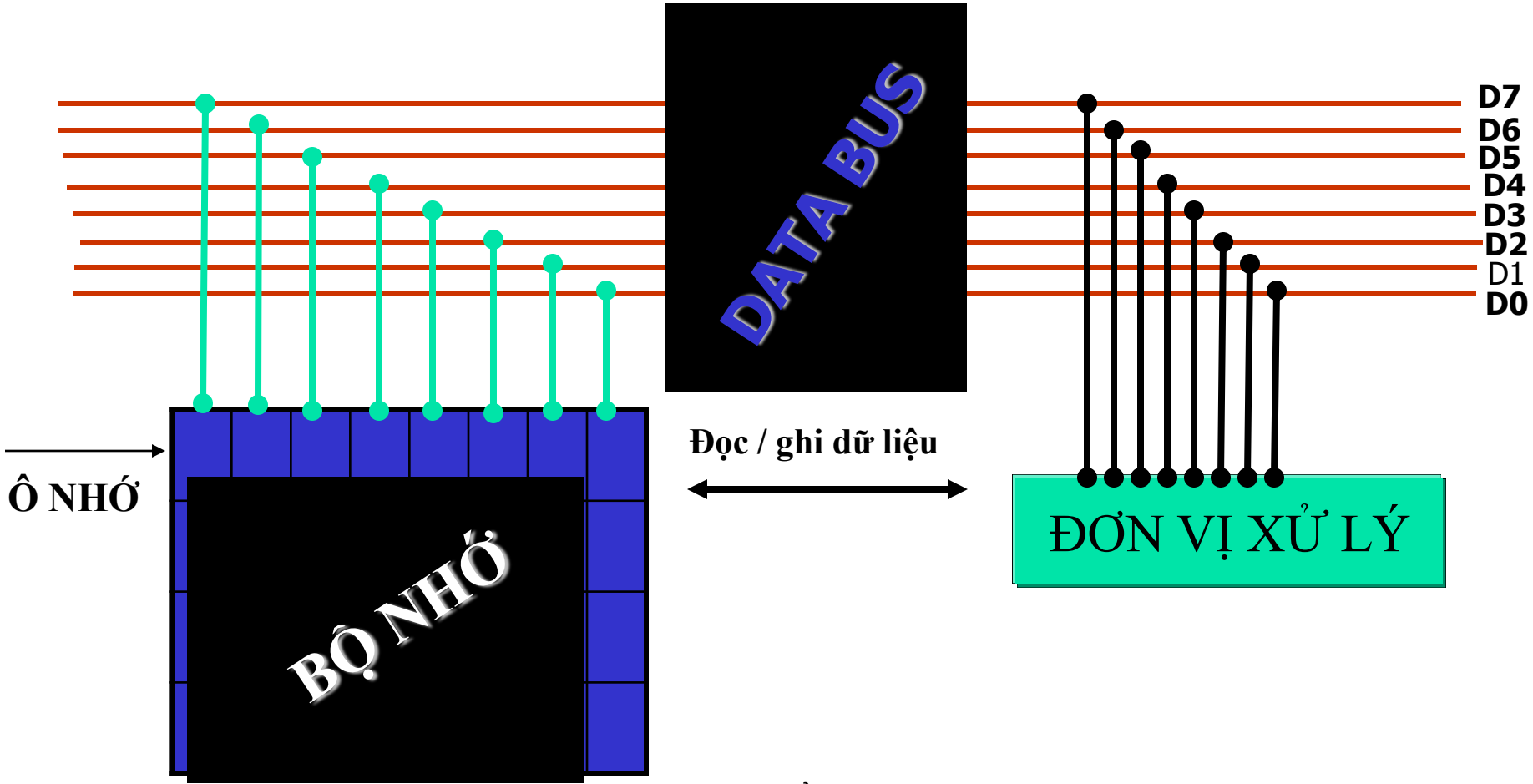
Mạch điện có nhiệm vụ xác định đúng ô nhớ cần truy xuất đang có địa chỉ lưu trong thanh ghi địa chỉ.

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)

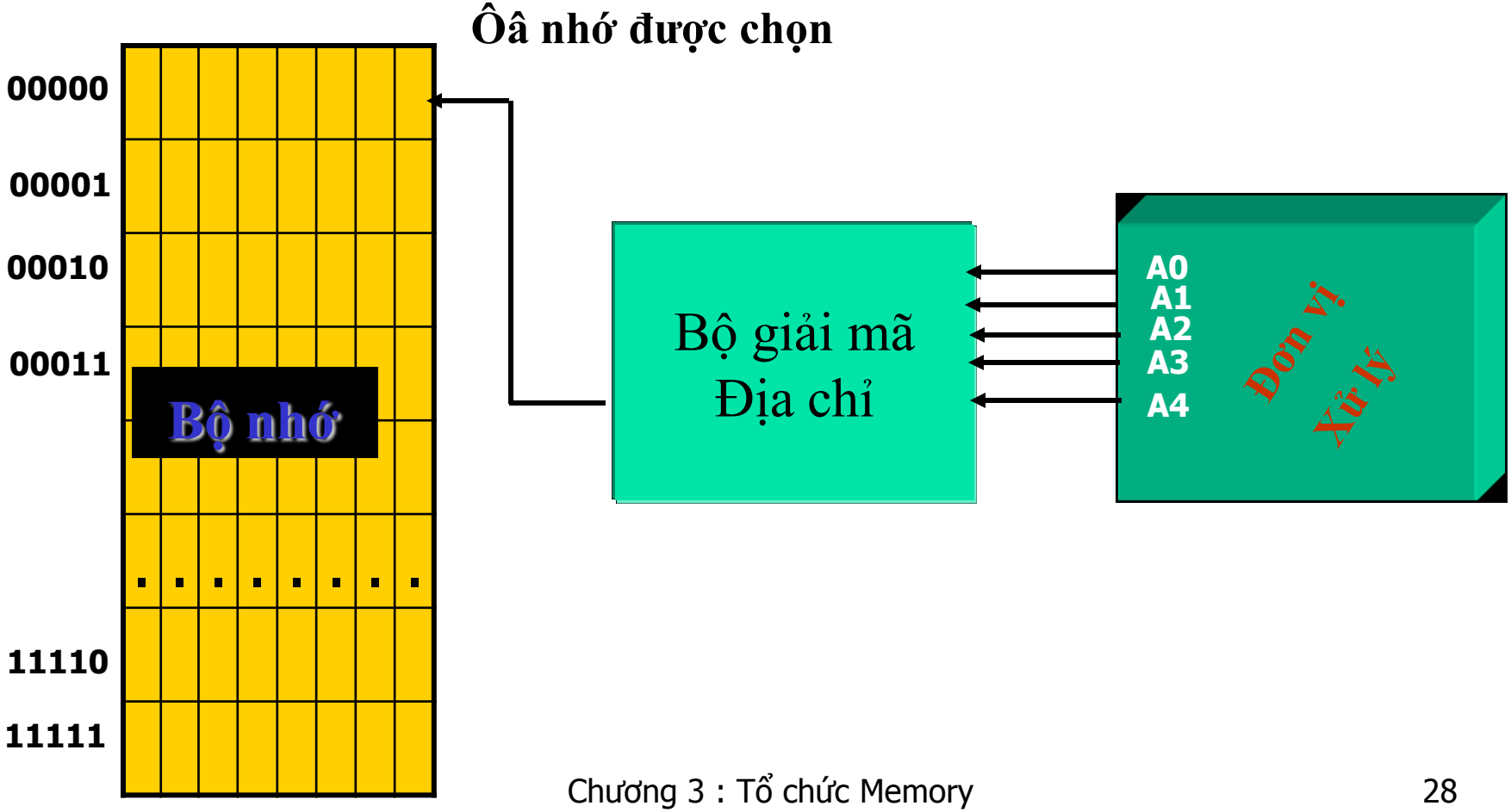
Truy xuất bộ nhớ (cont)

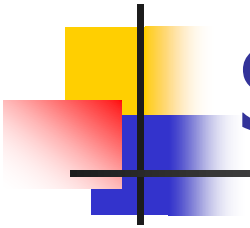


Trình tự tác vụ ghi ô nhớ

- **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

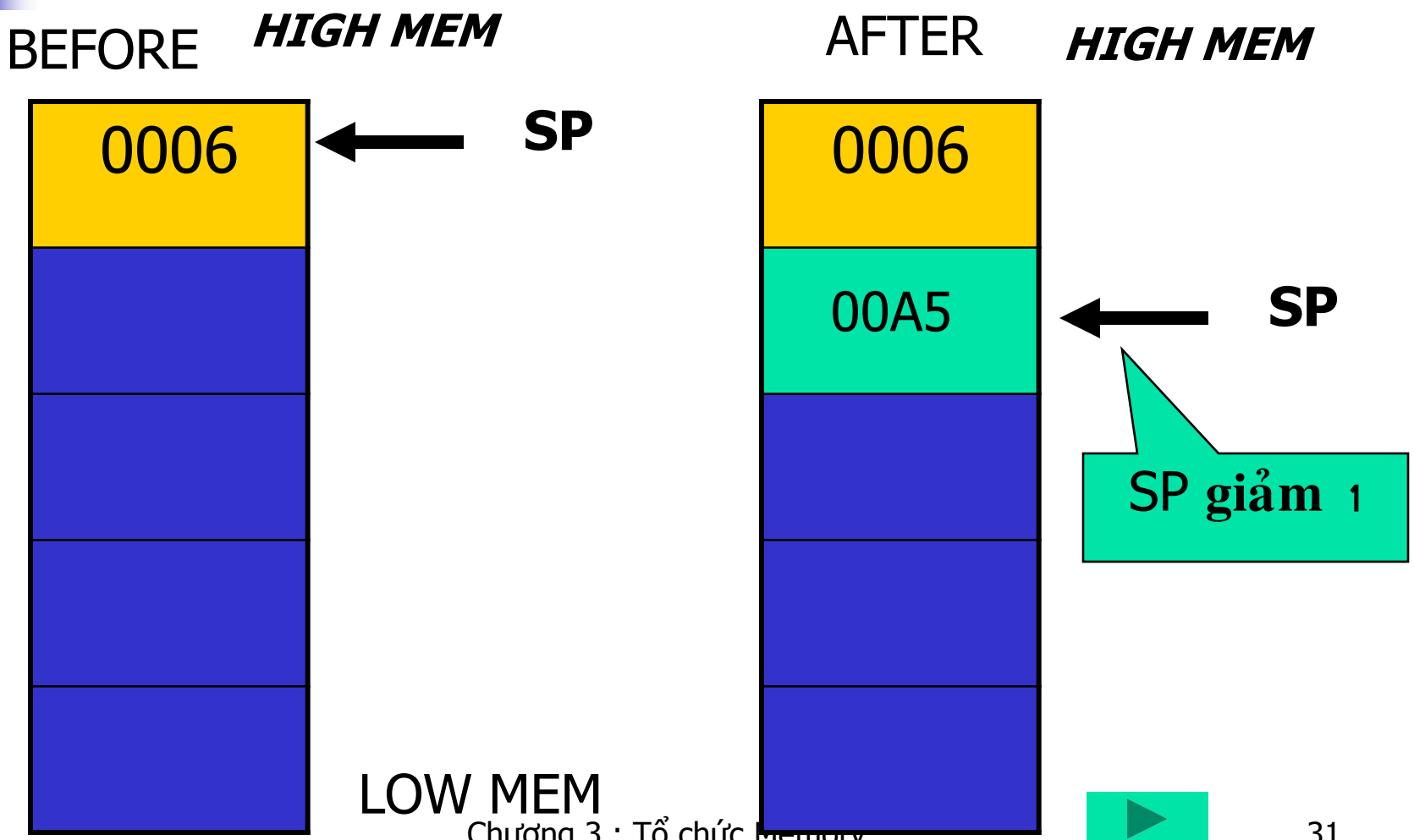
phần tử dữ liệu cuối mới được thêm vào này lại là phần tử sẽ được lấy ra (popped trước tiên).

