HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

HOÀNG XUÂN DẬU

BÀI GIẢNG
KIẾN TRÚC MÁY TÍNH

HÀ NỘI 2010

LỜI NÓI ĐẦU

Kiến trúc máy tính là một trong các lĩnh vực khoa học cơ sở của ngành Khoa học máy tính nói riêng và Công nghệ thông tin nói chung. Kiến trúc máy tính là khoa học về lựa chọn và ghép nối các thành phần phần cứng của máy tính nhằm đạt được các mục tiêu về hiệu năng cao, tính năng đa dạng và giá thành thấp.

Môn học Kiến trúc máy tính là môn học cơ sở chuyên ngành trong chương trình đào tạo công nghệ thông tin hệ đại học và cao đẳng. Mục tiêu của môn học là cung cấp cho sinh viên các kiến thức cơ sở của kiến trúc máy tính, bao gồm bao gồm kiến trúc máy tính tổng quát, kiến trúc bộ xử lý trung tâm và các thành phần của bộ xử lý trung tâm, kiến trúc tập lệnh máy tính, cơ chế ống lệnh; hệ thống phân cấp của bộ nhớ, bộ nhớ trong, bộ nhớ cache và các loại bộ nhớ ngoài; hệ thống bus và các thiết bị vào ra.

Kiến trúc máy tính là một lĩnh vực đã được phát triển trong một thời gian tương đối dài với lượng kiến thức đồ sộ, nhưng do khuôn khổ của tài liệu có tính chất là bài giảng môn học, tác giả cố gắng trình bày những vấn đề cơ sở nhất phục vụ mục tiêu môn học. Nội dung của tài liệu được biên soạn thành sáu chương:

Chương 1 là phần giới thiệu các khái niệm cơ sở của kiến trúc máy tính, như khái kiệm kiến trúc và tổ chức máy tính; cấu trúc và chức năng các thành phần của máy tính; các kiến trúc máy tính von-Neumann và kiến trúc Harvard. Khái niệm về các hệ đếm và cách tổ chức dữ liệu trên máy tính cũng được trình bày trong chương này.

Chương 2 giới thiệu về khối xử lý trung tâm, nguyên tắc hoạt động và các thành phần của nó. Khối xử lý trung tâm là thành phần quan trọng và phức tạp nhất trong máy tính, đóng vai trò là bộ não của máy tính. Thông qua việc thực hiện các lệnh của chương trình bởi khối xử lý trung tâm, máy tính có thể thực thi các yêu cầu của người sử dụng.

Chương 3 giới thiệu về tập lệnh của máy tính, bao gồm các khái niệm về lệnh, dạng lệnh, các thành phần của lệnh; các dạng địa chỉ và các chế độ địa chỉ. Chương cũng giới thiệu một số dạng lệnh thông dụng kèm ví dụ minh hoạ. Ngoài ra, cơ chế ống lệnh – xử lý xen kẽ các lệnh cũng được đề cập.

Chương 4 trình bày về bộ nhớ trong: khái quát về hệ thống bộ nhớ và cấu trúc phân cấp của hệ thống nhớ; giới thiệu các loại bộ nhớ ROM và RAM. Một phần rất quan trọng của chương là phần giới thiệu về bộ nhớ cache - một bộ nhớ đặc biệt có khả năng giúp tăng tốc hệ thống nhớ nói riêng và cả hệ thống máy tính nói chung.

Chương 5 giới thiệu về bộ nhớ ngoài, bao gồm các loại đĩa từ, đĩa quang, các hệ thống RAID, NAS và SAN. Bộ nhớ ngoài là dạng bộ nhớ thường có dung lượng lớn và dùng để lưu trữ thông tin ổn định, không phụ thuộc nguồn điện nuôi.

Chương 6 trình bày về hệ thống bus và các thiết bị ngoại vi. Phần trình bày về hệ thống bus đề cập đến các loại bus như ISA, EISA, PCI, AGP và PCI-Express. Phần giới thiệu các thiết bị vào ra đề cập đến nguyên lý hoạt động của một số thiết bị vào ra thông dụng, như bàn phím, chuột, màn hình và máy in.

Tài liệu được biên soạn dựa trên kinh nghiệm giảng dạy môn học Kiến trúc máy tính trong nhiều năm của tác giả tại Học viện Công nghệ Bưu chính – Viễn thông, kết hợp tiếp thu các đóng góp của đồng nghiệp và phản hồi từ sinh viên. Tài liệu có thể được sử dụng làm tài liệu học tập cho sinh viên hệ đại học và cao đẳng các ngành công nghệ thông tin và điện tử viễn thông. Trong quá trình biên soạn, mặc dù tác giả đã rất cố gắng song không thể tránh khỏi có những thiếu sót. Tác giả rất mong muốn nhận được ý kiến phản hồi và các góp ý cho các thiếu sót, cũng như ý kiến về việc cập nhật, hoàn thiện nội dung của tài liệu.

Hà nội, tháng 8 năm 2010

Tác giả

TS. Hoàng Xuân Dậu

Email: dauhx@ptit.edu.vn

MỤC LỤC

DANH MỤC CÁC THUẬT NGỮ TIẾNG ANH VÀ VIẾT TẮT

Thuật ngữ tiếng Anh	Từ viết tắt	Thuật ngữ tiếng Việt/Giải thích
Central Processing Unit	CPU	Bộ/Đơn vị xử lý trung tâm
Control Unit	CU	Bộ/Đơn vị điều khiển
Arithmethic and Logic Unit	ALU	Bộ/Đơn vị tính toán số học và logic
Program Counter	PC	Bộ đếm chương trình
System Bus		Buýt hệ thống
Memory		Bộ nhớ
Cache		Bộ nhớ đệm / bộ nhớ kết
Random Access Memory	RAM	Bộ nhớ truy cập ngẫu nhiên
Read Only Memory	ROM	Bộ nhớ chỉ đọc
Basic Input Output System	BIOS	Hệ thống vào ra cơ sở
Pipeline		Cơ chế ống lệnh hay cơ chế xử lý xen kẽ các lệnh
Hit		Đoán trúng – là sự kiện CPU truy tìm một mục tin và tìm thấy trong cache.
Miss		Đoán trượt – là sự kiện CPU truy tìm một mục tin và không tìm thấy trong cache.
Advanced Technology Attachments	ATA	Chuẩn ghép nối đĩa cứng ATA
Parallel Advanced Technology Attachments	PATA	Chuẩn ghép nối đĩa cứng PATA – hay ATA song song
Integrated Drive Electronics	IDE	Chuẩn ghép nối đĩa cứng IDE
Serial ATA	SATA	Chuẩn ghép nối đĩa cứng SATA – hay ATA nối tiếp
Small Computer System Interface	SCSI	Chuẩn ghép nối đĩa cứng SCSI
Redundant Array of Independent Disks	RAID	Công nghệ lưu trữ RAID – tạo thành từ một mảng liên kết các đĩa cứng vật lý
Network Attached Storage	NAS	Hệ thống lưu trữ gắn vào mạng
Storage Area Network	SAN	Mạng lưu trữ
Industrial Standard Architecture	ISA	Buýt theo chuẩn công nghiệp ISA
Extended ISA	EISA	Buýt theo chuẩn công nghiệp mở rộng EISA
Peripheral Component Interconnect	PCI	Bus PCI
Accelerated Graphic Port	AGP	Cổng tăng tốc đồ hoạ AGP
PCI Express	PCIe	Buýt PCIe
Cathode Ray Tube	CRT	Màn hình ống điện tử âm cực
Liquid Crystal Display	LCD	Mình hình tinh thể lỏng

CHƯƠNG 1 GIỚI THIỆU CHUNG

1.1 KHÁI NIỆM VỀ KIẾN TRÚC VÀ TỔ CHỨC MÁY TÍNH

Kiến trúc máy tính (Computer Architecture) và *Tổ chức máy tính* (Computer Organization) là hai trong số các khái niệm cơ bản của ngành *Công nghệ máy tính* (Computer Engineering). Có thể nói kiến trúc máy tính là bức tranh toàn cảnh về hệ thống máy tính, còn tổ chức máy tính là bức tranh cụ thể về các thành phần phần cứng của hệ thống máy tính.

Kiến trúc máy tính là khoa học về việc lựa chọn và kết nối các thành phần phần cứng để tạo ra các máy tính đạt được các yêu cầu về chức năng (functionality), hiệu năng (performance) và giá thành (cost). Yêu cầu chức năng đòi hỏi máy tính phải có thêm nhiều tính năng phong phú và hữu ích; yêu cầu hiệu năng đòi hỏi máy tính phải đạt tốc độ xử lý cao hơn và yêu cầu giá thành đòi hỏi máy tính phải càng ngày càng rẻ hơn. Để đạt được cả ba yêu cầu về chức năng, hiệu năng và giá thành là rất khó khăn. Tuy nhiên, nhờ có sự phát triển rất mạnh mẽ của công nghệ vi xử lý, các máy tính ngày nay có tính năng phong phú, nhanh hơn và rẻ hơn so với máy tính các thế hệ trước.

Kiến trúc máy tính được cấu thành từ 3 thành phần con: (i) *Kiến trúc tập lệnh* (Instruction Set Architecture), (ii) *Vi kiến trúc* (Micro Architecture) và *Thiết kế hệ thống* (System Design).

- Kiến trúc tập lệnh là hình ảnh của một hệ thống máy tính ở mức ngôn ngữ máy. Kiến trúc tập lệnh bao gồm các thành phần: tập lệnh, các chế độ địa chỉ, các thanh ghi, khuôn dạng địa chỉ và dữ liệu.
- Vi kiến trúc là mô tả mức thấp về các thành phần của hệ thống máy tính, phối ghép và việc trao đổi thông tin giữa chúng. Vi kiến trúc giúp trả lời hai câu hỏi (1) Các thành phần phần cứng của máy tính kết nối với nhau như thế nào? và (2) Các thành phần phần cứng của máy tính tương tác với nhau như thế nào để thực thi tập lệnh?
- Thiết kế hệ thống: bao gồm tất cả các thành phần phần cứng của hệ thống máy tính, bao gồm: Hệ thống phối ghép (các bus và các chuyển mạch), Hệ thống bộ nhớ, Các cơ chế giảm tải cho CPU (như truy nhập trực tiếp bộ nhớ) và Các vấn đề khác (như đa xử lý và xử lý song song).

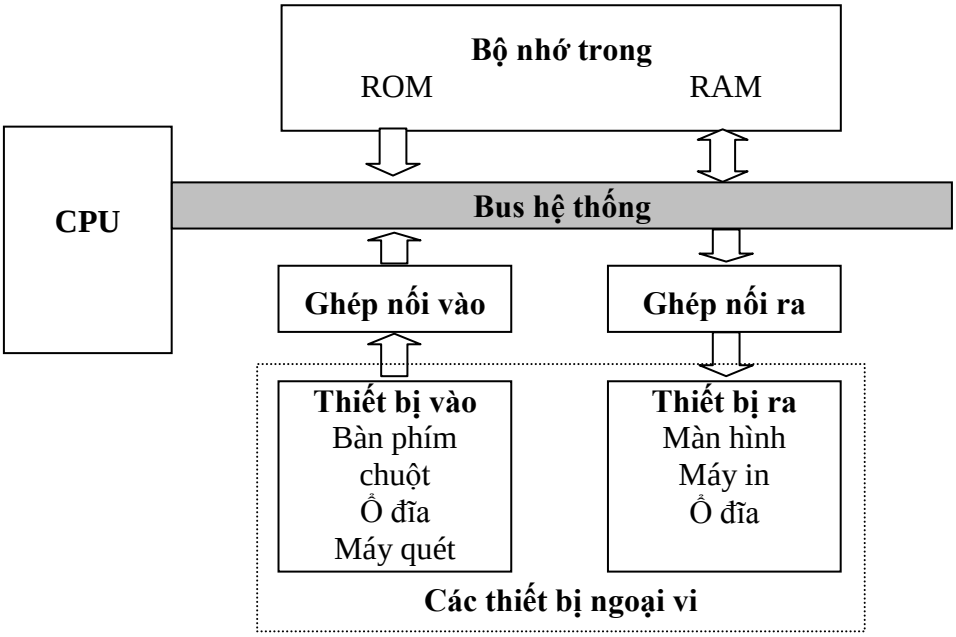
Tổ chức máy tính hay cấu trúc máy tính là khoa học nghiên cứu về các bộ phận của máy tính và phương thức làm việc của chúng. Với định nghĩa như vậy, tổ chức máy tính khá gần gũi với vi kiến trúc – một thành phần của kiến trúc máy tính. Như vậy, có thể thấy rằng, kiến trúc máy tính và khái niệm rộng hơn, nó bao hàm cả tổ chức hay cấu trúc máy tính.

1.2 CẤU TRÚC VÀ CHỨC NĂNG CÁC THÀNH PHẦN CỦA MÁY TÍNH

1.2.1 Sơ đồ khối chức năng

Hình 1 minh hoạ sơ đồ khối chức năng của một hệ thống máy tính. Theo đó, hệ thống máy tính gồm bốn thành phần chính: (1) CPU – Khối xử lý trung tâm, (2) Bộ nhớ trong, gồm bộ nhớ ROM và bộ nhớ RAM, (3) Các thiết bị ngoại vi, gồm các thiết bị vào và các thiết bị ra và (4) Bus hệ thống, là hệ thống kênh dẫn tín hiệu ghép nối các thành phần kể trên. Ngoài ra, còn

có các giao diện ghép nối vào và ghép nối ra dùng để ghép nối các thiết bị ngoại vi vào bus hệ thống. Mục 1.2.2 tiếp theo sẽ mô tả chi tiết chức năng của từng khối.



Hình 1. Sơ đồ khối chức năng của hệ thống máy tính

1.2.2 Các thành phần của máy tính

1.2.2.1 Khối xử lý trung tâm

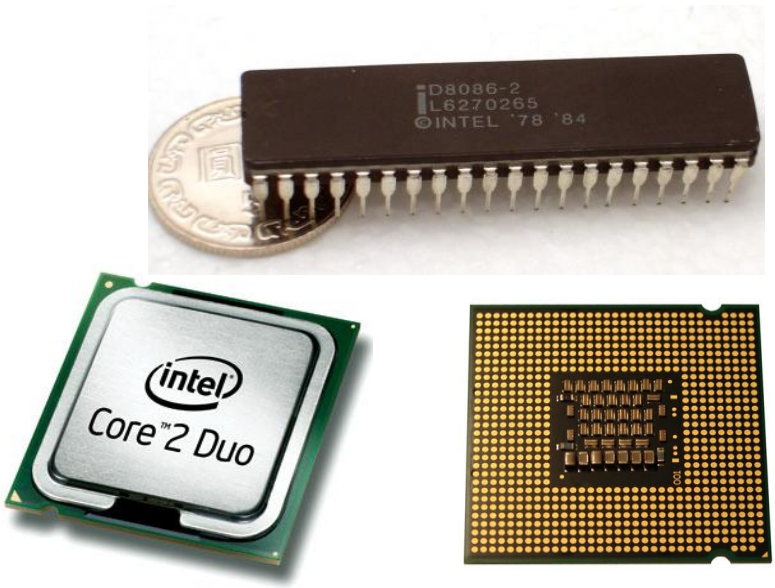
Khối xử lý trung tâm (Central Processing Unit - CPU) là thành phần quan trọng nhất - được xem là bộ não của máy tính. Các yêu cầu của hệ thống và của người sử dụng thường được biểu diễn thành các chương trình máy tính, trong đó mỗi chương trình thường được tạo thành từ nhiều lệnh của CPU. CPU đảm nhiệm việc đọc các lệnh của chương trình từ bộ nhớ, giải mã và thực hiện lệnh. Thông qua việc CPU thực hiện các lệnh của chương trình, máy tính có khả năng cung cấp các tính năng hữu ích cho người sử dụng.

CPU là vi mạch tích hợp với mật độ rất cao, được cấu thành từ bốn thành phần con: (1) Bộ điều khiển (Control Unit - CU), (2) Bộ tính toán số học và logic (Arithmetic and Logic Unit - ALU), (3) Các thanh ghi (Registers) và bus trong CPU (Internal Bus). Bộ điều khiển có nhiệm vụ đọc, giải mã và điều khiển quá trình thực hiện lệnh. Bộ tính toán số học và logic chuyên thực hiện các phép toán số học như cộng trừ, nhân, chia, và các phép toán lôgic như và, hoặc, phủ định và các phép dịch, quay. Các thanh ghi là kho chứa lệnh và dữ liệu tạm thời cho CPU xử lý. Bus trong CPU có nhiệm vụ truyền dẫn các tín hiệu giữa các bộ phận trong CPU và kết nối với hệ thống bus ngoài. Hình 2 minh hoạ hai CPU của hãng Intel là 8086 ra đời năm 1978 và Core 2 Duo ra đời năm 2006.

1.2.2.2 Bộ nhớ trong

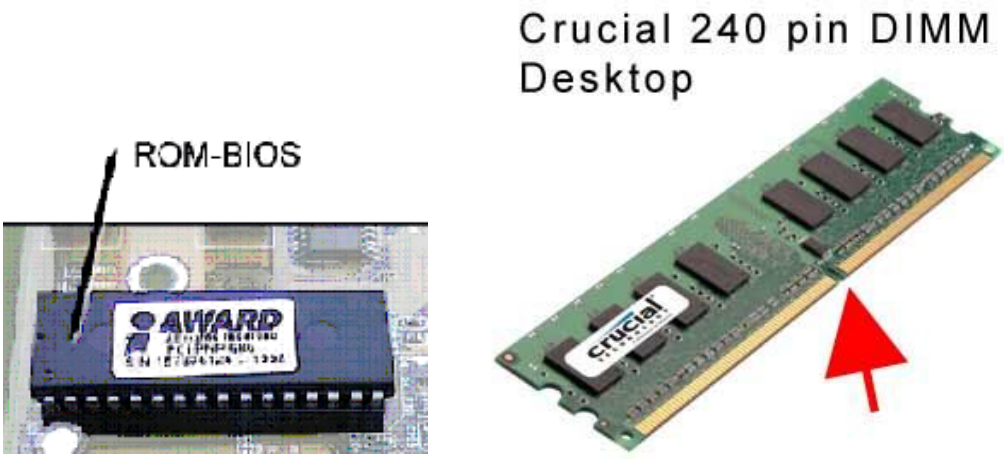
Bộ nhớ trong, còn gọi là bộ nhớ chính (Internal Memory hay Main Memory) là kho chứa lệnh và dữ liệu của hệ thống và của người dùng phục vụ CPU xử lý. Bộ nhớ trong thường là bộ nhớ bán dẫn, bao gồm hai loại: (1) Bộ nhớ chỉ đọc (Read Only Memory – ROM) và (2) Bộ nhớ truy cập ngẫu nhiên (Random Access Memory – RAM). ROM thường được sử dụng để lưu lệnh và dữ liệu của hệ thống. Thông tin trong ROM được nạp từ khi sản xuất và thường

chỉ có thể đọc ra trong quá trình sử dụng. Hơn nữa thông tin trong ROM luôn tồn tại kể cả khi không có nguồn điện nuôi.



Hình 2. CPU của hãng Intel: 8086 và Core 2 Duo

Khác với bộ nhớ ROM, bộ nhớ RAM thường được sử dụng để lưu lệnh và dữ liệu của cả hệ thống và của người dùng. RAM thường có dung lượng lớn hơn nhiều so với ROM. Tuy nhiên, thông tin trong RAM chỉ tồn tại khi có nguồn điện nuôi. Hình 3 minh hoạ vi mạch bộ nhớ ROM và các vi mạch nhớ RAM gắn trên một thanh nhớ RAM.



Hình 3 Bộ nhớ ROM và RAM

1.2.2.3 Các thiết bị vào ra

Các thiết bị vào ra (Input – Output devices), hay còn gọi là các thiết bị ngoại vi (Peripheral devices) đảm nhiệm việc nhập dữ liệu vào, điều khiển hệ thống và kết xuất dữ liệu ra. Có hai nhóm thiết bị ngoại vi: (1) Các thiết bị vào (Input devices) và (2) Các thiết bị ra (Output devices). Các thiết bị vào dùng để nhập dữ liệu vào và điều khiển hệ thống, gồm: bàn phím (keyboard), chuột (mouse), ổ đĩa (Disk Drives), máy quét ảnh (Scanners),... Các thiết bị ra dùng để xuất dữ liệu ra, gồm: màn hình (Screen), máy in (Printers), ổ đĩa (Disk Drives), máy vẽ (Plotters),...

1.2.2.4 Bus hệ thống

Bus hệ thống (System Bus) là một tập các đường dây kết nối CPU với các thành phần khác của máy tính. Bus hệ thống thường gồm ba bus con: Bus địa chỉ – Bus A (Address bus), Bus dữ liệu – Bus D (Data bus), Bus điều khiển - Bus C (Control bus). Bus địa chỉ có nhiệm vụ truyền tín hiệu địa chỉ từ CPU đến bộ nhớ và các thiết bị ngoại vi; Bus dữ liệu vận chuyển các tín hiệu dữ liệu theo hai chiều đi và đến CPU; Bus điều khiển truyền tín hiệu điều khiển từ CPU đến các thành phần khác, đồng thời truyền tín hiệu trạng thái của các thành phần khác đến CPU.

1.3 LỊCH SỬ PHÁT TRIỂN MÁY TÍNH

Lịch sử phát triển máy tính có thể được chia thành 5 thế hệ chính phục thuộc vào sự phát triển của mạch điện tử.

1.3.1 Thế hệ 1 (1944-1959)

Máy tính thế hệ 1 sử dụng đèn điện tử làm linh kiện chính và băng từ làm thiết bị vào ra. Mật độ tích hợp linh kiện vào khoảng 1000 linh kiện / foot^3 (1 foot = 30.48 cm). Đại diện tiêu biểu của thế hệ máy tính này là siêu máy tính ENIAC (Electronic Numerical Integrator and Computer), trị giá 500.000 USD.

1.3.2 Thế hệ 2 (1960-1964)

Máy tính thế hệ 2 sử dụng bóng bán dẫn (transitor) làm linh kiện chính. Mật độ tích hợp linh kiện vào khoảng 100.000 linh kiện / foot^3. Các đại diện tiêu biểu của thế hệ máy tính này là UNIVAC 1107, UNIVAC III, IBM 7070, 7080, 7090, 1400 series, 1600 series. Máy tính UNIVAC đầu tiên ra đời vào năm 1951, có giá khởi điểm là 159.000 USD. Một số phiên bản kết tiếp của UNIVAC có giá nằm trong khoảng 1.250.000 – 1.500.000 USD.

1.3.3 Thế hệ 3 (1964-1975)

Máy tính thế hệ 3 sử dụng mạch tích hợp (IC – Integrated Circuit) làm linh kiện chính. Mật độ tích hợp linh kiện vào khoảng 10.000.000 linh kiện / foot^3. Các đại diện tiêu biểu của thế hệ máy tính này là UNIVAC 9000 series, IBM System/360, System 3, System 7.

1.3.4 Thế hệ 4 (1975-1989)

Máy tính thế hệ 4 sử dụng mạch tích hợp loại lớn (LSI – Large Scale Integrated Circuit) làm linh kiện chính. Mật độ tích hợp linh kiện vào khoảng 1 tỷ linh kiện / foot^3. Các đại diện tiêu biểu của thế hệ máy tính này là IBM System 3090, IBM RISC 6000, IBM RT, Cray 2 XMP.

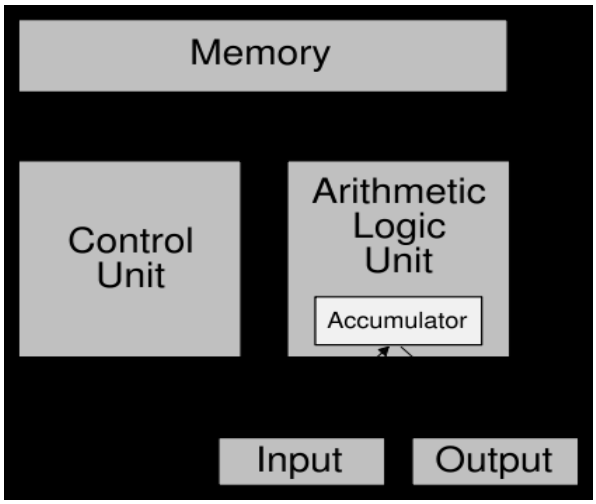
1.3.5 Thế hệ 5 (1990 - nay)

Máy tính thế hệ 5 sử dụng mạch tích hợp loại siêu lớn (VLSI – Very Large Scale Integrated Circuit) làm linh kiện chính. Mật độ tích hợp linh kiện rất cao với các công nghệ 0.180µm – 0.045µm (kích thước transitor giảm xuống còn 180 – 45 nano mét). Các đại diện tiêu biểu của thế hệ máy tính này là máy tính sử dụng CPU Intel Pentium II, III, IV, M, D, Core Duo, Core 2 Duo, Core Quad,... Máy tính thế hệ 5 đạt hiệu năng xử lý rất cao, cung cấp nhiều tính năng tiến tiến, như hỗ trợ xử lý song song, tích hợp khả năng xử lý âm thanh và hình ảnh.

1.4 KIẾN TRÚC MÁY TÍNH VON-NEUMANN

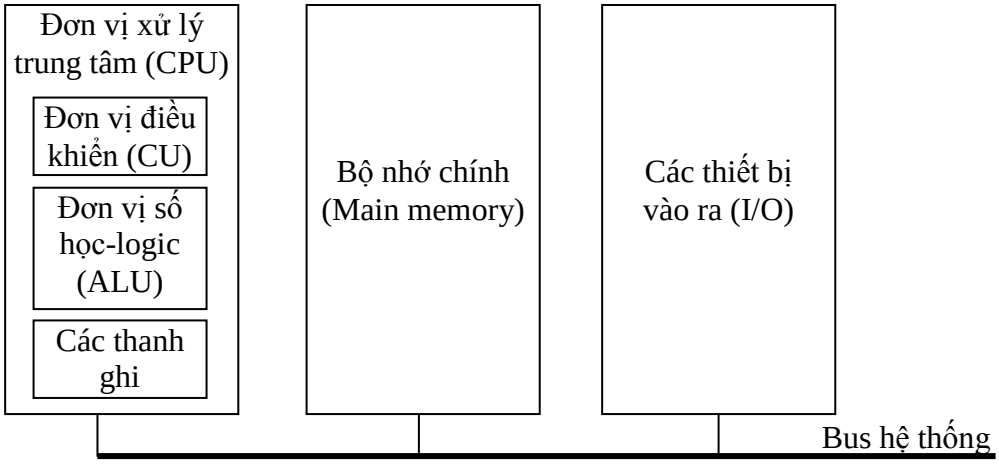
1.4.1 Sơ đồ kiến trúc máy tính von-Neumann

Kiến trúc máy tính von-Neumann được nhà toán học John von-Neumann đưa ra vào năm 1945 trong một báo cáo về máy tính EDVAC như minh hoạ trên Hình 4 - Kiến trúc máy tính von-Neumann nguyên thuỷ.



Hình 4 Kiến trúc máy tính von-Neumann nguyên thuỷ

Các máy tính hiện đại ngày nay sử dụng kiến trúc máy tính von-Neumann cải tiến – còn gọi là kiến trúc máy tính von-Neumann hiện đại, như minh hoạ trên Hình 5.



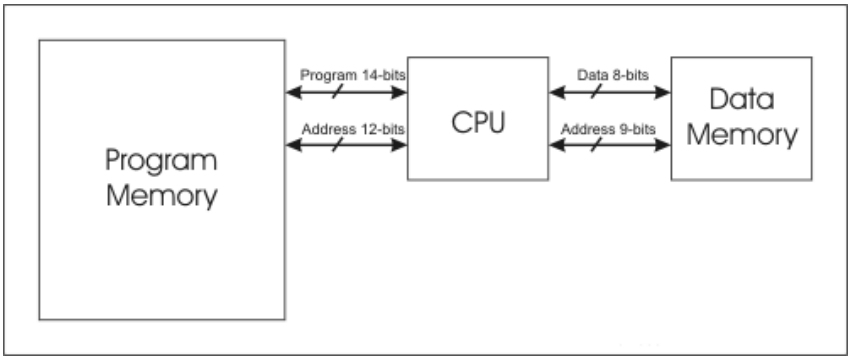
Hình 5 Kiến trúc máy tính von-Neumann hiện đại

1.4.2 Các đặc điểm của kiến trúc von-Neumann

Kiến trúc von-Neumann dựa trên 3 khái niệm cơ sở: (1) Lệnh và dữ liệu được lưu trữ trong bộ nhớ đọc ghi chia sẻ - một bộ nhớ duy nhất được sử dụng để lưu trữ cả lệnh và dữ liệu, (2) Bộ nhớ được đánh địa chỉ theo vùng, không phụ thuộc vào nội dung nó lưu trữ và (3) Các lệnh của một chương trình được thực hiện tuần tự. Quá trình thực hiện lệnh được chia thành 3 giai đoạn (stages) chính: (1) CPU đọc (fetch) lệnh từ bộ nhớ, (2) CPU giải mã và thực hiện lệnh; nếu lệnh yêu cầu dữ liệu, CPU đọc dữ liệu từ bộ nhớ; và (3) CPU ghi kết quả thực hiện lệnh vào bộ nhớ (nếu có).

1.5 KIẾN TRÚC MÁY TÍNH HARVARD

Kiến trúc máy tính Harvard là một kiến trúc tiên tiến như minh hoạ trên Hình 6.



Hình 6 Kiến trúc máy tính Harvard

Kiến trúc máy tính Harvard chia bộ nhớ trong thành hai phần riêng rẽ: Bộ nhớ lưu chương trình (Program Memory) và Bộ nhớ lưu dữ liệu (Data Memory). Hai hệ thống bus riêng được sử dụng để kết nối CPU với bộ nhớ lưu chương trình và bộ nhớ lưu dữ liệu. Mỗi hệ thống bus đều có đầy đủ ba thành phần để truyền dẫn các tín hiệu địa chỉ, dữ liệu và điều khiển.

Máy tính dựa trên kiến trúc Harvard có khả năng đạt được tốc độ xử lý cao hơn máy tính dựa trên kiến trúc von-Neumann do kiến trúc Harvard hỗ trợ hai hệ thống bus độc lập với băng thông lớn hơn. Ngoài ra, nhờ có hai hệ thống bus độc lập, hệ thống nhớ trong kiến trúc Harvard hỗ trợ nhiều lệnh truy nhập bộ nhớ tại một thời điểm, giúp giảm xung đột truy nhập bộ nhớ, đặc biệt khi CPU sử dụng kỹ thuật đường ống (pipeline).

1.6 CÁC HỆ SỐ ĐẾM VÀ TỔ CHỨC DỮ LIỆU TRÊN MÁY TÍNH

1.6.1 Các hệ số đếm

Trong đời sống hàng ngày, hệ đếm thập phân (Decimal Numbering System) là hệ đếm thông dụng nhất. Tuy nhiên, trong hầu hết các hệ thống tính toán hệ đếm nhị phân (Binary Numbering System) lại được sử dụng để biểu diễn dữ liệu. Trong hệ đếm nhị phân, chỉ 2 chữ số 0 và 1 được sử dụng: 0 biểu diễn giá trị Sai (False) và 1 biểu diễn giá trị Đúng (True). Ngoài ra, hệ đếm thập lục phân (Hexadecimal Numbering System) cũng được sử dụng. Hệ thập lục phân sử dụng 16 chữ số: 0-9, A, B, C, D, E, F.

1.6.1.1 Hệ đếm thập phân

Hệ đếm thập phân là hệ đếm cơ số 10, sử dụng 10 chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Mỗi số trong hệ 10 có thể được biểu diễn thành một đa thức:

$$a_n a_{n-1}...a_1 = a_n*10^{n-1} a_{n-1}*10^{n-2}*...*a_1*10^0$$

Ví dụ:

$$123 = 1*10^2 + 2 * 10^1 + 3*10^0 = 100 + 20 + 3$$

$$123{,}456 = 1*10^2 + 2 * 10^1 + 3*10^0 + 4*10^{-1} + 5*10^{-2} + 6*10^{-3}$$

$$= 100 + 20 + 3 + 0.4 + 0.05 + 0.006$$

1.6.1.2 Hệ đếm nhị phân

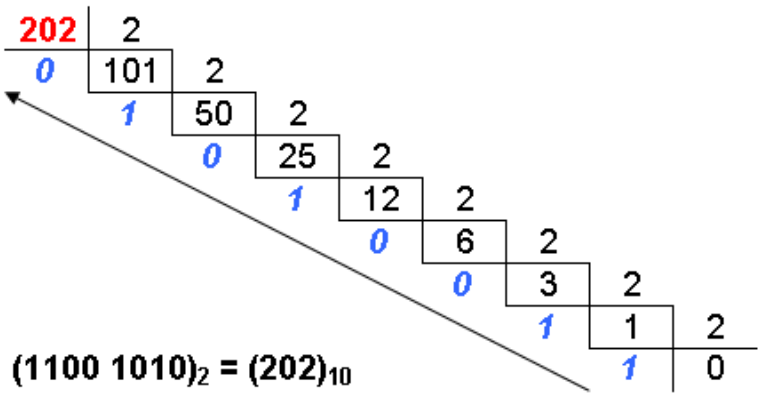
Hệ đếm nhị phân là hệ đếm cơ số 2, chỉ sử dụng 2 chữ số: 0 và 1. Mỗi số trong hệ 2 cũng có thể được biểu diễn thành 1 đa thức:

$$(a_n a_{n-1} ... a_1)_2 = a_n*2^{n-1} a_{n-1}*2^{n-2}*...*a_1*2^0$$

Ví dụ:

$$(11001010)_2 = 1*2^7 + 1*2^6 + 0*2^5 + 0*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0$$
$$= 128 + 64 + 8 + 2 = (202)_{10}$$

Việc chuyển đổi số hệ thập phân sang số hệ nhị phân có thể được thực hiện theo thuật toán đơn giản như minh hoạ trên Hình 7.



Hình 7 Chuyển đổi số hệ thập phân sang số hệ nhị phân

1.6.1.3 Hệ đếm thập lục phân

Hệ đếm thập lục phân là hệ đếm cơ số 16, sử dụng 16 chữ số: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Mỗi số trong hệ 16 được biểu diễn bởi 4 chữ số trong hệ nhị phân như minh hoạ trên Hình 8. Ưu điểm của hệ thập lục phân là số thập lục phân có thể chuyển đổi sang số hệ nhị phân và ngược lại một cách dễ dàng và cần ít chữ số hơn hệ nhị phân để biểu diễn cùng một đơn vị dữ liệu.

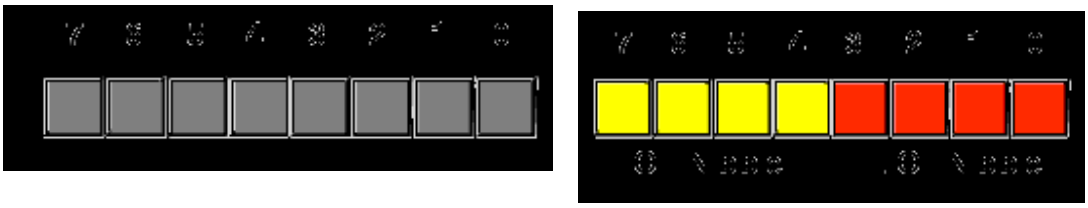
Hexa	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

Hình 8 Giá trị các số thập lục phân theo hệ thập phân và nhị phân

1.6.2 Tổ chức dữ liệu trên máy tính

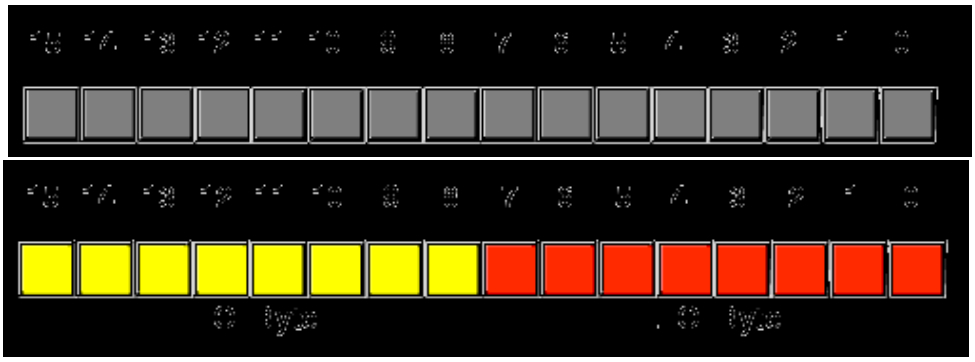
Dữ liệu trên máy tính được biểu diễn theo các đơn vị (unit). Các đơn vị biểu diễn dữ liệu cơ sở gồm: bit, nibble, byte, word và double-word. Bit là đơn vị dữ liệu nhỏ nhất: mỗi bit chỉ lưu được tối đa 2 giá trị: 0 hoặc 1, hay đúng hoặc sai. Nibble là đơn vị kế tiếp bit. Mỗi nibble là một nhóm 4 bit. Một nibble có thể lưu tối đa 16 giá trị, từ $(0000)_2$ đến $(1111)_2$, hoặc một chữ số thập lục phân.

Byte là đơn vị dữ liệu kế tiếp nibble. Một byte là một nhóm của 8 bits hoặc 2 nibbles. Một byte có thể lưu đến 256 giá trị, từ $(0000\ 0000)_2$ đến $(1111\ 1111)_2$, hoặc từ $(00)_{16}$ đến $(FF)_{16}$. Hình 9 minh hoạ đơn vị biểu diễn dữ liệu Byte.



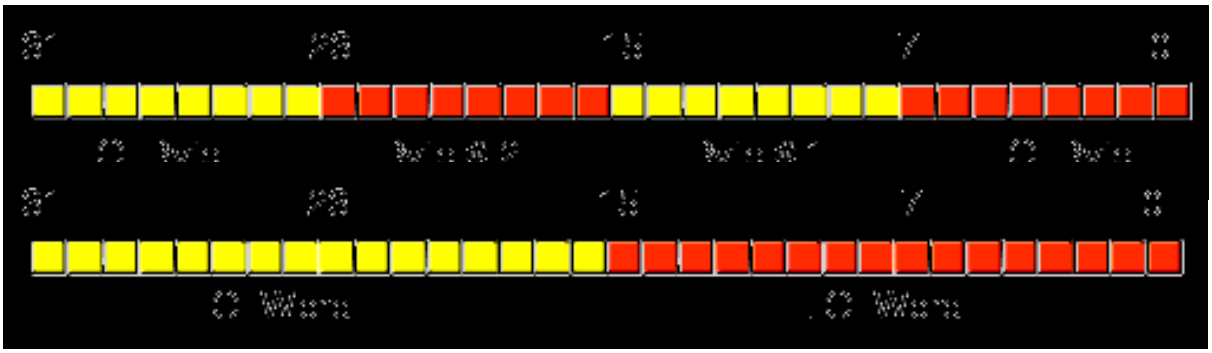
Hình 9 Đơn vị biểu diễn dữ liệu Byte

Word (từ) là đơn vị dữ liệu kế tiếp byte. Một word là một nhóm của 16 bits, hoặc 2 bytes. Một word có thể lưu đến 2^{16} (65536) giá trị, từ $(0000)_{16}$ đến $(FFFF)_{16}$. Hình 10 minh hoạ đơn vị biểu diễn dữ liệu word.



Hình 10 Đơn vị biểu diễn dữ liệu Word

Double words (từ kép) là đơn vị biểu diễn dữ liệu cơ sở lớn nhất. Một double word là một nhóm 32 bits, hoặc 4 bytes, hoặc 2 words. Một double word có thể lưu đến 2^{32} giá trị, từ $(0000\ 0000)_{16}$ đến $(FFFF\ FFFF)_{16}$. Hình 11 minh hoạ đơn vị biểu diễn dữ liệu double word.



Hình 11 Đơn vị biểu diễn dữ liệu Double word

1.6.3 Số có dấu và số không dấu

Trong các hệ thống tính toán, với cùng một số bit có thể biểu diễn các giá trị khác nhau nếu số được biểu diễn là có dấu hoặc không dấu. Để biểu diễn số có dấu, người ta sử dụng bit cao nhất (bên trái nhất) để biểu diễn dấu của số - gọi là bit dấu, chẳng hạn bít dấu có giá trị 0 là số dương và bít dấu có giá trị 1 là số âm. Với số không dấu, tất cả các bit được sử dụng để biểu diễn giá trị của số. Như vậy, miền giá trị có thể biểu diễn của một số gồm n bít như sau:

- Số có dấu: miền biểu diễn từ từ -2^{n-1} đến $+ 2^{n-1}$

- 8 bits: từ -128 đến +128
- 16 bits: từ -32768 đến +32768
- 32 bits: từ -2.147.483.648 đến +2.147.483.648

- Số không dấu: từ 0 đến 2^n
  - 8 bits: từ 0 đến 256
  - 16 bits: từ 0 đến 65536
  - 32 bits: từ 0 đến 4.294.967.296

1.6.4 Bảng mã ASCII

Bảng mã ASCII (American Standard Code for Information Interchange) là bảng mã các ký tự chuẩn tiếng Anh dùng cho trao đổi dữ liệu giữa các hệ thống tính toán. Bảng mã ASCII sử dụng 8 bít để biểu diễn 1 ký tự, cho phép định nghĩa tổng số 256 ký tự, đánh số từ 0 đến 255. 32 ký tự đầu tiên và ký tự số 127 là các ký tự điều khiển (không in ra được). Các ký tự từ số 32 đến 126 là các ký tự có thể in được (gồm cả dấu trắng). Các vị trí còn lại trong bảng (128-255) để dành cho sử dụng trong tương lai. Hình 12 và Hình 13 lần lượt là minh hoạ các ký tự điều khiển và các ký tự in được của bảng mã ASCII.

Binary	Oct	Dec	Hex	Abbr	PR[t 1]	CS[t 2]	CEC[t 3]	Description
000 0000	000	0	00	NUL	NUL	^@	\0	Null character
000 0001	001	1	01	SOH	SOH	^A		Start of Header
000 0010	002	2	02	STX	STX	^B		Start of Text
000 0011	003	3	03	ETX	ETX	^C		End of Text
000 0100	004	4	04	EOT	EOT	^D		End of Transmission
000 0101	005	5	05	ENQ	ENQ	^E		Enquiry
000 0110	006	6	06	ACK	ACK	^F		Acknowledgment
000 0111	007	7	07	BEL	BEL	^G	\a	Bell
000 1000	010	8	08	BS	BS	^H	\b	Backspace[t 4][t 5]
000 1001	011	9	09	HT	HT	^I	\t	Horizontal Tab
000 1010	012	10	0A	LF	LF	^J	\n	Line feed

Hình 12 Bảng mã ASCII - Một số ký tự điều khiển

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	sp
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l

Hình 13 Bảng mã ASCII - Các ký tự in được

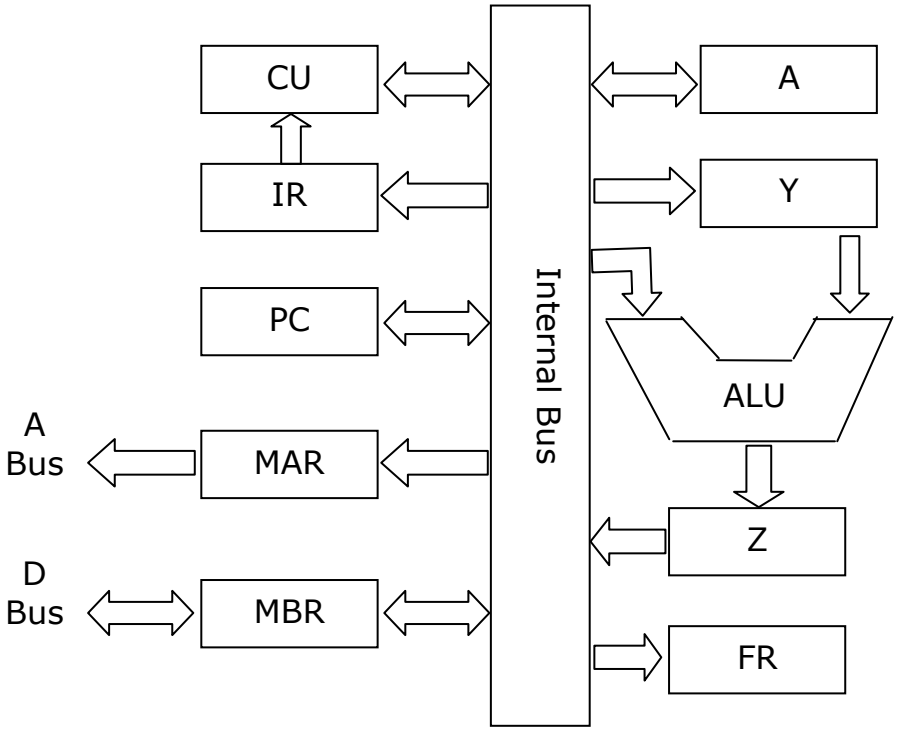
1.7 CÂU HỎI ÔN TẬP

1. Phân biệt khái niệm kiến trúc & tổ chức máy tính.
2. Nêu sơ đồ khối và mô tả chức năng từng khối của máy tính?
3. So sánh hai kiến trúc von-Neumann và Harvard.
4. Các hệ đếm 2, 10 và 16.
5. Các đơn vị lưu trữ dữ liệu trên máy tính.

CHƯƠNG 2 KHỐI XỬ LÝ TRUNG TÂM

2.1 SƠ ĐỒ KHỐI TỔNG QUÁT VÀ CHU TRÌNH XỬ LÝ LỆNH

2.1.1 Sơ đồ khối tổng quát của CPU



Hình 14 Sơ đồ khối tổng quát của CPU

Hình 14 trình bày sơ đồ khối nguyên lý tổng quát của CPU. Các thành phần của CPU theo sơ đồ này gồm:

- Bộ điều khiển (Control Unit – CU)
- Bộ tính toán số học và logic (Arithmetic and Logic Unit)
- Bus trong CPU (CPU Internal Bus)
- Các thanh ghi của CPU:
  - Thanh ghi tích luỹ A (Accummulator)
  - Bộ đếm chương trình PC (Program Counter)
  - Thanh ghi lệnh IR (Instruction Register)
  - Thanh ghi địa chỉ bộ nhớ MAR (Memory Address Register)
  - Thanh ghi đệm dữ liệu MBR (Memory Buffer Register)
  - Các thanh ghi tạm thời Y và Z
  - Thanh ghi cờ FR (Flag Register)

2.1.2 Chu trình xử lý lệnh

Như đã trình bày trong chương 1, nhiệm vụ chủ yếu của CPU là đọc lệnh từ bộ nhớ, giải mã và thực hiện lệnh của chương trình. Khoảng thời gian để CPU thực hiện xong một lệnh kể từ khi CPU cấp phát tín hiệu địa chỉ ô nhớ chứa lệnh đến khi nó hoàn tất việc thực hiện lệnh được gọi là *chu kỳ lệnh* (Insruction Cycle). Mỗi chu kỳ lệnh của CPU được mô tả theo các bước sau:

1. Khi một chương trình được kích hoạt, hệ điều hành (OS - Operating System) nạp mã chương trình vào bộ nhớ trong;
2. Địa chỉ của ô nhớ chứa lệnh đầu tiên của chương trình được nạp vào bộ đếm chương trình PC;
3. Địa chỉ ô nhớ chứa lệnh từ PC được chuyển đến bus địa chỉ thông qua thanh ghi MAR;
4. Bus địa chỉ chuyển địa chỉ ô nhớ đến đơn vị quản lý bộ nhớ (MMU - Memory Management Unit);
5. MMU chọn ra ô nhớ và thực hiện lệnh đọc nội dung ô nhớ;
6. Lệnh (chứa trong ô nhớ) được chuyển ra bus dữ liệu và tiếp theo được chuyển tiếp đến thanh ghi MBR;
7. MBR chuyển lệnh đến thanh ghi lệnh IR; IR chuyển lệnh vào bộ điều khiển CU;
8. CU giải mã lệnh và sinh các tín hiệu điều khiển cần thiết, yêu cầu các bộ phận chức năng của CPU, như ALU thực hiện lệnh;
9. Giá trị địa chỉ trong bộ đếm PC được tăng lên 1 đơn vị lệnh và nó trỏ đến địa chỉ của ô nhớ chứa lệnh tiếp theo;
10. Các bước từ 3-9 được lặp lại với tất cả các lệnh của chương trình.

2.2 CÁC THANH GHI

2.2.1 Giới thiệu về thanh ghi

Thanh ghi (registers) là các ô nhớ bên trong CPU, có nhiệm vụ lưu trữ tạm thời lệnh và dữ liệu cho CPU xử lý. Thanh ghi thường có kích thước nhỏ, nhưng tốc độ làm việc rất cao - bằng tốc độ CPU. Các CPU cũ (80x86) có khoảng 16-32 thanh ghi. Các CPU hiện đại (như Pentium 4 và Core Duo) có thể có đến hàng trăm thanh ghi. Kích thước thanh ghi phụ thuộc vào thiết kế CPU. Các kích thước thông dụng của thanh ghi là 8, 16, 32, 64, 128 và 256 bit. CPU Intel 8086 và 80286 có các thanh ghi 8 bit và 16 bit. CPU Intel 80386 và Pentium II có các thanh ghi 16 bit và 32 bit. Các CPU Pentium 4 và Core Duo có các thanh ghi 32 bit, 64 bit và 128 bit.

2.2.1.1 Thanh tích luỹ A

Thanh tích luỹ A (Accummulator) là một trong các thanh ghi quan trọng nhất của CPU. Thanh ghi A không những được sử dụng để lưu toán hạng vào mà còn dùng để chứa kết quả ra. Ngoài ra, thanh ghi A còn thường được dùng trong các lệnh trao đổi dữ liệu với các thiết bị vào ra. Kích thước của thanh ghi A bằng kích thước từ xử lý của CPU: 8 bit, 16 bit, 32 bit hoặc 64 bit.

Ví dụ về việc sử dụng thanh ghi A trong phép toán: x + y → s

- Nạp toán hạng x vào thanh ghi A
- Nạp toán hạng y vào thanh ghi tạm thời Y
- ALU thực hiện phép cộng A + Y và lưu kết quả vào thanh ghi Z
- Kết quả phép tính từ Z được chuyển về thanh ghi A.
- Kết quả trong thanh ghi A được lưu vào ô nhớ s.

2.2.1.2 Bộ đếm chương trình PC

Bộ đếm chương trình PC (Program Counter) hoặc con trỏ lệnh (IP – Instruction pointer) luôn chứa địa chỉ của ô nhớ chứa lệnh kế tiếp được thực hiện. Đặc biệt, PC chứa địa chỉ của ô nhớ chứa lệnh đầu tiên của chương trình khi chương trình được kích hoạt và được hệ điều hành nạp vào bộ nhớ. Khi CPU thực hiện xong một lệnh, địa chỉ của ô nhớ chứa lệnh tiếp theo được nạp vào PC. Kích thước của PC phụ thuộc vào thiết kế CPU. Các kích thước thông dụng của PC là 8 bit, 16 bit, 32 bit và 64 bit.

2.2.1.3 Thanh ghi lệnh IR

Thanh ghi lệnh IR (Instruction register) lưu lệnh đang thực hiện. IR nhận lệnh từ MBR và chuyển tiếp lệnh đến CU giải mã và thực hiện.

2.2.1.4 Các thanh ghi MAR và MBR

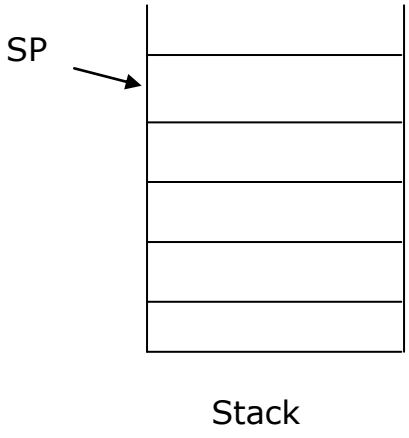
MAR là thanh ghi địa chỉ bộ nhớ (Memory address register) - giao diện giữa CPU và bus địa chỉ. MAR nhận địa chỉ ô nhớ chứa lệnh tiếp theo từ PC và chuyển tiếp ra bus địa chỉ.

MBR là thanh ghi đệm dữ liệu (Memory buffer register) - giao diện giữa CPU và bus địa chỉ. MBR nhận lệnh từ bus địa chỉ và chuyển tiếp lệnh đến IR thông qua bus trong CPU.

2.2.1.5 Các thanh ghi tạm thời

CPU thường sử dụng một số thanh ghi tạm thời để chứa toán hạng đầu vào và kết quả đầu ra, như các thanh ghi tạm thời X, Y và Z. Ngoài ra, các thanh ghi tạm thời còn tham gia trong việc hỗ trợ xử lý song song (thực hiện nhiều lệnh cùng một thời điểm) và hỗ trợ thực hiện lệnh theo cơ chế thực hiện tiên tiến kiểu không theo trật tự (OOO – Out Of Order execution).

2.2.1.6 Con trỏ ngăn xếp SP



Hình 15 Con trỏ ngăn xếp SP

Ngăn xếp (Stack) là bộ nhớ đặc biệt hoạt động theo nguyên lý vào sau ra trước (LIFO). Con trỏ ngăn xếp SP (Stack Pointer) là một thanh ghi luôn chứa địa chỉ đỉnh ngăn xếp. Có hai thao tác chính với ngăn xếp:

- Push - đẩy dữ liệu vào ngăn xếp:

  SP ← SP + 1 ; tăng địa chỉ đỉnh ngăn xếp

  {SP} ← Dữ liệu ; nạp dữ liệu vào ngăn xếp

- Pop - lấy dữ liệu ra khỏi ngăn xếp

  Thanh ghi ← {SP} ; chuyển dữ liệu từ đỉnh ngăn xếp vào thanh ghi

  SP ← SP – 1 ; giảm địa chỉ đỉnh ngăn xếp

2.2.1.7 Các thanh ghi tổng quát

Các thanh ghi tổng quát (General Purpose Registers) là các thanh ghi đa năng, có thể được sử dụng cho nhiều mục đích: để chứa toán hạng đầu vào hoặc chứa kết quả đầu ra. Chẳng hạn, CPU Intel 8086 có 4 thanh ghi tổng quát: AX - Thanh tích luỹ, BX - thanh ghi cơ sở, CX - thanh đếm và DX - thanh ghi dữ liệu.

2.2.1.8 Thanh ghi trạng thái FR

Thanh ghi trạng thái (SR - Status Register) hoặc thanh ghi cờ (FR – Flag Register) là một thanh ghi đặc biệt của CPU: mỗi bít của thanh ghi cờ lưu trạng thái của kết quả của phép tính ALU thực hiện. Có hai loại bít cờ: cờ trạng thái (CF, OF, AF, ZF, PF, SF) và cờ điều khiển (IF, TF, DF). Các bít cờ thường được sử dụng như là các điều kiện trong các lệnh rẽ nhánh để tạo logic chương trình. Kích thước của thanh ghi FR phụ thuộc thiết kế CPU.

Flag	ZF	SF	CF	AF	IF	OF	PF	1
Bit No	7	6	5	4	3	2	1	0

Hình 16 Các bit của thanh ghi cờ FR 8 bit

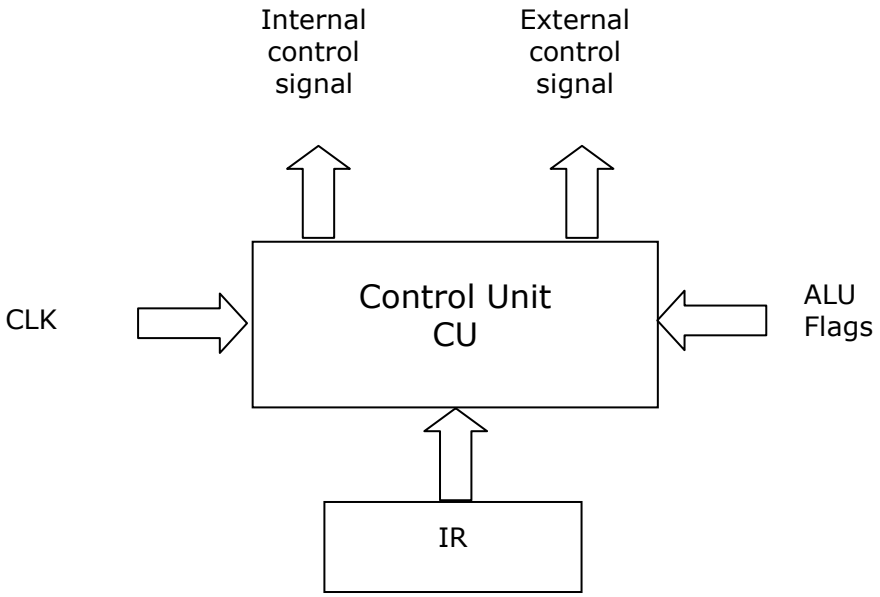
Hình 16 biểu diễn các bit của thanh ghi cờ FR. Ý nghĩa cụ thể của các bit như sau:

- ZF: Cờ Zero, ZF=1 nếu kết quả=0 và ZF=0 nếu kết quả<>0.
- SF: Cờ dấu, SF=1 nếu kết quả âm và SF=0 nếu kết quả dương.
- CF: Cờ nhớ, CF=1 nếu có nhớ/mượn, CF=0 trong trường hợp khác.
- AF: Cờ nhớ phụ, AF=1 nếu có nhớ/mượn ở nửa thấp của toán hạng.
- OF: Cờ tràn, OF=1 nếu xảy ra tràn, OF=0 trong trường hợp khác.
- PF: Cờ chẵn lẻ, PF=1 nếu tổng số bit 1 trong kết quả là lẻ và PF=0 nếu tổng số bit 1 trong kết quả là chẵn.
- IF: Cờ ngắt, IF=1: cho phép ngắt, IF=0: cấm ngắt.

2.3 KHỐI ĐIỀU KHIỂN

Khối điều khiển (Control Unit – CU) là một trong các khối quan trọng nhất của CPU. CU đảm nhiệm việc điều khiển toàn bộ các hoạt động của CPU theo xung nhịp đồng hồ. CU sử dụng nhịp đồng hồ để đồng bộ các đơn vị chức năng trong CPU và giữa CPU với các bộ phận

bên ngoài. Hình 17 minh hoạ phương thức làm việc của khối điều khiển CU. Khối điều khiển CU nhận ba tín hiệu đầu vào: (1) Lệnh từ thanh ghi lệnh IR, (2) Giá trị các cờ trạng thái của ALU và (3) Xung nhịp đồng hồ CLK và CU sản sinh hai nhóm tín hiệu đầu ra: (1) Nhóm tín hiệu điều khiển các bộ phận bên trong CPU (Internal control signal) và (2) Nhóm tín hiệu điều khiển các bộ phận bên ngoài CPU (External control signal).



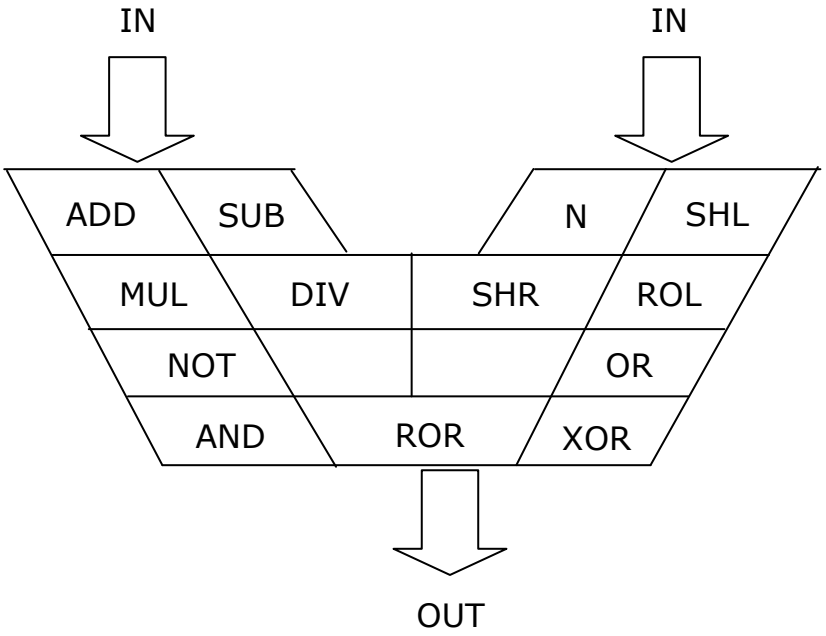
Hình 17 Khối điều khiển CU và các tín hiệu

2.4 KHỐI SỐ HỌC VÀ LOGIC

Khối số học và logic (Arithmetic and Logic Unit – ALU) đảm nhiệm chức năng tính toán trong CPU. ALU bao gồm một loạt các đơn vị chức năng con để thực hiện các phép toán số học trên số nguyên và logic:

- Bộ cộng (ADD), bộ trừ (SUB), bộ nhân (MUL), bộ chia (DIV), ...
- Các bộ dịch (SHIFT) và quay (ROTATE)
- Bộ phủ định (NOT), bộ và (AND), bộ hoặc (OR) và bộ hoặc loại trừ (XOR)

Hình 18 minh hoạ các khối con của ALU cũng như các cổng vào và cổng ra của ALU. Hai cổng vào IN nhận các toán hạng đầu vào từ các thanh ghi và một cổng OUT kết nối với bus trong để chuyển kết quả tính toán đến thanh ghi.



Hình 18 Bộ tính toán ALU

2.5 BUS TRONG CPU

Bus trong CPU (Internal bus) là kênh giao tiếp giữa các bộ phận bên trong CPU, cụ thể giữa bộ điều khiển CU với các thanh ghi và bộ tính toán ALU. Bus trong hỗ trợ kênh giao tiếp song công (full duplex) và cung cấp giao diện để kết nối với bus ngoài (bus hệ thống). So với bus ngoài, bus trong thường có băng thông lớn hơn và có tốc độ nhanh hơn.

2.6 CÂU HỎI ÔN TẬP

1. Nêu sơ đồ khối tổng quát và các thành phần chính của CPU?
2. Nêu chu trình xử lý lệnh của CPU?
3. Nêu vai trò và chức năng của các thanh ghi của CPU?
4. Nêu sơ đồ và chức năng của CU và ALU?

CHƯƠNG 3 TẬP LỆNH MÁY TÍNH

3.1 GIỚI THIỆU VỀ TẬP LỆNH MÁY TÍNH

3.1.1 Lệnh máy tính là gì?

Có thể nói, nếu coi phần mạch điện tử của CPU là "phần xác" thì tập lệnh (Instruction Set) chính là "phần hồn" của bộ não máy tính. Nhờ có tập lệnh, CPU có khả năng lập trình được để thực hiện các công việc hữu ích cho người dùng.

Vậy lệnh máy tính là gì? Có thể định nghĩa lệnh máy tính một cách đơn giản: Lệnh máy tính (Computer Instruction) là một từ nhị phân (binary word) được gán một nhiệm vụ cụ thể. Các lệnh của chương trình được lưu trong bộ nhớ và chúng lần lượt được CPU đọc, giải mã và thực hiện. Tập lệnh máy tính thường gồm nhiều lệnh có thể được chia thành một số nhóm theo chức năng: nhóm các lệnh vận chuyển dữ liệu (data movement), nhóm các lệnh tính toán (computational), nhóm các lệnh điều kiện và rẽ nhánh conditonal and branching) và một số lệnh khác.

Việc thực hiện lệnh có thể được chia thành các pha (phase) hay giai đoạn (stage). Mỗi lệnh có thể được thực hiện theo 4 giai đoạn: (1) Đọc lệnh (Instruction fetch - IF): lệnh được đọc từ bộ nhớ về CPU; (2) Giải mã (Instruction decode - ID): CPU giải mã lệnh; (3) Thực hiện lệnh (Instruction execution – EX): CPU thực hiện lệnh; và (4) Lưu kết quả (Write back - WB): kết quả thực hiện lệnh (nếu có) được lưu vào bộ nhớ.

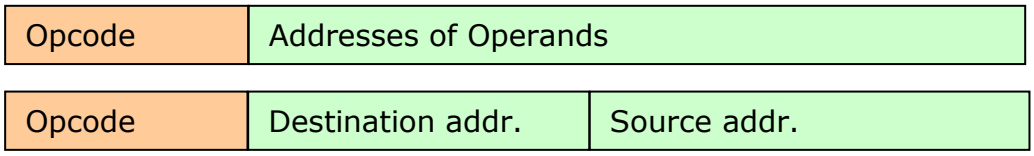
3.1.2 Chu kỳ thực hiện lệnh

Chu kỳ thực hiện lệnh (Instruction execution cycle) được định nghĩa là khoảng thời gian mà CPU thực hiện xong một lệnh. Một chu kỳ thực hiện lệnh có thể gồm một số giai đoạn thực hiện lệnh và một giai đoạn thực hiện lệnh có thể gồm một số chu kỳ máy. Một chu kỳ máy có thể gồm một số chu kỳ đồng hồ. Cụ thể hơn, chu kỳ thực hiện lệnh có thể gồm các thành phần sau:

- Chu kỳ đọc lệnh
- Chu kỳ đọc bộ nhớ (dữ liệu)
- Chu kỳ ghi bộ nhớ (dữ liệu)
- Chu kỳ đọc thiết bị ngoại vi
- Chu kỳ ghi thiết bị ngoại vi
- Chu kỳ bus rỗi.

3.2 DẠNG VÀ CÁC THÀNH PHẦN CỦA LỆNH

Dạng tổng quát của lệnh máy tính như minh hoạ trên Hình 19, gồm có 2 phần chính: (1) mã lệnh (opcode – operation code) và (2) địa chỉ của các toán hạng (Addresses of Operands). Mỗi lệnh có một mã lệnh riêng và được biểu biễn bằng một số bit. Chẳng hạn, mã lệnh của CPU Intel 8086 được biểu diễn bởi 6 bit. Mỗi lệnh có thể có một hoặc nhiều toán hạng và mỗi toán hạng là một địa chỉ. Tựu chung, có 5 dạng toán hạng của lệnh: 3 địa chỉ, 2 địa chỉ, 1 địa chỉ, 1,5 địa chỉ và 0 địa chỉ. Chi tiết về từng dạng toán hạng được trình bày trong mục 3.3.



Hình 19 Dạng và các thành phần của lệnh

3.3 CÁC DẠNG ĐỊA CHỈ / TOÁN HẠNG

3.3.1 Toán hạng dạng 3 địa chỉ

Dạng:

opcode addr1, addr2, addr3

Mỗi địa chỉ addr1, addr2, addr3 tham chiếu đến một ô nhớ hoặc một thanh ghi.

Ví dụ:

ADD R_1, R_2, R_3; $R_1 \leftarrow R_2 + R_3$; R_2 cộng với R_3, kết quả lưu vào R_1.

R_i là thanh ghi của CPU.

ADD A, B, C; M[A] ← M[B] + M[C];

Lấy nội dung của ô nhớ B cộng với nội dung của ô nhớ C, kết qua lưu vào ô nhớ A

A, B, C là địa chỉ các ô nhớ. M[..] quy ước là phép tham chiếu nội dung ô nhớ.

3.3.2 Toán hạng dạng 2 địa chỉ

Dạng:

opcode addr1, addr2

Mỗi địa chỉ addr1, addr2 tham chiếu đến một ô nhớ hoặc một thanh ghi.

Ví dụ:

ADD R_1, R_2; $R_1 \leftarrow R_1 + R_2$; R_1 cộng với R_2, kết quả lưu vào R_1.

R_i là thanh ghi của CPU.

ADD A, B; M[A] ← M[A] + M[B];

Lấy nội dung của ô nhớ A cộng với nội dung của ô nhớ B, kết qua lưu vào ô nhớ A

A, B là địa chỉ các ô nhớ.

3.3.3 Toán hạng dạng 1 địa chỉ

Dạng:

opcode addr2

Địa chỉ addr2 tham chiếu đến một ô nhớ hoặc một thanh ghi. Ngoài ra, thanh ghi tích luỹ R_{acc} được sử dụng và có vai trò như addr1 trong toán hạng dạng 2 địa chỉ.

Ví dụ:

ADD R_2; $R_{acc} \leftarrow R_{acc} + R_2$; R_{acc} cộng với R_2, kết quả lưu vào R_{acc}.

R_2 là thanh ghi của CPU.

ADD B; $R_{acc} \leftarrow R_{acc} + M[B]$;

Lấy nội dung của thanh ghi R_{acc} cộng với nội dung của ô nhớ B, kết qua lưu vào R_{acc}. A là địa chỉ một ô nhớ.

3.3.4 Toán hạng dạng 1,5 địa chỉ

Dạng:

opcode addr1, addr2

Một địa chỉ tham chiếu đến một ô nhớ và địa chỉ còn lại tham chiếu đến một thanh ghi. Dạng 1,5 địa chỉ là dạng toán hạng hỗn hợp giữa ô nhớ và thanh ghi.

Ví dụ:

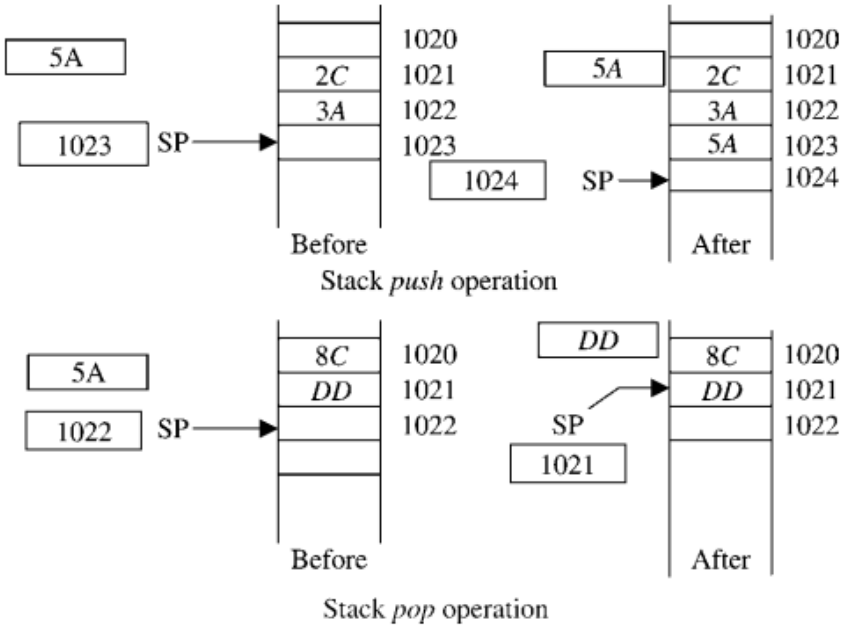
ADD R_1, A; $R_1 \leftarrow R_1 + M[A]$;

Lấy nội dung của R_1 cộng nội dung của ô nhớ A, kết quả lưu vào R_1.

R_1 là thanh ghi của CPU và A là địa chỉ ô nhớ.

3.3.5 Toán hạng dạng 0 địa chỉ

Toán hạng 0 địa chỉ thường được sử dụng trong các lệnh thao tác với ngăn xếp: PUSH và POP như minh hoạ trên Hình 20.



Hình 20 Thao tác PUSH và POP với ngăn xếp

3.4 CÁC CHẾ ĐỘ ĐỊA CHỈ

3.4.1 Giới thiệu về chế độ địa chỉ

Chế độ địa chỉ (Addressing modes) là phương thức hoặc cách thức CPU tổ chức các toán hạng của lệnh. Chế độ địa chỉ cho phép CPU kiểm tra dạng lệnh và tìm các toán hạng của lệnh. Số lượng các chế độ địa chỉ phụ thuộc vào thiết kế của CPU. Sau đây là một số chế độ địa chỉ thông dụng:

1. Tức thì (Immediate)

2. Trực tiếp (Direct)
3. Gián tiếp (indirect)
4. Chỉ số (Indexed)
5. Tương đối (Relative)

Mô tả chi tiết từng chế độ địa chỉ được thực hiện trong mục 3.4.2. Các ví dụ minh hoạ các chế độ địa chỉ sử dụng lệnh LOAD (nạp) với dạng sau:

LOAD <toán hạng đích> <toán hạng gốc>

Ý nghĩa: Nạp giá trị của <toán hạng gốc> vào <toán hạng đích>

Hay: <toán hạng đích> ← <toán hạng gốc>

3.4.2 Các chế độ địa chỉ

3.4.2.1 Chế độ địa chỉ tức thì (Immediate)

Trong chế độ địa chỉ tức thì, giá trị hằng của toán hạng nguồn (source operand) được đặt nằm ngay sau mã lệnh, còn toán hạng đích có thể là 1 thanh ghi hoặc 1 địa chỉ ô nhớ.

Ví dụ:

LOAD R1, #1000; R1 ← 1000 ; Nạp giá trị 1000 vào thanh ghi R1.

LOAD B, #100; M[B] ← 100 ; Nạp giá trị 100 vào ô nhớ B.

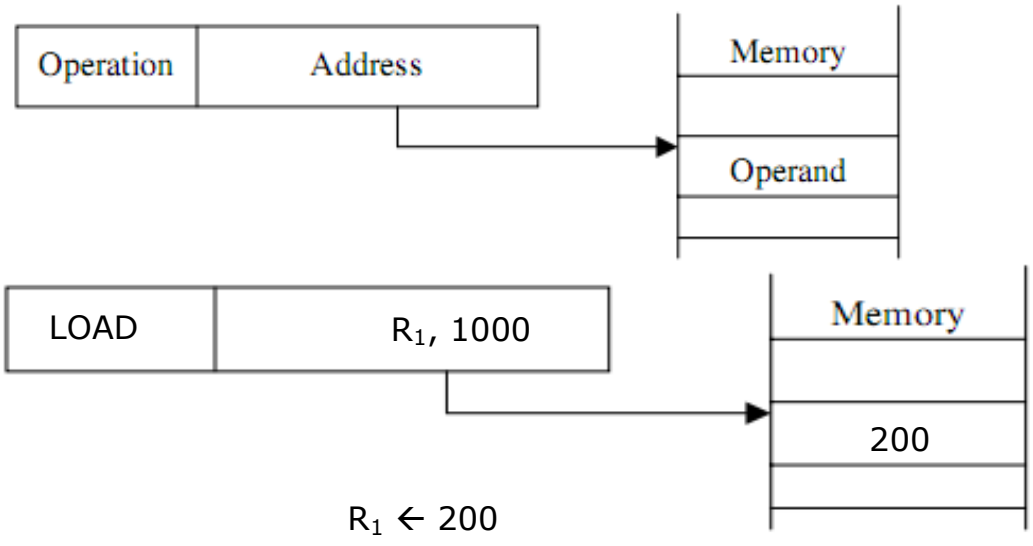
3.4.2.2 Chế độ địa chỉ trực tiếp (Direct)

Khác với chế độ địa chỉ tức thì, chế độ địa chỉ trực tiếp sử dụng một hằng để biểu diễn địa chỉ một ô nhớ làm một toán hạng. Toán hạng còn lại có thể là 1 thanh ghi hoặc 1 địa chỉ ô nhớ.

Ví dụ:

LOAD R1, 1000; R1 ← M[1000]

Nạp nội dung ô nhớ có địa chỉ 1000 vào thanh ghi R1. Hình 21 minh hoạ việc tham chiếu trong chế độ địa chỉ trực tiếp ở ví dụ trên. Địa chỉ 1000 trỏ đến ô nhớ chứa giá trị 200 và giá trị này được nạp vào thanh ghi R_1.



Hình 21 Tham chiếu với chế độ địa chỉ trực tiếp

3.4.2.3 Chế độ địa chỉ gián tiếp (Indirect)

Trong chế độ địa chỉ gián tiếp, một thanh ghi hoặc một ô nhớ được sử dụng để lưu địa chỉ một ô nhớ làm một toán hạng. Toán hạng còn lại có thể là một hằng, một thanh ghi hoặc địa chỉ một ô nhớ. Nếu thanh ghi được sử dụng để lưu địa chỉ ô nhớ ta có chế độ địa chỉ gián tiếp qua thanh ghi (register indirect); ngược lại nếu ô nhớ được dụng để lưu địa chỉ ô nhớ khác ta có chế độ địa chỉ gián tiếp qua ô nhớ (memory indirect).

Ví dụ:

Gián tiếp qua thanh ghi:

LOAD R_j, (R_i); $R_j \leftarrow M[R_i]$

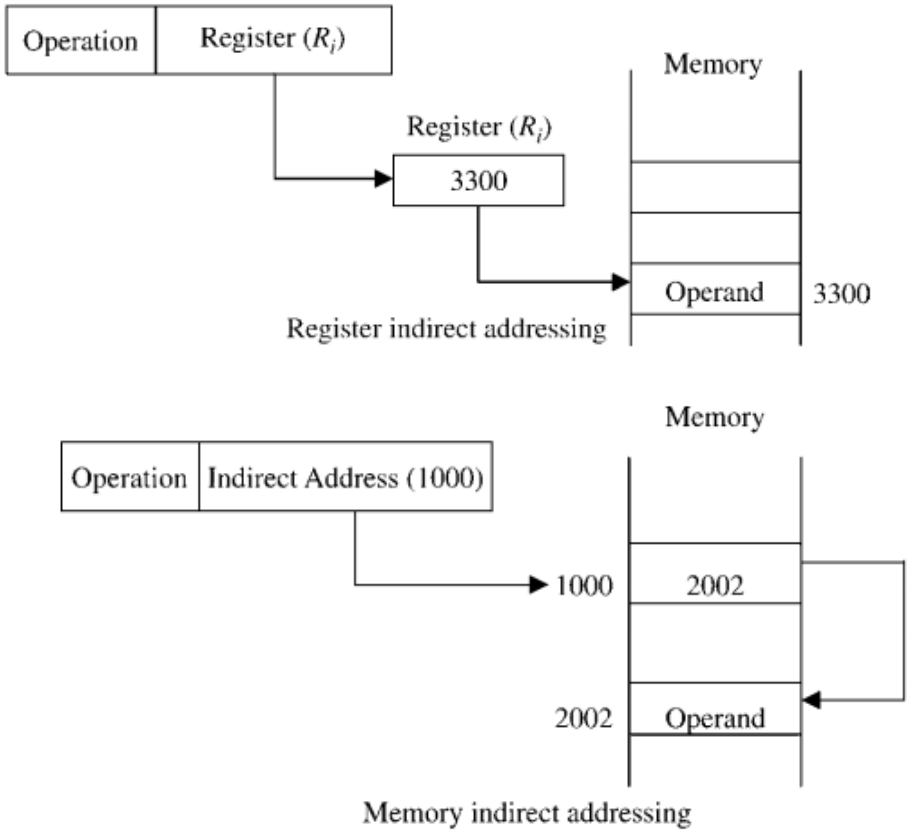
Nạp nội dung ô nhớ có địa chỉ lưu trong thanh ghi R_i vào thanh ghi R_j.

Gián tiếp qua ô nhớ:

LOAD R_i, (1000); $R_i \leftarrow M[M[1000]]$

Nạp nội dung ô nhớ có địa chỉ lưu trong ô nhớ 1000 vào thanh ghi R_i.

Hình 22 minh hoạ việc tham chiếu trong chế độ địa chỉ gián tiếp qua thanh ghi và gián tiếp qua ô nhớ. Có thể thấy rằng, chế độ địa chỉ gián tiếp qua thanh ghi chỉ yêu cầu một tham chiếu bộ nhớ cho một truy nhập, còn chế độ địa chỉ gián tiếp qua ô nhớ phải cần tới hai tham chiếu bộ nhớ cho một truy nhập.



Hình 22 Tham chiếu trong chế độ địa chỉ gián tiếp

3.4.2.4 Chế độ địa chỉ chỉ số (Indexed)

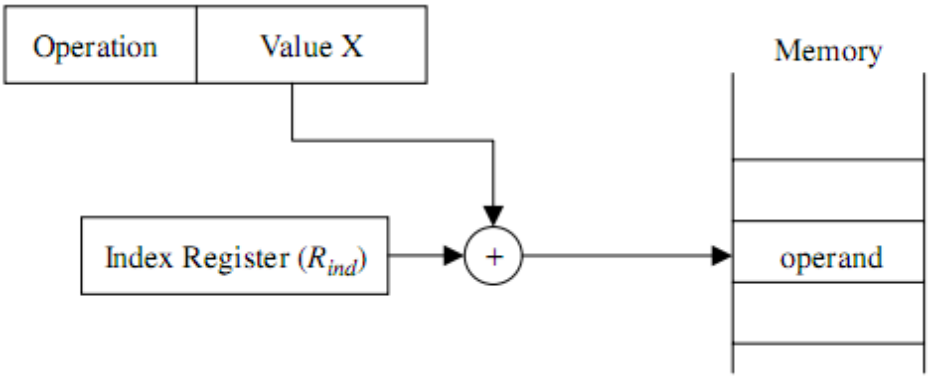
Trong chế độ địa chỉ chỉ số, địa chỉ của 1 toán hạng được tạo thành bởi phép cộng giữa 1 hằng và thanh ghi chỉ số (index register). Toán hạng còn lại có thể là một hằng, một thanh ghi hoặc địa chỉ một ô nhớ.

Ví dụ:

LOAD R_i, X(R_{ind}); $R_i \leftarrow M[X+R_{ind}]$

X là một hằng và R_{ind} là thanh ghi chỉ số.

Hình 23 minh hoạ phép tham chiếu trong chế độ địa chỉ chỉ số.



Hình 23 Tham chiếu trong chế độ địa chỉ chỉ số

3.4.2.5 Chế độ địa chỉ tương đối (Relative)

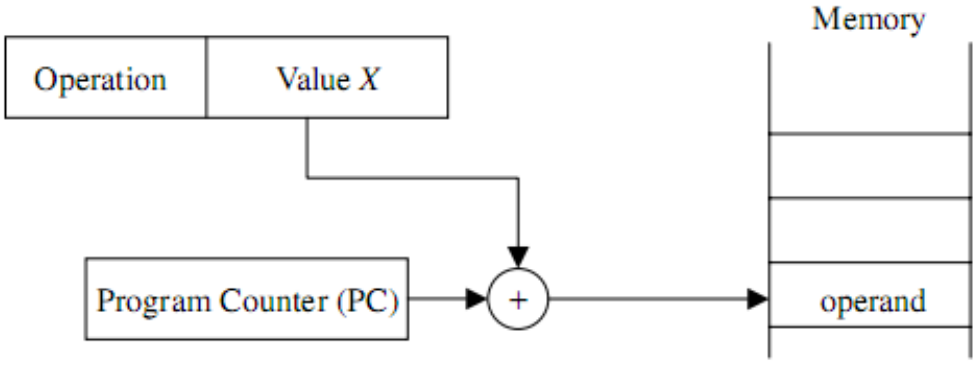
Trong chế độ địa chỉ tương đối, địa chỉ của 1 toán hạng được tạo thành bởi phép cộng giữa 1 hằng và bộ đếm chương trình PC (program counter). Toán hạng còn lại có thể là một hằng, một thanh ghi hoặc địa chỉ một ô nhớ.

Ví dụ:

LOAD R_i, X(PC); $R_i \leftarrow M[X+PC]$

X là một hằng và PC là bộ đếm chương trình.

Hình 24 minh hoạ phép tham chiếu trong chế độ địa chỉ chỉ số.



Hình 24 Tham chiếu trong chế độ địa chỉ tương đối

3.5 MỘT SỐ DẠNG LỆNH THÔNG DỤNG

Phụ thuộc thiết kế CPU, tập lệnh của CPU có thể có số lượng các lệnh rất khác nhau, từ vài chục lệnh đến vài trăm lệnh. Tuy nhiên, một hầu hết các tập lệnh máy tính thường bao gồm các nhóm lệnh cơ sở sau: (1) Các lệnh vận chuyển dữ liệu (Data Movement Instructions), (2) Các lệnh toán học và logic (Arithmetic and Logical Instructions), (3) Các lệnh điều khiển chương trình (Control/Sequencing Instructions) và (4) Các lệnh vào ra (Input/Output Instructions). Phần tiếp theo của mục này trình bày một số lệnh thông dụng thuộc các nhóm lệnh kể trên.

3.5.1 Các lệnh vận chuyển dữ liệu

Các lệnh vận chuyển dữ liệu vận chuyển dữ liệu giữa các bộ phận của máy tính. Cụ thể, vận chuyển dữ liệu giữa các thanh ghi của CPU, nạp dữ liệu từ các ô nhớ về các thanh ghi của CPU và ngược lại ghi dữ liệu từ các thanh ghi ra các ô nhớ. Ngoài ra, dữ liệu cũng có thể được vận chuyển giữa các ô nhớ trong bộ nhớ trong.

Ví dụ:

Vận chuyển dữ liệu giữa các thanh ghi của CPU:

MOVE R_i, R_j; $R_i \leftarrow R_j$

Chuyển (sao chép) nội dung của thanh ghi R_j sang thanh ghi R_i.

Vận chuyển dữ liệu giữa 1 thanh ghi của CPU và một ô nhớ:

MOVE 1000, R_j; M[1000] $\leftarrow R_j$

Lưu nội dung của thanh ghi R_j vào ô nhớ có địa chỉ 1000.

Vận chuyển dữ liệu giữa các ô nhớ:

MOVE 1000, (R_j); M[1000] $\leftarrow$ M[R_j]

Chuyển (sao chép) nội dung của ô nhớ có địa chỉ chứa trong thanh ghi R_j sang số nhớ có địa chỉ 1000.

Một số lệnh vận chuyển dữ liệu thông dụng

Tên lệnh	**Ý nghĩa**
MOVE	Chuyển dữ liệu giữa thanh ghi – thanh ghi, ô nhớ - thanh ghi và ô nhớ - ô nhớ.
LOAD	Nạp nội dung 1 ô nhớ vào 1 thanh ghi.
STORE	Lưu nội dung 1 thanh ghi ra 1 ô nhớ.
PUSH	Đẩy dữ liệu vào ngăn xếp.
POP	Lấy dữ liệu ra khỏi ngăn xếp.

3.5.2 Các lệnh toán học và logic

Các lệnh tính toán số học và logic được sử dụng để thực hiện các thao tác tính toán trên nội dung các thanh ghi và / hoặc nội dung các ô nhớ. Các lệnh tính toán hỗ trợ hầu hết các phép toán số học thông dụng như cộng, trừ, nhân, chia các số nguyên và các phép toán logic, như phủ định, và, hoặc, hoặc loại trừ.

Ví dụ:

Lệnh cộng:

ADD R_1, R_2, R_3; $R_1 \leftarrow R_2 + R_3$

Cộng nội dung 2 thanh ghi R_2 và R_3, kết quả lưu vào thanh ghi R_1.

ADD A, B, C; M[A] ← M[B] + M[C]

Cộng nội dung 2 ô nhớ B và C, kết quả lưu vào ô nhớ A.

Lệnh trừ:

SUBSTRACT R_1, R_2, R_3; $R_1 \leftarrow R_2 - R_3$

Lấy nội dung thanh ghi R_2 trừ đi nội dung thanh ghi R_3, kết quả lưu vào thanh ghi R_1.

Lệnh logic:

NOT R_1; $R_1 \leftarrow !(R_1)$

Lấy giá trị đảo (phủ định) của nội dung thanh ghi R_1.

AND R_1, R_2; $R_1 \leftarrow R_1 \otimes R_2$

Nhân bit nội dung 2 thanh ghi R_1 và R_2, kết quả lưu vào R_1.

Một số lệnh tính toán và logic thông dụng

Tên lệnh	**Ý nghĩa**
ADD	Cộng các toán hạng
SUBSTRACT	Trừ các toán hạng
MULTIPLY	Nhân các toán hạng
DIVIDE	Chia các toán hạng
INCREMENT	Tăng một đơn vị
DECREMENT	Giảm một đơn vị
NOT	Phủ định bit
AND	Phép và (nhân) bit
OR	Phép hoặc (cộng) bit
XOR	Phép hoặc loại trừ bit
COMPARE	So sánh 2 toán hạng
SHIFT	Phép dịch bit (dịch trái, dịch phải)
ROTATE	Phép quay bit (quay trái, quay phải)

3.5.3 Các lệnh điều khiển chương trình

Các lệnh điều khiển chương trình được sử dụng để thay đổi trật tự thực hiện các lệnh khác trong chương trình hay làm thay đổi logic chương trình. Đây là nhóm lệnh gây ra các *rẽ nhánh* (branching), hoặc *nhảy* (jumping) làm cho quá trình thực hiện chương trình phức tạp hơn. Một trong các đặc tính của các lệnh này là chúng làm thay đổi nội dung của bộ đếm chương trình PC – nơi chứa địa chỉ ô nhớ chứa lệnh tiếp theo được thực hiện, có nghĩa là yêu

cầu CPU thực hiện chương trình từ một vị trí mới thay vì thực hiện lệnh kế tiếp lệnh đang thực hiện. Các lệnh điều khiển chương trình sử dụng các cờ của ALU (lưu trong thanh ghi cờ FR) để xác định điều kiện rẽ nhánh hoặc nhảy. Có thể chia các lệnh điều khiển chương trình thành 3 loại chính sau:

- Các lệnh nhảy / rẽ nhánh có điều kiện (CONDITIONAL BRANCHING/ CONDITIONAL JUMP);
- Các lệnh nhảy/ rẽ nhánh không điều kiện (UNCONDITIONAL BRANCHING / JUMP);
- Các lệnh gọi thực hiện (CALL) và trở về (RETURN) từ chương trình con.

Ví dụ: Cộng nội dung 100 ô nhớ cạnh nhau, bắt đầu từ địa chỉ 1000. Kết quả lưu vào R_0.

```
        LOAD R1, #100;      R1 ← 100
        LOAD R2, #1000;     R2 ← 1000
        LOAD R0, #0;        R0 ← 0
Loop:   ADD R0, (R2);       R0 ← R0 + M[R2]
        INCREMENT R2;       R2 ← R2 + 1
        DECREMENT R1;       R1 ← R1 – 1
        BRANCH-IF-GREATER-THAN  Loop;
        ; Quay lại thực hiện lệnh sau nhãn Loop nếu R1 còn lớn hơn 0.
```

Một số lệnh điều khiển chương trình thông dụng

Tên lệnh	**Ý nghĩa**
BRANCH-IF-CONDITION	Chuyển đến thực hiện lệnh ở địa chỉ mới nếu điều kiện là đúng.
JUMP	Chuyển đến thực hiện lệnh ở địa chỉ mới.
CALL	Chuyển đến thực hiện chương trình con.
RETURN	Trở về (từ chương trình con) thực hiện tiếp chương trình gọi.

3.5.4 Các lệnh vào ra

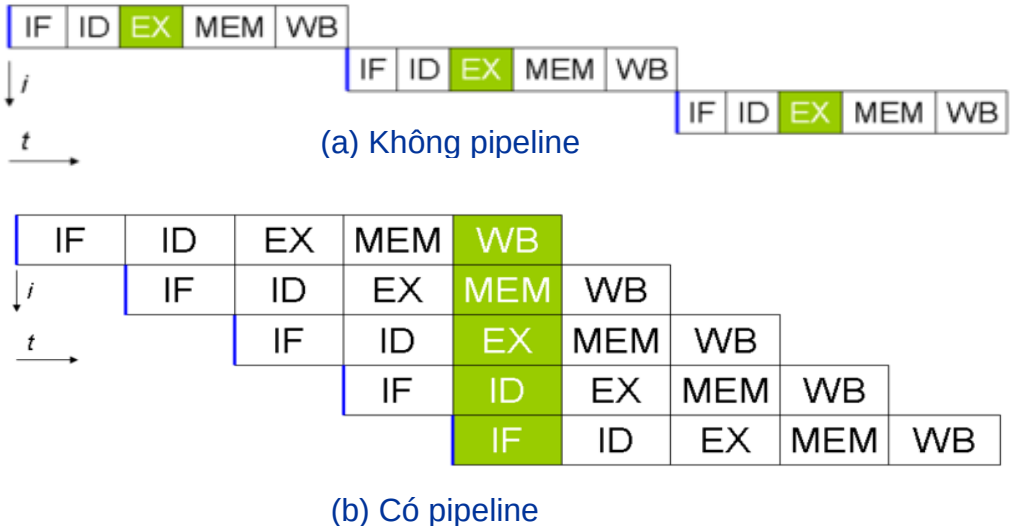
Các lệnh vào ra (I/O instructions) được sử dụng để vận chuyển dữ liệu giữa máy tính và các thiết bị ngoại vi. Các thiết bị ngoại vi giao tiếp với máy tính thông qua các cổng vào ra chuyên dụng (IO dedicated ports). Mỗi cổng vào ra được gán một địa chỉ riêng biệt. Có hai lệnh vào ra cơ bản:

- INPUT: sử dụng để chuyển dữ liệu từ thiết bị vào (input devices) đến CPU;
- OUTPUT: sử dụng để chuyển dữ liệu từ CPU đến thiết bị ra (output devices).

3.6 GIỚI THIỆU CƠ CHẾ ỐNG LỆNH (PIPELINE)

3.6.1 Giới thiệu cơ chế ống lệnh

Cơ chế ống lệnh (pipeline) hay còn gọi là cơ chế thực hiện xen kẽ các lệnh của chương trình là một phương pháp thực hiện lệnh tiên tiến, cho phép đồng thời thực hiện nhiều lệnh, giảm thời gian trung bình thực hiện mỗi lệnh và như vậy tăng được hiệu năng xử lý lệnh của CPU. Việc thực hiện lệnh được chia thành một số giai đoạn và mỗi giai đoạn được thực thi bởi một đơn vị chức năng khác nhau của CPU. Nhờ vậy CPU có thể tận dụng tối đa năng lực xử lý của các đơn vị chức năng của mình, giảm thời gian chờ cho từng đơn vị chức năng.



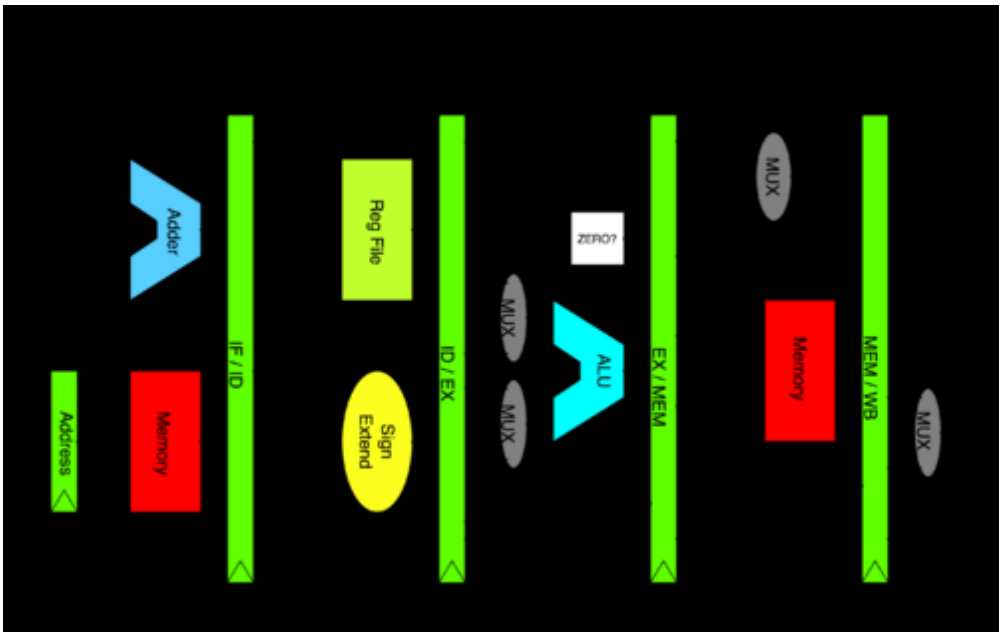
Hình 25 Thực hiện lệnh (a) không pipeline và (b) có pipeline

Hình 25 minh hoạ cơ chế thực hiện lệnh (a) không pipeline và (b) có pipeline. Trong đó, việc thực hiện lệnh được chia thành 5 giai đoạn:

- Instruction Fetch - IF: Đọc lệnh từ bộ nhớ (hoặc cache);
- Instruction Decode - ID: giải mã lệnh và đọc các toán hạng;
- Execute - EX: thực hiện lệnh; nếu là lệnh truy nhập bộ nhớ: tính toán địa chỉ bộ nhớ;
- Memory Access - MEM: Đọc/ghi bộ nhớ; no-op nếu không truy nhập bộ nhớ; no-op là giai đoạn chờ, tiêu tốn thời gian CPU, nhưng không thực hiện thao tác có nghĩa;
- Write Back - WB: Ghi kết quả vào các thanh ghi.

Có thể thấy, với cơ chế thực hiện không pipeline, tại mỗi thời điểm chỉ có một lệnh được thực hiện và chỉ có một đơn vị chức năng của CPU làm việc, các đơn vị chức năng khác trong trạng thái chờ. Ngược lại, với cơ chế thực hiện có pipeline, có nhiều lệnh đồng thời được thực hiện gối nhau trong CPU và hầu hết các đơn vị chức năng của CPU liên tục tham gia vào quá trình xử lý lệnh. Số lượng lệnh được xử lý đồng thời đúng bằng số giai đoạn thực hiện lệnh. Với 5 giai đoạn thực hiện lệnh, để xử lý 5 lệnh, CPU cần 9 nhịp đồng hồ với cơ chế thực hiện có pipeline, trong khi CPU cần đến 25 nhịp đồng hồ để thực hiện 5 lệnh với cơ chế thực hiện không pipeline. Hình 26 minh hoạ việc các đơn vị chức năng của CPU phối hợp thực hiện lệnh trong cơ chế pipeline.

Việc lựa chọn số giai đoạn thực hiện lệnh sao cho phù hợp là một trong các vấn đề quan trọng của cơ chế ống lệnh. Về mặt lý thuyết, thời gian thực hiện lệnh trung bình sẽ giảm khi tăng số giai đoạn thực hiện lệnh. Cho đến hiện nay, không có câu trả lời chính xác về số giai đoạn thực hiện lệnh tối ưu mà nó phụ thuộc nhiều vào thiết kế của CPU. Với các CPU cũ (họ Intel 80x86 và tương đương) số giai đoạn là 3 đến 5. Với các CPU Intel Pentium III và Pentium M, Core Duo, Core 2 Duo số giai đoạn là khoảng 10 đến 15. Riêng họ Intel Pentium IV có số giai đoạn vào khoảng 20 và cá biệt phiên bản Intel Pentium IV Prescott chia việc thực hiện lệnh thành 31 giai đoạn.