➔ 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

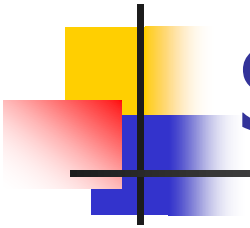

BEFORE
HIGH MEM
0006
SP
LOW MEM
AFTER
HIGH MEM
0006
00A5
SP
SP giảm 1

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 trở 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?
- Nêu 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 4Kx8.

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

Input /Output Devices

Mục tiêu

- Nắm nguyên lý cấu tạo và đặc điểm của thiết bị I/O.
- Nhiệm vụ và yêu cầu của thiết bị I/O.
- Cách giao tiếp giữa CPU và thiết bị I/O.
- Hiểu các bước trong quá trình ngắt quãng.
- Nắm được cơ chế DMA

Nội dung

- Nguyên lý xuất nhập trong máy tính
- Cách CPU giao tiếp với thiết bị I/O.
- Ngắt quãng
- DMA
- Các thiết bị I/O :
  Hard Disk,Floopy Disk, Printer, Keyboard,Mouse

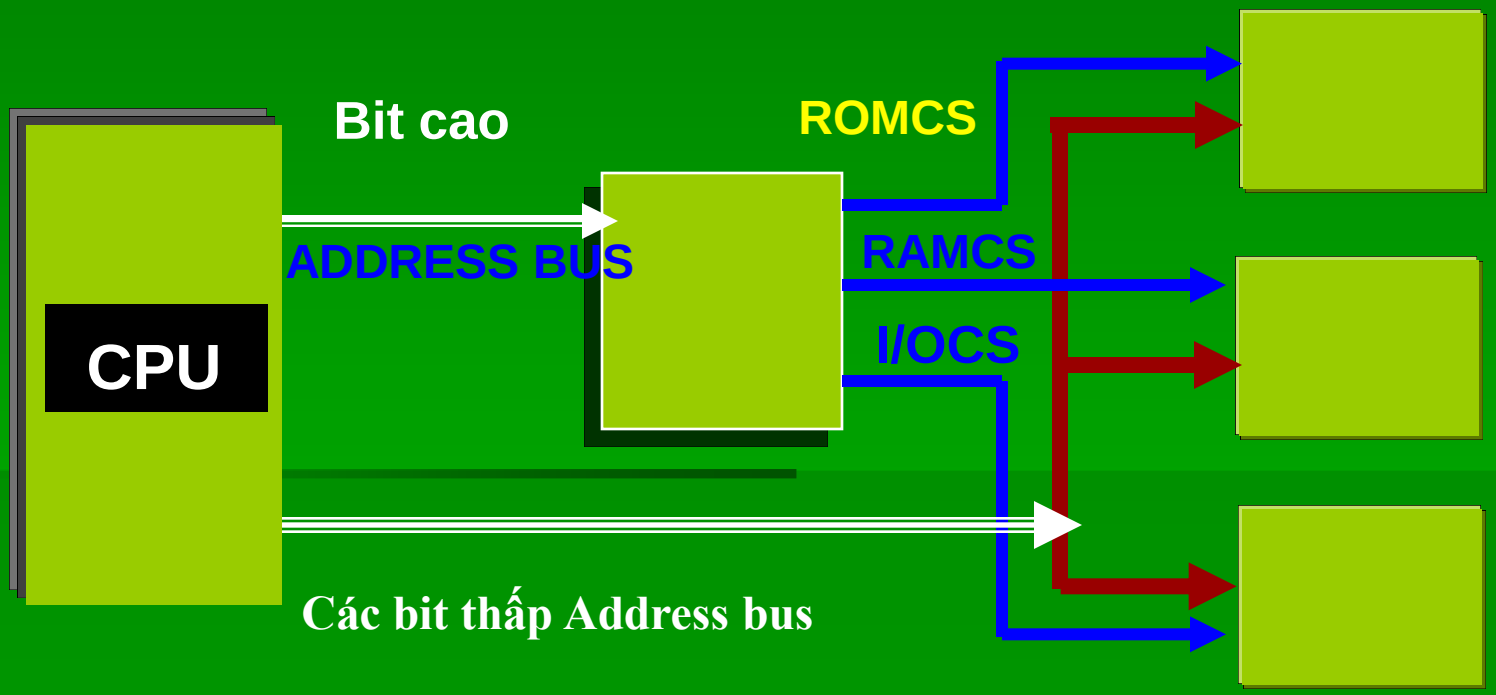
Thiết bị I/O :

- **Thiết bị I/O là 1 thiết bị có khả năng cung cấp dữ liệu khi CPU yêu cầu trong tác vụ đọc và có khả năng ghi dữ liệu vào khi CPU thực thi 1 tác vụ ghi.**

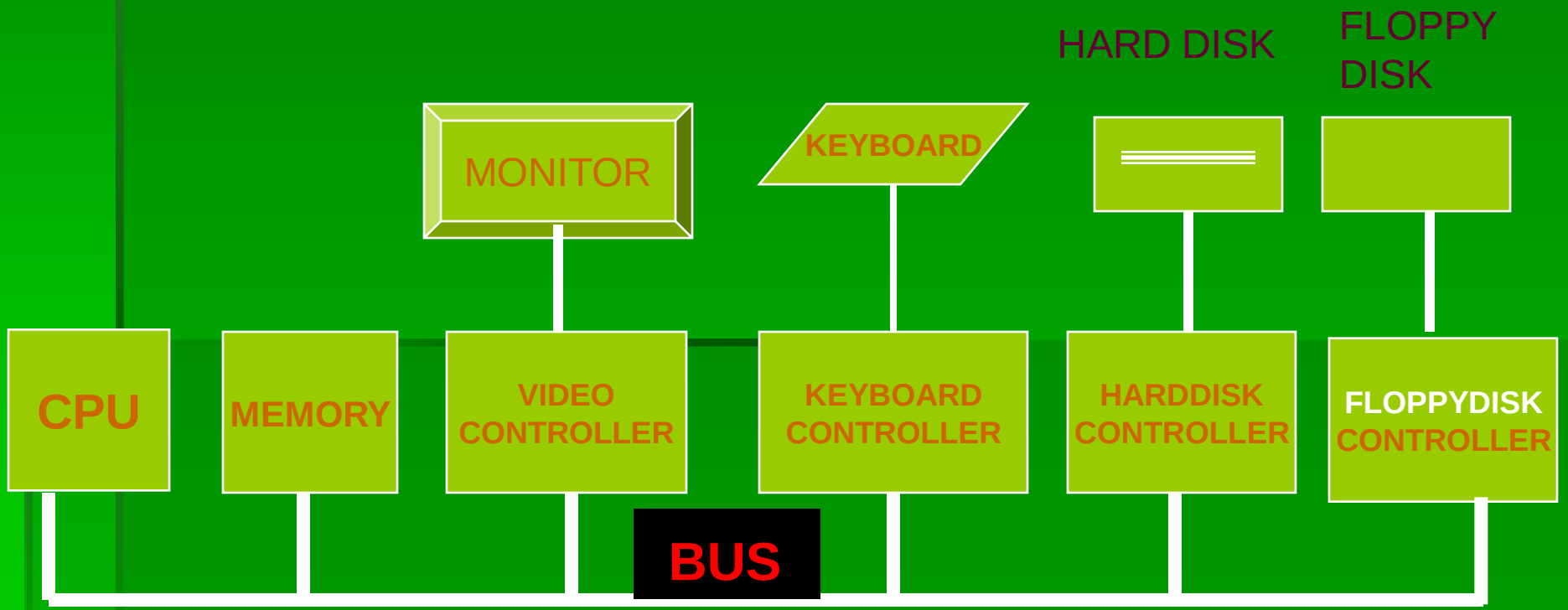
Làm sao CPU nhận biết một I/O

- Mỗi I/O có 1 địa chỉ riêng gọi là cổng (port). Khi CPU truy xuất I/O , CPU xuất ra 1 địa chỉ.
- Một số bit cao của địa chỉ đi vào bộ giải mã, trên đường ra của bộ bộ giải mã sẽ có tín hiệu Chip select tương ứng với I/O mà CPU muốn truy xuất.
- Các địa chỉ thấp còn lại sẽ đi đến mọi I/O nhưng chỉ có I/O nào có đường Chip Select tích cực mới được truy xuất.

Sơ đồ giải mã địa chỉ



Minh họa


HARD DISK
FLOPPY DISK
MONITOR
KEYBOARD
CPU
MEMORY
VIDEO CONTROLLER
KEYBOARD CONTROLLER
HARDDISK CONTROLLER
FLOPPYDISK CONTROLLER
BUS

CPU liên lạc với thiết bị I/O

- Thiết bị ngoại vi liên lạc với CPU thông qua các cổng I/O .

Các thiết bị I/O có tốc độ làm việc chậm hơn tốc độ của CPU rất nhiều ➔ để khắc phục nhược điểm này người ta dùng vùng nhớ đệm.

Sự truyền thông tin giữa thiết bị I/O và CPU được thực hiện theo 2 bước :

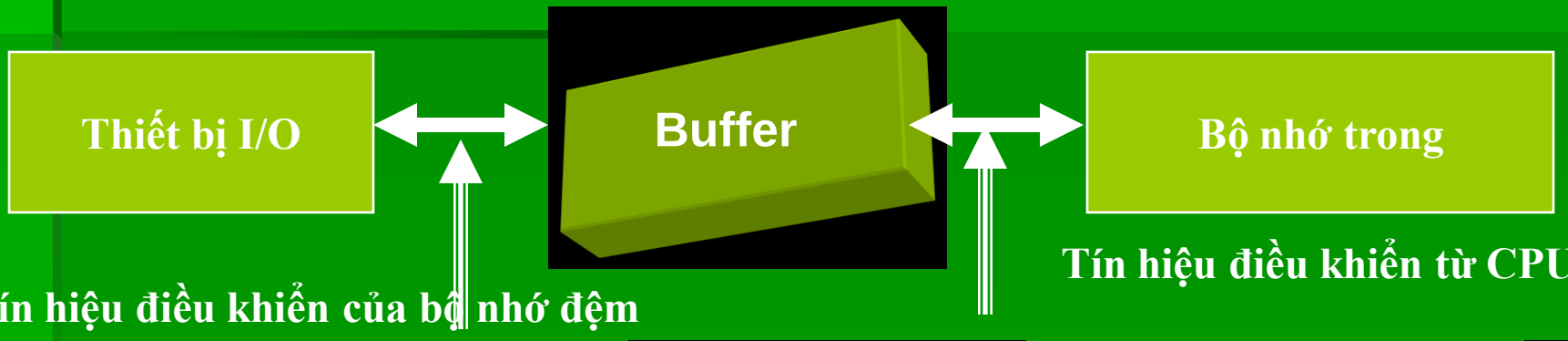
CPU liên lạc với thiết bị I/O

Bước 1 : truyền thông tin giữa bộ nhớ trong và bộ nhớ đệm.

Bước 2 : truyền thông tin giữa bộ nhớ đệm và thiết bị I/O.

CPU liên lạc với thiết bị I/O

- Có thể tổ chức để 1 CPU làm việc đồng thời với nhiều thiết bị ngoại vi bằng cách phân chia thời gian.



Ngắt quãng (Interrupt)

- Ngắt (Interrupt) là gì ? :
  Ngắt là sự làm ngừng chương trình đang chạy.

- Một interrupt xuất hiện khi 1 chương trình đang thực thi bị ngưng.

- Interrupt được tạo ra bởi nhiều lý do khác nhau

Ngắt quãng (Interrupt)

- **Do user lập trình có lệnh INT <number> yêu cầu phục vụ ngắt quãng (như xuất nhập chẳng hạn).**

- **Do hệ thống gây ra vì 1 lý do nào đó không mong muốn (như lỗi của phép chia 0, phép tính bị tràn số…)**

- Do thiết bị I/O gây ra : máy in, bàn phím, ổ đĩa ...

Software Interrupt

- Ngắt mềm :

Do thi hành lệnh INT trong chương trình.

Xãy ra khi cần 1 chương trình con trong hệ điều hành và thường là chương trình con xuất nhập.