Hình 26 Thực hiện lệnh theo cơ chế pipeline với các đơn vị chức năng của CPU

3.6.2 Các vấn đề của cơ chế ống lệnh và hướng giải quyết

Như đã trình bày, cơ chế ống lệnh giúp giảm thời gian trung bình thực hiện từng lệnh và tăng đáng kể hiệu suất xử lý lệnh của CPU. Tuy nhiên, cơ chế ống lệnh cũng gặp phải một số vấn đề làm giảm hiệu suất thực hiện lệnh. Tựu chung, có ba vấn đề thường gặp với cơ chế ống lệnh: (1) Vấn đề xung đột tài nguyên (resource conflicts), (2) Vấn đề tranh chấp dữ liệu (Data hazards) và (3) Vấn đề nảy sinh do các lệnh rẽ nhánh (Branch instructions). Trong phạm vi của bài giảng này, hướng giải quyết các vấn đề của cơ chế ống lệnh chỉ dừng ở mức giới thiệu phương pháp.

3.6.2.1 Vấn đề xung đột tài nguyên

Vấn đề xung đột tài nguyên xảy ra khi hệ thống không cung cấp đủ tài nguyên phần cứng phục vụ CPU thực hiện đồng thời nhiều lệnh trong cơ chế ống lệnh. Hai xung đột tài nguyên thường gặp nhất là xung đột truy cập bộ nhớ và xung đột truy cập các thanh ghi. Giả sử bộ nhớ chỉ hỗ trợ một truy cập tại mỗi thời điểm và nếu tại cùng một thời điểm, có hai yêu cầu truy cập bộ nhớ đồng thời từ 2 lệnh được thực hiện trong ống lệnh (đọc lệnh – tại giai đoạn IF và đọc dữ liệu – tại giai đoạn ID) sẽ nảy sinh xung đột. Điều tương tự cũng có thể xảy ra với các thanh ghi khi có 2 hay nhiều lệnh đang thực hiện đồng yêu cầu đọc/ghi cùng một thanh ghi.

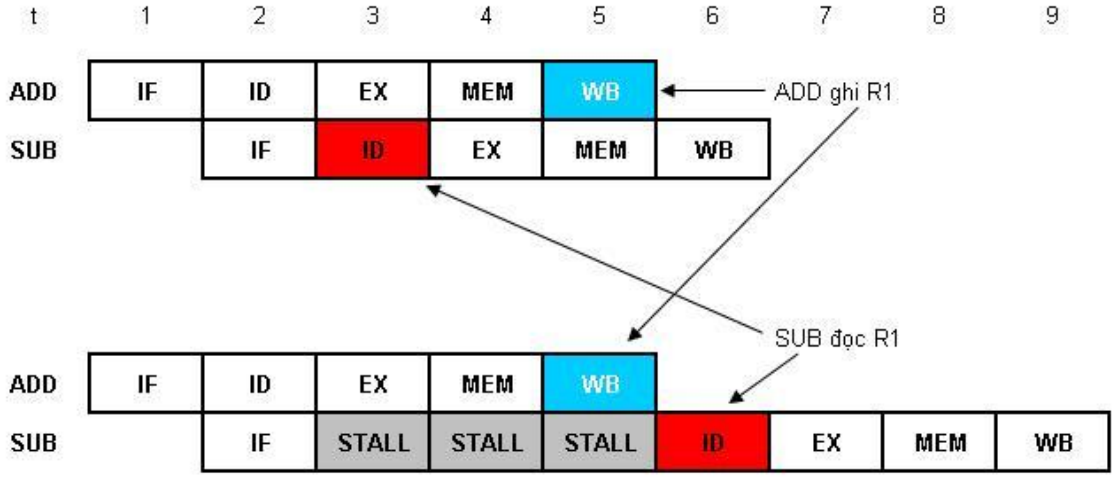
Giải pháp tối ưu cho vấn đề xung đột tài nguyên là nâng cao năng lực phục vụ của các tài nguyên phần cứng. Với xung đột truy cập bộ nhớ có thể sử dụng hệ thống nhớ hỗ trợ nhiều lệnh đọc ghi đồng thời, hoặc sử dụng các bộ nhớ tiên tiến như bộ nhớ cache. Với xung đột truy cập các thanh ghi, giải pháp là tăng số lượng thanh ghi vật lý và có cơ chế cấp phát thanh ghi linh hoạt khi thực hiện các lệnh.

3.6.2.2 Vấn đề tranh chấp dữ liệu

Tranh chấp dữ liệu cũng là một trong các vấn đề lớn của cơ chế ống lệnh và tranh chấp dữ liệu kiểu *đọc sau khi ghi* (RAW – Read After Write) là dạng xung đột dữ liệu hay gặp nhất. Để hiểu rõ tranh chấp dữ liệu kiểu RAW, ta xem xét hai lệnh sau:

ADD R_1, R_2, R_3; $R_1 \leftarrow R_2 + R_3$ (1)

SUB R_4, R_1, R_2; $R_4 \leftarrow R_1 + R_2$ (2)



Hình 27 Tranh chấp dữ liệu kiểu RAW

Hình 27 minh hoạ tranh chấp dữ liệu kiểu RAW giữa hai lệnh ADD và SUB được thực hiện kề nhau trong cơ chế ống lệnh. Có thể thấy lệnh SUB sử dụng kết quả của lệnh ADD (thanh ghi R_1 là kết quả của ADD và là đầu vào cho SUB) và như vậy hai lệnh có sự phụ thuộc dữ liệu. Tuy nhiên, lệnh SUB đọc thanh ghi R_1 tại giai đoạn giải mã (ID), trước khi lệnh ADD ghi kết quả vào thanh ghi R_1 ở giai đoạn lưu kết quả (WB). Như vậy, giá trị SUB đọc được từ thanh ghi R_1 là giá trị cũ, không phải là kết quả tạo ra bởi ADD. Để SUB đọc được giá trị mới nhất của R_1, giai đoạn ID của SUB phải lùi 3 nhịp, đến vị trí giai đoạn WB của ADD kết thúc.

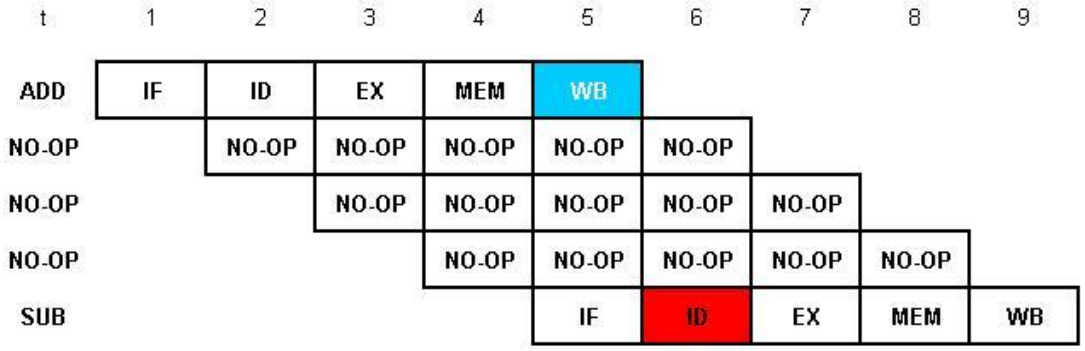
Có một số giải pháp cho vấn đề tranh chấp dữ liệu kiểu RAW. Cụ thể:

1. Nhận dạng tranh chấp RAW khi nó diễn ra;
2. Khi tranh chấp RAW xảy ra, tạm dừng (stall) ống lệnh cho đến khi lệnh phía trước hoàn tất giai đoạn WB;
3. Có thể sử dụng trình biên dịch (compiler) để nhận dạng tranh chấp RAW và thực hiện:
   - Chèn thêm các lệnh NO-OP vào giữa các lệnh có thể gây ra tranh chấp RAW; NO-OP là lệnh rỗng, không thực hiện tác vụ hữu ích mà chỉ tiêu tốn thời gian CPU.
   - Thay đổi trật tự các lệnh trong chương trình và chèn các lệnh độc lập vào giữa các lệnh có thể gây ra tranh chấp RAW;

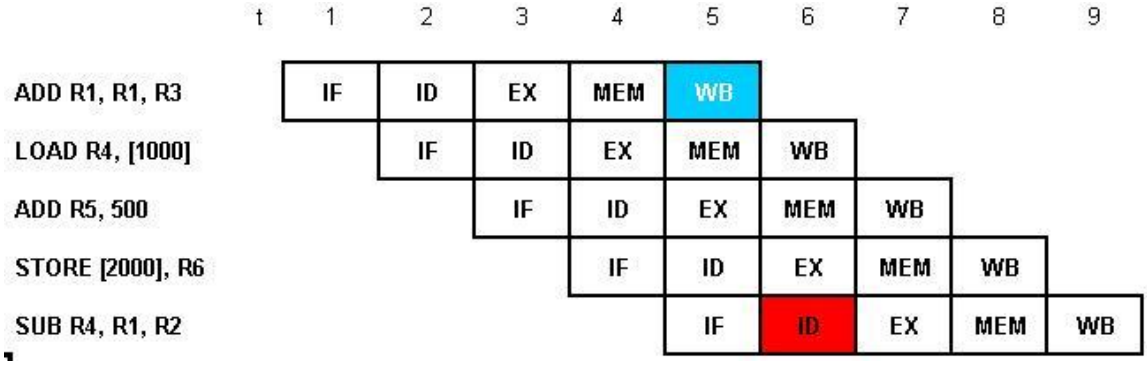
Mục đích của cả hai phương pháp kể trên là lùi việc thực hiện lệnh gây tranh chấp dữ liệu cho đến khi lệnh trước nó hoàn tất việc lưu kết quả.

4. Sử dụng phần cứng để nhận dạng tranh chấp RAW và dự đoán trước giá trị dữ liệu phụ thuộc.

Hình 28 minh hoạ giải pháp khắc phục tranh chấp RAW bằng cách chèn thêm các lệnh NO-OP. Hình 29 minh hoạ giải pháp khắc phục tranh chấp RAW bằng cách chèn thêm các lệnh độc lập với hai lệnh có tranh chấp. Các lệnh độc lập có thể có được bằng cách thay đổi trật tự thực hiện các lệnh của chương trình mà không thay đổi kết quả thực hiện nó. Cũng có thể sử dụng giải pháp kết hợp chèn NO-OP và lệnh độc lập.



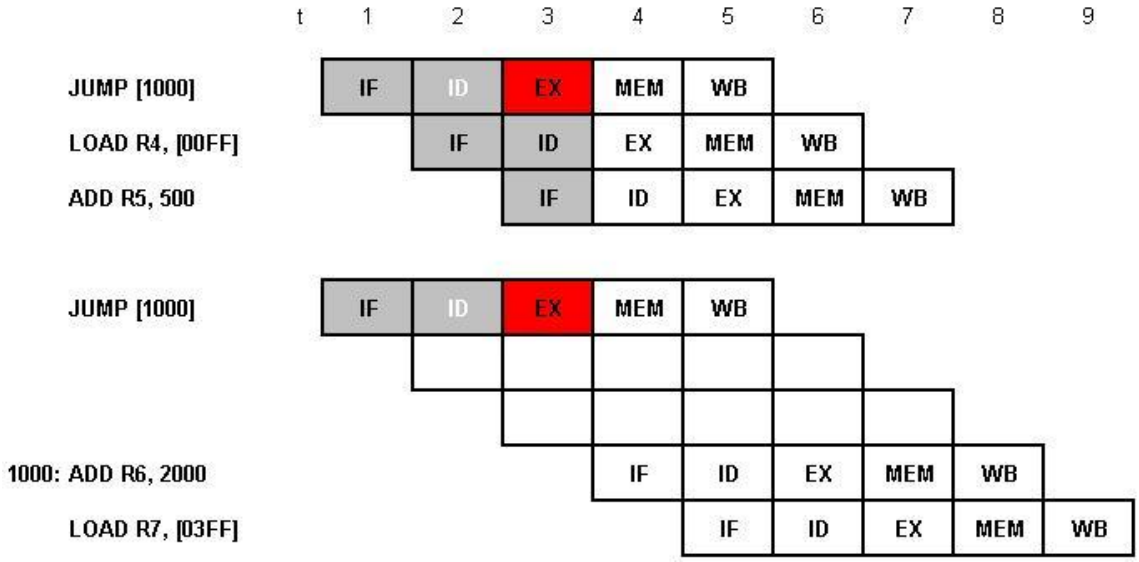
Hình 28 Khắc phục tranh chấp RAW bằng chèn thêm NO-OP



Hình 29 Khắc phục tranh chấp RAW bằng chèn các lệnh độc lập

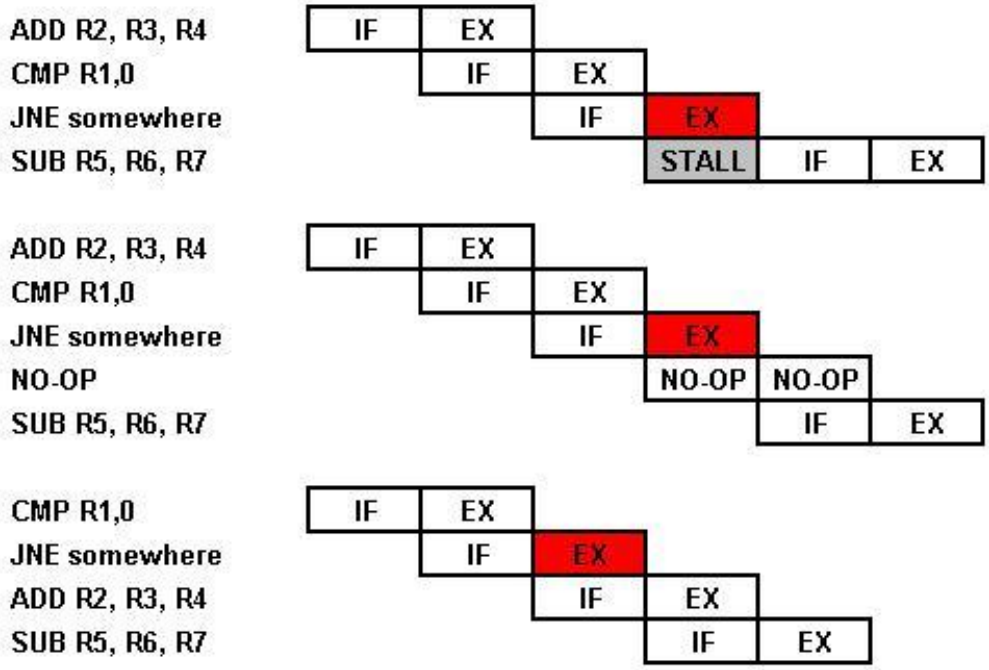
3.6.2.3 Vấn đề nảy sinh do các lệnh rẽ nhánh

Theo thống kê, tỷ lệ các lệnh rẽ nhánh trong chương trình khoảng 10-30%. Do lệnh rẽ nhánh thay đổi nội dung của bộ đếm chương trình, chúng có thể phá vỡ tiến trình thực hiện tuần tự các lệnh trong ống lệnh vì lệnh được thực hiện sau lệnh rẽ nhánh có thể không phải là lệnh liền sau nó mà là một lệnh ở vị trí khác. Như vậy, do kiểu thực hiện gối đầu, các lệnh liền sau lệnh rẽ nhánh đã được nạp và thực hiện dở dang trong trong ống lệnh sẽ bị đẩy ra làm cho ống lệnh bị trống rỗng và hệ thống phải bắt đầu nạp mới các lệnh từ địa chỉ đích rẽ nhánh. Hình 30 minh hoạ vấn đề nảy sinh trong ống lệnh do lệnh rẽ nhánh. Các lệnh sau lệnh rẽ nhánh LOAD và ADD bị đẩy ra và hệ thống nạp mới các lệnh từ địa chỉ đích rẽ nhánh 1000.



Hình 30 Vấn đề nảy sinh do lệnh rẽ nhánh

Có nhiều giải pháp khắc phục các vấn đề nảy sinh do các lệnh rẽ nhánh, như sử dụng đích rẽ nhánh (branch targets), làm chậm rẽ nhánh (delayed branching) và dự đoán rẽ nhánh (branch prediction). Tài liệu này chỉ giới thiệu phương pháp làm chậm rẽ nhánh. Ý tưởng chính của phương pháp làm chậm rẽ nhánh là lệnh rẽ nhánh sẽ không gây ra sự rẽ nhánh tức thì mà được làm "trễ" một số chu kỳ, phụ thuộc vào chiều dài của ống lệnh. Phương pháp này cho hiệu quả khá tốt với các ống lệnh ngắn, thường là 2 giai đoạn và với ràng buộc lệnh ngay sau lệnh rẽ nhánh luôn được thực hiện, không phụ thuộc vào kết quả của lệnh rẽ nhánh. Cách thực hiện của phương pháp chậm rẽ nhánh là chèn thêm một lệnh NO-OP hoặc một lệnh độc lập vào ngay sau lệnh rẽ nhánh. Hình 31 minh hoạ vấn đề nảy sinh do lệnh rẽ nhánh có điều kiện JNE (nhảy nếu R1 không bằng 0), giải pháp chèn một lệnh NO-OP hoặc một lệnh độc lập vào sau lệnh nhảy để khắc phục.



Hình 31 Khắc phục vấn đề lệnh rẽ nhánh bằng cách chèn NO-OP hoặc lệnh độc lập

3.7 CÂU HỎI ÔN TẬP

1. Khái niệm lệnh và tập lệnh? Chu kỳ lệnh và các giai đoạn thực hiện lệnh.
2. Dạng lệnh và các dạng địa chỉ toán hạng.
3. Khái niệm chế độ địa chỉ và các chế độ địa chỉ.
4. Nêu một số dạng lệnh thông dụng.
5. Nguyên lý hoạt động của cơ chế ống lệnh của CPU?
6. Các vấn đề của cơ chế ống lệnh của CPU và hướng khắc phục.

CHƯƠNG 4 BỘ NHỚ TRONG

4.1 PHÂN LOẠI BỘ NHỚ MÁY TÍNH

4.1.1 Phân loại bộ nhớ

Bộ nhớ máy tính gồm nhiều thành phần với tốc độ truy cập và và dung lượng khác nhau được kết hợp với nhau tạo thành hệ thống nhớ. Có nhiều cách phân loại bộ nhớ máy tính. Tựu chung, có thể chia bộ nhớ máy tính dựa trên ba tiêu chí: (1) kiểu truy cập, (2) khả năng duy trì dữ liệu và (3) công nghệ chế tạo.

Dựa trên kiểu truy cập, có thể chia bộ nhớ máy tính thành ba loại: Bộ nhớ truy cập tuần tự (Serial Access Memory - SAM), bộ nhớ truy nhập ngẫu nhiên (Random Access Memory - RAM), và bộ nhớ chỉ đọc (Read Only Memory - ROM). Trong bộ nhớ truy cập tuần tự, các ô nhớ được truy cập một cách tuần tự, có nghĩa là muốn truy cập đến ô nhớ sau phải duyệt qua ô nhớ trước nó. Tốc độ truy cập các ô nhớ có vị trí khác nhau là không giống nhau. Ngược lại, trong bộ nhớ truy nhập ngẫu nhiên, các ô nhớ có thể được truy cập ngẫu nhiên, không theo một trật tự định trước. Với bộ nhớ chỉ đọc, thông tin được ghi vào bộ nhớ một lần nhờ một thiết bị đặc biệt và sau đó chỉ có thể đọc ra.

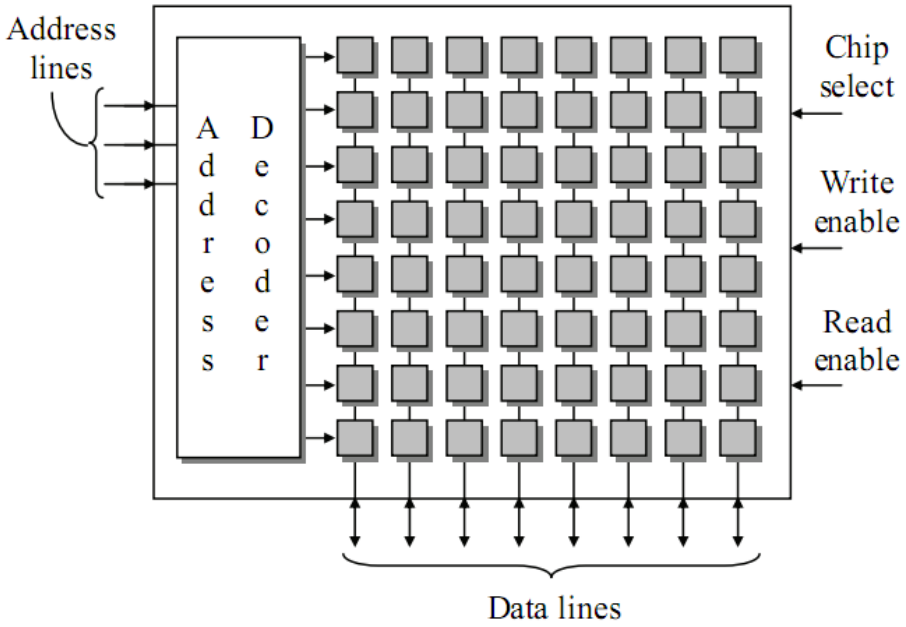
Dựa trên khả năng duy trì dữ liệu, có hai loại bộ nhớ: bộ nhớ ổn định (Non-volatile memory) và bộ nhớ không ổn định (Volatile memory). Bộ nhớ ổn định có khả năng duy trì dữ liệu kể cả khi không có nguồn nuôi. Đại diện tiêu biểu cho bộ nhớ ổn định là bộ nhớ ROM. Ngược lại, thông tin trong bộ nhớ không ổn định chỉ tồn tại khi có nguồn nuôi và sẽ mất khi mất nguồn nuôi. Đại diện tiêu biểu cho bộ nhớ không ổn định là bộ nhớ RAM.

Dựa trên công nghệ chế tạo, có ba loại bộ nhớ: bộ nhớ bán dẫn (Semiconductor memory), bộ nhớ từ tính (Magnetic memory), bộ nhớ quang học (Optical memory). Bộ nhớ bán dẫn được chế tạo bằng vật liệu bán dẫn, thường có tốc độ truy cập rất cao, nhưng giá thành đắt. Đại diện cho bộ nhớ bán dẫn là bộ nhớ ROM và RAM. Bộ nhớ từ tính là bộ nhớ dựa trên từ tính của các vật liệu có khả năng nhiễm từ để lưu trữ và đọc / ghi thông tin. Đại diện cho bộ nhớ từ tính là các loại đĩa từ (đĩa mềm, đĩa cứng) và băng từ. Bộ nhớ quang học là bộ nhớ hoạt động dựa trên các nguyên lý quang – điện. Đại diện cho bộ nhớ quang học là các loại đĩa quang, như đĩa CD, DVD,...

4.1.2 Tổ chức mạch nhớ

Một mạch nhớ (memory chip) thường gồm nhiều ô nhớ (memory cells) được tổ chức thành một ma trận nhớ gồm một số hàng và một số cột. Hình 32 minh hoạ tổ chức một mạch nhớ RAM. Ngoài ma trận nhớ gồm các ô nhớ, mạch nhớ còn gồm các đường địa chỉ (Address lines), bộ giải mã địa chỉ (Address decoder), các đường dữ liệu (Data lines) và các tín hiệu điều khiển như tín hiệu chọn mạch (Chip select - CS), tín hiệu cho phép đọc (Read enable - RE) và tín hiệu cho phép ghi (Write enable - WE).

Các đường địa chỉ là một tập các chân tín hiệu kết nối với bus địa chỉ nhận các tín hiệu địa chỉ ô nhớ từ CPU. Bộ giải mã địa chỉ giải mã các tín hiệu địa chỉ ô nhớ thành các địa chỉ hàng và cột để có thể chọn ra được ô nhớ. Các đường dữ liệu là một tập các chân tín hiệu kết nối với bus dữ liệu để nhận tín hiệu dữ liệu từ CPU và gửi tín hiệu dữ liệu đọc được từ ô nhớ về CPU.



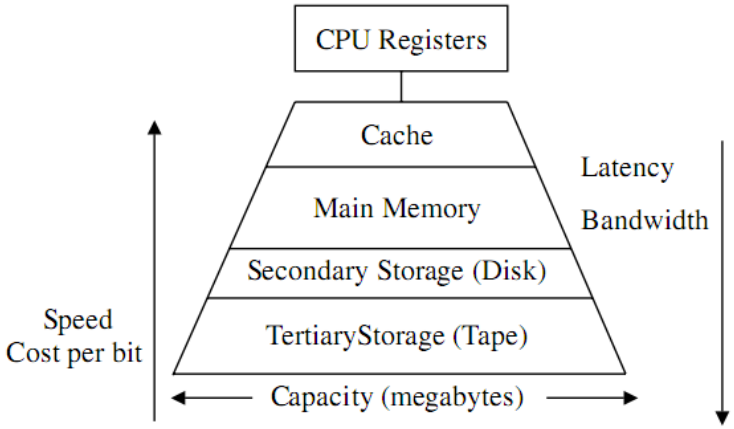
Hình 32 Tổ chức mạch nhớ

Các tín hiệu điều khiển có nhiệm vụ điều khiển hoạt động của mạch nhớ theo các tín hiệu lệnh gửi đến từ CPU. Tín hiệu chọn mạch CS cho phép kích hoạt mạch nhớ làm việc với CPU khi CS = 0. Thông thường, tại mỗi thời điểm chỉ có một mạch nhớ được chọn kích hoạt làm việc với CPU, còn các mạch khác ở trạng thái không được kích hoạt. Tín hiệu cho phép ghi WE = 0 sẽ cho phép ghi thông tin vào các ô nhớ trong một dòng. Tương tự, tín hiệu cho phép đọc RD = 0 sẽ cho phép đọc dữ liệu từ các ô nhớ trong một dòng.

4.2 CẤU TRÚC PHÂN CẤP BỘ NHỚ MÁY TÍNH

4.2.1 Giới thiệu cấu trúc phân cấp hệ thống nhớ

Hầu hết hệ thống nhớ trong các thiết bị tính toán hiện đại được tổ chức theo cấu trúc phân cấp (hierachical structure). Cấu trúc phân cấp không chỉ được sử dụng trong các hệ thống nhớ mà nó còn sử dụng rộng rãi trong đời sống xã hội, như cấu trúc tổ chức các cơ quan nhà nước, doanh nghiệp và cả các trường học. Hình 33 minh hoạ cấu trúc phân cấp hệ thống nhớ, gồm các phần chính: các thanh ghi của CPU (CPU Registers), bộ nhớ cache (Cache), bộ nhớ chính (Main Memory) và bộ nhớ ngoài (Secondary / Tertiary Storage).



Hình 33 Cấu trúc phân cấp hệ thống nhớ

	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

Hình 34 Dung lượng, thời gian truy cập và giá thành các loại bộ nhớ

Trong cấu trúc phân cấp hệ thống nhớ, dung lượng các thành phần tăng theo chiều từ các thanh ghi của CPU đến bộ nhớ ngoài. Ngược lại, tốc độ truy nhập hay băng thông và giá thành một đơn vị nhớ tăng theo chiều từ bộ nhớ ngoài đến các thanh ghi của CPU. Như vậy, các thanh ghi của CPU có dung lượng nhỏ nhất nhưng có tốc độ truy cập nhanh nhất và cũng có giá thành cao nhất. Bộ nhớ ngoài có dung lượng lớn nhất, nhưng tốc độ truy cập thấp nhất. Bù lại, bộ nhớ ngoài có giá thành rẻ nên có thể được sử dung với dung lượng lớn.

Các thanh ghi được tích hợp trong CPU và thường hoạt động theo tần số làm việc của CPU, nên đạt tốc độ truy cập rất cao. Tuy nhiên, do không gian trong CPU rất hạn chế nên tổng dung lượng của các thanh ghi là khá nhỏ, chỉ khoảng vài chục byte đến vài kilobyte. Các thanh ghi thường được sử dụng để lưu toán hạng đầu vào và kết quả đầu ra của các lệnh phục vụ CPU làm việc.

Bộ nhớ cache có dung lượng tương đối nhỏ, khoảng từ vài chục kilobyte đến vài chục megabyte (khoảng 64KB đến 32MB với các máy tính hiện nay). Tốc độ truy cập cache cao, nhưng giá thành còn khá đắt. Cache được coi là bộ nhớ "thông minh" do có khả năng đoán trước được nhu cầu lệnh và dữ liệu của CPU. Cache "đoán" và tải trước các lệnh và dữ liệu CPU cần sử dụng từ bộ nhớ chính, nhờ vậy giúp CPU giảm thời gian truy cập hệ thống nhớ, tăng tốc độ xử lý.

Bộ nhớ chính gồm có bộ nhớ ROM và bộ nhớ RAM, có dung lượng khá lớn (khoảng từ 256MB đến 4GB với các hệ thống 32 bit), nhưng tốc độ truy cập tương đối chậm so với cache. Giá thành bộ nhớ chính tương đối thấp nên có thể sử dụng với dung lượng lớn. Bộ nhớ chính được sử dụng để lưu lệnh và dữ liệu của hệ thống và của người dùng.

Bộ nhớ ngoài hay bộ nhớ thứ cấp, gồm các loại đĩa từ, đĩa quang và băng từ. Bộ nhớ ngoài thường có dung lượng rất lớn, khoảng 20GB đến 1000GB, nhưng tốc độ truy cập rất chậm. Bộ nhớ ngoài có ưu điểm là giá thành rẻ và thường được sử dụng để lưu trữ dữ liệu lâu dài dưới dạng các tệp (files).

4.2.2 Vai trò của cấu trúc phân cấp hệ thống nhớ

Không hoàn toàn giống với vai trò của cấu trúc phân cấp trong các cơ quan và doanh nghiệp là "chia để trị", cấu trúc phân cấp trong hệ thống nhớ có hai vai trò chính: (1) tăng hiệu năng hệ thống thông qua việc giảm thời gian truy cập các ô nhớ và (2) giảm giá thành sản xuất.

Sở dĩ cấu trúc phân cấp trong hệ thống nhớ có thể giúp tăng hiệu năng hệ thống là do nó giúp dung hoà được CPU có tốc độ cao và phần bộ nhớ chính và bộ nhớ ngoài có tốc độ thấp. CPU sẽ chủ yếu trực tiếp truy cập bộ nhớ cache có tốc độ cao, và cache sẽ có nhiệm vụ chuyển

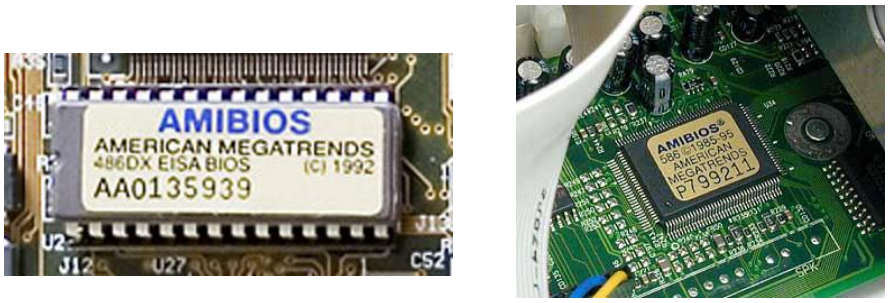
trước các dữ liệu cần thiết về từ bộ nhớ chính. Nhờ vậy, CPU sẽ không phải thường xuyên truy cập trực tiếp bộ nhớ chính và bộ nhớ ngoài để tìm dữ liệu – các thao tác tốn nhiều thời gian do các bộ nhớ này có tốc độ chậm. Như vậy, có thể nói rằng, thời gian trung bình CPU truy nhập dữ liệu từ hệ thống nhớ tiệm cận thời gian truy nhập bộ nhớ cache.

Cùng với việc có thể giúp cải thiện hiệu năng, cấu trúc phân cấp trong hệ thống nhớ có thể giúp giảm giá thành chế tạo hệ thống. Cơ sở chính là trong hệ thống nhớ phân cấp, các thành phần có tốc độ cao và đắt tiền được sử dụng với dung lượng rất nhỏ, còn các thành phần có tốc độ thấp và rẻ tiền được sử dụng với dung lượng lớn hơn. Nhờ vậy có thể giảm được giá thành chế tạo hệ thống nhớ mà vẫn đảm bảo được tốc độ cao cho cả hệ thống. Nếu ta có hai hệ thống nhớ hoạt động với cùng tốc độ thì hệ thống nhớ phân cấp sẽ có giá thành thấp hơn.

4.3 BỘ NHỚ ROM VÀ RAM

4.3.1 Bộ nhớ ROM

ROM (Read Only Memory) là bộ nhớ chỉ đọc, có nghĩa là thông tin lưu trữ trong ROM chỉ có thể đọc ra mà không được ghi vào. Trên thực tế, việc ghi thông tin vào ROM chỉ có thể được thực hiện bằng các thiết bị chuyên dùng hoặc phương pháp đặc biệt. Thông tin trong ROM thường được các nhà sản xuất ghi sẵn, gồm các thông tin về hệ thống như thông tin về cấu hình máy và hệ thống các mô đun phần mềm phục vụ việc vào ra cơ sở (BIOS - Basic Input Output System). ROM thuộc loại bộ nhớ bán dẫn và là bộ nhớ ổ định - thông tin trong ROM vẫn được duy trì kể cả khi không có nguồn điện nuôi. Hình 35 minh hoạ vi mạch nhớ ROM-BIOS được gắn trên bảng mạch chính.



Hình 35 Vi mạch nhớ ROM-BIOS

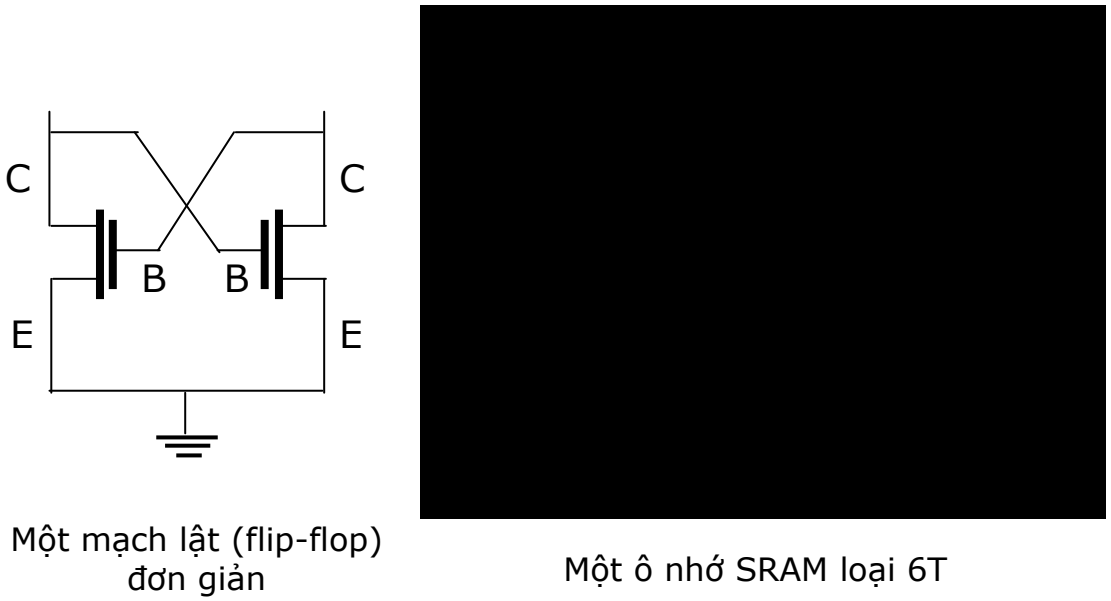
Quá trình phát triển ROM đã trải qua nhiều thế hệ. ROM các thế hệ đầu tiên hay còn gọi là ROM nguyên thuỷ (Ordinary ROM) sử dụng tia cực tím để ghi thông tin. Trong thế hệ tiếp theo - ROM có thể lập trình được (PROM - Programmable ROM), thông tin có thể được ghi vào PROM nhờ một thiết bị đặc biệt gọi là bộ lập trình PROM. Tiến thêm một bước, với ROM có thể lập trình và xoá được (EPROM - Erasable programmable read-only memory), thông tin trong EPROM có thể xoá được sử dụng tia cực tím có cường độ cao. Kế tiếp EPROM, EEPROM (Electrically Erasable PROM) là loại ROM tiến tiến nhất hiện nay. EEPROM có thể xoá được bằng điện và có thể ghi được thông tin sử dụng phần mềm chuyên dụng. Bộ nhớ Flash là một dạng bộ nhớ EEPROM được dùng phổ biến làm thiết bị lưu trữ trong các thiết bị cầm tay. Flash có tốc độ đọc ghi thông tin nhanh hơn EEPROM và thông tin được đọc ghi theo từng khối.

4.3.2 Bộ nhớ RAM

Bộ nhớ RAM được chế tạo theo công nghệ bán dẫn và thuộc loại bộ nhớ không ổn định, tức là, thông tin trong RAM chỉ tồn tại khi có nguồn điện nuôi và mất khi không còn nguồn điện nuôi. RAM là bộ nhớ cho phép truy cập ngẫu nhiên – các ô nhớ của RAM có thể được truy cập một cách ngẫu nhiên không theo trật tự nào và tốc truy cập các ô nhớ là tương đương nhau. RAM thường có dung lượng lớn hơn nhiều so với ROM và thường được sử dụng để lưu trữ các thông tin của hệ thống và của người dùng.

Có hai loại RAM cơ bản: RAM tĩnh (Static RAM hay SRAM) và RAM động (Dynamic RAM hay DRAM). Mỗi bit RAM tĩnh cấu tạo dựa trên một *mạch lật* (flip flop) – còn gọi là *mạch trigơ lưỡng ổn* (bistable latching circuit). Thông tin trong SRAM luôn ổn định và không phải "làm tươi" định kỳ. Tốc độ truy nhập SRAM cũng nhanh hơn nhiều so với DRAM. Ngược lại, mỗi bit DRAM cấu tạo dựa trên một tụ điện. Do bản chất của tụ điện luôn có khuynh hướng tự phóng điện tích, thông tin trong bit DRAM sẽ dần bị mất. Vì vậy, DRAM cần được làm tươi (refresh) định kỳ để bảo toàn thông tin. DRAM thường có tốc độ truy cập thấp hơn so với SRAM, nhưng bù lại, DRAM có cấu trúc gọn nhẹ nên có thể tăng mật độ cấy linh kiện dẫn đến giá thành một đơn vị nhớ DRAM thấp hơn SRAM.

4.3.2.1 Bộ nhớ SRAM



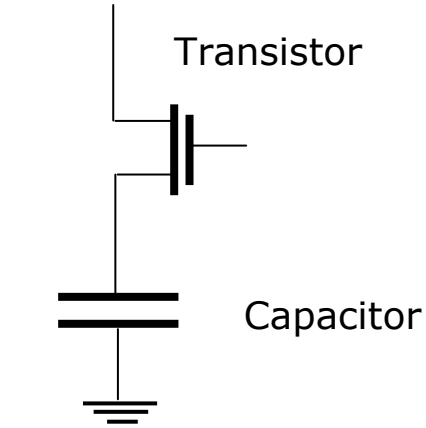
Hình 36 Cấu tạo một mạch lật trong bộ nhớ SRAM

Hình 36 minh hoạ cấu tạo một mạch lật đơn giản gồm 2 transitor và một mạch lật phức tạp hơn với 6 transitor (6T) – hình thành một bit nhớ của SRAM. Thông thường, mỗi bit nhớ của SRAM được cấu tạo từ một mạch lật 6T, 8T hoặc 10T. SRAM có tốc độ truy cập cao là do các bit SRAM có cấu trúc đối xứng và thông tin trong bit SRAM ổn định nên không cần quá trình làm tươi. Tuy nhiên, do mỗi bit SRAM cần nhiều transitor và có cấu trúc khá phức tạp nên mật độ cấy linh kiện thường thấp và giá thành SRAM khá cao.

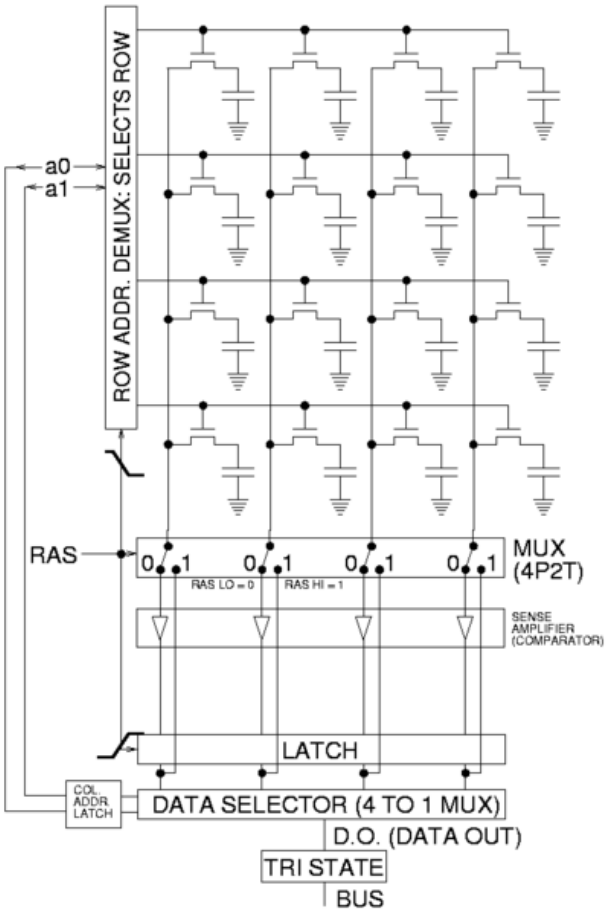
4.3.2.2 Bộ nhớ DRAM

Khác với SRAM, các bit DRAM được hình thành dựa trên tụ điện. Hình 37 minh hoạ một bit DRAM và mạch nhớ DRAM tổ chức thành ma trận nhớ gồm các hàng và cột. Mỗi bit DRAM

có cấu tạo khá đơn giản, gồm 1 tụ điện và 1 transitor cấp nguồn. Mức điện tích trong tụ điện được sử dụng để biểu diễn các giá trị 0 và 1, chẳng hạn mức đầy điện tích ứng với mức 1, không tích điện ứng với mức 0.



Một bit DRAM



Hình 37 Một bit DRAM và mạch nhớ DRAM

Do bản chất tụ thường tự phóng điện nên điện tích trong tụ có xu hướng giảm dần dẫn đến thông tin trong tụ cũng bị mất theo. Để tránh bị mất thông tin, điện tích trong tụ cần được nạp lại thường xuyên – quá trình này được gọi là quá trình làm tươi các bit DRAM. DRAM thường có tốc độ truy cập chậm hơn so với SRAM là do: (1) có trễ khi nạp điện vào tụ, (2) cần quá trình làm tươi cho tụ và (3) các mạch DRAM thường dùng kỹ thuật dồn kênh (địa chỉ cột/hàng) để tiết kiệm đường địa chỉ. Tuy nhiên, do mỗi bit DRAM có cấu trúc đơn giản, sử dụng ít transitor nên mật độ cấy linh kiện thường cao và giá thành rẻ hơn nhiều so với SRAM.

Trong các loại DRAM, SDRAM (Synchronous DRAM) được sử dụng phổ biến nhất. SDRAM là DRAM hoạt động đồng bộ với nhịp đồng hồ của bus. SDRAM được chia thành 2 loại theo khả năng truyền dữ liệu: (1) SRD SDRAM (Single Data Rate SDRAM) – SDRAM có tỷ suất dữ liệu đơn, chấp nhận một thao tác đọc/ghi và chuyển 1 từ dữ liệu trong 1 chu kỳ đồng hồ với các tần số làm việc 100MHz và 133MHz và (2) DDR SDRAM (Double Data Rate SDRAM) - SDRAM có tỷ suất dữ liệu kép, chấp nhận hai thao tác đọc/ghi và chuyển 2 từ dữ liệu trong 1 chu kỳ đồng hồ. DDR SDRAM có 3 loại cho đến hiện nay:

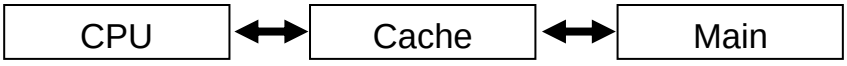
- DDR1 SDRAM: tần số làm việc 266, 333, 400 MHz: có khả năng chuyển 2 từ dữ liệu trong 1 chu kỳ đồng hồ;

- DDR2 SDRAM: tần số làm việc 400, 533, 800 MHz: có khả năng chuyển 4 từ dữ liệu trong 1 chu kỳ đồng hồ;
- DDR3 SDRAM: tần số làm việc 800, 1066, 1333, 1600 MHz: có khả năng chuyển 8 từ dữ liệu trong 1 chu kỳ đồng hồ.

4.4 BỘ NHỚ CACHE

4.4.1 Cache là gì?

Cache hay còn gọi là bộ nhớ đệm, bộ nhớ khay là một thành phần của cấu trúc phân cấp của hệ thống bộ nhớ như trình bày trong mục 4.2. Cache đóng vai trong trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại. Hình 38 minh hoạ vị trí của bộ nhớ cache trong hệ thống nhớ. Với các hệ thống CPU cũ sử dụng công nghệ tích hợp thấp, bộ nhớ cache thường nằm ngoài CPU; với các CPU mới sử dụng công nghệ tích hợp cao, bộ nhớ cache thường được tích hợp vào trong CPU nhằm nâng cao tốc độ và băng thông trao đổi dữ liệu giữa CPU và cache.



Hình 38 Vị trí của bộ nhớ cache trong hệ thống nhớ

Dung lượng của bộ nhớ cache thường nhỏ so với dung lượng của bộ nhớ chính và bộ nhớ ngoài. Với các hệ thống máy tính cũ, dung lượng cache là khoảng 16KB, 32KB,..., 128KB; với các hệ thống máy tính gần đây, dung lượng cache lớn hơn, khoảng 256KB, 512KB, 1MB, 2MB, 4MB, 8MB và 16MB. Cache có tốc độ truy cập nhanh hơn nhiều so với bộ nhớ chính, đặc biệt với cache được tích hợp vào CPU. Tuy nhiên, giá thành bộ nhớ cache (tính theo bit) thường đắt hơn nhiều so với bộ nhớ chính. Với các hệ thống CPU mới, cache thường được chia thành hai hay nhiều mức (levels): mức 1 có dung lượng khoảng 16-32KB có tốc độ truy cập rất cao và mức 2 có dung lượng khoảng 1-16MB có tốc độ truy cập thấp hơn.

4.4.2 Vai trò và nguyên lý hoạt động

4.4.2.1 Vai trò của cache

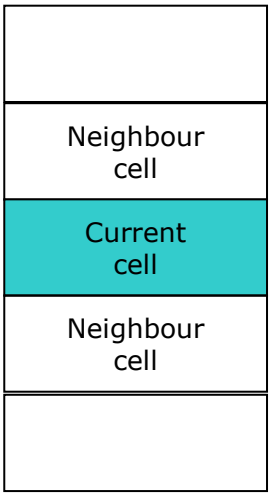
Do nhớ cache là một thành phần của hệ thống nhớ phân cấp, nên vai trò của cache tương tự như vai trò của cấu trúc phân cấp hệ thống nhớ: tăng hiệu năng hệ thống vào giảm giá thành sản xuất. Sở dĩ cache có thể giúp tăng hiệu năng hệ thống là nhờ cache có khả năng dung hoà được CPU có tốc độ cao và bộ nhớ chính có tốc độ thấp làm cho thời gian trung bình CPU truy nhập dữ liệu từ bộ nhớ chính tiệm cận thời gian truy nhập cache. Ngoài ra, do cache là một loại bộ nhớ "thông minh" có khả năng đoán và chuẩn bị trước các dữ liệu cần thiết cho CPU xử lý nên xác xuất CPU phải trực tiếp truy nhập dữ liệu từ bộ nhớ chính là khá thấp và điều này cũng giúp làm giảm thời gian trung bình CPU truy nhập dữ liệu từ bộ nhớ chính.

Tuy cache có giá thành trên một đơn vị nhớ cao hơn bộ nhớ chính, nhưng do tổng dung lượng cache thường khá nhỏ nên cache không làm tăng giá thành hệ thống nhớ quá mức. Nhờ vậy, cache hoàn toàn phù hợp với cấu trúc phân cấp và có thể giúp làm giảm giá thành sản xuất trong tương quan với tốc độ của cả hệ thống nhớ. Có thể kết luận rằng, nếu hai hệ thống nhớ

có cùng giá thành, hệ thống nhớ có cache có tốc độ truy cập nhanh hơn; và nếu hai hệ thống nhớ có cùng tốc độ, hệ thống nhớ có cache sẽ có giá thành rẻ hơn.

4.4.2.2 Nguyên lý hoạt động của cache

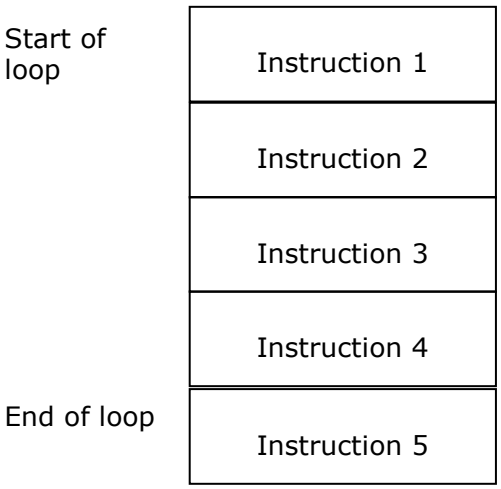
Cache sở dĩ được coi là bộ nhớ "thông minh" là do nó có khả năng đoán trước yêu cầu về dữ liệu và lệnh của CPU. Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache và CPU chỉ cần truy nhập cache, giúp giảm thời gian truy nhập hệ thống nhớ. Để có được sự thông minh, cache hoạt động dựa trên hai nguyên lý cơ bản: nguyên lý *lân cận về không gian* (Spatial locality) và nguyên lý *lân cận về thời gian* (Temporal locality).



Hình 39 Lân cận về không gian trong không gian chương trình

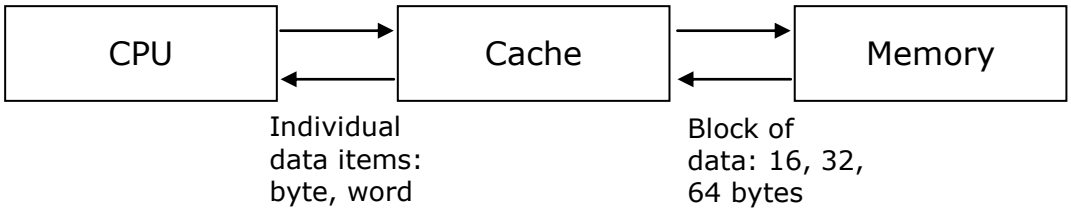
Nguyên lý lân cận về không gian có thể phát biểu như sau: "*Nếu một ô nhớ đang được truy nhập thì xác xuất các ô nhớ liền kề với nó được truy nhập trong tương lai gần là rất cao*". Lân cận về không gian thường được áp dụng cho nhóm lệnh hoặc dữ liệu có tính tuần tự cao trong không gian chương trình, như minh hoạ trên Hình 39. Do các lệnh trong một chương trình thường tuần tự, cache có thể đọc cả khối lệnh từ bộ nhớ chính và khối lệnh đọc được bao phủ cả các ô nhớ lân cận (neighbour cell) của ô nhớ đang được truy nhập (current cell).

Khác với nguyên lý lân cận về không gian, nguyên lý lân cận về thời gian chú trọng hơn đến tính *lặp lại* của việc truy nhập các mẩu thông tin trong một khoảng thời gian tương đối ngắn. Có thể phát biểu nguyên lý này như sau: "*Nếu một ô nhớ đang được truy nhập thì xác xuất nó được truy nhập lại trong tương lai gần là rất cao*". Lân cận về thời gian được áp dụng cho dữ liệu và nhóm các lệnh trong vòng lặp như minh hoạ trên Hình 40. Với các phần tử dữ liệu, chúng được CPU cập nhập thường xuyên trong quá trình thực hiện chương trình nên có tính lân cận cao về thời gian. Với các lệnh trong vòng lặp, chúng thường được CPU thực hiện lặp đi lặp lại nhiều lần nên cũng có tính lân cận cao về thời gian; nếu cache nạp sẵn khối lệnh chứa cả vòng lặp sẽ phủ được tính lân cận về thời gian.



Hình 40 Lân cận về thời gian với việc thực hiện vòng lặp

4.4.2.3 Trao đổi dữ liệu giữa CPU – cache – bộ nhớ chính



Hình 41 Trao đổi dữ liệu giữa CPU với cache và bộ nhớ chính

Hình 41 minh hoạ việc trao đổi dữ liệu giữa CPU với cache và bộ nhớ chính: CPU trao đổi dữ liệu với cache theo các đơn vị cơ sở như byte, từ và từ kép. Còn cache trao đổi dữ liệu với bộ nhớ chính theo các khối, với kích thước 16, 32 hoặc 64 bytes. Sở dĩ CPU trao đổi dữ liệu với cache theo các đơn vị cơ sở mà không theo khối do dữ liệu được lưu trong các thanh ghi của CPU – vốn có dung lượng rất hạn chế. Vì vậy, CPU chỉ trao đổi các phần tử dữ liệu cần thiết theo yêu cầu của các lệnh. Ngược lại, cache trao đổi dữ liệu với bộ nhớ chính theo các khối, mỗi khối gồm nhiều byte kề nhau với mục đích bao phủ các mẩu dữ liệu lân cận theo không gian và thời gian. Ngoài ra, trao đổi dữ liệu theo khối (hay mẻ) với bộ nhớ chính giúp cache tận dụng tốt hơn băng thông đường truyền và nhờ vậy có thể tăng tốc độ truyền dữ liệu.

4.4.2.4 Các hệ số Hit và Miss

Hit (đoán trúng) là một sự kiện mà CPU truy nhập một mục tin và mục tin ấy có ở trong cache. Xác suất để có một hit gọi là hệ số hit, hoặc H. Dễ thấy hệ số hit H thuộc khoảng (0, 1). Hệ số hit càng cao thì hiệu quả của cache càng cao. Ngược lại, *Miss* (đoán trượt) là một sự kiện mà CPU truy nhập một mục tin và mục tin ấy không có ở trong cache. Xác suất của một miss gọi là hệ số miss, hoặc 1-H. Cũng có thể thấy hệ số miss 1-H thuộc khoảng (0, 1). Hệ số miss càng thấp thì hiệu quả của cache càng cao.

4.4.3 Các dạng kiến trúc cache

Kiến trúc cache đề cập đến việc cache được bố trí vào vị trí nào trong quan hệ với CPU và bộ nhớ chính. Có hai loại kiến trúc cache chính: kiến trúc Look Aside (cache được đặt ngang hàng với bộ nhớ chính) và kiến trúc Look Through (cache được đặt giữa CPU và bộ nhớ chính). Mỗi kiến trúc cache kể trên có ưu điểm và nhược điểm riêng.

4.4.3.1 Kiến trúc Look Aside

Trong kiến trúc Look Aside, cache và bộ nhớ chính cùng được kết nối vào bus hệ thống. Như vậy, cả cache và bộ nhớ chính đều "thấy" chu kỳ bus của CPU tại cùng một thời điểm. Hình 42 minh hoạ kiến trúc cache kiểu Look Aside. Kiến trúc Look Aside có thiết kế đơn giản, dễ thực hiện. Tuy nhiên, các sự kiện hit của kiến trúc này thường chậm do cache kết nối với CPU sử dụng bus hệ thống – thường có tần số làm việc không cao và băng thông hẹp. Bù lại, các sự kiện miss của kiến trúc Look Aside thường nhanh hơn do khi CPU không tìm thấy mục tin trong cache, nó đồng thời tìm mục tin trong bộ nhớ chính tại cùng một chu kỳ xung nhịp.

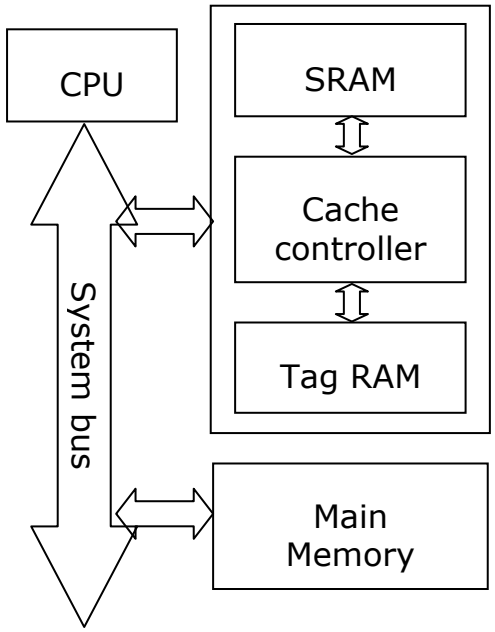
SRAM: RAM lưu dữ liệu cache

Tag RAM: RAM lưu địa chỉ dòng nhớ trong bộ nhớ

Cache controller: bộ điều khiển cache

Main memory: bộ nhớ chính

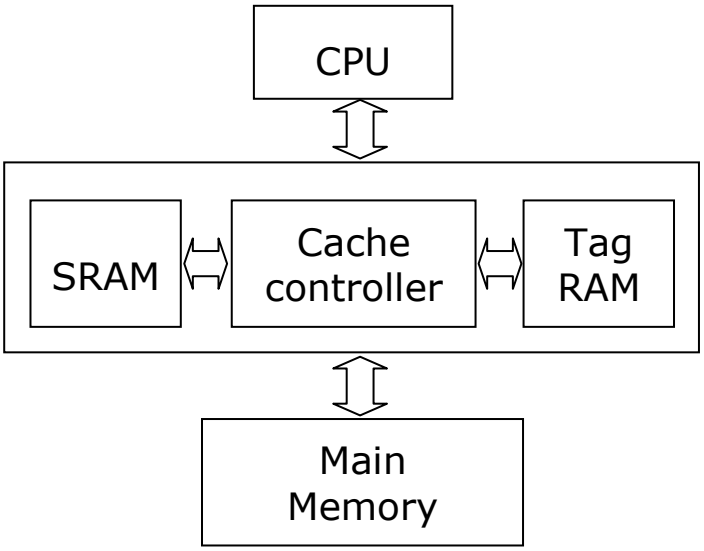
System bus: bus hệ thống



Hình 42 Kiến trúc cache kiểu Look Aside

4.4.3.2 Kiến trúc Look Through

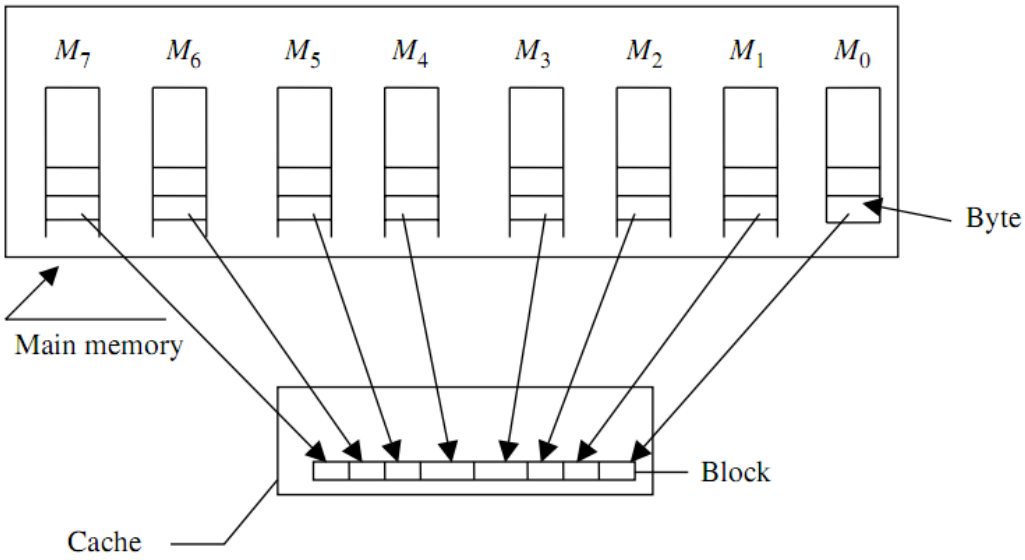
Trong kiến trúc kiểu Look Through, cache được đặt nằm giữa CPU và bộ nhớ chính như minh hoạ trên Hình 43. Như vậy cache có thể "thấy" chu kỳ bus của CPU trước, rồi nó mới "truyền" lại cho bộ nhớ chính. Cache kết nối với CPU bằng hệ thống bus riêng tốc độ cao và băng thông lớn, thường là bus mặt sau (BSB – Back Side Bus). Cache kết nối với bộ nhớ chính thông qua bus hệ thống hay bus mặt trước (FSB – Front Side Bus). FSB thường có tần số làm việc và băng thông thấp hơn nhiều so với BSB. Kiến trúc Look Through phức tạp hơn kiến trúc Look Aside. Ưu điểm chính của kiến trúc này là các sự kiện hit của kiến trúc này thường rất nhanh do CPU kết nối với cache bằng kênh riêng có tốc độ cao. Tuy nhiên, các sự kiện miss của kiến trúc Look Through thường chậm hơn do khi CPU không tìm thấy mục tin trong cache, nó cần tìm mục tin đó trong bộ nhớ chính tại một chu kỳ xung nhịp tiếp theo.



Hình 43 Kiến trúc cache kiểu Look Through

4.4.4 Các dạng tổ chức/ánh xạ cache

4.4.4.1 Giới thiệu tổ chức/ánh xạ cache



Hình 44 Quan hệ giữa các khối của bộ nhớ chính và dòng của cache

Như đã trình bày trong mục 4.2, kích thước của cache thường rất nhỏ so với kích thước bộ nhớ chính. Do vậy, tại mỗi thời điểm, chỉ có một phần nhỏ thông tin của bộ nhớ chính được chuyển vào cache. Câu hỏi đặt ra là, phải xây dựng mô hình tổ chức / ánh xạ trao đổi dữ liệu giữa các phần tử nhớ bộ nhớ chính và các phần tử nhớ của cache như thế nào để hệ thống nhớ đạt được tốc độ truy cập tối ưu.

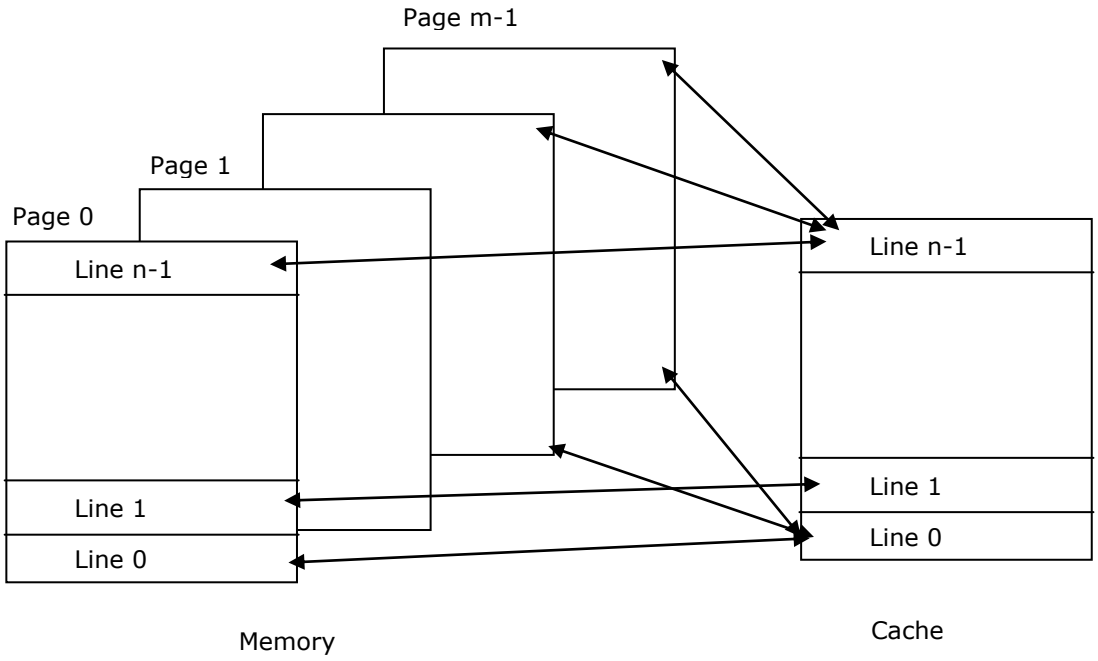
Cho đến hiện nay, có ba phương pháp tổ chức / ánh xạ cache đã được sử dụng, bao gồm: Ánh xạ trực tiếp (Direct mapping), Ánh xạ kết hợp đầy đủ (Fully associative mapping) và Ánh xạ tập kết hợp (Set associative mapping). Phương pháp ánh xạ trực tiếp có ưu điểm là thiết kế đơn giản và nhanh. Tuy nhiên, đây là dạng ánh xạ cố định dễ gây xung đột dẫn đến hiệu quả cache không cao. Phương pháp ánh xạ kết hợp đầy đủ sử dụng ánh xạ mềm, ít gây xung đột và có thể đạt hệ số hit rất cao. Tuy nhiên, phương pháp này có thiết kế phức tạp và có tốc độ

chậm. Phương pháp ánh xạ tập kết hợp là sự kết hợp của hai phương pháp ánh xạ trực tiếp và ánh xạ kết hợp đầy đủ, tận dụng được ưu điểm của cả hai phương pháp: nhanh, ít xung đột và không quá phức tạp, cho hiệu quả cao.

4.4.4.2 Ánh xạ trực tiếp

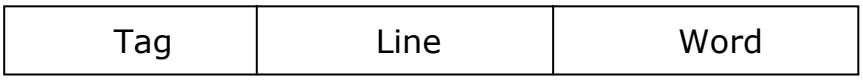
Hình 45 minh hoạ phương pháp ánh xạ trực tiếp bộ nhớ - cache. Cache được chia thành n dòng (line) đánh số từ 0 đến n-1. Bộ nhớ chính được chia thành m trang (page), đánh số từ 0 đến m-1. Mỗi trang nhớ lại được chia thành n dòng (line) đánh số từ 0 đến n-1. Kích thước mỗi trang của bộ nhớ chính bằng kích thước cache và kích thước một dòng trong trang bộ nhớ cũng bằng kích thước một dòng cache. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- $Line_0$ của các trang ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_0$ của cache;
- $Line_1$ của các trang ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_1$ của cache;
- ....
- $Line_{n-1}$ của các trang ($page_0$ đến $page_{m-1}$) ánh xạ đến $Line_{n-1}$ của cache.



Hình 45 Phương pháp ánh xạ trực tiếp bộ nhớ - cache

Có thể thấy với phương pháp ánh xạ trực tiếp, tại mọi thời điểm luôn có cố định m dòng bộ nhớ cùng cạnh tranh một dòng cache. Khi biết được địa chỉ của dòng trong bộ nhớ, ta biết vị trí của nó trong cache – vì thế phương pháp ánh xạ trực tiếp còn gọi là ánh xạ cứng hay ánh xạ cố định. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ trực tiếp gồm 3 thành phần: *Tag, Line* và *Word* như minh hoạ trên Hình 46. *Tag* (bit) là địa chỉ trang trong bộ nhớ chứa dòng được nạp vào cache, *Line* (bit) là địa chỉ dòng trong cache và *Word* (bit) là địa chỉ của từ trong dòng.



Hình 46 Địa chỉ ô nhớ trong ánh xạ trực tiếp

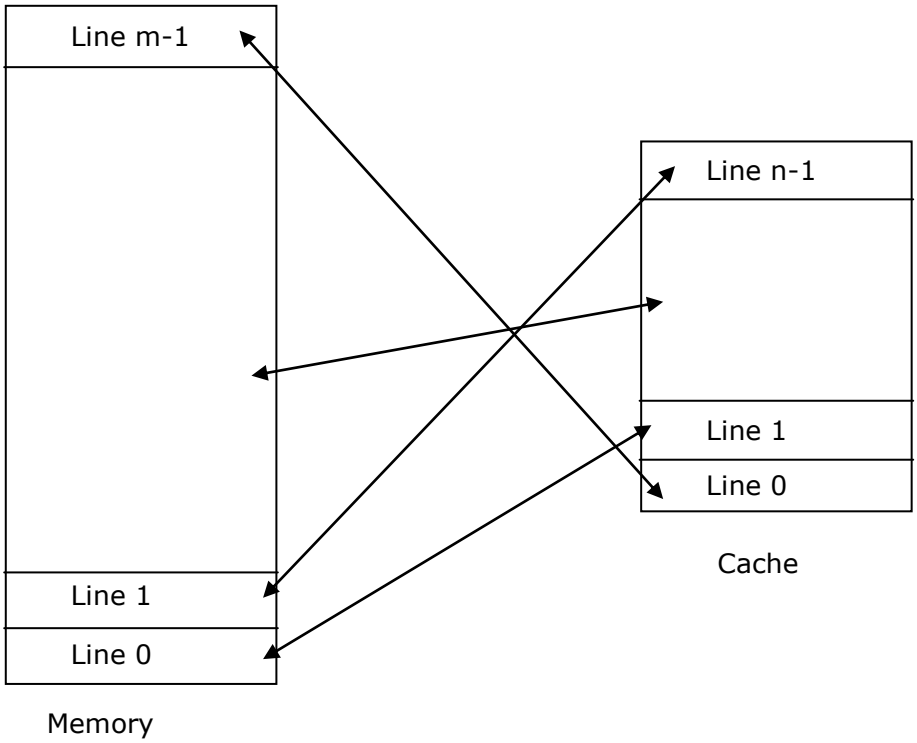
Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ trực tiếp:

- Vào:
  - Dung lượng bộ nhớ = 4GB
  - Dung lượng cache = 1MB
  - Kích thước dòng = 32 byte
- Ra:
  - Kích thước dòng Line = 32 byte = 2^5 , vậy Word = 5 bit
  - Dung lượng Cache = 1MB = 2^{10} $\rightarrow$ có 2^{10} / 2^5 = 2^5 dòng, vậy Line = 5 bit
  - Địa trang Tag: 4GB = 2^{32}, cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

    Tag = 32 bit địa chỉ – Line – Word = 32 – 5 – 5 = 22 bit.

Phương pháp ánh xạ trực tiếp có thiết kế đơn giản và rất nhanh do không tốn nhiều thời gian truy tìm địa chỉ ô nhớ trong cache. Do các ánh xạ là cố định, nên khi biết địa chỉ ô nhớ có thể tìm được vị trí của nó trong cache rất nhanh chóng. Tuy nhiên, cũng do ánh xạ cố định nên phương pháp này dễ gây xung đột vì có thể tạo ra nhiều dòng cache bị nút cổ chai trong quá trình hoạt động của cache. Có thể có nhiều dòng cache rảnh rỗi hay ít được sử dụng, nhưng cũng có nhiều dòng cache quá tải do bị nhiều dòng bộ nhớ cùng cạnh tranh. Cũng vì lý do dễ gây xung đột nên hiệu quả tận dụng không gian cache của phương pháp ánh xạ trực tiếp không cao và hệ số hit thấp.

4.4.4.3 Ánh xạ kết hợp đầy đủ

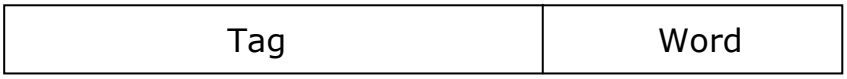


Hình 47 Phương pháp ánh xạ kết hợp đầy đủ bộ nhớ - cache

Phương pháp ánh xạ kết hợp đầy đủ hay còn gọi là ánh xạ liên kết đầy đủ được minh hoạ trên Hình 47. Cache được chia thành n dòng (line) đánh số từ 0 đến n-1. Bộ nhớ chính được chia thành m dòng (line), đánh số từ 0 đến m-1. Kích thước một dòng bộ nhớ bằng kích thước một dòng cache. Do bộ nhớ chính có kích thước lớn hơn nhiều kích thước cache, nên m >> n. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- Một dòng trong bộ nhớ chính có thể ánh xạ đến một dòng bất kỳ trong cache, hay
- $Line_i$ (i = 0 ÷ m-1) của bộ nhớ chính ánh xạ đến $Line_j$ (j = 0 ÷ n-1) của cache;

Có thể thấy với phương pháp ánh xạ kết hợp đầy đủ, có n dòng cache để lựa chọn ánh xạ – vì thế phương pháp ánh xạ kết hợp đầy đủ còn gọi là ánh xạ mềm hay ánh xạ không cố định. Ngược lại với phương pháp ánh xạ trực tiếp, khi biết được địa chỉ của dòng trong bộ nhớ, ta chưa biết vị trí của nó trong cache. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ kết hợp đầy đủ chỉ gồm 2 thành phần: *Tag* và *Word* như minh hoạ trên Hình 46. *Tag* (bit) là địa chỉ dòng trong bộ nhớ được nạp vào cache và *Word* (bit) là địa chỉ của từ trong dòng. Phần địa chỉ *Line* như trong địa chỉ ánh xạ trực tiếp bị bỏ do bộ nhớ chính chỉ còn là một trang duy nhất với m dòng.



Hình 48 Địa chỉ ô nhớ trong ánh xạ kết hợp đầy đủ

Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ kết hợp đầy đủ:

- Vào:
  - Dung lượng bộ nhớ = 4GB

- Dung lượng cache = 1MB
- Kích thước dòng = 32 byte

- Ra:
  - Kích thước dòng Line = 32 byte = 2^5 , vậy Word = 5 bit
  - Địa trang Tag: 4GB = 2^{32}, cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

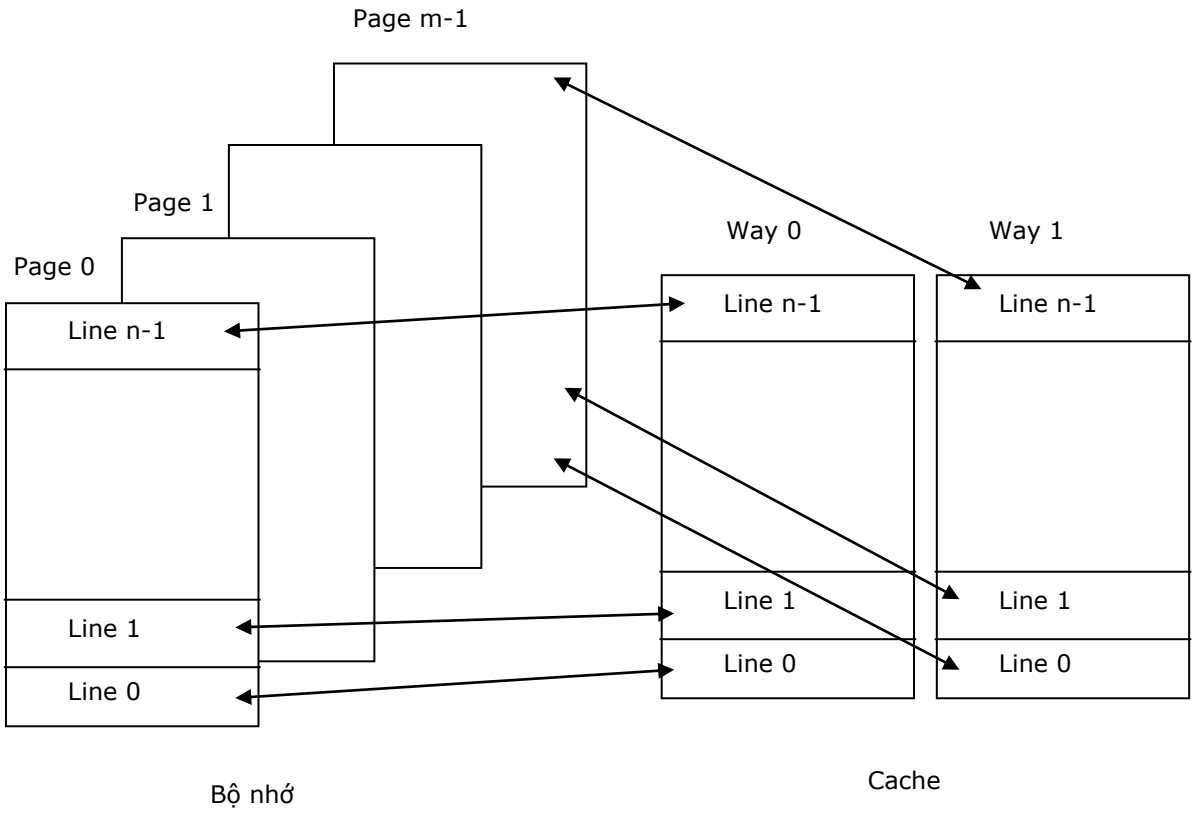
    Tag = 32 bit địa chỉ – Word = 32 – 5 = 27 bit.

Phương pháp ánh xạ kết hợp đầy đủ sử dụng ánh xạ không cố định nên có ưu điểm là mềm dẻo, giảm được xung đột sử dụng dòng cache. Việc sử dụng các dòng cache có thể được điều phối hướng đến phân bố hợp lý hơn, giảm hiện tượng tạo các dòng bị nút cổ chai với mức độ cạnh tranh lớn. Nhờ vậy, phương pháp này có hiệu suất sử dụng không gian cache cao hơn và có khả năng cho hệ số hit cao. Tuy nhiên, cũng do việc sử dụng ánh xạ không cố định, nên việc truy tìm địa dòng nhớ trong cache tốn nhiều thời gian, gây chậm trễ, đặc biệt với các cache có kích thước lớn. Ngoài ra, phương pháp này cũng có thiết kế phức tạp hơn so với phương pháp ánh xạ trực tiếp do cần bổ sung thêm các bộ so sánh địa chỉ dòng cache nhằm tăng tốc cho quá trình truy tìm địa dòng nhớ trong cache. Do vậy, phương pháp ánh xạ kết hợp đầy đủ chỉ thích hợp với các cache có dung lượng nhỏ.

4.4.4.4 Ánh xạ tập kết hợp

Phương pháp ánh xạ tập kết hợp hay còn gọi là ánh xạ liên kết nhóm được minh hoạ trên Hình 49. Cache được chia thành k đường (way) đánh số từ 0 đến k-1. Mỗi đường cache lại được chia thành n dòng (line) đánh số từ 0 đến n-1. Bộ nhớ chính được chia thành m trang (page), đánh số từ 0 đến m-1. Mỗi trang lại được chia thành n dòng (line) đánh số từ 0 đến n-1. Kích thước mỗi trang của bộ nhớ chính bằng kích thước một đường của cache và kích thước một dòng trong trang bộ nhớ cũng bằng kích thước một dòng của đường cache. Ánh xạ từ bộ nhớ chính vào cache được thực hiện theo quy tắc sau:

- Ánh xạ trang bộ nhớ đến đường cache (ánh xạ không cố định):
  - Một trang của bộ nhớ có thể ánh xạ đến một đường bất kỳ của cache.
- Ánh xạ dòng của trang đến dòng của đường (ánh xạ cố định):
  - $Line_0$ của $page_i$ của bộ nhớ ánh xạ đến $Line_0$ của way_j của cache;
  - $Line_1$ của $page_i$ của bộ nhớ ánh xạ đến $Line_1$ của way_j của cache;
  - ....
  - $Line_{n-1}$ của $page_i$ của bộ nhớ ánh xạ đến $Line_{n-1}$ của way_j của cache.



Hình 49 Phương pháp ánh xạ tập kết hợp bộ nhớ - cache

Có thể thấy phương pháp ánh xạ tập kết hợp đảm bảo được sự kết hợp hài hoà giữa ánh xạ mềm từ trang nhớ đến đường cache và ánh xạ cố định từ dòng của trang nhớ đến dòng của đường cache. Do số đường cache không lớn (thường chỉ khoảng 4, 8, 16, 32 hoặc 64 đường) nên việc tìm kiếm địa chỉ trang nhớ trong các đường cache không ảnh hưởng nhiều đến tốc độ truy cập cache. Hơn nữa, do ánh xạ từ dòng của trang nhớ đến dòng của đường cache là cố định, có thể nhanh chóng xác định được vị trí của dòng nhớ trong đường cache khi biết địa chỉ của nó. Để có thể quản lý được các ô nhớ được nạp, cache sử dụng địa chỉ ánh xạ trực tiếp gồm 3 thành phần: *Tag, Set* và *Word* như minh hoạ trên Hình 50. *Tag* (bit) là địa chỉ trang trong bộ nhớ chứa dòng được nạp vào cache, *Set* (bit) là địa chỉ dòng trong đường cache và *Word* (bit) là địa chỉ của từ trong dòng.

Hình 50 Địa chỉ ô nhớ trong ánh xạ tập kết hợp kết hợp

Ví dụ tính các thành phần địa chỉ ô nhớ trong ánh xạ tập kết hợp:

- Vào:
  - Dung lượng bộ nhớ = 4GB
  - Dung lượng cache = 1MB, 2 đường
  - Kích thước dòng = 32 byte
- Ra:

- Kích thước dòng Line = 32 byte = 2^5 , vậy Word = 5 bit
- Dung lượng Cache = 1MB = 2^{10} $\rightarrow$ có 2^{10} / 2 đường / 2^5 = 2^4 dòng / đường, vậy Set = 4 bit
- Địa trang Tag: 4GB = 2^{32}, cần 32 bít địa chỉ tổng cộng để địa chỉ hoá các ô nhớ:

  Tag = 32 bit địa chỉ – Set – Word = 32 – 4 – 5 = 23 bit.

Phương pháp ánh xạ tập kết hợp tận dụng được ưu điểm của cả hai phương pháp ánh xạ trực tiếp và ánh xạ kết hợp đầy đủ: nhanh do ánh xạ trực tiếp được sử dụng cho ánh xạ dòng - chiếm số lớn ánh xạ và mềm dẻo, ít xung đột do ánh xạ từ các trang bộ nhớ đến các đường cache là không cố định. Nhờ vậy, phân bố sử dụng không gian cache đồng đều hơn và đạt hệ số hit cao hơn. Nhược điểm lớn nhất của phương pháp này là có độ phức tạp thiết kế và điều khiển cao do cache được chia thành một số đường, thay vì chỉ một đường duy nhất.

4.4.5 Các phương pháp đọc ghi và các chính sách thay thế

4.4.5.1 Các phương pháp đọc ghi cache

Việc trao đổi thông tin giữa CPU – cache và giữa cache – bộ nhớ chính là một trong các vấn đề có ảnh hưởng lớn đến hiệu năng cache. Câu hỏi đặt ra là cần có chính sách trao đổi hay đọc ghi thông tin giữa các thành phần này như thế nào để đạt được hệ số hit cao nhất và giảm thiểu miss.

Xét trường hợp đọc thông tin và nếu đó là trường hợp hit (mẩu tin cần đọc có trong cache): mẩu tin được đọc từ cache vào CPU và bộ nhớ chính không tham gia. Như vậy thời gian CPU truy nhập mẩu tin bằng thời gian CPU truy nhập cache. Ngược lại, nếu đọc thông tin và đó là trường hợp miss (mẩu tin cần đọc không có trong cache): mẩu tin trước hết được chuyển từ bộ nhớ chính vào cache, sau đó nó được đọc từ cache vào CPU. Đây là trường hợp xấu nhất: thời gian CPU truy nhập mẩu tin bằng thời gian truy nhập cache cộng với thời gian cache truy nhập bộ nhớ chính – còn gọi là *miss penalty* (gấp đôi thời gian truy cập khi đoán trượt).

Với trường hợp ghi thông tin và nếu đó là trường hợp hit, có thể áp dụng một trong 2 chính sách ghi: *ghi thẳng* (write through) và *ghi trễ* (write back). Với phương pháp ghi thẳng, mẩu tin cần ghi được lưu đồng thời ra cache và bộ nhớ chính. Phương pháp ghi này luôn đảm bảo tính nhất quá dữ liệu giữa cache và bộ nhớ chính, nhưng có thể gây chậm trễ và tốn nhiều băng thông khi tần suất ghi lớn với nhiều mẩu tin có kích thước nhỏ. Ngược lại, với phương pháp ghi trễ, mẩu tin trước hết được ghi ra cache và dòng cache chứa mẩu tin sẽ được ghi ra bộ nhớ chính khi nó bị thay thế. Như vậy, mẩu tin có thể được ghi ra cache nhiều lần, nhưng chỉ được ghi ra bộ nhớ chính một lần duy nhất, giúp tăng tốc độ và giảm băng thông sử dụng. Phương pháp ghi trễ đang được ứng rộng rãi trong các hệ thống cache hiện nay.

Với trường hợp ghi thông tin và nếu đó là trường hợp miss, cũng có thể áp dụng một trong hai chính sách ghi: *ghi có đọc lại* (write allocate / fetch on write) và *ghi không đọc lại* (write non-allocate). Với phương pháp ghi có đọc lại, mẩu tin trước hết được ghi ra bộ nhớ chính, và sau đó dòng nhớ chứa mẩu tin vừa ghi được đọc vào cache. Việc đọc lại mẩu tin vừa ghi từ bộ nhớ chính vào cache có thể giúp giảm miss đọc kế tiếp áp dụng nguyên lý lân cận theo thời gian: mẩu tin vừa được truy nhập có thể được truy nhập lại trong tương lai gần. Với phương pháp ghi không đọc lại, mẩu tin chỉ được ghi ra bộ nhớ chính. Không có thao tác đọc dòng nhớ chứa mẩu tin vừa ghi vào cache.

4.4.5.2 Các chính sách thay thế dòng cache

Như đã đề cập, với cả ba phương pháp ánh xạ bộ nhớ chính – cache, luôn có nhiều dòng nhớ cùng ánh xạ đến một dòng cache. Do có nhiều dòng bộ nhớ chia sẻ một dòng cache, các dòng bộ nhớ được nạp vào cache sử dụng một thời gian và được thay thế bởi dòng nhớ khác theo yêu cầu thông tin phục vụ CPU. Các chính sách thay thế (replacement policies) xác định các dòng cache nào được chọn để thay thế bởi các dòng khác từ bộ nhớ nhằm đạt hệ số hit cao nhất. Có ba chính sách thay thế được sử dụng hiện nay: *thay thế ngẫu nhiên* (Random Replacement), *thay thế kiểu vào trước ra trước* (FIFO – First In First Out) và *thay thế các dòng ít được sử dụng gần đây nhất* (LRU – Least Recently Used).

Thay thế ngẫu nhiên là phương pháp đầu tiên được sử dụng do có thiết kế đơn giản và dễ cài đặt. Các dòng cache được lựa chọn để thay thế một cách ngẫu nhiên, không theo một quy luật nào. Do vậy, phương pháp thay thế ngẫu nhiên thường có hệ số miss cao do phương pháp này không xem xét đến các dòng cache đang thực sự được sử dụng. Nếu một dòng cache đang được sử dụng và bị thay thế sẽ xảy ra miss và nó lại cần được đọc từ bộ nhớ chính vào cache.

Trong phương pháp thay thế kiểu vào trước ra trước, các dòng nhớ được nạp vào cache trước sẽ được chọn để thay thế trước. Phương pháp này luôn có khuynh hướng loại bỏ các dòng cache có thời gian sử dụng lâu nhất, hay "già nhất". Nó có khả năng cho hệ số miss thấp hơn so với thay thế ngẫu nhiên do phương pháp này có xem xét đến yếu tố lân cận theo thời gian – các dòng nhớ có thời gian tồn tại trong cache lâu nhất có thể có xác suất được sử dụng thấp hơn. Tuy nhiên, phương pháp này vẫn chưa thực sự xem xét đến các dòng cache đang thực sự được sử dụng - một dòng cache "già" vẫn có thể đang được sử dụng. Một nhược điểm khác của thay thế kiểu vào trước ra trước là thiết kế và cài đặt phức tạp hơn, do cần phải có mạch mạch điện tử chuyên dụng để theo dõi trật tự nạp các dòng bộ nhớ vào cache.

Phương pháp thay thế các dòng ít được sử dụng gần đây nhất hoạt động theo nguyên tắc: các dòng cache được lựa chọn để thay thế là các dòng ít được sử dụng gần đây nhất. Phương pháp này cho hệ số miss thấp nhất so với thay thế ngẫu nhiên và thay thế FIFO, do thay thế LRU có xem xét đến các dòng đang thực sự được sử dụng – tuân theo yếu tố lân cận theo thời gian một cách chặt chẽ. Nhược điểm duy nhất của phương pháp này là thiết kế và cài đặt phức tạp hơn, do cần phải có mạch điện tử chuyên dụng để theo dõi tần suất sử dụng các dòng cache.

4.4.6 Hiệu năng cache và các yếu tố ảnh hưởng

4.4.6.1 Hiệu năng cache

Hiệu năng của cache (Cache Performance) có thể được đánh giá tổng thể theo thời gian truy nhập trung bình của CPU đến hệ thống nhớ. Thời gian truy nhập trung bình của một hệ thống nhớ có cache được tính như sau:

t_{access} = (Hit cost) + (miss rate) * (miss penalty)

$t_{access} = t_{cache} + (1 - H) * (t_{memory})$

trong đó, t_{access} là thời gian truy nhập trung bình hệ thống nhớ, t_{memory} là thời gian truy nhập bộ nhớ chính, t_{cache} là thời gian ruy nhập cache và H là hệ số hit.

Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=80%, ta có:

$t_{access} = 5 + (1 – 0.8) * (60) = 5+12 = 17ns$

Nếu t_{cache} = 5ns, t_{memory} = 60ns và H=95%, ta có:

t_{access} = 5 + (1 – 0.95) * (60) = 5+3 = 8ns

Như vậy, thời gian truy nhập trung bình hệ thống nhớ tiệm cận thời gian truy nhập cache với trường hợp cache đạt hệ số hit cao.

4.4.6.2 Các yếu tố ảnh hưởng

Có nhiều yếu tố ảnh hưởng đến hiệu năng cache, trong đó ba vấn đề (1) kích thước cache, (2) chia tách cache và (3) tạo cache nhiều mức có ảnh hưởng lớn nhất. Chúng ta lần lượt xem xét ảnh hưởng của từng yếu tố đến hiệu năng cache.

Vấn đề kích thước cache

Vấn đề kích thước cache liên quan đến việc trả lời câu hỏi: nên lựa chọn kích thước cache lớn hay nhỏ? Nhiều số liệu thống kê cho thấy, kích thước cache không ảnh hưởng nhiều đến hệ số miss và hệ số miss của cache lệnh thấp hơn nhiều so với cache dữ liệu:

8KB cache lệnh có hệ số miss nhỏ hơn 1%

256KB cache lệnh có hệ số miss nhỏ hơn 0.002%

như vậy, tăng kích thước cache lệnh không giảm miss hiệu quả.

8KB cache dữ liệu có hệ số miss nhỏ hơn 4%

256KB cache dữ liệu có hệ số miss nhỏ hơn 3%

như vậy, tăng kích thước cache dữ liệu lên 32 lần, hệ số miss giảm 25%.

Trên thực tế, xu hướng chung mong muốn kích thước cache càng lớn trong giới hạn cho phép của giá thành. Với kích thước lớn, có thể tăng được số dòng bộ nhớ lưu trong cache và nhờ vậy giảm tần suất tráo đổi các dòng cache của các chương trình khác nhau với bộ nhớ chính. Đồng thời, cache lớn hỗ trợ đa nhiệm, xử lý song song và các hệ thống CPU nhiều nhân tốt hơn do không gian cache lớn có khả năng chứa đồng thời thông tin của nhiều chương trình. Nhược điểm của cache lớn là chậm, do có không gian tìm kiếm lớn hơn cache nhỏ.

Vấn đề chia tách cache

Cache có thể được tách thành cache lệnh (I-Cache) và cache dữ liệu (D-Cache) để cải thiện hiệu năng, do:

- Dữ liệu và lệnh có tính lân cận khác nhau;
- Dữ liệu thường có tính lân cận về thời gian cao hơn lân cận về không gian; lệnh có tính lân cận về không gian cao hơn lân cận về thời gian;
- Cache lệnh chỉ cần hỗ trợ thao tác đọc; cache dữ liệu cần hỗ trợ cả 2 thao tác đọc và ghi và tách cache giúp tối ưu hoá dễ dàng hơn;
- Tách cache hỗ trợ nhiều lệnh truy nhập đồng thời hệ thống nhớ, nhờ vậy giảm xung đột tài nguyên cho CPU pipeline.

Vấn đề tạo cache nhiều mức

Khi cache được chia thành nhiều mức với kích thước tăng dần và tốc độ truy nhập giảm dần sẽ giúp cải thiện được hiệu năng hệ thống do hệ thống cache nhiều mức có khả năng dung hoà tốt hơn tốc độ của CPU với tốc độ của bộ nhớ chính.

Ví dụ: xem xét 2 hệ thống nhớ có số mức cache khác nhau: hệ thống 3 mức cache (L1, L2 và L3) và hệ thống 1 mức cache (L1). Giả thiết CPU có thời gian truy nhập là 1ns, các mức cache L1, L2, L3 có thời gian truy nhập lần lượt là 5ns, 15ns và 30ns. Bộ nhớ chính có thời gian truy nhập là 60ns.

	CPU	L1	L2	L3	Bộ nhớ chính
Cache 3 mức:	1ns	5ns	15ns	30ns	60ns
Cache 1 mức:	1ns	5ns			60ns

Có thể thấy hệ thống nhớ với nhiều mức cache có khả năng dung hoà tốc độ giữa các thành phần tốt hơn và có thời gian truy nhập trung bình hệ thống nhớ thấp hơn. Trên thực tế, đa số cache được tổ chức thành 2 mức: L1 và L2. Một số cache có 3 mức: L1, L2 và L3. Ngoài ra, nhiều mức cache có thể giúp giảm giá thành hệ thống nhớ.

4.4.7 Các phương pháp giảm miss cho cache

4.4.7.1 Các loại miss của cache

Một hệ thống nhớ với cache tốt cần đạt được các yếu tố: (1) hệ số hit cao, (2) hệ số miss thấp và (3) nếu xảy ra miss thì không quá chậm. Để có thể có giải pháp giảm miss hiệu quả, ta cần phân biệt rõ các loại miss. Cụ thể, tồn tại ba loại miss chính: *miss bắt buộc* (Compulsory misses), *miss do dung lượng* (Capacity misses) và *miss do xung đột* (Conflict misses). Miss bắt buộc thường xảy ra tại thời điểm chương trình được kích hoạt, khi mã chương trình đang được tải vào bộ nhớ và chưa được nạp vao cache. Miss do dung lượng lại thường xảy ra do kích thước của cache hạn chế, đặc biệt trong môi trường đa nhiệm. Do kích thước cache nhỏ nên mã của các chương trình thường xuyên bị tráo đổi giữa bộ nhớ và cache. Theo một khía cạnh khác, miss do xung đột xảy ra khi có nhiều dòng bộ nhớ cùng cạnh tranh một dòng cache.

4.4.7.2 Các phương pháp giảm miss cho cache

Trên cơ sở các loại miss đã được đề cập, hai phương pháp giảm miss có thể phối hợp áp dụng nhằm đạt hiểu quả giảm miss tối đa, gồm: *tăng kích thước dòng cache* và *tăng mức độ liên kết cache*. Biện pháp tăng kích thước dòng cache có thể giúp giảm miss bắt buộc do dòng có kích thước lớn sẽ có khả năng bao phủ các mục tin lân cận tốt hơn. Tuy nhiên, biện pháp này sẽ làm tăng miss xung đột, do dòng kích thước lớn sẽ làm giảm số dòng cache, dẫn đến tăng mức độ cạnh tranh của các dòng nhớ đến một dòng cache. Ngoài ra, dòng kích thước lớn có thể gây lãng phí dung lượng cache do có thể có nhiều phần của dòng cache lớn không bao giờ được sử dụng. Hiện nay, kích thước dòng cache thường dùng hiện nay là 64 bytes.

Biện pháp tăng mức độ liên kết cache hay tăng số đường cache có thể giúp giảm miss xung đột, do tăng số đường cache làm tăng tính mềm dẻo của ánh xạ trang bộ nhớ đến đường cache do có nhiều lựa chọn hơn. Tuy nhiên, nếu tăng số đường cache quá lớn, có thể làm cache chậm do tăng không giam tìm kiếm các đường cache. Hiện nay, số đường cache hợp lý cho miss tối ưu thường dùng là khoảng 8 đường.

4.5 CÂU HỎI ÔN TẬP

1. Hệ thống bộ nhớ phân cấp: đặc điểm, vai trò.
2. ROM là gì? các loại ROM.
3. RAM, SRAM, DRAM là gì? Cấu tạo của SRAM và DRAM.
4. Bộ nhớ cache:
   - Cache là gì? vai trò và nguyên lý hoạt động.
   - Kiến trúc cache
   - Tổ chức/ánh xạ cache
   - Đọc ghi thông tin trong cache
   - Các chính sách thay thế dòng cache
   - Hiệu năng cache và các yếu tố ảnh hưởng
   - Các biện pháp giam miss cho cache.

CHƯƠNG 5 BỘ NHỚ NGOÀI

5.1 ĐĨA TỪ

5.1.1 Giới thiệu

Đĩa từ (Magnetic Disks) là một trong các loại thiết bị lưu trữ được sử dụng rộng rãi nhất trong các thiết bị tính toán nói chung và các máy tính cá nhân nói riêng. Đĩa từ thuộc loại bộ nhớ ổn định – thông tin lưu trên đĩa từ luôn được duy trì, không phụ thuộc vào nguồn điện nuôi bên ngoài. Đĩa từ cũng là bộ nhớ kiểu khối có dung lượng lớn, đặc biệt là các đĩa cứng, dùng để lưu trữ thông tin lâu dài dưới dạng các tệp (files). Để lưu được thông tin, đĩa từ sử dụng các đĩa nhựa hoặc đĩa kim loại có phủ lớp bột từ trên bề mặt. Bột từ được sử dụng thường là oxit sắt hoặc các hợp kim của sắt.

Có hai dạng đĩa từ chủ yếu là đĩa từ mềm (gọi tắt là đĩa mềm – Floppy Disks) và đĩa từ cứng (gọi tắt là đĩa cứng – Hard Disks). Đĩa mềm làm bằng plastic, có dung lượng nhỏ, tốc độ chậm và dễ bị hư hỏng. Người ta sử dụng ổ đĩa mềm (FDD – Floppy Disk Drive) để đọc ghi đĩa mềm. Hình 51 minh hoạ đĩa mềm và ổ đĩa mềm dung lượng 1,44MB với kích thước đĩa 3,5 inches. Ngày nay, do sự phát triển mạnh mẽ của các loại đĩa quang và đặc biệt là các thẻ nhớ flash kết nối qua cổng USB, đĩa mềm ngày càng ít được sử dụng. Nhiều hệ thống máy tính lắp mới không đi kèm ổ đĩa mềm.

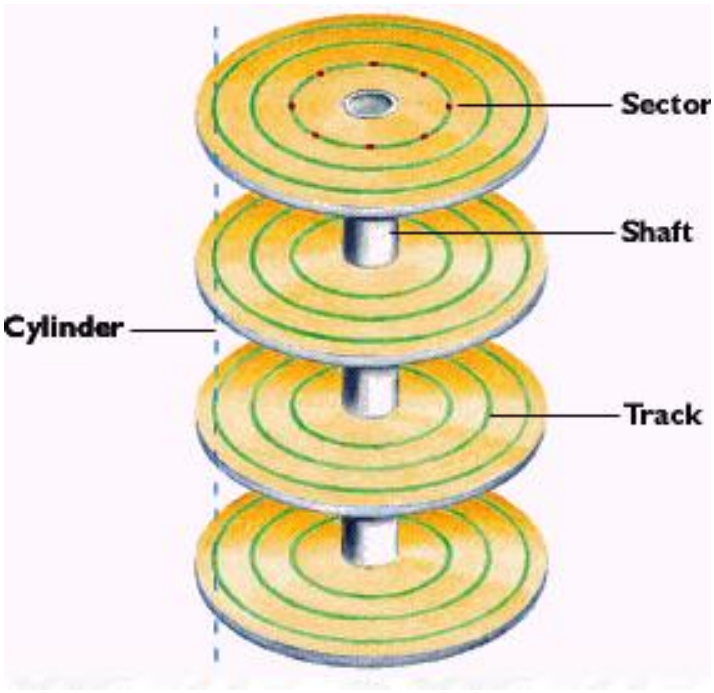
Hình 51 Đĩa mềm và ổ đĩa mềm kích thước 3,5 inches

Khác với đĩa mềm, đĩa cứng thường được gắn cố định trong ổ đĩa và được bọc trong một hộp kim loại bảo vệ như minh hoạ trên hình Hình 52. Đĩa cứng được làm bằng kim loại hoặc bằng thuỷ tinh, có dung lượng lớn và tốc độ cao hơn nhiều lần so với đĩa mềm. Hiện nay, các ổ đĩa cứng thường có dung lượng rất lớn, từ vài chục gigabyte đến hàng ngàn gigabyte và là thiết bị lưu trữ chủ yếu của các hệ thống máy tính. Do đĩa từ mềm ngày càng ít được sử dụng, phần tiếp theo của chương này chỉ đề cập đến đĩa từ cứng và ổ đĩa cứng.

Hình 52 Ổ đĩa cứng kích thước 3,5 inches

5.1.2 Đĩa cứng

5.1.2.1 Cấu tạo đĩa cứng



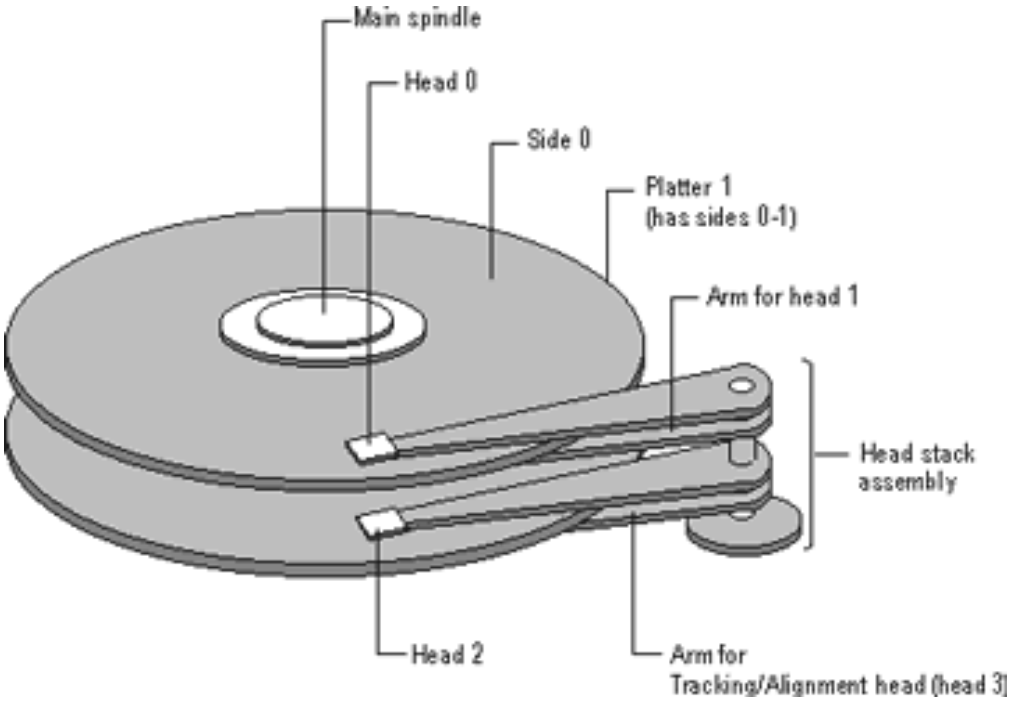
Hình 53 Các thành phần của đĩa cứng

Hình 53 minh hoạ cấu trúc của đĩa cứng và Hình 54 minh hoạ hệ thống đĩa và đầu từ đọc/ghi đĩa cứng. Đĩa cứng thường gồm các thành phần chính: các đĩa từ (Disks), các đầu từ đọc/ghi (Heads), các rãnh (Tracks), các mặt trụ (Cylinders) và các cung (Sectors).

Một ổ đĩa cứng có thể gồm một hoặc nhiều đĩa được lắp đồng trục. Các đĩa thường phẳng và được chế tạo bằng nhôm hoặc thuỷ tinh với lớp bột từ rất mỏng (khoảng 10-20nm) phủ trên bề mặt đĩa để lưu thông tin. Vật liệu từ thường dùng là oxit sắt ba (Fe_2O_3) với các ổ đĩa cứng cũ. Hiện nay vật liệu từ thường dùng là hợp kim của côban. Đĩa có thể lưu thông tin trên cả hai mặt (side), được đánh số mặt 0 và mặt 1.

Đầu từ hay đầu đọc ghi cũng là một trong các bộ phận chủ chốt của ổ đĩa cứng. Mỗi đầu từ đĩa cứng thường có kích thước rất nhỏ, được sử dụng để đọc và ghi thông tin lên đĩa. Khoảng

cách giữa đầu từ và bề mặt đĩa là rất nhỏ, nhưng không tiếp xúc mà "bay" trên mặt đĩa. Mỗi ổ đĩa cứng thường có nhiều đầu từ kết hợp thành một hệ thống đầu từ trên cùng một giá đỡ, như minh hoạ trên Hình 54. Số lượng đầu từ của mỗi ổ đĩa phụ thuộc vào thiết kế và dung lượng đĩa và thường rất khác nhau: 4, 8, 12, 16, 24, 32, 64...



Hình 54 Hệ thống đĩa và đầu từ đọc/ghi đĩa cứng

Rãnh có dạng là một đường tròn đồng tâm trên mặt đĩa để lưu thông tin. Các rãnh được đánh số từ 0 theo trật từ từ phía ngoài đĩa vào trong tâm và mỗi mặt đĩa có thể chứa hàng ngàn rãnh. Tiếp theo rãnh, mặt trụ là tập hợp của các rãnh ở các mặt đĩa khác nhau nằm trên cùng một vị trí đầu từ. Trên thực tế, mặt trụ là tham số được sử dụng nhiều hơn rãnh trong các hệ thống đĩa cứng.

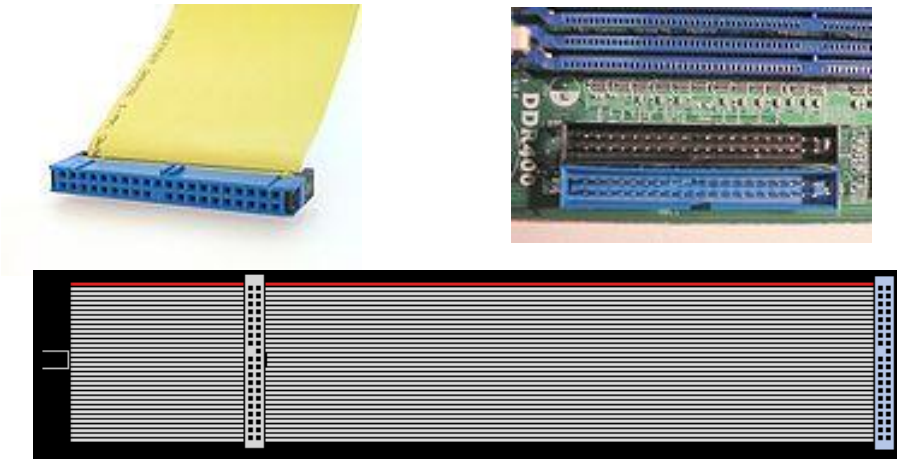
Cung là một phần của rãnh trên bề mặt đĩa và là đơn vị lưu trữ nhỏ nhất có thể quản lý của đĩa. Kích thước thông dụng của mỗi cung là 512 bytes. Với ổ đĩa cứng, ba tham số được sử dụng để tính dung lượng đĩa là: Số lượng mặt trụ (C), số lượng đầu từ (H) và số lượng cung trong một rãnh (S). Như vậy, dung lượng của đĩa cứng tính theo các tham số trên là:

Dung lượng của đĩa cứng = C x H x S x 512 bytes

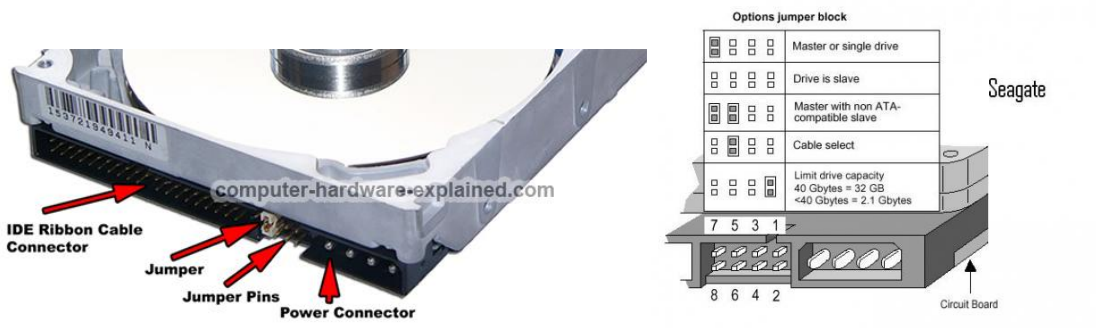
5.1.2.2 Các chuẩn ghép nối đĩa cứng

Các chuẩn hay giao diện ghép nối ổ đĩa cứng giải quyết vấn đề các ổ đĩa cứng được ghép nối và trao đổi dữ liệu với CPU như thế nào. Cho đến hiện nay, các giao diện thông dụng ghép nối ổ đĩa cứng với máy tính gồm: (1) Parallel ATA (PATA - Parallel Advanced Technology Attachments), còn gọi là ATA/IDE/EIDE (Integrated Drive Electronics), (2) Serial ATA (SATA), (3) SCSI – Small Computer System Interface (phát âm là scuzzy /skʌzi/), (4) Serial Attached SCSI (SAS) và (5) iSCSI – Internet SCSI. Trong tài liệu này, ta đề cập chi tiết ba chuẩn ghép nối thông dụng nhất cho máy tính là PATA/ATA/IDE, SATA và SCSI.

Chuẩn ghép nối ATA/IDE/PATA



Hình 55 Giao diện ghép nối và cáp ATA/IDE/PATA



Hình 56 IDE HDD jumpers & cài đặt jumpers

Chuẩn ghép nối ATA/IDE/PATA sử dụng cáp dẹt 40 hoặc 80 sợi để ghép nối ổ cứng với bảng mạch chính của máy tính. Mỗi cáp thường hỗ trợ ghép nối với 2 ổ đĩa: một ổ đĩa chủ (master) và một ổ đĩa tớ (slave). Băng thông đường truyền là 16 bít, đạt các mức thông lượng theo tần số làm việc: 16, 33, 66, 100 và 133MB/s. Hình 55 và Hình 56 minh hoạ khe cắm, cáp ghép nối và các chuyển mạch chế độ làm việc (jumpers) của ổ đĩa chuẩn ATA/IDE/PATA.

Chuẩn ghép nối SATA

Khe cắm
dữ liệu SATA

Đầu cắm
dữ liệu SATA

Đầu cắm
nguồn SATA

Hình 57 Khe cắm và cáp ghép nối SATA

Hình 57 minh hoạ các thành phần ghép nối ổ đĩa cứng với bảng mạch chính theo chuẩn SATA. Chuẩn SATA sử dụng cùng tập lệnh mức thấp như chuẩn ATA nhưng SATA sử dụng đường truyền tin nối tiếp tốc độ cao qua 2 đôi dây với bộ điều khiển SATA sử dụng chuẩn AHCI (Advanced Host Controller Interface). SATA hỗ trợ nhiều tính năng tiên tiến vượt trội so với ATA, như truyền dữ liệu nhanh và hiệu quả hơn và đặc biệt là tính năng cắm nóng (hot plug). SATA cung cấp tốc độ truyền dữ liệu cao hơn nhiều so với ATA. Với SATA thế hệ 1, tốc độ đạt 1,5 Gb/s và lần lượt đạt 3,0 Gb/s và 6,0 Gb/s với các thế hệ 2 và thế hệ 3.

Chuẩn ghép nối SCSI

SCSI là một tập các chuẩn về kết nối vật lý và truyền dữ liệu giữa máy tính và thiết bị ngoại vi, thường được sử dụng trong các máy chủ. Tất cả các thiết bị SCSI đều kết nối đến bus SCSI theo cùng một kiểu và mỗi bus SCSI có thể kết nối 8-16 thiết bị SCSI. Tương tự SATA, chuẩn SCSI cũng cung cấp nhiều tính năng tiên tiến như tốc độ truyền dữ liệu và tính ổn định rất cao và tính năng cắm nóng. Tính năng cắm nóng rất hữu dụng trong các máy chủ do SCSI cho phép thêm, bớt các ổ cứng mà không phải tắt máy, giảm thời gian ngừng cung cấp dịch vụ. SCSI đạt được tốc độ truyền dữ liệu: 5, 10, 20, 40MB/s với các ổ SCSI cũ và 160, 320, 640 MB/s với các ổ SCSI mới. Các ổ cứng SCSI thường rất đắt tiền và được thường được sử dụng cho các máy chủ và các hệ thống lưu trữ tiên tiến như RAID, NAS và SAN.

5.1.2.3 Quản lý đĩa cứng

Các đĩa cứng được quản lý theo hai mức: mức thấp (lower level) và mức cao (high level). Quản lý đĩa ở mức thấp được thực hiện bởi các chức năng của ROM-BIOS, đĩa được quản lý ở mức cao bởi hệ điều hành. Các vấn đề liên quan đến quản lý đĩa cứng gồm: định dạng đĩa cứng, phân khu và bảng phân khu đĩa cứng, cung khởi động, hệ thống file và thư mục gốc.

Định dạng đĩa cứng

Đĩa cứng cần được định dạng (format) trước khi sử dụng. Có hai mức định dạng đĩa cứng: định dạng mức thấp (lower level format) và định dạng mức cao (high level format). Định dạng mức thấp là quá trình gán địa chỉ cho các cung vật lý trên đĩa và có thể được thực hiện bởi các chức năng của BIOS. Hiện nay, hầu hết các ổ đĩa cứng đều đã được định dạng mức thấp khi xuất xưởng. Sau khi được định dạng mức thấp, ổ đĩa cần được định dạng ở mức cao bởi hệ điều hành trước khi có thể lưu thông tin. Định dạng mức cao là quá trình gán địa chỉ cho các cung logic và khởi tạo hệ thống file.

Phân khu và bảng phân khu đĩa cứng

Một đĩa cứng vật lý có thể được chia thành nhiều phần để thuận tiện cho quản lý và lưu trữ. Mỗi phần được gọi là một phân đoạn hay một phân khu (partition). Có hai loại phân khu: phân khu chính (primary partition) và phân khu mở rộng (extended partition). Thông thường, mỗi ổ đĩa chỉ có thể có một phân khu chính và một hoặc một số phân khu mở rộng. Một phân khu lại có thể được chia thành một hoặc một số ổ đĩa logic. Phân khu chính chỉ có thể chứa duy nhất một ổ đĩa logic, nhưng phân khu mở rộng có thể được chia thành một hoặc một số ổ đĩa logic.

Bảng phân khu (partition table) là một bảng gồm các bản ghi lưu thông tin quản lý các phân khu đĩa cứng. Các thông tin cụ thể về mỗi phân khu như sau:

- Phân khu có thuộc loại tích cực (active) ?
- Số mặt trụ (C), đầu từ (H) và cung (S) điểm bắt đầu phân khu;
- Số mặt trụ (C), đầu từ (H) và cung (S) điểm kết thúc phân khu;
- Kiểu định dạng phân khu (FAT, NTFS, EXT);
- Kích thước của phân khu tính theo số cung.

Cung khởi động

Cung khởi động (boot sector) là một cung đặc biệt, luôn nằm ở vị trí cung số 1 của ổ đĩa logic. Cung khởi động chứa chương trình mồi khởi động (Bootstrap loader) có nhiệm vụ kích hoạt việc nạp các thành phần của hệ điều hành từ đĩa vào bộ nhớ.

Hệ thống file

Hệ thống file (file system) là một dạng bảng danh mục (directory) để quản lý việc lưu trữ các files trên đĩa. Các files thường được lưu trữ trong các thư mục (folders) và các thư mục được tổ chức theo mô hình cây. Hệ thống file là một thành phần của hệ điều hành và có thiết kế khác khác nhau. Sau đây là một số hệ thống file thông dụng kèm theo hệ điều hành:

- FAT (DOS, Windows 3.x, Windows 95, 98, ME)
- NTFS (Windows NT, 2000, XP, 2003, Vista, 7)
- Ext2, Ext3 (Unix, Linux)
- MFS (Macintosh FS)/HFS (Hierarchical FS) (Mac OS)

Thư mục gốc

Thư mục gốc (Root directory) là thư mục ở mức thấp nhất trong hệ thống cây thư mục của ổ đĩa logic. Thư mục gốc là điểm bắt đầu khi hệ thống tìm kiếm và truy nhập file. Cũng như các thư mục khác, thư mục gốc có thể chứa các thư mục con và các file. Điểm khác biệt của thư mục gốc với các thư mục khác là nó không có thư mục mẹ.

5.2 ĐĨA QUANG

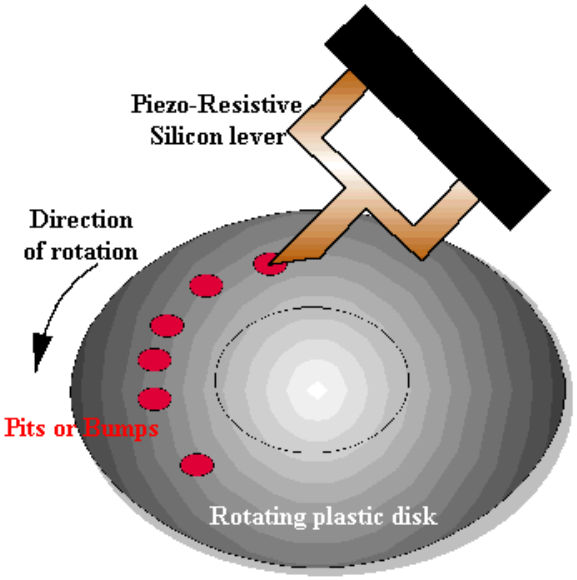
5.2.1 Giới thiệu và nguyên lý

Đĩa quang (Optical Disks) hoạt động dựa trên nguyên lý quang học: sử dụng ánh sáng để đọc và ghi thông tin trên đĩa. Các đĩa quang thường được chế tạo bằng plastic với một mặt được tráng một lớp nhôm mỏng để phản xạ tia laser. Mặt đĩa quang được "khắc" rãnh và mức lõm của rãnh được sử dụng để biểu diễn các bit thông tin, như minh hoạ trên Hình 58. Trên thực tế, các đĩa quang âm nhạc và phim được chế tạo hàng loạt theo kiểu chế bản in gồm 2 khâu: Trước hết, tạo bản đĩa chủ chứa thông tin ở dạng "âm bản" bằng thiết bị chuyên dụng, sau đó sử dụng bản đĩa chủ để "in" thông tin lên các đĩa quang trắng.

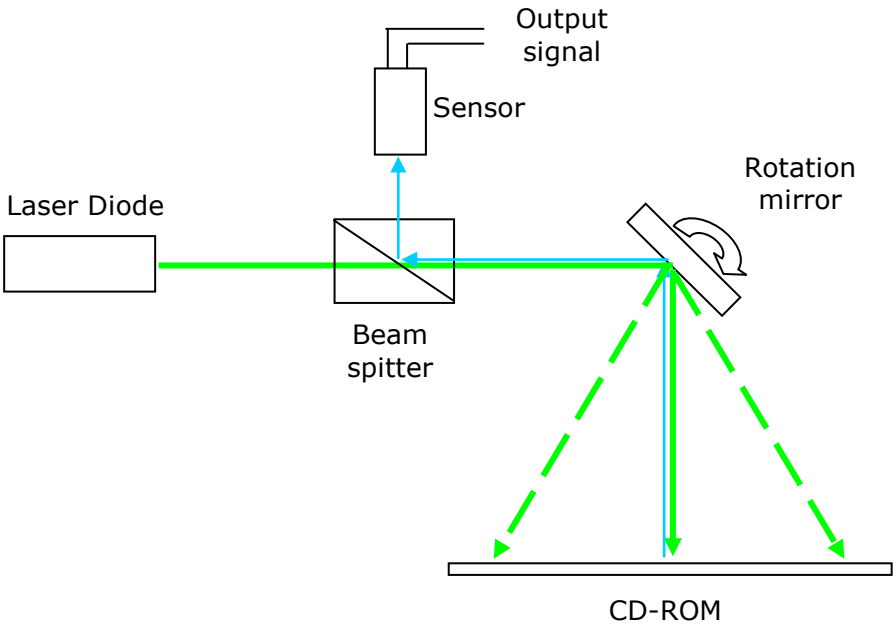
Việc đọc thông tin trên đĩa quang được thực hiện trong ổ đĩa quang (Optical Disk Drive), như minh hoạ trên Hình 59 theo các bước:

1. Tia laser từ điốt phát laser đi qua bộ tách tia đến gương quay;
2. Gương quay được điều khiển bởi tín hiệu đọc, lái tia laser đến vị trí cần đọc trên mặt đĩa;
3. Tia phản xạ từ mặt đĩa phản ánh mức lồi lõm trên mặt đĩa quay trở lại gương quay;

4. Gương quay chuyển tia phản xạ về bộ tách tia và sau đó đến bộ cảm biến quang điện;
5. Bộ cảm biến quang điện chuyển đổi tia laser phản xạ thành tín hiệu điện đầu ra. Cường độ của tia laser được biểu diễn thành mức tín hiệu ra.



Hình 58 Lưu thông tin trên đĩa quang



Hình 59 Nguyên lý đọc thông tin trên đĩa CD-ROM

5.2.2 Các loại đĩa quang

Có hai họ đĩa quang chính: đĩa CD (Compact Disk) và đĩa DVD (Digital Video Disk). Đĩa CD ra đời trước có dung lượng nhỏ, tốc độ chậm, thường được sử dụng để lưu dữ liệu, âm thanh và phim ảnh có chất lượng thấp. Đĩa DVD ra đời sau, có dung lượng lớn, tốc độ truy nhập cao và cho phép lưu dữ liệu, âm thanh và phim ảnh có chất lượng cao hơn.

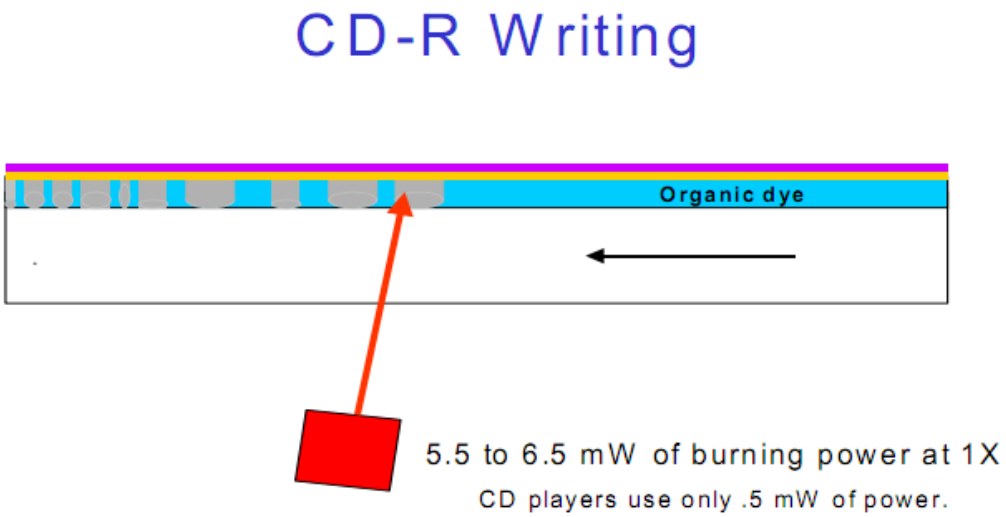
Họ đĩa CD gồm 3 loại chính: đĩa CD chỉ đọc (CD-ROM - Read Only CD), đĩa CD có thể ghi 1 lần (CD-R - Recordable CD) và đĩa CD có thể ghi lại (CD-RW - Rewritable CD). Đĩa CD-ROM được ghi sẵn nội dung từ khi sản xuất và chỉ có thể đọc ra trong quá trình sử dụng. CD-ROM thường được sử dụng để lưu âm nhạc và các phần mềm. Đĩa CD-R là đĩa có thể ghi một lần duy nhất bởi người sử dụng. Sau khi thông tin được ghi, đĩa trở thành loại chỉ đọc. Ngược lại, đĩa CD-RW cho phép xoá thông tin đã ghi và ghi lại nhiều lần. Đĩa CD-RW thường có giá thành cao và có thể ghi lại khoảng 1000 lần.

Tương tự họ CD, họ DVD cũng gồm nhiều loại: đĩa DVD chỉ đọc (DVD-ROM - Read Only DVD), đĩa có thể ghi 1 lần (DVD-R - Recordable DVD), đĩa có thể ghi lại (DVD-RW - Rewritable DVD), đĩa DVD mật độ cao (HD-DVD - High-density DVD) và đĩa DVD mật độ siêu cao (Blu-ray DVD - Ultra-high density DVD). DVD-ROM thường được sử dụng để lưu phim ảnh và các phần mềm có dung lượng lớn. Đĩa DVD-R là đĩa có thể ghi một lần duy nhất bởi người sử dụng. Sau khi thông tin được ghi, đĩa trở thành loại chỉ đọc. Ngược lại, đĩa DVD-RW cho phép xoá thông tin đã ghi và ghi lại nhiều lần. Đĩa HD-DVD và Blu-ray DVD là các loại đĩa DVD có dung lượng siêu cao với dung lương tương ứng vào khoảng 15GB và 25GB với đĩa một lớp.

5.2.3 Giới thiệu cấu tạo một số đĩa quang thông dụng

5.2.3.1 Đĩa CD-ROM, CD-R và CD-RW

Dung lượng tối đa của đĩa CD là 700MB hoặc 80 phút nếu lưu âm thanh. Ổ đĩa sử dụng tia laser hồng ngoại với bước sóng 780 nm để đọc thông tin. Tốc độ truyền thông tin của đĩa CD được tính theo tốc độ cơ sở (150KB/s) nhân với hệ số nhân. Ví dụ, đĩa có tốc độ đọc 4x thì tốc độ tối đa có thể đọc là 4 x 150KB/s = 600 KB/s; nếu đĩa có tốc độ đọc 50x thì tốc độ tối đa có thể đọc là 50 x 150KB/s = 7500 KB/s.



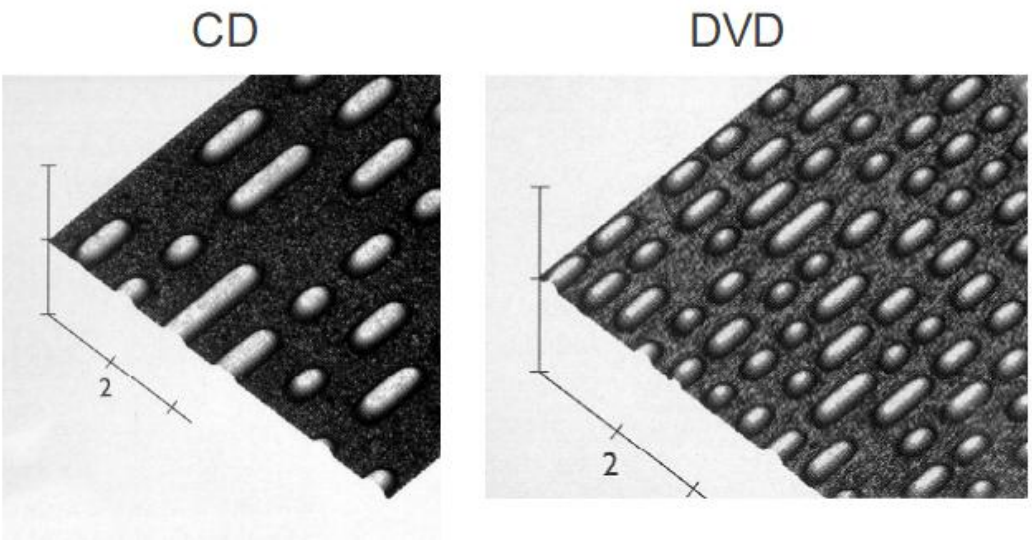
Hình 60 Cấu tạo đĩa CD-R

Đĩa CD-R về mặt hình thức và cấu tạo tương tự đĩa CD-ROM. Tuy nhiên, đĩa CD-R có thêm một lớp gọi là "organic dye", tạm dịch là *lớp hữu cơ* nằm giữa lớp plastic và lớp phản xạ bằng kim loại. Tia laser đã được điều chế bởi tín hiệu ghi được sử dụng để "đốt" lớp hữu cơ tạo thành các mức lồi lõm khác nhau trên lớp này để lưu thông tin. Sau khi đốt lớp hữu cơ bị cố

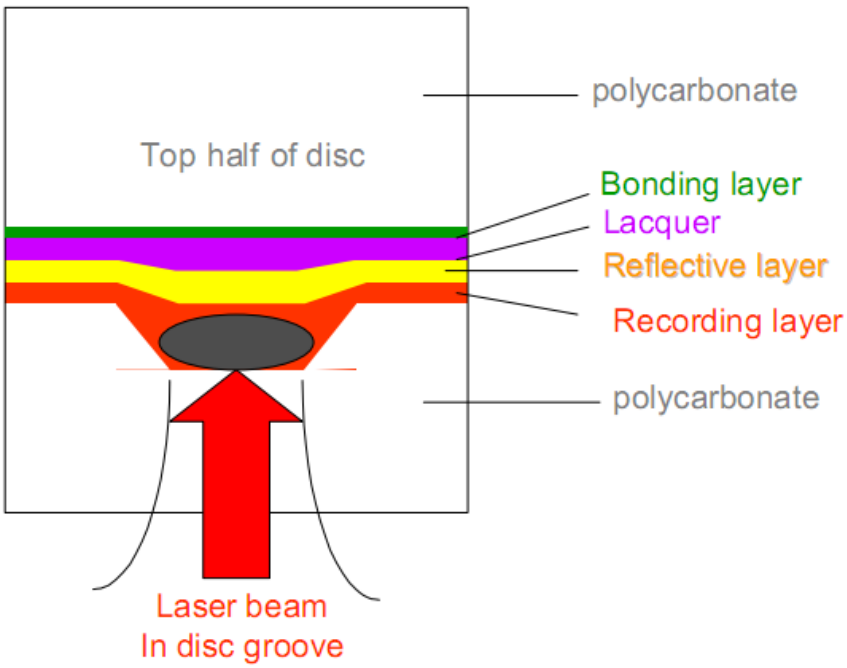
định và do vậy đĩa CD-R chỉ ghi được 1 lần. Trong đĩa CD-RW, lớp hữu cơ được thay bằng một lớp bán kim loại. Nhờ vậy, đĩa CD-RW có thể ghi được nhiều lần. Đa số các đĩa CD-RW cho phép ghi lại đến khoảng 1000 lần.

5.2.3.2 Đĩa DVD-ROM, DVD-R và DVD-RW

Dung lượng tối đa của đĩa DVD là 4,7GB với đĩa một mặt và 8,5GB với đĩa 2 mặt. Ổ đĩa DVD sử dụng tia laser hồng ngoại có bước sóng 650nm, ngắn hơn nhiều so với bước sóng tia laser dùng trong ổ đĩa CD. Nhờ sử dụng bước sóng laser ngắn hơn, đĩa DVD có mật độ ghi cao hơn nhiều so với CD, nhưng minh hoạ trên Hình 61. Tốc độ truyền thông tin của đĩa DVD được tính theo tốc độ cơ sở (1350KB/s) nhân với hệ số nhân. Ví dụ, đĩa có tốc độ đọc 4x thì tốc độ tối đa có thể đọc là 4 x 1350KB/s = 5400 KB/s; nếu đĩa có tốc độ đọc 16x thì tốc độ tối đa có thể đọc là 16 x 1350KB/s = 21600 KB/s.

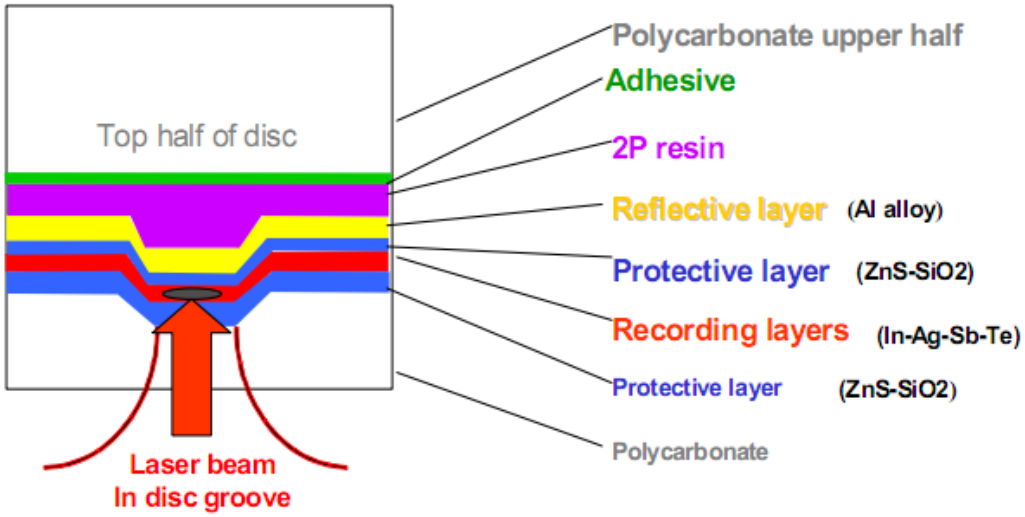


Hình 61 Mật độ ghi thông tin trên đĩa CD và DVD



Hình 62 Cấu tạo đĩa DVD-R

Đĩa DVD-R có cấu tạo tương tự đĩa CD-R, nhưng sử dụng tia laser có bước sóng ngắn hơn, là 650nm, như minh hoạ trên Hình 62. Hình 63 minh hoạ mặt cắt các lớp trong đĩa DVD-RW. Lớp bán kim loại để ghi thông tin được đặt trong hai lớp bảo vệ.



Hình 63 Cấu tạo đĩa DVD-RW

5.2.3.3 Đĩa HD-DVD và Blu-ray DVD

Đĩa HD-DVD và Blu-ray DVD là các "siêu" đĩa quang với dung lượng rất lớn và tốc độ truy nhập cao. Đĩa HD-DVD do Toshiba phát minh, sử dụng tia laser xanh với bước sóng rất ngắn. Đĩa HD-DVD đạt dung lượng 15GB cho một lớp và 30GB cho hai lớp. Do thất bại trong cạnh tranh với đĩa Blu-ray DVD, nên đĩa HD-DVD đã phải ngừng sản xuất từ tháng 2 năm 2008. Đĩa Blu-ray DVD do Sony phát minh, sử dụng tia laser với bước sóng 405nm. Đĩa Blu-ray DVD đạt dung lượng 30GB cho một lớp và 50GB cho hai lớp.

5.3 RAID

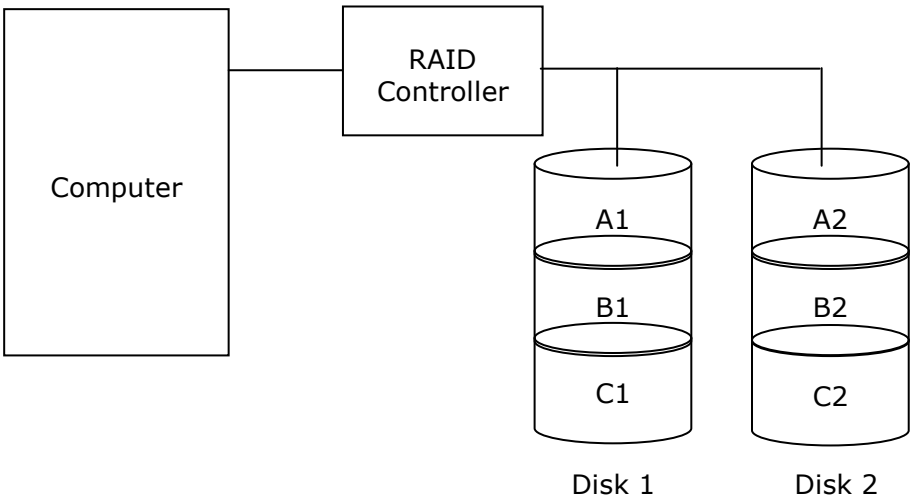
5.3.1 Giới thiệu RAID

RAID (Redundant Array of Independent Disks) là một công nghệ tạo các thiết bị lưu trữ tiên tiến trên cơ sở các ổ đĩa cứng, nhằm đạt các yêu cầu về tốc độ cao (high performance / speed), tính tin cậy cao (high reliability) và dung lượng lớn (large volume). Mặc dù RAID là một mảng của các ổ đĩa cứng, nhưng không phải tất cả các loại ổ cứng đều có thể sử dụng để tạo RAID. Trên thực tế, chỉ có các ổ cứng theo chuẩn SATA, SCSI và tương đương mới hỗ trợ tạo RAID.

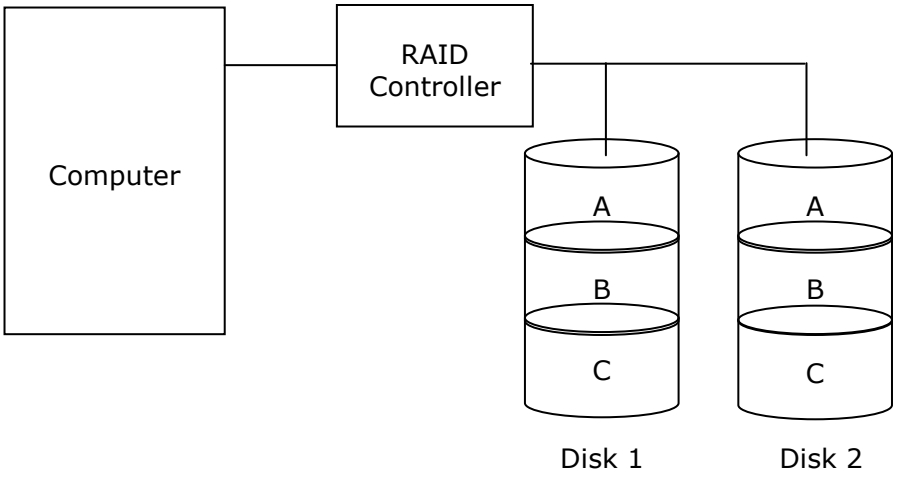
5.3.2 Các kỹ thuật tạo RAID

Có hai kỹ thuật chính được sử dụng để tạo RAID: kỹ thuật tạo lát đĩa (Disk Stripping) và kỹ thuật soi gương đĩa (Disk Mirroring). Hình 64 minh hoạ kỹ thuật tạo lát đĩa. Điểm mấu chốt của kỹ thuật này là điều khiển RAID cung cấp khả năng ghi và đọc song song các khối của cùng một đơn vị dữ liệu. Nhờ vậy tăng được tốc độ đọc ghi. Theo đó, các dữ liệu cần ghi được chia thành các khối cùng kích thước và được ghi đồng thời vào các ổ đĩa vật lý độc lập. Tương tự, trong quá trình đọc, các khối của dữ liệu cần đọc được đọc đồng thời từ các đĩa cứng độc lập, giúp giảm thời gian đọc.

Trong khi kỹ thuật tạo lát đĩa hướng đến tốc độ cao, kỹ thuật soi gương đĩa nhằm đạt độ tin cậy cao cho hệ thống lưu trữ. Hình 65 minh hoạ kỹ thuật soi gương đĩa. Theo đó, dữ liệu cũng được chia thành các khối và mỗi khối được ghi đồng thời lên hai hay nhiều ổ đĩa độc lập. Như vậy, tại mọi thời điểm ta đều có nhiều bản sao dữ liệu trên các đĩa cứng độc lập, đảm bảo tính an toàn cao.



Hình 64 RAID - Kỹ thuật tạo lát đĩa



Hình 65 RAID – Kỹ thuật soi gương đĩa

5.3.3 Giới thiệu một số loại RAID thông dụng

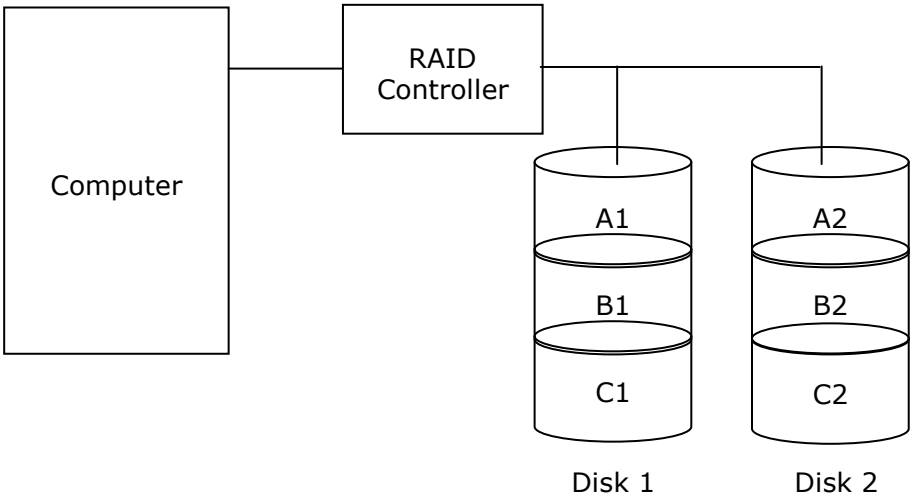
Trên cơ sở các kỹ thuật RAID được đề cập tại mục 5.3.2, trong mục này giới thiệu ba dạng RAID được sử dụng phổ biến nhất: RAID 0, RAID 1 và RAID 10. Các dạng RAID khác chẳng hạn RAID 2, 3, 4, 5, 6, 01, độc giả có thể tham khảo ở các tài liệu khác.

5.3.3.1 RAID 0

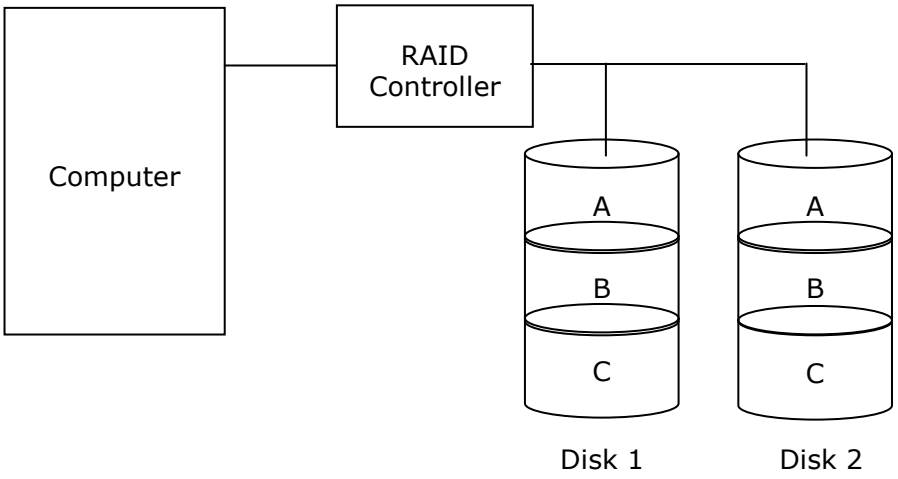
Cấu hình RAID 0 dựa trên kỹ thuật tạo lát đĩa và cần tối thiểu hai ổ đĩa vật lý, như minh hoạ trên Hình 66. Ưu điểm chính của RAID 0 là đạt tốc độ cao – tốc độ truy nhập RAID tỷ lệ thuận với số lượng đĩa độc lập của RAID. Ngoài ra, RAID 0 có thể giúp tăng dung lượng: dung lượng RAID 0 bằng tổng dung lượng của các đĩa độc lập tham gia. Hạn chế lớn nhất của RAID 0 là tính tin cậy – tính tin cậy của RAID 0 chỉ tương đương tính tin cậy của một ổ đĩa đơn.

5.3.3.2 RAID 1

Khác với RAID 0, cấu hình RAID 1 dựa trên kỹ thuật soi gương đĩa và cũng cần tối thiểu hai ổ đĩa vật lý, như minh hoạ trên Hình 67. Ưu điểm chính của RAID 1 là đạt độ tin cậy cao, do tại mỗi thời điểm luôn có nhiều bản sao lưu dữ liệu trên các đĩa độc lập. Tốc độ truy nhập và dung lượng của RAID 1 đều tương đương với một ổ đĩa đơn.



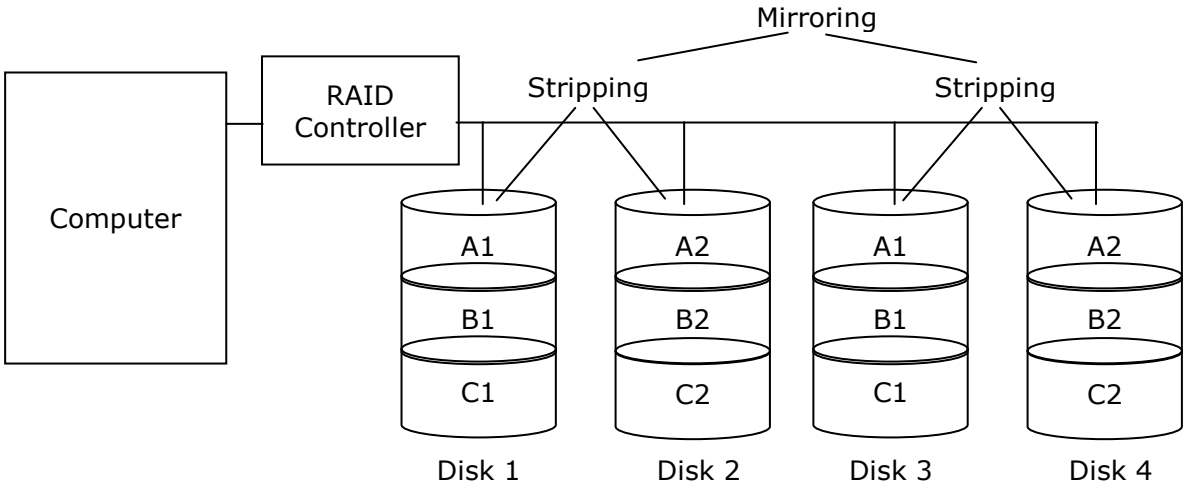
Hình 66 Cấu hình RAID 0



Hình 67 Cấu hình RAID 1

5.3.3.3 RAID 10

Cấu hình RAID 10 là sự kết hợp của RAID 1 và RAID 0, dựa trên cả hai kỹ thuật tạo lát đĩa và soi gương đĩa. RAID 10 cần tối thiểu 4 ổ đĩa độc lập như minh hoạ trên Hình 68. Ưu điểm của RAID 10 là đạt được cả tốc độ cao và tính tin cậy cao, nên rất phù hợp với các hệ thống máy chủ đòi hỏi tính an toàn cao, hiệu năng lớn như máy chủ cơ sở dữ liệu. Dung lượng RAID 10 bằng một nửa tổng dung lượng các đĩa độc lập tham gia tạo RAID. Nhược điểm duy nhất của RAID 10 là giá thành cao.



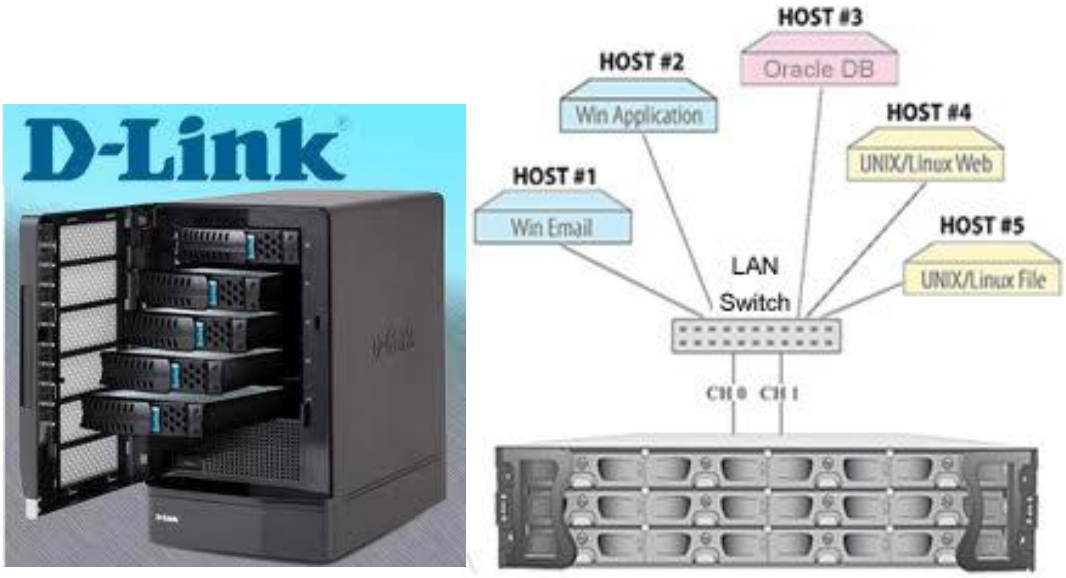
Hình 68 Cấu hình RAID 10

5.4 NAS

NAS (Network Attached Storage) là một dạng thiết bị lưu trữ được gắn trực tiếp vào mạng, thường là mạng cục bộ LAN. Hình 69 minh hoạ một thiết bị NAS USR8700 được gắn vào mạng LAN và cung cấp thiết bị lưu trữ cho cả mạng.

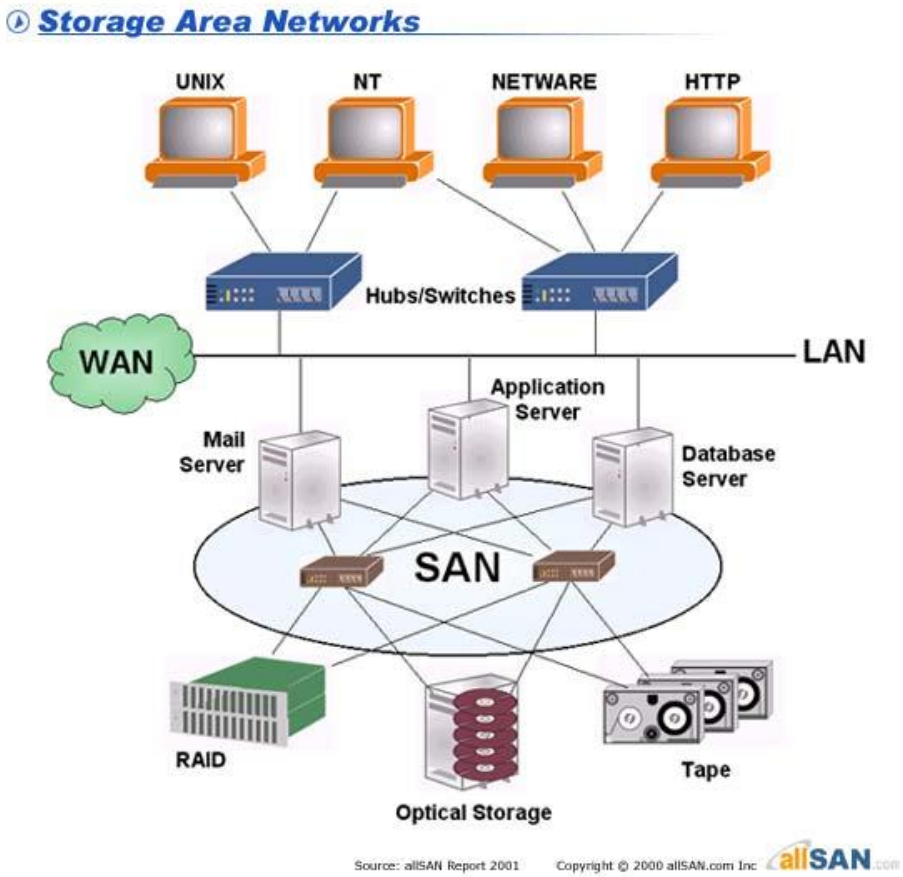
Hình 69 NAS trong một mạng LAN

Trên thực tế, NAS thường là một máy chủ chuyên dùng làm thiết bị lưu trữ, được kết nối vào mạng (thường là LAN tốc độ cao) và cung cấp các dịch vụ lưu trữ thông qua mạng. NAS thường dựa trên nền tảng là một RAID có tốc độ cao, dung lượng lớn và độ tin cậy rất cao. NAS có thể cung cấp dịch vụ lưu trữ cho hầu hết các loại máy chủ có cấu hình phần cứng khác nhau và chạy các hệ điều hành khác nhau, cũng như các phần mềm ứng dụng khác nhau, như minh hoạ trên Hình 70.



Hình 70 NAS của hãng D-Link và mô hình sử dụng trong mạng LAN

5.5 SAN



Hình 71 Mô hình mạng lưu trữ SAN và ứng dụng

SAN (Storage Area Network) là một mạng của các máy chủ chuyên dụng cung cấp dịch vụ lưu trữ, với các đặc điểm:

- Tốc độ truy nhập rất cao;
- Dung lượng cực lớn;
- Độ an toàn rất cao
- An toàn dữ liệu cục bộ
- An toàn dữ liệu với các bản copy được đồng bộ ở khoảng cách xa về địa lý.

Thông thường, các SAN thường được tổ chức dưới dạng các hệ thống file phân tán (Distributed File System) phục vụ tổ chức lưu trữ lượng dữ liệu khổng lồ cho các tổ chức và công ty lớn.

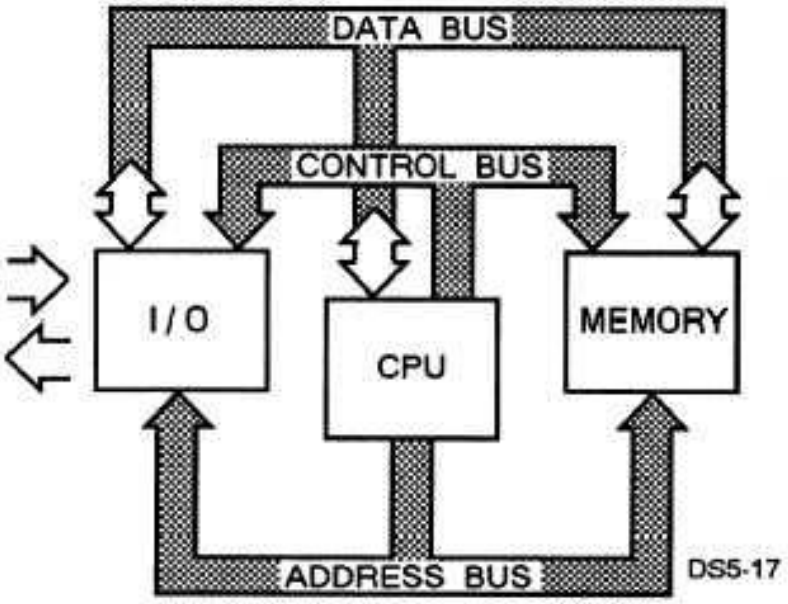
5.6 CÂU HỎI ÔN TẬP

1. Đĩa cứng: cấu tạo, các chuẩn ghép nối, bảng phân khu, thư mục gốc và hệ thống file.
2. Đĩa quang: cấu tạo, nguyên lý đọc CD và các loại đĩa quang.
3. RAID: RAID là gì? các kỹ thuật chính tạo RAID; các cấu hình RAID 0, 1 và 10.
4. Khái niệm về NAS.
5. Khái niệm về SAN.

CHƯƠNG 6 HỆ THỐNG BUS VÀ CÁC THIẾT BỊ NGOẠI VI

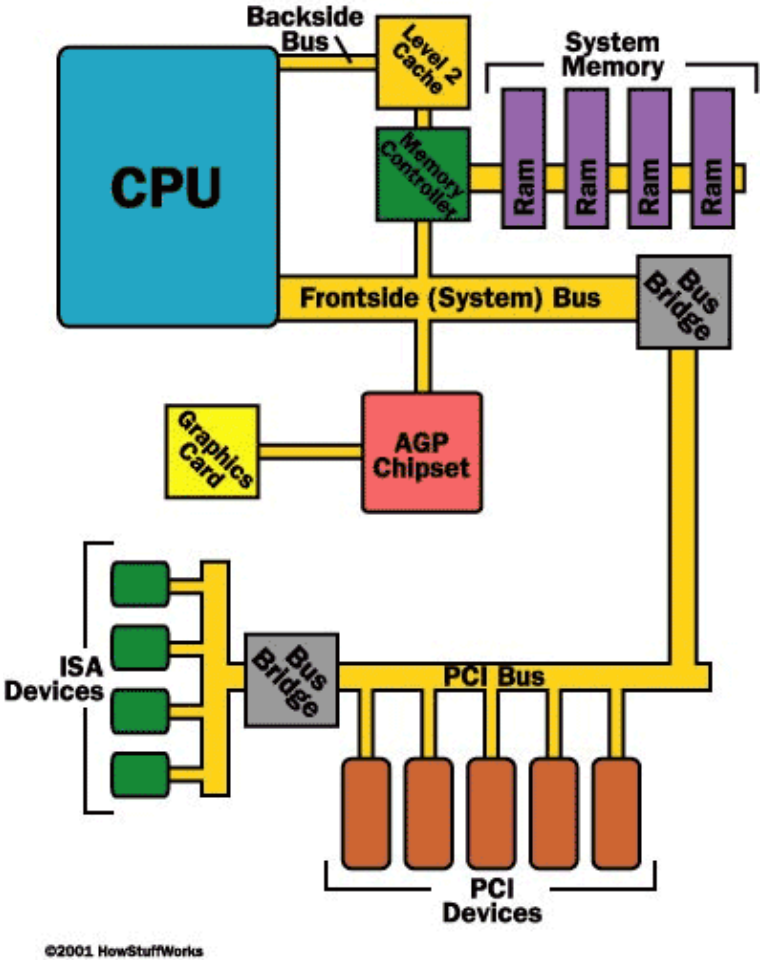
6.1 GIỚI THIỆU CHUNG VỀ HỆ THỐNG BUS

Bus là một hệ thống con (subsystem) có nhiệm vụ truyền dữ liệu giữa các bộ phận trong máy tính. Một hệ thống bus thường gồm ba thành phần: bus địa chỉ, bus dữ liệu và bus điều khiển. Bus địa chỉ (Address Bus – A Bus) là bus một chiều có nhiệm vụ truyền các tín hiệu địa chỉ phát hành bởi CPU đến bộ nhớ hoặc các thiết bị vào ra. Các tín hiệu địa chỉ giúp CPU chọn được ô nhớ cần đọc/ghi hoặc thiết bị vào ra cần trao đổi dữ liệu. Bus dữ liệu (Data Bus – D Bus) là bus hai chiều có nhiệm vụ truyền các tín hiệu dữ liệu đi và đến CPU. Dữ liệu được bus dữ liệu chuyển từ CPU đến bộ nhớ hoặc thiết bị vào ra và ngược lại. Bus điều khiển (Control Bus – C Bus) là bus một chiều theo một hướng, có nhiệm vụ truyền các tín hiệu điều khiển từ CPU đến bộ nhớ hoặc thiết bị vào ra, và truyền các tín hiệu trạng thái từ bộ nhớ hoặc thiết bị vào ra về CPU. Các bus địa chỉ, dữ liệu và điều khiển thường phối hợp cùng tham gia truyền dẫn các tín hiệu địa chỉ, dữ liệu và điều khiển trong quá trình CPU trao đổi thông tin với bộ nhớ hoặc các thiết bị vào ra.



Hình 72 Hệ thống bus nguyên lý

Hình 72 minh hoạ hệ thống bus nguyên lý – một hệ thống bus duy nhất kết nối ba thành phần quan trọng nhất của máy tính là CPU, bộ nhớ (memory) và các thiết bị vào ra (I/O). Trên thực tế, hệ thống bus thường được chia thành một số hệ thống bus con theo tần số làm việc và băng thông, nhằm làm cho hệ thống bus làm việc nhịp nhàng hơn với các thành phần có liên quan. Hình 73 minh hoạ hệ thống bus của các máy vi tính được sử dụng gần đây. Theo đó, hệ thống bus gồm các bus: Backside Bus (BSB), Frontside Bus (FSB), AGP Bus, PCI Bus và ISA Bus. BSB là bus riêng kết nối CPU với bộ nhớ cache, còn FSB kết nối CPU với bộ nhớ chính. AGP là bus dành riêng phục vụ card giao tiếp đồ hoạ và PCI bus thường được sử dụng để kết nối với các thiết bị ngoại vi. Bus ISA được sử dụng để kết nối với các thiết bị ngoại vi cũ. Các hệ thống bus con được kết nối với nhau thông qua các cầu bus (Bus Bridge).



Hình 73 Hệ thống bus thực tế

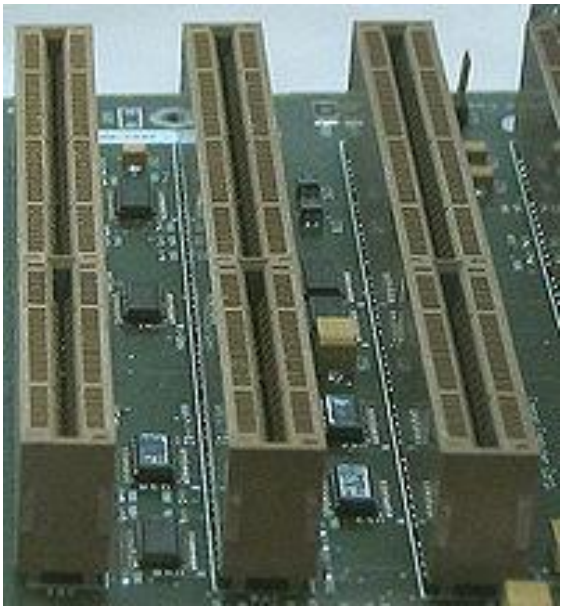
6.2 GIỚI THIỆU MỘT SỐ LOẠI BUS THÔNG DỤNG

6.2.1 Bus ISA và EISA

Bus ISA (Industrial Standard Architecture) là một trong các bus được phát triển sớm nhất. Bus ISA do IBM phát triển năm 1981 với băng thông 8 bit trên máy XT, hoặc 16 bit trên máy AT. ISA hỗ trợ tối đa 6 thiết bị kết nối đồng thời và hoạt động ở các xung nhịp 4, 6 và 8MHz. Hình 74 minh hoạ các khe cắm mở rộng của bus ISA được dùng để kết nối với các card mở rộng ISA.

Hình 74 Khe cắm mở rộng ISA

Bus EISA là một mở rộng của bus ISA ra đời vào năm 1988. EISA hỗ trợ băng thông 32 bits, nhưng nó vẫn tương thích với các thiết bị theo chuẩn ISA 8 và 16 bit. EISA hoạt động với xung nhịp 8.33MHz và đạt tốc độ truyền dữ liệu 33MB/s. Hình 75 minh hoạ các khe cắm mở rộng của bus EISA được dùng để kết nối với các card mở rộng ISA và EISA. Hiện nay, bus ISA và EISA đã lạc hậu và không còn được sử dụng.

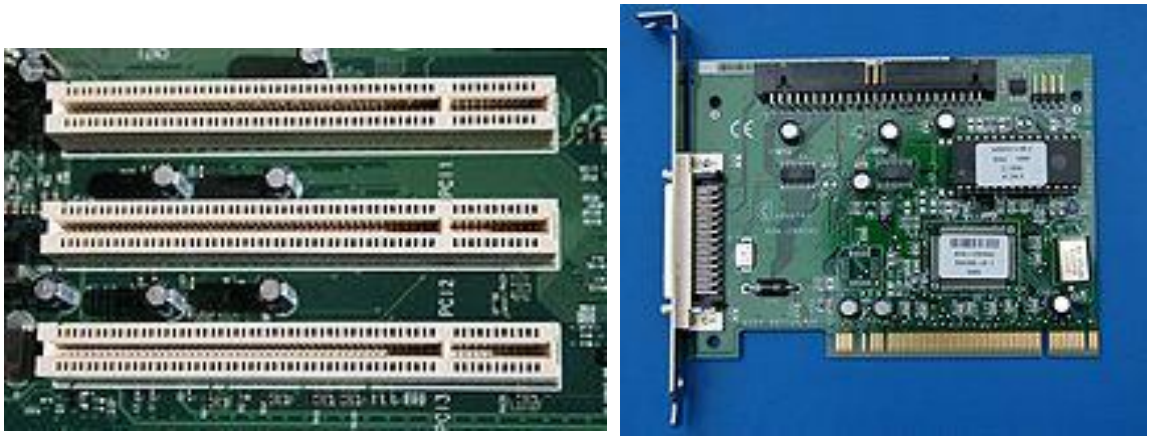


Hình 75 Khe cắm mở rộng EISA

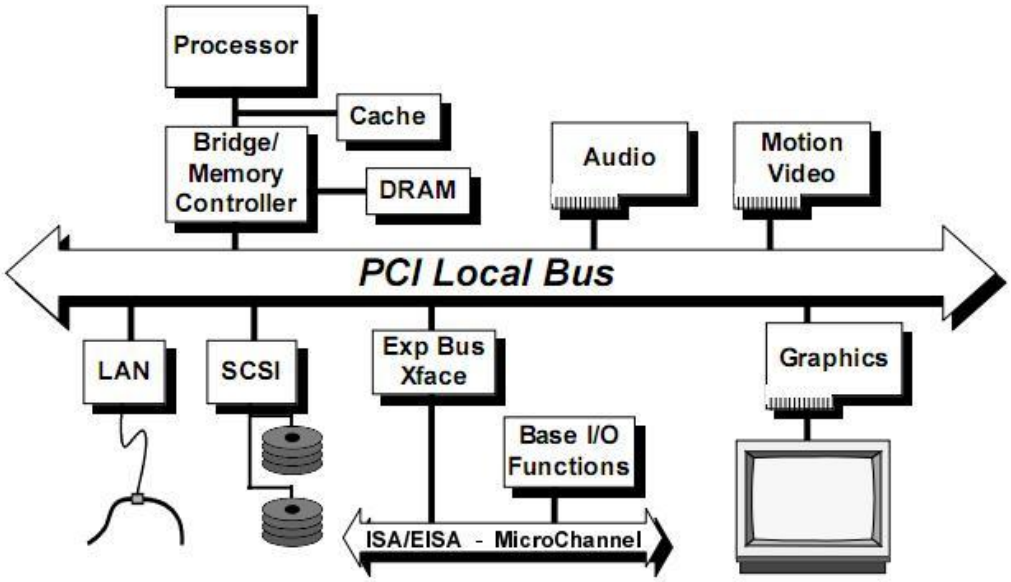
6.2.2 Bus PCI

6.2.2.1 Giới thiệu bus PCI

Bus PCI (Peripheral Component Interconnect) do Intel phát triển năm 1993 và được phát triển thành một trong các bus được sử dụng rộng rãi nhất cho đến ngày nay. PCI hỗ trợ băng thông 32 bit hoặc 64 bit và đạt tốc độ truyền dữ liệu khá cao theo tần số làm việc và băng thông. Với băng thông 32 bit, tốc độ truyền dữ liệu đạt 133 MB/s tại tần số 33MHz và 266 MB/s tại tần số 66MHz. Với băng thông 64 bit, tốc độ truyền dữ liệu đạt 266 MB/s tại tần số 33MHz và 533 MB/s tại tần số 66MHz. Hình 76 minh hoạ khe cắm PCI và card mở rộng thiết bị PCI và Hình 77 minh hoạ bus cục bộ PCI – các thành phần tham gia vào "gia đình" PCI.

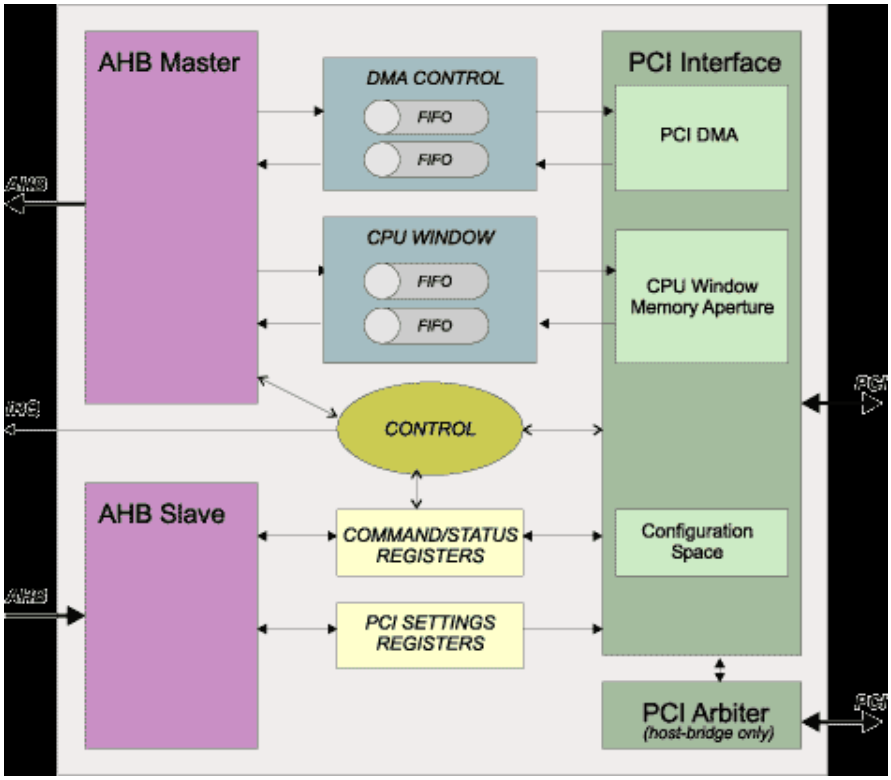


Hình 76 Khe cắm và card thiết bị PCI



Hình 77 Bus cục bộ PCI

6.2.2.2 Nguyên lý hoạt động của bus PCI



Hình 78 Sơ đồ khối nguyên lý hoạt động của bus PCI

Hình 78 nêu sơ đồ khối nguyên lý hoạt động của bus PCI. Theo đó PCI là một bus dùng chung hay bus chia sẻ (shared bus). PCI hỗ trợ nhiều thiết bị kết nối đồng thời, nhưng tại mỗi thời điểm, chỉ có một cặp thiết bị được sử dụng bus để trao đổi dữ liệu. Việc trao đổi dữ liệu trên bus PCI được thực hiện thông qua các *giao dịch* (transaction). Thiết bị khởi tạo (Initiator) quá trình truyền dữ liệu được gọi là *thiết bị chủ* (ABH Master) và thiết bị nhận dữ liệu hay

thiết bị đích (Target) là *thiết bị thợ* (ABH Slave). Một trọng tài có nhiệm vụ điều độ các giao dịch trên bus PCI được gọi là bộ tuỳ chọn (PCI Arbiter).

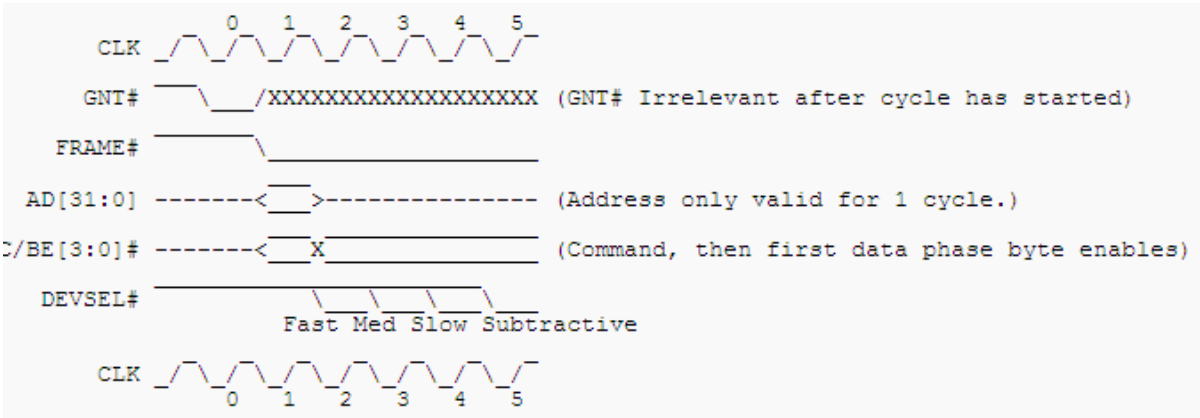
Việc thực hiện các giao dịch trên bus PCI được điều khiển bởi các tín hiệu. Hai nhóm tín hiệu chính được sử dụng, gồm: tín hiệu khởi tạo giao dịch và tín hiệu điều khiển giao dịch. Các tín hiệu khởi tạo một giao dịch, gồm tín hiệu REQ# do thiết bị khởi tạo giao dịch gửi tín hiệu yêu cầu sử dụng bus và tín hiệu GNT# do bộ tuỳ chọn gửi tín hiệu cho phép sử dụng bus. Các tín hiệu điều khiển một giao dịch, gồm tín hiệu FRAME# - bắt đầu chu kỳ bus, tín hiệu IRDY# - thiết bị khởi tạo đã sẵn sàng, tín hiệu DEVSEL# - thiết bị đích xác nhận bắt đầu giao dịch, tín hiệu TRDY# - thiết bị đích đã sẵn sàng và tín hiệu STOP# - dừng giao dịch.

Một giao dịch PCI được thực hiện theo 3 pha: pha tuỳ chọn (Arbitration), pha địa chỉ (Address) và pha dữ liệu (Data). Pha tuỳ chọn có nhiệm vụ khởi tạo giao dịch, pha địa chỉ xác định địa chỉ bên tham gia giao dịch và pha dữ liệu truyền dữ liệu giữa các bên. Pha tuỳ chọn được thực hiện thông qua các bước sau:

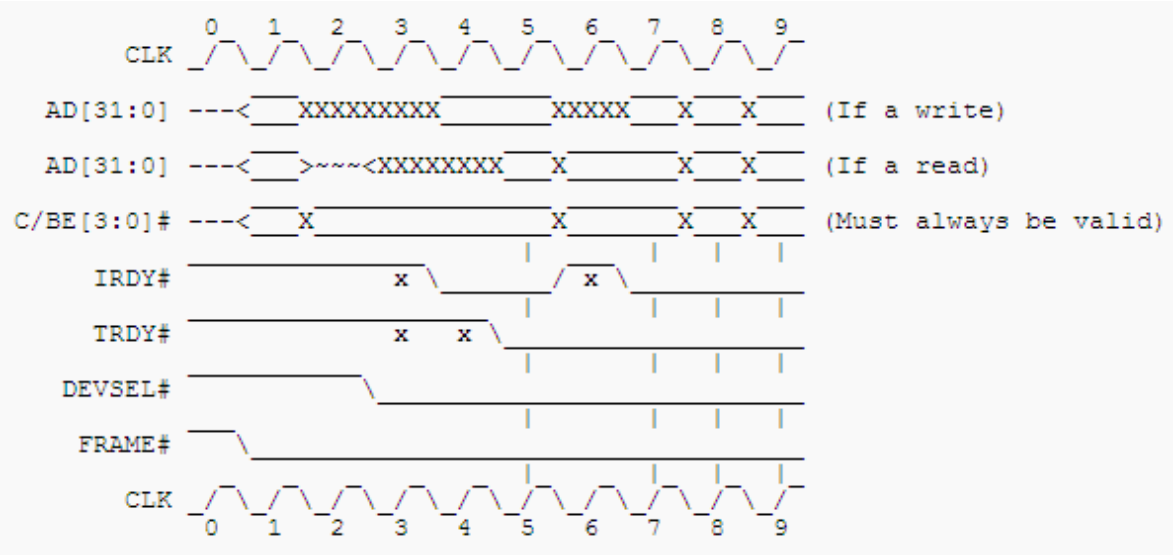
- Thiết bị PCI (Initiator) gửi tín hiệu REQ# đến Arbiter yêu cầu sử dụng bus;
- Nếu bus rỗi, Arbiter gửi tín hiệu cho phép sử dụng bus GNT# đến Initiator;
- Nếu bus bận, yêu cầu sử dụng bus được đưa vào hàng đợi;
- Tín hiệu cho phép sử dụng bus GNT# có thể bị Arbiter huỷ tại bất kỳ thời điểm nào;
- Thiết bị PCI được cấp tín hiệu cho phép sử dụng bus GNT# có thể bắt đầu phiên truyền dữ liệu nếu bus rỗi.

Pha địa chỉ của giao dịch như minh hoạ trên Hình 79, có thể gồm các bước:

- Thiết bị PCI (Initiator) có tín hiệu cho phép sử dụng bus GNT# có thể bắt đầu một giao dịch PCI bằng việc gửi tín hiệu FRAME# và gửi địa chỉ thiết bị đích cùng các lệnh liên quan (Read/Write);
- Mỗi thiết bị PCI sẽ kiểm tra địa chỉ và lệnh kèm theo để xác định mình có phải là thiết bị đích hay không. Thiết bị đích (có địa chỉ trùng với địa chỉ gửi bởi Initiator) sẽ gửi tín hiệu trả lời DEVSEL# đến Initiator;
- Thiết bị đích phải gửi tín hiệu trả lời DEVSEL# trong thời gian 3 chu kỳ đồng hồ.



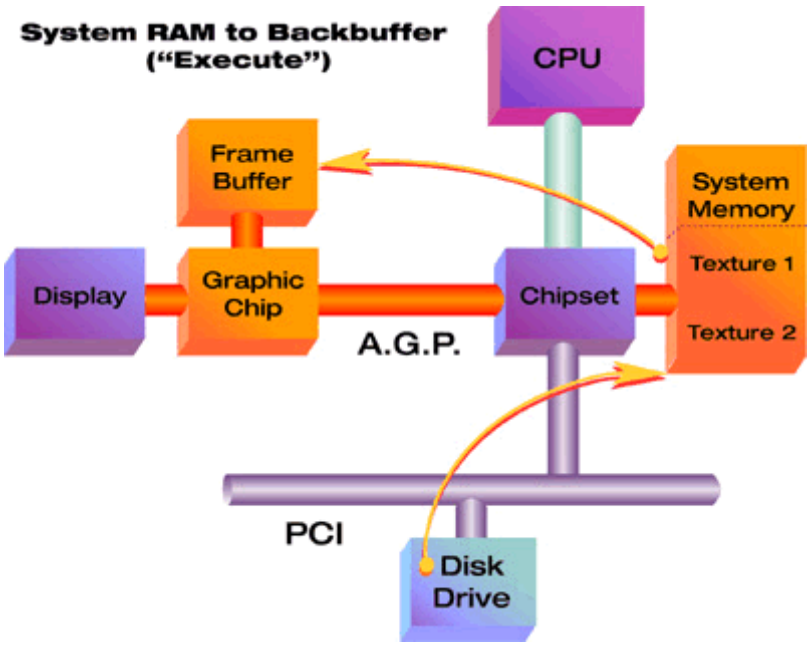
Hình 79 Pha địa chỉ giao dịch PCI



Hình 80 Pha dữ liệu giao dịch PCI

Hình 80 minh hoạ các tín hiệu trong pha dữ liệu của giao dịch PCI. Sau pha địa chỉ, khi tín hiệu DEVSEL# ở mức thấp là một hoặc một số pha dữ liệu. Kết thúc pha dữ liệu, thiết bị đích gửi tín hiệu STOP#.

6.2.3 Bus AGP

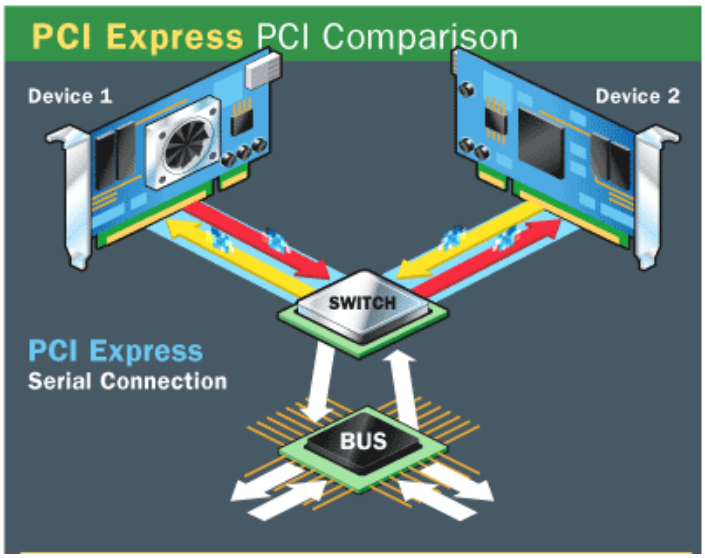


Hình 81 Sơ đồ nguyên lý hoạt động của AGP

Bus AGP (Accelerated Graphic Port) do Intel phát triển năm 1993 với mục đích chính sử dụng cho kết nối với các mạch xử lý đồ hoạ tốc độ cao. AGP đã hoàn toàn thay thế PCI trong lĩnh vực giao tiếp đồ hoạ trong các năm sau đó. AGP hỗ trợ băng thông 32 bít với tốc độ truyền dữ liệu nhanh gấp nhiều lần so với bus PCI. Cụ thể, AGP hỗ trợ 4 cấp tốc độ truyền dữ liệu là 1x, 2x, 4x và 8x, với tốc độ lần lượt là 266MB/s, 533MB/s, 1066MB/s và 2133MB/s tại các tần số tương ứng 66MHz, 133MHz, 266MHz và 533MHz.

6.2.4 Bus PCI Express

Bus PCI Express (còn gọi là PCIe) do Intel phát triển năm 2004, là một dạng bus truyền dữ liệu nối tiếp, kiểu điểm đến điểm (point to point) với tốc độ cao. Độ rộng bus là từ 1-32 bit tuỳ theo cấu hình. PCI Express được cấu trúc từ các liên kết nối tiếp điểm đến điểm và một cặp liên kết nối tiếp (theo 2 chiều ngược nhau) tạo thành một luồng (lane). Các luồng được định tuyến đồng thời qua một bộ chuyển mạch (crossbar switch). Tối đa, bus PCI Express có thể hỗ trợ đến 32 luồng. Tốc độ truyền dữ liệu của bus PCI Express phụ thuộc số luồng sử dụng và phiên bản của chuẩn. Với một luồng, tốc độ truyền đạt 250MB/s, 500MB/s và 1GB/s tương ứng với các phiên bản 1.x, 2.0 và 3.0.



Hình 82 Truyền dữ liệu qua bộ Switch trong PCI Express

Khác với PCI là bus chia sẻ, bus PCI Express có khả năng cung cấp đường truyền riêng cho các cặp thiết bị tham gia sử dụng bus. Đồng thời PCI Express cũng hỗ trợ nhiều cặp thiết bị cùng tham gia truyền dữ liệu sử dụng các luồng truyền khác nhau. Hình 82 minh hoạ việc truyền dữ liệu qua bộ chuyển mạch (Switch) trong PCI Express.

6.3 GIỚI THIỆU CHUNG VỀ CÁC THIẾT BỊ NGOẠI VI

6.3.1 Giới thiệu chung

Các thiết bị ngoại vi (peripheral devices) hay còn gọi là thiết bị vào ra, là các bộ phận của hệ thống máy tính có nhiệm vụ: (1) tiếp nhận các thông tin từ thế giới bên ngoài đi vào máy tính và (2) kết xuất các thông tin từ máy tính ra thế giới bên ngoài. Nhiệm vụ (1) được đảm bảo bởi nhóm các thiết bị vào (input devices) và nhiệm vụ (2) được đảm bảo bởi nhóm các thiết bị ra (output devices). Các thiết bị vào gồm có: bàn phím, chuột, ổ đĩa (đọc thông tin), máy quét ảnh và máy đọc mã vạch. Hình 83 minh hoạ thiết bị vào chuẩn là bàn phím và chuột. Các thiết bị ra gồm có: màn hình, máy in, ổ đĩa (ghi thông tin) và máy vẽ. Hình 84 minh hoạ hai loại màn hình thông dụng: màn hình CRT và LCD. Hình 85 minh hoạ các máy in laser và máy in phun mực.

Hình 83 Bàn phím và chuột

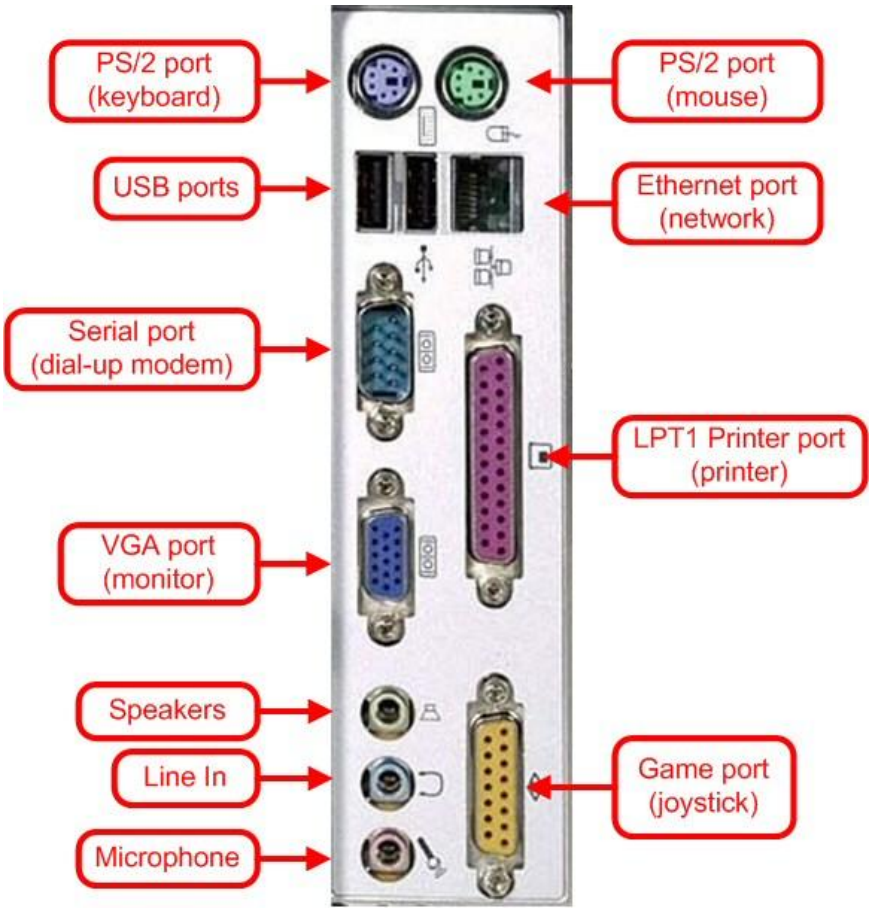
Hình 84 Màn hình CRT và LCD

Hình 85 Máy in Laser và máy in phun mực

6.3.2 Các cổng giao tiếp

Các thiết bị vào ra thường kết nối với máy tính thông qua các cổng giao tiếp (communication ports). Mỗi cổng giao tiếp được gán một địa chỉ và có tập tham số làm việc riêng. Hình 86 minh hoạ các cổng giao tiếp ở phía sau máy tính. Các cổng giao tiếp thông dụng:

1. PS/2: kết nối chuột và bàn phím.
2. Cổng COM và LPT.
3. Cổng USB: cổng giao tiếp đa năng theo chuẩn USB
   - USB 1.0: 12Mb/s
   - USB 2.0: 480Mb/s (hiện tại)
   - USB 3.0: 1.5Gb/s (tương lai).
4. Cổng IDE, SATA và E-SATA: ghép nối các loại ổ đĩa.
5. Cổng LAN: ghép nối mạng.
6. Cổng Audio: ghép nối âm thanh.
7. Cổng đọc các thẻ nhớ.
8. Cổng Firewire /IEEE 1394: ghép nối các loại ổ đĩa ngoài.
9. Cổng VGA/Video: ghép nối với màn hình.
10. Cổng DVI: ghép nối với màn hình số.

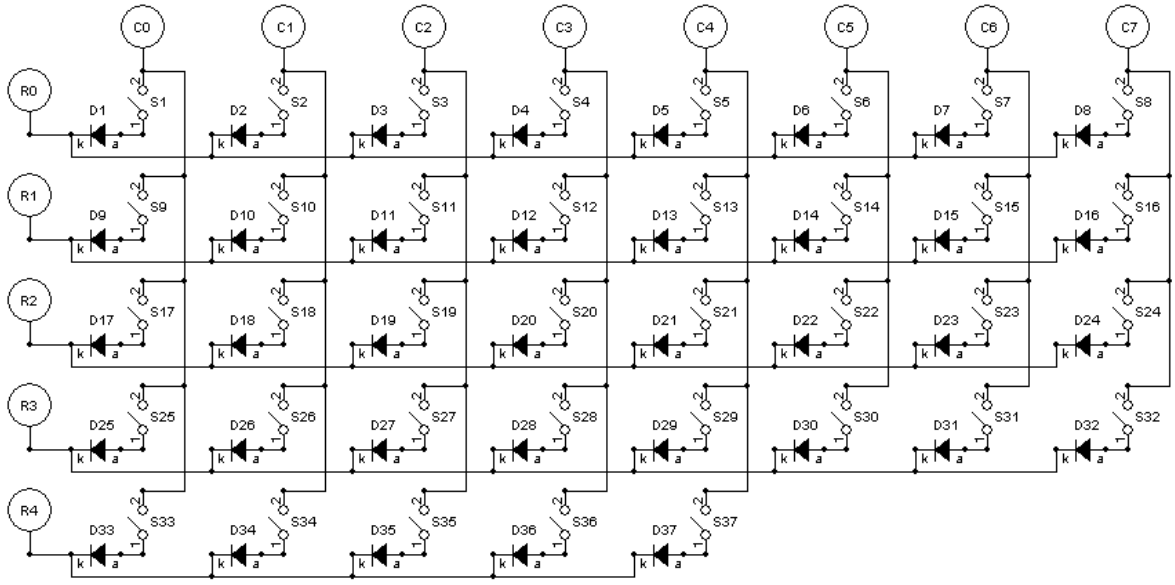


Hình 86 Một số cổng giao tiếp với máy tính

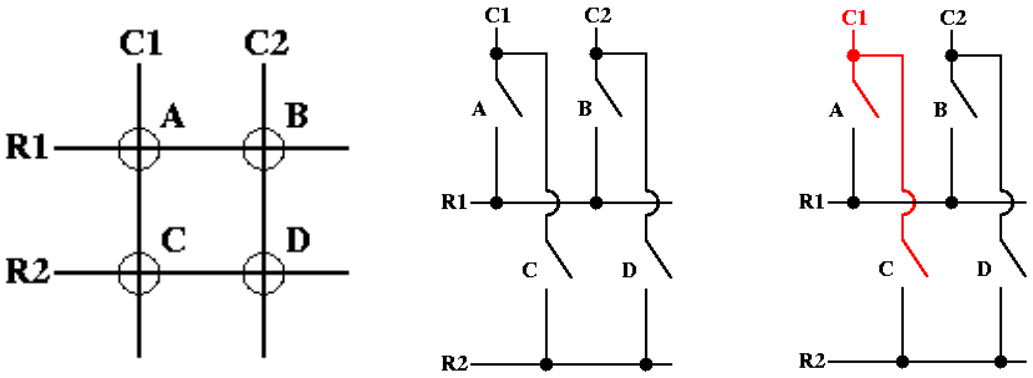
6.4 GIỚI THIỆU MỘT SỐ THIẾT BỊ VÀO RA THÔNG DỤNG

6.4.1 Bàn phím

Bàn phím (keyboard) là thiết bị vào chuẩn của máy tính do bàn phìm có thể đảm nhiệm cả chức năng nhập dữ liệu và điều khiển máy tính. Bàn phím tiêu chuẩn có 101 phím: các phím ký tự (a-z), các phím số (0-9), các phím phép toán (+, - *, /), các phím chức năng (F1-F12), các phím điều khiển (Ctrl, Alt, Shift, ..) và các phím di chuyển: Home, End, Page Up, Page Down, Up, Down, Left, Right, ...



Hình 87 Mạch tạo phím



Hình 88 Ma trận phím và phát hiện các phím được nhấn

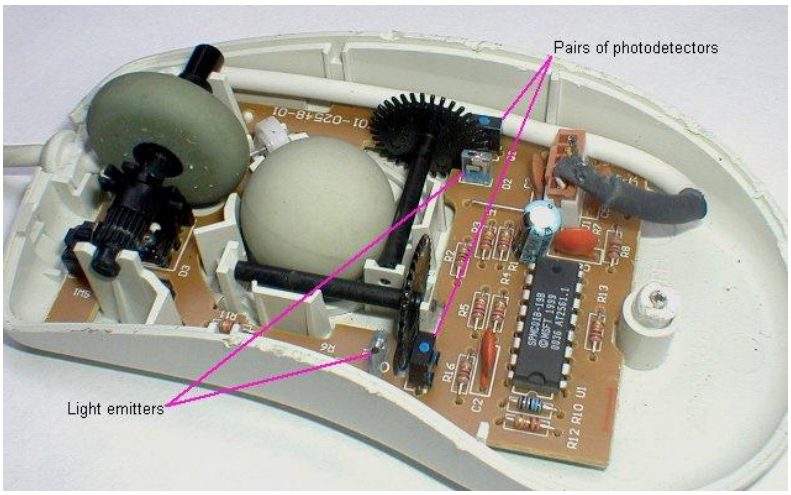
Bàn phím sử dụng một ma trận hình thành bởi các dòng và cột dây dẫn, như minh hoạ trên hình Hình 87 và Hình 88. Mỗi phím hoạt động như một công tắc điện. Khi phím được ấn, dây dẫn cột được nối với dây dẫn dòng tạo thành một mạch kín. Bộ điều khiển bàn phím liên tục quét ma trận phím để phát hiện mạch kín và ghi nhận phím được ấn. Quá trình xử lý phím ấn và tạo tín hiệu gửi CPU xử lý trong bàn phím có thể được tóm tắt như sau:

- Khi một phím được ấn, bộ điều khiển bàn phím phát hiện và sinh ra một mã quét tương ứng (scan code);
- Một ngắt (interrupt) bàn phím được gửi đến máy tính;

- Khi nhận được tín hiệu ngắt bàn phím:
  - Máy tính thực hiện chương trình điều khiển ngắt bàn phím:
    - Đọc mã quét phím
    - Chuyển mã quét phím thành mã ký tự tương ứng (thông thường là mã ASCII).
  - Một ký tự có thể được hiển thị theo nhiều hình thức khác nhau theo các bộ font.

6.4.2 Chuột

Chuột (mouse) là một trong các thiết bị vào của máy tính được sử dụng rộng rãi nhất. Chức năng chính của chuột là điều khiển. Thông qua các phần mềm, hình thức hiển thị của chuột được thể hiện rất đa dạng, từ hình mũi tên đơn giản, đến bàn tay, đồng hồ cát, ... theo các trạng thái làm việc của chương trình. Hiện nay, có rất nhiều loại chuột đang được sử dụng. Ngoài chuột bi (còn gọi là chuột cơ khí), còn có chuột quang, chuột laser, chuột cảm ứng và chuột không dây. Các phím bấm chuột cũng rất đa dạng: thông thường là loại 3 phím (trái, phải và cuộn); một số chuột có thể có thêm cả phím tiến (forward) và phím lùi (backward).

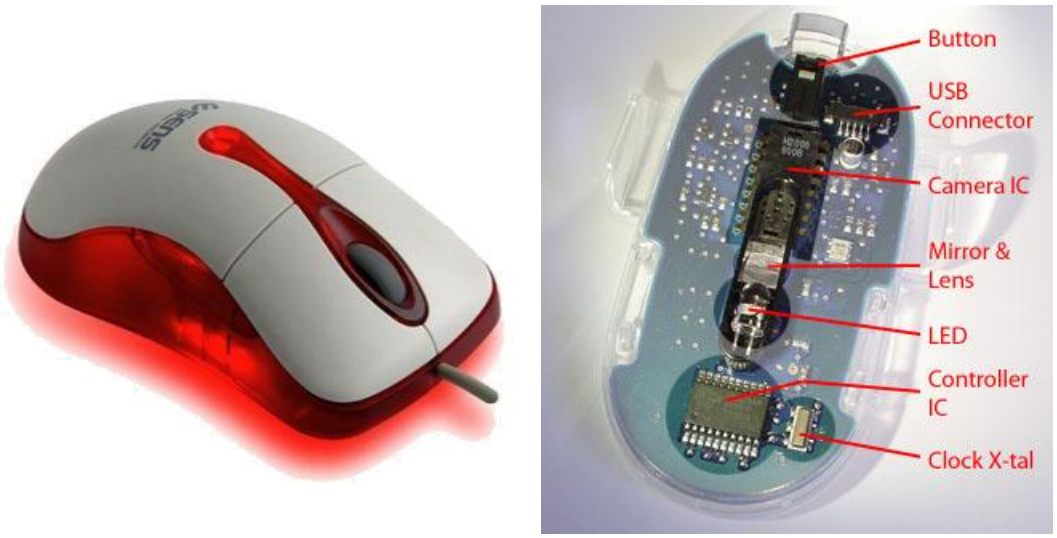


Hình 89 Chuột bi hay chuột cơ khí

Chuột bi hay chuột cơ khí là loại chuột có cấu tạo đơn giản và được sử dụng sớm nhất. Hình 89 cho thấy các thành phần bên trong của chuột bi. Chuột bi hoạt động theo nguyên tắc cơ khí – quang – điện: biến chuyển động của viên bi khi rê chuột thành các tín hiệu điện biểu diễn các chuyển động theo phương ngang và phương đứng của chuột. Cụ thể, nguyên tắc hoạt động của chuột bi có thể tóm tắt như sau:

- Khi chuột di chuyển, viên bi chuột quay;
- Khi bi quay nó kéo theo 2 trục áp vào quay theo. Hai trục được gắn bánh xe răng cưa ở 1 đầu:
  - Một trục dùng để phát hiện chuyển động theo phương đứng
  - Một trục dùng để phát hiện chuyển động theo phương ngang
- Hai đi-ốt sinh tia hồng ngoại chiếu qua phần bánh răng cưa gắn trên các trục kể trên:
  - Khi bánh răng cưa quay, ánh sáng hồng ngoại chiếu qua sẽ bị ngắt quãng;

- Ở phía đối diện có 2 bộ cảm biến chuyển ánh sáng hồng ngoại sau bánh răng cưa thành tín hiệu điện;
- Tín hiệu điện thu được phản ánh chuyển động của chuột được chuyển cho máy tính xử lý.