Cú pháp gọi 1 ngắt mềm trong chương trình :

INT number

Software Interrupt

- Một số ngắt mềm thông dụng :

INT 10h : Video services

INT 16h : Keyboard services

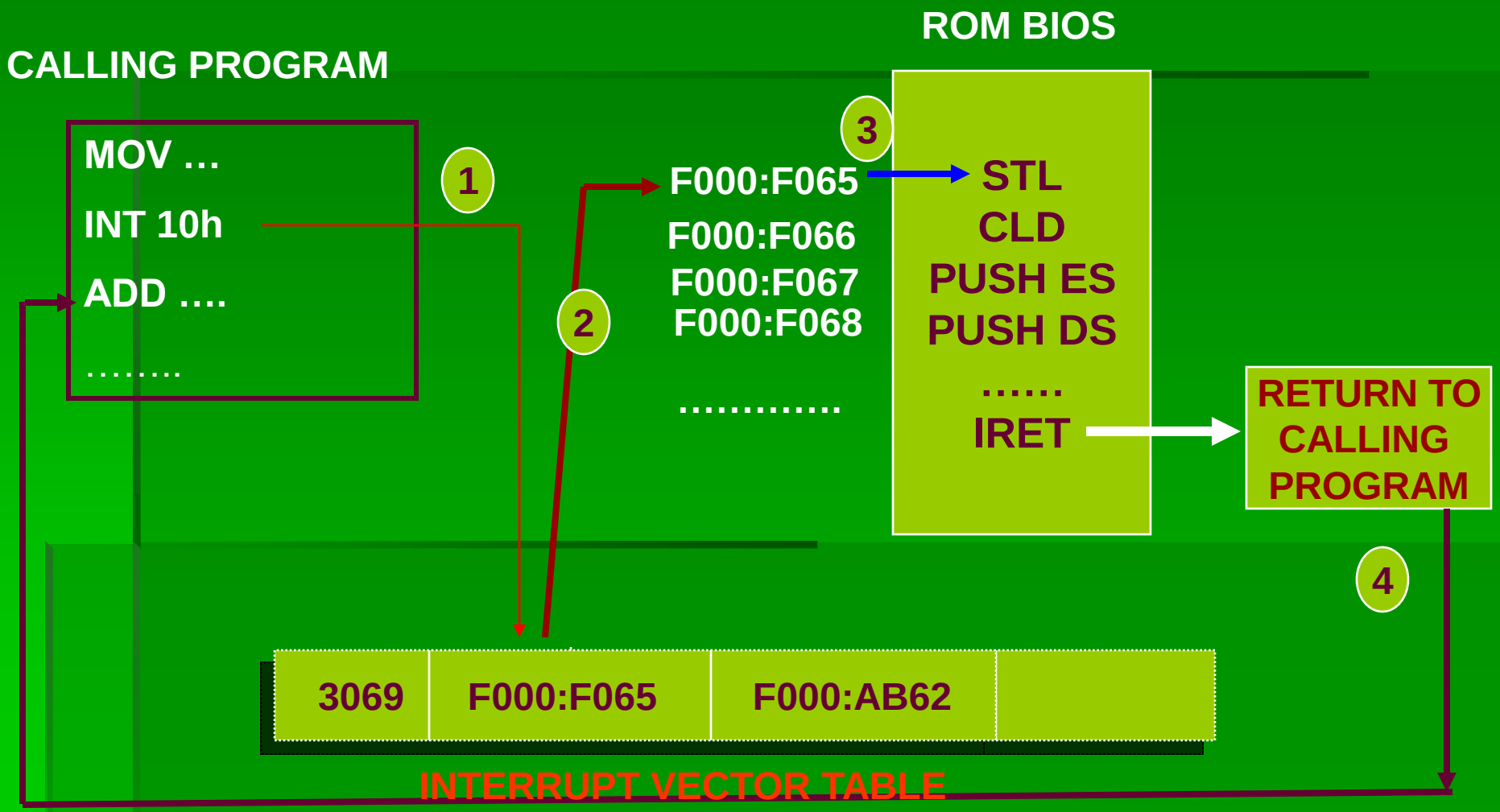
INT 17h : Printer services

INT 1AH : Time of Day

INT 1CH : User Time Interrupt

INT 21H : Dos Service

Thí dụ minh họa gọi ngắt mềm



Giải thích

1. Con số theo sau INT báo cho CPU biết phải định vị mục nào trong bảng vector ngắt quãng.

2. CPU nhảy đến địa chỉ lưu trong bảng vector ngắt quãng (F000:F065).

3. Một chương trình con (điều khiển ngắt) tại F000:F065 bắt đầu được thi hành và hoàn tất khi gặp lệnh IRET.

4. lệnh IRET giúp CT quay trở lại ngay sau lệnh gọi ngắt và tiếp tục thi hành lệnh này.

Hệ thống ngắt IBM PC/XT

Ngắt	địa chỉ logic	địa chỉ VL	công dụng
0	00E3:3072	03EA2	lỗl phép chia
1	0600:08ED	068ED	chạy từng lệnh
2	F000:E2C3	FE2C3	ngắt không che NMI
3	0600:08E6	068E6	điểm dừng
4	0700:0147	07147	tràn khi làm việc vớl số có dấu
5	F000:FF54	FFF54	In màn hình (BIOS)
6	,7 dự trữ		
8 đến F các ngắt của chip 8259			
10	F000:F065	FF065	Vào ra cho Video (BIOS)
11	F000:F84D	FF84D	kiểm tra cấu hình tbị (BIOS)
12	F000: F841	FF841	kiểm tra kích thước bộ nhớ (BIOS)

Hệ thống ngắt IBM PC/XT

Ngắt	địa chỉ logic	địa chỉ VL	công dụng
13	F000:EC59	FEC59	Vào/ra đĩa (BIOS)
14	F000:E739	FE739	vào/ra RS 232 (BIOS)
15	F000:F859	FF859	vào/ra cassette (BIOS)
16	F000:E82E	FE82E	Vào/ra bàn phím (BIOS)
0700:0147		07147	tràn khi làm việc vớI số có dấu
2	F000:FF54	FFF54	In màn hình (BIOS)
3 ,7	dự trữ		
8 đến F các ngắt của chip 8259			
10	F000:F065	FF065	Vào ra cho Video (BIOS)
11	F000:F84D	FF84D	kiểm tra cấu hình tbị (BIOS)
12	F000: F841	FF841	kiểm tra kích thước bộ nhớ (BIOS)

…….

Hệ thống ngắt IBM PC/XT

Ngắt	địa chỉ logic	địa chỉ VL	công dụng
1A	F000:FE6E	FFE6E	thờl gian hệ thống (BIOS)
1B	F000:0140	00840	điều khiển Ctrl+Break
.....			
20	PSP:0000	---------	Kết thúc chương trình DOS
21	Có thể đặt lại	----------	gọl chức năng DOS

F1 – FF không sử dụng

Tùy version DOS, dạng MT một số địa chỉ logic có thể khác nhau

Một số ví dụ minh họa

Ex : Xem bảng vector ngắt quãng trên MT

a. Sử dụng DEBUG để hiển thị nội dung của các ô nhớ 0000:002Fh

b. Tìm CS:IP của lỗi phép chia, NMI và INT 8

Đối vớl lỗi phép chia INT 0, CS:IP được đặt ở địa chỉ 0,1,2,3.

```
D:\>DEBUG
-D 0000:0000 002F
0000:0000  68 10 A7 00 8B 01 70 00-16 00 96 03 8B 01 70 00
0000:0010  8B 01 70 00 B9 06 0C 02-40 07 0C 02 FF 03 0C 02
0000:0020  46 07 0C 02 0A 04 0C 02-3A 00 96 03 54 00 96 03
-
```

Có thể dữ liệu trên máy PC của bạn khác vớI dữ liệu trên vì còn phụ thuộc vào version của DOS, ngày tháng của BIOS, việc sử dụng bộ nhớ kép (shadow memory).

Đối với ngắt lỗi phép chia (INT 0), CS:IP được đặt ở địa chỉ 0,1,2,3.

➔ CS = 00A7 IP = 1068

Còn INT 8 thì sao ?

Ex : minh họa INT 0 được gọi và thi hành

```
MOV AL,92
SUB CL,CL
DIV CL
INT 0
INT 3
```

Đoạn CT trên sẽ xuất thông báo

Divide Error

Interrupt nội

gắt nội : ngắt tự thân do CPU sinh ra còn được gọi là ngắt không che NMI.

Ngắt nội xãy ra khi CPU ở 1 trạng thái không mong muốn như lỗi phép chia 0 (DIV 0), phép chia bị tràn, điện áp nguồn bị giảm thấp ...

Hardware Interrupt

- Ngắt cứng :

được tạo ra khi thiết bị ngoại vi cần đến CPU.

Ngắt cứng được phát sinh bởi chip 8259 Interrupt Controler, phát tín hiệu cho CPU tạm đình chỉ sự thi hành của CT hiện hành và xử lý ngắt.

Đăïc trưng của ngắt cứng là tín hiệu yêu cầu ngắt quãng INTR

Ngắt bàn phím là 1 điển hình ngắt cứng.

Khi cần thiết chương trình có thể cấm ngắt cứng.

Hardware Interrupt

Đăïc trưng của ngắt cứng là tín hiệu yêu cầu ngắt quãng INTR.

Ngắt bàn phím là 1 điển hình ngắt cứng.

Khi cần thiết chương trình có thể cấm ngắt cứng bằng lệnh CLI (Clear Interrupt Flag).

Bảng vector Interrupt

- **Một vùng nhớ dài 1024 bytes đặt ở đầu bộ nhớ chính (0h – 400h) , chứa 256 phần tử, mỗi phần tử là 1 bộ 4 bytes đánh số từ 0h-FFh và được gọi là các vector ngắt , tạo thành bảng vector ngắt .**

Mỗi vector ngắt chứa địa chỉ của 1 chương trình phục vụ ngắt đặt trong bộ nhớ.

Các chương trình phục vụ này liên lạc trực tiếp với các thiết bị I/O thông qua 1 số thanh ghi gọi là cổng (port) vào/ra.

Bảng Interrupt vector (cont)

- Khi 1 ngắt được yêu cầu, CPU không cần biết địa chỉ của chương trình con phục vụ ngắt này mà chỉ quan tâm đến **số hiệu i** của ngắt và số này chỉ đến phần tử thứ i của bảng interrupt vector .

Ex : Khi ta gỏ vào 1 phím, 1 tín hiệu sẽ tạm thời ngắt ngang công việc của CPU. CPU sẽ tìm đến vector ngắt số 9 (của bàn phím). Vector này ở địa chỉ 0:24h.

- CPU sẽ lấy ra địa chỉ của thủ tục chuyên phục vụ bàn phím (có sẵn trong ROM BIOS).
- Thực hiện thủ tục này xong
- Quay trở lại chỗ bị ngắt để tiếp tục thực hiện công việc dở dang

Những chỉ thị ngắt này lấy địa chỉ CTC ở đâu ?

Ex : INT 21h

Để tính địa chỉ của CTC phục vụ ngắt ta :

21h * 4 = 84h ➔ cần dùng 2 word (4 bytes)cho mỗi vector ngắt hay địa chỉ CTC.