Hình 90 Chuột quang và cấu tạo

Khác với chuột bi, chuột quang (optical mouse) không có bi nên thường nhẹ và đạt độ chính xác cao hơn. Hiện nay, chuột quang đã thay thế hầu hết các chuột bi. Hình 90 minh hoạ chuột quang và cấu tạo của nó. Chuột quang sử dụng nguyên tắc liên tục chụp và phân tích ảnh bề mặt chuột di chuyển để phát hiện chuyển động của chuột. Cụ thể, nguyên tắc hoạt động của chuột quang có thể tóm tắt như sau:

- Một đi-ốt phát ánh sáng đỏ qua ống kính chiếu xuống mặt phẳng di chuột; ánh sáng phản xạ từ mặt phẳng di chuột quay ngược trở lại phía dưới chuột;
- Một camera đặt phía dưới chuột liên tục chụp ảnh của bề mặt di chuột nhờ ánh sáng phản xạ. Tốc độ chụp là khoảng 1500 ảnh/giây;
- IC điều khiển chuột sẽ phân tích và so sánh các ảnh kề nhau và qua đó phát hiện ra chuyển động chuột;
- Tín hiệu biểu diễn chuyển động chuột do IC điều khiển chuột sinh ra được chuyển cho máy tính xử lý.

Tương tự như chuột quang, chuột laser cũng sử dụng phương pháp chụp và phân tích ảnh bề mặt kề nhau để phát hiện chuyển động. Tuy nhiên, chuột laser sử dụng ánh sáng laser với tốc độ chụp ảnh lên đến 6000 ảnh/giây. Nhờ vậy, chuột laser thường có độ chính xác và độ nhạy cao hơn sơ với chuột quang.

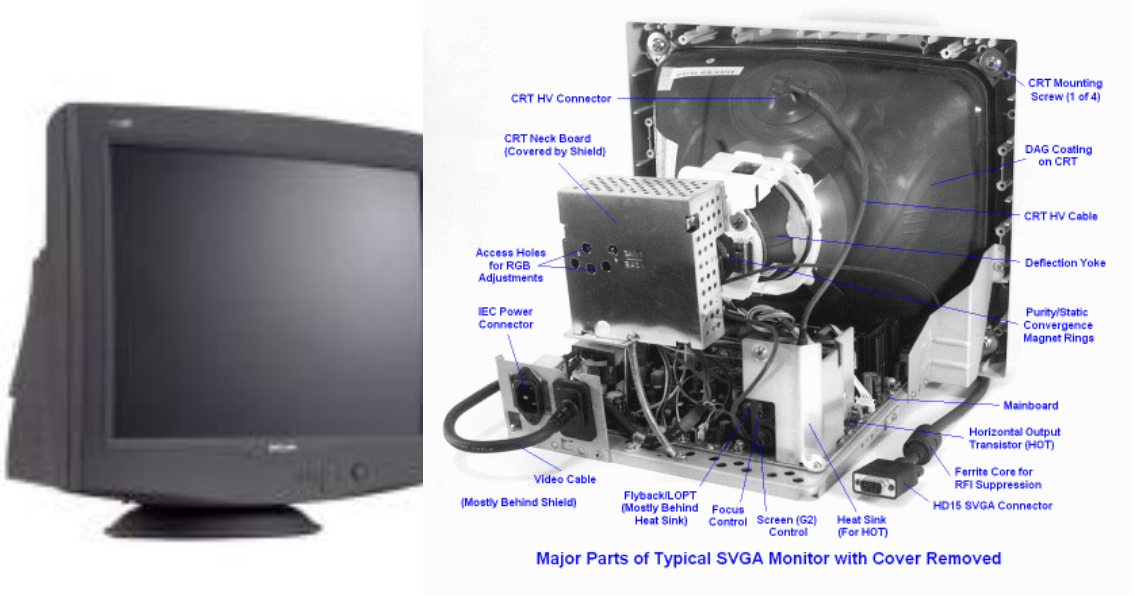
6.4.3 Màn hình

Màn hình (monitor / screen) là thiết bị ra chuẩn có thể hiện thị thông tin dưới dạng văn bản hoặc hình ảnh. Cùng với bàn phím và chuột, màn hình là thiết bị không thể thiếu đối với máy tính. Có ba dạng màn hình được sử dụng thông dụng: màn hình ống điện tử CRT, màn hình

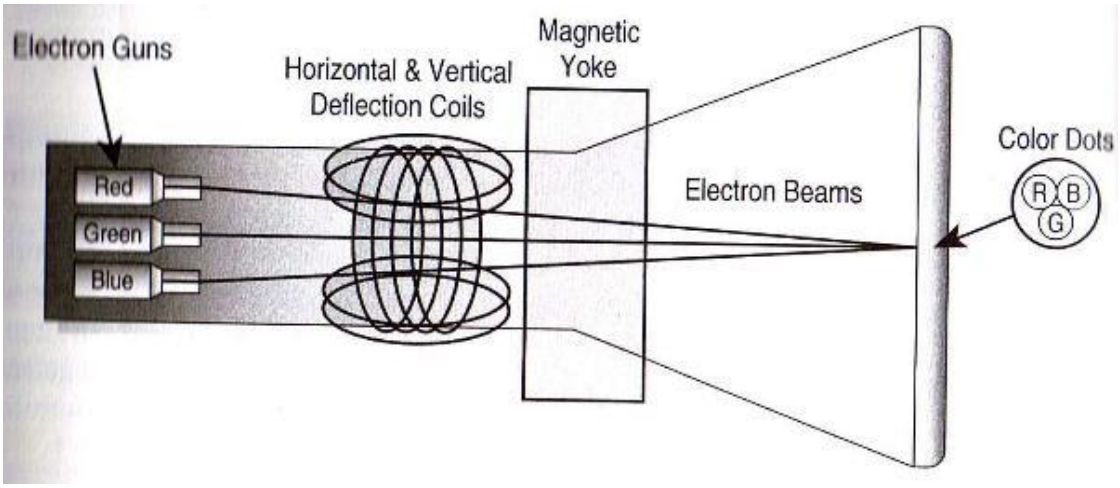
tinh thể lỏng LCD và màn hình plasma. Tài liệu này chỉ đề cập đến hai loại màn hình được sử dụng phổ biến cho máy tính là màn hình CRT và LCD.

6.4.3.1 Màn hình CRT

Màn hình CRT (Cathode Ray Tube) sử dụng tia điện tử phát ra từ cực Cathode bắn lên mặt huỳnh quang phốt pho để tạo ảnh. Tia điện tử được điều khiển bởi 2 cuộn lái tia (dòng và mành) để quét hết cả màn hình, đảm bảo tốc độ quét tối thiểu là 24 màn hình/giây. Tín hiệu hình ảnh (video) được sử dụng để điều khiển mật độ tia điện tử bắn lên màn huỳnh quang tạo các mức sáng/tối khác nhau. Màn hình đen trắng sử dụng 1 súng điện tử, còn màn hình mầu sử dụng 3 súng điện tử ứng với 3 mầu cơ bản Đỏ (Red), Xanh lá cây (Green) và Xanh da trời (Blue). Ba mầu này được trộn với nhau theo tỷ lệ khác nhau tạo thành tất cả các mầu có trong tự nhiên cho điểm ảnh. Hình 91 minh hoạ bên ngoài và các bộ phận bên trong của màn hình CRT, còn Hình 92 minh hoạ nguyên lý tạo điểm ảnh của màn hình CRT mầu.



Hình 91 Bên ngoài và bên trong màn hình CRT

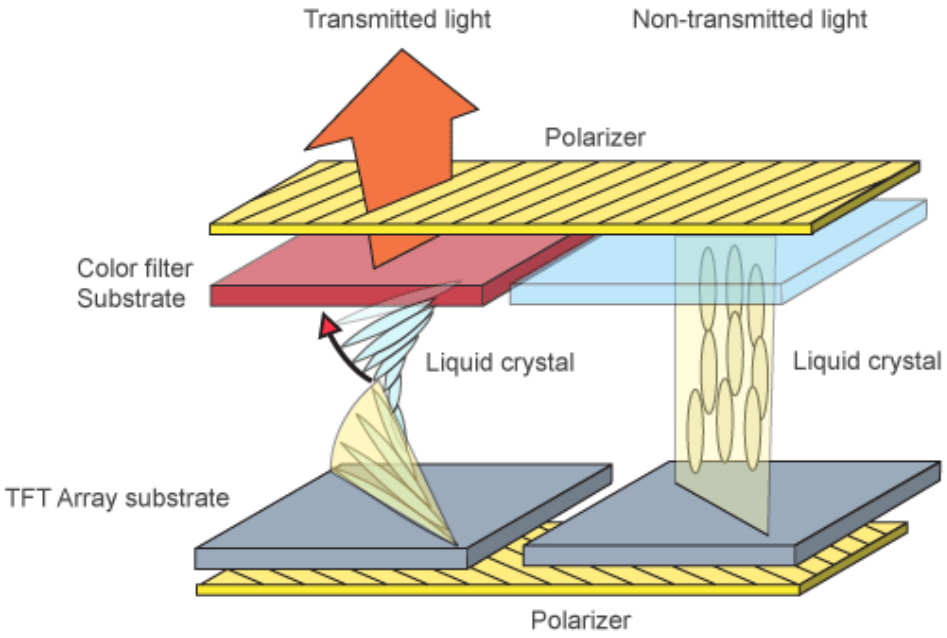


Hình 92 Nguyên lý điểm ảnh của màn hình CRT mầu

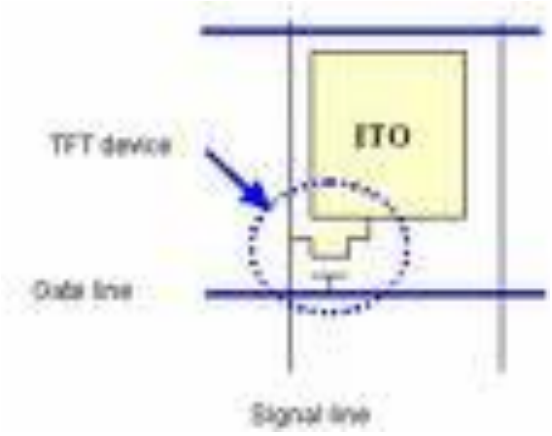
6.4.3.2 Màn hình LCD

Màn hình LCD (Liquid Crystal Display) là màn hình tạo ảnh dựa trên sự linh động của các "*tinh thể lỏng*" (Liquid Crystals). Tinh thể lỏng là các chất bán rắn lỏng rất nhạy cảm với nhiệt độ và dòng điện. So với màn hình CRT, màn hình LCD mỏng hơn, nhẹ hơn và tiêu thụ ít điện năng hơn. Ngoài ra, phần diện tích màn hình thực để hiển thị ảnh (viewable size) của LCD cũng lớn hơn. Chẳng hạn màn hình LCD 15" có phần màn hình thực tương đương màn hình CRT 17". Nhược điểm của LCD so với CRT là không hỗ trợ nhiều độ phân giải, chất lượng ảnh không cao, thời gian đáp ứng (response time) lớn và góc nhìn (view angle) nhỏ.

Có thể phân loại màn hình LCD thành 2 loại theo nguồn pháp sáng: LCD chiếu sau (backlit) và LCD phản xạ (reflective). LCD chiếu sau sử dụng nguồn sáng riêng đặt ở phía sau, thường dùng trong các LCD có công suất lớn, như màn hình máy tính và màn hình tivi. LCD phản xạ sử dụng ánh sáng phản xạ của nguồn sáng từ bên ngoài. LCD phản xạ có thiết kế đơn giản, rẻ tiền, thường thích hợp với các màn hình có công suất nhỏ, như màn hình đồng hồ, màn hình máy tính tay.

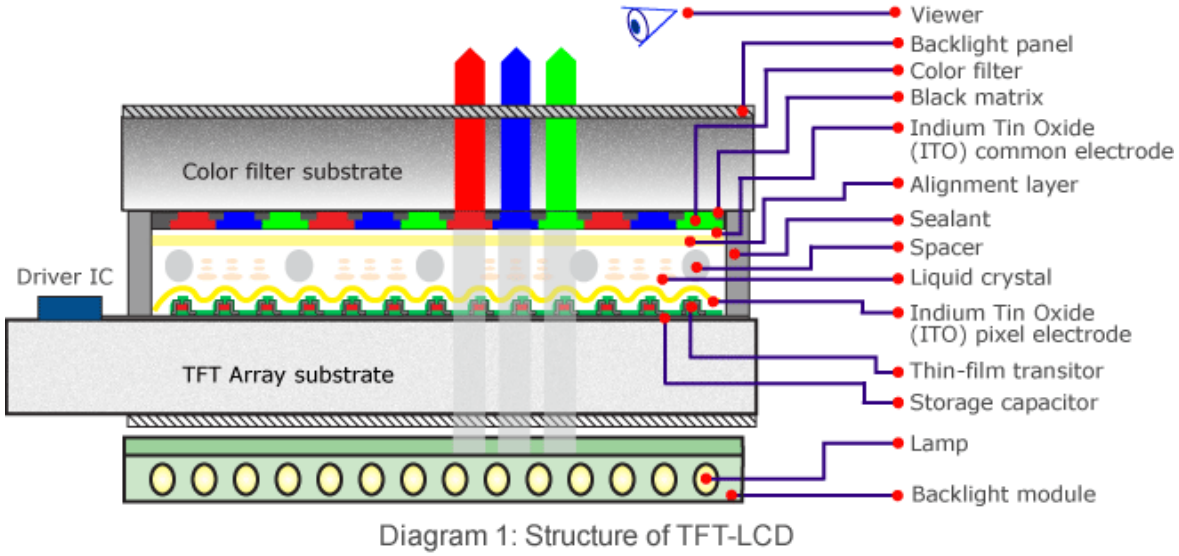


Hình 93 Mô hình lọc ánh sáng của tinh thể lỏng điều khiển bằng điện



Hình 94 Một TFT - Thin Film Transitor

Bản thân tinh thể lỏng không có khả năng phát sáng, nhưng chúng có khả năng "lọc" hay thay đổi cường độ ánh sáng đi qua theo điện áp dòng điện đặt vào. Hình 93 minh hoạ mô hình lọc ánh sáng của tinh thể lỏng được điều khiển bằng điện. Dựa trên phương pháp điều khiển các tinh thể lỏng, ta có 2 loại LCD: LCD ma trận thụ động (Passive matrix) và LCD ma trận chủ động (Active matrix). LCD ma trận thụ động sử dụng lưới hoặc ma trận để định nghĩa từng điểm ảnh (pixel) bởi hàng và cột của nó. Một điểm ảnh (giao giữa 1 hàng và 1 cột) được kích hoạt khi điện áp được đặt vào cột và dòng tương ứng được nối đất. Ngược lại, LCD ma trận chủ động sử dụng một TFT (Thin Film Transistor) để điều khiển một phần tử tinh thể lỏng. Các TFT hoạt động tương tự như các bộ chuyển mạch, như minh hoạ trên Hình 94.



Hình 95 Cấu trúc của màn hình TFT-LCD

Hình 95 minh hoạ cấu trúc của màn hình TFT-LCD. TFT-LCD hoạt động theo nguyên lý tóm tắt như sau:

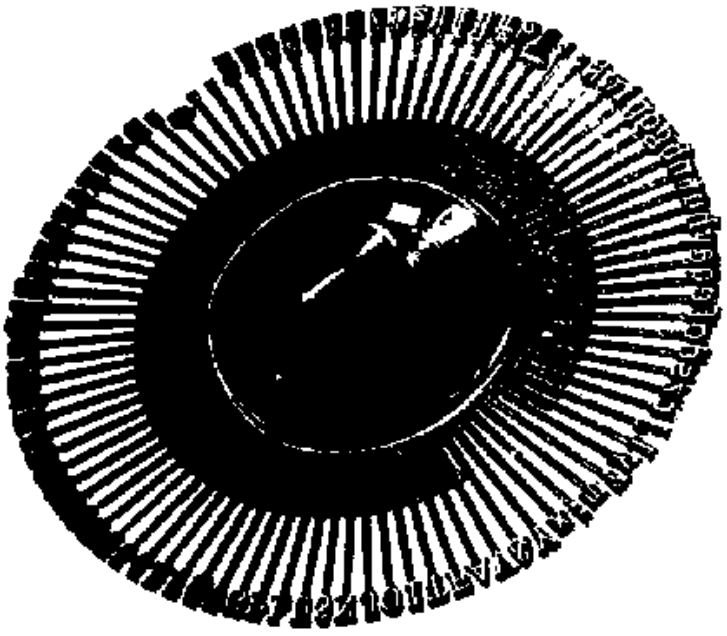
- TFT LCD là thiết bị được điều khiển bằng các tín hiệu điện;
- Lớp tinh thể lỏng năm giữa 2 lớp trong suốt chứa các điện cực ITO (Indium Tin Oxide);
- Các phần tử tinh thể lỏng được sắp đặt theo các hướng khác nhau theo sự thay đổi điện áp đặt vào các điện cực ITO;
- Hướng của các phần tử tinh thể lỏng trực tiếp ảnh hưởng đến cường độ ánh sáng đi qua và nó gián tiếp điều khiển mức sáng / tối (còn gọi là mức xám) của ảnh hiện thị;
- Mầu của hình ảnh được tạo bởi một lớp lọc mầu;
- Mức xám của các điểm ảnh được thiết lập theo mức điện áp của tín hiệu video đưa vào điện cực điều khiển.

6.4.4 Máy in

Máy in (printer) là thiết bị ra phổ biến dùng để kết xuất thông tin ra giấy. Qua quá trình phát triển, có nhiều loại máy in được sử dụng như máy in búa (Typewriter-derived printers), máy in kim (Dot-matrix printers), máy in laser (Laser printers), máy in phun mực (Inkjet printers), máy in mầu (Colour printers) và các máy in đa chức năng (Multi-function printers).

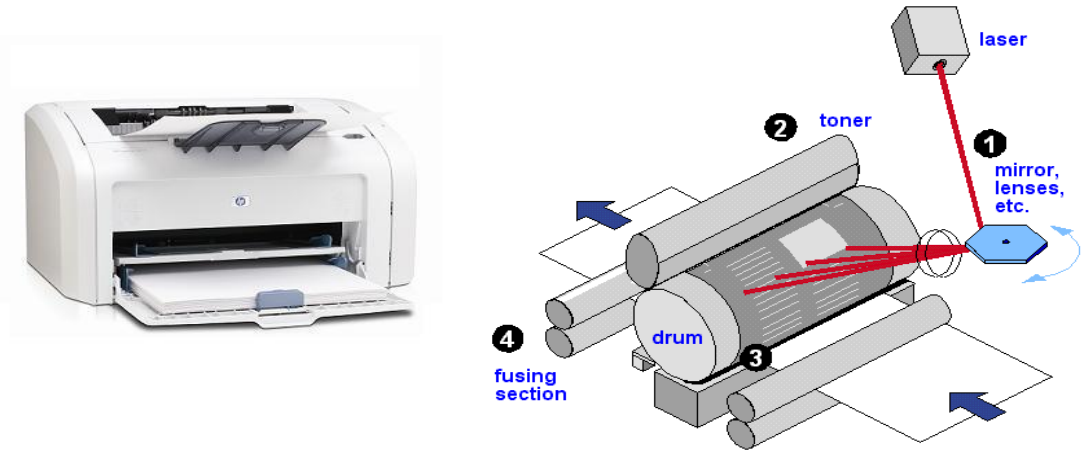
Hình 96 minh hoạ máy in búa. Máy in búa sử dụng các con chữ có kích thước cố định như máy đánh chữ. Ngược lại, máy in kim sử dụng bộ kim để tạo ma trận các chấm để tạo khuôn chữ như minh hoạ trên Hình 97. Hình 98 và Hình 99 minh hoạ nguyên lý hoạt động của máy in laser. Khác với các dòng máy in đi trước, máy in laser sử dụng phương pháp chụp ảnh điện tích bằng tia laser để tạo chữ. Cụ thể như sau:

- Trống cảm quang được nạp một lớp điện tích nhờ 1 điện cực;
- Tia laser từ nguồn sáng laser đi qua một gương quay và bộ điều chế tia được điều khiển bởi tín hiệu cần in đến mặt trống;
- Ánh sáng laser làm thay đổi mật độ điện tích trên mặt trống; Như vậy, mật độ điện tích trên mặt trống thay đổi theo tín hiệu cần in;
- Khi trống cảm quang quay đến hộp mực thì điện tích trên trống hút các hạt mực được tích điện trái dấu. Các hạt mực dính trên trống biểu diễn âm bản của văn bản/thông tin cần in;
- Giấy từ khay được kéo lên cũng được điện cực nạp điện tích trái dấu với điện tích của mực nên hút các hạt mực khỏi trống cảm quang.
- Giấy tiếp tục đi qua trống sấy nóng làm các hạt mực chảy ra và bị ép chặt vào giấy.

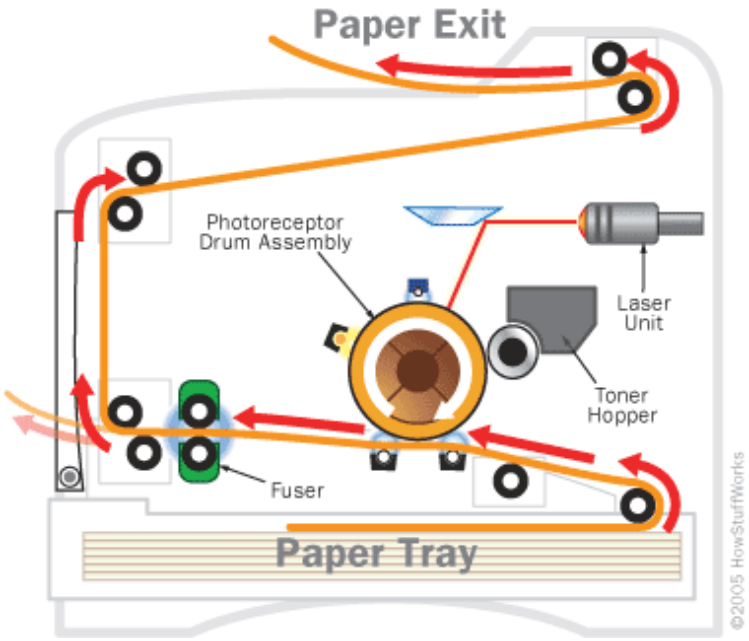


Hình 96 Máy in búa

Hình 97 Máy in kim



Hình 98 Máy in Laser



Hình 99 Nguyên lý in Laser

6.5 CÂU HỎI ÔN TẬP

1. Các thành phần của hệ thống bus và các loại bus.
2. Nguyên lý làm việc của bus PCI.
3. Nguyên lý làm việc của bus PCI Express.
4. Giới thiệu các thiết bị vào ra và các cổng vào ra.
5. Nguyên lý hoạt động của bàn phím.
6. Nguyên lý hoạt động của chuột quang.
7. Nguyên lý hoạt động của màn hình CRT.
8. Nguyên lý hoạt động của màn hình TFT LCD.
9. Nguyên lý hoạt động của máy in laser.

TÀI LIỆU THAM KHẢO

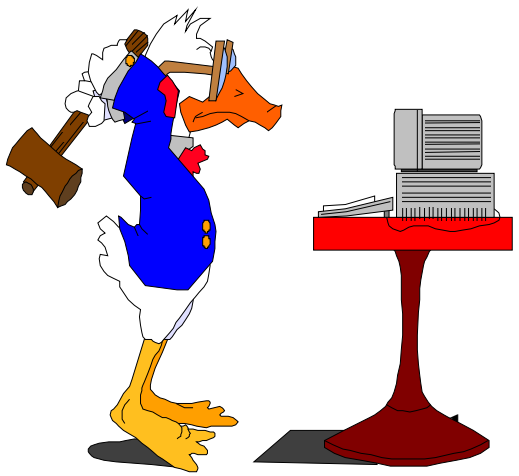
1. Stallings W., *Computer Organization and Architecture: Designing for Performance*, 8th Edition, Prentice – Hall 2009.
2. Mostafa Abd-El-Barr and Hesham El-Rewini, *Fundamentals of Computer Organization and Architecture*, John Wiley & Sons, Inc, 2005.
3. Hennesy J.L. and Patterson D.A., *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann, 4th Edition, 2006.
4. Hồ Khánh Lâm, *Kỹ thuật vi xử lý*, Nhà xuất bản Bưu điện, 2005
5. Trần Quang Vinh, *Cấu trúc máy vi tính*, Nhà xuất bản Giáo dục, 1999.
6. Trang Wikipedia.org, tham khảo năm 2009 và 2010.
7. Trang Howstuffworks.com, tham khảo năm 2009 và 2010.
8. Trang PCGuide.com, tham khảo năm 2009 và 2010.

TRƯỜNG ĐẠI HỌC THỦY SẢN
KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

CẤU TRÚC MÁY TÍNH

(Computer Structure)



Giới thiệu

Cấu trúc Máy tính

(Computer Structure)

Trình bầy: Đinh Đồng Lưỡng.

ĐT: 058.832078

Mobile: 0914147520

Email: luongdd10@yahoo.com

Mục đích và yêu cầu

***Mục đích*:**

- Tìm hiểu *cấu trúc và tổ chức* các máy tính.
- Tìm hiểu nguyên lý hoạt động cơ bản Máy tính.
- Giới thiệu cấu trúc máy tính tiên tiến của Intel.

***Yêu cầu*:**

- Có kiến thức lập trình cơ bản.
- Sinh viên đọc tài liệu và làm việc theo nhóm để thực hiện báo cáo trên lớp.

Tài liệu tham khảo (sách)

1.Willian Stallings - Computer Organization and Architecture.

2.Andrew Stamenbaum – Structure Computer Organization.

3.Cẩm nang sữa chữa và nâng cấp máy tính cá nhân

Nguyễn Đăng Khoa

4.Giáo trình bảo trì và nâng cấp máy tính

(Trường KHTN - TPHCM)

Lê Công Bảo

5. Cấu tạo và nguyên lý hoạt động của hệ thông máy

RON WHITE - Nguyễn Trọng Tuấn (Dịch)

Tài liệu tham khảo (trang web)

@www.williamstallings.com

@ocw.mit.edu

@www.intel.com

@www.asus.com

@www.gigabyte.com

@www1.guidePC.com

Nội dung môn học

1. Giới thiệu chung.
2. Hệ thống máy tính.
3. Biểu diễn dữ liệu và số học máy tính.
4. Bộ xử lý trung tâm.
5. Bộ nhớ Máy tính.
6. Hệ thống vào ra.

Chương 1

Giới thiệu chung

1.1 Khái niệm chung máy tính

1.2 Phân loại máy tính

1.3 Sự tiến hóa của máy tính

1.1 Khái niệm chung

Máy tính(computer) là thiết bị điện tử thực hiện công việc sau:

- ✓ nhận thông tin vào.
- ✓ xử lý thông tin theo chương trình nhớ sẵn bên trong bộ nhớ máy tính.
- ✓ đưa thông tin ra.

Chương trình (Program): chương trình là dãy các câu lệnh nằm trong bộ nhớ, nhằm mục đích hướng dẫn máy tính thực hiện một công việc cụ thể nào đấy. Máy tính thực hiện theo chương trình.

1.1 Khái niệm chung

Phần mềm (Software): Bao gồm chương trình và dữ liệu.

Phần cứng (Hardware): Bao gồm tất cả các thành phần vật lý cấu thành lên hệ thống Máy tính.

Phần dẻo (Firmware): Là thành phần chứa cả hai thành phần trên.

Kiến trúc máy tính(Computer Architecture) đề cập đến các thuộc tính của hệ thống máy tính dưới cái nhìn của người lập trình. Hay nói cách khác, là những thuộc tính ảnh hưởng trực tiếp đến quá trình thực hiện logic của chương trình. Bao gồm: tập lệnh, biểu diễn dữ liệu, các cơ chế vào ra, kỹ thuật đánh địa chỉ,…

1.1 Khái niệm chung

Tổ chức máy tính(Computer Organization): đề cập đến các khối chức năng và liên hệ giữa chúng để thực hiện những đặc trưng của kiến trúc.

Ví dụ: trong kiến trúc bộ nhân: đây là thuộc tính của hệ thống xử lý. Bộ nhân này sẽ được tổ chức riêng bên trong máy tính hoặc nó được tính toán nhiều lần trên bộ cộng để cũng được một kết qủa nhân tương ứng.

Cấu trúc máy tính(Computer Structure): là những thành phần của máy tính và những liên kết giữa các thành phần.

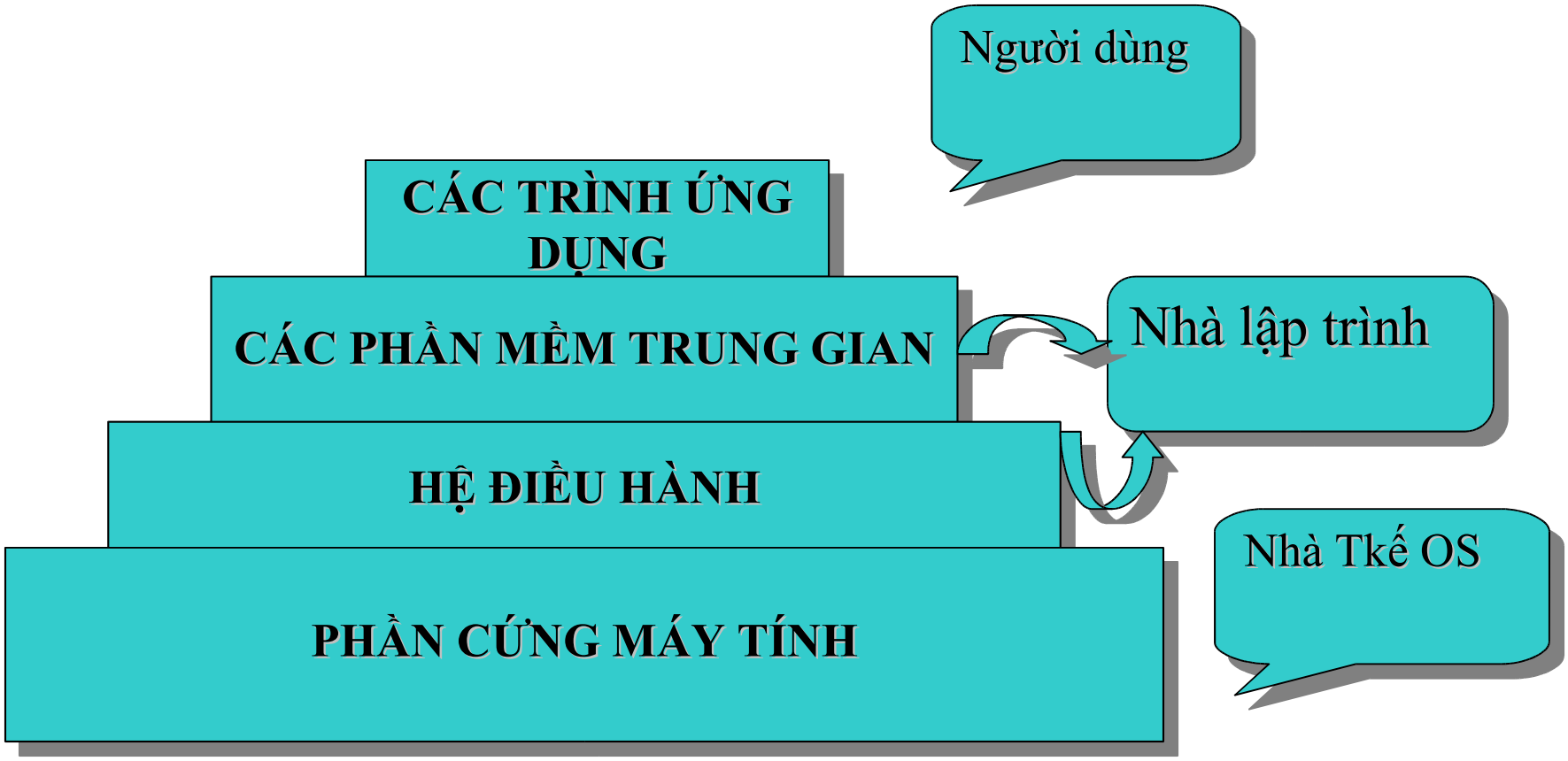
Ở mức cao nhất máy tính bao gồm 4 thành phần:

1.1 Khái niệm chung

- ✓Bộ xử lý : điều khiển và xử lý số liệu.
- ✓Bộ nhớ : chứa chương trình và dữ liệu.
- ✓Hệ thống vào ra : trao đổi thông tin giữa máy tính với bên ngoài.
- ✓Liên kết giữa các hệ thống : liên kết các thành phần của máy tính lại với nhau.

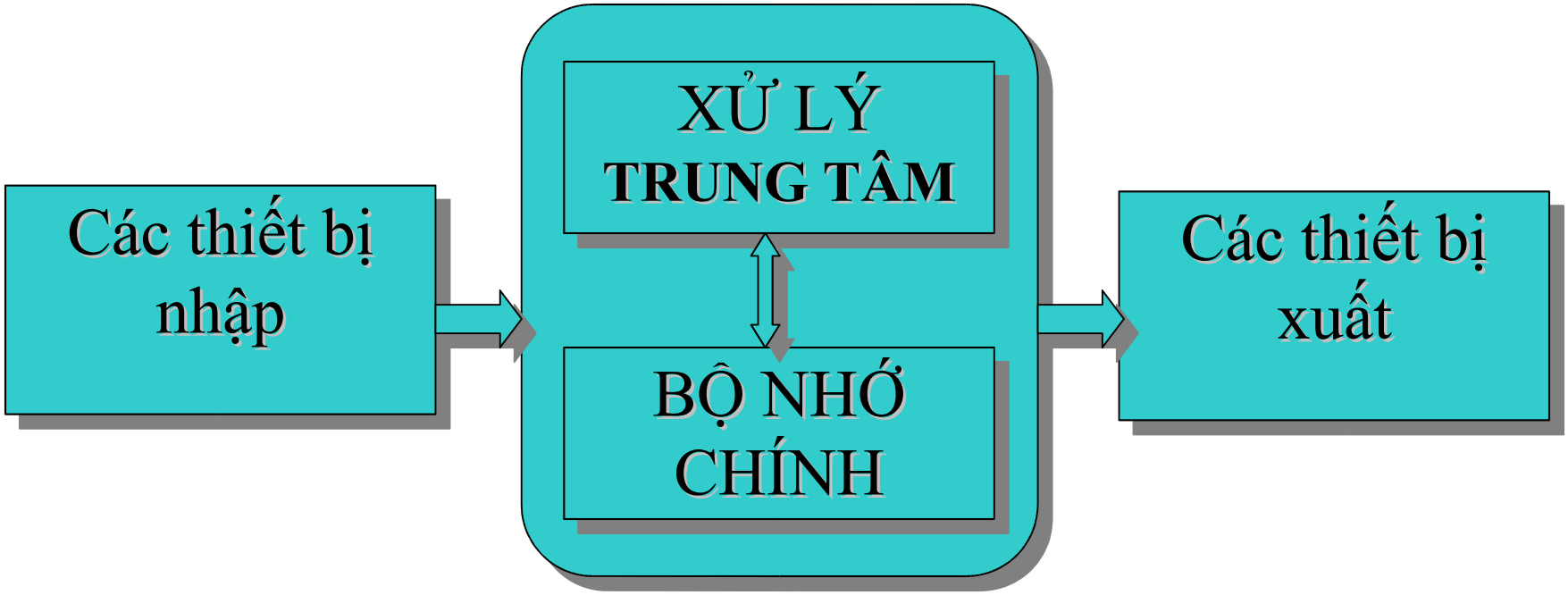
1.1 Khái niệm chung

- Mô hình phân lớp của hệ thống



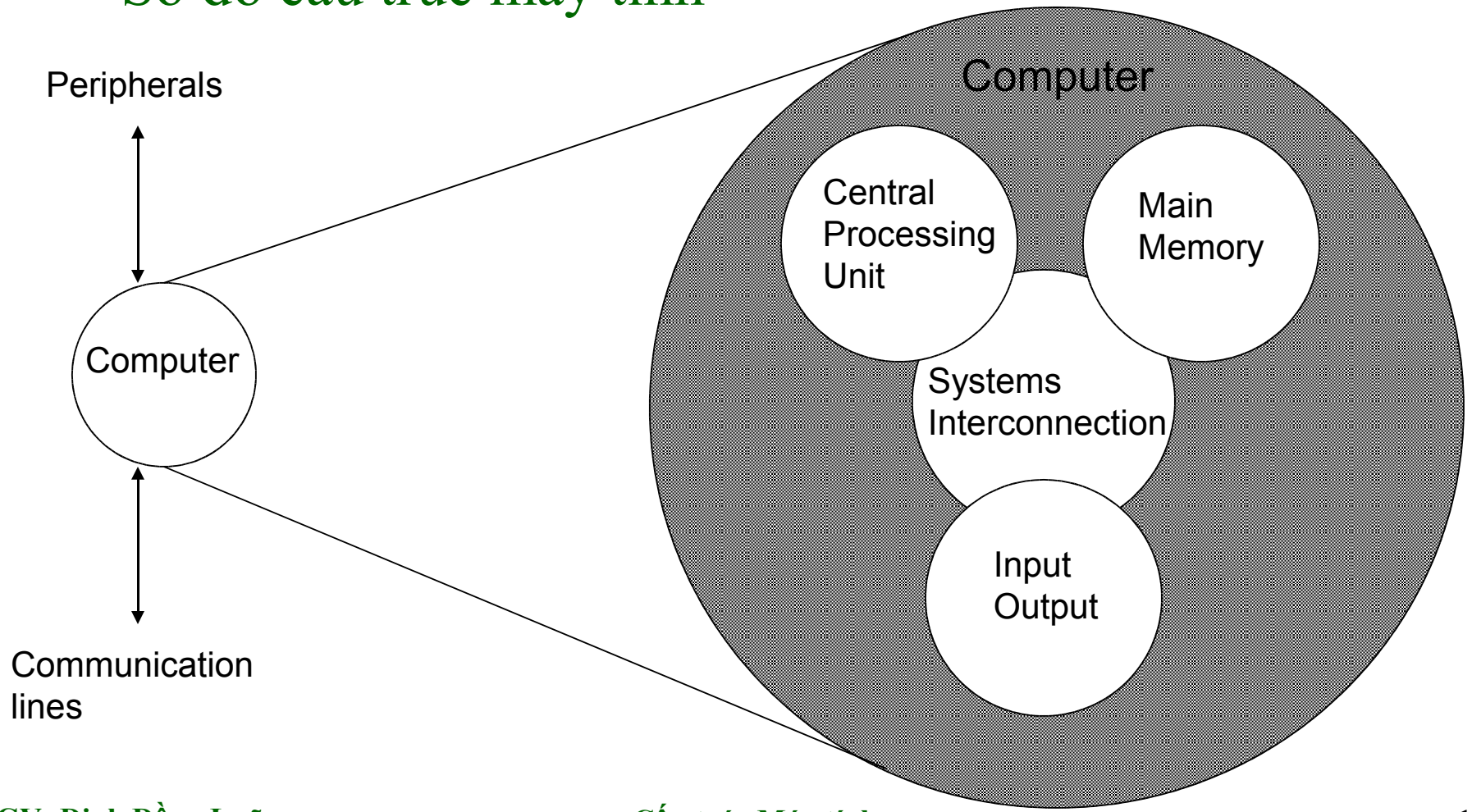
1.1 Khái niệm chung

- Mô hình cơ bản



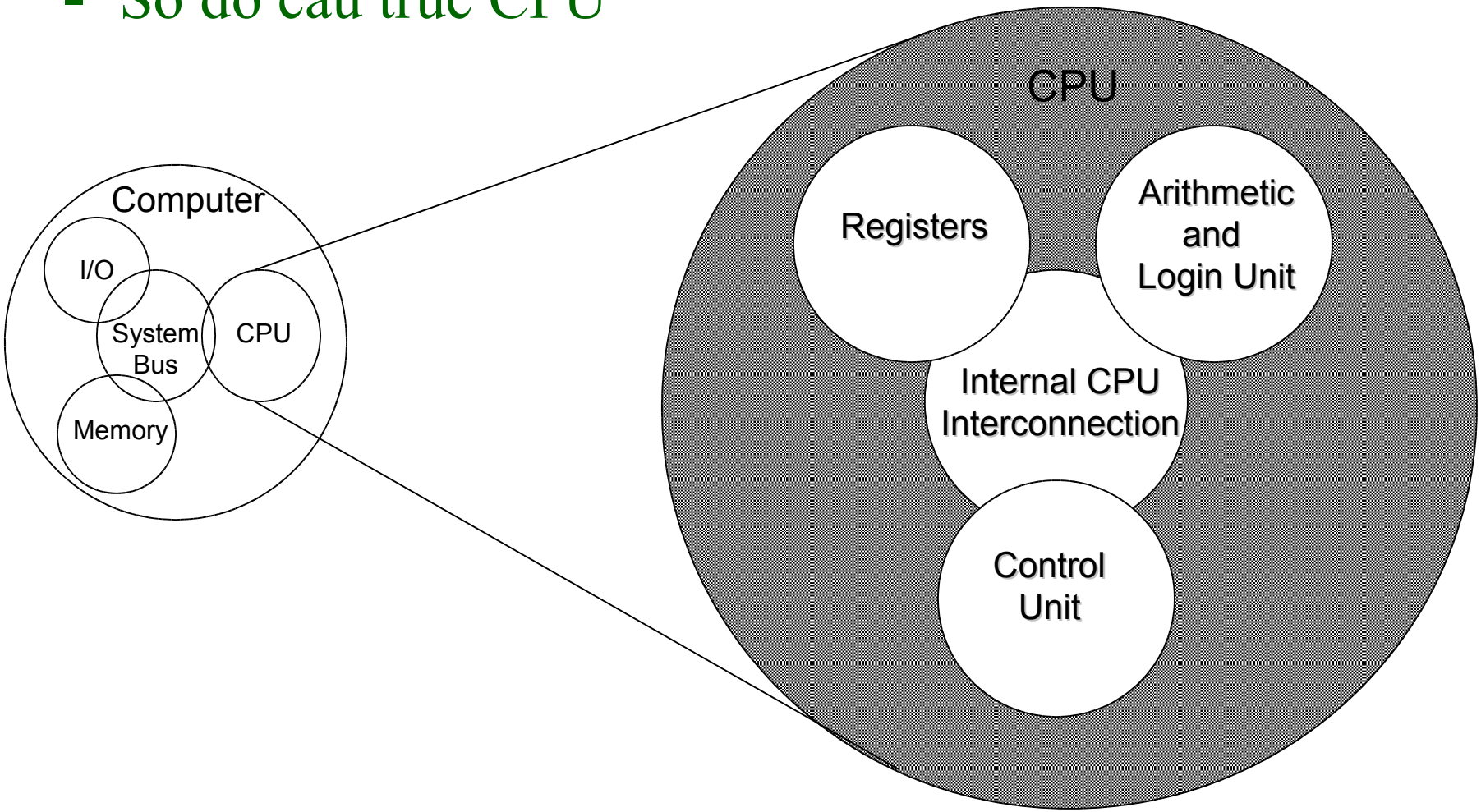
1.1 Khái niệm chung

- Sơ đồ cấu trúc máy tính



1.1 Khái niệm chung

- Sơ đồ cấu trúc CPU



1.1 Khái niệm chung

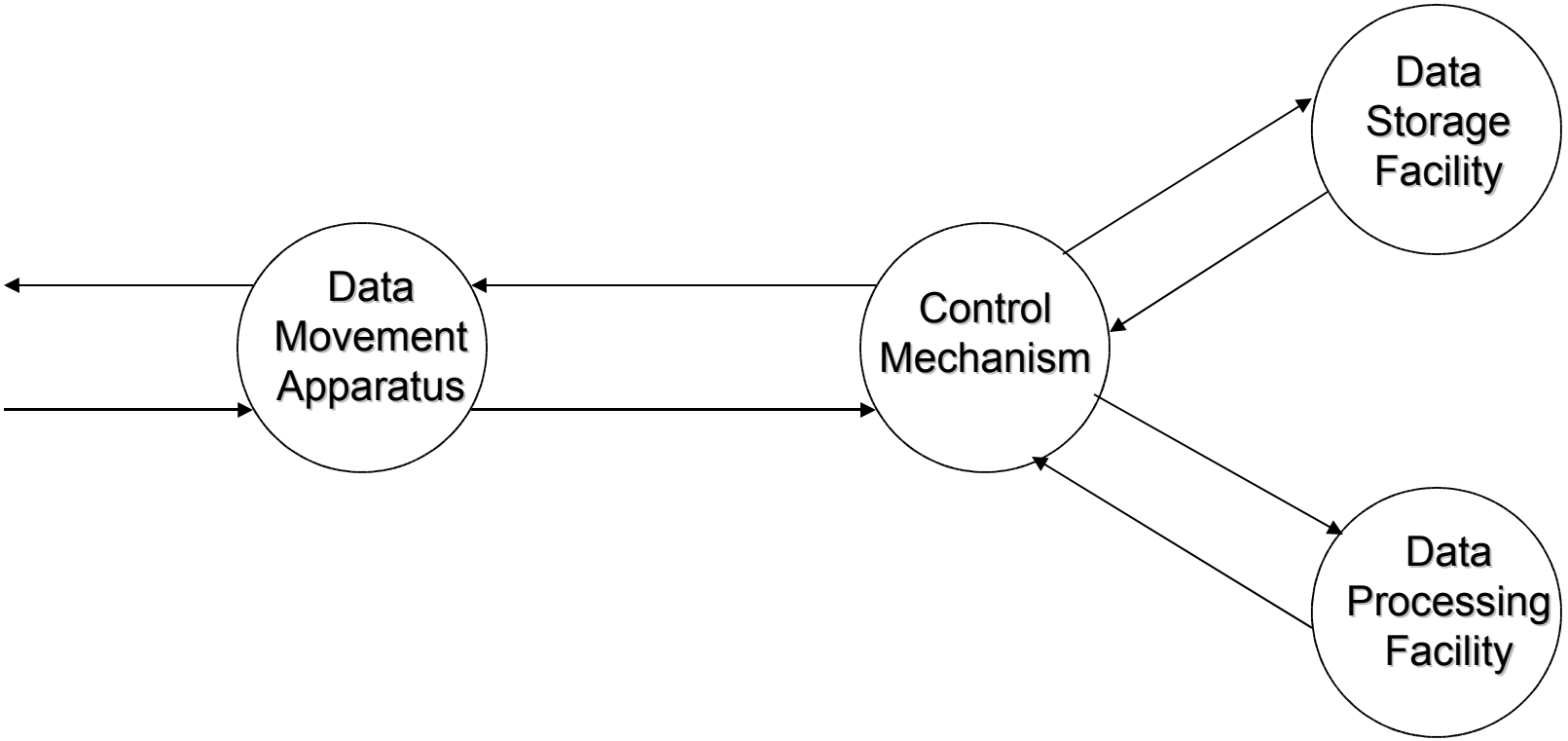
Chức năng(Computer Function): là mô tả hoạt động của hệ thống hay từng thành phần của hệ thống.

Chức năng chung của một hệ thống bao gồm:

- ✓Xử lý dữ liệu.
- ✓Lưu trữ dữ liệu.
- ✓Vận chuyển dữ liệu.
- ✓Điều khiển

1.1 Khái niệm chung

- Sơ đồ chức năng máy tính



1.2 Phân loại máy tính

Phân loại theo phương pháp truyền thống

- ✓Máy vi tính (*Microcomputer*)
- ✓Máy tính nhỏ (*Minicomputer*)
- ✓Máy tính lớn (*Mainframe Computer*)
- ✓Siêu máy tính (*Super Computer*)

Phân loại theo phương pháp hiện đại

- ✓Máy tính để bàn (*Desktop Computer*)
- ✓Máy chủ (*Servers*)
- ✓Máy tính nhúng (*Embedded Computer*)

1.2 Phân loại máy tính

Máy để bàn:

- là loại máy thông dụng nhất hiện nay.
- bao gồm máy tính cá nhân (PC: Persional Computer) và trạm (Workstation Computer).
- giá mua 100$ đến 10.000$

Máy chủ

- là máy phục vụ(*server*)
- dùng trong mạng theo mô hình Clent/Server
- có tốc độ, hiệu năng, bộ nhớ và độ tin cậy cao
- giá vài chục nghìn đến vài chục triệu đô

1.2 Phân loại máy tính

Máy tính nhúng

- được đặt trong nhiều thiết bị khác nhau để điều khiển thiết bị làm việc
- được thiết kế chuyên dụng
- ví dụ: điện thoại di động, bộ điều khiển các thiết gia đình, Router định tuyến,…

1.3 Sự tiến hóa của máy tính

Sự phát triển của máy tính chia ra 4 thế hệ:

- ✓ Thế hệ 1: Máy tính dùng đèn chân không (Vacumm Tube) 1946-1955
- ✓ Thế hệ 2: Máy tính dùng Transitor (1955-1965)
- ✓ Thế hệ 3: Máy tính dùng mạch tích hợp IC (Intergrated Circuit) 1966 – 1980
- ✓ Thế hệ 4: Máy tính dùng mạch tích hợp cực lớn VLSI (Very Large Scale Intergrated)1980 đến nay

Máy tính ENIAC

Electronic Numerical Integrator And Computer

1.3 Sự tiến hóa của máy tính

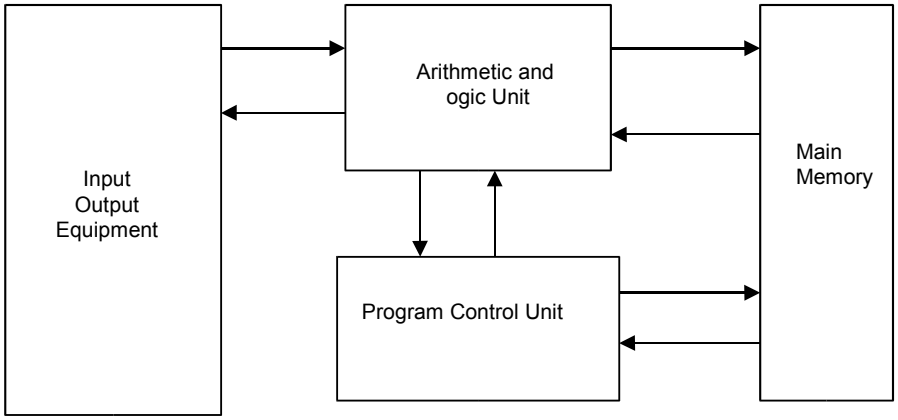
Đề xuất năm 1943 và hoàn thành 1946 được sử dụng đến1955 do thầy trò Eckert và Mauchly Trường đại học Pennsylvania của Mỹ

Đặc điểm chính:

- ✓Nặng 30 tấn, chiến diện tích 150m^2 và sử dụng 140KW.
- ✓5000 nghìn phép cộng trên giây.
- ✓Sử dụng hệ thập phân.
- ✓Lập trình bằng công tắc.
- ✓Sử dụng 18000 bóng đèn điện tử (vacuum tubes)

Máy tính Von Neumann

- Máy tính IAS(Institute for Advanced Studies)
- Máy có mô hình cơ bản là máy tính này nay
- Thế kế 1947 hoàn thành 1952
- Xây dựng dựa trên ý tưởng của Turring (Mỹ) và Von Neumann(Anh)



1.3 Sự tiến hóa của máy tính

- Các sản phẩm của công nghệ VLSI(Very Large Scale Integrated)
  - ✓ Bộ vi xử lý được chế tạo trên một con chip
  - ✓ Vi mạch điều khiển tổng hợp (Chipset)
  - ✓ Bộ nhớ bán dẫn độc lập(ROM, RAM) thiết kế thành Module
  - ✓ Các bộ vi điều khiển chuyên dụng.
- Bộ vi xử lý đầu tiên của Intel 4004 năm1971
- Bộ xử lý được coi hoàn thiện nhất là 8088/8086 năm 1978,1979 đây được coi là ngày sinh nhật của các máy tính sau này

Máy Micral, André Trương Trọng Thi sáng chế

- Micral Pháp, máy vi tính lắp ráp hoàn toàn đầu tiên

1.3 Sự tiến hóa của máy tính

Lịch sử phát triển máy tính thế hệ thứ 4		
1978	8086 (Intel)	
1979	8088 (Intel)	
1980	80286 (Intel)	
1993	Pentium (Intel)	
1997	Pentium II (Intel)	Celeron
1999	Pentium III (Intel)	Celeron
2003	Pentium 4 (Intel)	Celeron

Chương 2

Hệ thống máy tính

2.1 Các thành phần cơ bản của máy tính
2.2 Hoạt động cơ bản của máy tính
2.3 Liên kết hệ thống

2.1 Các thành phần cơ bản của máy tính

- **Mô hình cơ bản của máy tính.**

  Các mô hình máy tính hiện nay được thiết kế dựa trên kiến trúc Von Neumann.

- **Các đặc điểm kiến trúc của Von Neumann:**

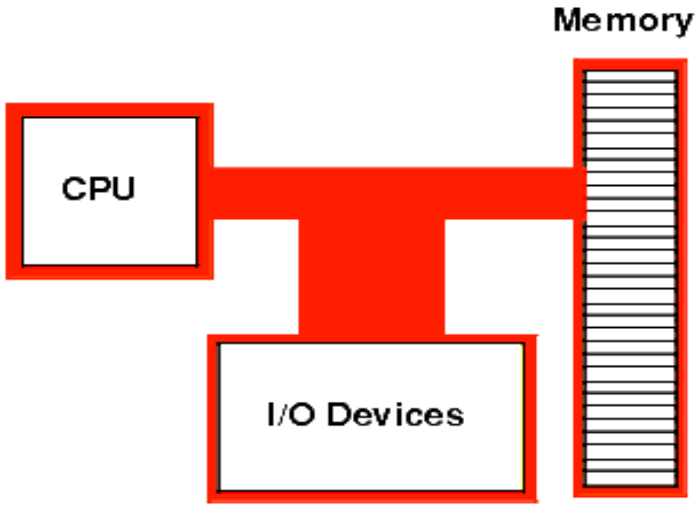
✓ Dữ liệu và chương trình chứa trong bộ nhớ đọc ghi.

✓ Bộ nhớ được đánh địa chỉ cho các ngăn nhớ không phụ thuộc vào nội dung của chúng.

✓ Máy tính thực hiện lệnh một cách tuần tự.

2.1 Các thành phần cơ bản của máy tính

- **Sơ đồ cấu trúc cơ bản c**



- ✓ Bộ xử lý trung tâm (CPU): Điều khiển hoạt động của máy tính và xử lý số liệu
- ✓ Hệ thống nhớ: chứa chương trình và dữ liệu đang được xử lý.
- ✓ Hệ thống vào/ra (I/O: Input/Output) : trao đổi thông tin giữa bên ngoài và bên trong máy tính
- ✓ Liên kết hệ thống (Interconnection): kết nối và vận chuyển thông tin giữa các thành phần với nhau

1.Bộ xử lý trung tâm
(CPU: Central Processing Unit)

Chức năng: Điều khiển toàn bộ hoạt động của máy tính.
Xử lý dữ liệu (vd: các phép toán số học và logic)

Nguyên tắc hoạt động: CPU hoạt động theo chương trình nằm trong bộ nhớ chính.

Cấu trúc cơ bản CPU

✓Đơn vị điều khiển (CU:Control Unit): Điều khiển hoạt động của máy tính theo chương trình đã định sẵn.

✓Đơn vị số học và logic (ALU: Arithmetic And Logic Unit): thực hiện các phép toán số học và logic trên các dữ liệu cụ thể.

✓Tập thanh ghi (RF: Register File): Lưu trữ các thông tin tạm thời phục vụ cho hoạt động của CPU.

✓Đơn vị nối ghép BUS(BIU: Bus Interface Unit): kết nối và trao đổi thông tin giữa Bus bên trong và Bus bên ngoài CPU.

1.Bộ xử lý trung tâm
(CPU: Central Processing Unit)

- ✓ Bộ vi xử lý hoạt động theo xung nhịp(clock) có tần số xác định.
- ✓ Tốc độ vi xử lý được đánh giá gián tiếp thông qua tần số xung nhịp.
- ✓ Gọi T_o : chu kỳ xung nhịp, $f_o = 1/T_o$ tần số xung nhịp.
- ✓ Mỗi thao tác của bộ xử lý cần kT_o. T_o càng nhỏ thì bộ xử lý chạy càng nhanh
- ✓ Ví dụ: Một máy tính Pentium 4 tốc độ 2GHz

  Ta có $f_o = 2GHz = 2.10^9 Hz$

  $T_o = 1/f_o = 1/2.10^9 = 0.5ns$

2. Bộ nhớ máy tính

- Chức năng: Lưu trữ chương trình và dữ liệu.

  Các thao tác cơ bản:

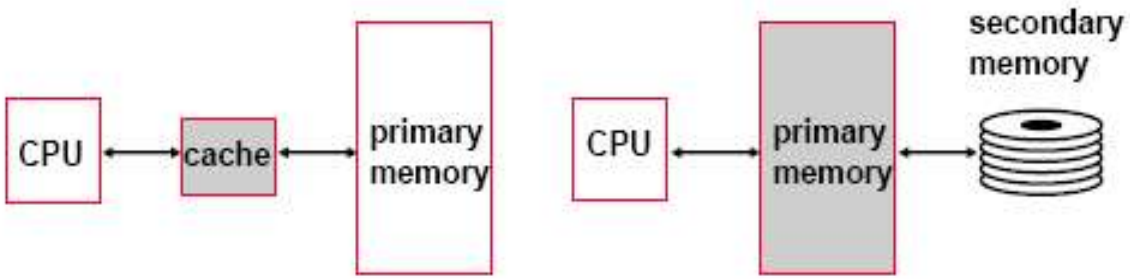
  ✓Thao tác đọc dữ liệu (Read)

  ✓Thao tác ghi dữ liệu (Write)

- Các thành phần chính

  ✓Bộ nhớ trong (Internal Memory)

  ✓Bộ nhớ ngoài (External Memory)



Bộ nhớ trong(Internal Memory)

- Chức năng và đặc điểm:
  - ✓Chứa thông tin mà CPU có thể trao đổi trực tiếp
  - ✓Tốc độ rất nhanh
  - ✓Dung lượng không lớn
  - ✓Sử dụng bộ nhớ bán dẫn RAM, ROM
- Các loại bộ nhớ
  - ✓Bộ nhớ chính (Main memory)
  - ✓Bộ nhớ Cache (Cache Memory) hay gọi bộ nhớ đệm

Bộ nhớ chính (main memory)

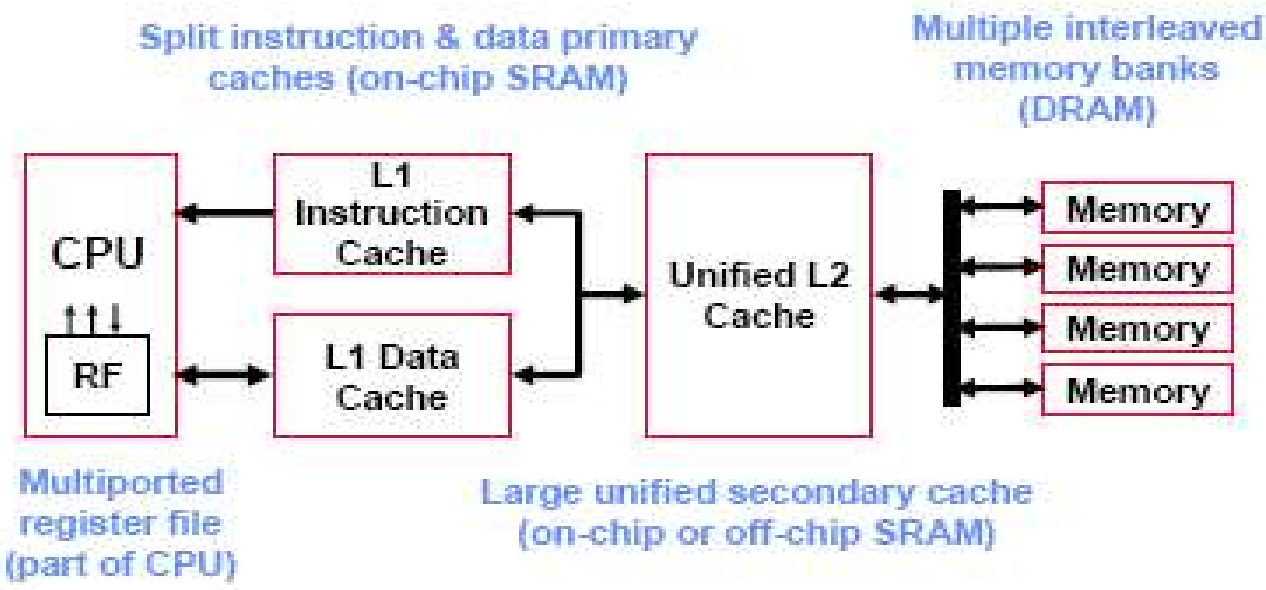
- Chứa chương trình và dữ liệu đang được sử dụng bởi CPU
- Bộ nhớ chính được tổ chức thành các ngăn nhớ và được đánh địa chỉ
- Ngăn nhớ thường được tổ chức theo byte
- Nội dung của một ngăn nhớ có thể thay đổi nhưng địa chỉ vật lý của nó đã được đánh là không thay đổi

Bộ nhớ đệm nhanh(cache memory)

- ✓ Đây là bộ nhớ bán dẫn có tốc độ nhanh và chúng được đặt đệm giữa CPU và bộ nhớ chính nhằm tăng tốc truy xuất của CPU tới bộ nhớ chính.
- ✓ Dung lượng nhỏ hơn rất nhiều bộ nhớ chính
- ✓ Tốc độ nhanh hơn rất nhiều lần
- ✓ Ngay nay Cache được tích hợp vào trong bộ vi xử lý và nó trong suốt với người sử dụng.
- ✓ Bộ nhớ Cache thông thường được chia ra thành 2 mức.
- ✓ Cache có thể có hoặc không

2. Bộ nhớ máy tính

Chi tiết cấu trúc bộ nhớ Cache



Bộ nhớ ngoài(External memory)

Chức năng và đặc điểm

- Lưu trữ tài nguyên phần mềm Máy tính.
- Được kết nối với hệ thống như thiết bị vào ra.
- Dung lượng rất lớn (vài trăm GB)
- Tốc độ chậm

Các loại bộ nhớ ngoài

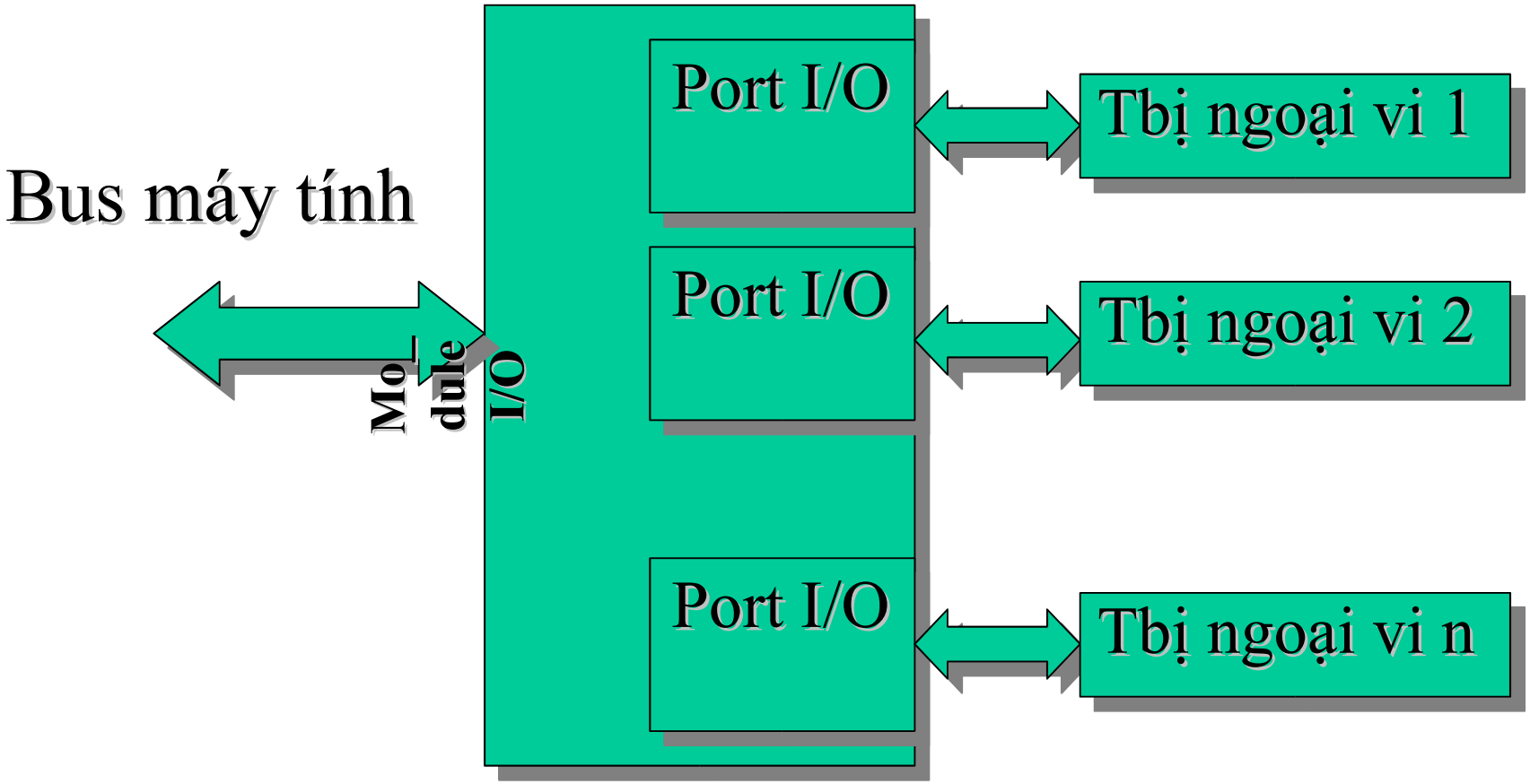
- Bộ nhớ từ: Đĩa cứng, đĩa mềm,…
- Bộ nhớ quang: CD, VCD, DVD,…
- Bộ nhớ bán dẫn: flash Disk, memory Card, pen Disk,…

Hệ thống vào ra (Input/Output System)

- Chức năng: trao đổi thông tin giữa máy tính với thế giới bên ngoài.
- Thao tác cơ bản
  - ✓ Vào dữ liệu (In)
  - ✓ Ra dữ liệu (Out)
- Các thành phần chính
  - ✓ Thiết bị ngoại vi (Peripheral Devices)
  - ✓ Các Module I/O (IO Module)

Hệ thống vào ra (Input/Output System)

- Cấu trúc vào ra cơ bản



Thiết bị ngoại vi (Peripherals)

Các thiết bị ngoại vi (Peripherals)

- *Chức năng*: chuyển đổi thông tin từ bên ngoài thành dữ liệu máy tính và ngược lại.
- *Các thiết bị ngoại vi cơ bản*:
  - ✓ Thiết bị vào: bàn phím, chuột, …
  - ✓ Thiết bị ra: máy in, màn hình,…
  - ✓ Thiết bị nhớ: đĩa từ, quang,….
  - ✓ Thiết bị truyền thông: Modem,…

Module vào ra

Chức năng: nối ghép thiết bị ngoại vi với máy tính

- ☞ Mỗi Module có 1 hay nhiều cổng vào ra
- ☞ Mỗi cổng được đánh địa chỉ xác định

Các thiết bị ngoại vi được kết nối với máy tính thông qua cổng vào ra (ví dụ: COM, LPT, USB, VGA,…)

2.2 Hoạt động của máy tính

1. Thực hiện chương trình

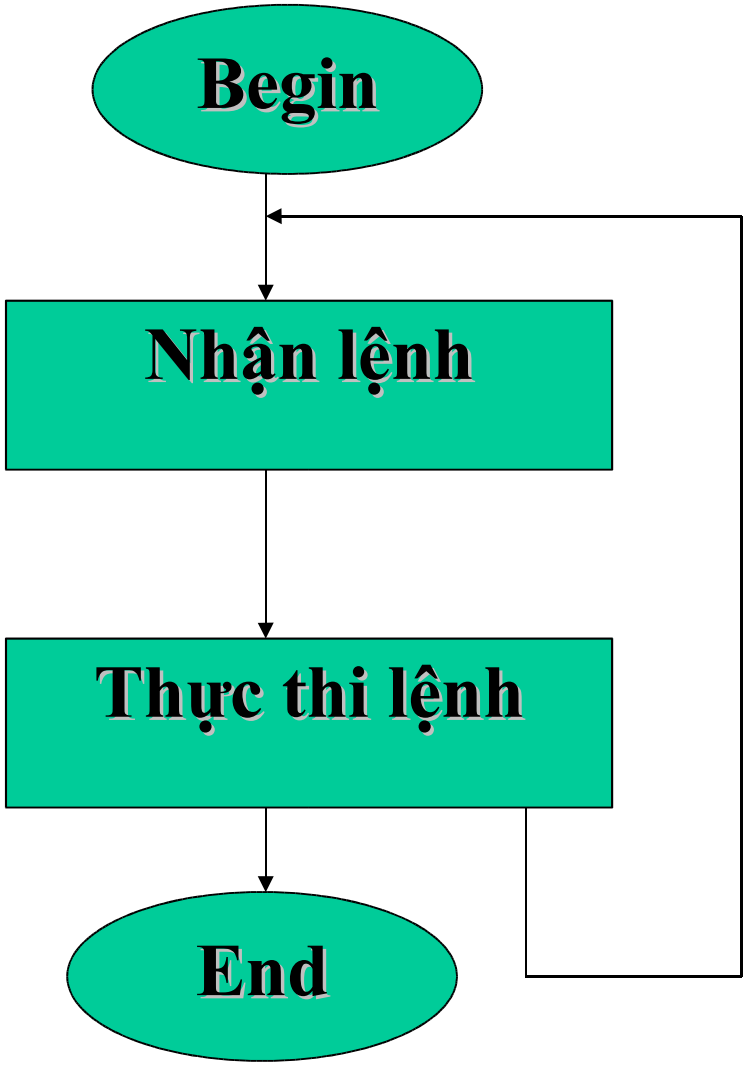
Là hoạt động cơ bản của Máy tính. Máy tính lặp đi lặp lại quá trình thực hiện lệnh gồm hai bước cơ bản:

- ✓ Nhận lệnh (Fetch)
- ✓ Thực hiện lệnh (Execute)

Thực hiện chương trình dừng khi:

- ✓ Mất nguồn
- ✓ Gặp lệnh dừng
- ✓ Gặp tình huống không giải quyết được(lỗi)

Chu kỳ thực hiện lệnh



1. Thực hiện chương trình

Nhận lệnh (Fetch)

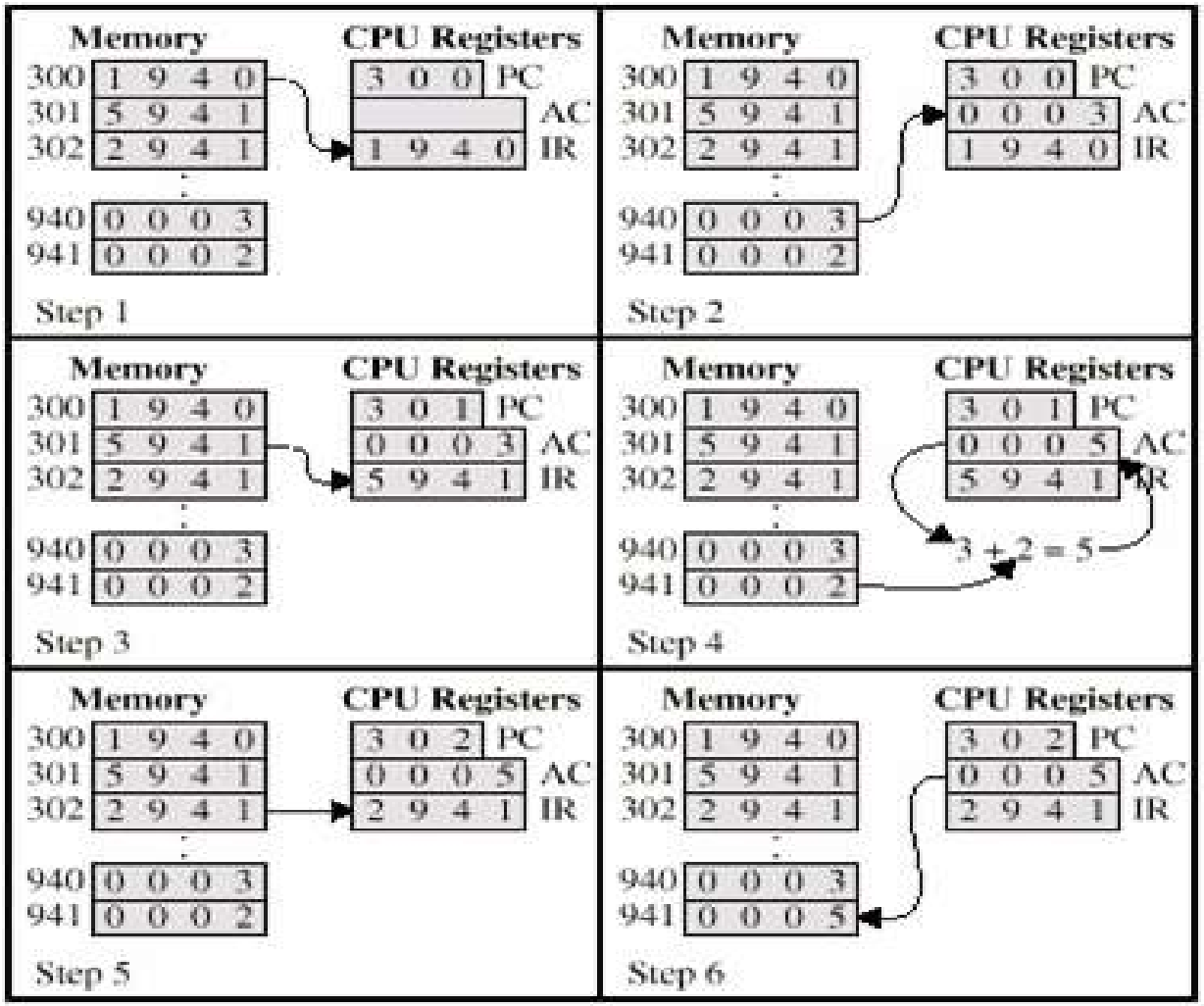
- Bắt đầu mỗi chu kỳ lệnh là CPU tiến hành lấy lệnh từ bộ nhớ chính. Trong quá trình lấy và thực hiện lệnh có 2 thanh ghi CPU mà ta quan tâm đó PC (Program Counter)và thanh ghi IR(Instruction Register)
- Bộ đếm chương trình thanh ghi PC giữ địa chỉ của lệnh *sẽ được nhận*.
- CPU lấy lệnh từ ngăn nhớ được *trỏ bởi PC* đưa vào thanh ghi lệnh IR lưu giữ
- Sau mỗi lệnh được nhận thì nội dung của thanh ghi PC tự động tăng để trỏ tới lệnh kế tiếp sẽ được thực hiện.

1. Thực hiện chương trình

Thực hiện (Execute)

- Bộ xử lý giải mã lệnh đã được nhận và phát tín hiệu điều khiển thực hiện thao tác mà lệnh yêu cầu.
- Thực hiện trao đổi giữa CPU và bộ nhớ chính
- Thực hiện trao đổi giữa CPU và Module I/O.
- Xử lý dữ liệu thực hiện các phép toán số học và logic.
- Điều khiển rẽ nhánh.
- Kết hợp các thao tác trên.

Ví dụ: Thực hiện chương trình



0001: loader

0010: store

0101: add

Ví dụ: Thực hiện chương trình

Evaluated **d = a + b × c**, when **a** = 5, **b** = 3 and **c** = 11

Program Pseudocode:	Assembly Code:	Meaning:
1. Multiply **b** with **c**	MOV R1, [1058H]	R1 ← [1058H]
	MOV R2, [1059H]	R2 ← [1059H]
	MUL R1, R2	R1 ← R1 * R2
2. Add the result of multiplication to **a**	MOV R3, [1057H]	R3 ← [1057H]
	ADD R1, R3	R1 ← R1 + R3
3. Save the result as **d**	STR R1, [105AH]	R1 → [105AH]

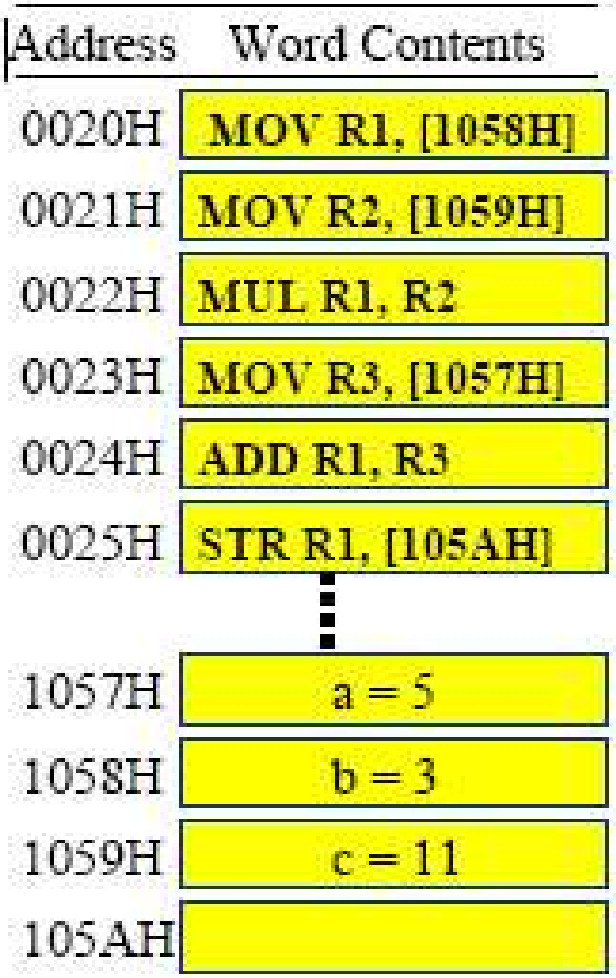
Ví dụ: Thực hiện chương trình

Following arithmetic expression to be evaluated for **d**,

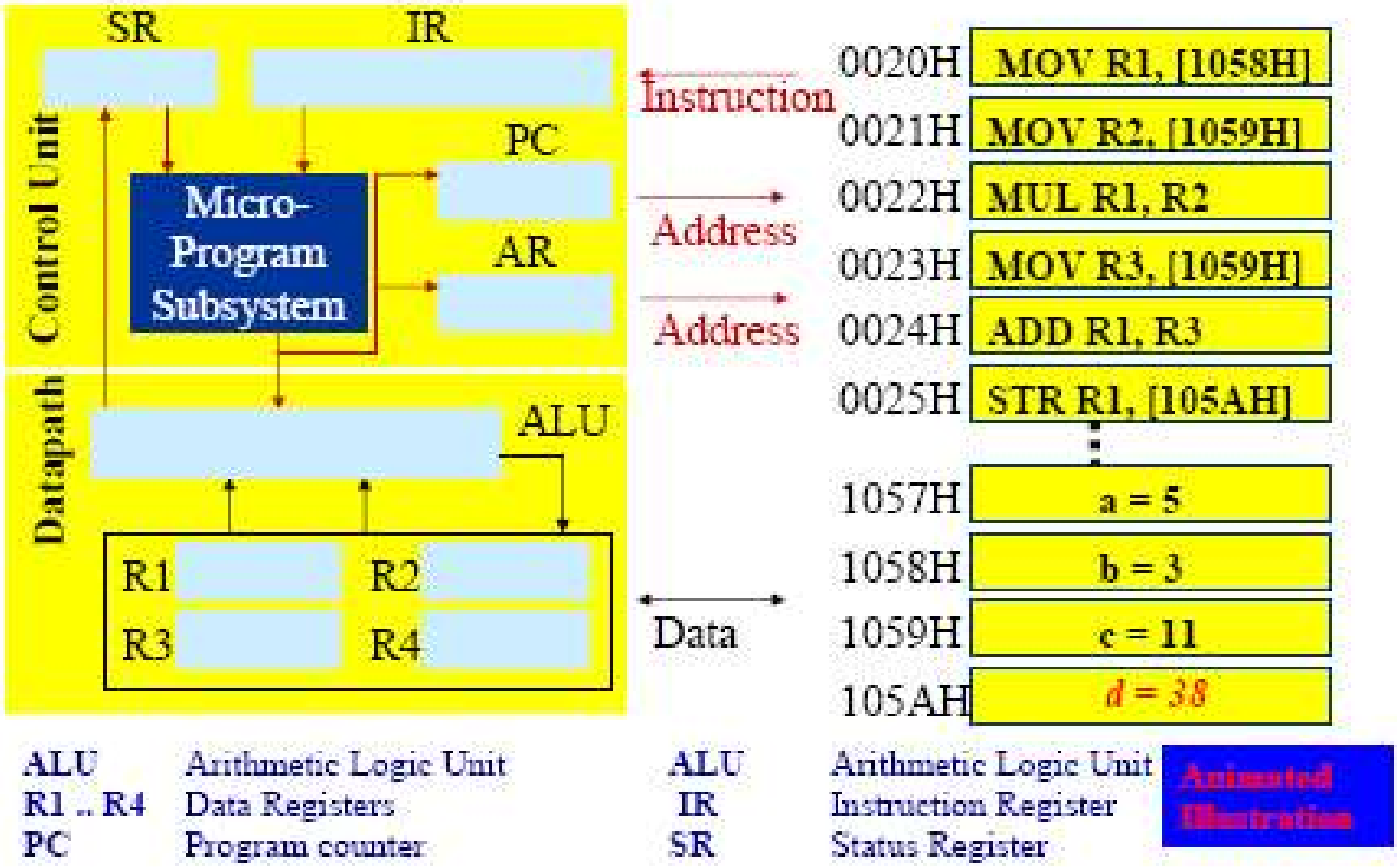
$$d = a + b * c$$

when **a** = 5, **b** = 3 and **c** = 11

Memory map shows where the program and input data are loaded



Ví dụ: Thực hiện chương trình



2. Ngắt (Interrupt)

Khái niệm chung về ngắt: Ngắt là cơ chế cho phép CPU tạm dừng chương trình đang thực hiện chuyển sang thực hiện một chương trình khác, gọi là chương trình con phục vụ ngắt.

Các loại ngắt

- ✓ Ngắt do lỗi thực hiện chương trình: chia cho 0
- ✓ Ngắt do lỗi phần cứng: lỗi RAM
- ✓ Ngắt do module I/O phát ra tín hiệu ngắt đến CPU yêu cầu trao đổi dữ liệu

Hoạt động của ngắt

2. Ngắt (Interrupt)

- ✓ Sau khi hoàn thành một lệnh, bộ xử lý kiểm tra tín hiệu ngắt.
- ✓ Nếu không có ngắt thì bộ xử lý tiếp tục nhận lệnh tiếp theo.
- ✓ Nếu có tín hiệu ngắt:

Tạm dừng chương trình đang thực hiện. Cất ngữ cảnh (thông tin có liên quan đến chương trình đang thực hiện).

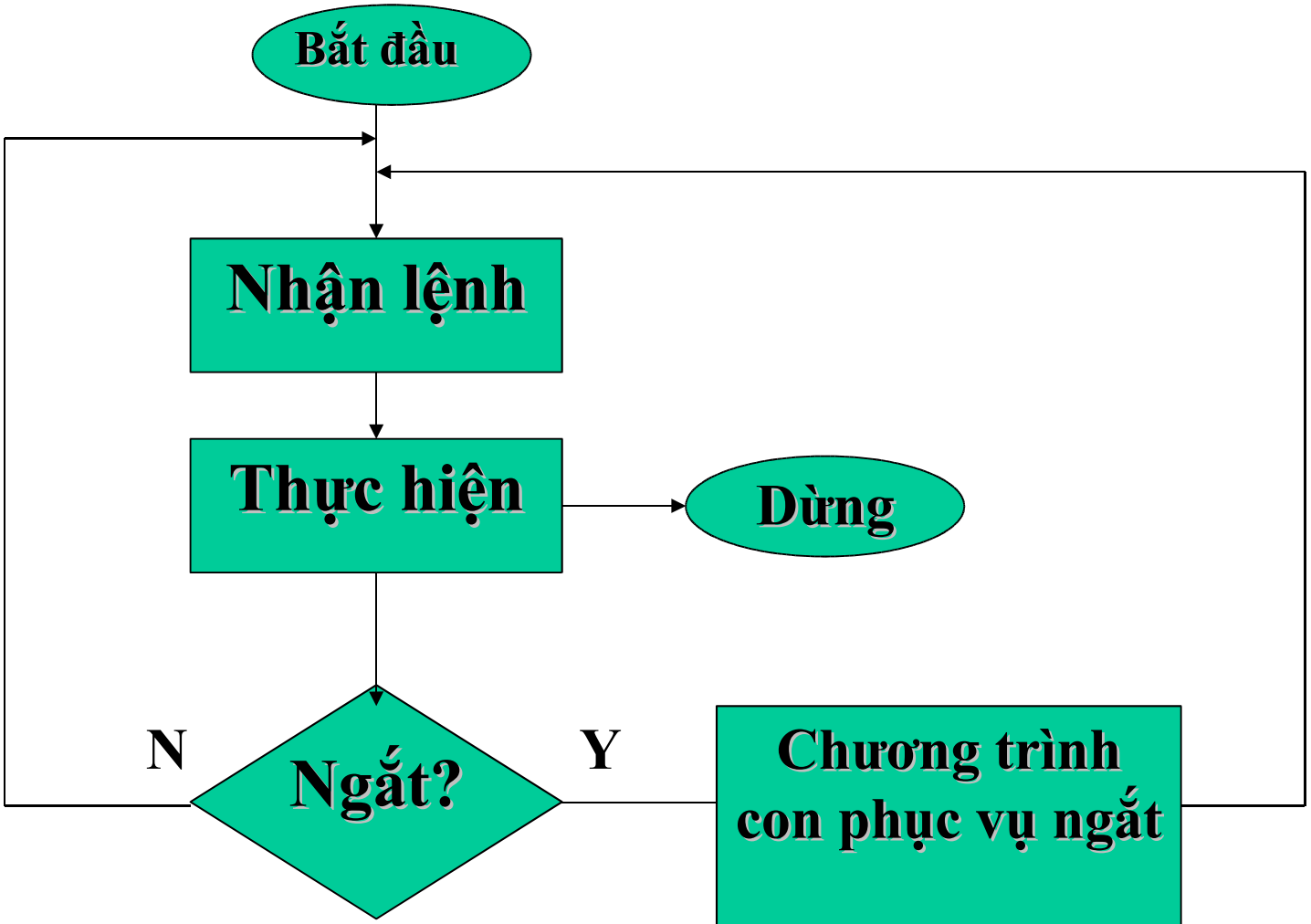
Thiết lập bộ đếm chương trình PC trỏ đến chương trình con phục vụ ngắt

Thực hiện chương trình con phục vụ ngắt.

Cuối chương trình con phục vụ ngắt. Khôi phục lại ngữ cảnh và tiếp tục chương trình đang bị tạm dừng.

2. Ngắt (Interrupt)

- **Chu kỳ lệnh với ngắt**



2. Ngắt (Interrupt)

Xử lý tín hiệu ngắt

- Cấm ngắt: Bộ xử lý bỏ qua các ngắt tiếp theo trong khi đang xử lý ngắt.
- Các ngắt vẫn đang đợi và được kiểm tra sau khi ngắt đầu tiên được thực hiện xong
- Các ngắt được thực hiện tuần tự nếu cùng thứ tự ưu tiên.
- Các ngắt trong máy tính máy tính được định nghĩa mức độ ưu tiên khác nhau.
- Ngắt có mức ưu tiên thấp có thể bị ngắt bởi ngắt có ưu tiên cao hơn. Vì vậy có thể xảy ra tình trạng ngắt lồng nhau

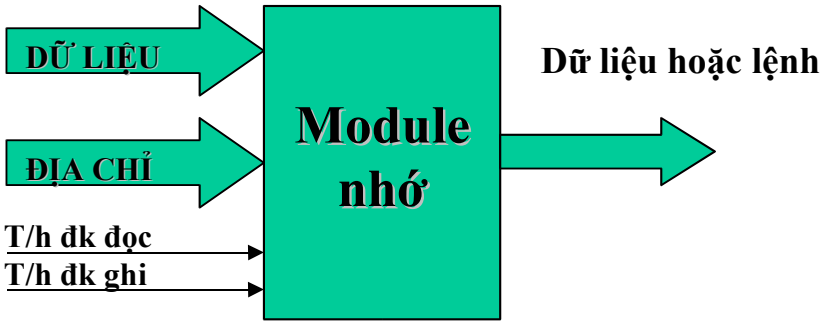
3. Hoạt động vào ra

- Là hoạt động trao đổi dữ liệu giữa thiết bị ngoại vi với bên trong máy tính
- Các kiểu hoạt động I/O: CPU trao đổi dữ liệu với module vào ra. Module vào ra trao đổi dữ liệu trực tiếp với bộ nhớ chính

2.3 Liên kết hệ thống

1. Thông tin các thành phần trong máy tính

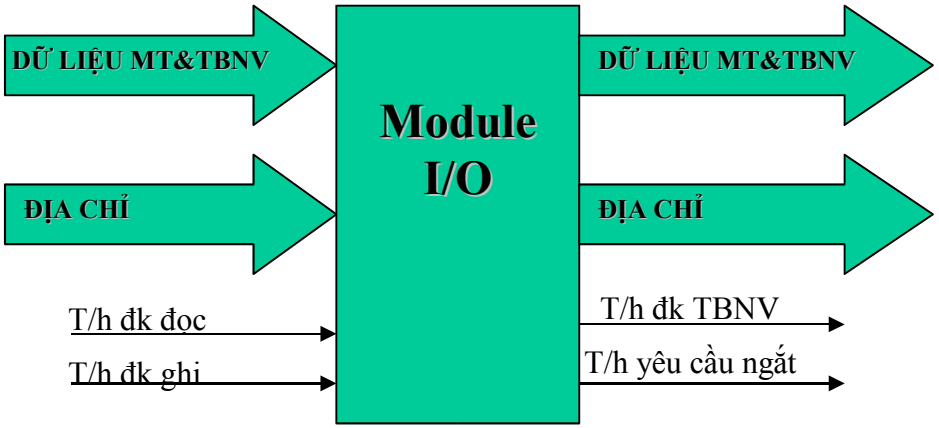
- *Kết nối Module nhớ bao gồm*



- ✓ Địa chỉ: nhận địa chỉ để xác định ngăn nhớ
- ✓ Dữ liệu: truyền nhận dữ liệu và lệnh từ bộ nhớ
- ✓ Tín hiệu điều khiển: Bao gồm tín hiệu điều khiển đọc và tín hiệu điều khiển ghi

2.3 Liên kết hệ thống

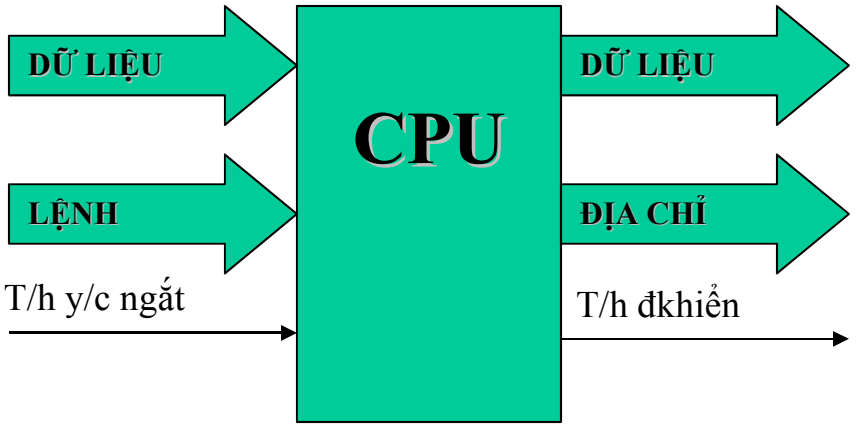
- *Kết nối Module I/O*



- ✓ Địa chỉ: nhận địa chỉ để xác định cổng vào ra
- ✓ Dữ liệu: nhận dữ liệu từ thiết bị ngoại vi, CPU hay bộ nhớ chính, đưa ra dữ liệu tới thiết bị ngoại vi, CPU hay bộ nhớ chính.
- ✓ Nhận các tín hiệu điều khiển từ CPU
- ✓ Phát tín hiệu điều khiển đến TBNV
- ✓ Phát tín hiệu yêu cầu của TBNV tới CPU

2.3 Liên kết hệ thống

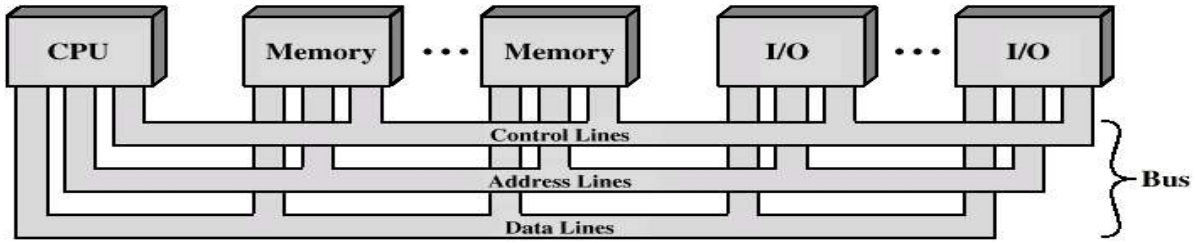
- *Kết nối CPU*



- ✓ CPU phát địa chỉ đến bộ nhớ hay Module vào ra.
- ✓ Đọc lệnh và dữ liệu
- ✓ Đưa dữ liệu ra sau khi xử lý
- ✓ Phát tín hiệu điều khiển đến Module nhớ hay Module vào ra
- ✓ Nhận các tín hiệu ngắt.

2. Cấu trúc BUS

- Khái niệm BUS: Bus là tập hợp các đường dây dùng để vận chuyển thông tin từ thành phần này tới thành phần khác bên trong máy tính.
- Độ rộng của BUS : là số đường dây có khả năng vận chuyển các bit thông tin đồng thời.
- Phân loại BUS: theo chức năng ta chia bus ra làm 3 loại: BUS địa chỉ, BUS dữ liệu và BUS điều khiển



2.3 Liên kết hệ thống

- **BUS địa chỉ :**

✓*Chức năng:* dùng để vận chuyển địa chỉ từ CPU đến các Module nhớ hay các Module vào ra, nhằm để xác định ngăn nhớ hay cổng vào ra nào cần truy xuất trao đổi thông tin. (đây là BUS một chiều).

Độ rộng của BUS địa chỉ $(A_0, A_1, \ldots, A_{n-1})$

Cho biết khả năng quản lý cực đại số các ngăn nhớ. Nếu sử dụng độ rộng bus địa chỉ n đường thì dung lượng cực đại của bộ nhớ có thể quản lý là 2^n ngăn nhớ hay tương đương với 2^n byte nhớ (nếu mỗi ngắn nhớ 1 byte)

2.3 Liên kết hệ thống

Ví dụ: Bus địa chỉ của một số bộ VXL là

8088/8086	n=20	2^{20}(1MB)
80286	n=24	2^{24}(16MB)
80386…Pentium	n=32	2^{32}(4GB)
Pentium II, III,IV	n=36	2^{36}(64GB)

2.3 Liên kết hệ thống

- **BUS dữ liệu:**

Chức năng: vận chuyển lệnh từ bộ nhớ -> CPU, vận chuyển dữ liệu giữa CPU, bộ nhớ và cổng vào ra.

Độ rộng của Bus dữ liệu ($D_0 D_1, \ldots D_{m-1}$)

Cho biết số byte có khả năng trao đổi đồng thời m=8,16,32,64,128 bit.

Ví dụ:

8088	->	m=8
8086	->	m=16
80386	->	m=32
Pentium	->	m=64

2.3 Liên kết hệ thống

- **BUS điều khiển:**
  Tập hợp các tín hiệu điều khiển gồm có

✓ Các tín hiệu phát ra từ CPU để điều khiển Module nhớ và Module vào ra.

✓ Các tín hiệu từ Module nhớ, Module vào ra gởi đến CPU yêu cầu.

✓ Ngoài ra còn là BUS cung cấp nguồn tín hiệu xung nhịp (clock) với các BUS đồng bộ.