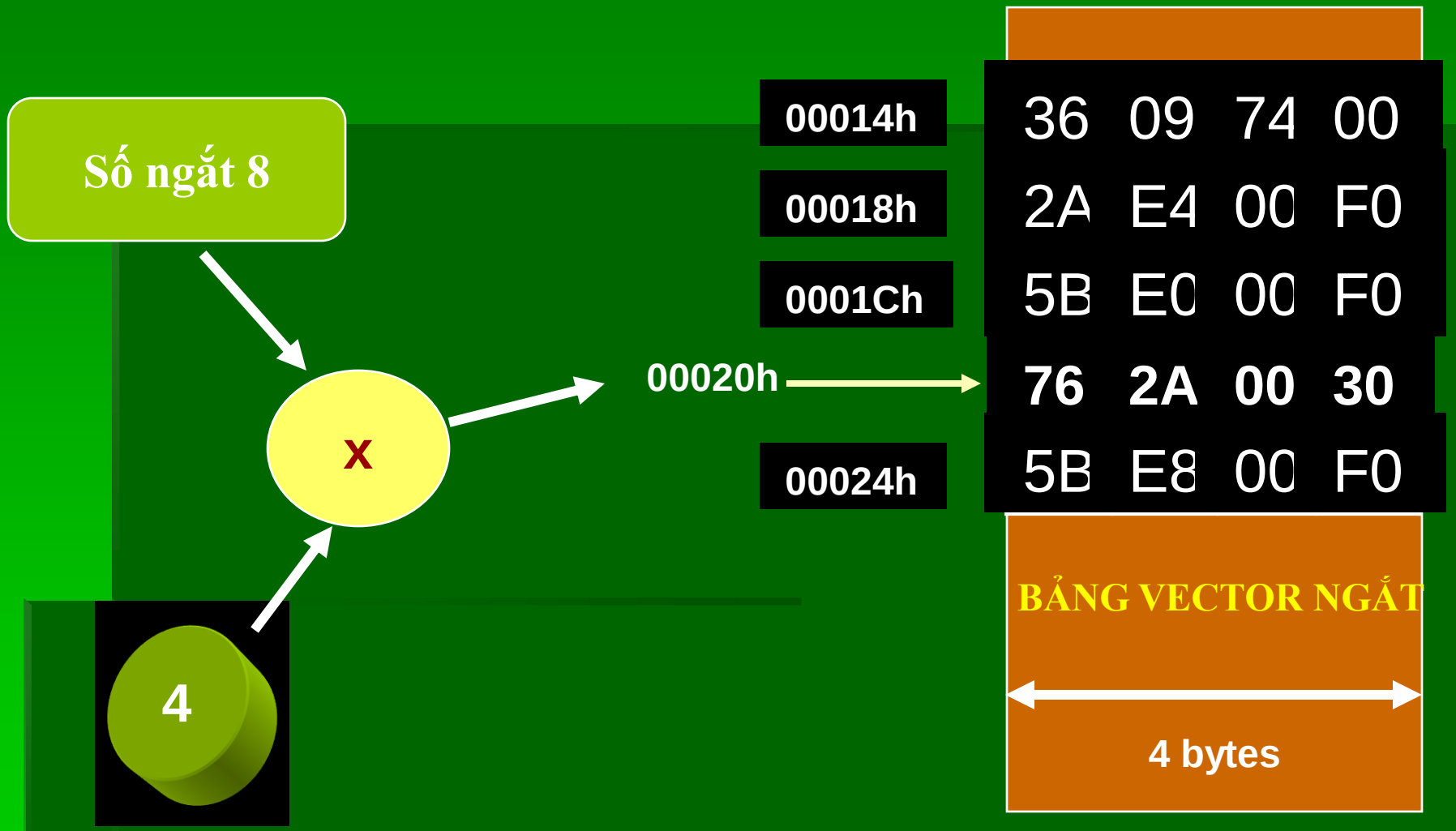
Địa chỉ 00A7:107C

CS **IP**

Các vector ngắt trỏ đến các thủ tục có sẵn trong ROM BIOS

Vector ngắt	Địa chỉ	Chức năng
5	14H	**In màn hình ra máy in.**
8	20H	**Đo thời gian**
9	24H	**Mã scan từ bàn phím**
10	40H	**Video display**
11	……	
12	……	
13	……	

Thí dụ minh họa Interrupt



Các loại cổng vào ra

- **Cổng nối tiếp (serial port) : IBM PC cung cấp 2 cổng nối tiếp : COM1 hay AUX và COM2**

  2 cổng này ở địa chỉ 400h và 402h trong vùng dữ liệu BIOS.

Cổng nối tiếp dùng cho modem điện thoại, một máy in nối tiếp hay nối trực tiếp với 1 máy tính khác.

Các loại cổng vào ra

- **Cổng song song (parallel port) : IBM PC cho phép sử dụng 3 cổng song song : PRN hay LPT1, LPT2 và LPT3.**

Địa chỉ của các cổng này lưu trong bộ nhớ tại 408, 40A, 40C.

Tên cổng	địa chỉ	Nơi chứa địa chỉ
COM1	3F8H	400
COM2	2F8H	402
PRN	3BCH	408
LPT2	378H	40A

DMA (Direct Memory Access)

DMA là gì ?

- Kỹ thuật cho phép I/O device hay Bus điều khiển việc truyền dữ liệu vào/ra MT mà không thông qua CPU.

  ➔ Nhờ thế CPU vẫn điều khiển các quá trình xử lý khác trong quá trình nhập xuất dữ liệu.

Thiết bị đầu cuối

- **Keyboard : Thiết bị nhập đơn giản.**

tập hợp các công tắc bố trí thành 1 ma trận.

Tín hiệu ngõ ra của ma trận công tắc này được đưa vào mạch tạo mã bàn phím.

Mỗi tổ hợp phím xác định được ấn xuống mạch sẽ tạo ra 1 con số nhận diện cho phím đó, sau đó con số này sẽ gửi cho CPU.

Scan code của Bàn phím

- Chip 8048 xử lý điều khiển bàn phím :

Theo dõi có phím nào được ấn không thì báo cho CPU thông qua ngắt 09h.

Nếu có phím nào bị ấn quá ½s , 8048 sẽ lặp lại phím này sau những khoảng thời gian nhất định (typematic)

Mỗi lần ấn 1 phím, các mạch điện tử của bàn phím sẽ tạo ra 1 mã dài 1 byte gọi là mã scan , đặc trưng cho vị trí trên bàn phím của phím tác động, giá trị nằm trong khoảng 1 ..83

Scan code của Bàn phím

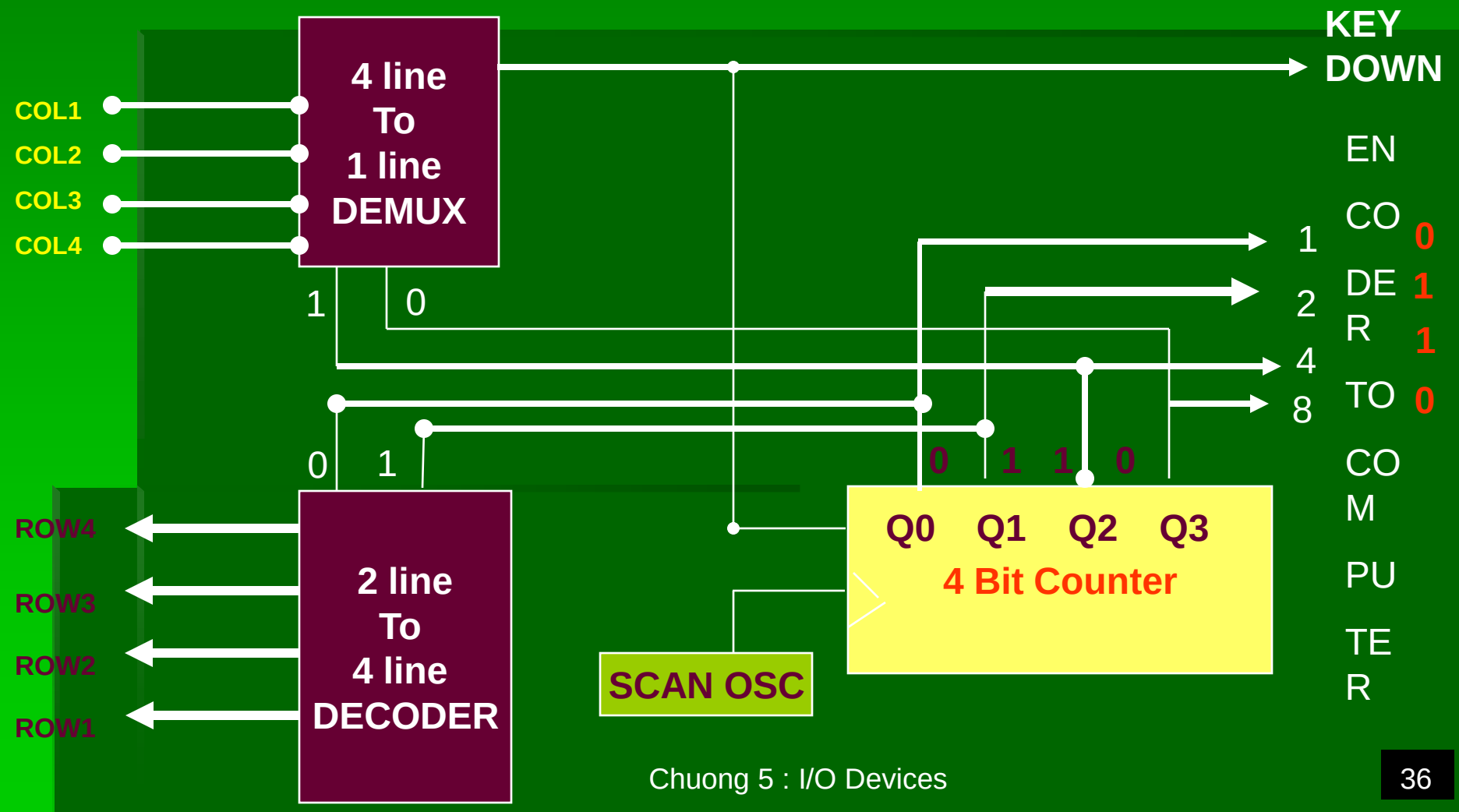
Làm sao MT phân biệt được khi 1 phím được nhấn và khi phím đó được nhả ?

- Khi nhả phím bị ấn, bàn phím tạo ra 1 mã scan khác với mã scan lúc phím bị ấn, có giá trị bằng mã trước cộng thêm 128 (80h) , nghĩa là đổi bit 7 của byte mã scan trước từ 0 ➔ 1

Ex : khi ta ấn chữ z , scan code là 44

Nhả phím này ra bàn phím tạo mã scan 172

Bộ mã hoá quét trên ma trận



HOẠT ĐỘNG NGẮT QUÃNG CỦA IO

Khi 1 IO có yêu cầu giao tiếp với CPU (xuất nhập data), IO này sẽ kích khởi 1 đường tín hiệu IRQ của mình (Interrupt request) để báo là mình cần phục vụ.

Các bước trong tiến trình ngắt quãng :

- IO có yêu cầu CPU phục vụ, sẽ gửi tín hiệu IRQ đến Interrupt controller.
- Nếu có nhiều I/O cùng yêu cầu ngắt , IntController sẽ giải quyết các yêu cầu bằng chế độ ưu tiên.
- IntController phát 1 tín hiệu đến CPU xin ngắt, CPU sẽ hoàn tất lệnh đang thực hiện , cất giá trị của thanh ghi IP và CS vào stack để biết địa chỉ trở về sau khi phục vụ ngắt hoàn tất.

Liên lạc giữa bàn phím và CPU(cont)

- Cứ mỗi lần có 1 tác động ấn phím
- → mạch bàn phím gây ra ngắt 9
- → gọi 1 chương trình con phục vụ ROM BIOS.

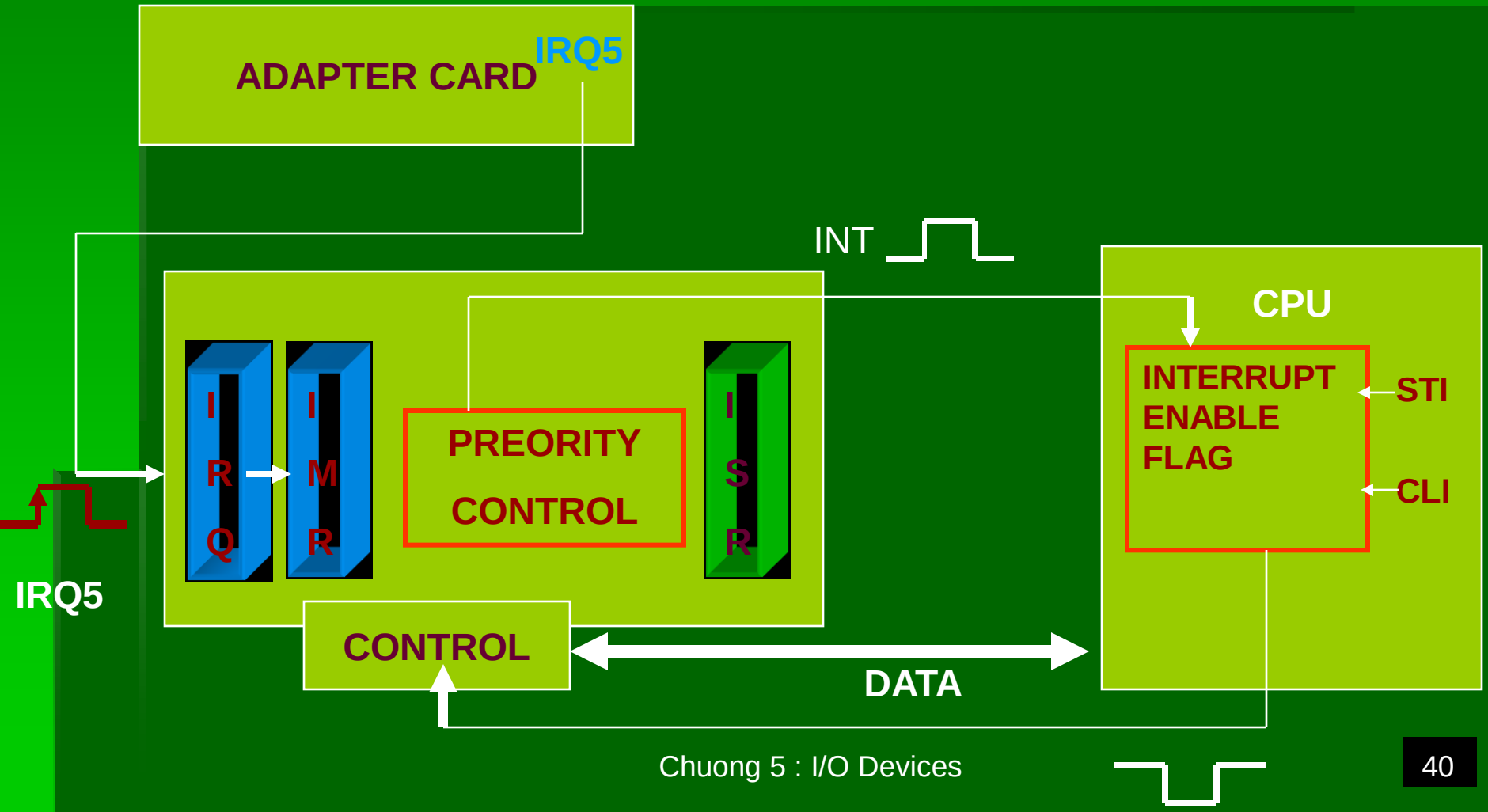
INT 9 sẽ đọc cổng 60H để biết tác động phím nào đã xãy ra (đọc mã scan tương ứng).

INT 9h chuyển mã scan này thành mã dài 2 bytes , byte thấp chứa mã ASCII của phím đó, byte cao chứa mã scan.

HOẠT ĐỘNG NGẮT QUÃNG CỦA I/O

- CPU yêu cầu mã nhận dạng để biết phục vụ cái gì? Nhờ mã này CPU vào bảng Interrupt vector để biết địa chỉ bắt đầu của chương trình con phục vụ ngắt nằm đâu trong bộ nhớ.
- **CPU chép địa chỉ bắt đầu của chương trình con phục vụ ngắt vào CS và thực thi mã lệnh của chương trình này.**
- **Sau khi thực hiện xong tác vụ của ngắt , lệnh cuối cùng là INTR, CPU sẽ lấy giá trị cũ của CS và IP trong stack ra để tiếp tục thực thi các lệnh còn lại của ứng dụng**

Thí dụ minh hoạ về interrupt



Giải thích

- **Card IO yêu cầu phục vụ bằng cách đưa đường tín hiệu IRQ trên card lên mức 1.**

Đường IRQ nối với bộ điều khiển ngắt, yêu cầu này sẽ làm bật bit 5 của thanh ghi IRR.

Interrupt controller sẽ so sánh IRR với thanh ghi mặt nạ IMR để xem hiện tại có cho phép IRQ5 hay không. Nếu cho phép thì sẽ kiểm tra tiếp.

Kiểm tra xem có 1 ngắt có ưu tiên cao hơn IRQ5 hay không. Nếu có thì IRQ5 phải chờ cho đến sau khi ngắt ưu tiên thực hiện xong.

Giải thích (cont)

- Lệnh EOI sẽ xóa bit 5 trong thanh ghi ISR để IntController có thể tiếp nhận 1 yêu cầu khác.

Chương trình phục vụ ngắt phát lệnh IRET kết thúc. CPU phục hồi giá trị CS và IP từ Stack để có thể tiếp tục thực hiện quá trình trước đó.

Giải thích (cont)

- **Interrupt controller đưa đường tín hiệu INT lên mức 1 để báo cho CPU biết có ngắt quãng.**

CPU nhận tín hiệu INT, sẽ báo lại cho Interrupt controller tín hiệu đồng ý bằng cách đưa tín hiệu INTA về mức 0.

Sau khi CPU chấp thuận ngắt, Interrupt controller gửi data cho CPU với trị 0Dh (giả sử IRQ5 tương ứng với ngắt 0Dh).

Bật bit 5 của thanh ghi đang phục vụ ISR, xoá bít 5 của IRR vì IRQ5 đã được giải quyết.

Giải thích (cont)

- **CPU tạm dừng quá trình hiện hành, lưu giá trị CS và IP vào stack. Nhảy đến địa chỉ 0000:4*0Dh , lấy địa chỉ offset của chương trình con phục vụ ngắt 0Dh.**

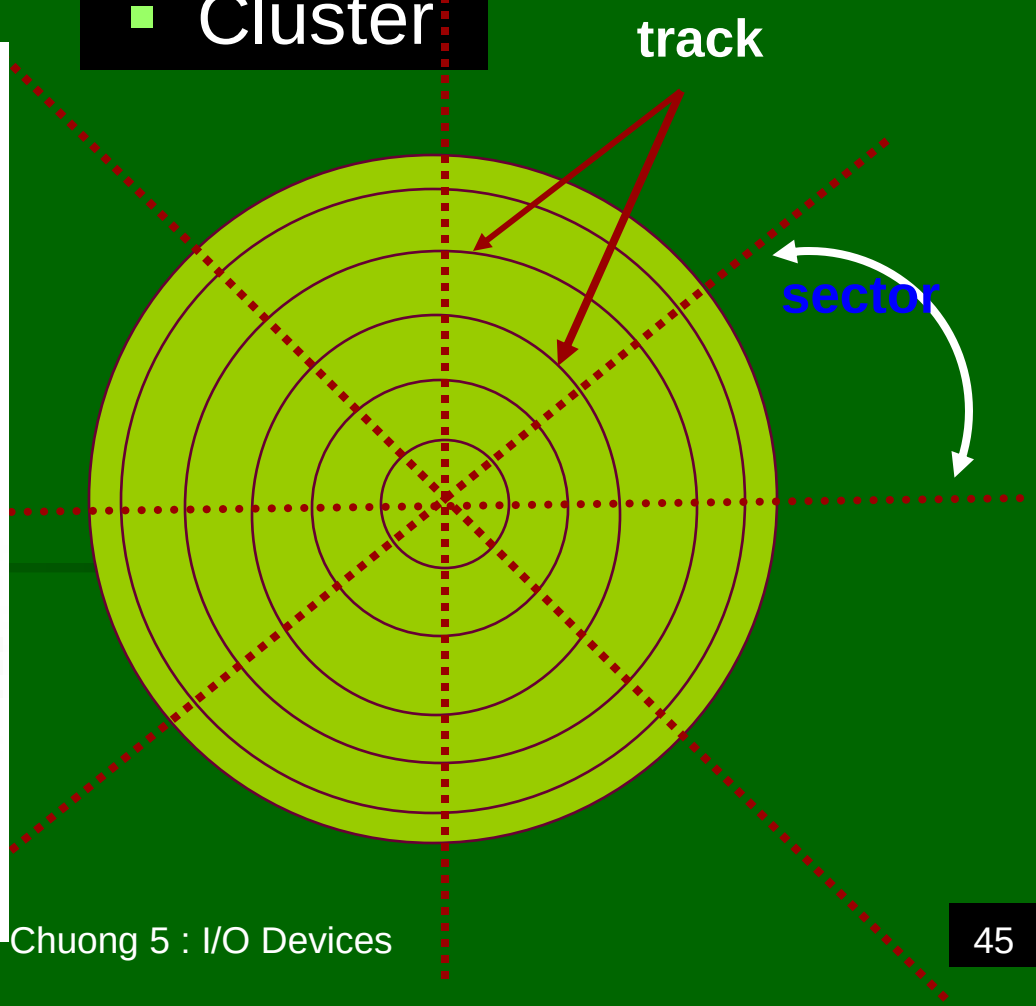
Nhảy đến nơi chứa các lệnh của ngắt này và thực thi các lệnh tương ứng.

Ơû cuối chương trình phục vụ ngắt, CPU gửi giá trị báo kết thúc phục vụ ngắt EOI (End of Interrupt =20h) cho IntControler.

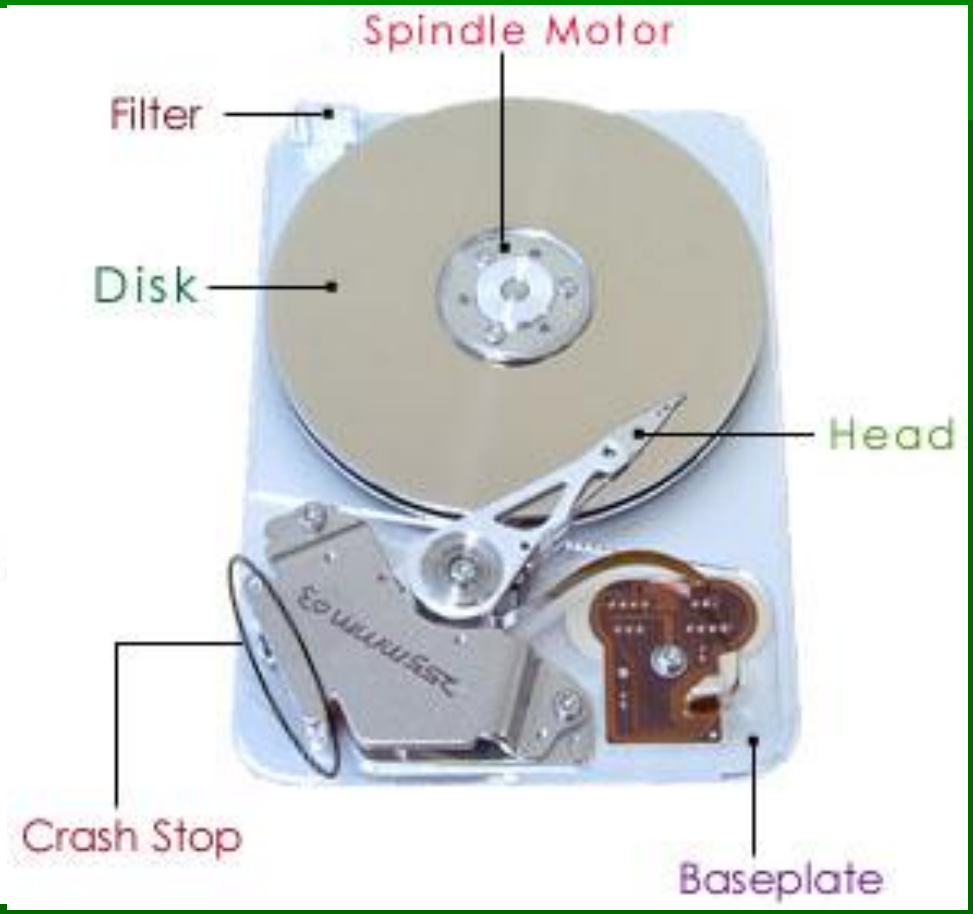
Hard Disk

Các thuật ngữ

- Track
- Cylinder
- Sector
- Cluster



Hard Disk



Hệ thống tập tin của DOS và điều khiển đĩa

- **Bảng FAT : (File Allocation Table)**

Nằm ngay Boot Sector (sector 0).