✓ Một số tín hiệu điển hình

2.3 Liên kết hệ thống

- ✓ Tín hiệu (MemR) điều khiển đọc dữ liệu từ bộ nhớ từ ngăn nhớ xác định. (IOR) Tín hiệu đọc dữ liệu từ một cổng vào ra.
- ✓ Tín hiệu (MemW) điều khiển ghi dữ liệu có sẵn trên BUS dữ liệu đến một ngăn nhớ xác định. Tín hiệu điều khiển (IOW) ghi dữ liệu có sẵn ra cổng.
- ✓ Interrupt Request(INTR) tín hiệu yêu cầu ngắt từ các thiết bị ngoại vi
- ✓ Interrupt Acknowlegde(INTA) tín hiệu chấp nhận ngắt phát ra từ CPU
- ✓ Ngoài ra còn có các tín hiệu khác như: t/h yêu cầu và chấp nhận CPU chuyển nhượng BUS (BRQ,BGT),…

2.3 Liên kết hệ thống

***Đặc điểm của cấu trúc đơn BUS*.**

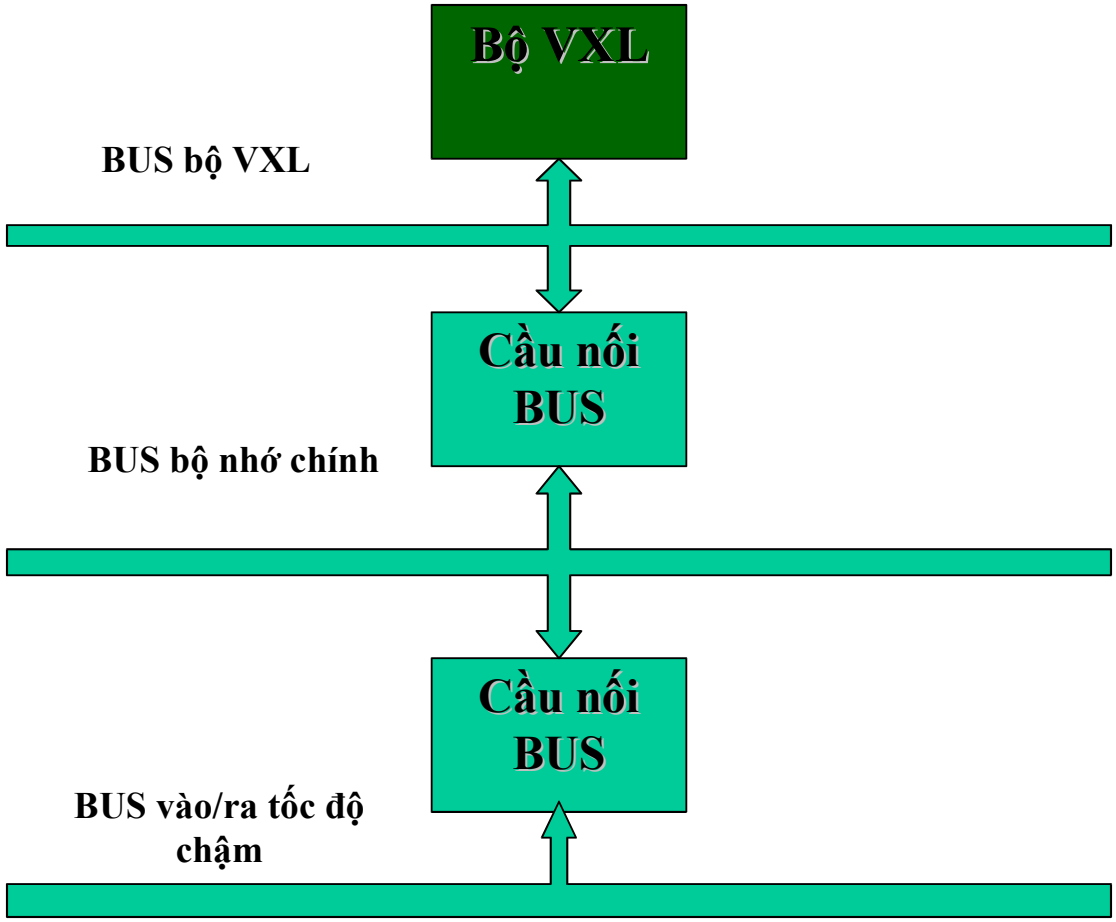
- Có nhiều thành phần nối vào một BUS chung.
- Tại một thời điểm chỉ phục vụ được một yêu cầu trao đổi dữ liệu.
- Các thành phần nối vào BUS có thể có tốc độ khác nhau.
- Các module nhớ và module vào ra phụ thuộc vào cấu trúc của CPU.

☛Khắc phục:

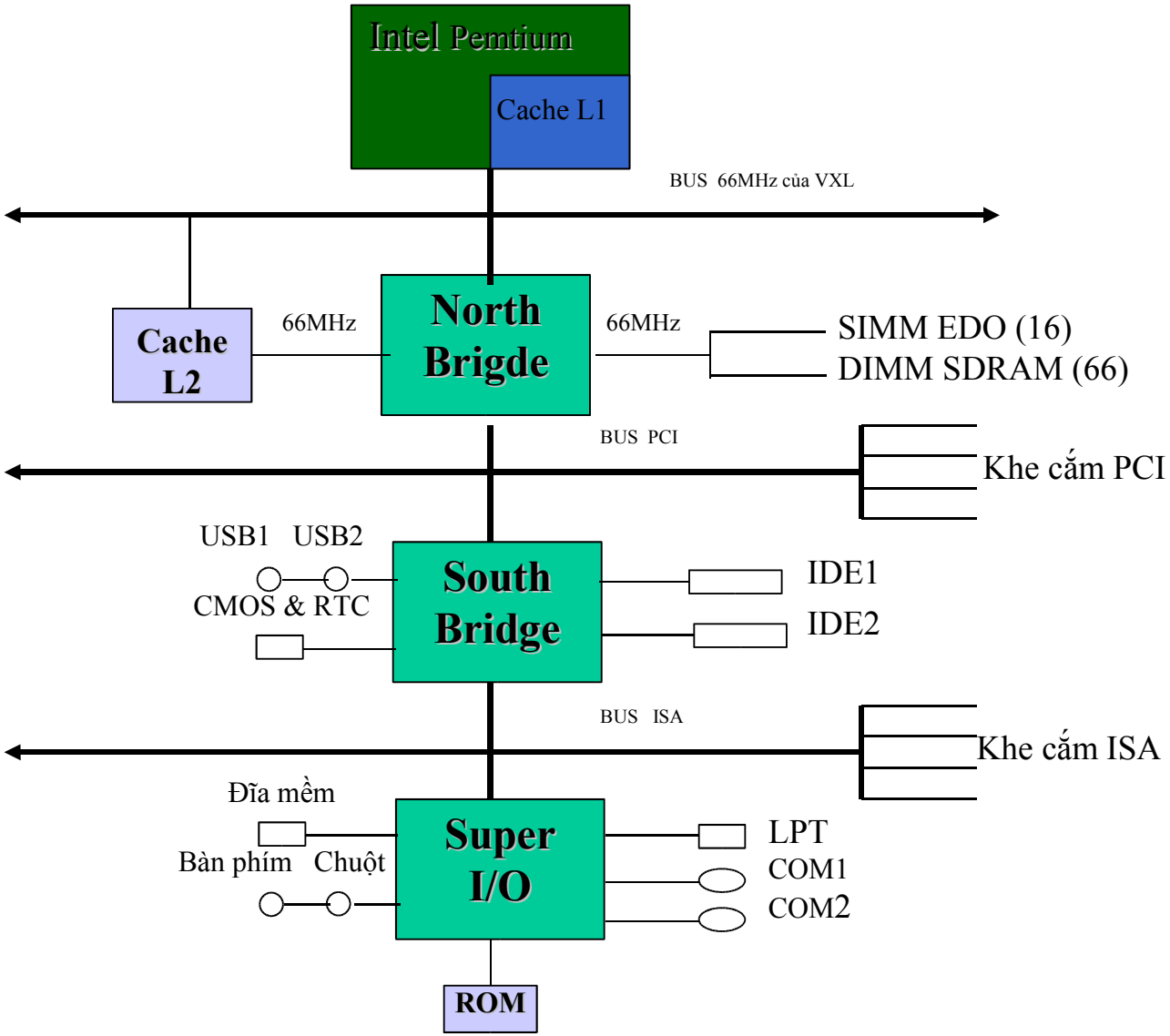
✓ Xây dựng cấu trúc đa BUS bao gồm các hệ thống BUS khác nhau về tốc độ.

✓ Trong hầu hết các máy PC bus được phân 3 cấp và các bus nối với nhau thông qua cầu nối BUS

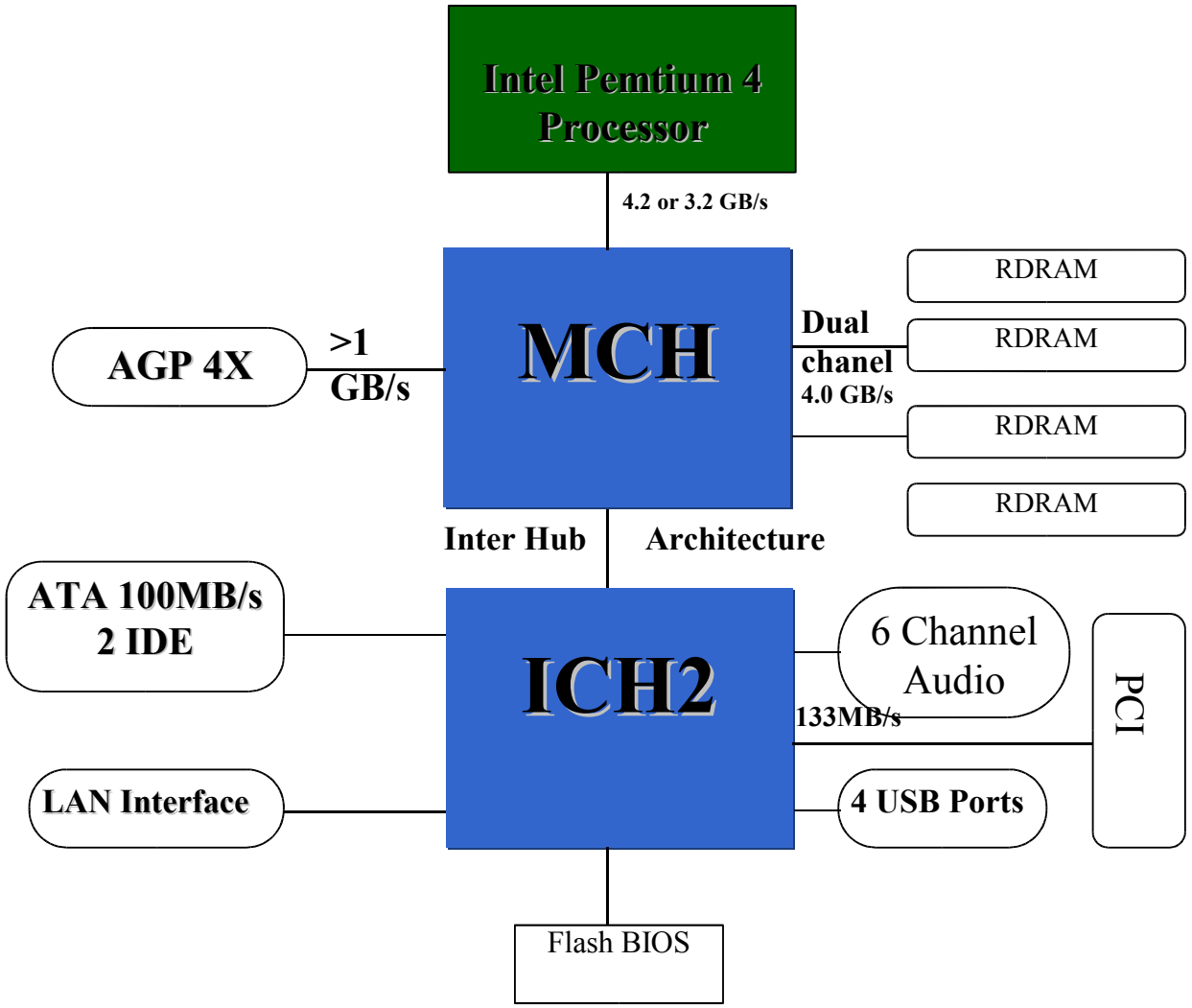
2.3 Liên kết hệ thống



Cấu trúc Pentium II điển hình



Cấu trúc Pentium 4



Ví dụ: Cấu hình một máy tính

- Intel MotherBoard D915PBLL, Socket T ATX 800FSB, DDR2 533, PCI-E 16x, SATA, 8ch Audio & LAN
- 3.2GHz Pentium IV processor.
- 512 MB DDRAM.
- 80 GB hard disk.
- keyboard and a mouse,
- foppy disk drive,
- 24x speed DVD drive,
- 19" monitor with 1280 x 1024 pixels resolution,
- 56 Kbit Modem,
- 100 Mbit Ethernet card.

Phần trao đổi và giải đáp

Tóm tắt chương 2

- Đặc điểm kiến trúc Von Neumann.
- Cấu trúc và chức năng cơ bản của hệ thống máy tính.
- Quy trình thực hiện chương trình trong máy tính.
- Ngắt là gì? Tại sao phải sử dụng ngắt trong hệ thống máy tính.
- BUS máy tính? Phân loại và chức năng BUS máy tính.
- Cấu trúc đa bus trong máy tính.
- Nhận diện được tất cả các thành phần phần cứng trong máy tính của bạn.

Chương 3

Biểu diễn dữ liệu và số học máy tính

3.1 Các hệ đếm cơ bản

3.2 Mã hoá và lưu trữ trong máy tính

3.3 Biểu diễn số nguyên

3.4 Số học nhị phân

3.5 Biểu diễn số dấu chấm động

3.6 Biểu diễn ký tự

3.1 Các hệ đếm cơ bản

- Hệ thập phân (Decimal System): con người sử dụng
- Hệ nhị phân (Binary System): máy tính sử dụng
- Hệ thập lục phân (Hexadecimal System): dùng biểu diễn rút ngắn số học nhị phân
- Cách chuyển đổi giữa các hệ đếm.

Hệ thập phân (decimal)

Bộ ký tự cơ sở gồm 10 số: 0…9

Dạng tổng quát: $a_{n-1}a_{n-2}a_{n-3}\ldots a_1a_0,a_{-1}\,a_{-2}\ldots a_{-m}$

$$A = \sum_{i=-m}^{n-1} a_i * 10^i \qquad \text{Trong đó } (a_i = 0\ldots9).$$

Ví dụ: 123,45

Phần nguyên : 123 : 10 = 12 dư 3

12 : 10 = 1 dư 2

1 : 10 = 0 dư 1

123

Phần phân : 0,45*10 = 4,5

0,5 *10 = 5

45

=123,45

Hệ nhị phân (Binary)
Hệ thập lục phân (Hexadecimal)

Hệ nhị phân(Binary)

Bộ ký tự cơ sở gồm 2 số: 0,1

Dạng tổng quát: $a_{n-1}a_{n-2}a_{n-3}\ldots a_1a_0,a_{-1}\,a_{-2}\ldots a_{-m}$

$$A = \sum_{i=-m}^{n-1} a_i * 2^i \qquad (a_i = 0,1)$$

Ví dụ: $11011,011_2 = 2^4+2^3+2^1+2^0+2^{-2}+2^{-3} = 27,375$

Thập lục phân (hexadecimal)

Bộ ký tự cơ sở: 0…9,A…F

Dạng tổng quát: $a_{n-1}a_{n-2}a_{n-3}\ldots a_1a_0,a_{-1}\,a_{-2}\ldots a_{-m}$

$$A = \sum_{i=-m}^{n-1} a_i * 16^i \qquad (a_i = 0..9, A..F)$$

Ví dụ: $89AB_H$ = 1000 1001 1010 1011B.

3.2 Mã hoá và lưu trữ trong máy tính

Nguyên tắc chung về mã hoá dữ liệu

Mọi dữ liệu được đưa vào máy tính được mã hoá thành số nhị phân.

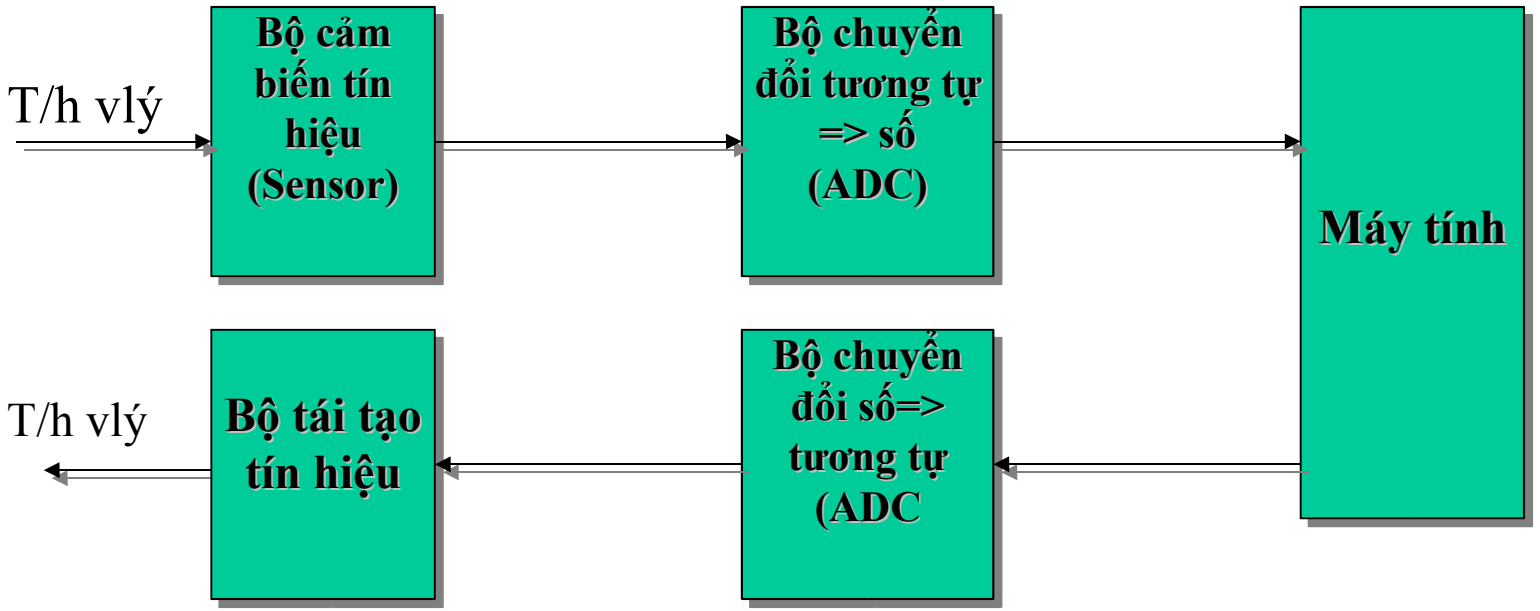
Các loại dữ liệu:

- Dữ liệu nhân tạo: do con người quy ước
- Dữ liệu tự nhiên: tồn tại khách quan với con người

Mã hoá dữ liệu nhân tạo

- Dữ liệu số nguyên: mã hoá theo một số chuẩn đã qui ước
- Dữ liệu số thực: mã hoá bằng số dấu chấm động
- Dữ liệu phi số (ký tự): mã hoá theo các bộ mã ký tự hiện hành như : ASCII, Unicode,…

Mô hình mã hoá và tái tạo tín hiệu vật lý



Các dữ liệu vật lý thông dụng

- ✓ Âm thanh
- ✓ Hình ảnh

Thứ tự lưu trữ các byte dữ liệu MT

- Bộ nhớ chính tổ chức lưu trữ dữ liệu theo đơn vị byte
- Độ dài từ dữ liệu có thể chiếm từ 1 đến 4 byte. Vì vậy cần phải biết thứ tự chúng lưu trữ trong bộ nhớ chính đối các dữ liệu nhiều byte.
- Có hai cách lưu trữ được đưa ra

✓ **Little Endian** (đầu nhỏ): Byte có ý nghĩa thấp hơn được lưu trữ trong bộ nhớ ở vị trí có địa chỉ nhỏ hơn.

✓ **Big Endian** (đầu to): Byte có ý nghĩa thấp hơn được lưu trữ trong bộ nhớ ở vị trí có địa chỉ lớn hơn.

Thứ tự lưu trữ các byte dữ liệu MT

☞ ***Ví dụ: lưu trữ một từ 32bit***

0001 1010 0010 1011 0011 1100 0100 1101B

1 A 2 B 3 C 4 D$_H$

Biểu diễn trong ngăn nhớ theo 2 cách

300	**4D**
301	**3C**
302	**2B**
303	**1A**

Little Endian

300	**1A**
301	**2B**
302	**3C**
303	**4D**

Big Endian

Thứ tự lưu trữ các byte dữ liệu MT

- Lưu trữ của các bộ vi xử lý điển hình
- Loại máy Intel: 80x86, Petium -> little endian
- Motorola 680x0 và các bộ xử lý RISC -> big endian
- Power PC & Itanium: tích hợp cả hai cách trên

3.3. Biểu diễn số nguyên

Máy tính biểu diễn số nguyên chia thành 2 loại

- Biểu diễn số nguyên không dấu (unsign integer)
- Biểu diễn số nguyên có dấu (sign integer)

Số nguyên không dấu:

Giả sử dùng *n* bit để biểu diễn số nguyên không dấu-> dải mà *n* bit biểu diễn được từ 0 -> 2^n-1. Giá trị của số nguyên đó được tính:

$$\sum_{i=0}^{n-1} a_i * 2^i$$

- Dải miền trị của số nguyên không dấu được biểu bằng hình tròn
- Giá trị nhỏ nhất bằng 0
- Giá trị lớn nhất bằng 2^n-1

Ví dụ: Số nguyên không dấu

- Ví dụ: n=8 $0 \ldots 2^8-1$ (255)
  n=16 $0 \ldots 2^{16}-1$ (65535).
  n=32 $0 \ldots 2^{32}-1$

Số nguyên có dấu

- Số bù một và số bù hai

ĐN: Cho một số nhị phân N được biểu diễn bởi n bit. Ta có

- ✓ Số bù một của N bằng $(2^n-1)-N$
- ✓ Số bù hai của N bằng 2^n-N

Ví dụ: Cho số N = 0001 0001_2 được biểu diễn bởi n=8bit. Xác định số bù 1 và bù 2 của N.

Ap dụng công thức

$$\begin{array}{rl} & 1111\ 1111 \quad (2^n-1) \\ - & 0001\ 0001 \quad N \\ \hline \textit{số bù một của } N & 1110\ 1110 \end{array}$$

Số nguyên có dấu

➔ Nhận xét: *số bù một của một số N được xác định bằng cách đảo các bit trong N*

Ap dụng công thức

$$\begin{array}{rr} & 1\ 0000\ 0000 \quad (2^n) \\ - & 0001\ 0001 \quad N \\ \hline \textit{số bù hai của N} & 1110\ 1111 \end{array}$$

➔ Nhận xét: *số bù hai của một số N được xác định bằng cách lấy số bù một của N cộng thêm 1*

Số bù 2 của N =(số bù 1 của N)+1

Số nguyên có dấu

Giả sử dùng *n* bit để biểu diễn số nguyên có dấu-> dải mà *n* bit biểu diễn được từ (- 2^{n-1} ..-1,0 .. 2^{n-1}-1). Giá trị của số nguyên đó được tính theo 2 phần riêng biệt:

- Phần giá trị dương (0 -> 2^{n-1}-1).
- Phần giá trị âm (- 2^{n-1}…-1).
- Dải miền trị của số nguyên có dấu được biểu bằng hình tròn
- Giá trị nhỏ nhất bằng - 2^{n-1}
- Giá trị lớn nhất bằng +2^{n-1}-1

Số nguyên có dấu

Trong đó: Bít có trọng số cao nhất (hay bit ngoài cùng bên trái của dãy nhị được máy tính sử dụng để biểu diễn dấu của giá trị) nếu:

= 0 : thì số nhị phân cần tính giá trị là số dương.

Dạng tổng quát là: $0a_{n-2}a_{n-3}\ldots a_0$

= 1 : thì số nhị phân cần tính giá trị là số âm.

Dạng tổng quát là: $1a_{n-2}a_{n-3}\ldots a_0$

Ví dụ 1

Ví dụ 1: *Cho số nguyên có dấu biểu diễn n=8bit sau:*

$A=B5_H$ *và* $B=6A_H$

Hãy xác định giá trị của hai số nguyên có dấu A và B dưới dạng hệ số người sử dụng

Bài giải

- Biểu diễn số nguyên A dưới dạng nhị phân

  $A=B5_H = 1011\ 0101_2$

  $=>A= -128 + 53 = -75$

- Biểu diễn số nguyên B dưới dạng nhị phân

  $B=6A_H = 0110\ 1010_2$

  $=> B = 64+32+8+2 = 106$

Ví dụ 2

Ví dụ 2: *Biểu diễn số nguyên có dấu sau đây A=+97 và B=-101 theo hai dạng kiểu n=8bit và n=16bit trong máy tính.*

Lời giải

✓ Biểu diễn số A dạng số nguyên có dấu trong máy tính

$A = 0110\ 0001_2$ (n=8bit)

✓ Biểu diễn số B dạng số nguyên có dấu trong máy tính

Biểu diễn số +101 = $0110\ 0101_2$

Lấy bù 2 $1001\ 1011_2$

=> B = - 101 = $1001\ 1011_2$

Ví dụ 2

- ✓ Biểu diễn số A dạng số nguyên có dấu trong máy tính

  $A = 0000\ 0000\ 0110\ 0001_2$ (n=16bit)

- ✓ Biểu diễn số B dạng số nguyên có dấu trong máy tính

  Biểu diễn số +101 $=0000\ 0000\ 0110\ 0101_2$

  Lấy bù 2 $1111\ 1111\ 1001\ 1011_2$

=> B =-101 = $1111\ 1111\ 1001\ 1011_2$

3.4 Biểu diễn số dấu chấm động

Cho hai giá trị:

Khối lượng mặt trời:
1990000000000000000000000000000000g

Khối lượng điện tử:
0.000000000000000000000000000910956g

Để lưu trữ con số này thì máy tính cần đến số bit rất lớn. Như vậy, trong trường hợp này thì loại số có dấu chấm tĩnh sẽ rất bất tiện. Vì vậy tất cả máy tính lưu trữ những số trên dưới dạng dấu chấm động (floating point) 1.990×10^{33} và 0.910956×10^{-27} hay theo số khoa học là : 1.999E+33 và 0.910956E-27.

3.4 Biểu diễn số dấu chấm động

➢ **Dạng tổng quát**

$M.R^E$ *Trong đó:* M (Matissa) phần định trị

R (Radix) cơ số

E(Exponent) số mũ

$X=(-1)^s\ 1.M\ 2^{E-B}$

Trong đó: s: là bit dấu (s=0 phần định trị là dương; s=1 phần định trị là âm)

M : là phần định trị.

E: là số mũ được dịch chuyển đi B đơn vị.

R đã được biết (R=2) máy tính lưu số dấu chấm động bao gồm hai thành phần chính

3.4 Biểu diễn số dấu chấm động

- Chuẩn IEEE 754-1985 phân định 3 dạng số dấu chấm động cơ bản

(IEEE: Institute of Electrical and Electronics Engineers)

  - Số có độ chính xác đơn dài 32 bit (single)
  - Số có độ chính xác kép dài 64 bit (double)
  - Số có độ chính xác mở rộng dài 128bit (quadruple)

3.4 Biểu diễn số dấu chấm động

Loại	Single	Double	Quadruple
Bề rộng của trường (bit)			
S	**1**	**1**	**1**
E	**8**	**11**	**15**
M	**23**	**52**	**111**
Tổng cộng	**32**	**64**	**128**
E cực đại	**255**	**2047**	**32767**
E cực tiểu	**0**	**0**	**0**
Độ dịch	**127**	**1023**	**16383**

Biểu diễn số dấu chấm động chuẩn IEEE 32bit

S=1 phần định trị là âm

S=0 phần định trị là dương

E: giá trị E nằm trong 8 bit, là số mũ được dịch chuyển đi 127

M: phần định trị, giá trị nằm trong 23 bit

Ta có số –2345,125 trong hệ thập phân. Hãy biểu diễn chúng dưới dạng chuẩn IEEE 32bit trong máy tính

3.4 Biểu diễn số dấu chấm động

B1: Chuyển đổi số trên ra hệ hai

-2345,125d = -1001 0010 1001.001b (dãy số nhị phân được biểu diễn bình thường)

B2: Chuẩn hoá theo IEEE 32bit

$-1.001\ 0010\ 1001\ 001 \times 2^{11}$

B3: Xác định các thông số biểu diễn s,M,E

S: phần định trị là số âm, nên s là 1

E : phần mũ được xác định e = E-127
=> E = 11+127=138=10001010

M: phần định trị được xác định là 001 0010 1001 0010 0000 0000 (số 32 bit)

3.4 Biểu diễn số dấu chấm động

☞ **Để thực một phép cộng hoặc trừ hai số dấu chấm động phải tiến hành theo các bước sau:**

- Tăng số mũ của số có số mũ nhỏ hơn cho bằng số có số mũ lớn hơn.
- Cộng (hoặc trừ) các phần định trị.
- Nếu cần thiết chuẩn hoá kết quả trả lại.

X_1 -> M_1 và E_1 để biểu diễn $X_1 = M_1 * R^{E1}$

X_2 -> M_2 và E_2 để biểu diễn $X_2 = M_2 * R^{E2}$

⇨ $X_1 * X_2 = (M_1 * M_2) * R^{E1+E2}$

⇨ $X_1 / X_2 = (M_1 / M_2) * R^{E1-E2}$

⇨ $X1 \pm X_2 = (M_1 * R^{(E1-E2)} \pm M_2) * R^{E2}$ (với giả thiết E1>E2)

3.4 Biểu diễn số dấu chấm động

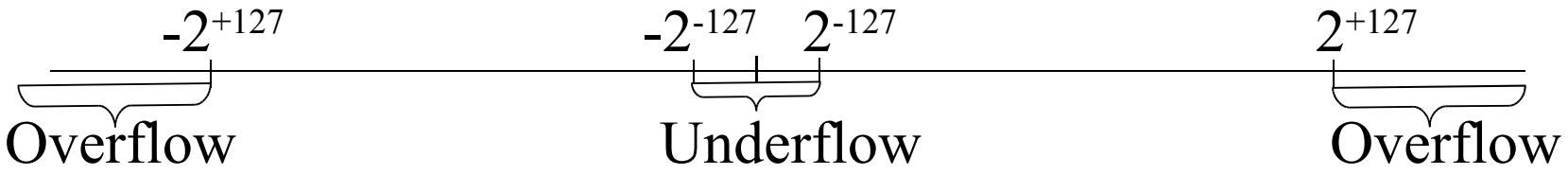
- **Một số quy ước**

Nếu e =255 và M<>0 -> không phải là số

Nếu e =255 và M=0 -> Giá trị âm hoặc dương vô cùng

Nếu e =0 và M=0 -> giá trị bằng 0

Dải biểu diễn: 2^{-127} đến 2^{+127} hay tương đương 10^{-38} đến 10^{+38}



3.5 Biểu diễn ký tự.

Có hai bộ mã thường sử dụng trên máy tính:

- Bộ mã ASCII
- Bộ mã Unicode

Bộ mã ASCII (American Standard Code for Information Interchange)

- ✓ Do ANSI (American National Standard Institute) thiết kế
- ✓ Bộ mã 8 bit -> mã hoá 2^8 ký tự có mã 00_H ->FF_H . Trong đó

3.5 Biểu diễn ký tự.

☞128 ký tự chuẩn cố định có mã **(00H -> 7FH)**

✓33 ký tự điều khiển không thể hiện thị nên màn hình. Bao gồm các ký tự điều khiển định dạng văn bản, điều khiển truyền số liệu và điều khiển phân cách thông tin.

✓ Còn lại các ký tự còn lại hiển thị được là bao gồm:

26 ký tự hoa **41h -> 5Ah**

26 ký tự thường **61h ->7Ah**

10 ký tự số **30h ->39h**

các dấu số học và ký tự đặc biệt.

3.5 Biểu diễn ký tự.

☞128 ký tự còn lại là ký tự mở rộng có thể thay đổi tuỳ ý nhà chế tạo máy tính hay người phát triển phần mền sử dụng vào những việc riêng. Có mã $\mathbf{80_H}$ -> $\mathbf{FF_H}$

Bộ mã hợp nhất Unicode:

☞ Do các hãng máy tính hàng đầu thế giới kết hợp thiết kế.

☞Bộ mã 16 bit có thể xây dựng bộ mã toàn cầu 2^{16} ký tự với 128 ký tự đầu có mã trùng mã trong bảng mã ASCII.

☞Có hỗ trợ các ký tự Tiếng Việt

Phần trao đổi và giải đáp

Ôn tập chương 3

- Các hệ đếm, ý nghĩa của chúng trong máy tính.
- Mã hóa dữ liệu trong máy tính.
- Mã hóa số nguyên(mã hóa số nguyên có và không dấu).
- Mã hóa số thực (số dấu chấm động)
- Mã hóa ký tự (ASCII, Unicode)
- Lưu trữ dữ liệu trong máy tính

Đặt câu hỏi

Câu 1: Kết quả hiển thi lên màn hình là bao nhiêu? Khi thực hiện đoạn lệnh sau:

```
Var a: shortint;
Begin
        a:=-1;
        writeln('Gia tri a:=',a);
        writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)]);
End.
```

Đặt câu hỏi

Câu 2: Kết quả hiển thi lên màn hình là bao nhiêu? Khi thực hiện đoạn lệnh sau:

```
Var a: shortint;
Begin
        a:=-128;
        writeln('Gia tri a:=',a);
        writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)]);
End.
```

Đặt câu hỏi

Câu 3: Kết quả hiển thi lên màn hình là bao nhiêu? Khi thực hiện đoạn lệnh sau:

```
Var a: shortint;
Begin
        a:=$6A;
        writeln('Gia tri a:=',a);
        writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)]);
End.
```

Đặt câu hỏi

Câu 4: Kết quả hiển thi lên màn hình là bao nhiêu? Khi thực hiện đoạn lệnh sau:

```
Var    b : integer absolute 3715:100;
       a : shortint absolute 3715:100;
Begin
   b:=$00B5;
   writeln('Gia tri a:=',a);
   writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)]);
End.
```

Đặt câu hỏi

Câu 5: Kết quả hiển thi lên màn hình là bao nhiêu? Khi thực hiện đoạn lệnh sau:

```
Var     b : integer absolute 3715:100;
        a: shortint absolute 3715:100;
Begin
    b:=-75;
    writeln('Gia tri a:=',a);
    writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)]);
    writeln('Gia tri ngan nho:=', mem[seg(a):ofs(a)+1]);
    writeln('Gia tri ngan nho:=', memw[seg(a):ofs(a)]);
End.
```

Chương 4

Bộ xử lý trung tâm

4.1 Cấu trúc của CPU

4.2 Tập lệnh (Instruction File)

4.3 Hoạt động của CPU

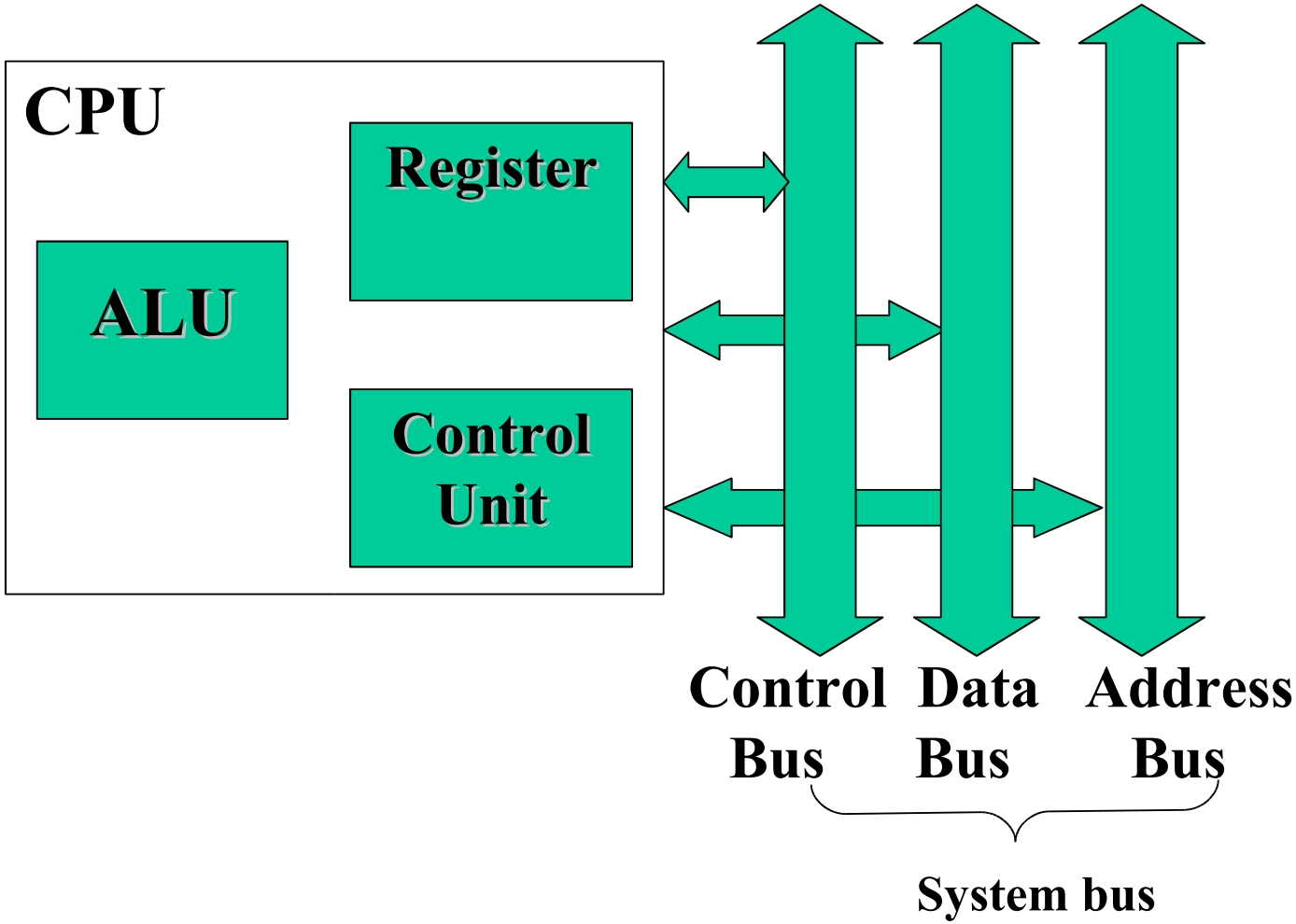
4.4 Kiến trúc Intel

4.1 Cấu trúc của CPU

Nhiệm vụ CPU: thực hiện lệnh của chương trình.

- Nhận lệnh (Fetch Instruction): CPU nhận lệnh từ bộ nhớ
- Giải mã lệnh(Decode Instruction): Xác định thao tác mà lệnh yêu cầu
- Nhận dữ liệu(Fetch Data): Nhận dữ liệu từ bộ nhớ hay cổng vào ra
- Xử lý dữ liệu(Process Data): thực hiện các phép toán số học và logic đối với dữ liệu
- Ghi dữ liệu (Write Data): Ghi dữ liệu ra bộ nhớ hay cổng vào ra.

a. Cấu trúc CPU

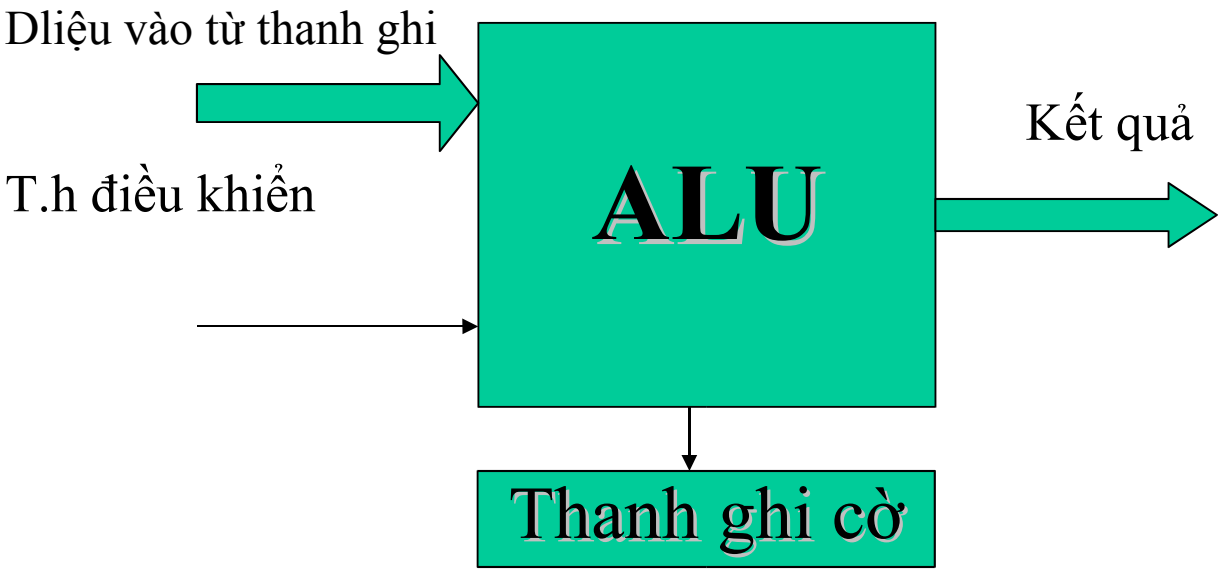


b. Đơn vị số học và logic (ALU)

Chức năng: thực các phép toán số học và logic

- Số học: cộng, trừ, nhân, chia, tăng, giảm, đảo,..
- Logic: AND, OR,XOR, NOT, dịch bit,…

- Mô hình kết nối của ALU



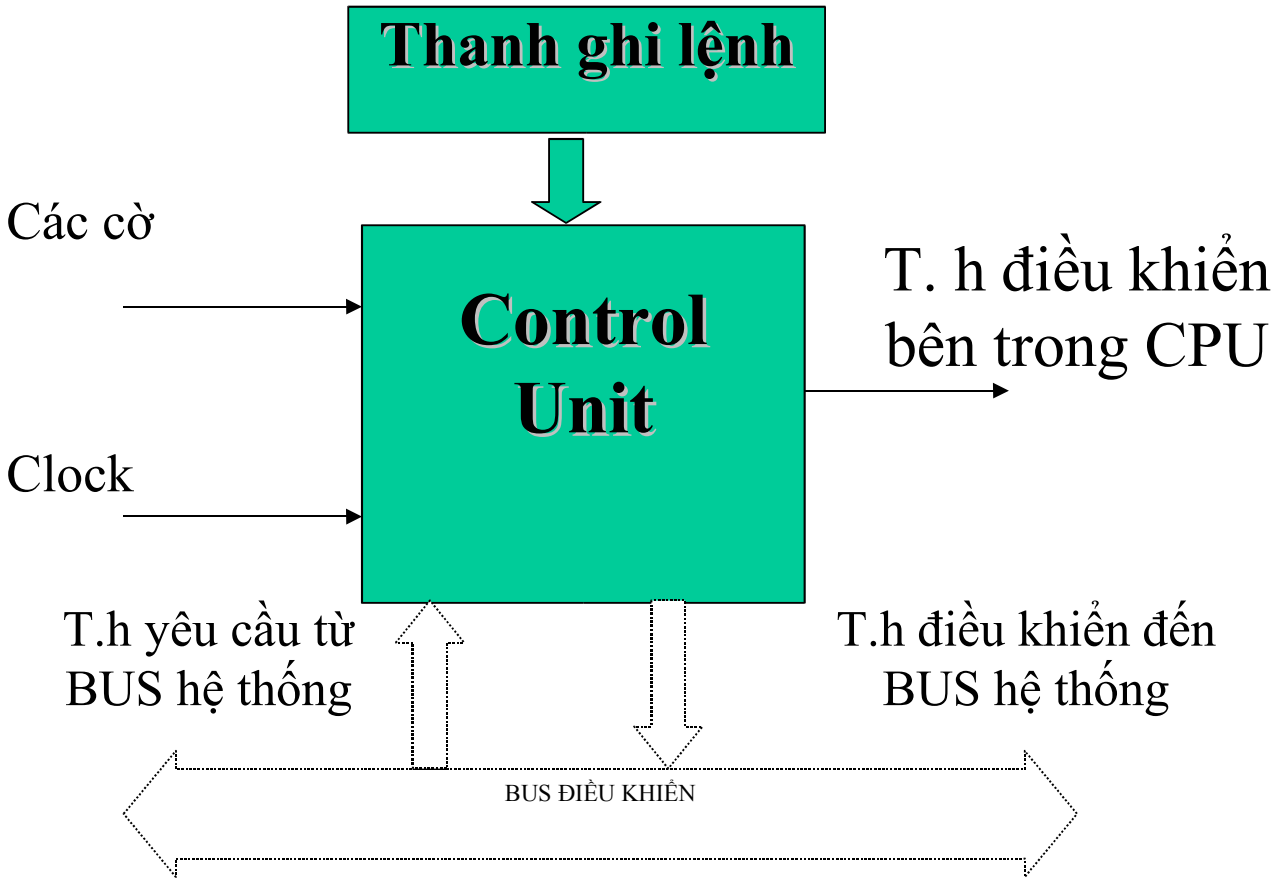
c. Đơn vị điều khiển

Chức năng:

- ✓ Nhận lệnh từ bộ nhớ đưa vào thanh ghi lệnh IP.
- ✓ Tăng nội dung thanh ghi PC mỗi khi nhận lệnh song
- ✓ Giải mã lệnh và xác định thao tác mà lệnh yêu cầu
- ✓ Phát ra tín hiệu điều khiển thực thi lệnh.
- ✓ Nhận các tín hiệu yêu cầu từ BUS hệ thống và giải quyết đáp ứng yêu cầu đó.

c. Đơn vị điều khiển

- **Mô hình kết nối đơn vị điều khiển**



c. Đơn vị điều khiển

- **Các thông tin kết nối đến CU**

✓ Clock: tín hiệu xung nhịp từ mạch tạo dao động.

✓ Mã lệnh từ thanh ghi lệnh đưa đến CU giải mã

✓ Các trạng thái cờ đưa đến cho biết trạng thái của CPU cũng như trạng thái thực hiện các phép toán trong ALU.

✓ Các tín hiệu điều khiển từ BUS điều khiển.

✓ Các tín hiệu điều khiển bên trong CPU: điều khiển thanh ghi, ALU.

✓ Các tín hiệu điều khiển bên ngoài CPU đó là Bộ nhớ hay cổng vào ra

4.2 Tập thanh ghi

- **Chức năng**

✓ Thực chất là vùng nhớ được CPU nhận biết qua tên thanh ghi và có tốc độ truy xuất cực nhanh.

✓ Chứa thông tin tạm thời phục vụ cho hoạt động ở thời điểm hiện tại của CPU

✓ Số lượng thanh ghi tuỳ thuộc vào bộ vi xử lý cụ thể -> tăng hiệu năng CPU

✓ Thanh ghi chia 2 loại: Loại lập trình được và loại không lập trình được

4.2 Tập thanh ghi

- **Phân loại thanh ghi theo chức năng**

✓ Thanh ghi địa chỉ: Thanh ghi được sử dụng để quản lý địa chỉ của ngăn nhớ hay cổng vào ra.

✓ Thanh ghi dữ liệu: Thanh ghi dùng để lưu trữ dữ liệu tạm thời

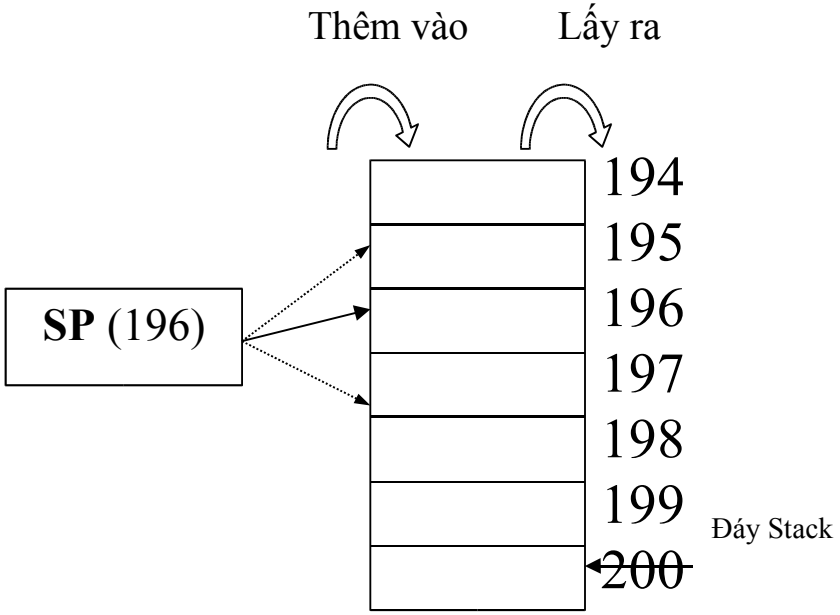
✓ Thanh ghi đa năng: Thanh ghi có thể chứa dữ liệu hoặc địa chỉ đều được.

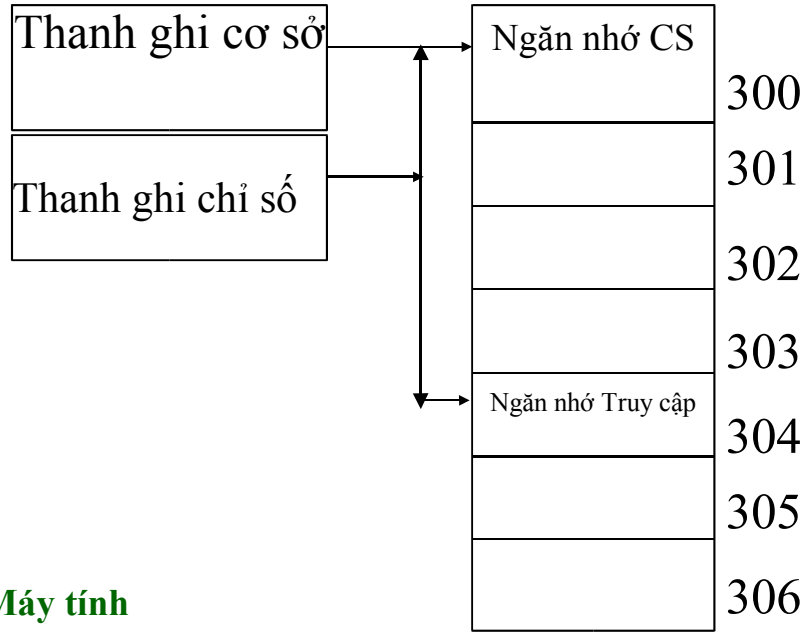
✓ Thanh ghi điều khiển/trạng thái: Thanh ghi chứa thông tin về trạng thái CPU.

✓ Thanh ghi lệnh: thanh ghi chứa lệnh đang được thực hiện

4.2 Tập thanh ghi

- **Một số thanh ghi điển hình**
  - ✓ Bộ đếm chương trình PC
  - ✓ Ngăn xếp SS (Stack)
  - ✓ Con trỏ ngăn xếp SP





4.2 Tập thanh ghi

- **Các thanh ghi dữ liệu**

✓ Chứa các dữ liệu tạm thời và kết quả trung gian.

✓ Các thanh ghi số nguyên 8,16,32,64 bit.

✓ Các thanh ghi số dấu chấm động.

- Thanh ghi trạng thái (State Register) &Thanh ghi cờ (FR: Flag Register).

✓ Chứa các thông tin trạng thái của CPU.

✓ Các cờ phép toán báo hiệu trạng thái kết quả.

✓ Các cờ điều khiển báo hiệu trạng thái của CPU

✓ Ví dụ: cờ ZF, CF, SF, OF, IF (cờ ngắt =1 cho ngắt =0 cấm ngắt

4.3 Tập lệnh

Giới thiệu chung về tập lệnh

- Mỗi bộ xử lý có tập lệnh xác định.
- Một tập lệnh thường đến vài chục đến vài nghìn lệnh
- Mỗi lệnh là chuỗi nhị phân mà bộ xử lý có thể phân tích và hiểu để thực hiện thao tác mà lệnh đó yêu cầu.
- Các lệnh khi viết thường được ánh xạ ra thành các ký hiệu gợi nhớ. ADD, MOV, IN, OUT, SHL, ROL,…
- Đây tựa của câu lệnh của hợp ngữ

  (Lập ASSEMBLY)

4.3 Tập lệnh

- Các thành phần một lệnh máy 2 phần:

- Mã thao tác (Operation Code: **Opcode**): Mã chỉ ra thao tác mà bộ vi xử lý cần phải thực hiện.
- Địa chỉ toán hạng (**Operand Address**): Chỉ ra nơi chứa các toán hạng mà mã thao tác sẽ tác động.
  - Toán hạng nguồn: dữ liệu vào của thao tác
  - Toán hạng đích: dữ liệu ra của thao tác

4.3 Tập lệnh

Các kiểu thao tác

- Thao tác chuyển dữ liệu
- Thao tác xử lý số học và logic
- Thao tác vào ra dữ liệu qua cổng
- Thao tác điều khiển rẽ nhánh
- Thao tác điều khiển hệ thống
- Thao tác xử lý số dấu chấm động
- Thao tác chuyên dụng khác: xử lý ảnh, âm thanh, tiếng nói,…

4.3 Tập lệnh

Các lệnh chuyển dữ liệu

- **Lệnh Mov**

  Sao chép dữ liệu từ toán hạng nguồn -> đích
- **Lệnh Load**

  Nạp dữ liệu từ bộ nhớ -> bộ xử lý
- **Lệnh XCHG**

  Trao đổi nội dung của hai toán hạng cho nhau
- **Lệnh PUSP**

  Cất nội dung của một toán hạng nguồn vào stack
- **Lệnh POP**

  Lấy nội dung ở đỉnh Stack ra toán hạng đích
- **Lệnh Set, Clear**

4.3 Tập lệnh

- **Các lệnh số học**
  - **Lệnh ADD** : cộng
  - **Lệnh SUB** : trừ
  - **Lệnh MUL**: nhân
  - **Chia DIV** : chia

- **Các lệnh logic**
  - **Lệnh Test**
    Thực hiện lệnh AND thiết lập cờ
  - **Lệnh Shift**
    Dịch trái, hoặc phải
  - **Lệnh Rotate**
    Quay trái hoặc quay phải
  - **Lệnh Convert**
    Chuyển đổi dữ liệu từ dạng này sang dạng khác
  - **Lệnh AND, OR, XOR, NOT,….**

4.3 Tập lệnh

- **Các lệnh vào ra**
  - **Lệnh Input**:
  - **Lệnh Output**
- **Các lệnh chuyển điều khiển**
  - **Lệnh Jump**
  - **Lệnh Call**
  - **Lệnh Return** : trở về từ chương trình con
- **Các lệnh điều khiển hệ thống**
  - **Lệnh Halt** : dừng thực hiện chương trình
  - **Lệnh Wait** : tạm dừng thực hiện chương trình, lặp kiểm tra cho đến khi thoả mãn thì tiếp tục thực hiện
  - **No Operation**: không thực hiện gì cả
  - **Lệnh Lock** : Cấm không cho chuyển nhượng BUS
  - **Lệnh Unlock**: cho phép chuyển nhượng BUS

Các phương pháp định địa chỉ (Addressing Models)

- **Toán hạng của của lệnh có thể là:**
  - Một thanh ghi cụ thể
  - Nội dung của thanh ghi
  - Nội dung của ngăn nhớ hay cổng vào ra
- **Các phương pháp định địa chỉ thông dụng:**
  - Định địa chỉ tức thời
  - Định địa chỉ thanh ghi
  - Định địa chỉ trực tiếp
  - Định địa chỉ gián tiếp qua thanh ghi
  - Định địa chỉ gián tiếp
  - Định địa chỉ dịch chuyển

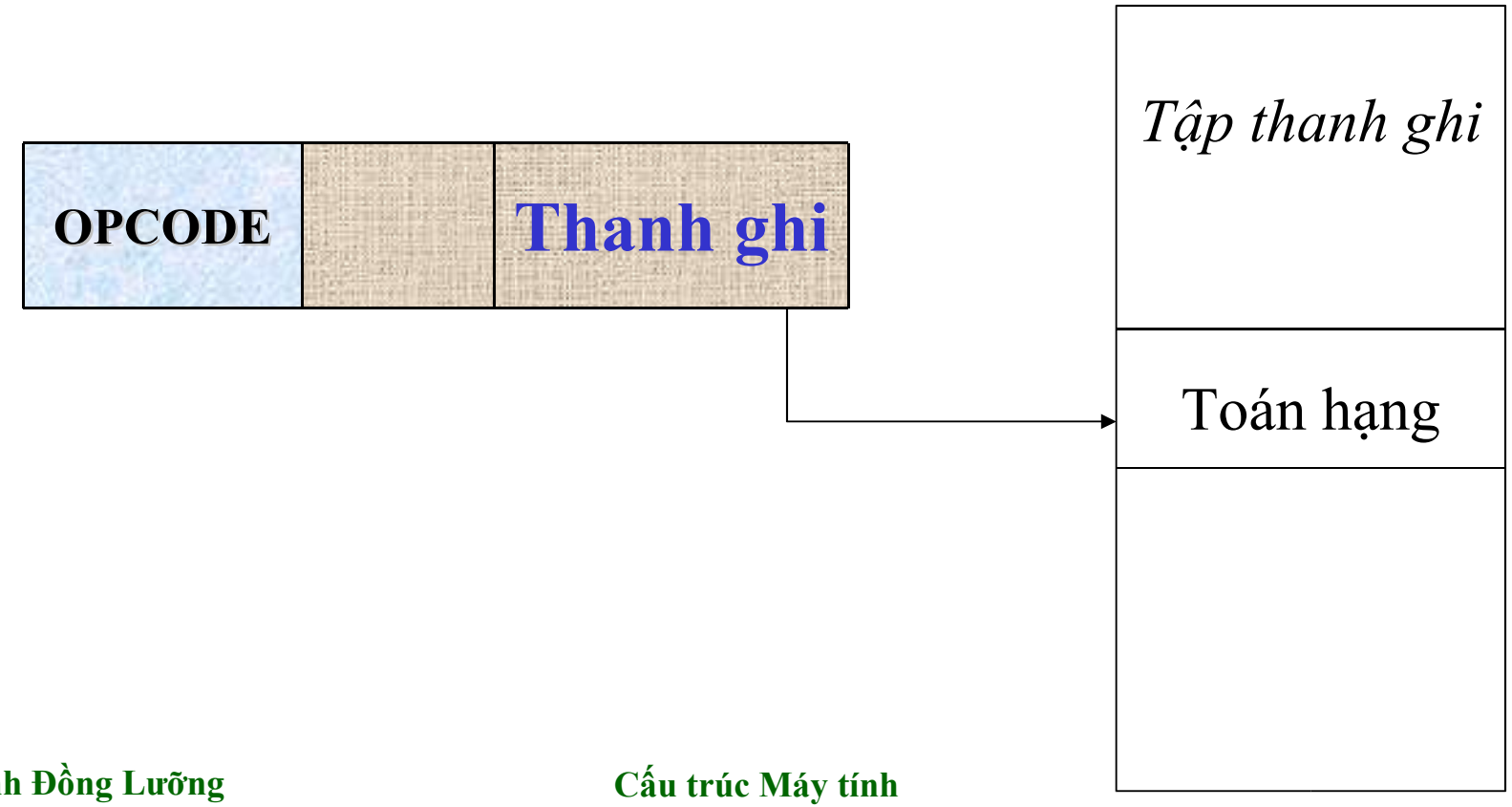
Định địa chỉ tức thì

- Toán hạng là hằng số nằm ngay trong trường địa chỉ toán hạng
- Đây chỉ có thể là toán hạng nguồn
- Không tham chiếu bộ nhớ
- Truy cập toán hạng rất nhanh
- Dải giá trị toán hạng bị hạn chế

ADD R1, const

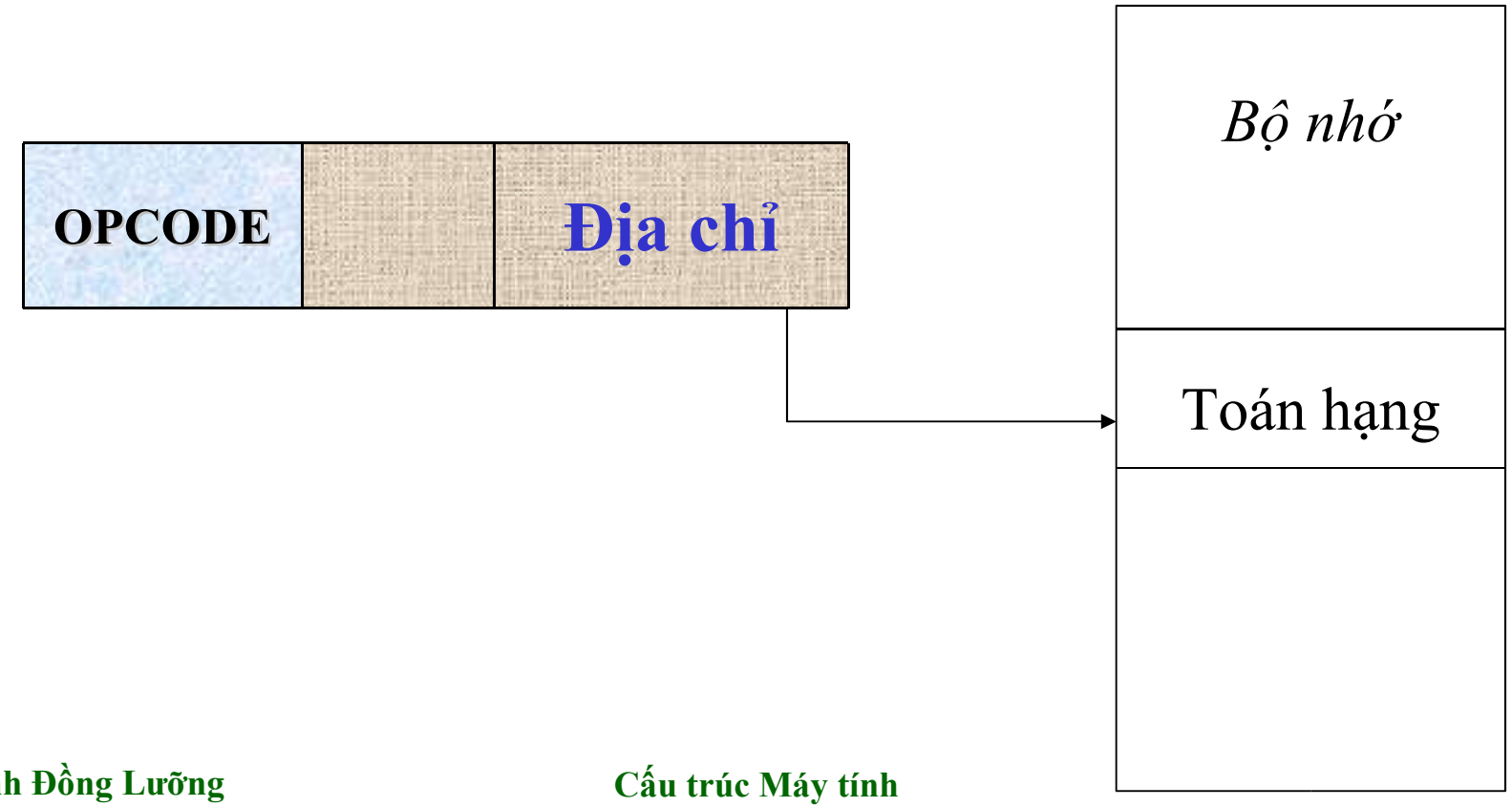
Định địa chỉ thanh ghi

- Toán hạng được chứa trong một thanh ghi, thanh ghi có tên trong trường địa chỉ toán hạng.



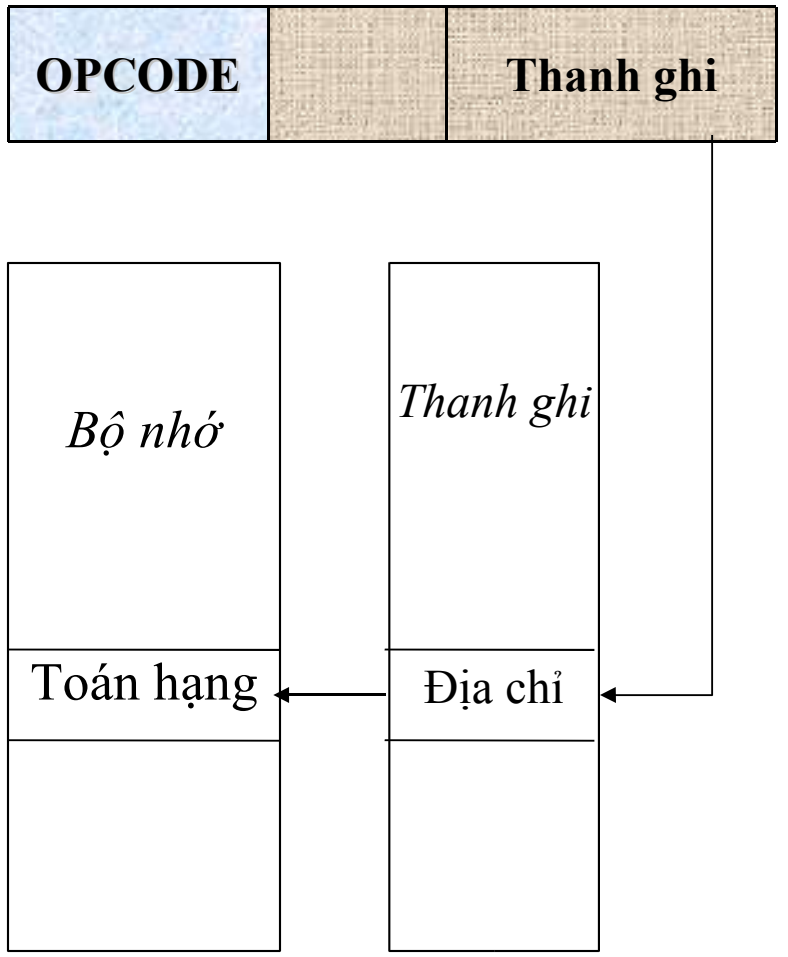
Định địa chỉ trực tiếp

- Toán hạng là ngăn nhớ có địa chỉ được chỉ ra ngay trong trường địa chỉ toán hạng

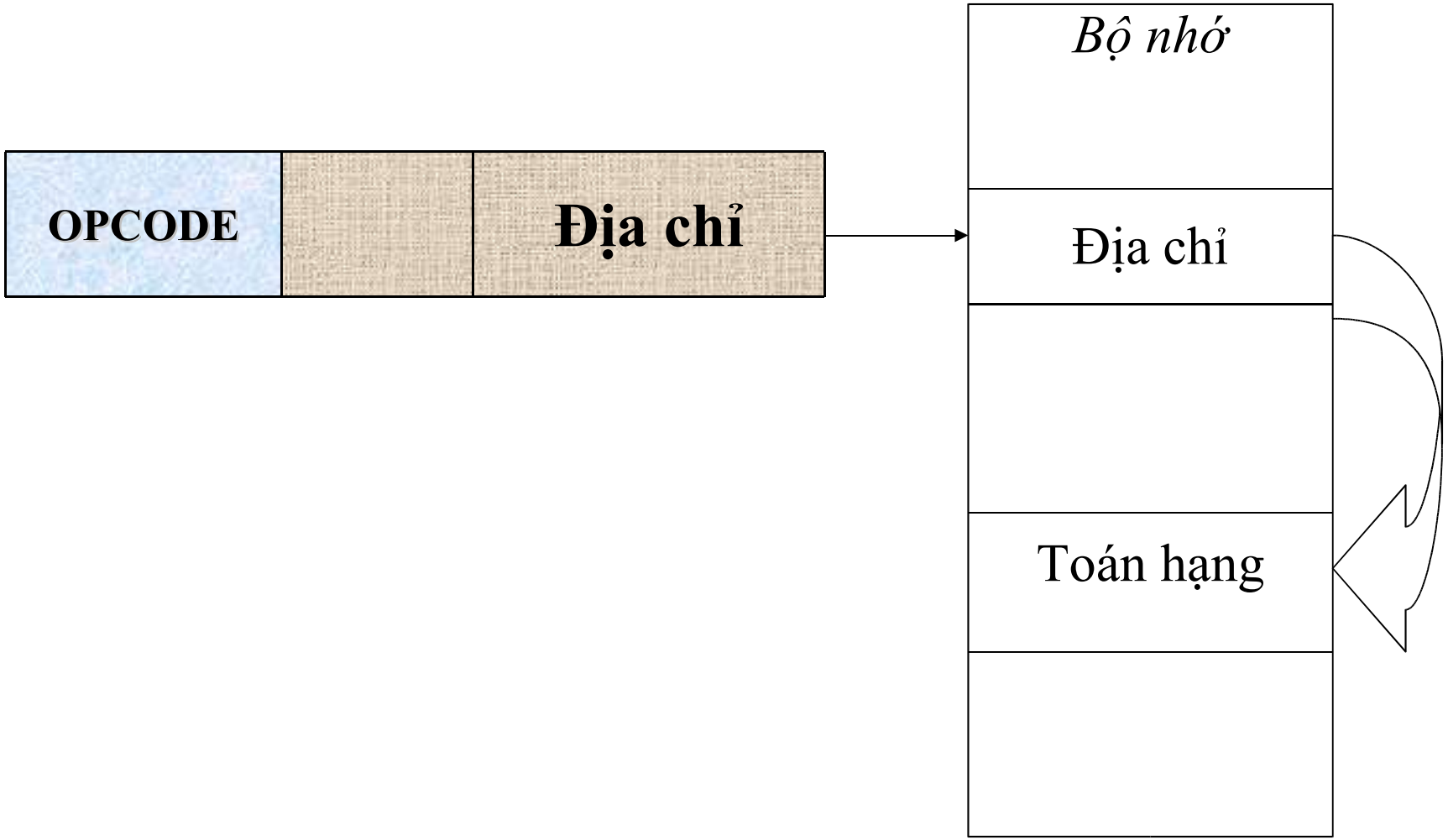


Định địa chỉ gián tiếp qua thanh ghi

- Toán hạng là ngăn ngăn nhớ có địa chỉ được chỉ ra trong thanh ghi. Trường địa chỉ toán hạng lưu trữ tên thanh ghi đó.
- Thanh ghi có thể là ngầm định
- Thanh ghi này được gọi là thanh ghi con trỏ

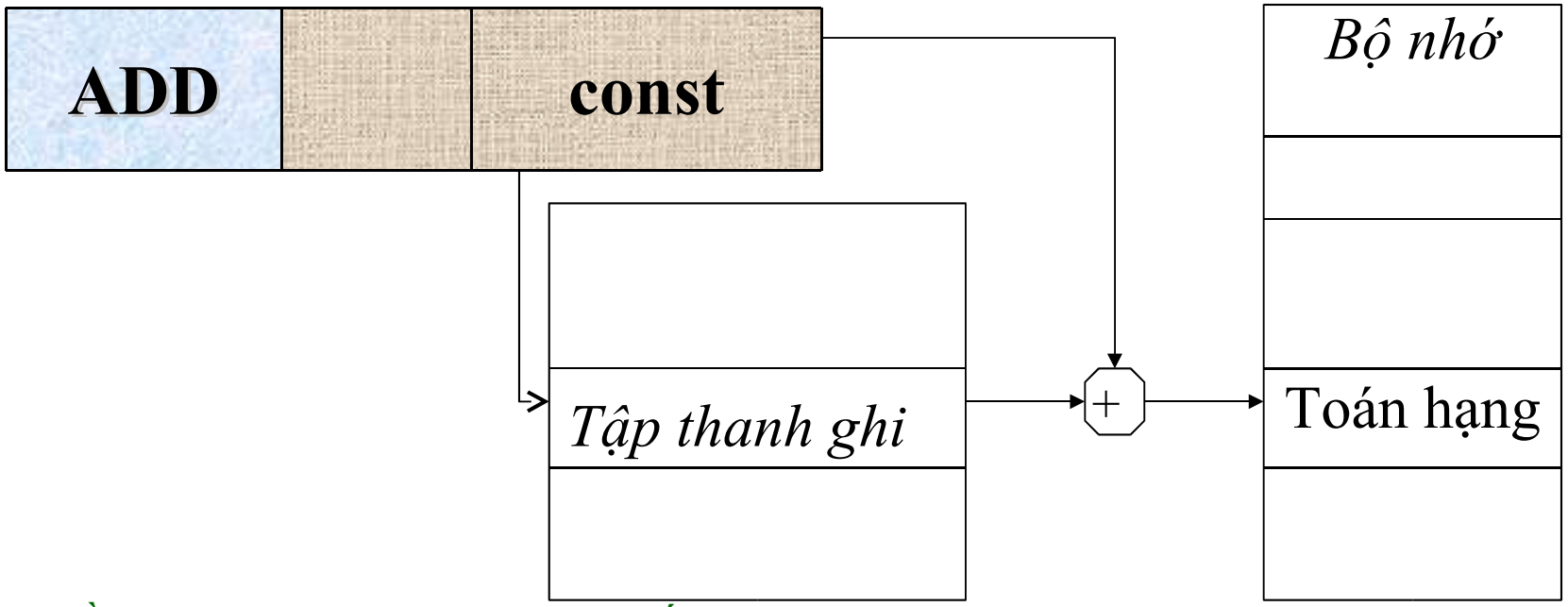


Định địa chỉ gián tiếp qua ngăn nhớ



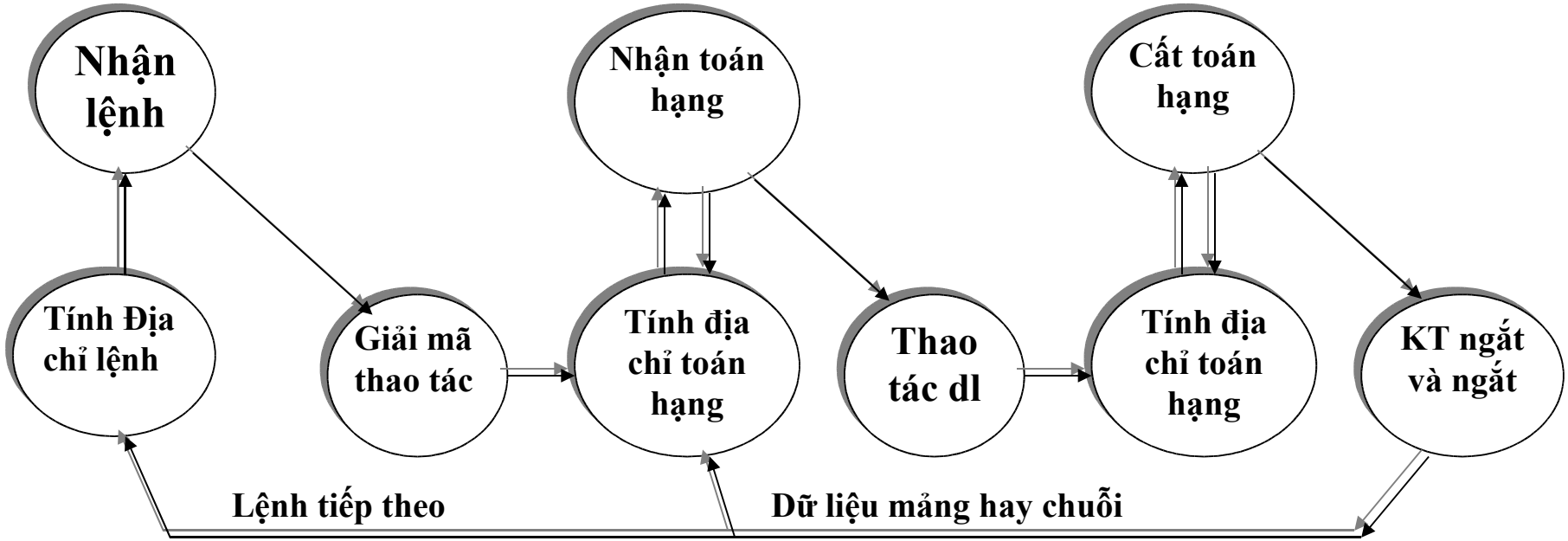
Định địa chỉ dịch chuyển

- Để xác định toán hạng gồm 2 thành phần
  - Tên thanh ghi và hằng số
  - Địa chỉ toán hạng = Nội dung thanh ghi + hằng số



4.4 Hoạt động của CPU

- Chu kỳ lệnh CPU bao gồm: Nhận lệnh, giải mã lệnh, nhận toán hạng, thực hiện lệnh, cất toán hạng, ngắt.
- Giản đồ trạng thái chu kỳ lệnh



Phần trao đổi và giải đáp

Câu hỏi ôn tập

- Cấu trúc và chức năng của CPU
- Kiến trúc tập lệnh
- Các phương pháp tham chiếu toán hạng.
- Hoạt động cơ bản CPU
- Kiến trúc BXL tiến tiến

Chương 5

Bộ nhớ máy tính

5.1 Tổng quan bộ nhớ trong Máy tính

5.2 Bộ nhớ bán dẫn

5.3 Bộ nhớ đệm nhanh (Cache)

5.4 Bộ nhớ ngoài (bộ nhớ phụ)

5.5 Hệ thống nhớ trên máy PC hiện nay

5.1 Tổng quan

Các đặc trưng của bộ nhớ

Ví trí:

- Bên trong CPU: tập thanh ghi, cache
- Bộ nhớ trong: Bộ nhớ chính và Cache
- Bộ nhớ ngoài: các thiết bị nhớ, RAID

Dung lượng:

- Độ dài từ nhớ (tính bằng bit)
- Số lượng từ nhớ

Đơn vị truyền:

- Từ nhớ
- Khối nhớ

5.1 Tổng quan

Phương pháp truy nhập:

- Truy nhập tuần tự (băng từ)
- Truy nhập trực tiếp (các loại đĩa)
- Truy nhập ngẫu nhiên (bộ nhớ bán dẫn)
- Truy nhập liên kết (cache)

Hiệu năng:

- Thời gian truy nhập
- Chu kỳ truy xuất bộ nhớ
- Tốc độ truyền

5.1 Tổng quan

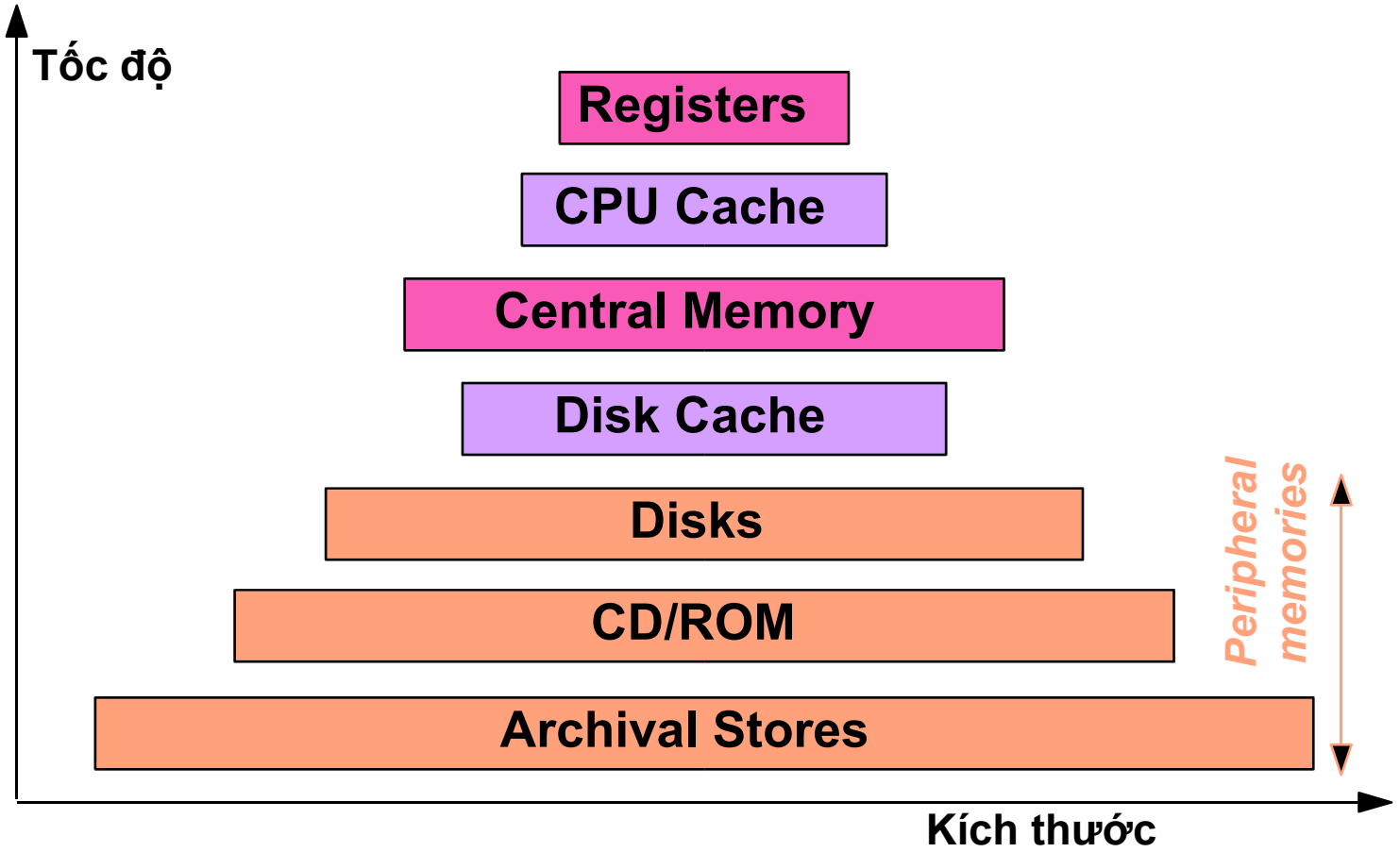
Kiểu bộ nhớ vật lý:

- Bộ nhớ bán dẫn
- Bộ nhớ từ
- Bộ nhớ quang

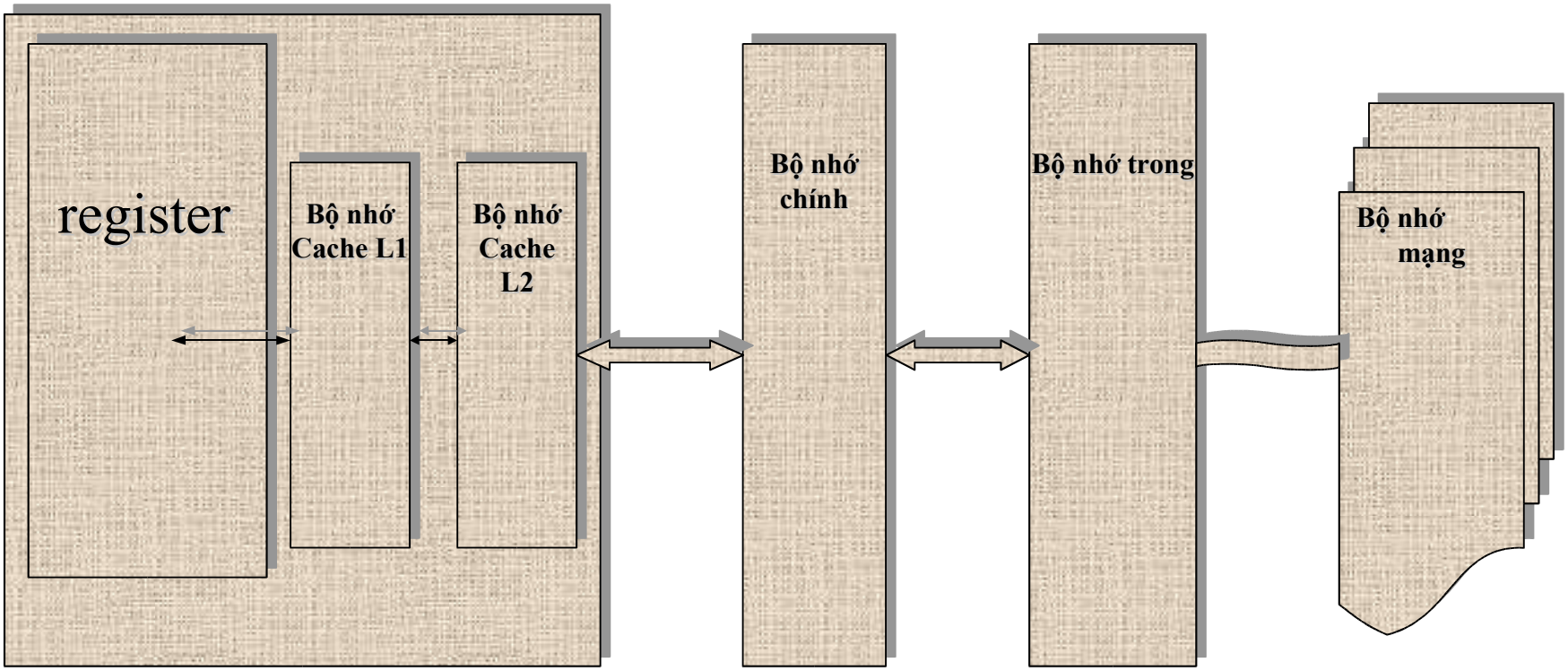
Các đặc tính vật lý:

- Khả biến/không khả biến
- Xoá được/không xoá được

Phân cấp bộ nhớ



Phân cấp bộ nhớ



Từ trái qua phải: dung lượng tăng dần, tốc độ giảm dần, giá thành tính theo đơn vị byte hoặc bit giảm dần.

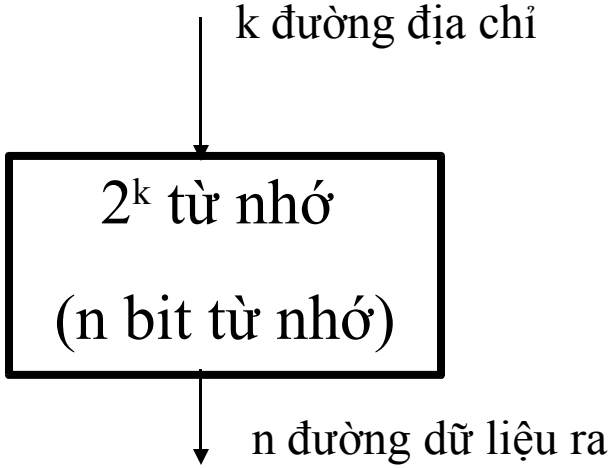
5.2 Bộ nhớ bán dẫn

Bộ nhớ chỉ đọc (ROM: Read Only Memory)

Bộ nhớ không khả biến

Sử dụng để lưu các thông tin sau:

- Thư viện các chương trình con.
- Các chương trình con điều khiển hệ thống (BIOS)
- Các bảng chức năng.



5.2 Bộ nhớ bán dẫn

Các kiểu ROM:

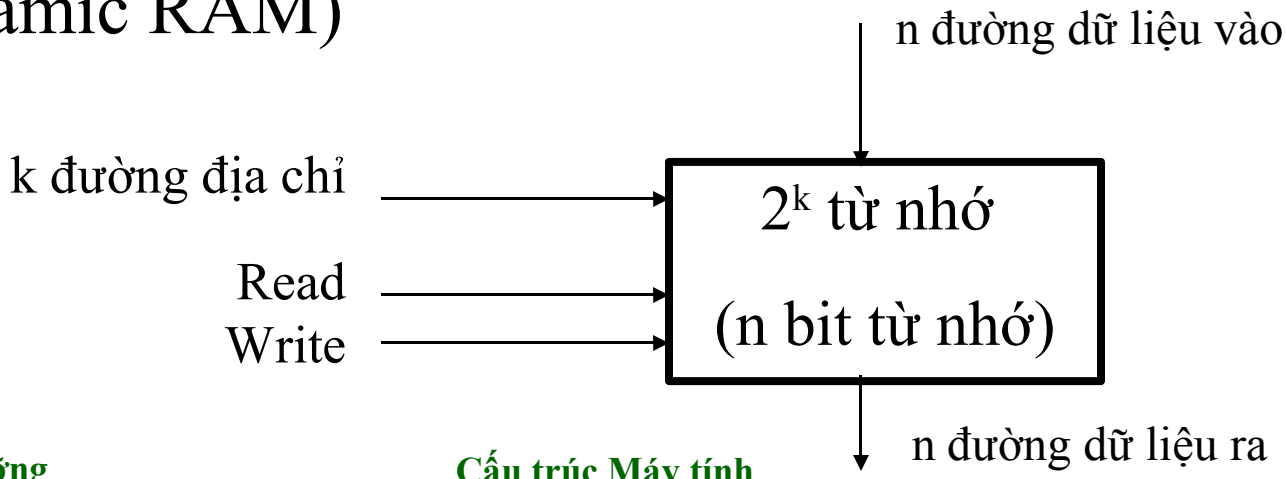
ROM mặt nạ, PROM: Programmable ROM, EPROM: Erasable PROM, EEPROM Electrically EPROM, Flash Memory (Bộ nhớ cực nhanh): Ghi theo khối, xoá bằng điện.

5.2 Bộ nhớ bán dẫn

Bộ nhớ truy cập ngẫu nhiên

(RAM : Random Access Memory)

- Bộ nhớ đọc ghi (R/W memory)
- Bộ nhớ khả biến
- Lưu thông tin tạm thời
- Có hai loại chính là SRAM (Static RAM) và DRAM (Dynamic RAM)



5.2 Bộ nhớ bán dẫn

RAM tĩnh (SRAM: Static RAM)

- Các bit được lưu dựa trên các Flip- Flop (4-8 FF lưu 1 bit)
- Thông tin lưu ổn định
- Cấu trúc phức tạm
- Dung lượng nhỏ(KB)
- Tốc độ nhanh (6-8 ns)
- Dùng làm cache
- Giá thành cao

5.2 Bộ nhớ bán dẫn

RAM động (DRAM: Dynamic RAM)

- Các bit được lưu dựa trên các tụ điện => nguyên nhân thường xuyên làm tươi.
- Dung lượng lớn.
- Tốc độ chậm (60-80ns).
- Dùng làm bộ nhớ chính
- Giá thành phải chăng.
- Các DRAM tiên tiến:

  SDRAM: Synchronous Dynamic RAM, DDRAM: Double Data RAM. Ram BUS RDRAM.

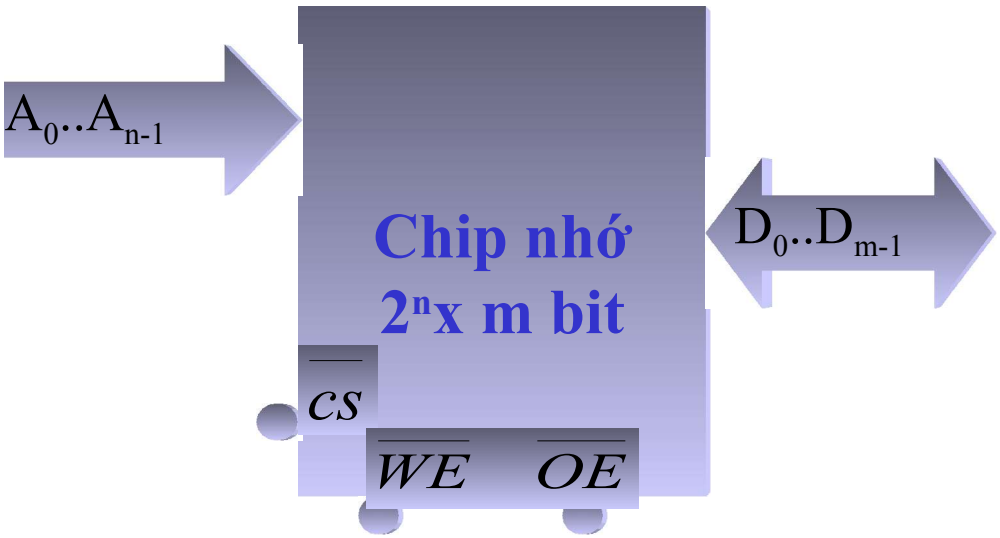
Bộ nhớ chính

Các đặc trưng cơ bản

- Tồn tại trên mọi hệ thống máy tính
- Chứa chương trình đang thực hiện và các dữ liệu có liên quan.
- Gồm các ngăn nhớ được đánh địa chỉ trực tiếp bởi CPU.
- Dung lượng bộ nhớ chính bao giờ nhỏ hơn không gian mà CPU có thể quản lý.
- Việc quản lý logic bộ nhớ phụ thuộc vào hệ điều hành.

Tổ chức của chip nhớ

- Sơ đồ cơ bản của chip nhớ



Tổ chức của chip nhớ

Các tín hiệu của chip nhớ

- ✓ Các đường địa chỉ: $A_0 \ldots A_{n-1}$ để xác định 2^n ngăn nhớ.
- ✓ Các đường dữ liệu: $D_0 \ldots D_{m-1}$ độ dài từ nhớ (m bit)

  =>dung lượng chip nhớ $= 2^n$ x m bit
- ✓ Các tín hiệu điều khiển
  - o Tín hiệu chọn chip hoạt động: CS (Chip Select)
  - o Tín hiệu điều khiển đọc hoặc ghi (WE: Write Enable; OE: Output Enable)
  - o Thường các tín hiệu điều khiển tích cực với mức 0

Thiết kế Mudule nhớ

Thiết kế module nhớ bán dẫn

- Cho chip nhớ 2^n x m bit
- Yêu cầu sử dụng chip nhớ trên thiết kế module nhớ dung lượng là bội kích thước chip nhớ trên.

Giải quyết vấn đề

Có hai cách:

- Thiết kế để tăng độ dài từ nhớ, số ngăn nhớ không thay đổi.
- Thiết kế để tăng số lượng ngăn nhớ, độ dài từ nhớ không thay đổi.

Thiết kế Mudule nhớ

Thiết kế tăng số lượng từ nhớ

- ***Giả thiết***: Cho các chip nhớ có dung lượng 2^n x m bit.
- ***Yêu cầu*:** Thiết kế module nhớ có kích thước:

  2^n x $(k.m)$ bit
- ***Giải quyết***:

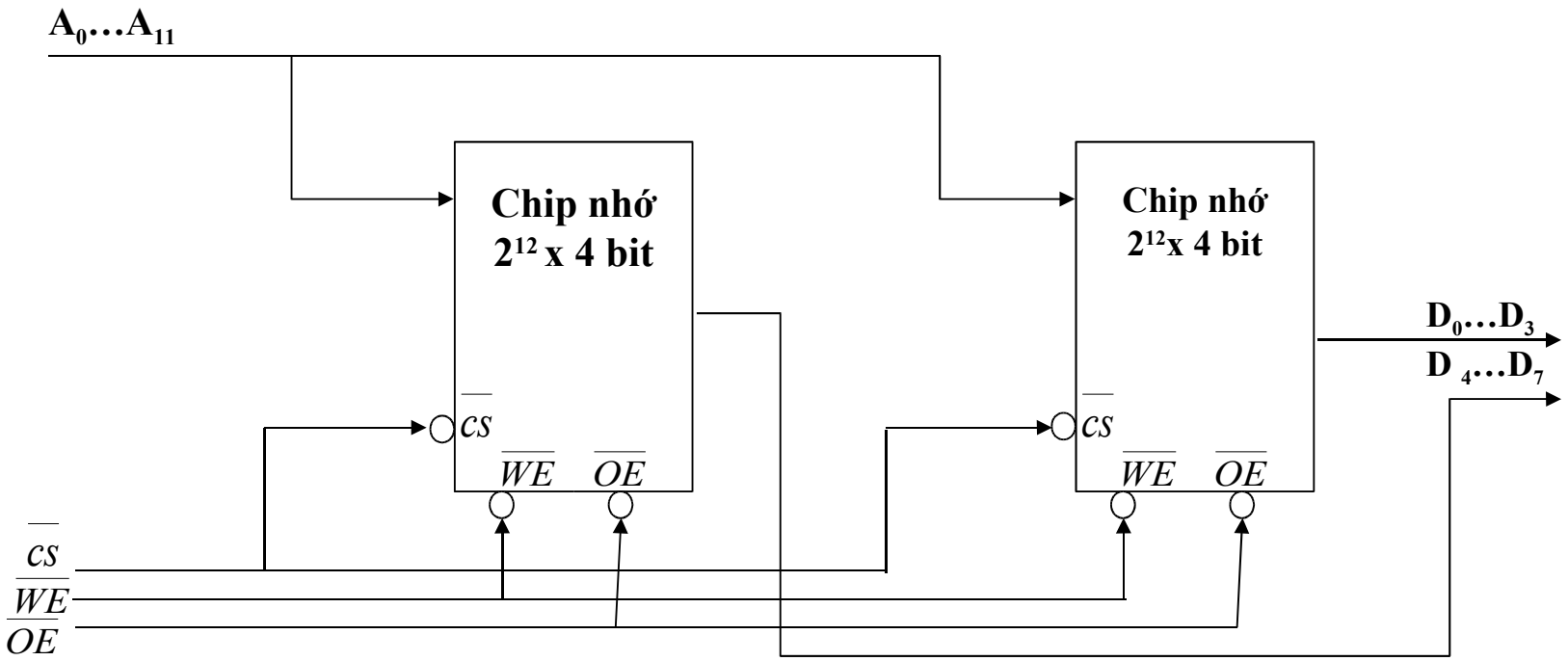
  Để thiết kế được yêu cầu ta xác định hai thông số n (số đường địa chỉ)và k(số chip nhớ cần để ghép vào module thiết kế

Thiết kế Mudule nhớ

Ví dụ: *Cho các chip nhớ SDRAM dung lượng 4K x 4 bit. Hãy thiết kế module nhớ có kích thước 4K x 8 bit*

- Dung lượng chip nhớ 2^{12} x 4 bit
- Thông tin cần cho chip nhớ số đường địa chỉ n =12 và số đường dữ liệu m=4
- Thông tin về module nhớ số đường địa chỉ là 12 đường (số ngăn nhớ không thay đổi), số đường dữ liệu là 8 đường và số chip sử dụng thiết kế 2(k=2)

Thiết kế Mudule nhớ



Thiết kế Mudule nhớ

Thiết kế tăng số lượng ngăn nhớ

- ***Giả thiết***: Cho các chip nhớ có dung lượng 2^n x m bit.
- ***Yêu cầu*:** Thiết kế module nhớ có kích thước:

  $2^k.2^n$ x m bit
- ***Giải quyết***:

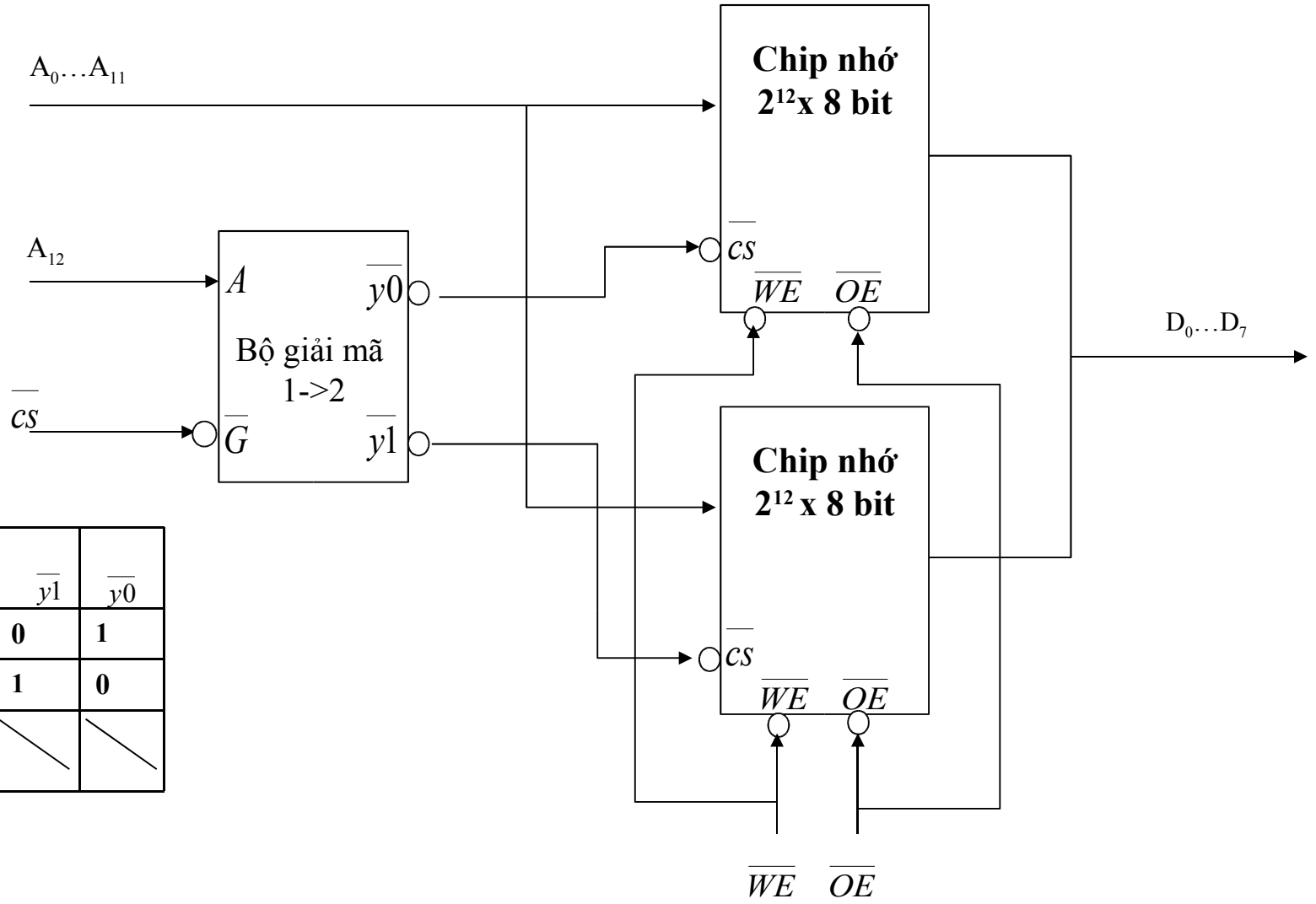
  Để thiết kế được ta xác định hai thông số n+k (số đường địa chỉ) và 2^k (số chip nhớ cần để ghép vào module thiết kế)

Thiết kế Mudule nhớ

Ví dụ : *Cho các chip nhớ SDRAM dung lượng 4K* x *8 bit. Hãy thiết kế module nhớ có kích thước 8K* x *8 bit.*

- Dung lượng chip nhớ giải thiết 2^{12} x 8 bit
- Thông tin cần cho chip nhớ số đường địa chỉ n =12 và số đường dữ liệu m=8
- Thông tin về module nhớ số đường địa chỉ là 13 đường (số ngăn nhớ thay đổi) và số đường dữ liệu là 8 đường(độ dài từ nhớ không đổi).