Bảng FAT được tạo ra khi ta partion đĩa cứng

Nội dung Bảng FAT mô tả trạng thái của các cluster còn tốt hay đã hư (vật lý), đã dùng hay chưa dùng…

Summary slide

- I/O là gì ?
- Mô tả tiến trình phục vụ ngắt quãng.
- Tính toán vị trí của vector ngắt của interrupt 20h.
- Viết các lệnh sử dụng ngắt 21h , hàm 9 để hiển thị ngày hiện tại.
- Bảng FAT là gì ?

Summary slide

- Thế nào là ngắt nội. Cho 1 thí dụ minh họa.
- Khi lập trình, ta thường gọi 1 chương trình phục vụ xuất nhập, lúc đó ta sử dụng loại ngắt nào. Cách gọi.
- Làm sao để phân biệt ngắt cứng và ngắt mềm.
- Khi dùng INT 21h để hiển thị 1 ký tự ra màn hình, thanh ghi nào chứa ký tự sẽ hiển thị?.

Chương 6 : Nhập môn Assembly

- 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.

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 mã 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 cácchương trình viết bằng ngôn ngữ cấp cao .

Lệnh máy

- **Là 1 chuỗi nhị phân có ý nghĩa đặc biệt – nó ra lênh 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

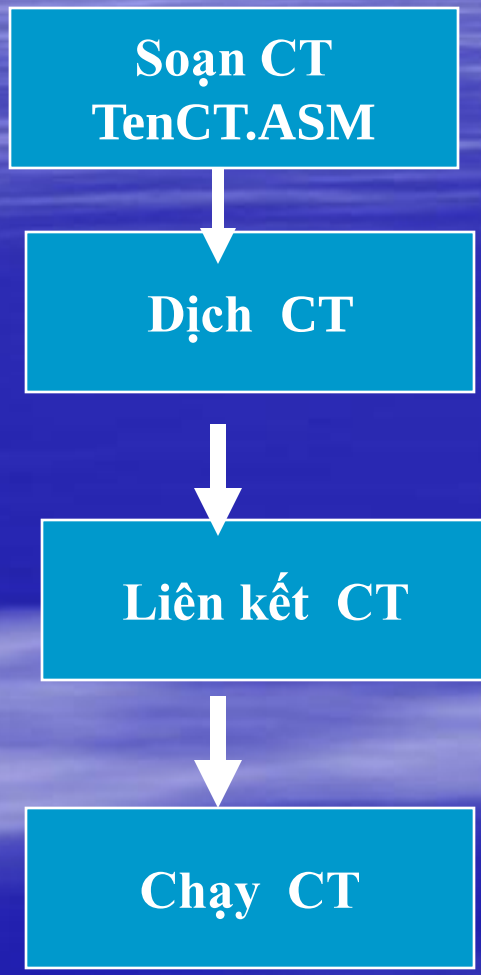
Lệnh máy (cont)

- 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

Ýù 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 soan CT Assembly như : NotePad, NC, màn hình C, Pascal ...

CT có phần mở rộng là .ASM

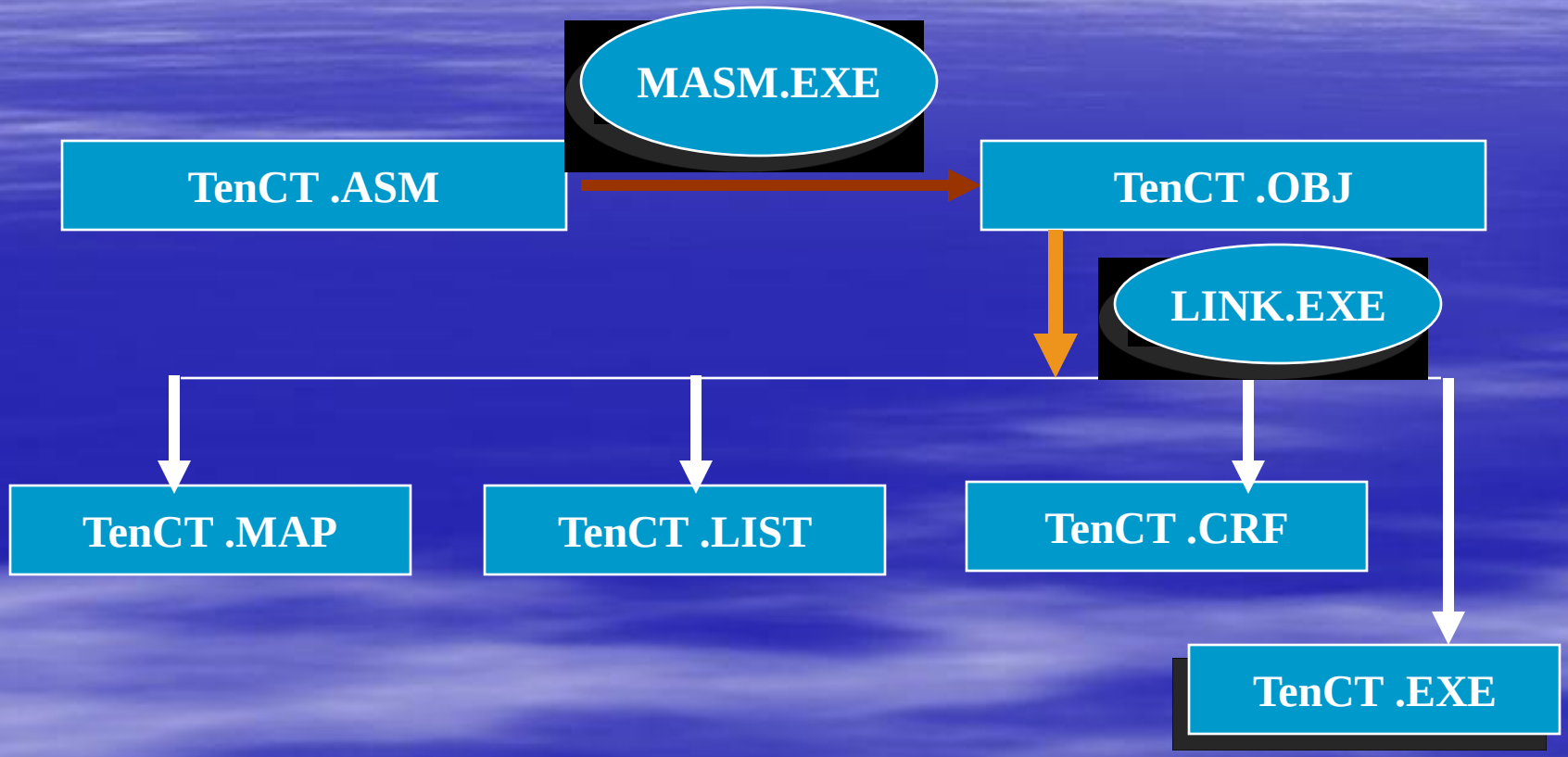
dùng MASM để dịch chương trình nguồn .ASM

➔ File Object.

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

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

Dịch và nối kết chương trình



Một chương trình minh hoạ

```
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

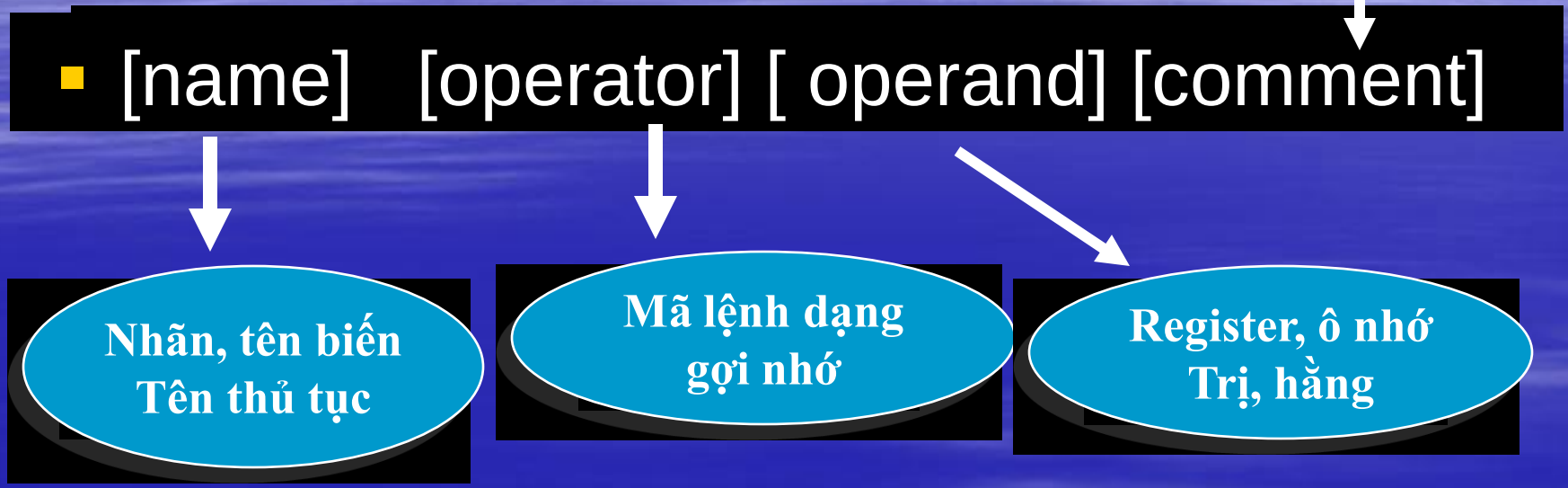
Giải thích

- .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**

Dạng lệnh



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 ngắt được dùng để gọi chương trình ngắt của DOS và BIOS.

Ngắt 21h

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

Function_number	chức năng
1	nhập 1 ký tự từ bàn phím
2	Xuất 1 ký tự ra mà hình.
9	Xuất 1 chuỗi ký tự ra màn hình

INT 21h (cont)

Hàm 1 : Nhập 1 ký tự

Input : AH =1
Output : AL = mã ASCCI của phím ấn
= 0 nếu 1 phím điều khiểân 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

[illegible] DUP (?)

DW 1000h, ‘AB’, 1024

DW ?

DW 5 DUP (1000h)

DW 256*2

DẠNG LƯU TRỮ DỮ LIỆU KIỂU WORD :

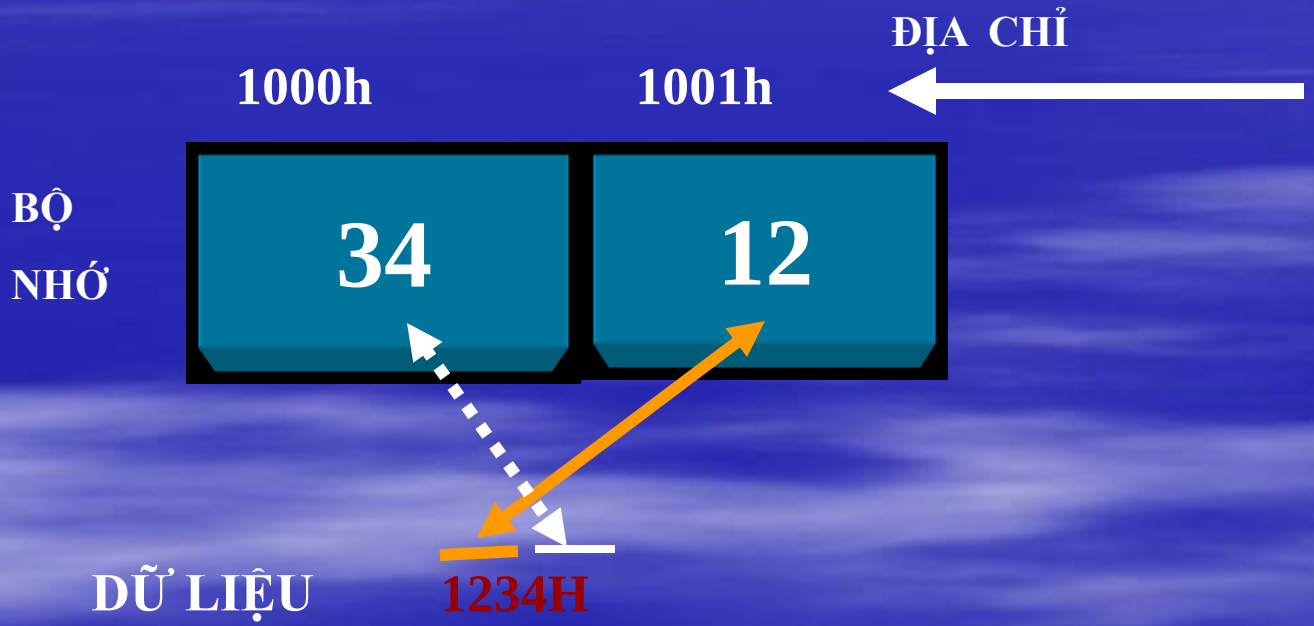
Trình hợp dịch đảo ngược các byte trong 1 giá trị kiểu WORD khi lưu trữ trong bộ nhớ :

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

Minh họa khai báo biến

KIỂU 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
```

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
.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
```

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 logic

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

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 xxxxx (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

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 hoạ 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 Ơ NHỚ 100h vào Reg AX

Áp dụng

Viết chương trình chuyển nội dung vùng nhớ bắt đầu tại địa chỉ 700 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 DS.

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 ⟵ 1000h

LEA SI, [DI][BX][2000h] ; SI ⟵ 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.**

Cacù 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 01111111b 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 ➔ AL =0

cờ SIGN được lập ➔ AL <0

cờ ZR và cờ SIGN không được lập ➔ 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 trỏ đến nhãn đích các lệnh sau nhãn này sẽ được thực hiện.

```
……
MOV AH, 2
MOV CX, 26
MOV DL, 41H
PRINT_LOOP :
  INT 21H
  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

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 ALø BL và CL . Kết quả để trong biến SMALL

```
MOV  SMALL, AL          ; giả sử AL nhỏ nhất
CMP SMALL, BL           ; nếu SMALL <= BL thì
JBE L1                  Nhảy đến L1
MOV SMALL, BL           ; nếu SMALL <= CL thì
L1 :
  CMP SMALL, CL         ; Nhảy đến L2
 JBE  L2
MOV SMALL, CL           ; CL là số nhỏ nhất
L2 : . . .
```

Các lệnh dịch và quay bit

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

Cú pháp : SHL toán hạng đích ,1

Dịch 1 vị trí.

Cú pháp : SHL toán hạng đích ,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 **EXCUTE**
CMP A1,OP2
JGE **EXCUTE**
CMP A1,OP3
JE **EXCUTE**
CMP A1,OP4
JL **EXCUTE**
JMP CONTINUE
EXCUTE : <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.

VÒNG LẶP WHILE

Cấu trúc 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 : …..
```

VÒNG LẶP WHILE CÓ LỒNG IF

Cấu trúc WHILE có lồng IF

Cấu trúc logic

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

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 : …..
```

VÒNG LẶP REPEAT UNTIL

Cấu trúc REPEAT UNTIL

Cấu trúc logic

REPEAT

<STATEMENT1>

<STATEMENT2>

<STATEMENT3>

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

Bằng nhau thoát Repeat

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

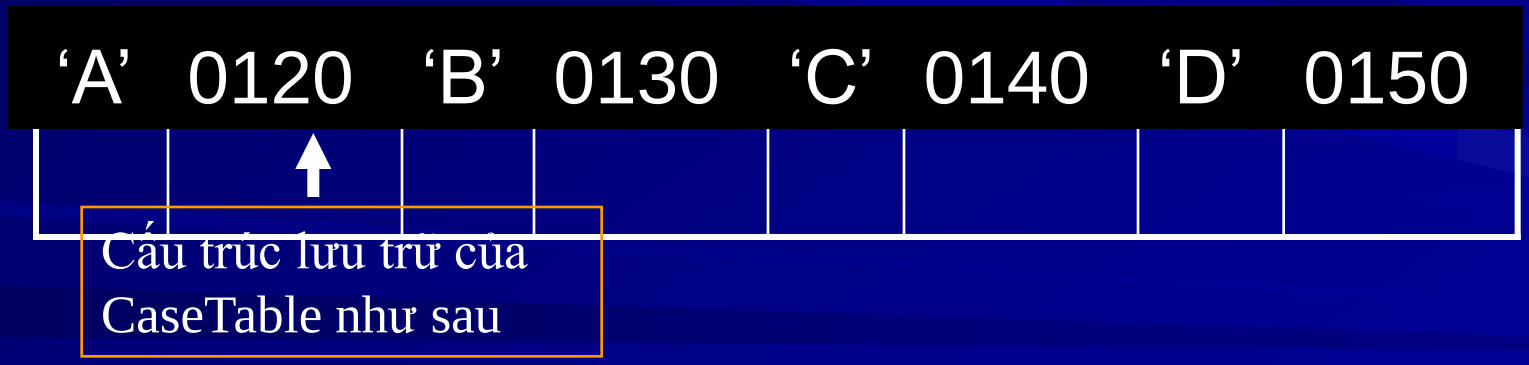
```
CASE : MOV AL, INPUT
CMP  AL, ‘A’
JNE TESTB
CALL PROC_A
JMP  ENDCASE
TESTB :
  CMP AL, ‘B’
 JNE TESTC
CALL PROC_B
JMP  ENDCASE
TESTC :
CMP AL, ‘C’
JNE TESTD
CALL  PROC_C
JMP  ENDCASE
TESTD : CMP  AL, ‘D’
JNE  ENDCASE
CALL  PROC_D
ENDCASE : …….
```

Rất hiệu quả khi xử lý phát biểu CASE là dùng bảng OFFSET chứa địa chỉ của nhãn hoặc của hàm sẽ nhảy đến tuỳ 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
Dw  Proc_A               giả sử  ở địa chỉ 0120
Db 'B'
Dw  Proc_B               giả sử ở địa chỉ 0130
Db 'C'                   giả sử ở địa chỉ 0140
Dw  Proc_C
Db 'D'                   giả sử ở địa chỉ 0150
Dw  Proc_D
```



Cấu trúc lưu trữ của CaseTable như sau

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 hoá 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 nấy.

- 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

ĐỊNH 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 cĩ nhiều đoạn code giống nhau, chúng ta cĩ thể dùng macro để thay thế, giống như ta dùng define trong C. Thí dụ chúng ta thay thế đọan lệnh sau bằng macro để in dấu xuống dịng.

```
MOV DL,13 ; về đầu dịng
MOV AH,2
INT 21H
MOV DL,10 ; xuống dịng mới
MOV AH,2
INT 21H
```

Thay vì phải viết lại 6 dịng lệnh trên, ta cĩ thể tạo 1 macro cĩ 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 OBJ.
  - 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
```

- 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 (SUPRESS ALL) PHẦN MỞ RỘNG MACRO KHÔNG ĐƯỢC IN. SỬ DỤNG KHI MACRO LỚN HAY MACRO ĐƯỢC THAM CHIẾU NHIỀU LẦN TRONG CT.
  - .XALL CHỈ NHỮNG DÒNG MACRO TẠO MÃ NGUỒN MỚI ĐƯỢC IN RA.THÍ DỤ 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
```

GỌI : 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 ĐỤNG 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

RONG MACRO CÓ THỂ CÓ NHÃN.

ỌI MACRO NHIỀU LẦN ➔ NHIỀU NHÃN ĐƯỢC TẠO RA

LÀM SAO GIẢI QUYẾT VẤN ĐỀ NHẢY ĐIỀU KHIỂN?

SEMBLY GIẢI QUYẾT VẤN ĐỀ NÀY BẰNG CHỈ THỊ LOCAL
ỖNG BỨC MASM TẠO RA 1 TÊN DUY NHẤT CHO MỖI MỘT
BEL 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 đưới 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
```

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 LƯỢT LẤY HẾT CÁC TRỊ TRONG DANH SÁCH.**

EX : PROCTABLE LABEL WORD

```
    IRP PROCNAME, <MOVEUP, MOVDOWN,MOVLEFT,MOVRGHT>
    DW  PROCNAME
  ENDM
```

CÁC LỆNH LẶP TRONG MACRO

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

PROCTABLE DW MOVUP, MOVDOWN,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 tốn USCLN như sau :

```
WHILE   N <> 0 DO
    M = M MOD N
    Hốn 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 chuyen tu chuoi thanh so chua trong ax

; in : DI =offset chuoi

; out : AX =so da doi

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

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

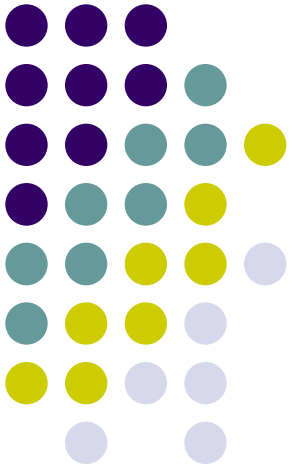
Bài 5 : Viết Macro in số hecxa chứa trong BL ra dạng binary

;Input: BL chứa số cần in

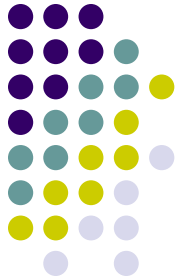
;Output: Nothing

Chương 10
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



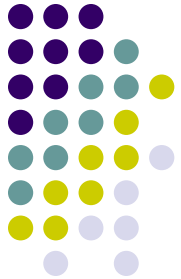
GiỚI THIỆU STACK



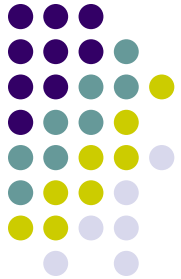
STACK : là một cấu trúc dữ liệu một chiều. Các phần tử cất vào và lấy ra theo phương thức LIFO (Last In First Out). Mỗi chương trình phải dành ra một khối bộ nhớ để làm stack bằng khai báo STACK. Ví dụ : .STACK 100H ; Xin cấp phát 256 bytes làm stack

- **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 STCAK luôn luôn bị thay đổi.**

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

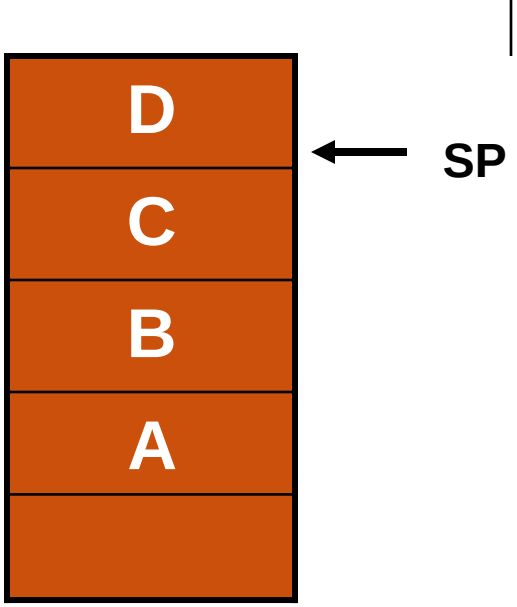
THÍ DỤ

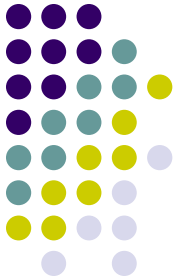
A,B,C là các Word
MOV BP,SP

```
MOV AX,[BP]      ;AX = D

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

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

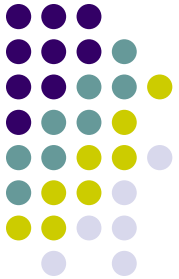




Để lưu 1 phần tử vào Stack ta dùng lệnh PUSH
Để lấy 1 phần tử ra từ Stack ta dùng lệnh POP

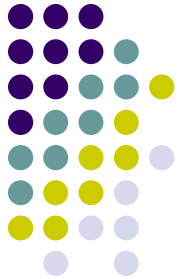
PUSH nguồn : đưa nguồn vào đỉnh STACK
PUSHF : cất nội dung thanh ghi cờ vào STACK

- **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 : cất 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

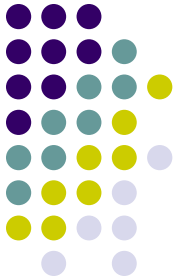
- **Truyền tham số cho các chương trình con**

- **Lưu tạm thời giá trị thanh ghi hay biến.**

THÍ DỤ 2

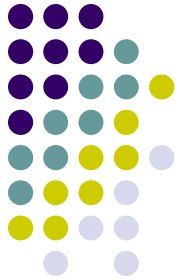
- Nhập vào 1 chuỗi, in chuỗi đảo ngược
  Ex : nhập : Cong nghe thong tin
  xuất : int gnoht ehgn gnoC

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
- Cất 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

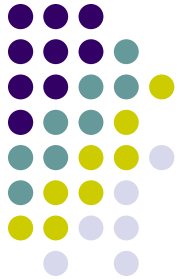


- **CTC là 1 nhóm các lệnh được gộp lại dưới 1 cái tên mà ta có thể gọi từ nhiều nơi khác nhau trong chương trình thay vì phải viết lại các nhóm lệnh này tại nơi cần đến chúng.**

Lợi ích

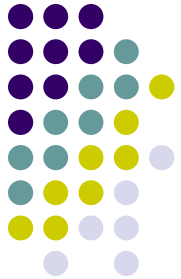
CTC làm cho cấu trúc logic của của CT dễ kiểm soát hơn, dễ tìm sai sót hơn và có thể tái sử dụng mã ➔ tiết kiệm được công sức và thời gian lập trình.