Thiết kế Mudule nhớ



$\overline{G}$	A	$\overline{y1}$	$\overline{y0}$
0	0	0	1
0	1	1	0
1	x		

Bài làm thêm

- Thiết kế module nhớ 16K x 8 bit từ các chip nhớ 4K x 8 bit
- Thiết kế module nhớ 32K x 8 bit từ các chip nhớ 4K x 8 bit
- Thiết kế module nhớ 8K x 8 bit từ các chip nhớ 4K x 4 bit
- Thiết kế module nhớ 32M x 32 bit từ các chip nhớ 4M x 32 bit

Phát hiện và chỉnh lỗi trong bộ nhớ

Phát hiện và chỉnh lỗi trong bộ nhớ

Nguyên tắc chung: Trong quá trình truyền dữ liệu có thể gặp sự thay đổi các bit thông tin do nhiễu hoặc do sai hỏng của thiết bị hay module vào ra. Vì vậy, thực tế đặt ra là phải làm sao phát hiện được lỗi và có thể sửa sai được. Một trong phương pháp phát hiện lỗi (EDC: Error Dectecting Code) và sửa lỗi (ECC: Error Correcting Code) là: Giả sử cần kiểm tra m bit thì người ta ghép thêm k bit kiểm tra được mã hoá theo cách nào đó rồi truyền từ ghép $m+k$ bit (k bit được truyền không mang thông tin nên gọi là bit dư thừa)

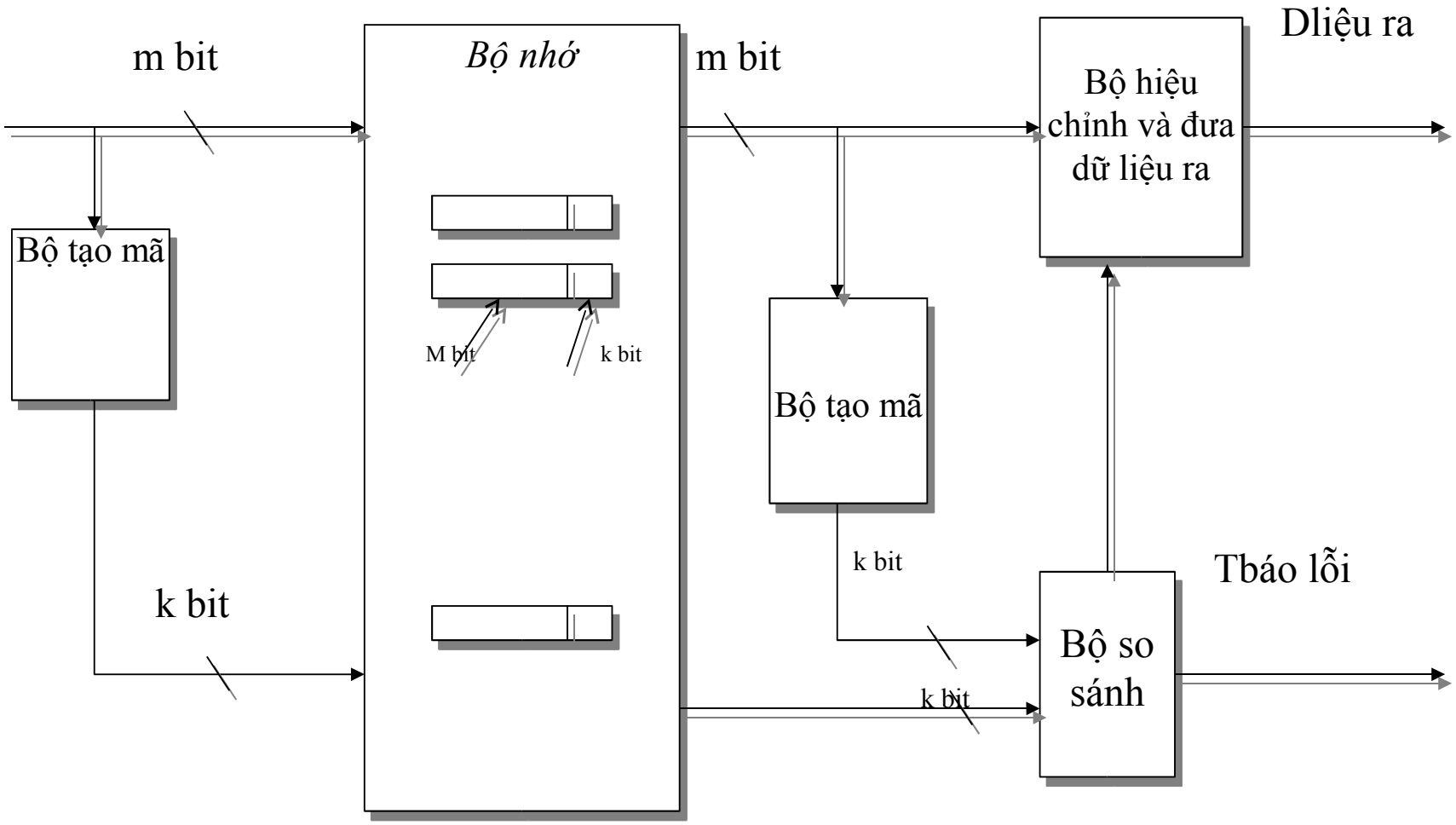
Trong đó m là số bit cần ghi vào bộ nhớ và k bit là số bit cần tạo ra kiểm tra lỗi trong m bit.

Phát hiện và chỉnh lỗi trong bộ nhớ

Khi đọc dữ liệu ra có khả năng sau:

- Không phát hiện dữ liệu có lỗi.
- Phát hiện thấy dữ liệu lỗi và có thể hiệu chỉnh dữ liệu lỗi thành đúng.
- Phát hiện thấy lỗi nhưng không có khả năng chỉ ra lỗi vì thế phát ra tín hiệu báo lỗi.
- Sơ đồ phát hiện lỗi và sửa lỗi

Phát hiện và chỉnh lỗi trong bộ nhớ



Phát hiện và chỉnh lỗi trong bộ nhớ

Ví dụ 1: Phát hiện lỗi với bit chẵn lẻ(Party)

Mã EDC đơn giản là bit chẵn lẻ được gắn thêm vào các bit dữ liệu.

Nếu bit chẵn lẻ =1: nếu số bit 1 trong xâu là lẻ

Hoặc sử dụng Nếu bit chẵn lẻ =0: nếu số bit 1 là chẵn

Ưu điểm: đơn giản và số bit dư thừa ít.

Nhược điểm: không định vị được lỗi, hoặc nếu có sự thay đổi cả hai bit hoặc 1 hoặc 0 thì không phát hiện được. Khắc phục nhược điểm trên xây dựng mã EDC khối.

Phát hiện và chỉnh lỗi trong bộ nhớ

Ví dụ 2: Phát hiện lỗi bằng mã dư thừa CRC (Cycle Redundary Check).

Nguyên tắc: Một xâu nhị phân bất kỳ có thể coi là tập hợp các hệ số của đa thức B(x) trong đó x là *hư số*. Chọn đa thức G(x) là đa nào đó ta quy định trước gọi *đa thức sinh*. Ta tiến hành chia module2 đa thức B(x) cho G(x) ta được thương số Q(x) và phần dư R(x).

⇨ Đa thức sinh do tổ chức viễn thông quốc tế quy định.

⇨ Khi đó ta cần truyền xâu B(x) + R(x) bit

⇨ Để kiểm tra lỗi ta cần chia giá trị nhận được cho đa thức sinh nếu phép chia có dư thì có lỗi xuất hiện trong xâu.

Phát hiện và chỉnh lỗi trong bộ nhớ

- Ví dụ:
- Xâu gốc: 1101011011 ⇔ $M(x)=x^9+x^8+x^6+x^4+x^3+x+1 (m=9)$
- Đa thức sinh $G(x) = x^4+x+1$ ⇔ 10011 (r=4)
- Xâu gốc: 11010110110000 ⇔ $x^4M(x)$
- Chia mod2 11010110110000 | 10011

  1100001010 -> thương

  1110 phần dư phép chia
- Xâu cần truyền đi: 11010110111110 ⇔ T(x)

Phát hiện và chỉnh lỗi trong bộ nhớ

Ví dụ 3: Mã sửa lỗi Hamming

Nguyên tắc: Một từ mã Hamming gồm m bit dữ liệu và k bit kiểm tra chẵn lẻ. Mỗi bit được chọn vị trí thích hợp để phát hiện chính xác ví trí để có thể sửa lỗi được. Ví dụ chọn m=4 => k=3 ($m=2^n$; $k=n+1$)

Ta có thứ tự sau:

7	6	5	4	3	2	1
I4	I3	I2	C3	I1	C2	C1

Các bít này được mã hoá theo quy luật sau:

C1=I1⊕ I2⊕ I4

C2=I1⊕ I3⊕ I4

C3=I2⊕ I3⊕ I4

Phát hiện và chỉnh lỗi trong bộ nhớ

- Giả sử các bit cần truyền là: I4 I3 I2 I1 = 1101 tính các C3C2C1=010
- Bit cần truyền 1100110
- Giả sử ta có bị lỗi, thí dụ bit I2 từ giá trị 0 thành giá trị 1 mã nhận được 1110110.
- Bên thu tính ra bit kiểm tra:

  C3=1⊕ 1⊕ 1=1

  C2=1⊕ 1⊕ 1=1

  C1=1⊕ 1⊕ 1=1

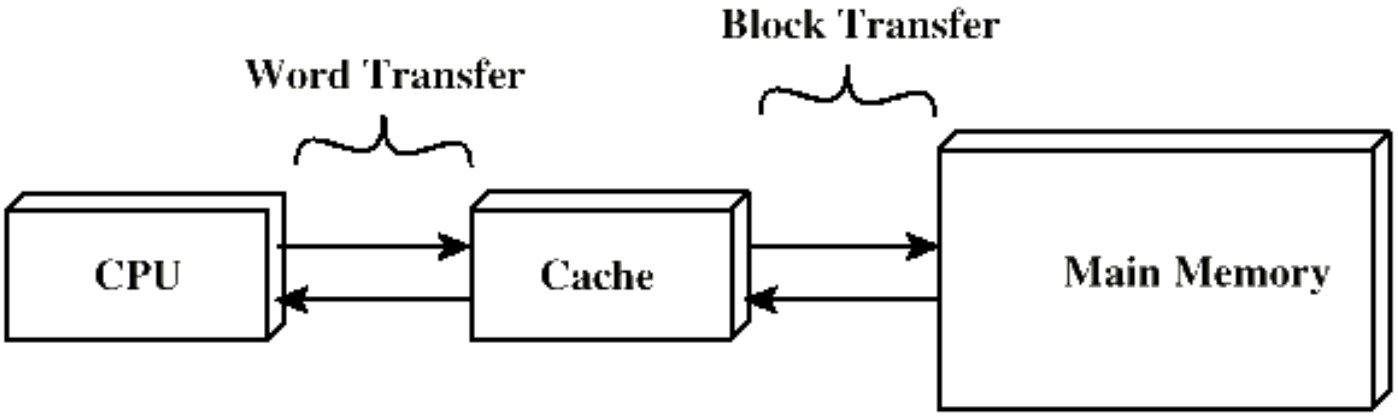
- Nếu module 2 số này ta được 111 ⊕ 010 = 101

(C1,C3 thay đổi và vị trí thay đổi là 101 (5))

5.3 Bộ nhớ đệm nhanh

Nguyên tắc:

- Cache có tốc độ truy xuất nhanh hơn rất nhiều bộ nhớ chính
- Cache được đặt giữa CPU và bộ nhớ chính nhằm tăng tốc độ trao đổi thông tin giữa CPU và bộ nhớ chính.
- Cache thường được đặt trong chip vi xử lý

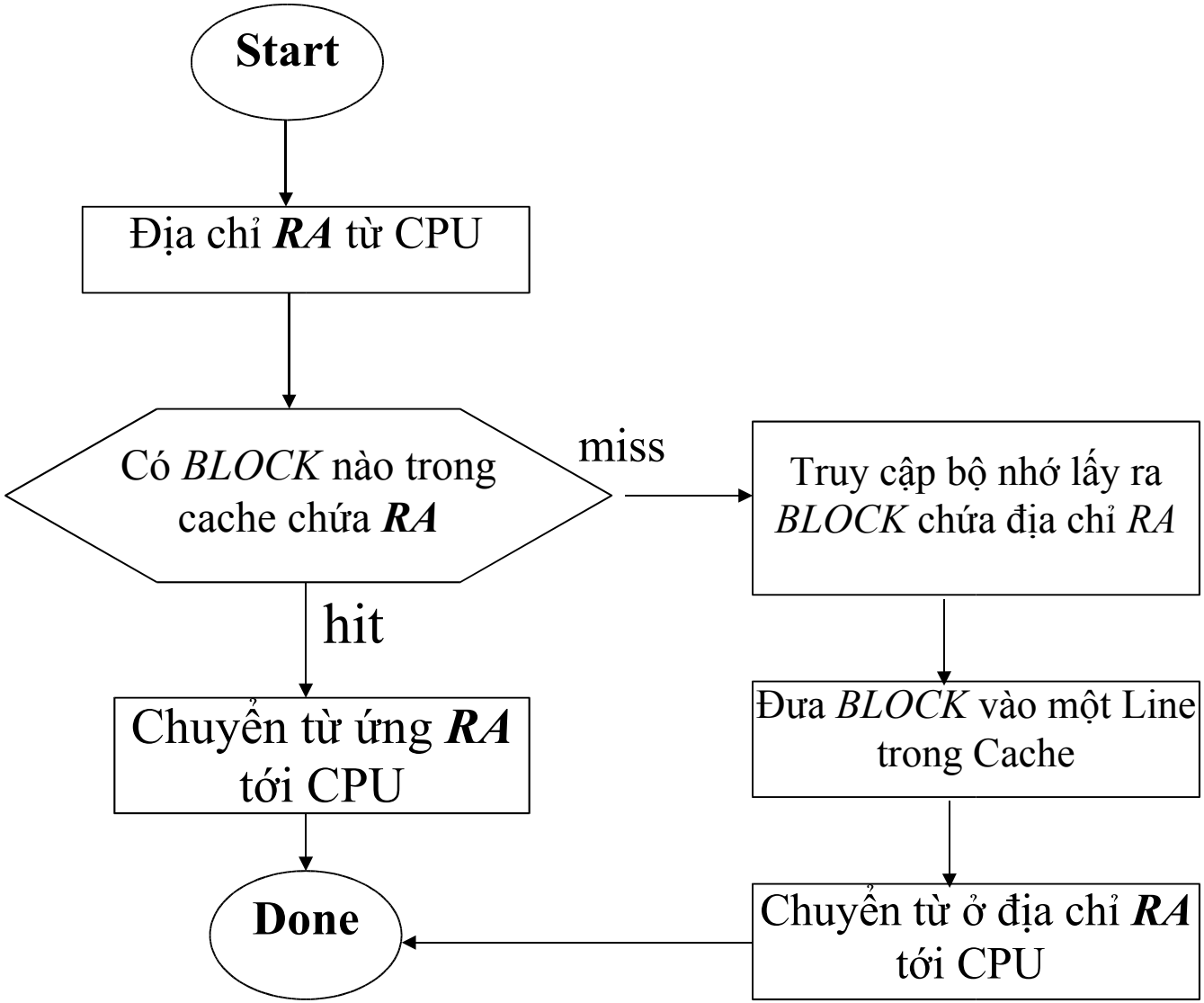


5.4 Bộ nhớ đệm nhanh

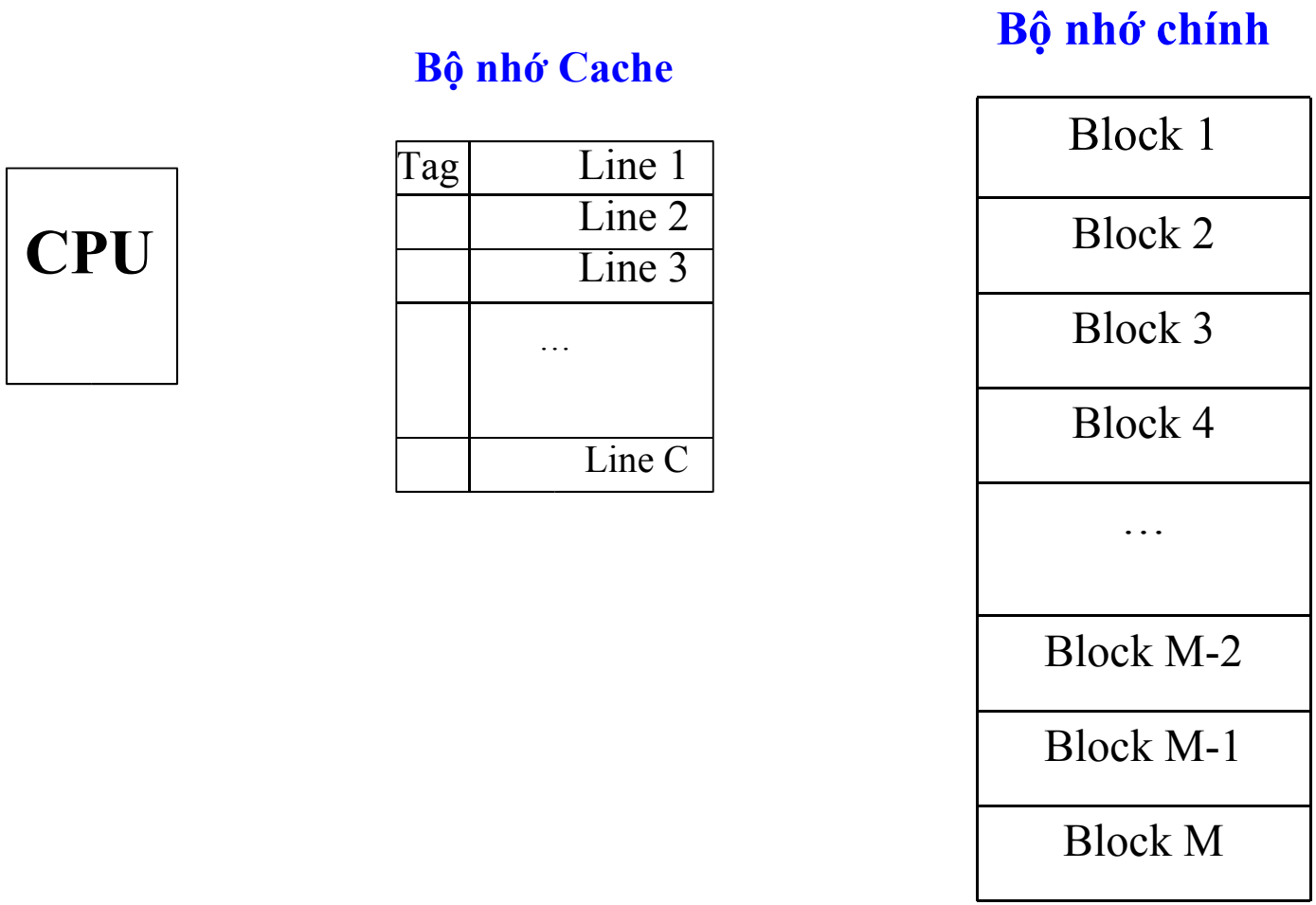
Thao tác của Cache

- CPU yêu cầu lấy nội dung của một ngăn nhớ bằng việc đưa ra một địa chỉ xác định ô nhớ.
- CPU kiểm tra xem có nội dung cần tìm trong Cache
- Nếu có: CPU nhận dữ liệu từ bộ nhớ Cache
- Nếu không có: Bộ điều khiển Cache đọc Block nhớ chứa dữ liệu CPU cần vào Cache.
- Tiếp đó chuyển dữ liệu từ Cache đến CPU
- Sơ đồ thao tác cache, bộ nhớ chính và CPU

5.4 Bộ nhớ đệm nhanh



5.4 Bộ nhớ đệm nhanh



5.3 Bộ nhớ đệm nhanh

- Tổ chức Cache

- Giả sử bộ nhớ chính gồm có 2^n từ nhớ đã được đánh địa chỉ (mỗi từ nhớ có địa chỉ duy nhất rộng n bit)
- Bộ nhớ chính chia thành M khối, mỗi khối có K từ nhớ $M=2^n/K$
- Bộ nhớ Cache có C khe mỗi khe có K từ nhớ.(C<<M)
- Tại một thời điểm luôn có một tập con các khối nhớ thường trú trong cache.
- Nếu một từ sẽ được đọc thì khối chứa từ đó sẽ được chuyển vào trong cache.

5.3 Bộ nhớ đệm nhanh

Ví dụ cho phương pháp ánh xạ cụ thể trong cache

- Cho dung lượng Cache là 64KB (m=16)
  Mỗi khối kính thước 4 bytes
  => C=16K(2^{14}) lines mỗi line kích thước 4 bytes
- Cho dung lượng bộ nhớ chính 16MB (n=24)
  Mỗi khối kính thước 4 bytes
  => M=4M(2^{22}) khối mỗi khối kích thước 4 bytes

5.3 Bộ nhớ đệm nhanh

Phương pháp ánh xạ trực tiếp (Direct mapping)

- Mỗi block được ánh xạ duy nhất tới 1 line trong cache
- Địa chỉ phát ra từ CPU được chia 2 phần
- w bits có trọng số thấp để xác định duy nhất từ cần truy xuất(WORD)
- s bits còn lại xác định khối nhớ. Trong s bits chia 2 nhóm r bits LINE và s-r bits TAG

Cụ thể hóa ví dụ:

Tag s-r	Line or Slot r	Word w
8	**14**	**2**

5.3 Bộ nhớ đệm nhanh

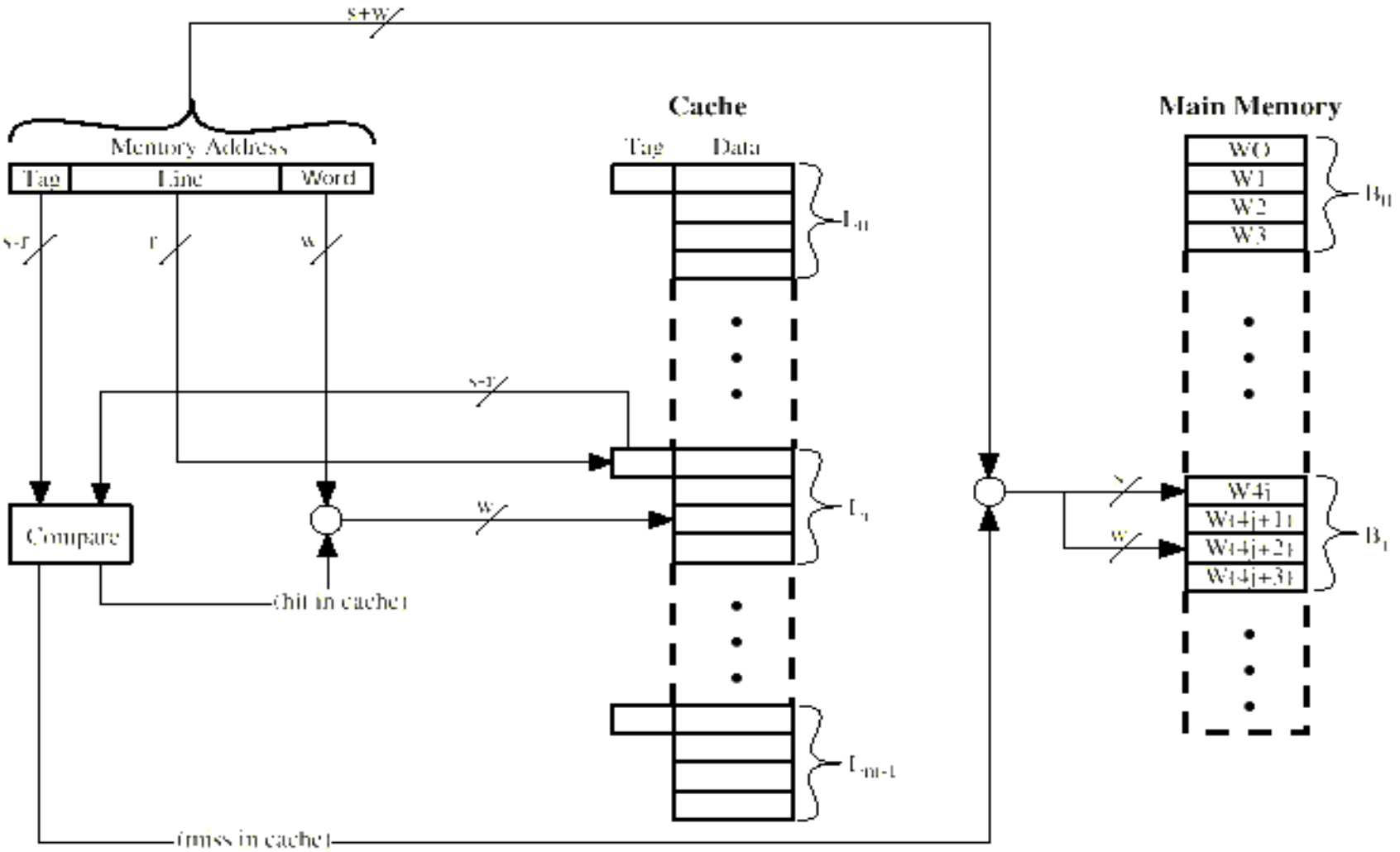
Tổng bit trong địa chỉ bộ nhớ chính n=24 bit: trong đó

2 bit phần word xác định chính xác 4 từ

22 bit xác định khối(8 bit tag (=22-14) và 14 bit slot or line)

- Không có hai block nào trong Cache có cùng Line và Tag.
- Kiểm tra nội dung từ tồn tại Cache chính là kiểm tra địa chỉ line và Tag

5.3 Bộ nhớ đệm nhanh



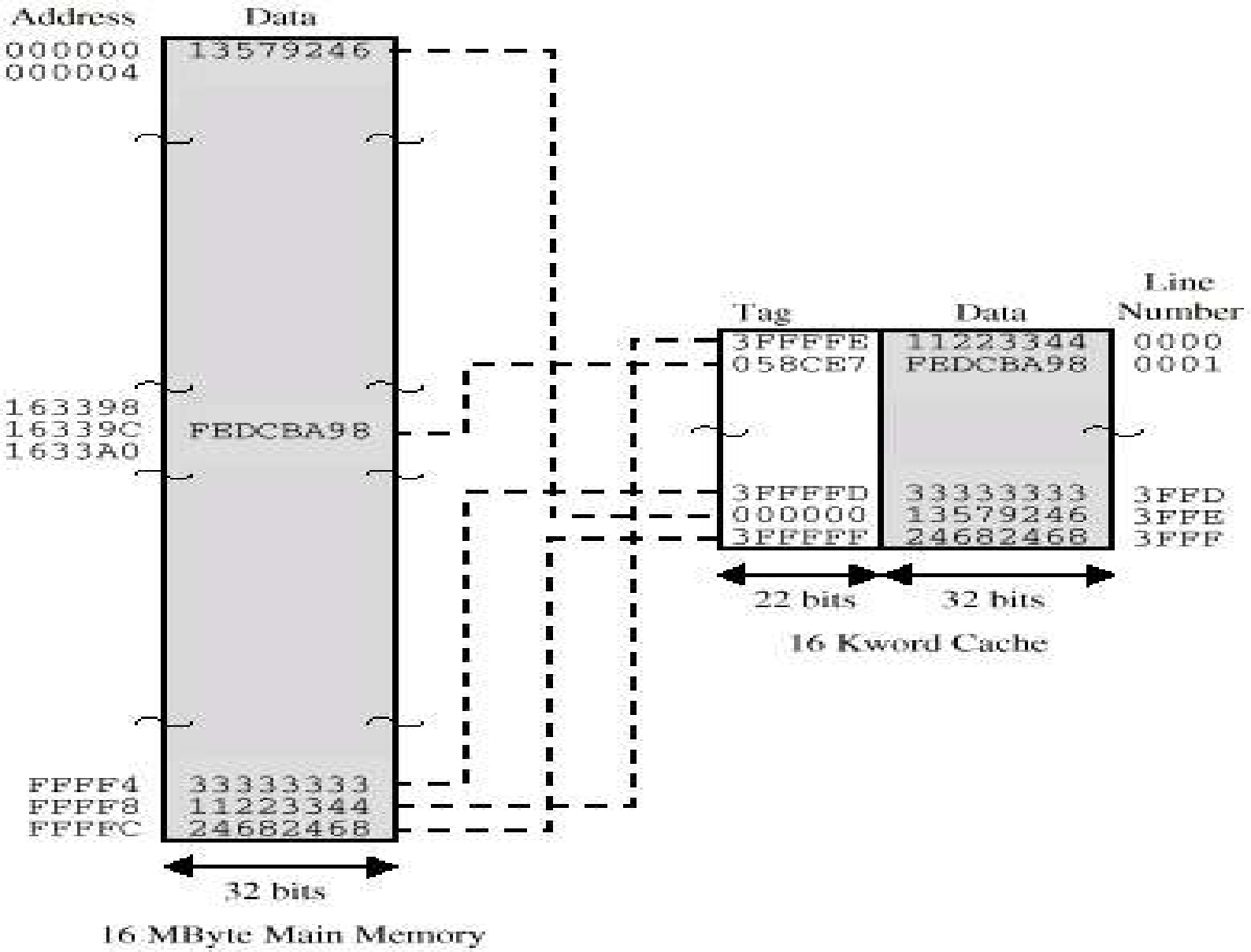
5.3 Bộ nhớ đệm nhanh

Cache line	Main Memory blocks
0	0, C, 2C, 3C…2^s-C
1	1,C+1, 2C+1…2^s-C+1
C-1	C-1, 2C-1,3C-1…2^s-1

Nhận xét:

- Đơn giản
- Chi phí ít
- Nhược điểm là sự cố định các khối trong các line của Cache. Trong trường hợp chương trình muốn truy xuất tới 2 Block tiên tục mà 2 block được phân nằm trong cùng line thì khả năng Cache miss rất cao.

5.3 Bộ nhớ đệm nhanh



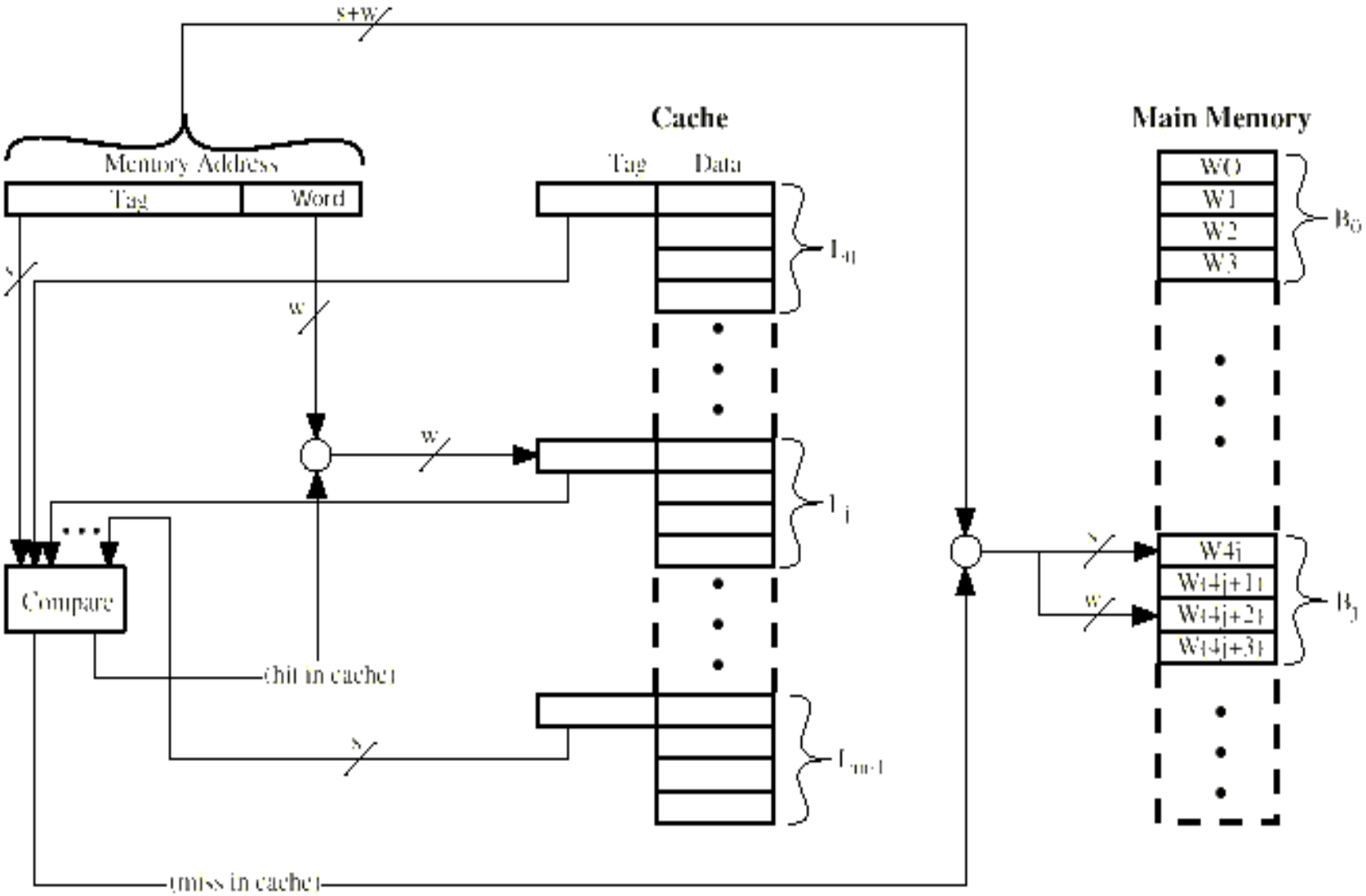
5.3 Bộ nhớ đệm nhanh

Phương pháp ánh xạ liên kết

(Associative mapping)

- Một Block của bộ nhớ chính có thể nhập bất kỳ line nào trong Cache.
- Địa chỉ CPU phát ra được chia thành 2 địa chỉ tag và word
- Địa chỉ Tag xác định khối duy nhất của bộ nhớ nằm trong Cache.
- Mỗi giá trị Tag của Line là khác nhau.
- Chi phí phương pháp này đối với Cache là cao.

5.3 Bộ nhớ đệm nhanh



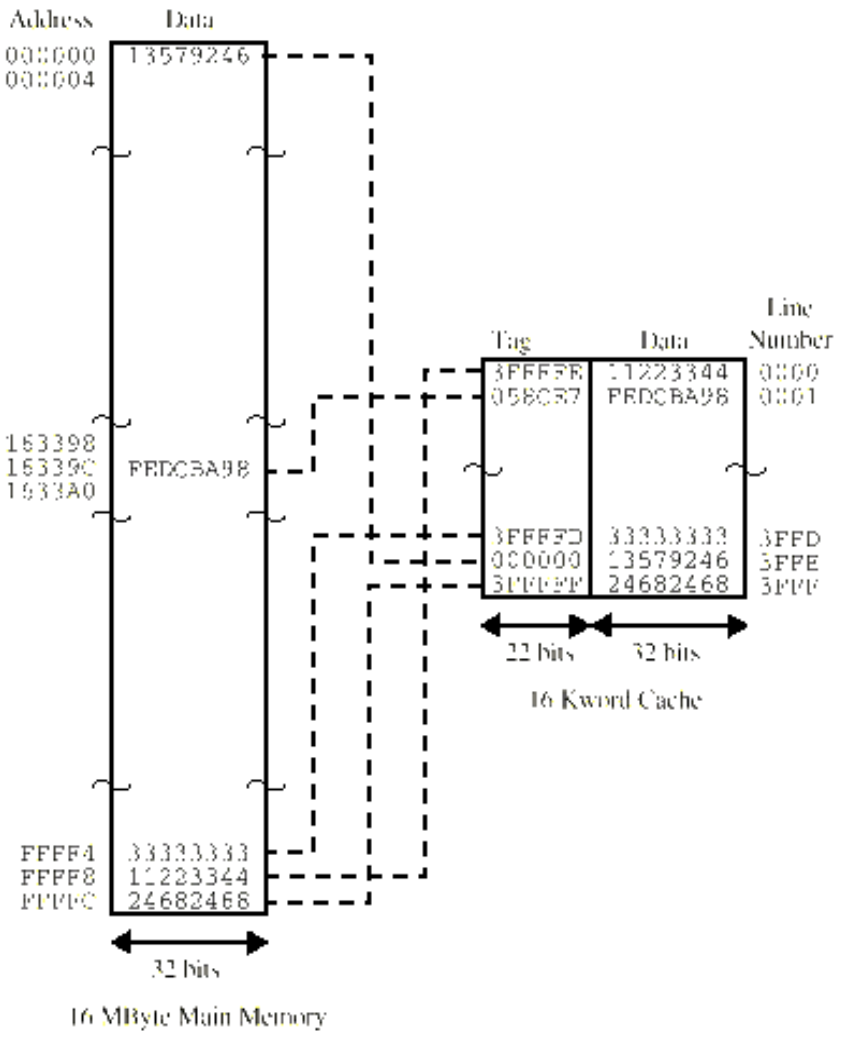
5.3 Bộ nhớ đệm nhanh

Tag 22 bit	Word 2 bit

- 22 bit Tag để lưu trữ Block 4 byte dữ liệu.
- Việc kiểm tra Cache dựa vào các giá trị Tag trong line (22 bit) để nhận biết Cache hit hay miss.
- 2 bits cuối xác định chính xác từ cần truy xuất
- Ví dụ

Địa chỉ	Tag	Dữ liệu	Cache line
FFFFFC	FFFFFC	24682468	3FFF

5.3 Bộ nhớ đệm nhanh



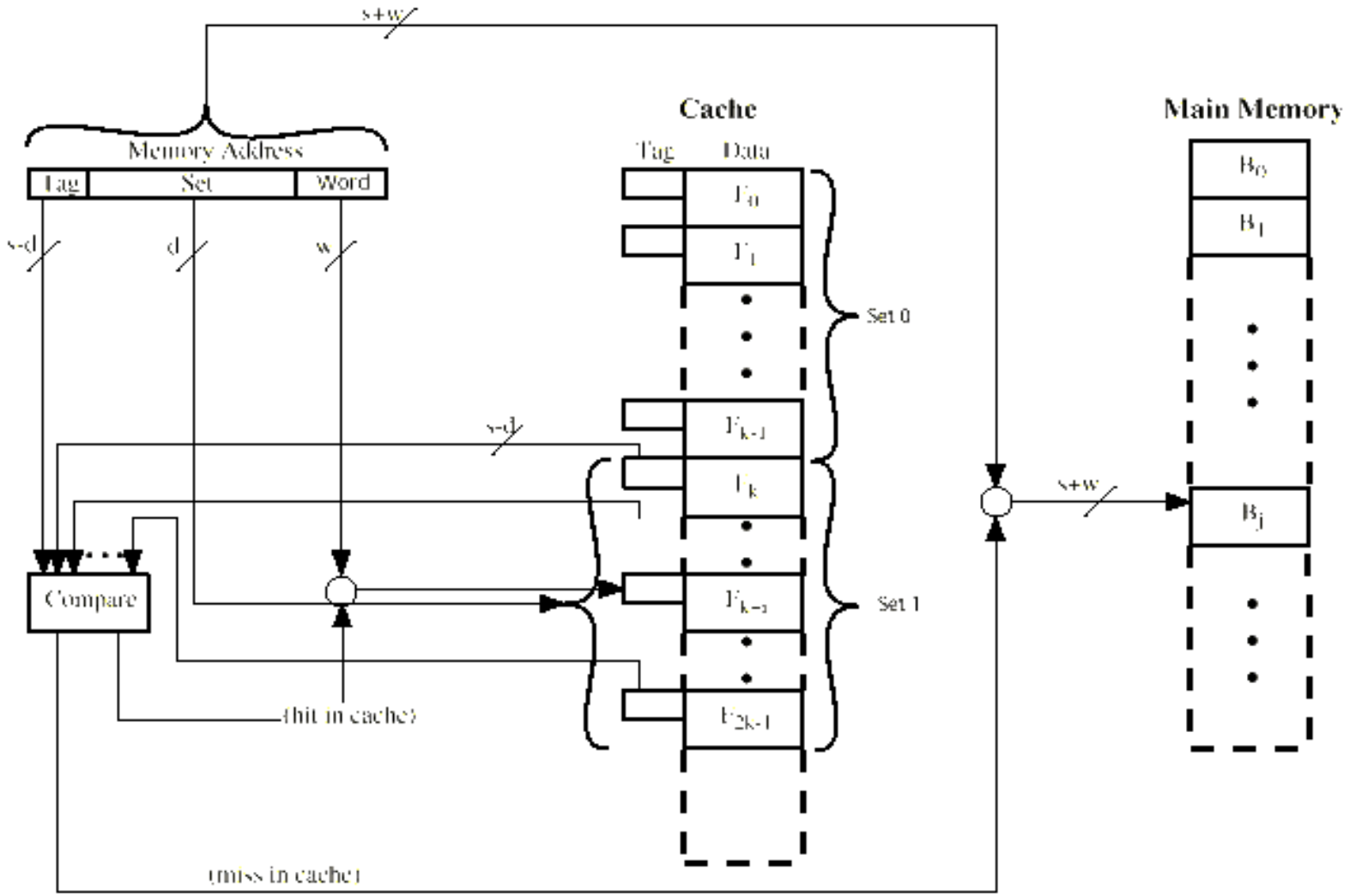
5.3 Bộ nhớ đệm nhanh

Phương pháp ánh xạ liên kết tập hợp

(Set Associative mapping)

- Các line trong Cache được chia ra thành tập(nhóm) line
- Mỗi block chỉ được ánh xạ vào bất kỳ line nào trong tập nào đó mà thôi. Ví dụ Block b chỉ có thể nập vào bất kỳ line nào trong nhóm các line thứ i. Ví dụ 2 lines một nhóm (two way associative mapping), Số Block bộ nhớ chính là modulo 2^{13}
- 000000, 00A000, 00B000, 00C000 … ánh xạ cùng nhóm.

5.3 Bộ nhớ đệm nhanh



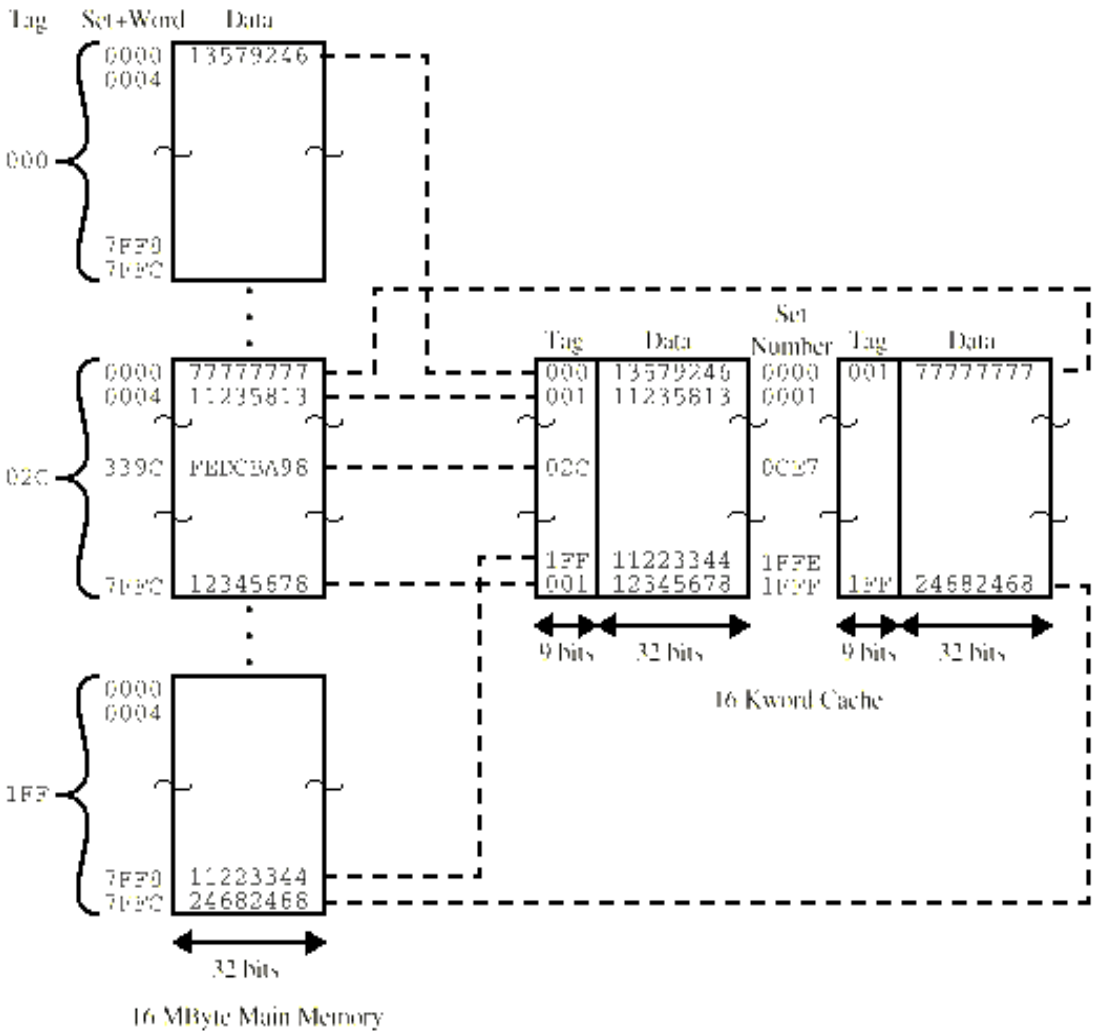
5.3 Bộ nhớ đệm nhanh

Tag 9 bit	Set 13 bit	Word 2 bit

- Sử dụng tập hợp để biết tập nào được truy xuất.
- So sánh trường Tag để xác đinh Cache hit hay miss
- Ví dụ:

Địa chỉ	Tag	Dữ liệu	số tập
1FF 7FFC	1FF	12345678	1FFF
001 7FFC	001	11223344	1FFF

5.3 Bộ nhớ đệm nhanh



5.4 Bộ nhớ đệm nhanh

Một số Block của bộ nhớ chính được nạp vào trong các line của Cache

- Nội dung thẻ *TAG* (thẻ nhớ) cho biết block nào của bộ nhớ chính hiện đang được chứa trong line
- Khi CPU truy nhập đọc hay ghi một từ nhớ của bộ nhớ chính, có 2 khả năng xảy ra :
- Từ nhớ đó có trong Cache (cache hit). Từ nhớ đó đang không có trong cache (Cache miss).

Phương pháp ghi dữ liệu khi cache hit

- Ghi xuyên qua (Write Through): nội dung sau khi xử lý xong được cập nhập vào cả Cache và bộ nhớ chính. Tốc độ chậm.

Cache trong các bộ xử lý Intel

- Ghi sau (Write back): Dữ liệu xử lý chỉ được ghi ra Cache, tốc độ nhanh. Tuy nhiên khi Block trong cache không dùng nữa thì phải ghi trả cả block tới bộ nhớ chính.

Dung lượng Cache được sử dụng cho thế hệ máy:

- 80486: có 3KB nhớ
- Pentium : có 2 cache L1 trên chip đó là Cache lệnh và cache dữ liệu (8KB). Cache L2 liên hợp
- Pentium 4: hai mức Cache L1 và L2 trên chip. Cache L1 mỗi cache 8KB. Cache L2: mỗi cache 256KB, 512KB, 1GB

5.5 Bộ nhớ ngoài

- Các kiểu bộ nhớ ngoài
- Đĩa từ
- Đĩa quang
- Bộ nhớ Flash
- RAID

Đĩa cứng (HDD: Hard Disk Driver)

- Là thành phần quan trọng lưu trữ hệ điều hành và các phần mềm tiện ích máy tính
- Một máy tính có thể một đĩa hoặc nhiều đĩa
- Dung lượng mỗi đĩa rất lớn. Năm 1993 đĩa lớn nhất 200MB đến nay 80 hay 120GB
- Tốc độ đọc ghi nhanh so các bộ nhớ ngoài khác
- Giá thành hạ
- Được sử dụng làm bộ nhớ RAID (Redundant Array of (Inexpensive) Independent Disks). Hệ thống nhớ gồm nhiều ổ đĩa cứng kết hợp với nhau mà HĐH coi như một ổ logic duy nhất.
- Dữ liệu được lưu trữ phân tán trên tất cả các đĩa
- Có thể tạo và lưu trữ thông tin dư thừa nhằm mục đích cho việc phục hồi khi đĩa nào đó bị hỏng. Độ tin cậy trong lưu trữ thông tin rất cao. Được sử dụng là bộ nhớ cho các hệ thống máy chủ.

Đĩa quang (CD-ROM, DVD)

- CD-ROM (Compact Disk ROM)
- CD-R (Recordable CD)
- CD-RW (Rewriteable CD)
- Dung lượng phổ biến 650MB
- Ổ đĩa CD
- ổ CD ROM: có thể đọc dữ liệu từ đĩa CD
- ổ CD RW : Có thể vừa ổ đọc đĩa CD và có thể ghi dữ liệu lên đĩa CD-R, và CD-RW.
- Tốc độ đọc cơ sở 150KB/s
- Tốc độ bội lần : 40x, 50x, 60x,…
- DVD(Digital Video Disk): chỉ dùng trên đầu đọc
- DVD (Digital Versatile Disk): dùng trên ổ đĩa máy tính
- Dung lượng thông dụng 4.7GB

Flash disk

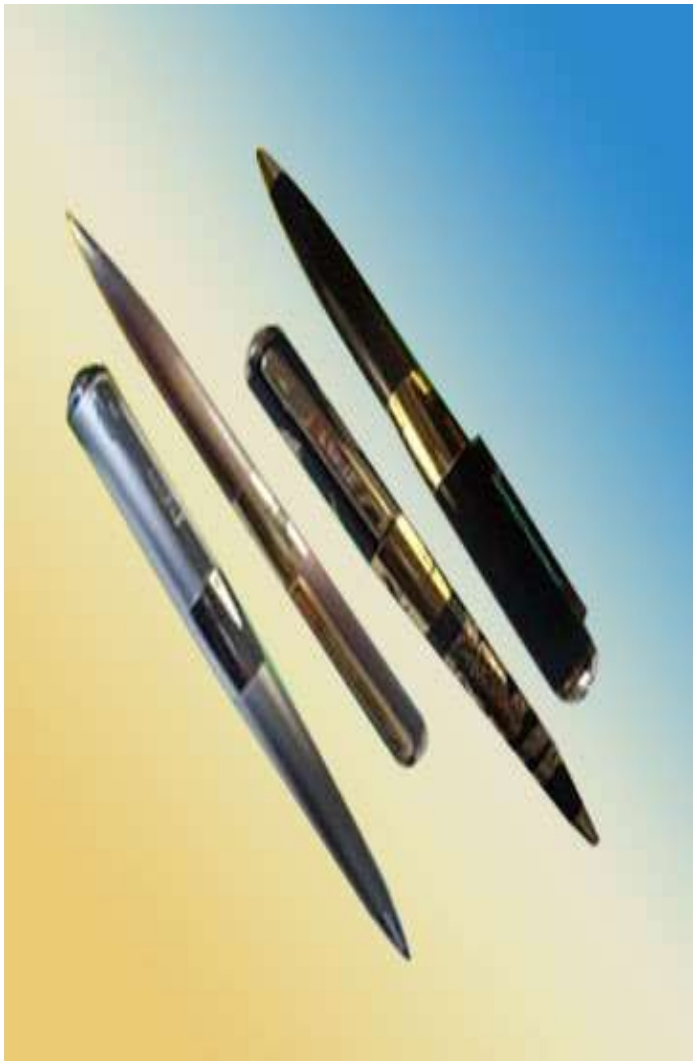
- Thường kết nối qua cổng USB
- Không phải dạng đĩa là bộ nhớ bán dẫn cực nhanh
- Dung lượng phát triển nhanh
- Gọn nhẹ và tiện lợi
- Đặc điểm đĩa Flash

1)Supports USB full-speed (12MBps) transmission
2) Driverless installation in Windows ME / 2000 / XP, Mac 9.0 and
above, Linux 2.4 and above
3) Supports boot-up by USB-HDD or USB-ZIP mode
4) LED indicator displays status

Flash disk

5) Write protection switch
6) Reading and writing speed: 900k/s and 700k/s
7) Password protection and data encryption prevents unauthorized

access to data
8) Application software support in Windows OS security function
9) Application software resize (partition) available
10) Capacity: 16MB, 32MB, 64MB, 128MB, 256MB, 512MB, 1GB
11) Compliance: FCC(B), CE, C-Tick



GiỚI THIỆU TỔNG QUAN VỀ RAID

Xuất xứ

- ✓ **RAID là cụm từ viết tắt nhóm từ Redundant Array of Inexpensive (Independent) Disks**
- ✓ **Thuật ngữ RAID được đưa ra trong một bài báo của một nhóm các nhà nghiên cứu tại Đại học tổng hợp California, Hoa Kỳ.**
- ✓ **RAID được đề xuất nhằm xóa bỏ khoảng trống lớn tốc độ CPU và các ổ đĩa điện cơ tương đối chậm.**
- ✓ **Hiệu suất thi hành vượt trội so với khi dùng một đĩa đơn lớn đắt tiền (SLED: Single Large Expensive Disk)**

GiỚI THIỆU TỔNG QUAN VỀ RAID

Khái niệm

RAID: là cấu trúc đa đĩa vật lý để tạo nên một đĩa logic có kích thước lớn, độ tin cậy và khả năng vận hành cao hơn.

Mục đích

- **Nâng cao hiệu suất vận hành của toàn bộ hệ thống. Khả năng làm việc song song các đĩa.**
- **An toàn dữ liệu tận dụng tính dư thừa dữ liệu nhằm cải thiện độ tin cậy đĩa.**
- **Cung cấp bộ nhớ lớn**

GiỚI THIỆU TỔNG QUAN VỀ RAID

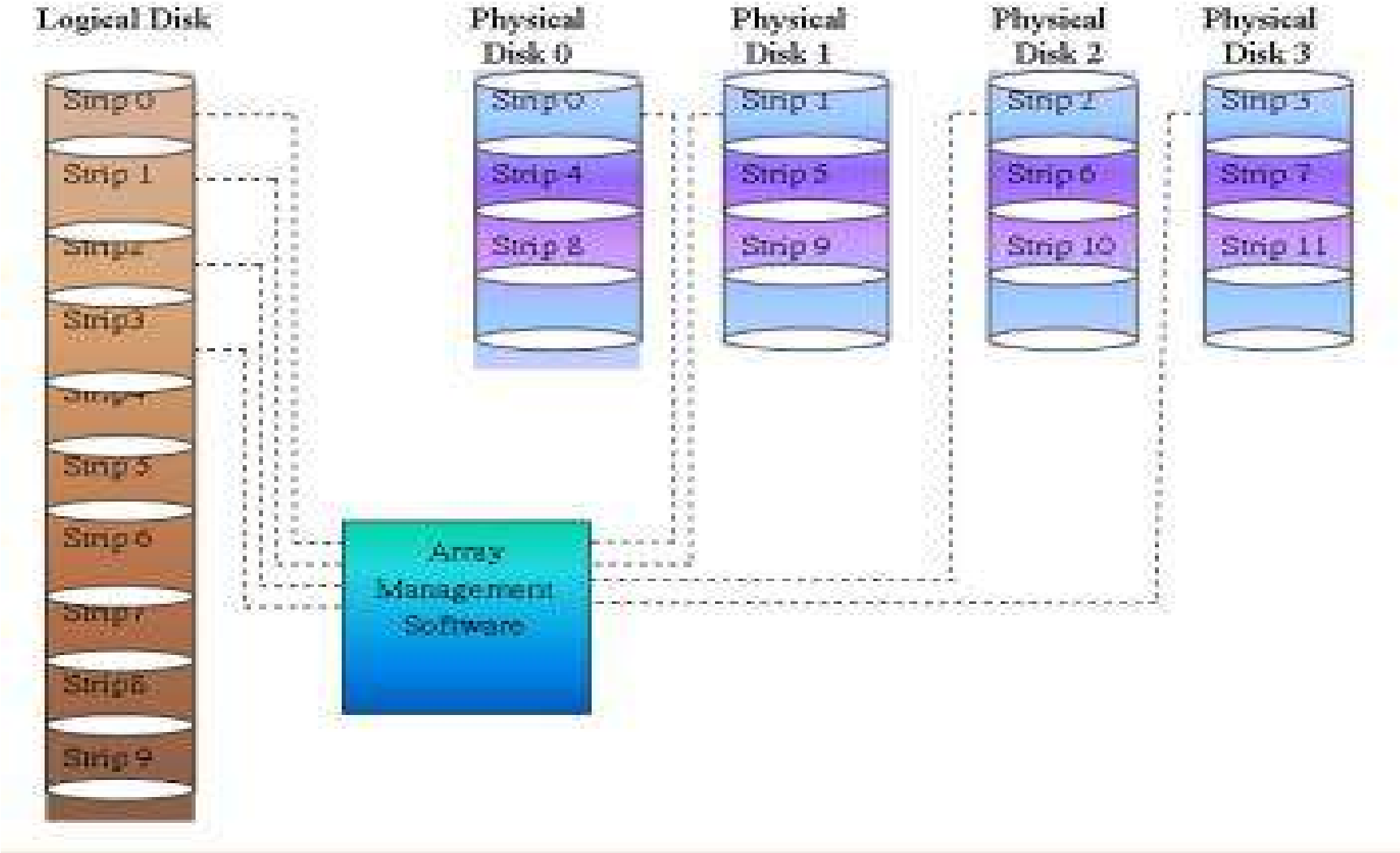
- **Đặc điểm chúng của RAID**

❖ **RAID là tập hợp các ổ đĩa vật lý được nhìn từ hệ điều hành như ổ đĩa logic đơn.**

❖ **Dữ liệu được phân bố trên mảng các ổ đĩa vật lý. Sử dụng kỹ thuật Striping. (Striping là kỹ thuật phân chia dữ liệu trên hai hay nhiều ổ đĩa làm tăng khả năng làm việc song song hệ thống)**

❖ **Dung lượng đĩa dư thừa được sử dụng để lưu trữ thông tin chẵn lẻ nhằm đảm bảo khả năng phục hồi dữ liệu trong trường hợp có hư hỏng về đĩa.**

GiỚI THIỆU TỔNG QUAN VỀ RAID

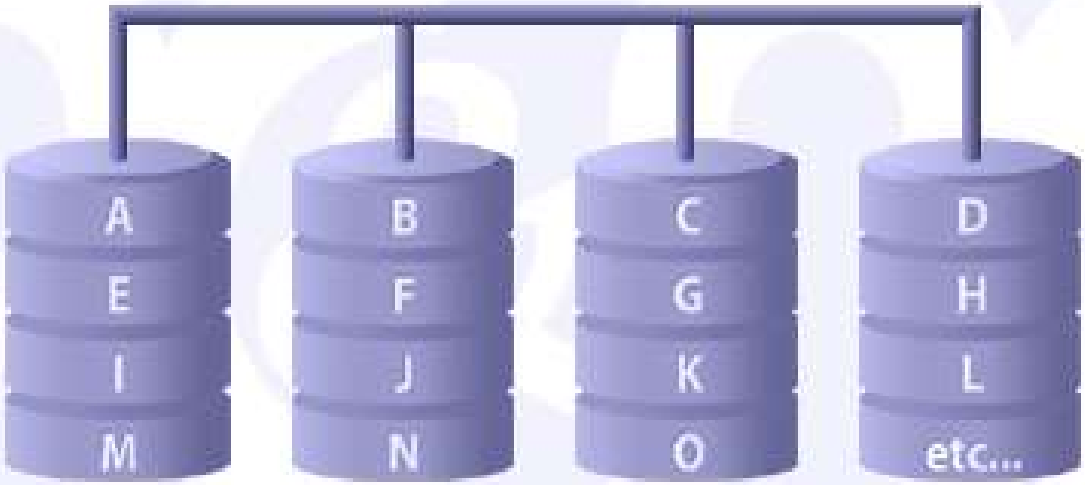


Các mức của RAID

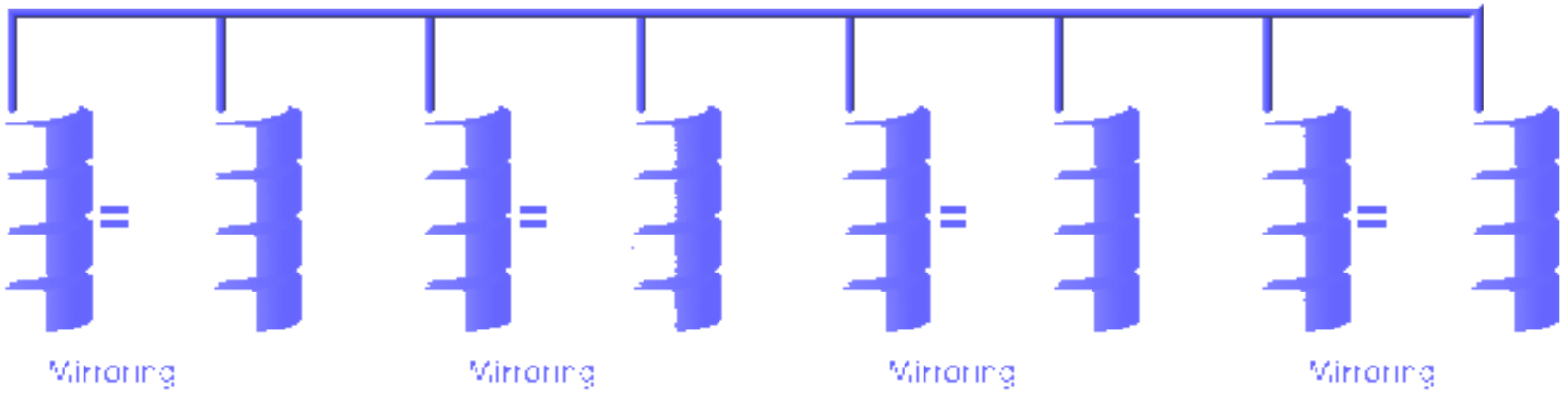
Có 6 mức chính của RAID

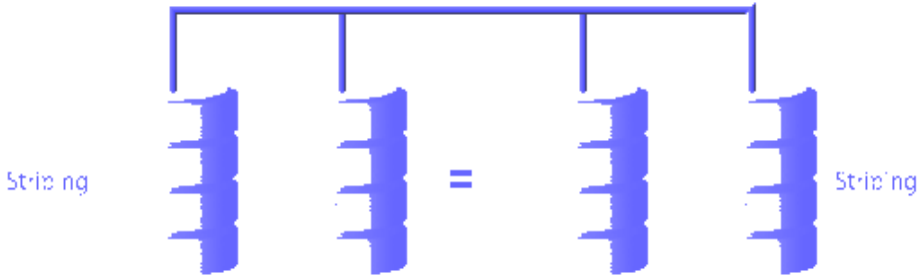
- RAID 0
- RAID 1
- RAID 2
- RAID 3
- RAID 4
- RAID 5

RAID LEVEL 0



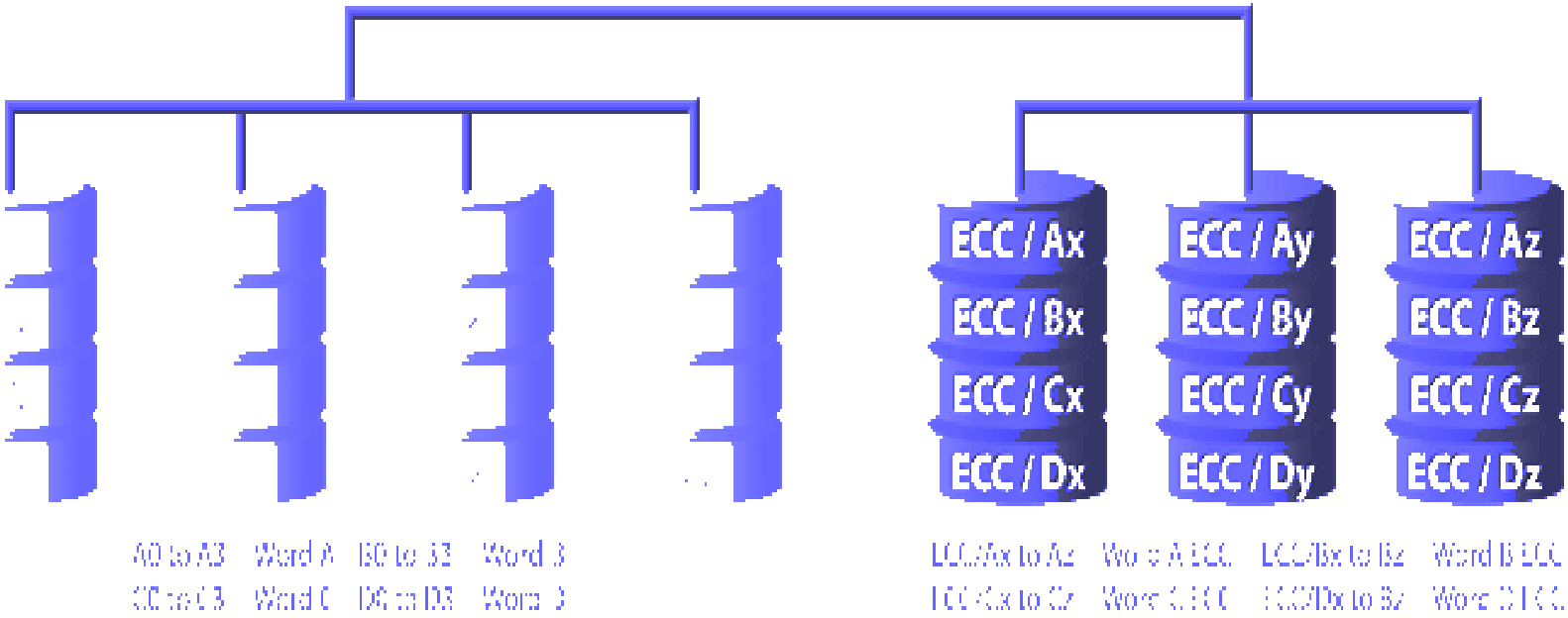
RAID LEVEL 1





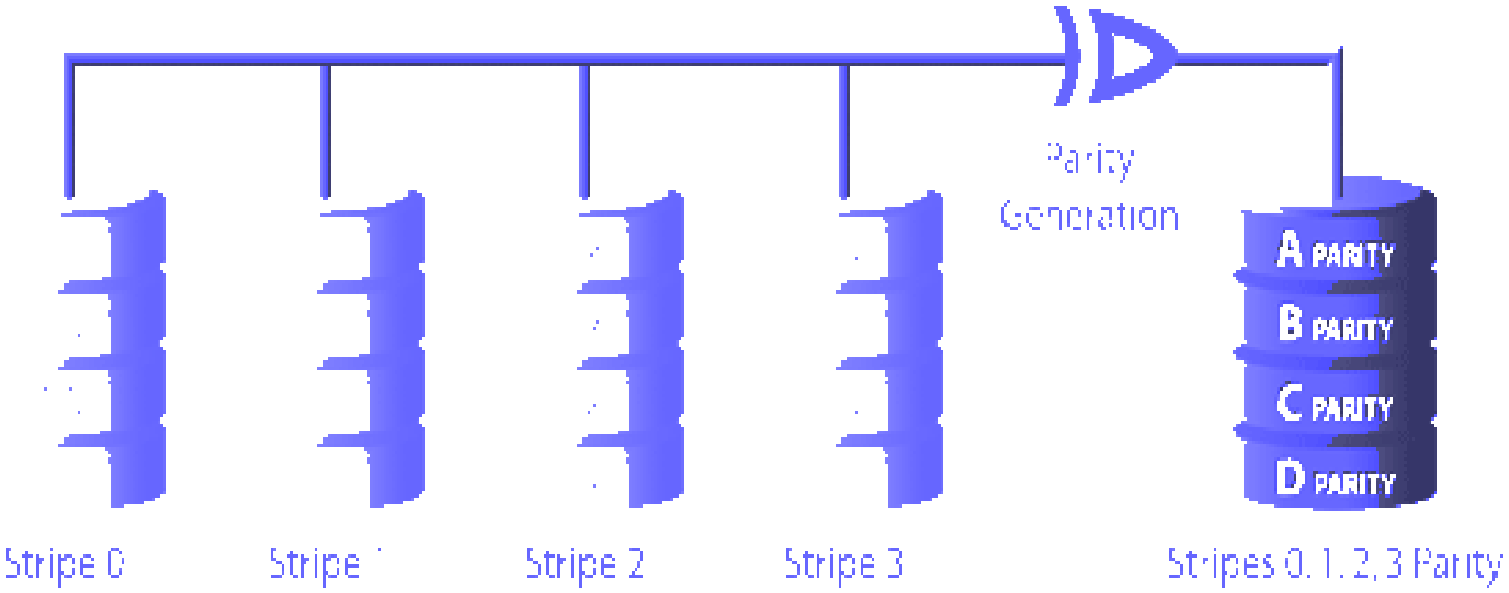
RAID LEVEL 1

RAID LEVEL 2



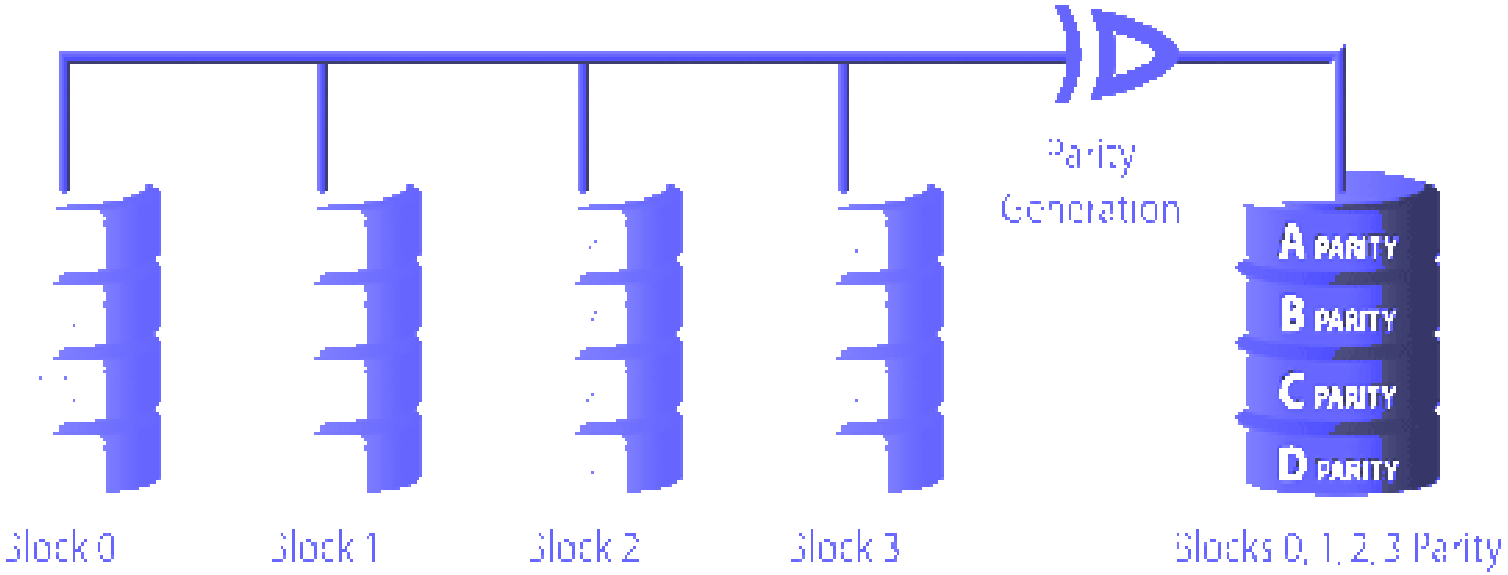
RAID LEVEL 2

RAID LEVEL 3



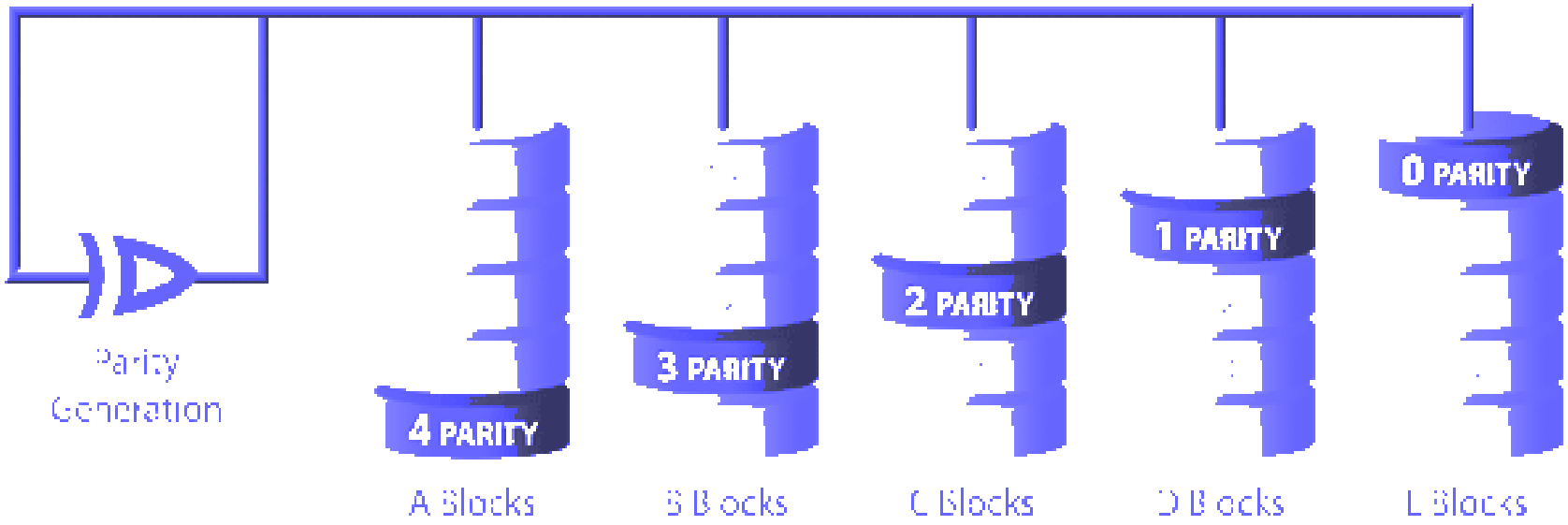
RAID LEVEL 3

RAID LEVEL 4



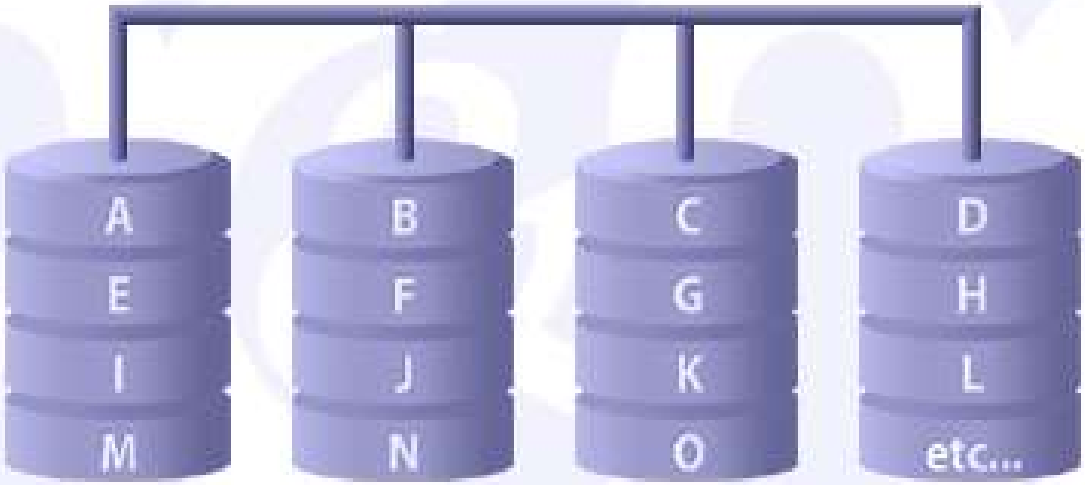
RAID LEVEL 4

RAID LEVEL 5



RAID LEVEL 5

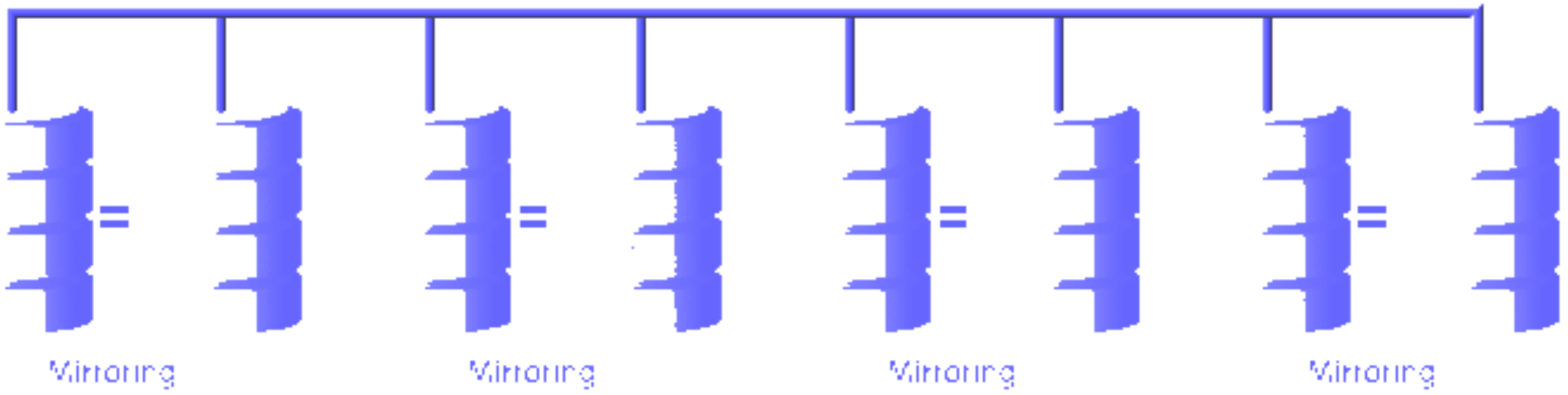
RAID LEVEL 0

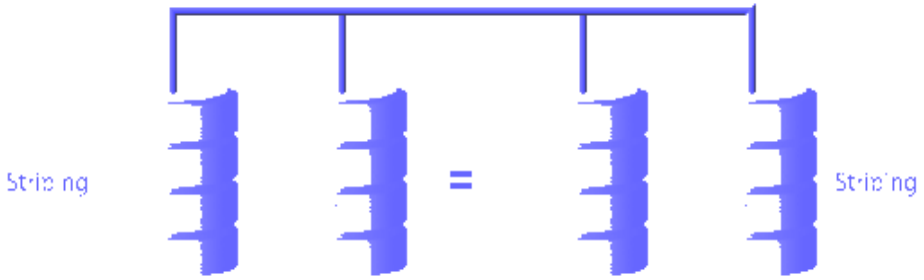


Đặc điểm chung RAID mức 0

- Có thể coi RAID 0 không là thành viên của RAID
- Dữ liêu được phân chia nhiều đĩa => có khả năng truyền dữ liệu song song.
- Không lưu trữ dữ liệu dư thừa
- Phù hợp hệ thống đòi hỏi dung lượng nhớ lớn và khả năng vận hành cao hơn là độ tin cậy trong hệ thống.

RAID LEVEL 1





RAID LEVEL 1

Đặc điểm chung RAID mức 1

- Là mức rất khác so các mức còn lại về cách lưu trữ dữ liệu dư thừa.
- Mỗi đĩa dữ liệu có một đĩa dự phòng đĩa dự phòng còn gọi mirror disk.

Ưu điểm:

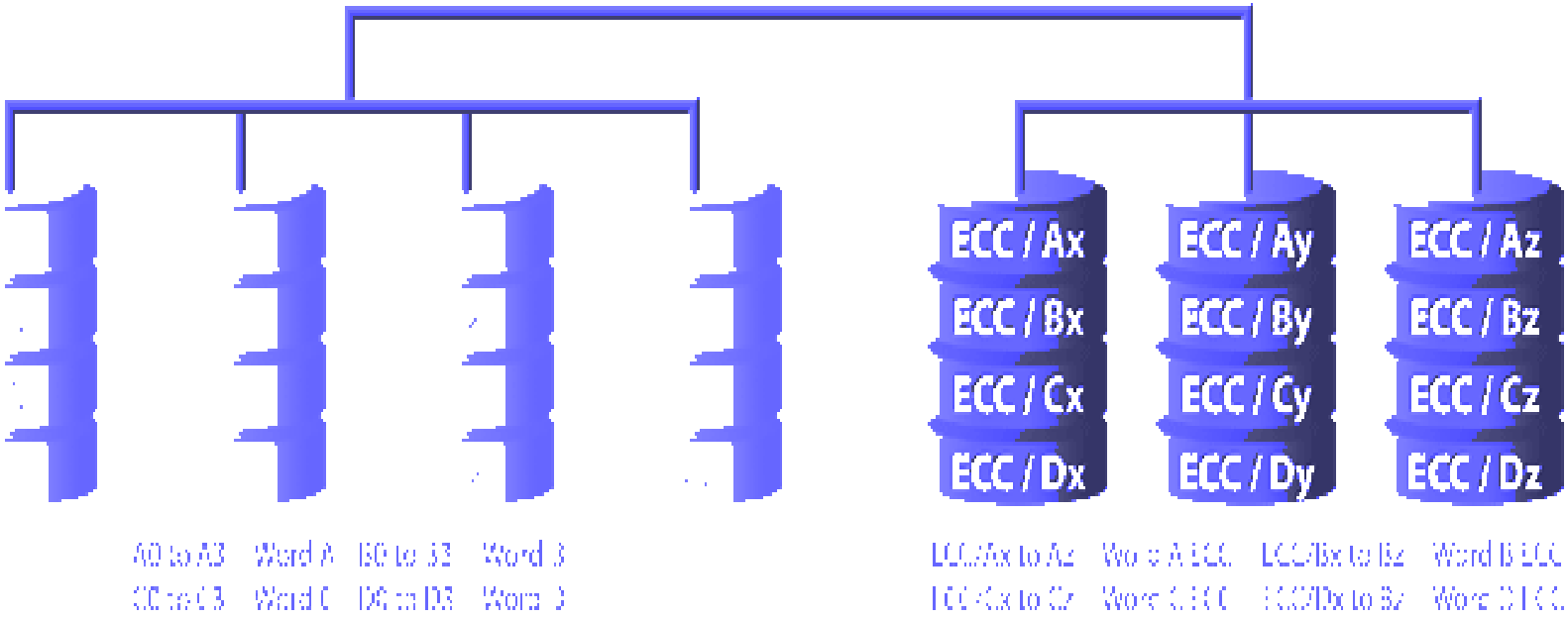
- Đáp ứng yêu cầu vào ra hệ thống
- Phục hồi dữ tốt nhất trong các mức của RAID

Nhược điểm:

- Khả năng cập nhật dữ liệu chậm
- Chi phí mua đĩa cao

KQ: Vận hành tốt cho hệ thống thường xuyên truy xuất dữ liệu

RAID LEVEL 2



RAID LEVEL 2

Đặc điểm chung RAID mức 2

- Sử dụng công nghệ truy cập song song.
- Tất cả đĩa đều vận hành tham gia yêu cầu trao đổi dữ liệu.
- Kích thước Strip có thể byte hay word.
- Có sử dụng mã Hamming để phát hiện lỗi và sửa lỗi

Ưu điểm:

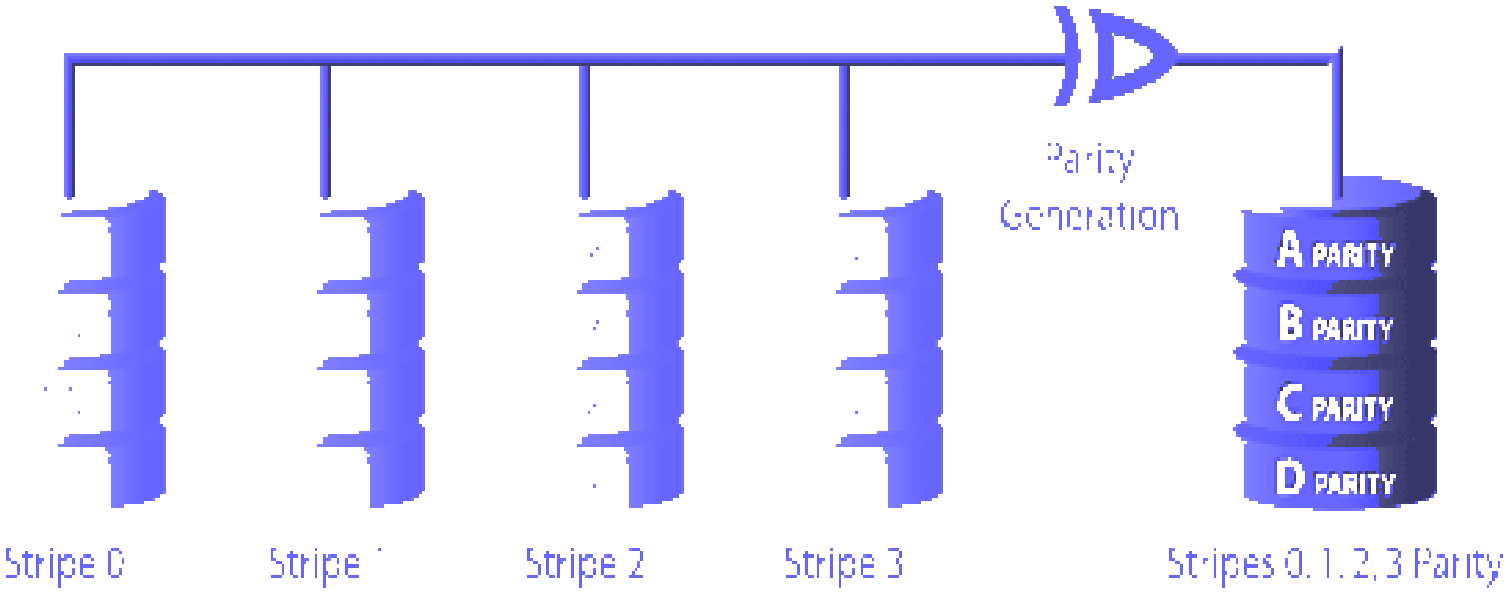
- Có khả năng phát hiện lỗi và sửa những lỗi đơn hệ thống.
- Số đĩa sử dụng ít hơn so mức RAID 1

Nhược điểm:

- Chi phí mua đĩa cao.

KQ: Ứng dụng trong hệ thống hay xuất hiện lỗi

RAID LEVEL 3



RAID LEVEL 3

Đặc điểm chung RAID mức 3

- Giống RAID 2 những tổ chức đơn giản hơn. Sử dụng một đĩa dự phòng.
- Tất cả đĩa đều vận hành tham gia yêu cầu trao đổi dữ liệu.
- Kích thước Strip có thể byte hay word.
- Có sử dụng mã Parity để phục hồi dữ liệu.

Ưu điểm:

➢ Có khả năng truyền dữ liệu song song.

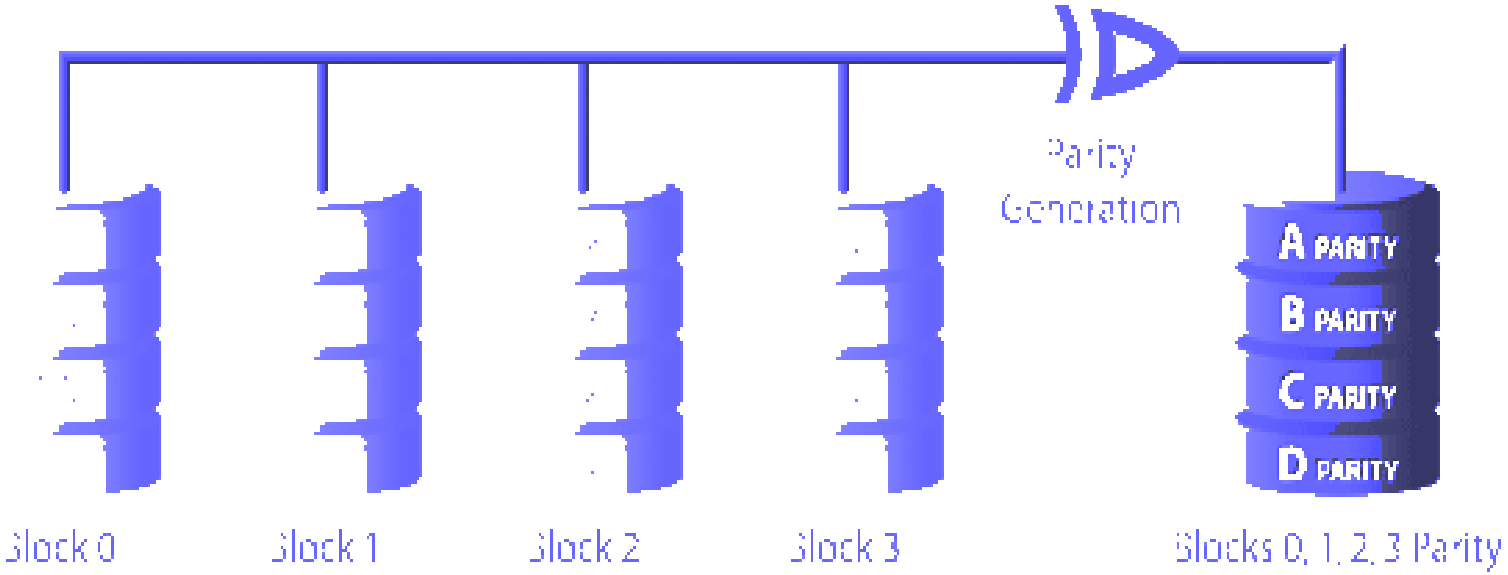
➢ Số đĩa sử dụng dự phòng là 1 đĩa. Chi phí thấp.

Nhược điểm:

➢ Tại một thời điểm chỉ thỏa mãn một yêu cầu vào ra.

KQ: Ứng dụng trong hệ thống hay xuất hiện lỗi

RAID LEVEL 4



RAID LEVEL 4

Đặc điểm chung RAID mức 4

- Giống RAID 3 những tổ chức đơn giản hơn. Sử dụng một đĩa dự phòng.
- Dữ liệu tổ chức thành khối.
- Các đĩa sử dụng phương pháp truy cập độc lập.
- Có sử dụng mã Parity để phục hồi dữ liệu.

Ưu điểm:

➢ Có khả năng đáp ứng nhiều yêu cầu vào ra đồng thời.

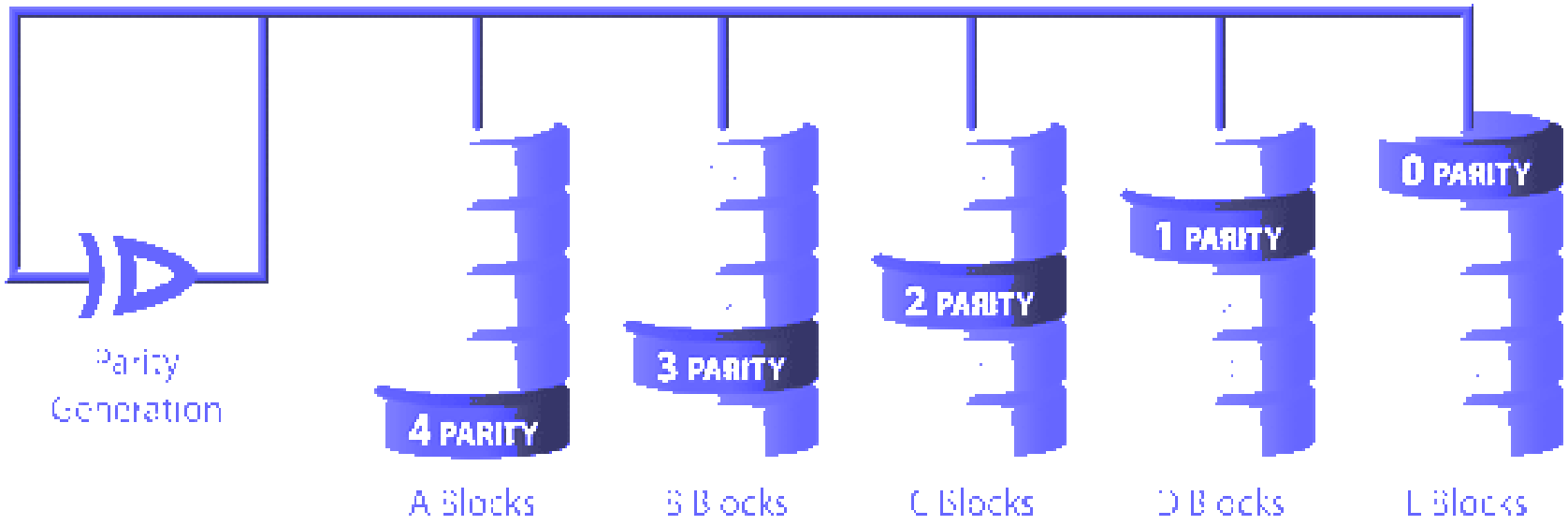
➢ Số đĩa sử dụng dự phòng là 1 đĩa. Chi phí thấp.

Nhược điểm:

➢ Khả năng truyền dữ liệu song song là kém.

KQ: Ứng dụng trong hệ thống hay xuất hiện lỗi

RAID LEVEL 5



RAID LEVEL 5

Đặc điểm chung RAID mức 5

- Giống RAID 4, tuy nhiên sự phân bố đều thông tin dư phòng tránh được hiện tượng tắc nghẽn(bottle neck)
- Dữ liệu tổ chức thành khối.
- Các đĩa sử dụng phương pháp truy cập độc lập.
- Có sử dụng mã Parity để phục hồi dữ liệu.

Ưu điểm:

➢ Có khả năng đáp ứng nhiều yêu cầu vào ra đồng thời.

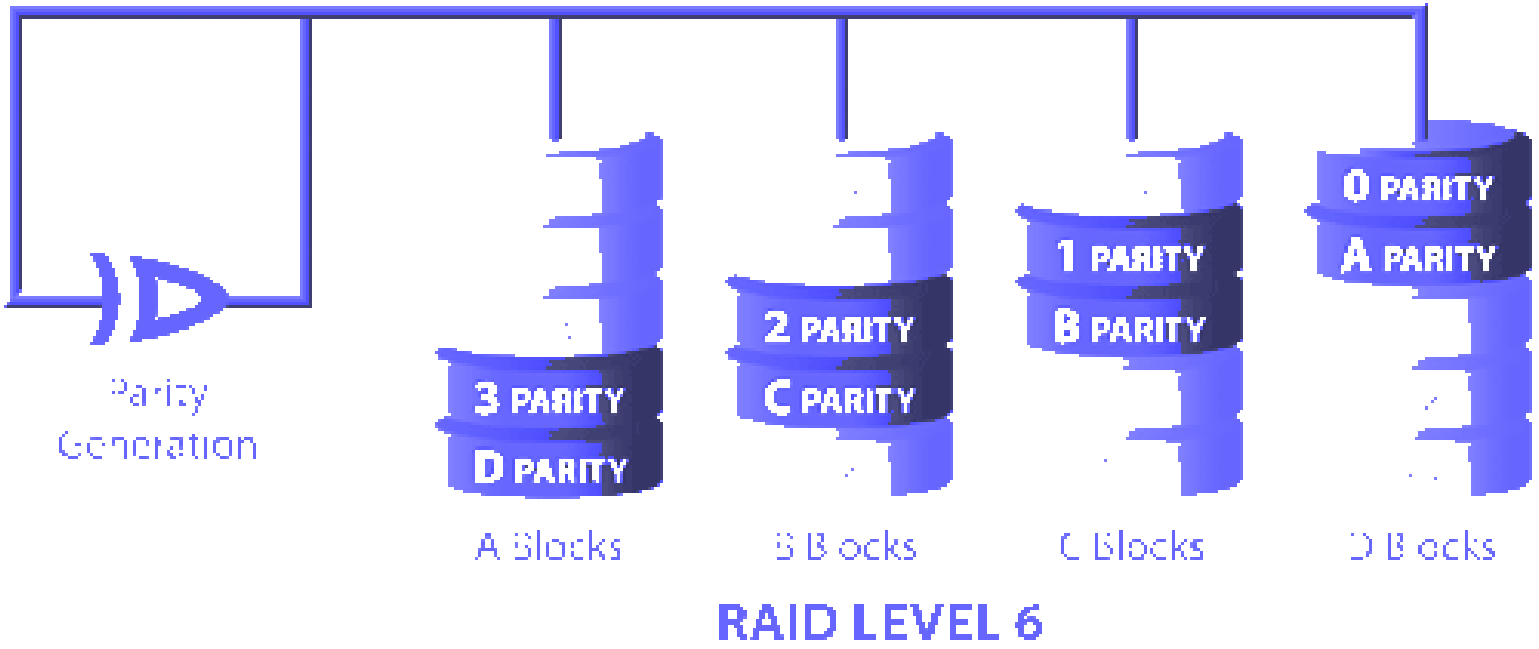
➢ Số đĩa sử dụng dự phòng là 1 đĩa. Chi phí thấp.