CẤU TRÚC CỦA CTCON



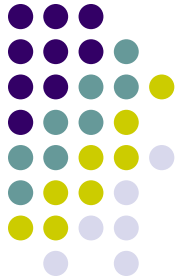
```
TÊNCTC  PROC  [NEAR|FAR]
    CÁC LỆNH CỦA CTC
   RET
TÊNCTC  ENDP
```

MINH HỌA



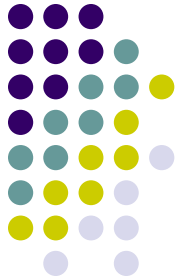
- Viết chương trình nhập 1 số n (n nguyên dương và <9). Tính giai thừa của n và xuất ra màn hình dưới dạng số hex (giới hạn kết quả 16 bit).
- Viết chương trình tìm số hoàn thiện (giới hạn 2 chữ số) và in nó ra màn hình.

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 gặp 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 gặp 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, Extrn 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.

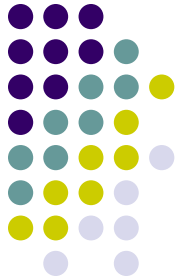
Cú pháp : EXTRN Tên nhãn : Kiểu

- BYTE
- DATAPTR
- WORD
- NEAR
- DWORD
- FAR
- PROC

THAY THẾ PUBLIC VÀ EXTRN.

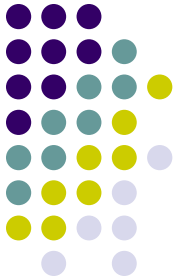
Viết chương trình nằm trên 2 file (2 module) với sự phân công như sau :
Module của chương trình chính (Main.ASM) có nhiệm vụ xác định Offset của 2 chuỗi ký tự và gọi CTC nối 2 chuỗi này và cho hiện kết quả ra màn hình.
Module CTC (Sub.ASM) làm nhiệm vụ nối 2 chuỗi và đưa vào bộ nhớ.

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



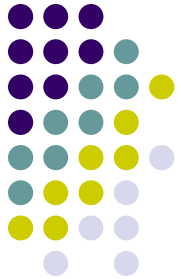
```
PrintNum10    PROC
; số nguyên N nằm trong AX
              PUSH   BX     CX    DX
              MOV    CX, 0          ; so lan push (so ky tu)
laysodu:
          XOR     DX, DX       ;  cho DX = 0 trước khi chia
                  MOV    BX, 10
                  DIV     BX             ; số dư trong 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**

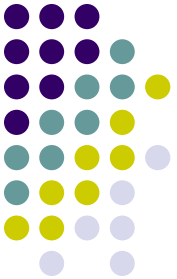
Có 3 cách truyền tham số

Thông qua thanh ghi

Thông qua biến toàn cục

Thông qua STACK

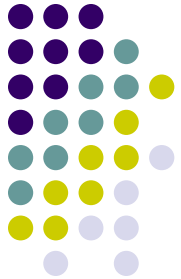
TRUYỀN THAM SỐ THÔNG QUA THANH GHI



- 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



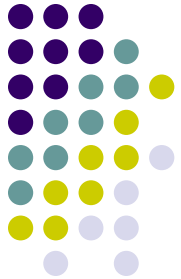
- KHAI BÁO BiẾN TOÀN CỤC.
- DÙNG NÓ ĐỂ CHUYỂN CÁC GIÁ TRỊ GiỮA CT GỌI VÀ CT ĐƯỢC GỌI.

CÁCH NÀY THƯỜNG ĐƯỢC DÙNG :

TRONG 1 CT ViẾT THUẦN TÚY BẰNG ASM

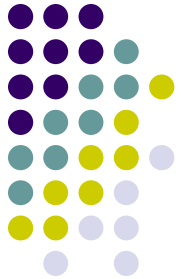
ViẾT HỖN HỢP GiỮA ASM VÀ 1 NGÔN NGỮ CẤP CAO

TRUYỀN THAM SỐ QUA STACK



- **PHỨC TẠP HƠN.**
- **DÙNG RẤT NHIỀU KHI ViẾT CHƯƠNG TRÌNH HỖN HỢP GiỮA ASM VÀ NGÔN NGỮ CẤP CAO.**

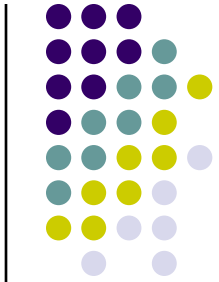
CHUYỂN GIÁ TRỊ TỪ CTCON LÊN CT CHÍNH.



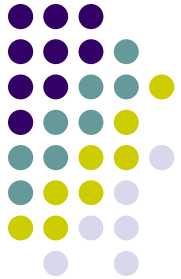
- CŨNG THÔNG QUA CÁC THANH GHI,BỘ NHỚ VÀ STACK.

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 CTCON LÊN CT CHÍNH.



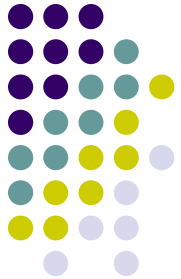
- **NẾU GIÁ TRỊ QUAY LẠI LÀ 32 BIT (CHO KHAI BÁO LONG, CON TRỎ XA) THÌ GIÁ TRỊ ĐÓ PHẢI ĐƯỢC ĐẶT TRONG THANH GHI DX,AX CỦA HÀM TRƯỚC KHI QUAY VỀ CT CHÍNH.**



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

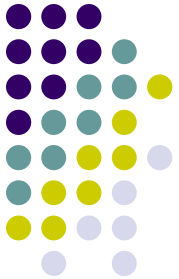
- **NEAR : lấy địa chỉ OFFSET (16BIT) trong STACK và gán vào thanh ghi IP.**
- **FAR : lấy địa chỉ OFFSET và SEGMENT trong STACK nạp vào thanh ghi CS:IP.**

VẤN ĐỀ BẢO VỆ CÁC THANH GHI



- CẦN ĐƯỢC QUAN TÂM TRONG QUÁ TRÌNH LẬP TRÌNH ASM.
- RẤT DỄ XẢY 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



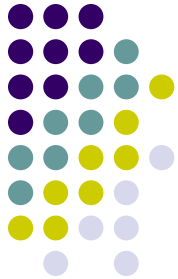
NHẬP VÀO 1 SỐ HỆ HEX. IN RA SỐ ĐÃ NHẬP VỚI YÊU CẦU SAU :

ViẾT CTCON NHẬP SỐ

ViẾT CTCON XUẤT SỐ

CTCHÍNH GỌI 2 CTCON TRÊN.

LUYỆN TẬP LẬP TRÌNH C10

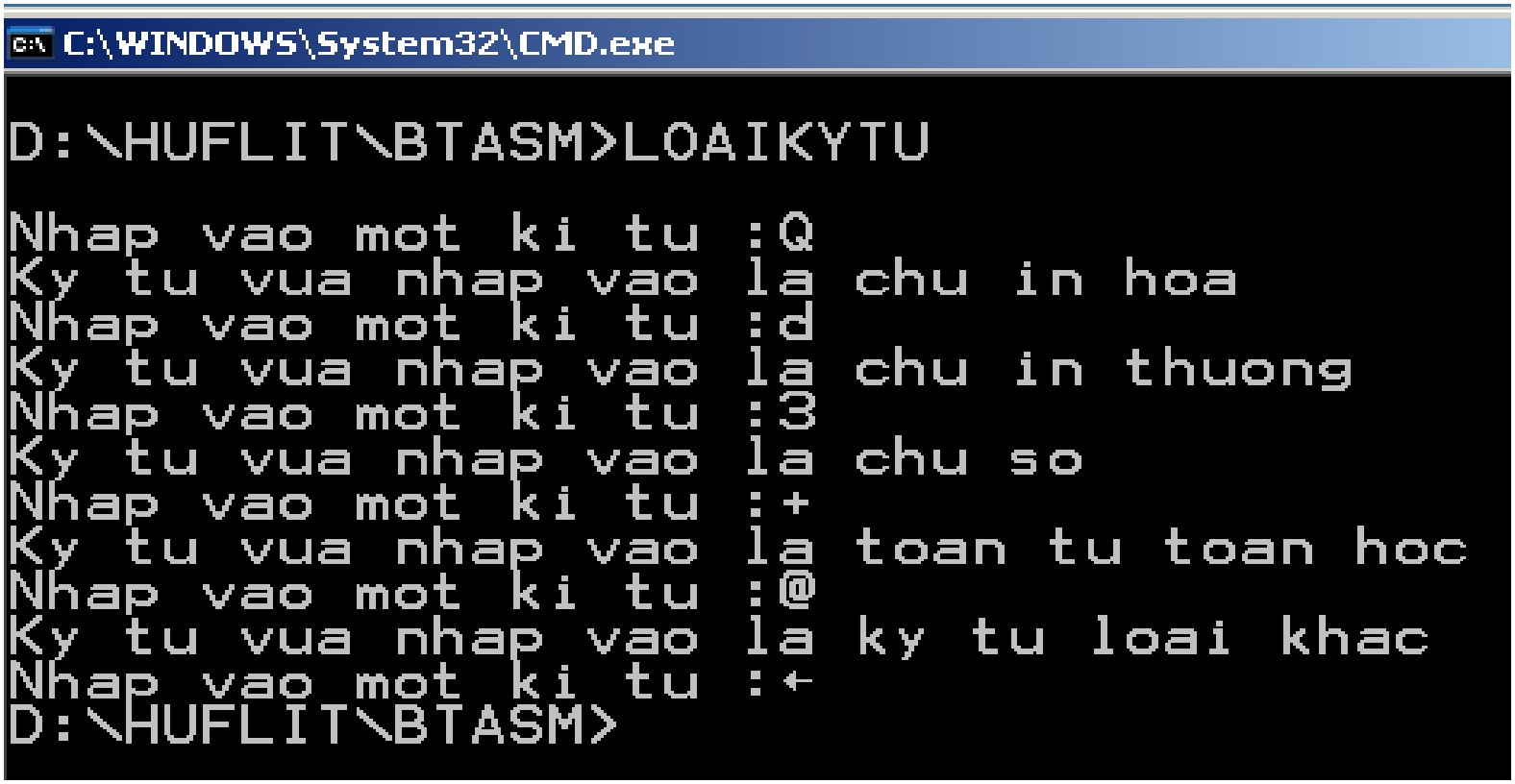


Bài 1 : Viết chương trình nhập 1 số nguyên n (n<9). Tính giai thừa của n và xuất kết quả ra màn hình dưới dạng số Hex (giới hạn 16 bits).

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 gi ? – ký tự, ký số ,toán tử toán học hay ký tự khác. Nếu ký tự là phím Escape thì thoát chương trình.



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.