Nhược điểm:

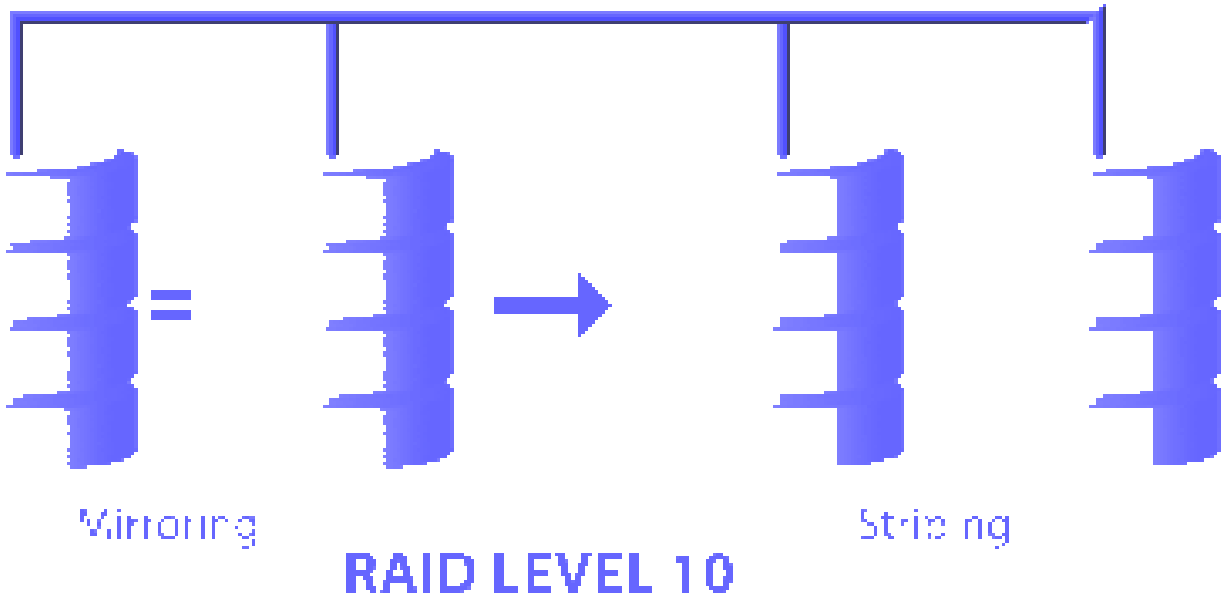
➢ Khả năng truyền dữ liệu song song là kém.

KQ: Ứng dụng nhiều trong thực tế.

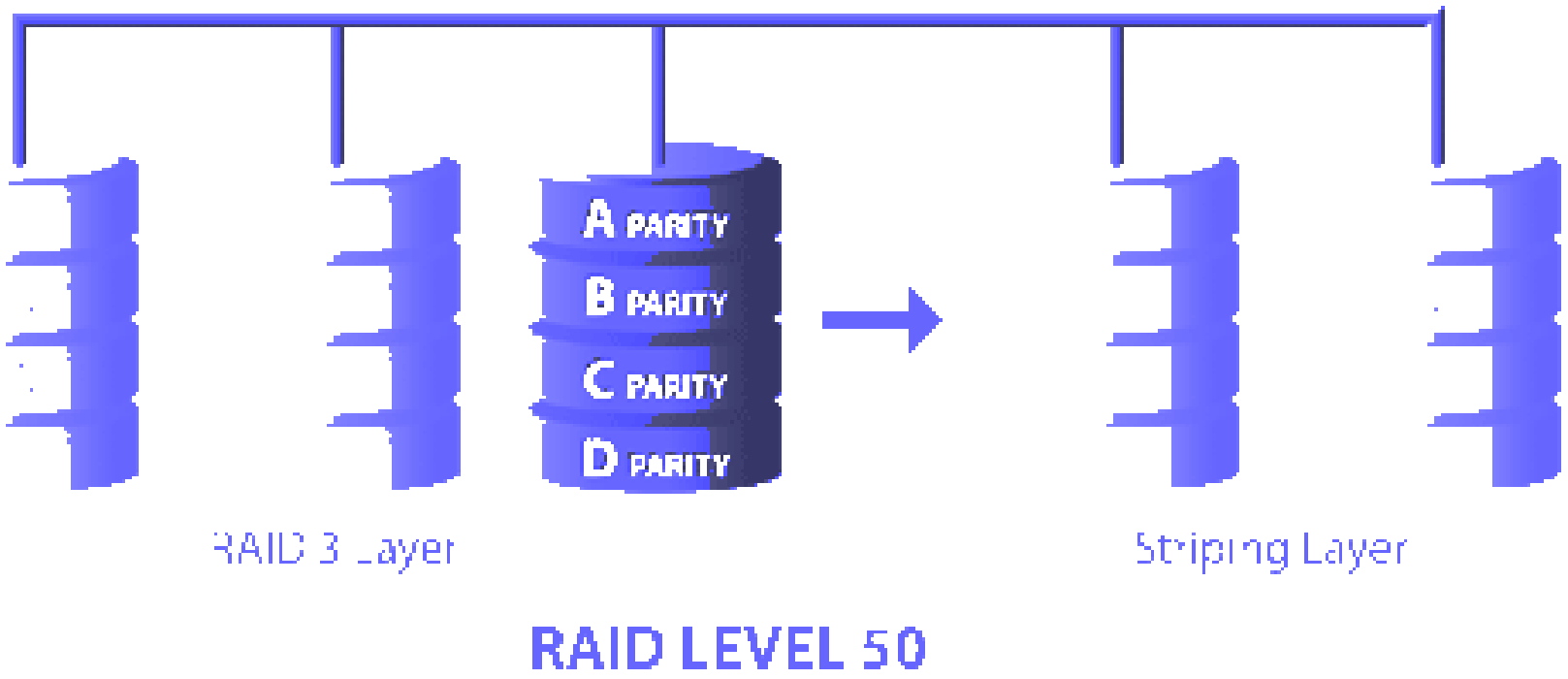
RAID LEVEL 6


Parity
Generation
3 PARITY
D PARITY
2 PARITY
C PARITY
1 PARITY
B PARITY
0 PARITY
A PARITY
A Blocks
B Blocks
C Blocks
D Blocks
RAID LEVEL 6

RAID LEVEL 10



RAID LEVEL 50



5.6 Hệ thống nhớ trên máy PC hiện nay

- Hệ thống Cache: tích hợp trực tiếp trên các chip vi xử lý
- Bộ nhớ chính: tồn tại dưới dạng module nhớ RAM
- SIMM: Single Inline Memory Module
- 30 pin : 8 đường dữ liệu
- 72 pin : 32 đường dữ liệu
- DIMM: Dual Inline Memory Module
- 168 pin: 64 đường dữ liệu
- RIMM:Rambus Inline Memory Module

ROM BIOS

ROM BIOS: Basic Input Output System ROM chứa chương trình sau:

- Chương trình POST (Power On Self Test)
- Chương trình CMOS setup (Compementary Metal Oxide Semiconductor)
- Chương trình Bootstrap Looader
- Chương trình điều khiển vào ra cơ bản (BIOS)
- CMOS RAM
- Chứa cấu hình hệ thống hiện thời
- Đồng hồ và ngày tháng năm hệ thống
- Có pin nuôi riêng

Chương 6

Giới thiệu chung

6.1 Tổng quan về hệ thống vào ra

6.2 Các phương pháp điều khiển vào ra

6.3 Nối ghép thiết bị ngoại vi

6.4 Các cổng vào ra thông dụng

6.1 Tổng quan về hệ thống vào ra

Giới thiệu chung hệ thống vào ra

***Chức năng*: Trao đổi thông tin giữa Máy tính với môi trường bên ngoài.**

Các thao tác cơ bản:

- ✓ Vào dữ liệu
- ✓ Ra dữ liệu

Các thành phần chính:

- ✓ Thiết bị ngoại vi
- ✓ Module ghép nối vào ra

6.1 Tổng quan về hệ thống vào ra

Thiết bị ngoại vi

Chức năng: phương tiện chuyển đổi thông tin giữa bên trong và bên ngoài máy tính

Đặc điểm các thiết bị

Trên thị trường tồn tại rất nhiều các thiết bị ngoại vi khác nhau về: Nguyên tắc hoạt động, tốc độ, định dạng dữ liệu truyền, v.v. Đồng thời các thiết bị này có tốc độ làm việc chậm hơn CPU và RAM rất nhiều. Chính vì lý do trên cần có Module vào ra để ghép nối các thiết bị ngoại vi vào hệ thống BUS máy tính.

6.1 Tổng quan về hệ thống vào ra

Phân loại:

- Thiết bị nhập: Keyboard, Mouse, Scan, Micro,…
- Thiết bị xuất: Monitor, Printer,
- Thiết bị xuất nhập: Modem, NIC, Driver,…

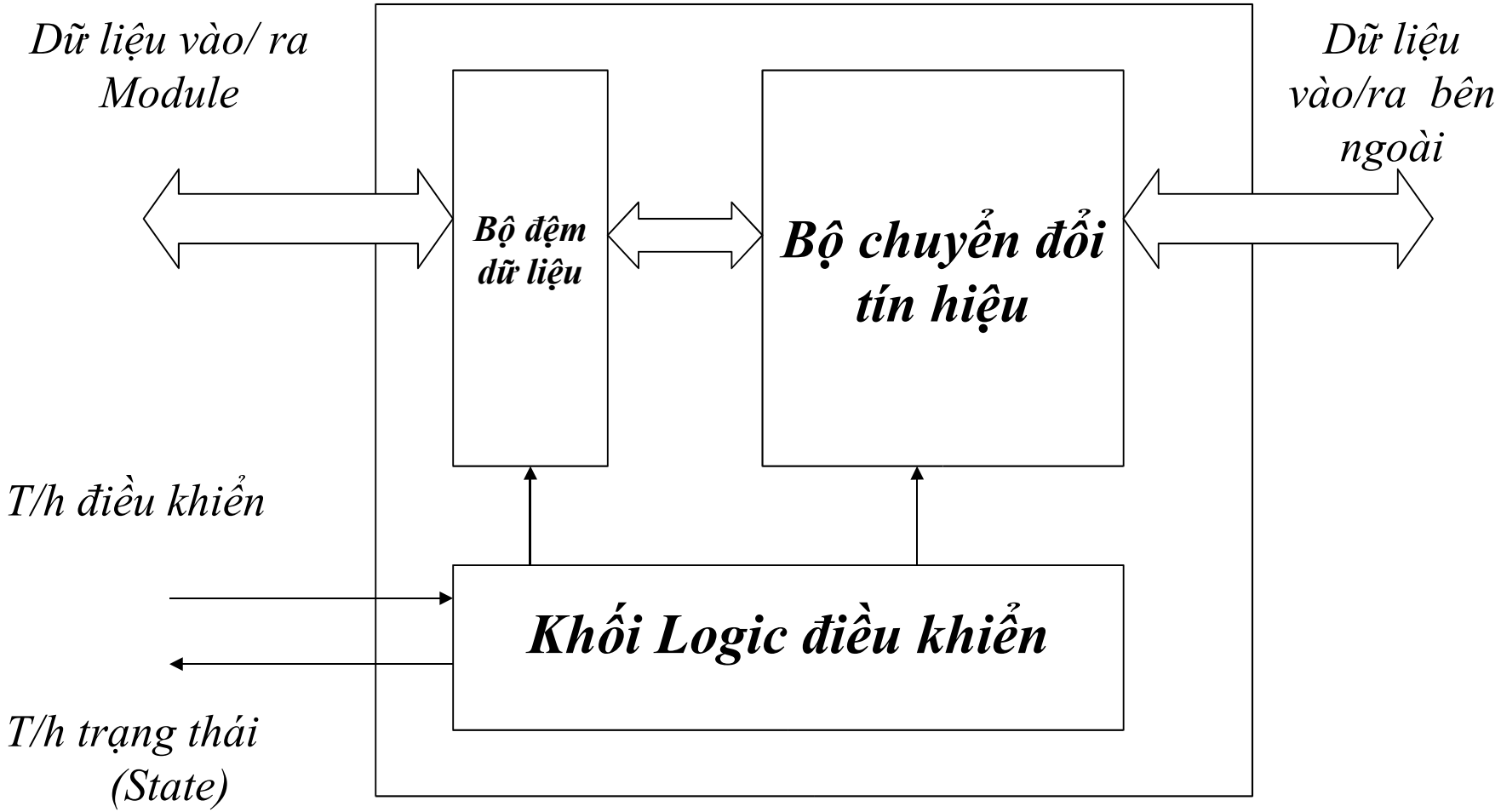
Cấu trúc tổng quát của thiết bị ngoại vi:

✓Bộ chuyển đổi tín hiệu: chuyển đổi dữ liệu giữa bên trong và bên ngoài Máy tính

✓Bộ đệm dữ liệu: nơi lưu trữ dữ liệu trung gian giữa Máy tính và thiết bị ngoại vi, đặt bên trong thiết bị ngoại vi.

✓Khối logic điều khiển: điều khiển hoạt động của thiết bị ngoại vi theo tín hiệu từ Module I/O gởi tới thiết bị.

6.1 Tổng quan về hệ thống vào ra



6.1 Tổng quan về hệ thống vào ra

Module I/O

Chức năng: Nối ghép thiết bị ngoại vi với bus của máy tính.

- ✓ Điều khiển và định thời
- ✓ Trao đổi thông tin với CPU
- ✓ Trao đổi thông tin với thiết bị ngoại vi
- ✓ Đệm giữa máy tính với thiết bị ngoại vi
- ✓ Phát hiện lỗi của các thiết bị ngoại vi.

Cấu trúc chung:

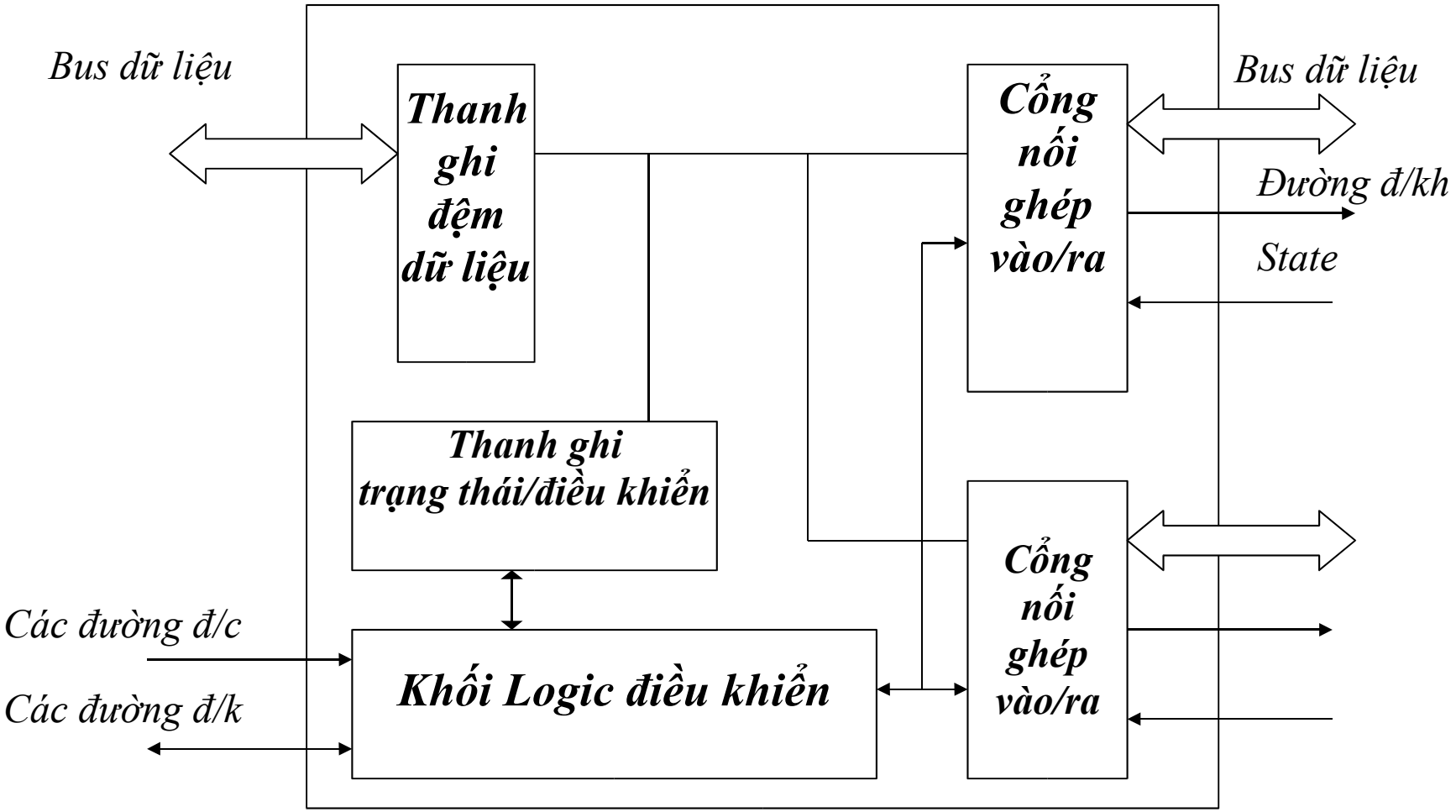
Thanh ghi đệm dữ liệu: đệm dữ liệu trong quá trình trao đổi

Cổng nối ghép vào ra: kết nối thiết bị ngoại vi, mỗi cổng có địa chỉ xác định và chuẩn kết nối riêng phụ thuộc sơ đồ chân.

Thanh ghi trạng thái/điều khiển: lưu trữ thông tin trạng thái cho các cổng vào ra

Khối logic điều khiển: điều khiển Module vào ra

6.1 Tổng quan về hệ thống vào ra



Ví dụ cổng ghép nối song song(LPT)

- Các đường dẫn của cổng song được nối với 3 thanh ghi 8 bit khác nhau:
  - ✓ Thanh ghi dữ liệu (Địa chỉ cơ sở)
  - ✓ Thanh ghi trạng thái (Địa chỉ cơ sở +1)
  - ✓ Thanh ghi điều khiển (Địa chỉ cơ sở +2)
- Các đại chỉ cổng có thể là:

  LPT1: 378h (379h ; 37Ah)

  LPT2: 3BCh

  LPT3: 278h

  LPT4: 2BCh

Ví dụ cổng ghép nối song song(LPT)

- Hợp ngữ:

Để xuất ra dữ liệu: **OUT** DX, AL hoặc **OUT** DX, AX

Để nhập vào dữ liệu: **IN** AL, DX hoặc **IN** AX, DX

(DX chứa địa chỉ; AL chứa giá trị)

- Turbo C

Để xuất ra dữ liệu: **outportb(đia_chỉ, giá_trị)**

Để nhập vào dữ liệu: **bien = inportb(địa_chỉ)**

- Turbo Pascal

Để xuất ra dữ liệu: **port[đia_chỉ]:= giá_trị**

Để nhập vào dữ liệu: **bien:=port[địa_chỉ]**

Ví dụ cổng ghép nối tiếp(COM)

- Các thanh ghi chính:
  - ✓ Thanh ghi đệm (Buffer Register) Địa chỉ cơ sở
  - ✓ Thanh ghi trạng thái (Status Register) ĐCCS+5
  - ✓ Thanh ghi điều khiển (Control Register) ĐCCS+3
- Các địa chỉ cổng có thể là:

  COM1: 3F8h (3FDh ; 3FBh)

  COM2: 2F8h

  COM3: 3E8h

  COM4: 2E8h

6.2 Các phương pháp điều khiển vào ra

Phân loại:

- Vào ra bằng chương trình
- Vào ra bằng ngắt
- Truy cập bộ nhớ trực tiếp DMA

Vào ra bằng chương trình

Nguyên tắc chung:

- ✓ Sử dụng lệnh vào ra trong chương trình để trao đổi dữ liệu với cổng vào ra.
- ✓ Khi CPU thực hiện chương trình gặp lệnh vào ra thì CPU điều khiển trao đổi dữ liệu với cổng vào ra.

Lệnh I/O:

- ✓ Với không gian địa chỉ vào ra riêng biệt: sử dụng các lệnh vào ra chuyên dụng
- ✓ Với không gian vào ra dùng chung bộ nhớ thì các lệnh trao đổi dữ liệu sử dụng như ngăn nhớ.

6.2 Các phương pháp điều khiển vào ra

Hoạt động vào ra bằng chương trình

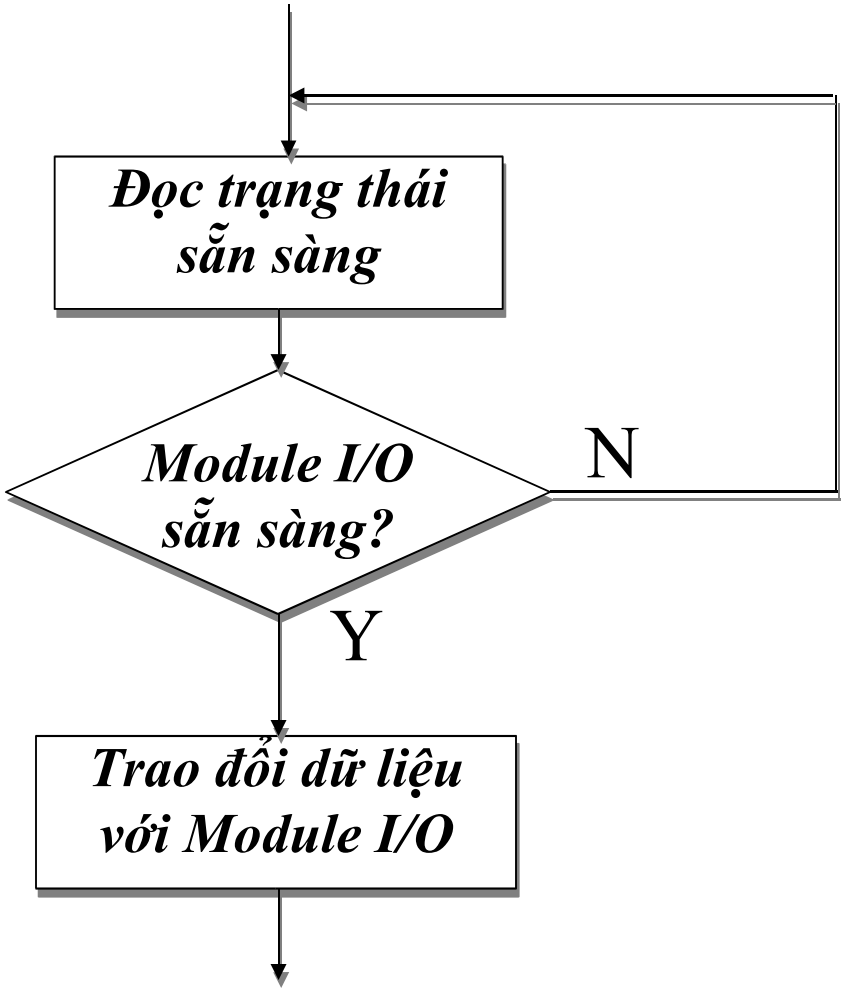
- CPU gặp lệnh trao đổi vào ra, yêu cầu thao tác vào ra
- Module vào ra thao tác vào ra
- Module vào ra thiết lập các bit trạng thái(State)
- CPU kiểm tra các bit trạng thái:

  Nếu chưa sẵn sàng thì quay lại kiểm tra lại

  Nếu sẵn sàng thì chuyển sang trao đổi dữ liệu với Module vào ra.

6.2 Các phương pháp điều khiển vào ra

Lưu đồ thực hiện chương trình:



6.2 Các phương pháp điều khiển vào ra

Nhận xét:

✓CPU trực tiếp điều khiển vào ra: đọc trạng thái, kiểm tra trạng thái, thực hiện trao đổi.

✓ Trong trường hợp nhiều thiết bị cùng cần trao đổi dữ liệu và thiết bị chưa sẵn sàng tốn rất nhiều thời gian CPU

✓ Việc thực hiện trao đổi đơn giản

Vào ra bằng ngắt

Nguyên tắc chung:

✓ CPU không phải đợi trạng thái sẵn sàng của Module vào ra.

✓ Module vào ra khi nó sẵn sàng phát ra tín hiệu yêu cầu ngắt CPU

✓ CPU thực hiện chương trình vào ra tương ứng để trao đổi dữ liệu.

✓ CPU trở lại chương trình đang bị ngắt.

6.2 Các phương pháp điều khiển vào ra

Các phương pháp nối ghép

- Sử dụng nhiều đường yêu cầu ngắt.
- Kiểm tra vòng bằng phần mềm (Polling)
- Kiểm tra vòng bằng phần cứng
- Sử dụng bộ điều khiển ngắt.
- Nhiều yêu cầu ngắt đồng thời
- CPU sử dụng nhiều đường yêu cầu ngắt. Nạp vào thanh ghi yêu cầu ngắt.
- Hạn chế số lượng Module vào ra
- Các đường ngắt được qui định mức ưu tiên.

6.2 Các phương pháp điều khiển vào ra

- CPU phát ra tín hiệu chấp nhận ngắt đến Module đầu tiên.
- Nếu Module đó không gây ra ngắt thi nó gởi tín hiệu đó tới các Module kế tiếp
- Module I/O gây ngắt sẽ đặt vector lên bus dữ liệu
- CPU sử dụng ngắt để xác định chương trình con điều khiển ngắt
- Thứ tự vào ra các Module trong chuỗi xác định thứ tự ưu tiên.

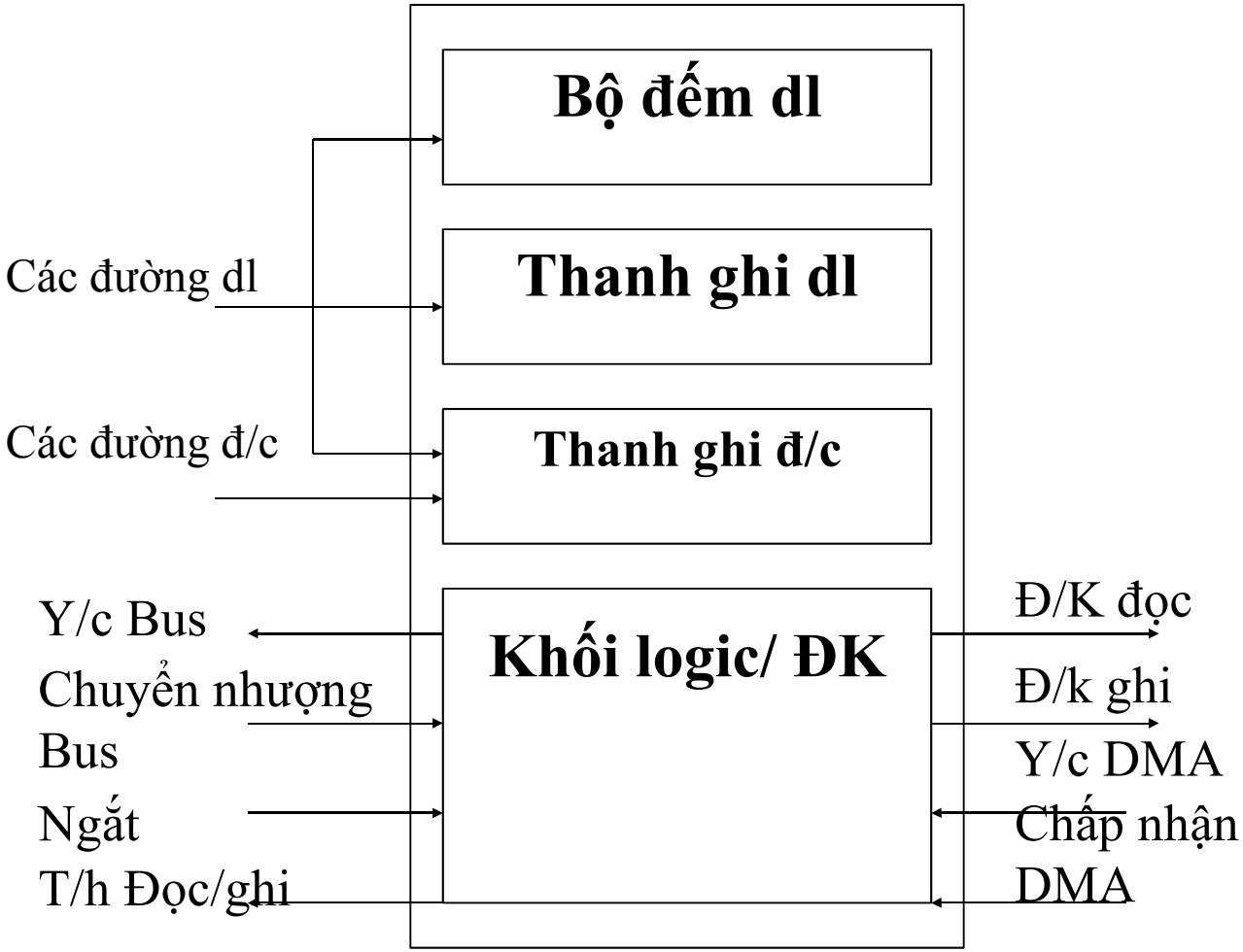
6.2 Các phương pháp điều khiển vào ra

Truy nhập bộ nhớ trực tiếp

(DMA: Direct Memory Access)

Với nhược điểm chính của hai phương pháp trên là: CPU tham gia trực tiếp vào trao đổi dữ liệu và việc trao đổi lượng dữ liệu nhỏ. Để khắc phục hai phương pháp trên một phương pháp mới có tên DMA sẽ sử dụng thêm một Module phần cứng có DMAC (DMA Controller). Vì vậy khi trao đổi dữ liệu không cần CPU.

6.2 Các phương pháp điều khiển vào ra



6.2 Các phương pháp điều khiển vào ra

Các thành phần của DMAC

- Thanh ghi dữ liệu: chứa dữ liệu trao đổi.
- Thanh ghi địa chỉ: chứa địa chỉ của ngăn nhớ dữ liệu
- Bộ đếm dữ liệu: chứa số từ dữ liệu cần trao đổi
- Khối logic điều khiển: điều khiển hoạt động của DMAC

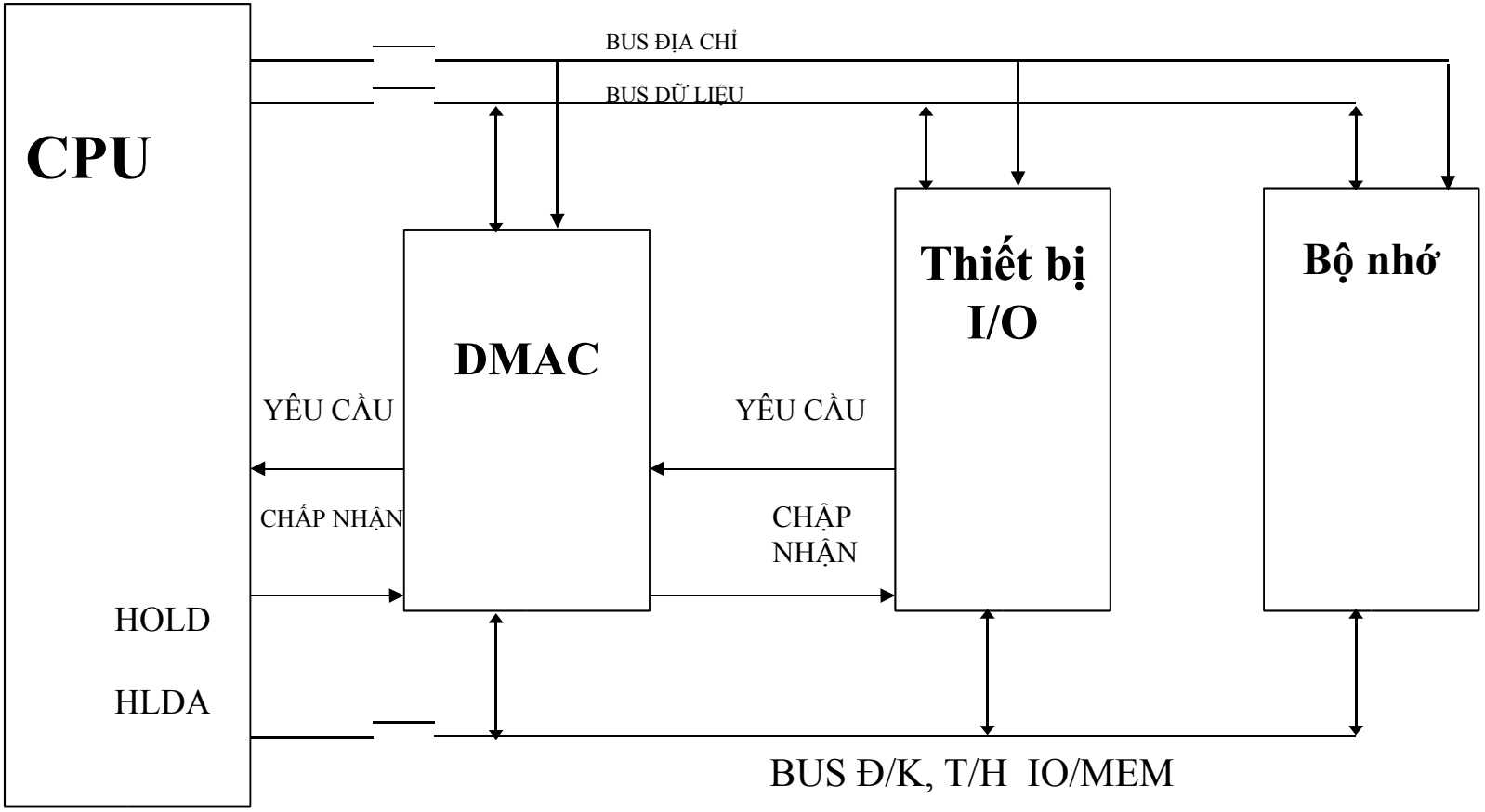
Hoạt động của DMA

- Khi cần vào ra dữ liệu thì CPU nhờ DMAC tiến hành vào ra dữ liệu với thông tin cho biết như sau:

6.2 Các phương pháp điều khiển vào ra

- Địa chỉ thiết bị vào ra
- Địa chỉ đầu của mảng nhớ chứa dữ liệu và DMAC nạp thanh ghi địa chỉ
- Số từ dữ liệu cần truyền và DMAC nạp vào bộ đếm dữ liệu
- CPU sẽ đi thực hiện việc khác
- DMAC điều khiển việc trao đổi dữ liệu sau khi truyền một từ dữ liệu thì nội dung thanh ghi địa chỉ tăng lên và nội dung bộ đếm dữ liệu giảm xuống một đơn vị.
- Khi bộ đếm bằng dữ liệu bằng 0, DMAC gởi tín hiệu ngắt CPU để báo kết thúc DMA

6.2 Các phương pháp điều khiển vào ra



6.2 Các phương pháp điều khiển vào ra

Các kiểu thực hiện DMA

- DMA truyền theo khối: DMAC sử dụng BUS để truyền cả khối dữ liệu (CPU chuyển nhượng BUS cho DMAC)
- DMA lấy chu kỳ: DMAC cưỡng bức CPU treo tạm thời từng chu kỳ BUS để thực hiện truyền một từ dữ liệu
- DMA trong suốt: DMAC nhận biết những chu kỳ nào CPU không sử dụng BUS thì chiếm BUS để trao đổi dữ liệu (DMAC lấy lén chu kỳ)

Đặc điểm DMA

- CPU không tham gia trong quá trình trao đổi dữ liệu
- DMAC điều khiển trao đổi dữ liệu giữa bộ nhớ chính và Module vào ra với tốc độ nhanh.
- Phù hợp với yêu cầu trao đổi mảng dữ liệu có kích thước lớn.

6.2 Các phương pháp điều khiển vào ra

Bộ xử lý vào ra

- Việc điều khiển vào ra được sử dụng bởi một bộ điều khiển vào ra chuyên dụng.
- Bộ xử lý vào ra hoạt động theo chương trình riêng của nó.
- Chương trình của bộ xử lý vào ra có thể nằm trong bộ nhớ chính hoặc bộ nhớ riêng.
- Hoạt động theo kiến trúc đa xử lý

6.3 Nối ghép thiết bị ngoại vi

Nối ghép thiết bị ngoại vi

Các kiểu nối ghép vào ra

- Nối ghép song song
- Nối ghép nối tiếp

Nối ghép song song

- ✓ Truyền các bit song song
- ✓ Tốc độ truyền nhanh
- ✓ Cần đường truyền song song
- ✓ Tốn nhiều dây dẫn

6.3 Nối ghép thiết bị ngoại vi

Nối ghép nối tiếp

- Truyền lần lượt từng bit
- Cần có bộ chuyển đổi từ song song sang nối tiếp
- Tốc độ chậm
- Cần ít đường truyền dữ liệu

Các cấu hình ghép nối ghép

- ✓ Điểm tới điểm (point to point): Qua một cổng vào ra chỉ có thể ghép một thiết bị ngoại vi (PS/2, COM, LPT,…)
- ✓ Điểm tới đa điểm (Point to multipoint): Thông qua một cổng vào ra ghép nhiều thiết bị vào ra. Ví dụ: SCSI(7,15), USB (127),…

6.4 Các cổng vào ra thông dụng

Các cổng vào ra thông dụng

- PS/2 : nối ghép bàn phím và chuột
- VGA(Video Graphic Adapter): Cổng nối ghép màn hình
- LPT (Line PrinTer): nối ghép với máy in là cổng song song
- COM (COMmunication): nối ghép với Modem, chuột, và thiết bị khác. Cổng nối tiếp 9 hoặc 25 chân.
- USB: cổng nối tiếp đa năng cho phép nối ghép nối tiếp tối đa 17 thiết bị thông qua Hub.

THE END

Bài giảng
Cấu trúc máy tính

MỤC LỤC

3-Cài đặt MSOFFICE

Phần 5 Tổng thành và nâng cấp máy tính

1-Lựa chọn các bộ phận để tổng thành lắp ráp 1 máy PC: Mainboard,RAM,card màn hình,card sound,I/O,ổ cứng ,CD-ROM

2-Nâng cấp : Thay Mainboard,RAM,card màn hình,card sound,I/O,
ổ cứng ,CD-ROM

Phần 6 Phụ lục : -1 số thông số của Mainboard và Card

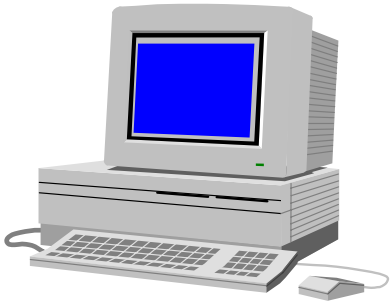
- Chương trình lưu Master boot

- Chương trình Lưu CMOS

PHẦN 1 CHỨC NĂNG NHIỆM VỤ ,CẤU TẠO CÁC BỘ PHẬN MÁY TÍNH .

CÁC BỘ PHẬN CỦA HỆ THỐNG MÁY TÍNH

Sơ đồ cấu thành về chức năng:



Một hệ thống máy PC thường có các thành phần cấu thành :

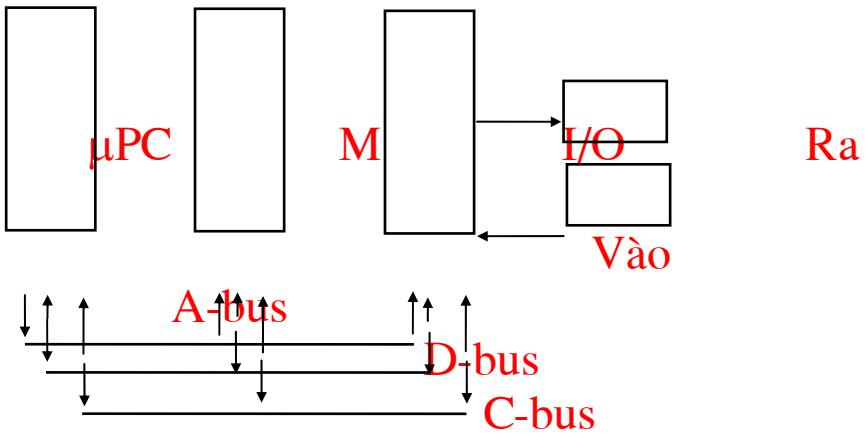
- Màn hình
- Bàn phím
- Chuột

- Hộp CPU:
  + Bảng mạch chính (Mainboard)
  + Đĩa cứng (Hard Disk)
  + ổ đĩa mềm
  + ổ CD ROM

Sau đây ta sẽ đI sâu vào hoạt động của từng phần

MAINBOARD

Mainboard chứa các linh kiện chính và các đường dây dẫn kết nối chúng lại tạo nên máy tính PC.



Từ sơ đồ tổng quát của hệ vi xử lý mà máy tính PC là 1 trường hợp tiêu biểu,so
sánh với 1 Mainboard cụ thể ta thấy trên Mainboard có gắn:

-µPC Microprossecor
- Bộ nhớ : ROM,RAM,Cache,PAL
- Các khe cắm để cắm các bảng mạch vào ra (I/O).Với các Mainboard đời mới
các card này được làm liền trên bảng mạch chính (onboard).
- Các vi xử lý bổ trợ :8087,8259,8037,8250...
-Các chuyển mạch hệ thống.

Các đường mạch in trên Mainboard làm dây dẫn có thể 2,3,4 lớp .

Có 2 kiểu Mainboard :

• Kiểu AT:

Những kiểu cũ có kích thước 12” x 13” hay 30cm x 32,5 cm . Về sau giảm xuống còn 8,5” x 11” hay 21,5cm x 28cm tương đương khổ giấy A4 gọi là bo mạch Baby/AT

Kiểu bo này hiện nay còn dùng nhiều có cấu hình hỗ trợ cho CPU 486 và sau đó từ Pentium 75 trở lên đến Pentium 200 .Phần lớn chúng giống nhau , chỉ thay đổi chút ít và đều có sẵn phần điều khiển EIDE và I/O. Bo mạch này hỗ trợ cho Pentium Pro 150 180 và 200 , còn Pentium II thì đã chuyển qua kiểu ATX

• Kiểu ATX :

Kiểu này hiện nay đã trở thành tiêu chuẩn cấu trúc cho bo mạch . Cấu trúc của nó được thiết kế với xu hướng đơn giản và tiện lợi để cho người dùng có thể sử dụng thiết bị hay phụ tùng của các hãng sản xuất khác nhau.Hình dáng bo mạch này khác và xoay ngang 90^0 so với hướng kiểu bo PC/AT và có những cải tiến tiện lợi như sau:

* CPU tuy đã có bộ phận toả nhiệt(heat - sink) nhưng lại nằm ngay dưới quạt của bộ nguồn lợi dụng quạt của bộ nguồn để làm mát cho CPU

* Rãnh PCI và ISA nằm thấp xuống dưới và xa CPU để dễ gắn card giao tiếp nhất là những loại có chiều dài bất thường như sound card , card video, card TV, card giải mã hình và âm thanh cho DVD,... mà không bị vướng mắc

* Chức năng kiểm soát giao tiếp có sẵn (built-in inteface):

- Chức năng điều khiển ổ đĩa mềm (ở bo nào cũng có).
- Chức năng điều khiển EIDE
- Chức năng điều khiển SCSI. Những bo mạch có sẵn chức năng SCSI thường là SCSI3
- Nếu có sẵn tính năng âm thanh trên bo mạch ta thấy có thêm :
    +một đầu nối dương (connector) 4 hay 3 chân (pin) để nhận âm thanh từ CD
    + Một cổng ra loa (speaker out)
    + Một cổng ra (output) cho thiết bị âm thanh ngoại vi
    + Một cổng vào cho micro
- Một cổng vào chỉ dùng được cho chuột PS/2
- Một cổng vào cho bàn phím PS/2
- Hai cổng ra USB (Universal Serial Bus= Cổng nối tiếp đa năng).Loại cổng này trong tương lai sẽ thay thế các cổng nối tiếp ,song song,bàn phím,chuột và những thiết bị mới khác .
- Một cổng ra song song dùng cho máy in và các thiết bị khác

- Hai cổng ra nối tiếp COM1 và COM 2

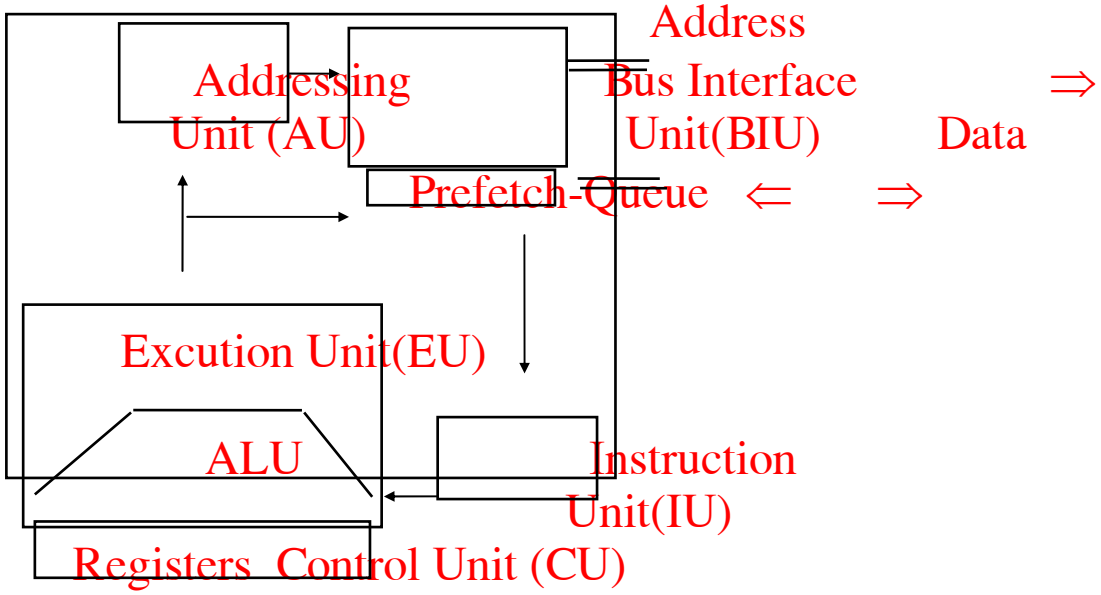
Trên thực tế còn tồn tại những loại những loại bo mạch không chuẩn của các hãng sản xuất máy nhái .

BỘ VI XỬ LÝ

Nếu bộ nguồn là trái tim của máy vi tính thì bộ vi xử lý chính là khối óc của nó . Bộ vi xử lý được phát triển trên công nghệ chế tạo các mạch vi điện tử có độ tích hợp rất lớn VLSI (Very Large Scale Integration) với các phần tử cơ bản là các tranzixtor trường MOS có độ tiêu hao công suất rất nhỏ .

Trong họ 80x86 :(8086,80186,80286,80386,80486,Pentium,Pentium I,II,III...)chúng thực hiện tất cả các hoạt động xử lý logic và số học. Nói chung bộ vi xử lý đọc số liệu từ bộ nhớ, xử lý nó theo cách được xác định bởi lệnh , cuối cùng cất kết quả vào bộ nhớ.

1/Cấu trúc chung:



PQ(Prefetch Queue) : Hàng đợi nhận trước
BIU(Bus Interface Unit): Đơn vị ghép nối Bus
IU (Instruction Unit) : Đơn vị lệnh
EU (Excution Unit) : Đơn vị thực hiện lệnh .EU gồm có:
ALU(Arithmetical Logical Unit) : Bộ tính số học
CU(Control Unit) : Bộ điều khiển
Registers : Các thanh ghi

EU duy trì trạng thái CPU ,Kiểm soát các thanh ghi đa năng và toán hạng lệnh.Các thanh ghi và đường truyền dữ liệu trong EU dài 16 bit (Với các loại mới có thể là 32 hoặc 64 bit).
BIU thực hiện tất cả các tác vụ về Bus cho EU ; Nó thiết lập khâu nối với thế giới bên ngoài là các bus số liệu , địa chỉ và điều khiển . Dữ liệu được truyền giữa CPU và bộ nhớ hoặc thiết bị I/O khi có yêu cầu từ EU . Việc truyền này không trực tiếp mà qua 1 vùng nhớ RAM có dung lượng nhỏ ở BIU gọi là PQ(Prefetch Queue):Hàng đợi nhận trước. sau đó được truyền vào IU . Tiếp đó IU sẽ điều khiển EU để cho lệnh đó được thực hiện bởi ALU.

Một chu kỳ lệnh của CPU có thể được coi đơn giản gồm 2 thời khoảng : Lấy lệnh từ bộ nhớ và thực hiện lệnh .PQ có thể có từ 4 đến 6 byte. Trong khi EU đang thực hiện lệnh trước thì BIU đã tìm và lấy lệnh sau vào CPU từ bộ nhớ và lưu trữ lệnh đó ở PQ rồi .Hai khối thực hiện lệnh và ghép nối bus BIU có thể làm việc độc lập với nhau và trong hầu hết các trường hợp có sự trùng lặp giữa thời gian thực hiện lệnh trước và lấy lệnh sau. Như vậy thời gian lấy lệnh có thể coi như bằng 0 vì EU chỉ thực hiện lệnh đã có sẵn trong PQ do BIU lấy vào. Điều này đã làm tăng tốc độ xử lý chung của máy tính.

2/Các thanh ghi của họ 80x86:

Thanh ghi thực ra là 1 bộ nhớ được cấy ngay trong CPU .Vì tốc độ truy cập các
thanh ghi nhanh hơn là với bộ nhớ chính RAM nên nó được dùng để lưu trữ các
dữ liệu tạm thời cho các quá trình tính toán,xử lý của CPU
Bộ nhớ được chia thành các vùng (đoạn) khác nhau :

- Vùng chứa mã chương trình (Code segment)
- Vùng chứa dữ liệu và kết quả trung gian của chương trình (Data segment)
- Vùng ngăn xếp (stack) để quản lý các thông số của bộ vi xử lý khi gọi chương trình con hoặc trở về từ chương trình con.(Stack segment)
- Vùng dữ liệu phụ (Extra segment)

Các thanh ghi đoạn 16 bit chỉ ra địa chỉ đầu (segment) của 4 đoạn trong bộ nhớ. Nội dung các thanh ghi đoạn xác định địa chỉ của ô nhớ nằm ở đầu đoạn(địa chỉ cơ sở) .

Địa chỉ của các ô nhớ khác nằm trong đoạn tính được bằng cách cộng thêm vào địa chỉ cơ sở 1 giá trị gọi là địa chỉ lệch (ofset)

Các thanh ghi của họ 80x86 như sau:

Thanh ghi con trỏ lệnh IP
Các thanh ghi dữ liệu: AX,BX,CX,DX
Các thanh ghi con trỏ,chỉ số: SP,BP,SI,DI
Các thanh ghi đoạn :CS,DS,SS,ES
Thanh ghi cờ

Số lượng các thanh ghi và độ lớn của chúng trong các bộ CPU hiện đạI ngày càng
được tăng lên cũng là 1 yếu tố làm cho các bộ vi xử lý này hoạt động nhanh hơn.

Dung lượng các thanh ghi trong 1 số vi xử lý hiện đạI:

Từ máy 386 các thanh ghi đa năng và thanh ghi cờ có độ lớn gấp đôI (32 bit)

Các thanh ghi đoạn (6 thanh ghi) độ lớn vẫn là 16 bit

3/ Bộ nhớ ẩn trong vi xử lý :

Cơ chế bộ nhớ ẩn đã làm cho các CPU hoạt động nhanh hơn ,hiệu quả
hơn ,chính vì vậy các CPU hiện đạI ngày nay đều có bộ nhớ ẩn (Cache).Dung lượng của bộ nhớ ẩn cũng ngày càng lớn hơn. Nguyên tắc hoạt động của bộ nhớ ẩn như thế nào xin xem tiếp mục 6 của phần tiếp sau .

4/ Một số cải tiến mới nhất trong kỹ thuật vi xử lý của 1 số hãng sản xuất:

Tính đến thời điểm này (8/1999) kỹ thuật vi xử lý đã có thêm 1 số thành tựu sau:

• Hạ thấp điện áp nuôi chip vi xử lý:

Các bộ vi xử lý Pentium Pro và Power PC thế hệ hiện nay đều dùng công nghệ CMOS(Công nghệ đơn cực sử dụng các cặp MOSFET kênh n và kênh p ở chế độ tải tích cực) với kích thước đặc trưng 0,35 micron (xấp xỉ kích thước của mỗi tranzixtor và các đường dẫn kim loại nối chúng). Các phiên bản sau của chúng sẽ rút xuống kích thước 0.25 micron.

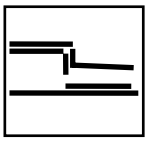
Khi giảm nhỏ kích thước thì công suất điện tiêu thụ(nhiệt lượng toả ra) trên mỗi đơn vị diện tích tăng lên theo quy luật bình phương . May mắn thay 1 đặc tính khác của công nghệ CMOS đã cứu nguy cho vấn đề này :điện áp và công suất tiêu thụ của tranzistor cũng quan hệ với nhau theo quy luật bình phương .Điều này có nghĩa là sự giảm nhỏ điện áp cung cấp sẽ bù lại việc tăng công suất tiêu thụ . Hạ điện áp hoạt động từ 5V xuống 2V sẽ tiết kiệm công suất 6 lần (25/4) ;hạ xuống 1V sẽ giảm nhỏ sự tiêu hao công suất 25 lần(25/1).

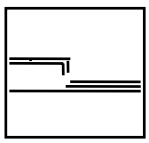
Đó chính là lý do tại sao các nhà thiết kế chip hạ thấp điện áp nuôi từ 5V xuống 3,3V rồi 2,8V và 2,5V thậm chí 1,8V đối với các chíp ở thế hệ kế tiếp

• Vấn đề **"thay đồng bằng nhôm " :**

Cùng thời gian(9/1997) khi mà Intel công bố bộ nhớ tế bào đa áp (Chúng ta sẽ khảo sát chúng ở phần sau "Bộ nhớ máy tính") thì IBM đã công bố quy trình chế tạo mới dùng đồng để tạo ra chip CPU . Họ đã giải quyết được các bế tắc trong việc mạ kim loại đồng cho quá trình CMOS 7S mới của họ .Trước đây các chip thường được dùng nhôm làm các mối dẫn .Nhưng khi thu nhỏ kích thước dưới 0,35micron điện trở của nhôm gây cản trở tốc độ - sự chuyển mạch tức thời không thể thực hiện trên đường tốc độ thấp . Đồng có điện trở thấp hơn , rõ ràng là vậy ; nhưng đồng thường gây nhiễm bẩn silic và vì thế sẽ làm hỏng các tranzistor của chip . IBM giải quyết vấn đề nhiễm bẩn bằng cách tách biệt mạch đồng với silic sau đó bọc mạch đồng lại . Quá trình thực hiện tích hợp 6 lớp đồng kích thước 0,2 micron để gắn vào silic .

So sánh kích thước giữa các đường dẫn trong các loại chip sử dụng đồng và nhôm





386 : 1,5micron (dùng nhôm)

PentiumII : 0,35micron (dùng nhôm)

IBM copper : 0,18micron (dùng đồng)

Một số số liệu

Vi xử lý	Bề rộng thanh ghi	bus địa chỉ	Bus số liệu	Không gian địa chỉ	Tổng số đồng hồ cực đạI
8086	16 bit	20 bit	16 bit	1 MB	10MHz
80286	16 bit	24 bit	16 bit	16MB	16MHz
80386DX	32 bit	32 bit	32 bit	4 GB	40MHz
80486SX	32 bit	32 bit	32 bit	4 GB	25MHz
Pentium	32 bit	32 bit	64 bit	4 GB	400MHz
Pentium “Merced”					800MHz

Trên thị trường máy tính Việt Nam hiện nay sử dụng nhiều loại chip của các hãng khác nhau : Intel , AMD , Centaur (Winchip),Cyrix. Giá thành của các chip AMD , Centaur,Cyrix thường rẻ hơn Intel 20% - 30% với tính năng cơ bản không thua kém gì vì vậy chúng có mặt rất nhiều trong các máy trong thực tế với tỷ lệ % tương đương Intel ; mặc dù tổng thể trên toàn thế giới Intel chiếm thị phần trên 80%.

BỘ NHỚ MÁY TÍNH

1/Khái niệm hoạt động của máy tính và vai trò của bộ nhớ trong hoạt động đó :

Nhìn vào 1 cách cụ thể ta thấy công việc của máy tính có nhiều loại :

- Dạng đơn giản hay gặp :soạn thảo,trò chơi,làm việc với môi trường NC...

Khi ta vào 1 môi trường soạn thảo,chơi 1 trò chơi,hay làm việc với NC ...đó chính là

khi máy tính đang thực hiện các chương trình .

- Tổng quát công việc của máy tính là gì ?

Đó là 1 công việc lặp đI lặp lại :

+ Nhận lệnh
+ Giải mã lệnh
+ Thực hiện lệnh

Quá trình lặp này cứ tiến hành liên tục cho đến khi có 1 lệnh mới (tức có 1 tác

động mới của con người vào quá trình).

- Các lệnh nằm ở đâu ?

Chương trình máy tính là 1 tập hợp các lệnh theo 1 trình tự nhất định do con

người nghĩ ra.

Ví dụ: +" Cộng 2 với 4 "
+" Hiển thị kết quả ra màn hình "
+" Vẽ 1 tàu vũ trụ trên bầu trời sao "...

Các chương trình được chia làm 2 loại :

+ Chương trình hệ thống : Các chương trình điều khiển của hệ điều

hành ,chương trình điều khiển thiết bị ngoại vi chuẩn...

+ Chương trình ứng dụng : Các chương trình này thường được lưu trữ trong bộ nhớ ngoài . Khi chạy mới đưa vào bộ nhớ trong (RAM)

Ví dụ : Ta chạy chương trình Tuvi.exe tức là :

Khi nhận lệnh Tuvi.exe ↵

Vi xử lý sẽ :

- Đọc vào bộ nhớ chương trình Tuvi.exe
- Đọc các dòng lệnh của Tuvi.exe
- Giải mã các lệnh này
- Thực hiện các lệnh

Như vậy : ***Chương trình và dữ liệu được nạp vào bộ nhớ trước khi thực hiện*** .

- Bộ nhớ do các IC nhớ tạo thành .Mỗi IC có 1 dung lượng nhớ nhất định.
- Tổng dung lượng nhớ của các IC nhớ là dung lượng bộ nhớ.
- Nếu dung lượng bộ nhớ nhỏ,chương trình ứng dụng lớn sẽ không chạy được

Ví dụ : Windows 3.11 cần tối thiểu 4 MB bộ nhớ

Windows 98 cần tối thiểu 16MB bộ nhớ

2/ Khả năng quản lý bộ nhớ của 1 bộ vi xử lý :

Phụ thuộc vào số chân địa chỉ của vi xử lý (số bít địa chỉ)

8086 có 20 bit địa chỉ → có khả năng phân biệt 2^{20} ô nhớ = 1MB

8386 có 32 bit địa chỉ → có khả năng phân biệt 2^{32} ô nhớ = 4GB

8486 có 32 bit địa chỉ → có khả năng phân biệt 2^{32} ô nhớ = 4GB

Pentium có 32 bit địa chỉ → có khả năng phân biệt 2^{32} ô nhớ = 4GB

Pentium Pro150 có 36 bit địa chỉ → có khả năng phân biệt 2^{36} ô nhớ = 64GB

Pentium Pro 200 có 36 bit địa chỉ → có khả năng phân biệt 2^{36} ô nhớ =64GB

Mặc dù có thể cắm thêm nhiều vi mạch nhớ vào máy, nhưng trong thực tế người ta cũng chỉ thường dùng đến 128 MB nhớ trở về trong các ứng dụng thông thường.

3/Các đặc trưng kỹ thuật cơ bản của bộ nhớ bán dẫn:

-Dung lượng
- Tốc độ hoạt động (truy nhập)
- Độ tin cậy sử dụng
- Giá thành , kích thước.

4/ Bộ nhớ RAM (Random Access Memory) :

- Bộ nhớ RAM giống như 1 caí bảng mà người ta có thể viết vào và sau đó lại có thể xoá đI để viết các thông tin mới
- Hai loại RAM
  + RAM tĩnh :
    Dùng phần tử triger làm phần tử nhớ
    Tốc độ truy nhập nhanh.Giá thành đắt
  + RAM động:
    Dùng tụ điện làm phần tử nhớ
    Tốc độ truy nhập không nhanh
    Luôn phải "làm tươi" thông tin
    Giá thành rẻ

- Trong máy tính các IC nhớ RAM thường được ghép thành các khối nhớ 1MB,4MB,8MB,16MB... để cắm vào máy cho tiện lợi.
- Hai loại modun nhớ RAM:
  • SIMM (Single Inline Memory Modules): Môdul nhớ 1 hàng chân
    Có loại 30 chân : Dùng cho các loại máy cũ như máy 386
    Có loại 72 chân : Dùng cho các loại máy cũ như máy 486,Pentium

Hình dáng bên ngoài và sơ đồ mạch điện , tên các chân của 1 khối SIMM như sau:

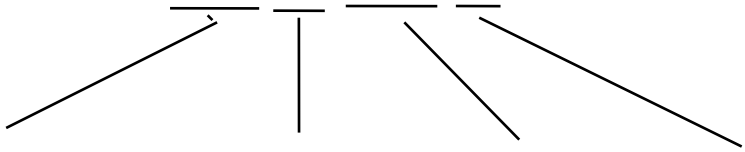
page 282

• DIMM (Dual Inline Memory Modules): Môdul nhớ 2 hàng chân.

Dùng cho các loại máy 486,Pentium,các loại đời mới hiện nay...

- Ký hiệu của 1 IC nhớ :

WWW XX YYYY ZZ



Mã sản phẩm Kiểu bộ nhớ Dung lượng Tốc độ truy nhập

EPROM : 27 Kilôbit nanô giây
SRAM : 62,7C
DRAM : 41,51

Ví dụ : 7C1024 - 15

là SRAM ,128KB , tốc độ truy nhập 15 nanô giây

Ta thấy 1 IC nhớ có dung lượng 128KB vậy

muốn có 512KB phải cần 4 IC ghép với nhau

muốn có 1MB phải cần 8 IC ghép với nhau...

Các loại RAM mới được sử dụng trong thời gian gần đây:

Hiện nay trong các lý lịch kỹ thuật máy , trong các bài khảo cứu chuyên ngành

máy tính ... thường có nhắc đến 1 số các danh từ về RAM . Đây là các vấn đề mới

cần cập nhật:

- VRAM (Video Random Access Memory): Bộ nhhớ truy nhập ngẫu nhiên video

và cùng họ với nó :WRAM (Windows RAM)cho độ rộng dải hơn .

Thuộc loại bộ nhớ 2 cổng (Dual - ported memory) .Đây là bộ nhớ RAM có

cổng trước ,cổng sau . Dữ liệu có thể đI vào cửa trước rồi đI ra trực tiếp cửa sau

nên có tốc độ cao hơn .

- EDOVRAM : Là dạng tốc độ nhanh của VRAM
- EDODRAM : Là dạng tốc độ nhanh của DRAM
- SDRAM (Synchronous DRAW = DRAW đồng bộ): DRAW là 1 loại RAM gia

tốc cho Windows :

- SGRAM (Synchronous Graphics RAM = RAM đồ hoạ đồng bộ)
- EDRAM (Enhanced DRAW = DRAW cải tiến)
- RAMDAC : Đây là loại chuyển đổi Digital - Analog dùng RAM

Trong tương lai ; cũng như với các chip vi xử lý danh sách này sẽ còn kéo dài nữa...

5/ Bộ nhớ ROM: (Read Only Memory)

-Là bộ nhớ vẫn giữ được thông tin sau khi cắt điện nuôi vi mạch

-Dung lượng của IC nhớ loại này thường nhỏ. Chương trình được nạp vào trong ROM bằng thiết bị chuyên dùng.Một thiết bị nạp,xoá ROM mức trung bình có giá khoảng hơn 500$. Một vi mạch ROM trắng(Loại EPROM: ghi được nhiều lần) dung lượng 512KB có giá khoảng 3$.

- Bộ nhớ PROM (Programable Read Only Memory): Ghi được 1 lần.
- Bộ nhớ EPROM(Erasable Programable ROM) : Ghi được nhiều lần.
- Bộ nhớ Flash ROM : Là loại ROM có thể thay đổi được nội dung trực tiếp từ máy

tính mà không cần có thiết bị ghi đặc biệt nào và cũng không cần xoá bằng tia cực

tím

Hầu hết các mainboard đời mới đều dùng Flash ROM để chứa BIOS,nhờ đó giúp người dùng cập nhật version mới được dễ dàng.Tuỳ theo hãng nào sản xuất , Flash ROM dùng 1 trong 2 mức điện áp làm việc là +5V hay +12V . Ta chỉ cần có phần mềm ghi Flash ROM (Của hãng tạo ra BIOS như Award ,AMI ...)rồi dùng nó

để cập nhật ROM BIOS. Chương trình này chỉ được sử dụng khi thật cần thiết.

-Ký hiệu của vi mạch : 27xxx ; 3 số sau chỉ dung lượng của ROM (KB)

2708(1KB x 8) : 8KB

27256(32K x 8) ;256KB

27512(64K x 8): 512KB

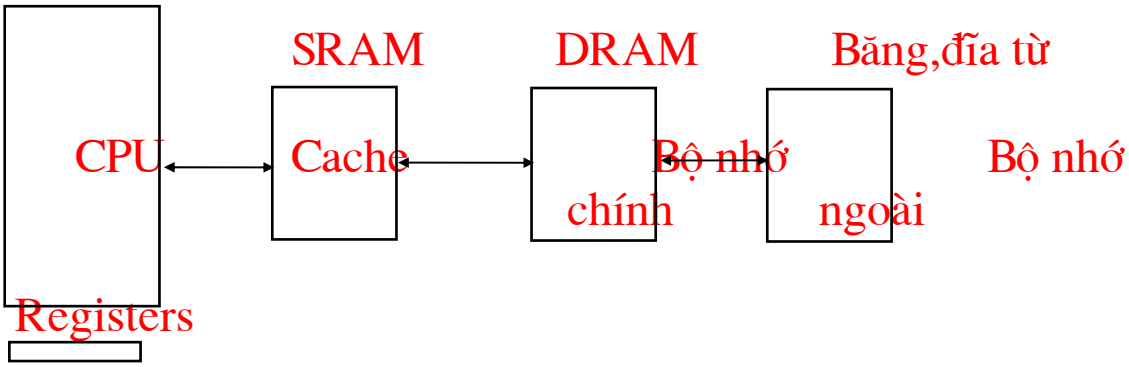
- Kích thước vật lý của các vi mạch ROM đều bằng nhau.

6/ Bộ nhớ tế bào đa áp :

Từ tháng 9 năm 1997 Intel đã công bố bộ nhớ StrataFlash đây là loại bộ nhớ đặc biệt dạng tế bào đa áp (multilevel-cell) có khả năng lưu giữ nội dung ngay cả khi tắt thiết bị . Thay vì phải xác định 1 hay 2 mức điện áp khả dĩ cho tế bào nhớ StrataFlash sẽ gán 1 trong 4 mức điện áp . Các tế bào StrataFlash sẽ có 4 mức điện áp : 2 cho trạng thái mở và 2 cho trạng thái tắt . Nhờ vậy mỗi tế bào có thể lưu dữ liệu gấp 2 lần loại chip nhớ flash thông thường (Loại ROM có thể ghi xoá bằng máy tính không cần thiết bị đặc biệt) . Loại này được sử dụng rộng rãi trong các máy ảnh số, máy tính cầm tay ,điện thoại di động ,các máy trả lời tự động . Tuy nhiên hiện nay tuổi thọ của loại này còn tương đối thấp : Số chu kỳ xoá là 10.000 lần so với 100.000 lần của các loại chip Flash thông thường . Với 1 máy ảnh số 10.000 chu kỳ xoá đủ để chụp 240.000 pô hình trên máy ảnh (Tương đương khoảng 6500 cuộn phim thông thường) . Tương lai của loại bộ nhớ này rất sáng sủa .

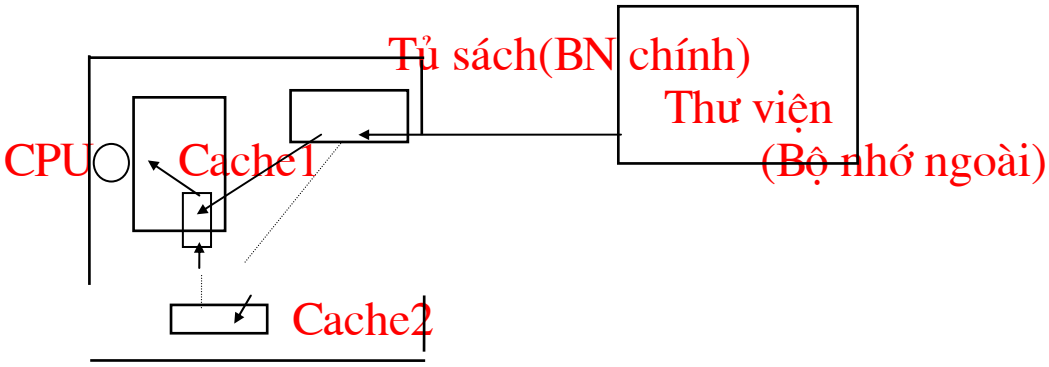
7/ Bộ nhớ ẩn trong vi xử lý :

a-Bộ nhớ ẩn:



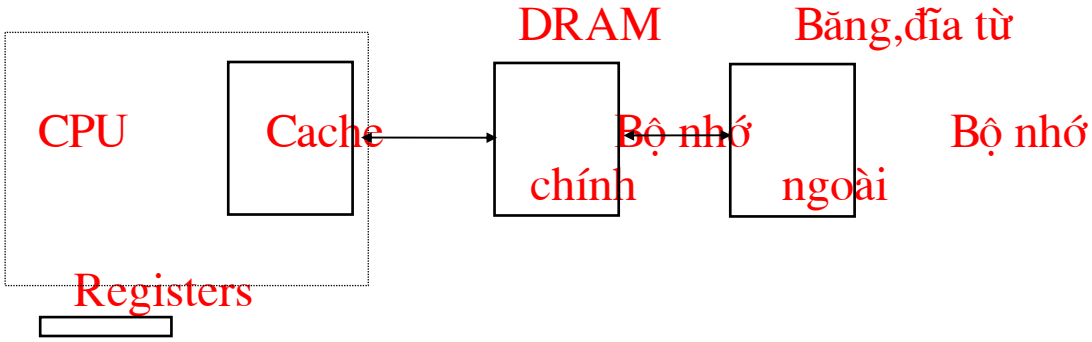
Khi CPU làm việc với 1 đối tượng ,thông tin (dữ liệu,lệnh) của đối tượng và các thông tin lân cận sẽ được đưa vào Cache .Khi CPU xử lý nó sẽ tìm thông tin ở Cache . Nếu không có nó sẽ tìm ở bộ nhớ chính ,khi copy thông tin vào Cache nó sẽ copy luôn cả các thông tin lân cận vào Cache . Nếu thông tin không có ở bộ nhớ chính thì nó sẽ tìm ở bộ nhớ phụ và khi copy nó cũng sẽ copy luôn cả các thông tin lân cận vào bộ nhớ chính để dự phòng cho các lần tiếp theo của CPU .

Ta có thể so sánh cơ chế này với mô hình mượn sách từ thư viện như sau:

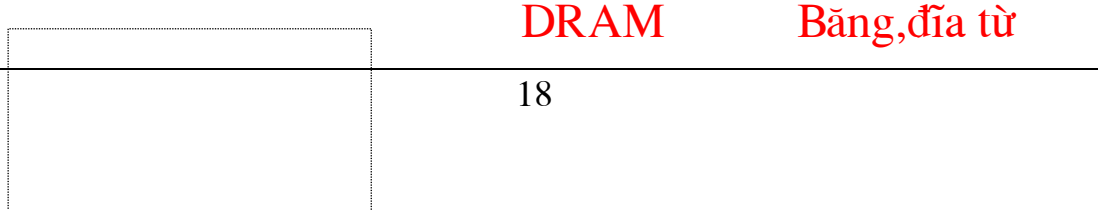


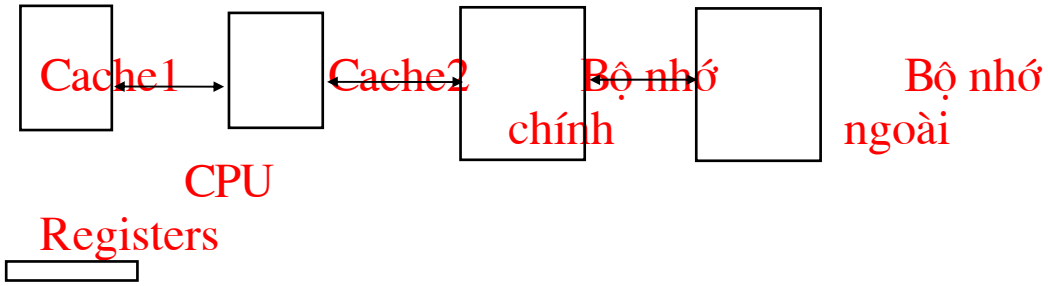
Người đọc (CPU) sẽ nhanh chóng tìm được các quyển sách cần thiết liên quan đến vấn đề anh ta đang quan tâm vì khi mượn sách từ thư viện về đưa vào tủ anh đã mượn 1số các quyển có nội dung liên quan đến nhau . Và khi lấy từ tủ ra để lên bàn trước mặt cũng theo nguyên tắc đó (Tất nhiên số sách bây giờ ít hơn,việc tìm kiếm càng nhanh hơn) .

b-Bộ nhớ ẩn được đưa vào trong vi xử lý :



c-Bộ nhớ ẩn được chia làm 2 (Cache1 và Cach2):

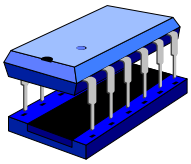




d- Dung lượng bộ nhớ ẩn trong 1 số vi xử lý hiện đạI:

Cơ chế bộ nhớ ẩn giúp đã làm cho các CPU hoạt động nhanh hơn ,hiệu quả hơn ,chính vì vậy các CPU hiện đạI ngày nay đều có bộ nhớ ẩn (Cache).Dung lượng của bộ nhớ ẩn cũng ngày càng lớn hơn :

80486DX	8KB
80486SX	8KB
80486DX2	8KB
80486DX4	16KB
Pentium 100	16KB
Pentium 200	512KB
Pentium II (L2)	512KB
Pentium Pro(L2)	512KB (Tuỳ chọn 256 hoặc 512)
Pentium MMX(L2)	512KB (Tuỳ chọn 256 hoặc 512)
Pentium Celeron(L2)	512KB (Được bổ xung vào 4/1998)
Pentium IIXeon(L2)	1MB



ĐĨA MỀM VÀ Ổ ĐĨA MỀM

1/Khái niệm:

- Maý tính làm việc dựa trên sự hoạt động của các chương trình chứa trong bộ nhớ.
- Người ta chia bộ nhớ làm 2 loại :
  +Bộ nhớ trong
  + Bộ nhớ ngoài

Đĩa mềm là 1 dạng bộ nhớ ngoài.

- Điểm lại sự phát triển của bộ nhớ ngoài:

Bắt đầu là các loại băng đục lỗ,bìa đục lỗ → xuyến từ → ổ đĩa mềm → ổ đĩa cứng → ổ zip → ổ CD ROM → DVD ROM → DVD RW.

- Đĩa mềm dã đến ngày tận số ? Việc nghiên cứu đĩa mềm có còn ý nghĩa nữa không ?

Việc sử dụng đĩa mềm ngày nay đã hạn chế .Tuy vậy người ta vẫn chưa bỏ hẳn đĩa mềm vì dùng nó để lưu trữ ,vận chuyển các lượng thông tin nhỏ vẫn tiện lợi,giá thành rẻ.

Việc nghiên cứu hoạt động của đĩa mềm và ổ đĩa mềm vẫn rất có ý nghĩa để tạo tiền đề cho việc tiếp thu kiến thức về hoạt động của các loại đĩa khác.

Xin trích dẫn đoạn văn sau từ tạp chí US-PCWold :

" Những báo cáo về sự lụi tàn của đĩa mềm đã cường điệu quá mức.Hàng năm ,khi các nhà sản xuất cho ra đời những phương tiện lưu trữ mới với dung lượng lớn , người ta lại viết hàng loạt bài cáo phó . Nhưng rồi chiếc đĩa mềm 3,5 inch đáng kính vẫn cứ tồn tại ,giống như 1 con mèo già lắm mưu nhiều mẹo sống dai hơn người ta tưởng ..."

2/ Nguyên lý ghi_đọc từ:

- Gồm 2 thàn phần chính :
  + Đầu từ: Là 1 lõi ferit hình xuyến ,có khe từ.Trên lõi có quấn cuộn dây điện từ. Các đầu ra của cuộn dây nối vào mạch thu_phát thông tin

Hình vẽ

+ Đĩa từ : Là đĩa nhựa dẻo ,trên bề mặt có phủ 1 lớp bột từ có đặc tính lưu giữ từ

Hình vẽ

- Hoạt động :

* Ghi : Thông tin cần ghi vào đĩa ở dạng 0-1 được biến đổi thành tín hiệu điện

(Ví dụ theo chuẩn TTL : 0 : 0..+0,8 Volt

1 : +2,8..+5Volt)

Các tín hiệu điện 0-1 này chạy trên cuộn dây đầu từ sẽ tạo ra từ trường tỉ lệ với 0-1

Trong khi đĩa từ quay và ở các vị trí khác nhau của đĩa sẽ được lưu giữ các phần đĩa nhiễm từ tỉ lệ với 0-1 khác nhau.

*Đọc: Ngược với quá trình ghi

3/Cấu tạo của đĩa từ 1.44MB:

- Là đĩa bằng nhựa dẻo ,ở giữa gắn 1 đĩa nhỏ hơn bằng sắt có khoét lỗ để trục motơ
kéo đĩa chuyển động (quay).
- Kích thước đĩa : $3^{1/2}$ “
- Đĩa được đặt trong 1 hộp nhựa vuông mỏng.

a- Tổ chức vật lý :

Một đĩa mềm được chia thành các đơn vị vật lý:

- Rãnh từ (Track): Là các vùng đường tròn đồng tâm mà dữ liệu được ghi trên đó.

Với đĩa 1.44MB có 80 Track từ ngoài vào trong

- Cung từ (Sector):Mỗi Track được chia làm nhiều cung từ (Sector).

Số Sector/ 1 Track tuỳ theo cách định dạng đĩa (format).

Cùng 1 đĩa $3^{1/2}$ “ nếu format 1,44 MB có 18 Sector

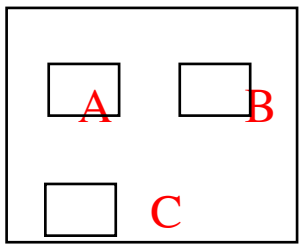
nếu format 1,66 MB có 20 Sector

nếu format 2,88 MB có 36 Sector

b- Tổ chức thông tin :

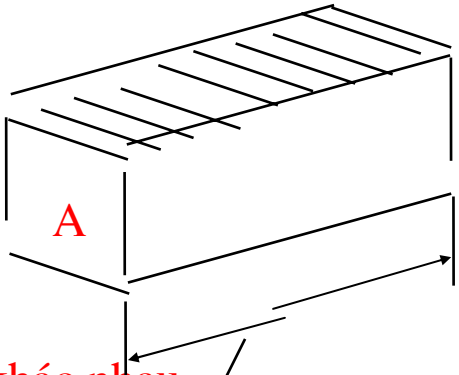
- Khaí niệm về tệp thông tin(File):

File là 1 tập hợp dữ liệu có liên quan với nhau và có cùng kiểu được nhóm lại với nhau tạo thành 1 dãy được chứa trong thiết bị nhớ ngoài.



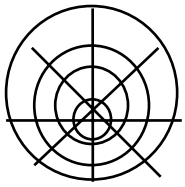
Tủ phiếu = 1 đĩa

Các ngăn kéo A,B,C= các file



Các file có độ lớn khác nhau đựng số bìa khác nhau ; tức cần nhiều it secto khác nhau

- Trong đĩa mềm người ta lưu trữ thông tin dưới dạng gắn với các đặc tính hình tròn



- Tổ chức lưu giữ thông tin :
  + Boot Sector : Chiếm 1 sector
  + 2 bảng FAT: Chiếm 2x9=18 sector
  + Bảng thư mục : Chiếm 14 sector
  + Còn lại là vùng dữ liệu

/* Có thể dùng chương trình DISKEDIT.EXE để xem Boot Sector,bảng FAT, Bảng thư mục*/

• Boot Sector : Nằm ở Sector vật lý 1 , mặt 0, Track 0

Trên Boot Sector có 1 chương trình khởi động .Nếu đĩa mềm là đĩa khởi động thì khi khởi động ,chương trình này sẽ nạp các chương trình hệ điều hành vào bộ nhớ. Trên Sector này còn có bảng thông số đĩa.

• Bảng FAT (File Allocation Table)

Là 1 bảng danh sách móc nối mà DOS sử dụng để theo dõi sát các vị trí vật lý của dữ liệu trên đĩa và để sắp đặt các chỗ còn trống để lưu giữ các file mới.

DOS cấp phát cho file các trang ,FAT lưu giữ bản đồ các trang.Nếu đĩa bị hỏng FAT thì không truy nhập được thông tin nữa; mặc dù chúng vẫn tồn tại trên đĩa.

• Bảng thư mục :

Chiếm 14 Sector .Bảng này lưu trữ danh sách các file đang lưu trên đĩa

- Cách ghi thông tin trên đĩa : DOS ghi hết rãnh của mặt này rồi mới tiếp tục rãnh của mặt kia.
- DOS đọc 1 file như thế nào ?

Để xem FAT được tổ chức ra sao chúng ta hãy lấy 1 ví dụ về việc DOS sử dụng

FAT để đọc 1 file như thế nào.

1- DOS nhận số hiệu cluster đầu tiên từ thư mục ,giả sử đó là 2 .

2- DOS đọc cluster từ đĩa và chứa nó trong 1 vùng nhớ gọi là vùng chuyển dữ liệu (Data Trannsfer Area -DTA),chương trình thực hiện việc đọc sẽ nhận dữ liệu từ DTA khi cần

3- Vì điểm nhập thứ 2 chứa giá trị 4 , cluster tiếp theo của file có số hiệu là 4. Nếu chương trình cần thêm dữ liệu DOS sẽ đọc cluster vào DTA

4- Điểm nhập 4 trong FAT chứa giá trị FFFh , giá trị này chỉ ra rằng đó là cluster cuối cùng trong file. Tóm lại quá trình lấy số hiệu cluster trong FAT là liên tục đọc dữ liệu vào DTA cho đến khi điểm nhập trong FAT chứa giá trị FFFh.

Điểm nhập 0 1 2 3 4 5 6 7 8 9

Trong hình ta cũng thấy có 1 file chiếm các cluster 3,5,6,7 và 8

- DOS lưu trữ các file như thế nào ?

Để lưu trữ các file DOS thực hiện các công việc sau đây:

1-DOS xác định 1 điểm nhập chưa sử dụng trong thư mục và lưu vào đó tên file,thuộc tính file ,ngày giờ tạo lập.

2-DOS tìm trong bảng FAT điểm nhập đầu tiên đánh dấu 1 cluster chưa sử dụng (giá trị 000 có nghĩa là cluster chưa sử dụng) và chứa số hiệu cluster đầu tiên của tập tin lấy trong thư mục vào đó .Chúng ta giả sử nó tìm thấy giá trị 000 ở điểm nhập 9.

3- Nếu dữ liệu chứa vừa trong 1 cluster .DOS chứa nó trong cluster 9 và đặt giá trị FFF vào điểm nhập thứ 9 của FAT . Nếu vẫn còn dữ liệu DOS tiếp tục tìm cluster chưa được sử dụng tiếp theo trong FAT .Ví dụ nó tìm thấy điểm nhập Ah ,nó

sẽ lưu dữ liệu vào cluster Ah và đặt giá trị 00A vào điểm nhập 9 của FAT.Quá trình tìm các cluster chưa được sử dụng trong FAT chứa dữ liệu vào đó ,cho điểm nhập của FAT trỏ tới cluster tiếp theo sẽ tiếp tục cho đến khi dữ liệu được lưu trữ hết .Điểm nhập cuối cùng của file trong FAT sẽ chứa giá trị FFFh

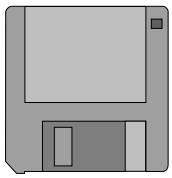
4/Cấu tạo ổ đĩa từ 1,44 MB:

Hình vẽ

Khi CPU đọc/viết các số liệu ,đĩa được quay bởi 1 motơ điều khiển tốc độ 300vg/phút .Đĩa dùng cả 2 mặt nên có 2 đầu từ đọc/viết . Hai đầu từ được gắn ở đầu cần truy xuất (access arm) .Chuyển động quay của 1 motơ bước (Steeping Motor) sẽ biến thành chuyển động tịnh tiến theo phương bán kính của cần truy xuất qua 1 cơ cấu trục vít xoắn .Do trên đầu từ có cuộn dây cảm ứng nên :

- Khi đọc : Sự biến đổi từ thông qua khe từ của các phần tử thông tin lưu trữ trên đĩa được biến thành điện thế cảm ứng trong cuộn dây trên đầu từ . ở 2 đầu ra cuộn dây tín hiệu số liệu (Data Signal) được tạo ra .
- Khi viết : Cuộn dây sẽ phát ra từ trường qua khe từ để từ hoá các bột từ trên mặt đĩa tạo thành các trạng thái tương ứng với các mức số liệu 0-1 cần lưu trữ.
- ổ đĩa được nối với bộ điều khiển qua dây cáp 34 dây . Với các máy đời mới bộ điều khiển đĩa mềm được làm liền vào bảng mạch chính (Onboard)

- Có thể truy nhập ổ đĩa mềm bằng các ngôn ngữ lập trình thông dụng : C,Pascal,Assembly ,Basic.Địa chỉ cơ sở :3F0h . Thay đổi tốc độ truyền số liệu DMA qua địa chỉ 3F4h



ĐĨA CỨNG

1/Cấu tạo vật lý:

ảnh chụp 1 đĩa cứng đã tháo nắp

Hình vẽ nguyên lý
ổ đĩa cứng

Gồm nhiều đĩa từ bằng kim loại cứng ,được sắp thành 1 chồng theo trục thẳng đứng
đặt trong 1 hộp kim loại kín để tránh bụi .

Mỗi đĩa có 2 đầu từ ở 2 mặt 0 và mặt 1.Khi làm việc đầu từ không tiếp xúc trực tiếp với mặt đĩa như đĩa mềm mà cách 0,0003mm.Tốc độ quay 3600vg/phút(Hiện nay đã có loại quay với tốc độ 7200vg/phút.So với đĩa mềm (300vg/phút) thì tốc độ truy xuất thông tin cao hơn rất nhiều

Đĩa cứng cũng được phân thành các đơn vị vật lý như đĩa mềm , nhưng ở đây có thêm 1 khái niệm nữa là từ trụ (Cylinder)

Cylinder: Vì chồng đĩa cứng có nhiều mặt nên vị trí đầu từ khi di chuyển sẽ tạo thành 1 mặt trụ ,đó là chồng các track sắp nằm lên nhau với 1 vị trí đầu từ.

2/Tổ chức logic:

a-Các khái niệm quan trọng:

- Một đĩa cứng (vật lý thực thể) có thể chia logíc thành nhiều đĩa logic mà DOS gán tên cho chúng từ C → Z
- Mỗi 1 ổ đĩa logic được chia ra từ ổ đĩa vật lý có cấu trúc giống 1 điã mềm :
  - +Boot sector
  - +2 FAT Tables
  - +Directory Table
  - +Data

- Sector Partition là Sector vật lý đầu tiên của đĩa cứng
- Boot Sector (Boot record) là Sector logic của ổ đĩa logic.Một ổ đĩa cứng có thể có nhiều Boot record ứng với nhiều ổ đĩa logic

b-Để sử dụng được 1 đĩa cứng cần phải qua các bước nào ?

- Format cấp thấp đĩa cứng (Low format)
- Phân chia 1 ổ đĩa cứng thành các ổ đĩa logic (fdisk)
- Format cấp cao đĩa cứng (high format)

* Format cấp thấp:

- Đĩa cứng phải được định dạng (Format)cấp thấp trước khi sử dụng . Đó là việc phân định ra những Sector và Cylinder trên đĩa bằng cách viết lên đĩa những thông tin liên quan đến Sector xác định 1 cách rõ ràng từng Sector riêng rẽ được đặt nằm ở đâuvà được đánh

số thứ tự. Những thông tin này được ghi vào 1 vùng Sector ID Header

Vùng này chứa các thông tin:

+Số thứ tự đầu từ

+Số Sector

+Số Cylinder

+Dấu khai báo ID từ đâu

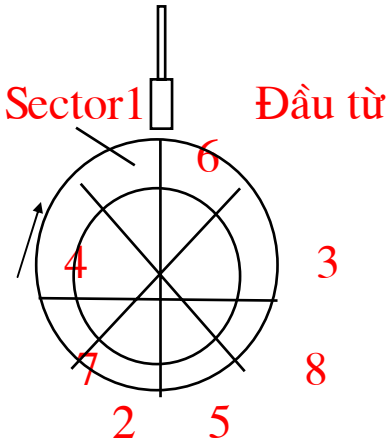
+Ký tự CRC phát hiện sai

Mạch điều khiển đĩa sẽ sử dụng thông tin ID để tìm đến đúng Sector mà nó nhận lệnh phải đến.

- Hệ số đan xen (Interleave):

Làm khớp tốc độ quay của đĩa từ(3600vg/ph=60vg/giây) với tốc độ mà đầu từ có thể xử lý dữ liệu khi chúng qua đầu từ.

Ví dụ



Ví dụ đĩa có hệ số đan xen = 3

Các đĩa cứng sử dụng trên máy 386 đến nay có hệ số đan xen = 1

* Phân chia 1 ổ đĩa cứng thành các ổ đĩa logic (fdisk) :

- Mỗi phân khu được chia thường chiếm trọn 1 số trụ (Cylinder)
- Có 3 loại phân khu trên đĩa cứng :

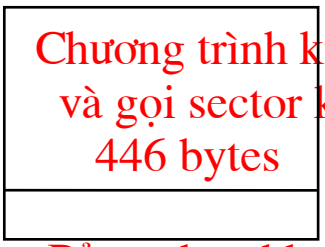
Phân khu chính : Dành cho DOS

Phân khu phi DOS

Phân khu mở rộng : Chia thành nhiều đĩa logic

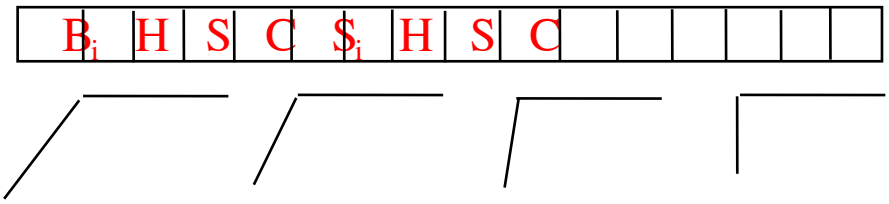
- Sector phân khu : Head 0 , Track 0 , Sector vật lý

Cấu trúc của Sector phân khu(Bảng Partition):



Toàn bộ 512 bytes

Một bảng phân khu có 4 điểm vào .Mỗi điểm vào 16 byte chứa nhhững thông tin mô tả trọn vẹn 1 phân khu:



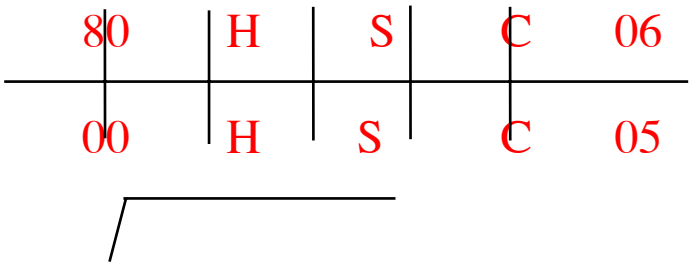
Điểm bắt đầu phân khu | Điểm kết thúc phân khu | Số Sector nằm trước ph.khu này | Số Sector nằm trong 1 phân khu

B_i : Nhận biết phân khu tích cực
- 80 : Active
- 0 : No active

S_i :
- 0: phi DOS
- 1: DOS với bảng FAT 12
- 4 : DOS với bảng FAT 16
- 5 : Phân khu DOS mở rộng
- 6 : Phân khu DOS lớn hơn 32 MB

Ví dụ : 80 00 01 04 06 1F BF 68 80 1F 00 00 60 D8 12 00

Ổ đĩa logic tiếp theo ở phần phân khu mở rộng



Địa chỉ bắt đầu ổ D:\

Việc chia ổ đĩa vật lý thành nhiều ổ đĩa logic do chương trình FDISK thực hiện

c- Boot Sector :

-Dài 512 byte tại Sector logic 0 của ổ đĩa logíc

- Có chứa 1 chương trình khởi động (Boot strap Loader)

- Và bảng thông số đĩa

d- 2 bảng FAT :

e - Thư mục gốc (Root directory): Giống đĩa mềm.

CÁC Ổ ĐĨA QUANG

1-Đĩa CD-ROM :(Compact Disk Read Only Memory)

Đĩa CD được phát minh vào năm 1982 .Các tiêu chuẩn đầu tiên Reed Book do hai hãng SONY và PHILIPS đưa ra . Với sự phát triển kỹ thuật các tiêu chuẩn này cũng thay đổi ;nhưng cơ bản vẫn dựa trên cơ sở Reed Book

Đĩa CD ngày nay không những được sử dụng trong lĩnh vực nghe nhìn mà còn được dùng làm bộ nhớ dung lượng lớn. Sự khác nhau giữa CD Player và CD ROM là CD ROM có thêm bộ ghép nối để truyền số liệu tới bus hệ thống của PC và các linh kiện ghép nối nhằm cho CPU truy nhập các số liệu nhất định với những lệnh phần mềm.

Cấu tạo đĩa :

Đường kính : 4.75 inches

Dày :1,2 mm

Lỗ ở giữa có đường kính :15mm

Dung lượng phổ biến hiện nay :640MB

Đĩa CD có những rãnh phản xạ ánh sáng được phủ bởi bột nhôm và sau đó phủ 1 lớp sơn bóng để bảo vệ .

Khi đĩa CD chế tạo ,thông tin được đưa vào trong đĩa CD dưới các rãnh được phủ nhôm dưới dạng pits (Sự lõm xuống) và lands (Sự lồi lên);những lồi lõm này chính là biểu hiện của các bit . pits và lands được sắp xếp dọc theo đường trôn ốc quanh trục bao phủ toàn bộ bề mặt đĩa CD,lượn vòng từ trong ra ngoài. Không như đĩa hát các loại đĩa CD bắt đầu ghi từ mép trong ra ngoài .

Do có cất tạo đặc biệt nên tốc độ truyền dữ liệu và thời gian thâm nhập của đĩa CD- ROM chưa cao so với đĩa cứng.

Nguyên tắc hoạt động :

hình vẽ

Sensor: Cảm biến Diode Laser: Điốt phát lazer
Beam Spliter: Bộ phân tích tia sáng Bit Signal : Tín hiệu số nhị phân(bit)
Reflected beam : Tia phản xạ Sensing beam:Tia tới
Movable Mirror: Gươngchuyển động Optical Disk :Đĩa quang

Điốt lazer phát ra được hội tụ qua hệ thống quang học hội tụ lên bề mặt đĩa CD-
ROM .Ta đã biết thông tin được ghi bởi các pits và lands .Cường độ tia phản xạ sẽ
yếu đI khi gặp chỗ lõm.Trong ổ đĩa có 1 sensor thu , nhạy với cường độ tia phản xạ

Cường độ tia phản xạ phụ thuộc vào các chỗ lồi lõm mà nó đI qua ,tức là phụ thuộc các thông tin ghi trên đĩa.Đầu ra của sensor là các tín hiệu thông tin đã được chuyển sang dạng điện.

2-ổ đĩa CD-WR (ổ đĩa CD ghi - đọc) :

Là loại ổ đĩa CD ghi lại được .Việc ghi được thực hiện bằng phần mềm trên máy tínhCD-WR cho phép ghi đè dữ liệu cũ , điều này làm cho đĩa CD có thể sử dụng lại gần như đĩa mềm .

Hạn chế : -Tốc độ ghi lại thấp (Thời gian ghi 1 đĩa CD-WR lớn gấp đôI thời gian ghi 1 đĩa CD-ROM thường)

- Giá thành 1 đĩa CD-WR cao hơn khoảng 8 lần 1 đĩa CD-ROM

Sau đây là 1 số số liệu về ổ CD-ROM và CD-RW hiện đang lưu hành do PCWold - US cung cấp :

		Tốc độ đọc		T/độ ghi lên CD-R			Tốc độ ghi lên đĩa CD-RW			
Ổ ĐĨA	Giao tiếp	Theo nhà sx	Thực tế	Theo nhà sx	Creator phút giây	Packet writing phut/gy	Theo nhà sx	Creator (phút giây)	Packet-wting	P-Writing Ghiđè
				CD-ROM						
Teac 4x 12	SCSI-PCInt	12X	12X	4X	13:36	18:11				
Plextor PXR412C		12X	10X	4X	13:46	12:33				
VerbatimC	nt	12X	12,	4X	13:3	17:				

DR 4x12			1X		6	43				
CR-2801TE	EIDE	8X	7,5X	2X	27:11	62:23				
PX-R412Ce/ISA	SCSI ISA	12X	10X	4X	13:46	12:59				
SonyCDU928E/C	EIDE	8X	8X	2X	26:43	-				
Micronet4X12PC	SCSI PCI	12X	10X	4X	13:47	18:03				
CD-RW										
HP CD-WPlus 7200I HP CD-W Plus7200e Hi Val 2x6x2 CD RW CD-RW 226 Plus CD RW 426 Deluxe CD RW 226 Yamaha CRW4001 ti-PC	EIDE	6X	5,2X	2X	27:16	-	2X	27:28	30:48	34:56

Ta có thể dùng các số liệu này để tham khảo khi mua các ổ đĩa

3/ Ổ đĩa DVD (Digital Versatile Disc) Đĩa quang công nghệ số đa dụng:

DVD là kế vị của phương tiện lưu trữ bằng vật liệu quang .Đĩa DVD có đường kính120mm có thể ghi thông tin trên cả 2 mặt với dung lượng lưu trữ 2,6 đến 17GB âm thanh ,video hay dữ liệu dạng số (Loại CD chỉ có thể ghi thông tin trên 1 mặt với dung lượng 650MB).Các loại DVD bao gồm đĩa DVD ROM lưu thông tin chỉ đọc; đĩa DVD-R ghi thông tin 1 lần và DVD-RAM ,DVD+WR là những đĩa ghi lại được nhiều lần .DVD được dùng với nhiều chức năng khác nhau như phân phối phần mềm chuyển file sao lưu file hệ thống và các file cần thiết.

Giới phân tích nhận định rằng loại đĩa DVD ghi được có triển vọng sẽ thay thế cho phương tiện lưu trữ tháo lắp được như đĩa mềm ,CD và zip của Iomega.

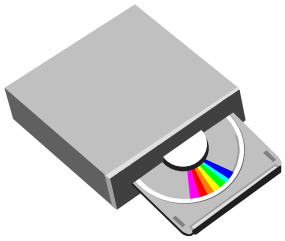
Cả 2 loại ổ DVD-RAM và DVD+RW đều có thể đọc được đĩa CD âm thanh ,CD ROM ,CD-R,CD-RW và DVD-ROM. Điều đáng nói là đĩa được tạo ra trên ổ DVD-RAM sẽ không làm việc trên ổ DVD+RW và ngược lại . Rất nhiều ổ DVD-ROM hiện nay cũng không được đảm bảo để đọc những đĩa sản xuất theo định dạng có thể ghi lại của DVD-RAM hay DVD+RW. Đây thực sự là 1 cuộc cạnh tranh giữa 2 chuẩn . ổ đĩa CD-RW giá khoảng 400$ còn DVD-RAM khoảng 800$.

Ta hãy xem bảng sau

	DVD-RAM	CD-RW	ZIP
Giá	750$		400$
150$			
Giá đĩa	25(đĩa 2,6GB)		20(đĩa 650MB)
16(đĩa 100MB)			
Những định	DVD-ROM,DVD-R		DVD-ROM,CD-R
Chỉ đĩa ZIP			
dạng đọc được	CD-RW,CD-R,CD-ROM		CD-ROM

THỰC HÀNH

1. Học sinh quan sát cấu tạo bên trong của ổ đĩa CD-ROM , so sánh với sơ đồ nguyên lý.
2. Hướng dẫn sửa chữa , tu chỉnh những bộ phận hay hỏng như mắt , gương.



CHUỘT
(Mouse)

Cấu tạo của chuột :

Một viên bi thép bọc nhựa luôn tiếp xúc với 2 trục lăn đặt vuông góc với nhau.Khi chuột dịch chuyển ,bi lăn,làm 2 trục quay theo .Các đĩa gắn trên 2 trục cũng quay
tỉ lệ với chuyển động theo 2 hướng X,Y .Trên 2 đĩa có các rãnh nhỏ .Các rãnh này sẽ liên tục đóng,mở 2 chùm ánh sáng tới các sensor nhạy sáng để tạo ra các xung điện.Số lượng xung tỷ lệ với chuyển động của chuột theo các hướng X,Y .Các xung này được đưa vào máy tính để xử lý. Trên chuột còn có 2 hoặc 3 phím .Khi các phím này đóng sẽ tạo ra các xung điều khiển tác động

Hình vẽ

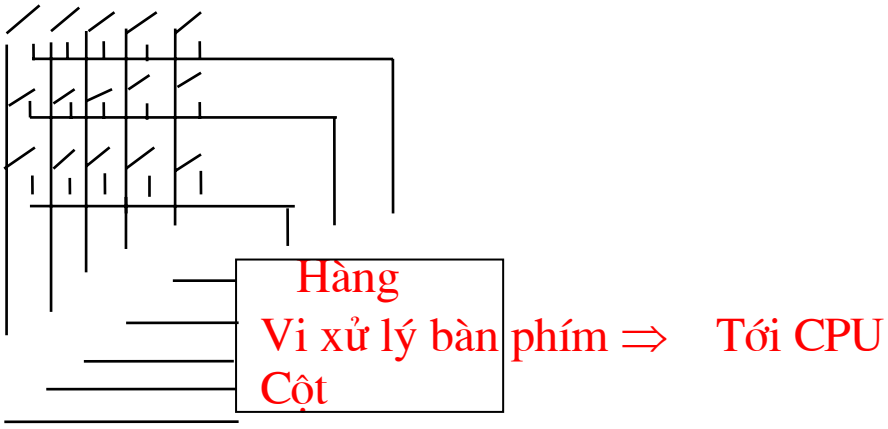
BÀN PHÍM

Có rất nhiều loại bàn phím với các nguyên lý khác nhau . Hiện thông dụng sử dụng
loại bàn phím áp dụng nguyên lý mã quét.

bàn phím có 104 phím . Đây là tập hợp các công tắc ,được bố trí thành 1 ma trận.

Khi tác động(ấn phím),tín hiệu ra được đưa đến 1 vi xử lý bàn phím . Chương trình phần mềm sẽ quét và xác định xem phím nào được ấn. Vi xử lý bànn phím sẽ biến đổi mã quét thành mã ASCII để CPU xử lý

.



Về mặt cấu tạo vật lý các bàn phím được cải tiến cho phù hợp với các tư thế hoạt động tự nhiên của tay người . Người ta gọi đây là các ***bàn phím công thái học (ergonomic)*** Bàn phím được chia thành 2 phần cách xa nhau vài inch ,đồng thời phím dành cho ngón cái được nâng cao hơn .Phím Back space và phần bàn phím số được đặt gần nhau hơn để các ngón tay và cánh tay không bị vơí ra xa .

Những sửa đổi này sẽ giúp tránh được mỏi mệt ,từ cánh tay,cổ tay ,đến vai của người dùng do cẳng tay được đặt sấp hoàn toàn (Với phím thông thường ,cẳng tay bị xoắn khi ngón cái và bàn tay đặt song song với bàn phím).

CÁC LOẠI BUS MỞ RỘNG VÀ CARD PHỐI GHÉP

1/ Các loại bus mở rộng:

Bus mở rộng cho phép PC liên lạc được với các thiết bị ngoại vi ,các thiết bị này được ghép nối với máy PC qua các khe cắm mở rộng (slot).

Hiện nay sử dụng thông dụng trong các máy PC các loại bus mở rộng sau :

* Bus ISA: (Industry Standard Architecture):

Dùng cho hệ thống chỉ được điều khiển bởi 1 CPU trên bản mạch chính tức là tất cả các chương trình và thiết bị đều chỉ được điều khiển bởi CPU đó

Tần số làm việc cực đạI 8,33MHz (8,33 Mbyte/giây cho số liệu 2 byte 1 lần)

Bề rộng dữ liệu 8 hoặc 16 bit

Bus địa chỉ 24 bit

Hình vẽ cấu tạo bus ISA

* Bus EISA: (Extended ISA):

Dùng cho hệ thống cho phép 1 vi xử lý nằm ngoài bản mạch chính có thể điều khiển toàn bộ bus

Tần số làm việc cực đạI 33MHz

Bề rộng dữ liệu có thể truy xuất 2 đường 8 hoặc 16 bit

Bus địa chỉ 32 bit

Hình vẽ cấu tạo bus EISA

* Bus PCI :(Peripheral component interconnect)

Đây là loại bus trong đó các số liệu và địa chỉ được gửi đI theo cách thức dồn kênh
(Multiplexing),các đường địa chỉ và số liệu được dồn chung trên trên các đường dây của PCI . Dữ liệu được truyền tải theo mode burst (Địa chỉ chỉ được truyền đI 1 lần
sau đó được hiểu ngầm bằng cách cho các đơn vị phát hoặc thu đếm lên trong mỗi xung đồng hồ. Đỡ phải phát lại địa chỉ).
Tốc độ truyền tối đa 120Mbyte/s

Hình vẽ cấu tạo bus PCI

2/Một số loại card thông dụng :

•Card vào ra (Card I/O):

Được ghép qua khe cắm ISA hoặc EISA phối ghép các thiết bị ngoại vi :

+ổ cứng
+ ổ mềm
+ Chuột
+ Cổng COM,LPT

với CPU

• Card màn hình : Cắm vào khe cắm ISA,EISA,VESA Local bus,PCI để phối ghép CPU với màn hình

Làm việc của Card màn hình :

* Cách hiện 1 ký tự trong chế độ text :

Ký tự hoặc hình vẽ được hiện lên màn hình bằng tập hợp các điểm sáng tối .Trong chế độ văn bản các điểm này được hình thành bằng việc có cho tia điện tử đập hay không vào màn huỳnh quang theo 1 khuôn mẫu có sẵn.Trong đó các điểm được tổ chức theo ma trận.

Các kích thước ma trận hay dùng trong thực tế là : 7x9,7x12,9x14 .Các mẫu chữ như vậy thường được tạo sẵn cho mỗi ký tự ASCII và được chứa trong 1 vi mạch nhớ ROM gọi là ROM tạo chữ . Vi mạch này là EPROM (Ký tự đầu của vi mạch là 27) , ta có thể dễ dàng thấy được vi mạch này trên bất cứ Card màn hình thông thường nào.

Sơ đồ mạch hiện chữ theo ma trận 9x14 trên màn hình hình vẽ

Sơ đồ khối của 1 mạch hiển thị được trang màn hình văn bản gồm có 80 ký tự theo chiều ngang và 25 ký tự theo chiều dọc (80x25).

Mã ASCII của các ký tự thuộc 1 trang màn hình cần hiển thị được chứa sẵn trong bộ nhớ RAM đệm màn hình (mỗi ký tự cần 1 byte) để ghi nhớ mã của nó . Nếu ta cần hiển thị 1 trang màn hình gồm 80x25=2000 ký tự thì ta cần dùng đến 1 bộ nhớ RAM đệm có dung lượng cỡ 2KB .Nội dung của bộ nhớ RAM đệm này được bộ điềud khiển màn hình đưa ra định kỳ để làm tươi màn hình sau 1 khoảng thời gian nhất định (Như vậy màn hình để hiển thị thông tin làm việc ở chế độ động) . Bộ nhớ RAM đệm này còn phải được thâm nhập bằng bộ vi xử lý để ta còn có khả năng thay đổi được nội dung cần đưa ra hiển thị . Các địa chỉ A0..A6 sẽ xác định vị trí của ký tự cần hiển thị trong 1 hàng còn các địa chỉ A7- A11 sẽ xác định toạ độ tính theo cột của cả 1 hàng ký tự cần hiển thị . Tổ hợp các bit địa chỉ A0-A11 của RAM đệm sẽ quyết định toạ độ cụ thể của 1 ký tự trên màn hình .

Như vậy : ***RAM đệm sẽ xác định ký tự đưa ra "ở đâu ?" trên màn hình***

"Cái gì ?" (chữ gì) được đưa ra thì lưu trong ROM tạo chữ

Trên Card màn hình ta cũng thấy rất dễ dàng RAM đệm này . Các loại Card màn hình thông thường phổ biến có RAM đệm = 1MB

* Cách hiện trong chế độ đồ hoạ :

Màn hình đồ hoạ 1 màu

Khi này không dùng đến ROM tạo chữ nữa và bộ nhớ RAM đệm lúc này thay vì chứa mã ASCII của ký tự thì lại chứa các điểm ảnh (pixel) mà tổ hợp của chúng chính là hình ảnh cần phải thể hiện. Chế độ làm việc này gọi là chế độ đồ hoạ.

Giả thiết ta phải hiện trên khung hình làm việc 640 điểm ảnh theo chiều ngang và

400 điểm theo chiều dọc thì cả khung hình làm việc này tương đương với 640x400=256.000điểm ảnh.Nếu để ghi nhớ mỗi điểm ảnh như vậy ta cần dùng 1 bit trong RAM đệm thì tức là ta cần đến bộ nhớ = 32.000bytes (gần 30 KB)

Màn hình đồ hoạ màu:

Màn hình màu khác màn hình 1 màu bởi sự có mặt của các cụm 3 phần tử trong lớp huỳnh quang phủ lên bề mặt phía trong của đèn hình,mỗi phần tử có khả năng phát ra 1 trong các màu R,B,G Màu của 1 điểm ảnh trên màn hình là sự kết hợp của 3 điểm sáng phát ra từ 3 phần tử màu đó khi chúng bị 3 tia điện tử phát ra từ 3 súng ở catốt đèn hình bắn vào . Để điều khiển điểm ảnh của màn hình màu ta phải có 3 tín hiệu để điều khiển 3 tia R,B,G kèm thêm 1 tín hiệu để điều khiển cường độ sáng(I) của điểm ảnh . Màn hình màu loại này gọi là màn hình màu RBGI . Để ghi nhớ thông tin cho 1 điểm sáng trên màn hình màu , trong bộ nhớ RAM đệm theo kiểu đã làm cho màn hình 1 màu ta phải tốn 4 bit thay vì 1 bit .Như vậy để hiện thị trên khung hình làm việc 640x400 điểm ảnh thì bộ nhớ RAM đệm cho màn hình màu phải có dung lượng 30kbx4. Đây là màn hình 16 màu.

ảnh 1 Card màn hình

• Card âm thanh :

Tín hiệu âm thanh- là dạng tín hiệu analog muốn làm việc với máy tính cần phải qua biến đổi thành tín hiệu số ,hoặc từ tín hiệu số ngược lại -thành tín hiệu tương tự .

Bản thân máy tính thông dụng không có bộ phận được thiết kế để làm nhiệm vụ này Phần các mạch điện tử được thiết kế thêm ,gắn vào máy tính qua các khe cắm mở rộng để làm nhiệm vụ này chính là các Card âm thanh.

Việc số hoá tín hiệu âm thanh và khôi phục lại tín hiệu âm thanh từ tín hiệu số

là quá trình gần đúng - có sai số . Muốn có âm thanh trung thực cần tăng tần số số hoá (tăng tần số lấy mẫu). Đây là 1 đặc trưng kỹ thuật cơ bản của Card âm thanh.

Trên Card âm thanh còn có thêm các mạch cải thiện chất lượng âm thanh : Nâng giảm các tần số , tạo hiệu ứng lập thể ...

Các Card âm thanh được ghép với máy tính qua các khe cắm ISA hoặc PCI

• Card đồ hoạ : Chức năng xử lý và hiển thị thông tin xuất từ máy tính.

Là 1 loại Card hình cao cấp ,giúp máy tính hiển thị hình ảnh nhanh hơn ví dụ card
PCI,card AGP,card 3D...

ảnhCard âm thanh , đồ hoạ

Số liệu về 1 số loại Card đồ hoạ

Board	Giá $ 6/97	Chip2D	Chip3D riêng	Đã cài RAM Video max	Loại RAM Video	T.độ RAM DAC	T.độ quét max ở 1024x768hz
ATI 3D Pro Turbo PC2TV	219	ATI Range II	Không	8/8	SGRAM	220	150
Diamond Stealth 3D 3000	170	S3 Virge / VX	Khong	4/4	VRAM	220	120
ATI 3D Xpression + PC2TV	129	ATI Range II	Không	4/4	SGRAM	170	150
STB Nitro 3D	149	S3Virge /GX	Không	4/4	EDOD RAM	170	120
Diamond Stealth 3D 2000 Pro	135	S3Virge /DX	Không	4/4	EDOD RAM	170	100
Hercules Terminator 3D/DX	149	S3Virge /DX	Không	4/4	EDOD RA	170	120

STB Velocity 3D	199	S3Virge /VX	Không	4/8	MEDOVRAM	220	120
Matrox Mystique 220	179	MatroxMGA1164SG	Không	4/8	SGRAM	220	140
Hercules Stingray 128/3D	249	Alliance Promotion-AT3D	Có	4/4	EDODRAM	180	120
Number NineFX Reality 772	279	S3Virge/VX	Không	4/4	VRAM	220	150

• Card MPEG :

Khác với card đồ hoạ ,card MPEG đọc từng frame ảnh trên CD ROM dưới dạng nén rồi giải nén nó để tạo lại các frame ảnh bitmap dạng rõ trước khi cho nó hiển thị lên màn hình(Thường thông qua video adapter).Với những CPU có tốc độ cao (Chẳng hạn từ Pentium 133 trở lên) ta có thể dùng phần mềm làm công việc của card MPEG với tốc độ chấp nhận được .Trong trường hợp này ta không cần trang bị card MPEG

Nếu ta có màn hình rộng và muốn chạy chương trình ứng dụng song song với việc xem phim thì vẫn phải trang bị card MPEG.

• Một số chuẩn giao diện thông dụng trong các máy tính hiện nay :

* ST506 ,ESDI : Những loại này do sử dụng cho máy XT ,hoặc không phổ biến ta sẽ không đề cập đến. Chủ yếu là các loại sau:

* Card IDE (Integrated Driver Electronics) và Card EIDE: ổ điện tử tích hợp

Các mạch điện tử sẽ kiểm soát các đơn vị được cất trong ổ đĩa . IDE chỉ quản lý được 2 đĩa cứng nối với hệ thống . Sau người ta đã cải tiến thành loại EIDE (Enhanced IDE) quản lý được 4 thiết bị . Các ổ IDE hiện hành đưa ra tốc độ chuyển giao từ 1MB đến 4MB mỗi giây.

Card IDE chỉ điều khiển được ổ đĩa cứng IDE mà thôi tức là các ổ đĩa chứa được dưới 540MB dữ liệu . Nếu muốn điều khiển các ổ lớn hơn phải dùng EIDE hoặc dùng IDE kèm theo 1 phần mềm (Disk Manager Ontrack)

* Card SCSI (Small Computer System Inteface) :

1 Card loại này,theo từng cấp độ cao dần, quản lý được từ 8 thiết bị (SCSI-1 ,SCSI-2) cho đến 14 thiết bị (SCSI-3). Card SCSI-3 quản lý được 14 thiết bị và trình tiện ích lại tự động đóng mở terminator khi cần thiết và có thể cho phép khởi động từ ổ đĩa cứng bất kỳ hay ổ đĩa CD-ROM,tuỳ ý người dùng.

Card IDE cũng như SCSI có thể dùng Bus ISA hay Bus PCI . Với các mainboard loại mới hiện nay các Card này đã được tích hợp luôn vào mainboard (On-board). Ta có thể xem các số liệu này ở phần phụ lục cuối sách.

* Cổng nối tiếp đa năng USB (Universal Serial Bus):

Chuẩn công nghiệp mới này dùng đầu nối loại 1 cỡ vừa với tất cả để thay cho mọi cổng cũ khác trên PC . Ta có thể cắm mọi thứ vào cổng USB : màn hình,bàn phím ,chuột,modem,joystick,máy in ,máy quét,video camera. Ta còn có thể cắm 1 chuỗi thiết bị ngoại vi cái này nối cái kia , nghĩa là ta có 1 chuỗi thiết bị chạy từ 1 cổng duy nhất trên PC.Một số sản phẩm USB như máy quét và Camera số có thể hoạt động không cần dây cắm điện riêng- Dây nối USB có khả năng cung cấp nguồn điện.

Cổng USB hoạt động nhanh gấp 10 lần cổng song song ,gấp 100 lần cổng nối tiếp

dữ liệu trao đổi 2 chiều có thể nhận tín hiệu phản hồi cưỡng bức từ Joystick,cho phép lắp đến 127 kiểu thiết bị ngoại vi theo kiểu nan hoa.

Ưu điểm đầu tiên của USB là tốc độ xuất nhập nhanh và dễ lắp đặt :Bạn chỉ việc cắm cáp nối vào phía sau máy tính .Chẳng cần phải bận tâm tới Driver ,card cắm thêm hay xác lập thông số hệ thống

mới , thậm chí cũng chẳng cần khởi động lại máy. USB là 1 sản phẩm đã được nhiều hãng có tên tuổi lưu tâm cải tiến và phát triển Compaq,Digital,Equipment,IBM,Microsoft,NEC và Northern Telecom. Các công ty này từ khoảng 1995 đã cùng tìm ra 1 loại cổng chuẩn mới nhằm đơn giản hoá việc lắp đặt các thiết bị nhập dữ liệu , đồng thời cho phép sử dụng điện thoại để nói chuyện vơí máy tính .Các thông số của USB9.0 được hoàn tất vào tháng 11/1995 . Sáu tháng sau Intel công bố các chíp Intel430HX và 430VX PCIset là các chip đầu tiên hỗ trợ USB Từ tháng 6/1998 USB đã được hỗ trợ hoàn toàn bởi hệ điều hành Windows98 .Nhiều máy tính mới đã được trang bị không phải chỉ 1 mà đến 2 cổng USB . Đến giai đoạn này đã có hơn 400 thiết bị dùng USB .

Với các cổng song song hay máy in cũ có thể dùng 1 thiết bị cắm vào để chuyển đổi ra USB .Tương lai của USB rất sáng sủa.

• *Ngoài các loại Card thông dụng đã trở thành hàng hoá ,tuỳ theo yêu cầu thực tế người ta sẽ chế tạo ra các loại Card phối ghép với máy tính theo yêu cầu riêng .Trong điều khiển công nghiệp các loại Card này thường được ghép với ISA hoặc EISA.*

MÀN HÌNH VÀ BỘ NGUỒN MÁY TÍNH

1/Các loại màn hiển thị :

- ống tia điện tử CRT(Cathode Ray Tube):
- Màn hình tinh thể lỏngLCD (Liquit Cristal Display)
- Màn hình Plasma
- Màn hình 3 chiều.

Thông dụng trong các máy để bàn là loại màn hình CRT

Cho loại máy xách tay là LCD

2/Nguyên lý làm việc của màn ống tia điện tử CRT(Cathode Ray Tube):

a/Sự lưu ảnh trong võng mạc mắt người:

Khi quan sát 1 hình ảnh hiện tắt với f >= 25 lần/giây mắt người không nhận ra được sự nhấp nháy đó .Người ta đã lợi dụng khuyết tật này của mắt để xây dựng nguyên lý quét ảnh.

b/ Cấu tạo ống CRT và Nguyên lý quét ảnh:

hình vẽ nguyên lý ống CRT

ống tia điện tử hình phễu, phần mở rộng là phần màn ảnh. Bên trong phần màn ảnh này có quét lớp phát quang (Khi có điện tử đạp vào thì chất này phát ra ánh sáng ,cường độ sáng phụ thuộc số lượng điện tử đập vào,phụ thuộc gia tốc của chúng khi bay đến .

Tia điện tử phát ra từ catot đập đến màn phát sáng . Cường độ tia điện tử này
lại phụ thuộc độ sáng tối của hình ảnh. Nếu không có quét tia điện tử sẽ đập mãi vào điểm giữa màn hình.

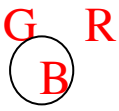
Bộ lái ngang sẽ làm tia điện tử chạy từ trái sang phải màn hình(Quét thuận) rồi lại trở về trở về bên phải màn hình (Quét ngược) . Thời gian quét ngược rất nhỏ so với thời gian quét thuận.

Bộ lái dọc sẽ làm tia điện tử chạy từ trên xuống dưới rồi lại từ dưới lên trên,cuối cùng trở lại vị trí đầu.

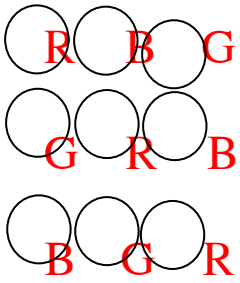
Việc quét 1 hình ảnh lên màn hình giống như ta cầm 1 cái bút vẽ rất nhanh theo kiểu quét ;"bút " ở đây là tia điện tử.

Với màn hình màu nguyên lý cũng tương tự .Chỉ khác là không phải 1 catot phát tia điện tử mà là 3 catot cho 3 màu Đỏ Xanh Lơ (R,B,G) và màn hình là 1 tổ hợp các điểm màu R,B,G kề sát nhau .Các điện tử phát ra từ catot Đỏ chỉ có thể đập vào các điểm phát màu đỏ . Cũng tương tự như vậy với các màu Xanh và Lơ. Một điểm ảnh sẽ là tổng hợp giá trị của 3 điểm màu .

Các điểm màu bố trí như sau :



Hãng Sony không dùng nguyên lý bố trí các điểm màu như trên mà dùng nguyên lý TINITRON



Trong màn hình máy tính độ sáng của các điểm không phải do các tín hiệu video đưa đến mà được lưu giữ trong bộ nhớ trên card màn hình.

Card màn hình là bộ phối ghép giữa CPU và màn hình

3/ Màn hình tinh thể lỏngLCD (Liquit Cristal Display):

LCD là công nghệ hiển thị dựa trên các đặc tính cản ánh sáng của tinh thể lỏng khi bị phân cực bởi điện áp .Tinh thể lỏng là 1 dạng đặc biệt của vật chất được cấu tạo từ các phân tử hình que.

LCD bao gồm 1 lớp tinh thể lỏng nằm giữa 2 tấm lọc phân cực .Tấm lọc là bản Plastic có đặc tính chỉ cho phép xuyên qua nó những sóng ánh sáng đI song song với 1 mặt phẳng xác định . Giữa các tấm lọc và lớp tinh thể lỏng là lưới điện cực mỏng trong suốt . Bởi LCD tiêu thụ ít năng lượng hơn các thiết bị phát xạ nên chúng được sử dụng nhiều trong những lĩnh vực cần tiết kiệm năng lượng.

Từ những năm 1996 về trước chỉ những máy tính xách tay (note-book) cao cấp nhất mới được trang bị màn hình LCD có độ phân giải 800x600 .Phần lớn dừng lại ở mức 640x460 . Nhưng đến thời điểm này những màn ảnh LCD có độ phân giải 1024x728 đã phổ biến .

Vấn đề hiện còn tồn tại với LCD là chưa có bộ tăng tốc đồ hoạ để có thể hiển thị màu thực ở độ phân giải 1280x1024. Vấn đề chắc sẽ được giải quyết trong thời gian tới.

4/ Bộ nguồn máy tính :

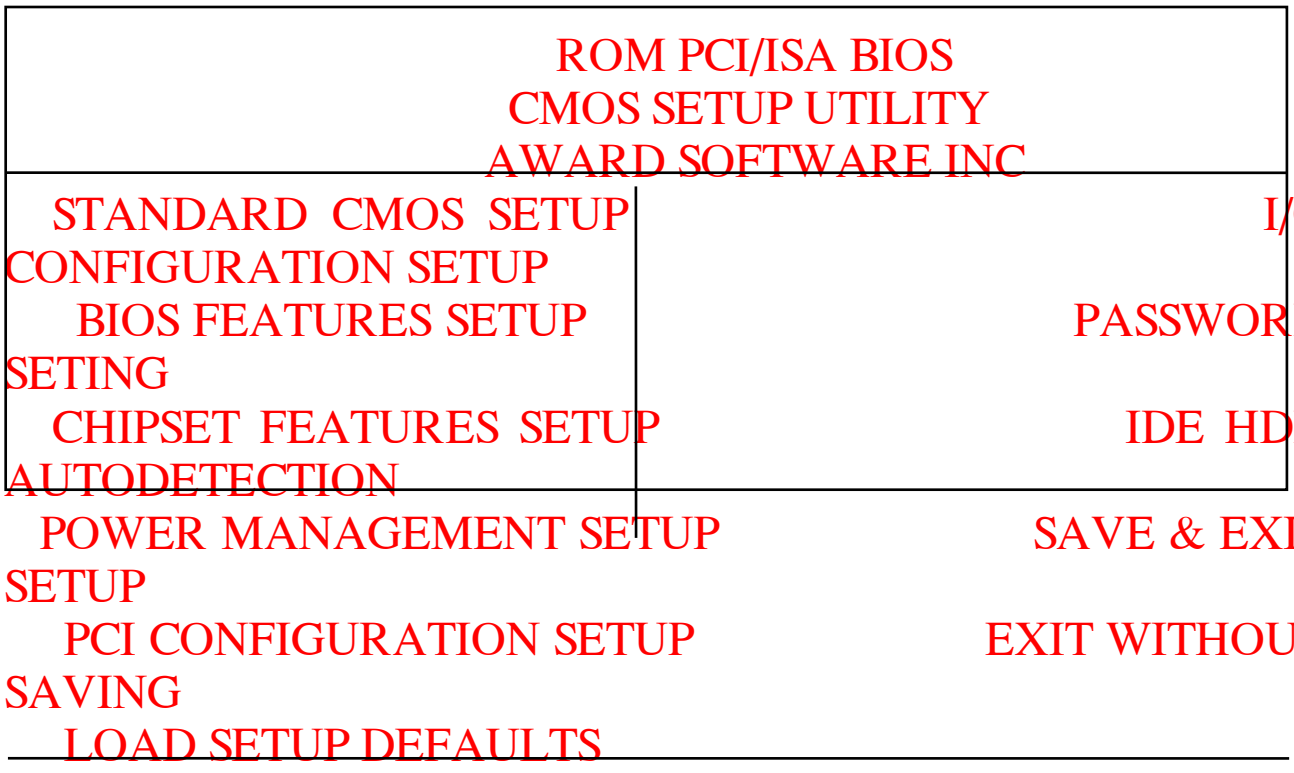
Cung cấp các điện áp +12,-12V,+5V,-5V để cung cấp cho các vi mạch và thiết bị ngoại vi. Một bộ nguồn tốt phải cho ra các mức điện áp đúng theo yêu cầu như trên.
Ta kiểm tra tình trạng đúng đắn của bộ nguồn bằng cách đo các chân điện áp ra.

PHẦN 2
RAM-CMOS VÀ CẤU HÌNH HỆ THỐNG

1-Khái niệm : Một máy PC do nhiều bộ phận ghép nối với nhau .Điều này xuất phát từ yêu cầu giải quyết công việc và từ khả năng tài chính của người dùng . Có máy dùng ổ cứng lớn , có máy dùng ổ cứng nhỏ ,màn hình khác nhau VGA,EGA...Nói tóm lại cấu hình của 1 máy PC rất đa dạng . Để cho hệ điều hành biết được cấu hình của từng thiết bị ngoại vi ,của bộ nhớ để điều khiển chính xác hoạt động của hệ thống và đối với 1 hệ thống đang hoạt động ổn định thì khi ta thêm vào hay thay thế 1 thiết bị ngoại vi bằng 1 loại khác thì hệ thống có nhận biết được sự thay đổi này không . Để hệ thống có thể nhận diện được cấu hình máy ,các thông tin cấu hình này cần được khai báo trong 1 bảng được BIOS chuẩn bị sẵn ,đó là bảng thông số SETUP . Các thông tin đã được khai báo này tồn tại thường xuyên ở 1 vùng nhớ ghi đọc được nhờ vào nguồn nuôi là 1 quả pin nhỏ , vùng nhớ này được gọi là RAM-CMOS . Nguyên thuỷ vùng nhớ CMOS trong IBM /PC dài 64Bytes .Các thông tin lưu trong CMOS bao gồm các thông số ổ đĩa ,ngày giờ thực , chế độ hoạt động của bàn phím ,chế độ khởi động... Sau này do máy tính bổ xung thêm nhiều thiết bị ngoại vi và các thiết bị khác nữa nên vùng nhớ này được tăng lên 128 bytes rồi hiện nay 256bytes .

2-Sử dụng chương trình SETUP :

Để vào Setup ta phải ấn 1 phím hoặc tổ hợp phím nào đó khi máy đang khởi động (Có thể quan sát thông báo trên màn hình) .Thường là Del hoặc Ctr + Alt + Esc . Màn hình Setup của BIOS sẽ xuất hiện như sau :



Đây là màn hình SETUP của hãng AWARD INC lắp trên máy Pentiun 586 .Tuỳ theo
hãng mà màn hình này có hình dáng khác nhau ,nhưng các mục thì cơ bản cũng vẫn như vậy . Các chức năng :

1. STANDARD CMOS SETUP
   Cho phép đặt các tham số về ổ đĩa , ngày giờ , loại ổ đĩa cứng,mềm...
2. BIOS FEATURES SETUP
   Đặt các chế độ báo có chương trình lạ xâm nhập boot sector hoặc bảng partition
   trình tự khởi động từ đĩa nào ,đặt mật khẩu...
3. CHIPSET FEATURES SETUP
   Đặt các thông số của RAM ,

4. POWER MANAGEMENT SETUP
   Đặt các chế độ tiết kiệm điện
5. PCI CONFIGURATION SETUP
   Đặt các thông số cho các thiết bị PCI
6. LOAD SETUP DEFAULTS
   Load các thông số ngầm định của BIOS
7. I/O CONFIGURATION SETUP
   Đặt cấu hình cho các cổng vào ra
8. PASSWORD SETING
   Thiết lập chế độ đặt mật khẩu bảo vệ chống tự tiện truy nhập (Phải kết hợp với
   phần 2)
9. IDE HDD AUTODETECTION
   Tự động tìm ổ đĩa cứng
10. SAVE & EXIT SETUP
    Ghi các thay đỏi và thoát ra để khởi động
11. EXIT WITHOUT SAVING
    Thoát ra nhưng không ghi

Khi thiết lập các thông tin này nếu ta không có kiến thức hoặc thiết lập sai thì máy sẽ hoạt động không bình thường : Không nhận ra ổ cứng ,không khởi động được, không có ổ đĩa mềm,không có chuột ,máy in không ghi được ... Các trục trặc kiểu này rất đa dạng do ta đặt CMOS sai , mặc dù các bộ phận vật lý không có gì hư hỏng cả. Vì vậy nếu không được hướng dẫn thì không nên thay đổi các thông số của RAM CMOS

3-Cất giữ phục hồi CMOS:

a-Dùng đĩa Rescue :

Việc lưu giữ thông tin trong CMOS và các thông tin trên phần khởi động của đĩa cứng rất quan trọng . Người ta hay dùng chương trình Rescue.exe trong bộ Norton Utility để thực hiện . Tiến hành như sau:

- Vào Norton Utility → Rescue Disk ↵
- Run Program → Continue → OK
- Chọn đĩa mềm A
- Creat : Chuẩn bị 3 đĩa mềm

- Sau đó làm lần lượt theo chỉ dẫn trên máy
- Cuối cùng xuất hiện thông báo có thử đĩa vừa mới làm xong không ? Ta có thể không thử .

b- Dùng phương pháp in màn hình :

Nếu bạn có máy in có thể lưu các thông tin trong chương trình SETUP bằng cách in màn hình . Mở các mục cần in rồi ấn phím Print screen. Cất các bản in để lưu trữ , khi có vấn đề ta dựa theo các thông tin này để thiết lập lại hệ thống.

c- Dùng chương trình để lưu thông tin trong CMOS:

Lưu các thông tin CMOS vào 1 file .Khi có sự cố , mở file , viết lại thông tin đã cất vào CMOS.

4/ Dấu đĩa cứng - Chống xâm nhập trái phép - Mật khẩu bảo vệ CMOS :

ĐôI khi chúng ta có nhu cầu bảo vệ máy tính của mình tránh khỏi những người khác tò mò ,hoặc không muốn người không am hiểu kỹ thuật thay đổi các thông số trên máy ta có thể áp dụng các phương pháp bảo vệ như đặt mật khẩu hoặc dùng các chương trình đặc biệt để che dấu đĩa cứng . Ngay trong CMOS của máy tính cũng đã cài sẵn 1 chức năng đặt mật khẩu . Chúng ta có thể tận dụng chức năng này để chống người lạ sử dụng máy.

a- Chống khởi động máy và truy nhập CMOS :

- Đầu tiên trong mục BIOS FEATURES SETUP

Chọn Security mục này tuỳ loại CMOS có thể có 2 hay3 tuỳ chọn , chẳng hạn 3 tuỳ chọn:

- System : Hỏi mật khẩu khi bật nguồn khởi động máy
- Setup : Hỏi mật khẩu khi muốn vào CMOS
- None : Không sử dụng mật khẩu

Dùng các phím PgUp,PgDwn (hoặc dùng + ,-với loại BIOS Phoenix) để thay đổi các giá trị này. Giả sử ta chọn có mật khẩu (System hoặc Setup).

- Tiếp theo , sang mục PASSWORD SETING :

Gõ Enter . Xuất hiện thông báo :

Enter Password :-

Ta gõ vào 1 mật khẩu dài từ 1 đến 8 ký tự sau đó gõ Enter .BIOS sẽ hiển thị thông báo yêu cầu xác nhận mật khẩu như sau :

Cornfirm Password :-

Gõ lại mật khẩu 1 lần nữa ;đúng như lần trước -sau đó gõ Enter ,nếu thấy có thông báo:

PASSWORD ENABLED

thì có nghĩa là mật khẩu đã được thiết lập đúng . Sau này mỗi khi muốn vào CMOS
hoặc khi khởi động máy ta phải đánh vào đúng mật khẩu . Nếu đánh sai sẽ không vào CMOS , hay khởi động được.

Để gỡ bỏ 1 mật khẩu đã đặt trước đó ta vào chức năng PASSWORD SETING
khi thấy thông báo

Enter Password :-

thì ta không gõ gì cả mà ấn Enter . BIOS sẽ hiển thị thông báo :

PASSWORD ENABLED
Press any key to Continue

để thông báo mật khẩu đã bị bỏ .

Trường hợp quên mật khẩu thì có thể dùng các biện pháp sau:

- Dùng mật khẩu tên hãng BIOS : Tức là khi có yêu cầu mật khẩu ta đánh vào các từ sau : Với BIOS của AWARD INC thì gõ vào CONCAT . Với BIOS của AMI thì gõ vào AMI.
- Dùng các phần mềm để phá khoá : Các chương trình tiện ích để kiểm tra máy như PCCHECK,AMIDIAGS đều có thể dùng để phá khoá . Dưới đây là 1 chương trình dùng để phá khoá CMOS được viết bằng Pascal . Chương trình có thể phá được hầu hết các loại khoá CMOS của các BIOS đang có mặt tại Việt Nam :

Program Delete_CMOS_Password;

```
BEGIN
   Port[$70]:=$2F;
   Port[$71]:=$FF;
END.
```

- Thay đổi Jumper (Cầu nối trên Mainboard) : Một số loại Mainboard có sẵn 1 Jumper để xoá CMOS . Cầu nối này có 2 vị trí : khi để sang vị trí Clear thì CMOS sẽ bị xoá . Để có thể dùng ứng dụng này ta phải biết chắc chắn Jumper nào làm nhiệm vụ gì ; tức là ta phải có sơ đồ của Mainboard . Nếu đấu mò rất dễ chết Mainboard .
- Tháo pin nuôi CMOS : Đây là biện pháp mà chẳng còn khoá nào tác dụng ! Nhưng nếu chuyên môn không cao sau đó không khôi phục lại được CMOS thì cũng làm cho hệ thống không hoạt động được,hoặc hoạt động trục trặc.

b-Dấu đĩa cứng :

Chúng ta có 2 phương pháp để dấu đĩa cứng .Phương pháp thứ nhất là dùng CMOS và phương pháp thứ 2 là dùng phần mềm.

- Dùng CMOS :

Chúng ta biết rằng CMOS có thể chống người khác xâm nhập bằng cách đặt mật khẩu (dù vẫn có cách phá), ta có thể lợi dụng đặc tính này.

Vào STANDARD CMOS SETUP . Trong mục Hard Disk đặt thông số đĩa cứng là None (Không có ổ đĩa cứng) và ta phải ghi nhớ lại các thông số của đĩa cứng bao gồm
số Cylinder ,Số Head,Số Sector rồi thiết lập mật khẩu cho CMOS ,lưu lại và thoát ra ,khởi động lại máy . Bây giờ muốn sử dụng lại máy ta phải đặt lại các đúng các thông số cho đĩa cứng . Phương pháp này chỉ dấu được những người không chuyên
- Nhưng rõ ràng là trong thực tế thì đối tượng này là đa số .

- Dùng phần mềm :

Người ta đã làm ra 1 số chương trình cho phép dấu đĩa cứng ,khi khởi động chương trình yêu cầu cho mật khẩu đúng thì mới tiếp tục ,nếu sai mật khẩu chương trình sẽ tự động làm treo máy. Chẳng hạn chương trình HDL.exe (Hard Disk Lock)của Đặng Minh Tuấn . Cơ chế hoạt động của chương trình như sau : Khi cài đặt lên máy

chương trình sẽ thêm vào Partition của đĩa cứng 1 đoạn mã của mình và chuyển bảng Partition đI lưu ở 1 chỗ khác trên đĩa.Đoạn mã của chương trình sẽ được nạp vào bộ nhớ khi máy đọc bảng Partition để khởi động , nó sẽ cho người sử dụng gõ vào 1 mật khẩu và kiểm tra mật khẩu đó .Nếu đúng thì chuyển điều khiển cho hệ điều hành nạp bảng Partition thật nếu sai chương trình cho phép gõ lại mật khẩu 1 số làn nhất định (thường là 3) khi gõ lại vẫn sai thì chương trình treo máy . Cách sử dụng chương trình như sau:

Chương trình có 3 tham số : I-Cài đặt ,U-Gỡ bỏ , C-Thay đổi mật khẩu .

Để đặt mật khẩu ta làm như sau : C:\>HDL.EXE I ↵

Khi thấy thông báo Enter "Password :" ta gõ vào 1 mật khẩu tối đa 10 ký tự và gõ Enter ,khi thấy thông báo yêu cầu xác nhận "RÊNTER PASSWORD:"bạn hãy gõ vào mật khẩu lần nữa và gõ Enter .

Để gỡ bỏ mật khẩu : C:\>HDL.EXE U ↵

Khi thấy thông báo " ENTER OLD PASSWORD :" hãy gõ vào mật khẩu của mình

chương trình sẽ tự động gỡ bỏ và phục hồi bảng Partition của đĩa cứng.

Để thay đổi mật khẩu : C:\>HDL.EXE C ↵

Chương trình sẽ yêu cầu cho mật khẩu cũ và mật khẩu mới . Hãy gõ vào mật khẩu cũ và gõ vào mật khẩu mới 2 lần.

c-Chống sự xâm nhập của các chương trình lạ vào Boot Sector hoặc bảng Partition của đĩa.

Muốn dùng tính năng này ,ta vào CMOS chọn mục BIOS FEATURES SETUP

ở tính năng Vius Warning hoặc Boot Sector Protection đặt giá trị ENABLED

Bây giờ mỗi khi có hiện tượng ghi lên các vùng quan trọng trên của đĩa cứng

BIOS sẽ hiển thị 1 thông báo như sau :

```
! WARNING !
Disk boot sector is to be modified
```

Type “Y” to accept or Write or “N” to abord write
Award Software Inc.

Thông báo này có ý nghĩa như sau :” Cảnh báo . Boot Sector trên đĩa đang bị sửa đổi .Gõ”Y” để chấp nhận, gõ “N” huỷ bỏ .Nếu bạn không dùng các lệnh tác động lên vùng hệ thống như lệnh SYS chẳng hạn mà bạn thấy thông báo này có nghĩa là virus
boot đang tấn công máy của bạn .Hãy bấm phím “N”để huỷ bỏ và tìm cách diệt virus.

THỰC HÀNH

1- Đặt các thông số cho ổ đĩa mềm :
   - Dấu ổ mềm
   - Đặt thành hiển thị 2 ổ mềm A,B
   - Đặt không cho phép truy nhập ổ mềm trên bộ điều khiển ổ mềm
2- Đặt các thông số cho ổ đĩa cứng
   - Đặt để không xuất hiện đĩa cứng (Dấu đĩa cứng)
   - Đặt để không truy nhập được đĩa cứng trên bộ điều khiển
3- Đặt không truy nhập được chuột
4- Đặt không truy nhập được cổng máy in
5- Đặt các tính năng sử dụng mật khẩu,xoá mật khẩu
6- Thực hiện lưu trữ thông tin CMOS

PHẦN 3

SỬA CHỮA CÁC HƯ HỎNG CỦA HỆ THỐNG MÁY TÍNH

CÁCH ĐỌC SƠ ĐỒ ĐIỆN - CÁC DỤNG CỤ TỐI THIỂU TRONG SỬA CHỮA

I/ Đọc sơ đồ mạch điện:

Giới thiệu ký hiệu của 1 số phần tử thông dụng trong mạch điện :

-Điốt bán dẫn
-Tranzixtor
-Vi mạch
-Điện trở
-Tụ điện
-Ký hiệu nguồn điện xoay chiều ,1 chiều
-Biến áp

II/Các dụng tối thiểu:

1/Mỏ hàn :

-Để tháo lắp các linh kiện khi cần thiết.Ngoài yêu cầu về công suất : đủ nóng để
làm nóng chảy thiếc còn 1 yêu cầu rất quan trọng là :Không bị rò điện.Mỏ hàn rò
điện sẽ làm hỏng linh kiện khi tháo lắp ; nhất là các linh kiện CMOS,FET

- Có 2 loại thông dụng :
  +Mỏ hàn dây quấn :40W-60W.Để hàn các loại vi mạch thường dùng loại 40W.
  loại mỏ hàn này có đặc điểm không bền,hay đứt.
  +Mỏ hàn chập mạch :(Một số người do thói quen gọi là mỏ hàn xung)
  Loại này bền nhưng chỉ hành được các mối hàn thông dụng
  (*Trong khi giảng nói thêm về hành nhúng trong sản xuất lớn và hàn laser trong hàn các mạch in nhiều lớp : máy điện thoại di động panen có 8 lớp mạch in ...*)
- Thiếc hàn

- Chất làm sạch chỗ cần hàn : Trước đây hay dùng sunfat kẽm (Bỏ 1 ít mẩu kẽm
vào axit sunfuric để có sunfat kẽm) .Phổ biến có thể dùng nhựa thông . Ngày
nay dùng thiếc cuộn :Trong lõi của dây thiếc đã có nhựa thông sẵn ,vì vậy
dùng rất tiện lợi.
- Nối mat đầu mỏ hàn phòng ngừa rò điện:
Dùng dây mềm nối vào phần kim loại của mỏ hàn , đầu kia của dây nối đất.

2/Đồng hồ vạn năng:

Trong những điều kiện đầy đủ các trang thiết bị :có sự trợ giúp của Osiloscop,
máy đếm xung , các thiết bị chuyên dùng ... thì việc phát hiện và sửa chữa hư
hỏng sẽ nhẹ nhàng hơn .Tuy nhiên với 1 khả năng phân tích cẩn thận ,chu đáo
và bao quát thì 1 đồng hồ vạn năng thông thường với trở kháng vào 5KΩ/V -
20 KΩ/V trong tay 1 kỹ thuật viên máy tính cũng sẽ phát huy tác dụng không
kém .Thông dụng hiện nay có 2 loại :

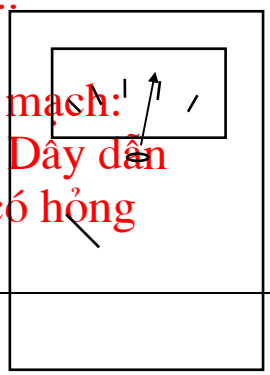
+ Loại kim chỉ thị
+ Loại chỉ thị hiện số + âm thanh

Loại chỉ thị kim thường có hình dạng như sau:

- Mặt hiện số trên thể hiện các giá trị đo : U,I,R...
- Que đo : Dây - và dây + .
- Pin nuôi đồng hồ phục vụ cho việc đo R (Không có pin vẫn đo được điện
áp)
- Cầu chuyển mạch đo : U,I,R...

Tác dụng :

- Đo điện trở ,Kiểm tra thông mạch:
Xem cầu chì có đứt không ? Dây dẫn
có thông không ? Công tắc có hỏng



AC

không ? Loa, bóng đèn còn tốt không?

...

- Đo điện áp xoay chiều:
- Đo điện áp 1 chiều:

Ví dụ : Nguồn +5V,-5V, +12V,-12V trong máy tính có không ?

Điện áp tại chân các vi mạch cần kiểm tra bằng bao nhiêu ?

- Đo dòng điện tiêu thụ

Cách kiểm tra 1 số linh kiện thông dụng :

- Điện trở và mã màu điện trở

Màu : Đen Nâu Đỏ Cam Vàng Xanh Lục Tím Xám Trắng

Giá trị : 0 1 2 3 4 5 6 7 8 9

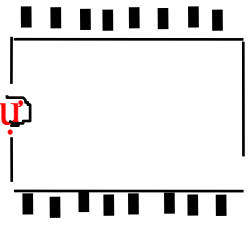
1 2 3 4

Vạch 4: Sai số của điện trở (%)

- Kiểm tra đI ôt :
- Kiểm tra tranzixtor
- Kiểm tra biến áp:
- Kiểm tra tụ điện :

Quy ước đánh số chân của vi mạch: 16 15 14 13 12 11 10 9

Nhìn từ trên xuống
Ngược chiều kim đồng hồ số thứ tự chân tăng dần



1 2 3 4 5 6 7 8

3/Các loại dụng cụ khác :

- Bút thử điện

- Panh gắp
- Hút thiếc
- Kính lúp
- Kéo,dùi

KIỂM TRA,SỬA CHỮA CHUỘT

Chuột là 1 thiết bị ngoại vi chuẩn dùng để đưa các mệnh lệnh của con người cho máy tính . Thường chuột được lắp vào cổng nối tiếp ở cổng COM1 (Địa chỉ 3F8).
Có thể truy nhập bằng ngắt 23h . Hàm cấm chuột là 20h.

- Đầu cắm chuột vào máy tính thường là 9 chân ,(loại 25 chân hiện nay không dùng nữa) theo chuẩn RS-232 có điện áp 12V
- Để chạy được chuột cần có :
  +Chuột tốt
  +Phần mềm điều khiển tốt
- Thiết lập phần mềm : Trong các file : .bat, setting
  Chú ý các khai báo trong RAM-CMOS đảm bảo sao cho
  cổng COM1 không bị khoá . Nếu khai báo sai cũng thông
  báo như chuột hỏng thực.
- Nếu chuột đang chạy bình thường mà bị hỏng thường do chuột hỏng:
  + Đứt dây : Khắc phục : Cắt đoạn hỏng bỏ đI ,nối lại .Trên các đầu dây nối vào chuột thường có đánh dấu các đầu dây bằng số theo luật mã màu.
  + Hỏng công tắc tác động :
  Khắc phục : Thay công tắc giữa sang .Đánh lại các tiếp điểm.

THỰC HÀNH

1. Khắc phục hư hỏng chuột dạng đứt dây
2. Thay công tắc tác động bị hỏng
3. Đánh lại các tiếp điểm của công tắc tác động
4. Kiểm tra diot phát quang,Sensor trên chuột

KHẮC PHỤC HƯ HỎNG TRUY NHẬP ĐĨA MỀM

Để có thể can thiệp vào hoạt động của ổ đĩa mềm , ta phải tác động qua các thanh ghi của cổng 3F0h. Có thể sử dụng ngôn ngữ C,Pascal hoặc tốt hơn cả là dùng Assembly ở đây với yêu cầu cho các kỹ thuật viên bảo trì phòng máy tính ta sẽ không đI sâu vào các vấn đề lập trình mà quan tâm đến các vấn đề kỹ thuật cụ thể.

1/Cách nối 1 ổ đĩa mềm vào bảng mạch chính:

Hiện nay , với các mainbord loại mới ,phần điều khiển vào ra của các thiết bị ngoại vi (ổ cứng , ổ mềm,chuột, máy in,bàn phím...) đã có ngay trên mainboard (onboard) nên không cần bảng mạch điều khiển I/O .Để nối ổ đĩa mềm hoặc các thiết bị ngoại vi khác với mainboard ta chỉ cần chú ý như sau : Cáp nối 34 chân của ổ đĩa mềm chân số 1 là chân nối với dây có dấu màu đỏ . Ta tìm trên mainboard tổ hợp chân cắm 34 chân, cắm sao cho chân số 1 của cáp(đánh dấu

màu đỏ) vào chân số 1 của tổ hợp chân cắm Đầu kia của cáp cũng được nối vào chân số 1 của tổ hợp chân cắm trên ổ đĩa mềm.

Với các mainboard cũ cần có bộ phối ghép I/O riêng (Card I/O)cắm vào EISA slot

Trên card I/O có các tổ hợp chân cắm: - 34 chân (đĩa mềm)
- 40 chân (cho đĩa cứng)
- 26 chân (cổng song song-máy in)
- 10 chân (cổng nối tiếp-chuột)

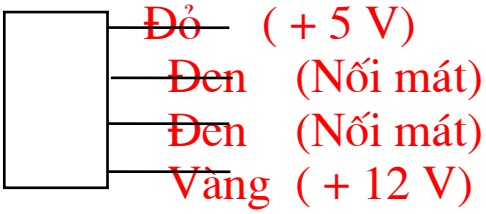
Khi cắm cáp nối ổ mềm(hoặc các thiết bị ngoại vi khác) ta cũng theo qui tắc chân số 1 (Đánh dấu màu đỏ) như trên.

Cấu tạo của cáp nối 34 chân như sau :

Hình vẽ

4,6 :Không sử dụng	18:
8 : Index signal	20:
10: Motor A	22:
11: Select A drive	24:
12: Select B drive	26:
13: Motor B	28:
14:	30:
16:	32:
	34:

Dây nguồn:

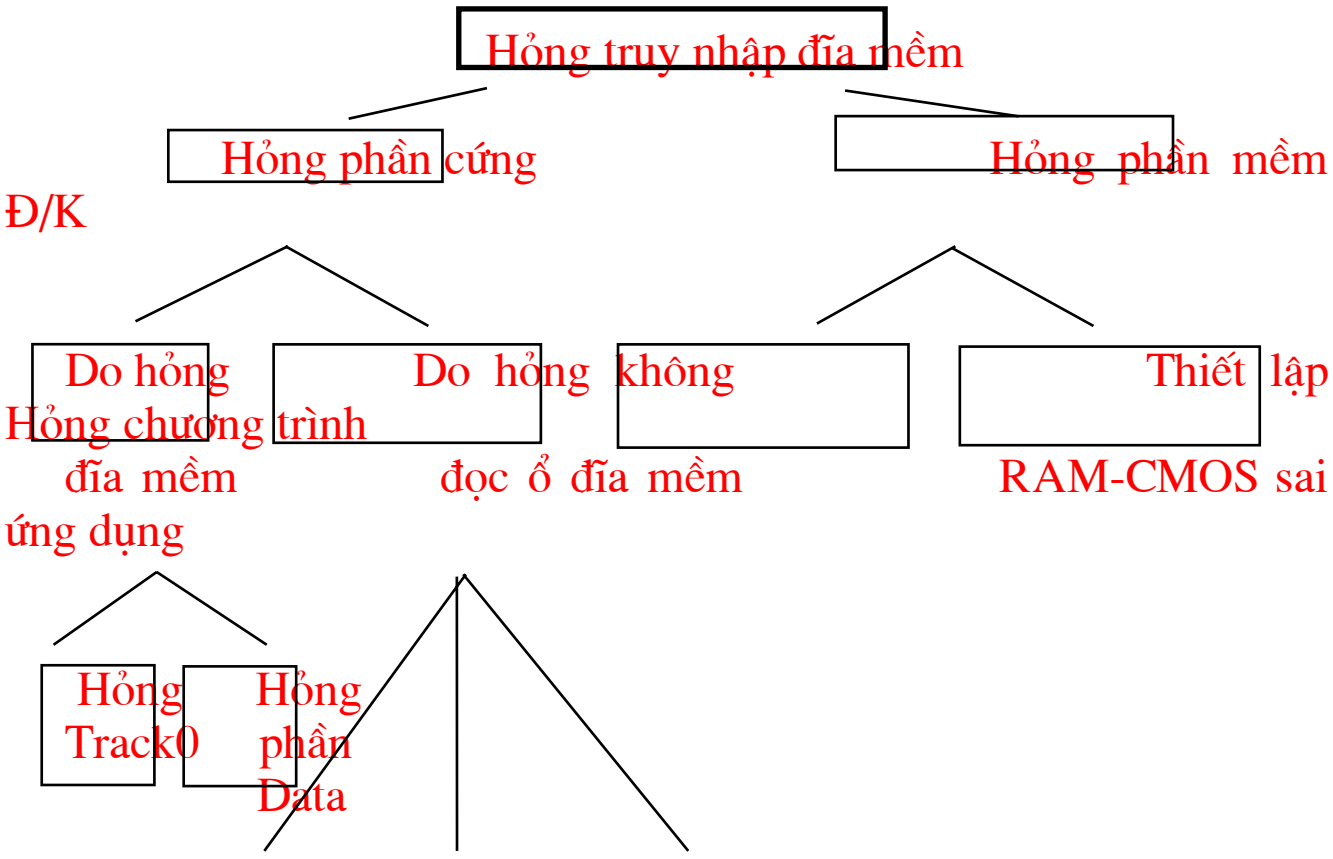


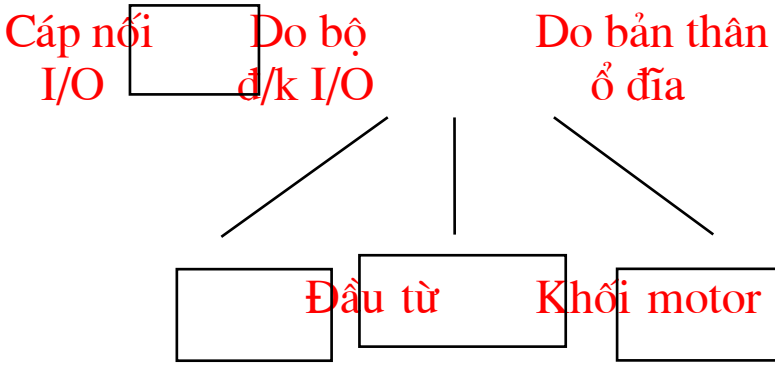
Cần chú ý không cắm nhầm đầu +5V sang đầu +12V sẽ làm hỏng các vi mạch

Thường thường thì giữa chân cắm và jăc cắm có hình dạng tương ứng không thể

cắm nhầm được.

2/Sơ đồ hỏng truy nhập đĩa mềm:





3/ Xử lý khắc phục hư hỏng:

a- Phần mềm :

- Thiết lập lại RAM-CMOS:

Chú ý các phần : STANDARD CMOS SETUP đặt ổ đĩa đúng loại :

A drive : 1.44 MB 3.5 inch

Không đặt sai thông số hoặc đặt None

I/O CONFIGURATION SETUP đặt

Onboard FDD Controller : Enable

Nếu đặt Disable sẽ không truy nhập được

b- Phần cứng :

- Sửa các đĩa hỏng bằng chương trình tiện ích NDD.EXE
- Cắm lại các đầu dây vào Mainboard, Card I/O, ổ đĩa theo qui tắc (Dây đỏ chân 1)
- Lau đầu từ bằng đĩa lau hoặc bằng bông tẩm cồn hoặc dung dịch tẩy
- Thay thế tráo đổi các phần ,dồn ghép các phần còn tốt của các ổ đĩa hỏng

THỰC HÀNH

1. Kiểm tra thiết lập CMOS phần ổ đĩa mềm
2. Cắm nối các cap nguồn , tín hiệu giữa card I/O (Mainboard) với ổ đĩa mềm
3. Thực hành lau đầu từ
4. Thay thế tráo đổi các phần ,dồn ghép các phần còn tốt của các ổ đĩa hỏng

VI RUT MÁY TÍNH -CÁCH PHÒNG VÀ CHỐNG SỬ DỤNG 1 SỐ CHƯƠNG TRÌNH QUÉT VI RUT THÔNG DỤNG

CÁCH TẠO ĐĨA " BẢO BỐI "

I-Vi rút máy tính - Cách phòng và chống:

1/ Khái niệm:

Nhiều người sử dụng máy tính nhầm tưởng rằng virus máy tính là 1 dạng sinh vật điện tử . Thậm chí họ còn nghĩ rằng virus có khả năng truyền từ máy tính này sang máy tính khác mà không cần tiếp xúc vật lý và chúng có khả năng sống ngay cả khi đã tắt nguồn máy tính . May thay ngày nay còn rất ít người suy nghĩ như thế . Vậy rốt cục thì virus máy tính là gì ?

Ta đã biết các chương trình máy tính là các dãy chỉ thị do con người nghĩ ra chỉ thị cho máy tính hoạt động nào được thực hiện và thực hiện như thế nào ; và người ta cũng biết rằng : virus thực chất cũng là các chương trình máy tính.

Các virus được nạp và chạy mặc dù người sử dụng không yêu cầu . Chúng hoạt động ,không để lại dấu vết giống như các chương trình bình thường.

Các virus có thể :

+ Tạo khuôn đĩa ,sao chép ,đổi tên tệp

+ Tự sao chép với các thông tin cấu hình mới
+ Thay đổi thuộc tính các tệp

Định nghĩa : 1 virus máy tính là 1 chương trình làm thay đổi các chương trình khác để thêm vào 1 bản sao thực hiện được và có thể thay đổi chính nó.

Tiêu chuẩn tối thiểu để thiết kế virus máy tính là :
+ Thực hiện được
+ Có khả năng tự sao chép
+ Biến đổi các đối tượng thực hiện được khác thành các đối tượng "nhiễm khuẩn"

2/ Phân loại:

Có nhiều loại phần mềm có hại:
+ Bom logic :
+ Bom thời gian :
+ Các chương trình " Đổi màu " ăn cắp mã khoá ngân hàng
+ Các "con sâu"
+ Các loại virus

Có hai loại virus :
+ virus file : Lây vào file xex,doc,dot...
+ virus boot : Nhiễm vào vùng khởi động ,ngăn cản quá trình khởi động , chiếm lĩnh bộ nhớ

3/ Cơ chế lây lan:

Xuất phát từ nguyên lý ta thấy chỉ khi nào chương trình mang virus được thực hiện thì mới bị nhiễm . Quá trình " thực hiện " đó là :
+ Khởi động bằng đĩa đã bị nhiễm virus
+ Đọc các file doc đã bị nhiễm
+ Thực hiện các chương trình com,exe đã bị nhiễm
+ Lây từ các chương trình sao chép từ mạng Internet
+ Lây từ đĩa mềm , đĩa cứng ,CD-ROM," gốc"

* Nếu dùng DIR của DOS hay NC để xem 1 đĩa bị nhiễm (xem từ ổ C:) thì cũng không bị lây vi rut.
* Copy sao chép cũng không bị ,chỉ khi nào chạy các chương trình đó mới bị lây nhiễm.

4/ Chuẩn đoán các máy tính bị nhiễm vi rút:

- Thao tác máy trở nên chậm chạp
- Nạp chương trình lâu hơn bình thường
- Các chương trình truy nhập bộ nhớ không bình thường,nhanh chóng tràn bộ nhớ
- Các chương trình truy nhập đĩa vào những thời điểm không bình thường với tần suất cao.
- Không gian tự do của đĩa giảm nhanh
- Số lượng sector đĩa hỏng tăng nhanh
- Các chương trình gặp những lỗi mà trước đây không hề có
- xuất hiện các thông báo lạ
- Các tệp bị mất không rõ nguyên nhân
- Tên ,thuộc tính tệp bị thay đổi
- Kích thước tệp bị thay đổi

Tóm lại là các hoạt động không bình thường

5/ Làm gì khi máy tính bị nhiễm vi rut ?

- Nếu đã khẳng định việc nhiễm virus thì cần làm các biện pháp loại trừ:

+ Tắt máy tính , chờ 60 giây
+ Tháo các môi trường lưu trữ ra : Đĩa mềm,băng từ...
+ Tháo các thiết bị ngoại vi : Máy in,modem...Chỉ để lại màn hình ,bàn phím,chuột
+ Đưa đĩa sạch vào (Đĩa này cần chống ghi)trên đĩa có chương trình tìm và diệt virus để thực hiện việc tìm và diệt
+ Đưa các file hệ thống sạch vào
+ Tắt máy , nối lại hệ thống.

Phòng ngừa :

+ Dùng đĩa mềm khởi động
+ Kiểm tra các chương trình trước khi đưa vào máy
+ Đổi tên các tệp dễ bị lây nhiễm ,mục tiêu của các chương trình virus
+ Khởi tạo lại hệ thống

6/ Một số chương trình diệt virus thông dụng:

+ Chương trình DW
+ Chương trình BKAV
+ Chương trình D2

II- Cách tạo đĩa " Bảo bối ":

1/ Khái niệm:

-Sự cần thiết của bộ đĩa sửa chữa :Hiện nay trong sử dụng chúng ta thường khởi

động máy PC từ ổ đĩa cứng ;điều đó rất nhanh chóng và thuận tiện.Tuy nhiên nếu khi ổ đĩa cứng bị gặp sự cố,nhiễm vi rut mà vẫn còn có khả năng cứu được ,mà khi đó ta lại không có bộ đĩa sửa chữa sẵn trong tay thì thật là khó khăn cho công việc sửa chữa .

- Bộ đĩa có thể 1 cho đến vài đĩa -đều là đĩa khởi động- trong đó có chứa :

+ Các chương trình quét vi rut.

+ Các file chương trình :

fdisk.exe,diskedit.exe,format.com,unerase.exe,unformat.exe,ndd.exe

himem.sys.

Các file chương trình trên phải được sao chép từ nguồn " sạch " (Không có khả năng đã nhiễm vi rut).

2/Thực hiện :

* Làm trên DOS:

Nguồn chương trình có thể từ đĩa mềm :

- Cho đĩa mềm khởi động có chứa các chương trình cần thiết như đã nói ở

mục 1 vào ổ đĩa .

-Khởi động máy .

- Tại dấu nhắc DOS gõ A:\> format A:/s

- Bỏ đĩa cũ ra đưa đĩa cần tạo vào rồi ấn ↵

- Sau bước này ta đã có đĩa khởi động .Chép các chương trình cần thiết vào.

- Tạo hoặc chép các file config.sys , .bat cần thiết.

-Gạt chống ghi.

Nguồn chương trình có thể từ đĩa cứng :

- Cho đĩa cần tạo lập vào ổ đĩa.

- Đánh lệnh C:\> format A:/s↵

- Chép các chương trình cần thiết vào

- Gạt chống ghi.

*Làm trên WINDOWS:

-ấn Start → Setting → Controlpanen → Add/Remove Program → Tab Startup →

Create Disk . Cho đĩa vào ổ A OK ↵ . Rồi về Add/Remove Program .Như vậy

đã xong

- Chép các chương trình cần thiết vào
- Gạt chống ghi.

* Nếu để đồng thời làm đĩa Rescue cho máy đang sử dụng thì ta phải làm 3 đĩa mềm (Hoặc đơn giản làm 1 đĩa - Dùng chương trình tạo đĩa Rescue trong NU).

Các đĩa bảo bối dùng cho sửa chữa của chúng ta được giữ gìn cẩn thận , khi ổ đĩa cứng của chúng ta (hoặc của 1 máy khác) bị sự cố ,chúng ta sẽ đem ra làm công cụ phục vụ sửa chữa.

THỰC HÀNH

1. Tạo đĩa "bảo bối "
2. Tạo bộ đĩa Rescue 3 đĩa bằng NU trên windows
3. Giới thiệu hoạt động của chương trình quét vius DW,D2,BKAV.
4. Giới thiệu hoạt động của chương trình DISKEDIT

CÁC BƯỚC THỰC HIỆN ĐỂ ĐƯA 1 Ổ ĐĨA CỨNG VÀO HOẠT ĐỘNG

I- Format cấp thấp :

Dùng chương trình Low format Hard disk Trong RAM CMOS

II - Fdisk :

Dùng đĩa mềm bảo bối khởi động máy:

A:\> FDISK ↵

Xuất hiện màn hình của chương trình FDISK như sau :

MENU A:

1.Create DOS partition or Logical DOS Drive

Tạo partition khởi động của DOS hoặc tạo các ổ logic

2.Set active partition

Xác định partition nào là chủ động

3.Delete partition or Logical DOS Drive

Xoá bỏ 1 partition hoặc 1 ổ đĩa logic

4.Display partition information

Hiển thị thông tin partition

Nếu máy có nhiều ổ đĩa cứng vật lý thì FDISK có thêm tuỳ chọn thứ 5 là:

5.Change fixed disk

Thay đổi Fdisk ổ đĩa nào ?

Thực hiện :

Thường thường ta chỉ thực hiện chương trình Fdisk khi có sự cố ; khi

đó ta thường phải thường phải tạo lại bảng Fdisk . Nhưng trước khi tạo ta
phải xoá bỏ partition hoặc ổ đĩa logic cũ .Cách làm như sau:
Từ menu A chọn mục 3 ↵ . Đến đây xuất hiện menu B:

MENU B:

1.Delete primary DOS partition
Xoá vùng phân khu DOS chính
2.Delete Extended DOS partition
Xoá vùng phân khu DOS mở rộng
3.Delete loggical DOS Driver(s) in the Extended DOS partition
Xoá các ổ đĩa logic trong vùng DOS mở rộng
4.Delete Non-DOS partition
Xoá vùng phân khu phi DOS

Ta sẽ xoá từ mục 3, rồi đến mục 2 , cuối cùng là mục 1 là kết thúc việc xoá.
Quay trở lại menu A:
Chọn mục 1 .Xuất hiện menu C :

MENU C:

1. Create Primary DOS Partition
2. Create Extended DOS Partition
3. Create Logical DOS Driver(s) inte Extended Partition

Ta sẽ thực hiện việc tạo các ổ đĩa logic hoặc partition từ mục 1, rồi đến mục 2 ,
cuối cùng là mục 3 là kết thúc việc tạo lập(Ngược với quá trình xoá).
Sau khi thực hiện qua các bước trên xong cần khởi động lại máy để ghi
nhận các việc tạo lập.

III-Format cấp cao các ổ đĩa đã tạo lập bằng Fdisk:

Bước tiếp theo : Từ dấu nhắc ổ A >\ ta đánh lệnh : A>\ Format C:/s ↵
Sau lệnh này ổ C đã được Format và có các file hệ thống - có thể làm đĩa khởi
động được .

Ta tiếp tục Format các đĩa logic khác (nếu có) bằng các lênh :

A>\ Format D: hoặc A>\ Format E: ....

Sau cùng là bước cài đặt các chương trình ứng dụng.

IV- Khi không khởi động máy tính được bằng đĩa cứng :

Master boot nằm ở Sector 1(sector đầu tiên) Track 0 (rãnh đầu tiên) Side0(mặt đầu tiên) của đĩa cứng .Còn Boot record (Bản ghi khởi động) thường nằm ở sector 1 ,track 0 ,side 1 của đĩa cứng . Cả 2 phần này đều quan trọng và quyết định sự khởi động máy tính .

512 bytes đầu tiên nằm ở sector 1 ,side 0 của đĩa cứng gồm 2 phần:

- Phần đầu (từ offset 0 đến 1BDh) là master boot
- Phần sau (từ 1BEh đến 1FFh) là bảng Partition

Hiện tại đoạn mã master boot chuẩn chỉ có độ dài từ 0 đến 0DFh .Phần còn lại từ
offset 0E0h đến 1BDh chưa dùng , phần trống này có thể là nơi ẩn náu của các đoạn mã virus.

Master boot là 1 đoạn chương trình ngắn được nạp vào RAM từ địa chỉ 0:7C00h
và được thực thi khi khởi động máy .Master boot có nhiệm vụ sau :

- Kiểm tra bảngPartition để xác định xem Partition nào là chủ
- Nạp Boot Record của partition chủ đó vào bộ nhớ rồi chuyển điều khiển cho boot record của đĩa chủ để tiếp tục quá trình khởi động máy. (Thông thường boot record nằm ở sector 1 track 0 side 1 của đĩa C .Tuy nhiên ta có thể cài đặt boot record ở bất kỳ chỗ nào trên đĩa ,miễn là bảng partition phải trỏ được tới đó

Khi vius xâm nhập master boot của đĩa C có thể gây ra các hậu quả sau:

- Không khởi động được từ đĩa cứng vì master boot bị nhiễm virus và không chuyển quyền điều khiển cho boot record chủ hoặc làm sai lệch partition.
- Xuất hiện các thông báo lạ trên màn hình hoặc tiếng động ở loa trước khi nạp hệ điều hành.
- Tạo ra các vòng lặp để nạp hệ điều hành hoặc nhân bản virus vaò nhiều vùng nhớ khác nhau gây ra hiện tượng tràn ô nhớ hoặc làm giảm dung lượng của bộ nhớ (chiếm lĩnh các ô nhớ),

làm cho máy không cóa khả năng tải các chương trình lớn vào bộ nhớ

- Làm sai lệch CMOS

Đặc điểm của các đoạn mã virus nhiễm vào master boot là làm máy không khởi động được và để lại hậu quả trong quá trình chạy máy.Các đoạn mã này chỉ sử dụng các dich vụ của ROM-BIOS mà không dùng các hàm của DOS nên không chịu ảnh hưởng của các version DOS khác nhau..Khi ta format đĩa bằng lệnh FORMAT C:/S sau đó cài lại các chương trình sạch mà vẫn có virus vì virus trú ngụ tại master boot ở mặt 0 của đĩa ,trong khi đó partition của đĩa đã phân chia cho DOS vùng can thiệp từ mặt 1 trở đI .Lệnh format c:/s của DOS không format được mặt 0 của đĩa nên virus trên master boot không bị diệt . Tác động đến được chỉ có phần mềm Low level format

Nếu sau khi khởi động máy tính ta thấy quá trình không thực hiện được
ta cho 1 mềm khởi động vào và vẫn khởi động được máy thì nguyên nhân chỉ do trong đĩa cứng có trục trặc.

Nguyên nhân gây không khởi động được bằng đĩa cứng thường do:

1- Đặt tham số cho ổ đĩa cứng trong CMOS bị sai
2- Bảng Partition bị sai lệch do virus
3- Boot sector khởi động bị hỏng do virus
4- Các file hệ thống bị hỏng

Để phát hiện hỏng hóc thuộc phần nào ta có thể dựa vào 1 số thông báo lỗi sau của ROM BIOS:

- **Missing Operating System :**

Thông báo lỗi này cho biết không tìm thấy hệ điều hành . Nguyên nhân dẫn đến
lỗi này là do các nguyên số 1 hoặc 2 .Để khắc phục ta phải kiểm tra từng phần 1 , trước tiên là các thiết lập RAM CMOS . Cách thức làm:

- Reset lại máy tính
- Bấm giữ phím Del
- Trong màn hình SETUP chọn IDE AUTO DETECTION hoặc AUTO DETECT HARDISK và ấn Enter .

Chương trình sẽ tự động nhận diện ổ đĩa cứng .

- Tiếp theo chọn SAVE SETTING & EXIT ấn Enter

- Nếu máy khởi động bình thường là tốt . Nếu không hãy dùng đĩa "bảo bối"

khởi động sau đó dùng chương trình FDISK với tham số MBR như sau :

A:>\FDISK.EXE /MBR ↵

(Đây là 1 tính năng chưa được công bố chính thức của chương trình

fdisk có tác dụng chuẩn lại master boot record . Khi bảng này chưa hỏng nặng lệnh này vẫn có tác dụng)

- Sau đó tạo các file hệ thống cho đĩa cứng bằng lệnh :

A:>\ SYS.COM C: ↵

- **Invalid Partition Table :**

Trường hợp này bảng Partition đã bị hỏng nặng ,thường do bị nhiễm vius .

Nếu trước khi xảy ra thảm hoạ này ta đã có đĩa cứu hộ lưu được các thông tin của bảng Partition thì vẫn phục hồi được . Hoặc dùng 1 chương trình lưu master boot (Xem phụ lục) để phục hồi . Nếu không bắt buộc phải fdisk , format lại đĩa cứng rồi cài đặt lại phần mềm.

- **None system disk or disk error**

  Replace and press any key when ready...

Thông báo này xuất hiện trong trường hợp Boot Sector khởi động bị hỏng hoặc các file hệ thống bị hỏng . Trong trường hợp này ta chỉ cần khởi động bằng đĩa mềm rồi dùng lệnh SYS C: ↵ là được.

THỰC HÀNH

1. Thực hành format cấp thấp ổ đĩa cứng
2. Thực hành chia 1 ổ đĩa cứng thành nhiều ổ đĩa logic , xác định đĩa chủ động
3. Tìm nguyên nhân không khởi động được bằng đĩa cứng , khắc phục các trường hợp bị virus nhiễm vào boot record , partition table . Sử dụng chương trình DISKEDIT để soạn thảo đĩa.

XÁC ĐỊNH VÀ KHẮC PHỤC CÁC NGUYÊN NHÂN GÂY MÀN HÌNH KHÔNG SÁNG
KIỂM TRA BỘ NGUỒN

1-Khác nhau giữa 1 Monitor và 1 máy thu hình (TV):

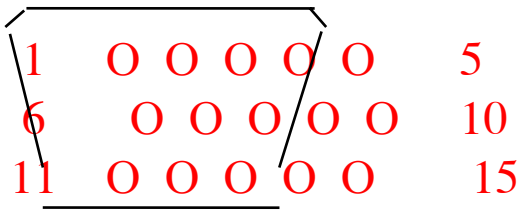
Monitor	TV
- Không có bộ thu tín hiệu truyền hình (Khuyếch đạI cao tần ,trung tần,tách hiệu sóng tách xung đồng bộ ...) - Xung tạo tín hiệu ,điều khiển tín hiệu dao động dòng ,mành được đưa từ CPU sang.	- TV có bộ thu tín hiệu truyền hình - Xung tạo tín hiệu dao động dòng ,mành được tạo ra trong TV

Màn hình TV không làm Monitor được vì :

Màn Monitor không sử dụng quét xen kẽ . Tần số dao động dòng của màn Monitor cao hơn màn TV 20% nên không tương thích.

2- Các tín hiệu từ CPU tới màn hình trên cáp nối :

ổ cắm phía sau CPU ,các chân tín hiệu như sau :



For The Monitor
VGA ,8514/A & XGA mapped
to the monitor

1 Red Video
2 Green Video
3 Blue Video
4 Ground
5 Self test
6 Red Ground
7 Green Ground
8 Blue Ground
9 No Connection
10 Digital Ground
11 Ground
12 Reserved (SDA for DDC)
13 Horizontal Sync.
14 Vertical Sync.
15 No Connection (SCL for DDC)

3-Điều kiện để việc hiển thị bình thường:

- Nguồn nuôi đủ.
- Mainboard tốt (Bao hàm cả CPU,BIOS tốt)
- Card màn hình tốt
- RAM tốt
- Màn hình tốt

Các điều kiện này phải đồng thời được đảm bảo thì việc hiển thị mới bình thường

Thiếu 1 trong các điều kiện trên là việc hiển thị của Monitor sẽ không thực hiện được.

4- Kiểm tra khi màn Monitor không sáng:

Khi bật điện máy tính ,nếu không thấy màn Monitor sáng ta phải:

- Vặn chiết áp Bright (Độ sáng)lên vì chiết áp có thể bị xê dịch đI và ở vị trí tối nhất.
- Kiểm tra nguồn xem có trục trặc gì không ? Nếu nghi ngờ có thể khởi động máy tính 1 lúc sau đó mới bật màn hình.

- Kiểm tra màn hình bằng cách thay sang 1 máy còn tốt
- Kiểm tra RAM bằng cách thay thử RAM chắc chắn còn tốt
- Kiểm tra Card màn hình bằng cách thay thử Card màn hình chắc chắn còn tốt
- Kiểm tra Mainbord sau khi đã qua các bước trên không có kết quả.

5- Khắc phục hiện tượng màn hình bị chuyển màu sắc :

Ta đã biết rằng : Các hình ảnh trên màn hình là tập hợp các điểm màu ,1 điểm màu trên màn hình màu là tổ hợp của 3 màu R,B,G theo 1 tỷ lệ nhất định . Nếu tỷ lệ này bị thay đổi thì việc hiển thị màu sẽ không đúng nữa : xanh quá , đỏ quá ,vàng quá ... thậm chí không lên được màu sắc nữa .

Việc kiểm tra hư hỏng phải bắt đầu từ đèn hình (các catot R,B,G) ngược về bộ giải mã màu . Đây là 1 việc yêu cầu chuyên môn cao . Trong thực tế ,may thay đại đa số các trường hợp chỉ cần kiểm tra bảng mạch điện trên đế đèn hình là đã tìm ra hư hỏng để khắc phục.

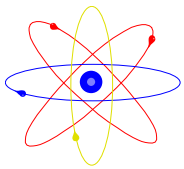
6-Kiểm tra tình trạng của bộ nguồn máy tính :

Để có thể sửa chữa được bộ nguồn ta cần có thêm các kiến thức của kỹ thuật điện tử . Còn để kiểm tra hoạt động của nó ta có thể đo các điện áp đầu ra cung cấp .Nếu các điện áp không có hoặc sai trị số thì có thể có hư hỏng trong bộ nguồn cần phải tháo ra thay bộ nguồn mới.

THỰC HÀNH

1. Tìm nguyên nhân không sáng màn hình do hỏng RAM
2. Tìm nguyên nhân không sáng màn hình do hỏng Card màn hình
3. Tìm nguyên nhân không sáng màn hình do hỏng nguồn

4. Tìm nguyên nhân không sáng màn hình do hỏng chiết áp bright
5. Tìm nguyên nhân không sáng màn hình do hỏng chip
6. Sửa chữa các trường hợp màn hình có màu sắc không chuẩn
7. Kiểm tra bộ nguồn máy tính.



PHẦN 4 CÀI ĐẶT CHƯƠNG TRÌNH

1-Các chương trình SCANDISK,DEFRAGMENTER
2-Cài đặt WINDOWS 98
3-Cài đặt MSOFFICE

THỰC HÀNH

1.Thực hành cài đặt các chương trình từ ổ chủ động lên đĩa chủ động
2.Thực hành cài đặt các chương trình từ ổ chủ động lên đĩa bị động
3. Kiểm tra hoạt động của các phần mềm với máy in

PHẦN 5 TỔNG THÀNH VÀ NÂNG CẤP MÁY TÍNH

I-Các yếu tố cần xem xét trước khi tổng thành máy:
1/ Hệ điều hành gì sẽ được cài trên máy ?

Tuỳ theo các hệ điều hành sẽ cài trên máy mà ta sẽ có các yêu cầu phần cứng khác nhau : Chẳng hạn xét trên phương diện bộ nhớ

Với DOS : Ta chỉ cần 1 cấu hình máy thấp : 286,386...1MB bộ nhớ

là đã chạy được nhiều ứng dụng

WINDOWS 3.X : Cũng chỉ cần 4MB nhớ là đủ

WINDOWS 98 : Cần 16MB

WINDOWS NT chạy đầy đủ các ứng dụng cần đến 64MB

2/ Sử dụng các chương trình ứng dụng gì ?

- Máy chỉ để dạy học các phần DOS , Pascal
- Máy chỉ để soạn thảo văn bản
- Ngoài các ứng dụng thông thường còn cần để truy nhập Internet , lập trình Java...cần có cấu hình mạnh .
- Máy để chơi trò chơi điện tử : Ngoài cấu hình mạnh còn cần có các phần cứng đặc biệt : Card hỗ trợ đồ hoạ , Card âm thanh ...
- Máy dùng cho các công việc điều khiển cụ thể : Có thể lại cần dùng loại tốc độ chậm hợp lý , dùng 386 lại tốt hơn 486...

3/ Các yếu tố về kinh tế :

- Lãi suất công việc.
- Khả năng đáp ứng tài chính trong khi tiến hành công việc.

II- Lựa chọn các bộ phận cấu thành:

1/ Chọn Mainboard :

- Mainboard là nền tảng của tốc độ vì vậy lựa chọn đúng đắn mainboard là điều rất cần thiết

- Ngoài các căn cứ như đã nói trên còn cần xem xét đến khả năng nâng cấp về sau

nếu có nhu cầu : Với 1 công việc cụ thể ta chỉ cần loại mainboard Pentium lắp chip 100mhz là đã đủ, nhưng với dự tính sẽ hoà nhập Internet ta cần mua loại lắp chip 166 và sau có thể thay chip 200 không cần thay mainboard.

(Thường các mainboard có thể cho phép cắm các chip có tốc độ trong 1 khoảng:

Bo Aristor chíp Pentium II tố độ từ 233-450mhz

Bo P2XBL cắm từ 266-450 mHz

Bo azza PT-61B cắm từ 233-450mHz

Bo MSI - 6116 dùng cho pentium II từ 233-400mHz

...Ta có thể tham khảo trong thuyết minh máy và cả các chuyên viên kỹ thuật ở nơi cung cấp thiết bị.

- Với các loại máy cũ nếu ta dùng mainboard 386 thì BIOS chỉ quản lý ổ cứng nhỏ hơn 540MB

- Nếu mainboard không có khe cắm PCI thì ta không cắm được các card đồ hoạ chuyên dụng mới sử dụng cho các ứng dụng đồ hoạ

- Từ 1 mainboard cụ thể đã lựa chọn ta có thể tiến hành các bước tiếp theo

2/Chọn chip :

- Khi mua mainboard thường ta mua luôn cả chip
- Thường thường như đã nói ỏ trên với 1 mainboard có thể cắm nhiều loại chip cụ thể ta phải tham khảo tài liệu
- Khi lựa chọn chip ta cũng cần lựa chọn loại quạt luôn,vì nếu quạt không đồng nhất với chip về hình dáng thì không thể gắn được vào chip để làm mát

3/ Chọn ổ cứng :

Theo yêu cầu dung lượng.

4/ Lựa chọn RAM theo khe cắm mở rộng

Các loại mới hiện nay đều dùng loại DIMM

5/ Chọn các thiết bị ngoại vi khác :

- Chọn ổ mềm
- Chuột
- Bàn phím
- Màn hình
- CD-ROM
- Các Card phối ghép

6/ Chọn nguồn :

Có các loại nguồn có công suất 180-240W

Để đảm bảo cho các phát triển ta nên chọn công suất của nguồn trên 200W

7/Chọn vỏ :

Tuỳ theo hình dáng kết cấu của mainboard để chọn vỏ .

III- Tiến hành:

- KIểm tra bộ nguồn :

Đo điện áp : Đây là công việc đầu tiên quan trọng cần làm .Ta đã biết rằng bộ

nguồn cung cấp các điện áp +12V,-12V,+5V,-5V cho các IC trên bảng mạch và các thiết bị ngoại vi. Rất có thể là bộ nguồn chúng ta mua về cho ra các điện áp không đúng yêu cầu. Nếu các đường cung cấp điện áp cho ra các điện áp nhỏ hơn định mức thì máy tính sẽ không hoạt động bình thường ; và nghiêm trọng hơn lại cho ra các điện áp lớn hơn định mức (do hỏng ổn áp chẳng hạn) kết quả thảm là thảm hại !

- Nối màn hình và card màn hình,RAM
- Nối bàn phím
- Bật điện kiểm tra :màn hình, mainboard,RAM,card màn hình,bàn phím:

Sau 3 bước này ,khi máy tính được bật lên nếu các bộ phận màn hình ,

mainboard,RAM,card màn hình , Chip vi xử lý tốt thì màn hình sẽ sáng . Kèm theo có các thông báo về thiếu đĩa cứng,đĩa mềm . Nếu màn hình không sáng ta phải áp dụng các kiến thức của bài xác định các nguyên nhân gây không sáng màn hình .

Sau đó chuyển sang bước tiếp.

- Nối card I/O
- Nối ổ mềm . Khởi động máy.
- Chạy SETUP thiết lập cấu hình máy.
- Khởi động ,xem xét các trục trặc :

Hết bước này mà không có gì trục trặc xảy ra thì ta đã phối ghép được card I/O và sau đó cả ổ mềm vào bảng mạch chính 1 cách trơn tru . Ta phải phối ghép dần dần từng thiết bị vào bởi nếu đưa ngay tất cả vào mà có vướng mắc ở 1 thiết bị nào đó sẽ rất khó xác định ; chưa kể 1 sai hỏng này lại có thể kéo theo sai hỏng khác

- Nối ổ cứng,chuột
- Chạy SETUP AUTODETECT HARD DISK
- Khởi động xem xét .

ổ cứng của chúng ta đã phải có cài hệ điều hành . Nếu máy khởi động thông suốt thì không có vấn đề gì .Nếu không khởi động

được ta phải xem lại ổ đĩa cứng là chính ,bởi máy của chúng ta đã khởi động bằng ổ mềm tốt rồi .
- Nối CD-ROM ,card âm thanh
- Chạy thử chương trình
- Chạy SCANDISK ,DEFRAGMENT :

Bước chạy SCANDISK ,DEFRAGMENT thực hiện sau khi các bước trên đã thông suốt nhằm sắp xếp tối ưu hoá đĩa cứng , đảm bảo cho máy tính của chúng ta hoạt động ở trạng thái tốt nhất. Công việc này nếu làm lần đầu tiên cũng tốn khá nhiều thời gian , vì các đĩa cứng hiện nay thường có dung lượng lớn.

IV-Nâng cấp :

1-Biện pháp Overclock: Biện pháp này cho kết quả khoảng 15-20% tốc độ nhưng hay gây hỏng chip nhất là với loại AMD-Kx ngày nay ít dùng.

2- Thay chip , thay mainboard: Bằng loại chip tốc độ cao hơn đúng chân.

* Những bước cần làm trước khi thực hiện:
- Xác định bo mạch chủ :

Xem trong thuyết minh máy.Nếu không có ; xem trên màn hình khi khởi động để biết số định danh BIOS .Ấn phím Pause để dừng màn hình và ghi lại số đó . Nừu BIOS có xuất xứ từ Award hoặc Ami hãy vào mạng Internet và tìm BIOS Page của Wim(www.ping.be/bios/),rồi đối chiếu phù hợp phiên bản BIOS của bạn với hãng sản xuất và model của bo mạch.
- Kiểm tra BIOS : Cần biết phiên bản của BIOS vì các chip của Intel và Evergreen có thể yêu cầu nâng cấp cả BIOS trong máy tính . Evergreen cung cấp cập nhật còn Intel thì không .Vào địa chỉ www.intel.com/overdrive/bios để xác định xem BIOS đang dùng có cần nâng cấp hay không . Việc nâng cấp BIOS tiến hành bằng cách khởi động từ đĩa mềm chứa chương trình cập nhật
- Kiểm tra tính tương thích của chip và bo mạch.

- Kiểm tra xem chân cắm của chip có vừa khe cắm trên bo mạch không .

* Thường thường khi thay các chip ta thường phải thay đổi các yếu tố sau :
- Đặt lại tốc độ CPU :
- Đặt lại hệ số nhân tốc độ làm việc CPU
- Đặt lại điện áp nuôi cho chip

Các công việc này được thực hiện nhờ việc thay đổi các chuyển mạch(Jumper) tương ứng. Thay đổi tổ hợp các Jumper như thế nào cần căn cứ vào thuyết minh của bảng mạch (mainboard)đang sử dụng .

Nhiều trường hợp ,khi thay chip khác loại ta còn phải đặt Jumper để tương ứng với loại chip sử dụng.

* Hướng dẫn học viên cắm các chuyển mạch (JUMPER) khi thay các chip khác nhau

3- Nâng cấp bộ nhớ :

Trước khi tiến hành lắp thêm các khối vi mạch nhớ (các thanh RAM)ta phải tìm mua đúng chủng loại (DIMM hoặc SIMM)để có thể lắp vừa vào máy của chúng ta

Tiến hành :

- Chạm tay vào vỏ sắt máy để khử tĩnh điện:
  (Máy đã phải nối đất ; nếu không không có tác dụng thậm chí còn nguy hiểm nếu máy bị rò điện)
- Kiểm tra lại bộ nhớ đang dùng:
  Các thông báo xuất hiện khi đang khởi động hoặc trong ***System Properties***
  Vào Start → Setings → Control Panel → System ↵
- Tìm vị trí bộ nhớ :
  Phân biệt khe cắm SIMM (72 chân)và DIMM (168 chân)
- Tháo bộ nhớ cũ (nếu cần):
  Hai ngón tay ấn xuống 2 lẫy giữ 2 bên thanh RAM . Thanh SIMM sẽ tự
  ngả ra , thanh DIMM sẽ tự trồi lên.

- Lắp bộ nhớ mới vào :

  Để lắp SIMM ta đưa thanh RAM vào hơi chéo xuống khe cắm ,rồi kéo đứng lên khi nghe thấy tiếng "cạch" vào khớp của 2 mấu giữ 2 bên là được

  Lắp DIMM chỉ cần ấn thẳng xuống hết cỡ rồi kéo 2 bên mấu giữ vào
- Khởi động lại máy tính :

  Quan sát nếu kích cỡ bộ nhớ tăng lên là đã hoàn thành .Cũng có trường hợp phải cập nhật lại trong BIOS

Các hình vẽ

4- Thay các thiết bị ngoại vi có tính năng kỹ thuật cao hơn:

*Lắp card âm thanh mới : Trên thị trường hay dùng loại Sound Blaster 16 ,32 hay 64

Tiến hành :

- Chạm tay vào vỏ sắt máy để khử tĩnh điện:

  (Máy đã phải nối đất ; nếu không không có tác dụng thậm chí còn nguy hiểm nếu máy bị rò điện)
- Xoá phần mềm sound card cũ :

Start → Seting → Control Panel → System → Devicer Manager. Nhấn dấu cộng bên cạnh tiêu đề " Sound ,Video and Game Controller". Rồi ấn Move

hình vẽ

- Tháo card cũ :

  Tắt máy tính ,giữ nguyên dây nối đất của máy để khử tĩnh điện .Mở vỏ máy .Tháo tất cả các dây cáp nối đằng sau sound card đang dùng (Các dây loa , micro,đường âm thanh vào,vv...) Trong hộp máy có 1 dây cáp âm thanh mỏng nối từ ổ CD-ROM đến sound card (hoặc đến board mẹ).Tháo đầu nối trên sound card.Tiếp theo tháo các vít giữ card và nhấc ra 1 cách cẩn thận.
- Lắp sound card mới :

  Trước khi lắp thêm card mới hãy nối nó với cap âm thanh của ổ CD-ROM .Vặn vít giữ card.Nối lại các dây nối ra ngoài.
- Cài đặt Driver và các ứng dụng của sound card:

  Khởi động lại PC ta thấy Windows thông báo là đang tìm phần cứng mới xuất hiện và cài phần mềm cho nó . Có thể phải đưa đĩa mềm hoặc đĩa CD-ROM vào cài đặt. Tiến hành các bước theo hướng dẫn trên máy.

hình vẽ

• Nghe thử :

Muốn kiểm tra sound card mới :

Chọn Start.Program.Accessories.Multimedia.Media Player.Nếu không có Media Player có thể cài bằng cách Start .Seting.Control Panel nhấn đúp chuột lên biểu tượng Add/Remove Programs và chọn mục Windows setup

Nhấn vào hộp kiểm tra kế mục Multimedia , nhấn OK rồi tiếp tục làm theo hướng dẫn Khi đã có Media Player trên màn hình , chọn file.Open và chọn 1 trong các tập tin âm thanh trong thư mục Media sau đó nhấn chuột vào nút Play -Có hình tam giác hướng lên .Nếu không nghe thấy âm thanh gì cả chọn Start.Setings.Control Panel nhấn đúp chuột vào biểu tượng System rồi chọn Device Manager. Nếu có dấu chấm than màu vàng xuất hiện bên cạnh " Sound ,video and game controller" là đã có trục trặc khi cài đặt phần cứng.chọn Start.Help tìm Hardware Conflit Traubleshooter rồi làm theo các hướng dẫn trên màn hình.

* Lắp đặt card đồ hoạ tốc độ cao : Các nhãn hiệu hay dùng :

ATI , Diamond , Hercules , Matrox , STB, Videologic

Tiến hành :

• Chạm tay vào vỏ sắt máy để khử tĩnh điện:

(Máy đã phải nối đất ; nếu không không có tác dụng thậm chí còn nguy hiểm nếu máy bị rò điện)

• Kiểm tra để đảm bảo máy đã có khe cắm PCI : Vì hầu hết các card loại này sử

dụng bus PCI

hình vẽ

• Chuẩn bị sao phòng dữ liệu trên ổ cứng tránh các rủi ro khi lắp thêm phần cứng mới
• Cắm card vào máy
• Cài đặt phần mềm :

Khởi động máy tính . Máy sẽ yêu cầu cài đặt phần cứng mới .Ta sẽ làm theo các yêu cầu và hướng dẫn trên máy

hình vẽ

• Thử và điều chỉnh board đồ hoạ :

Tiến hành chạy thử card mới với các chương trình ứng dụng . Nếu hình bị xé hoặc màu xê dịch dùng phần mềm của card để chọn lại độ phân giải thấp hơn ,ít màu hơn hoặc tốc độ làm tươi chậm hơn. Nếu không có kết quả tiếp tục thay đổi các thông số cài đặt khác.

THỰC HÀNH

1. Cho ra 1 yêu cầu giả định cấu hình PC :
   - Yêu cầu về tốc độ
   - Yêu cầu về bộ nhớ
   - Yêu cầu về khả năng nâng cấp
   - Hạn chế về tài chính...

Xác định các cấu thành từ yêu cầu đó :
   - Chọn mainboard
   - Chọn loại chip

- Chọn dung lượng bộ nhớ
- Chọn loại , dung lượng ổ cứng
- Chọn loại card màn hình...

2. Tiến hành lắp ráp máy này , khắc phục các trường hợp xảy ra
3. Thực hành thay đổi các Jumper trên máy tính ,kết hợp thay các chip để chuyển tốc độ của máy tính
4. Lắp thêm bộ nhớ ,
5. Thực hành lắp ráp ổ đĩa CD-ROM
6. Thực hành lắp ráp thêm card âm thanh , card đồ hoạ
7. Thực hành thay BIOS ,Flash ROM (nếu điều kiện vật chất cho phép)

TÀI LIỆU THAM KHẢO

1.The 80x86 Family Design , Programming and Interfacing
John Uffenbeck 1998

2. Kỹ thuật vi xử lý
Văn Thế Minh 1998

3. Cấu trúc máy vi tính
Trần Quang Vinh 1998

4. Bài giảng của Trung tâm tin học Thái Nguyên Phạm Việt Bình

5. Thuyết minh kỹ thuật của 1số loại máy tính trong thực tế

6. Tạp chí máy tính các năm 1997,1998,1999.

PHỤ LỤC:
THÔNG SỐ KỸ THUẬT CỦA MỘT SỐ LOẠI MAINBOARD
CARD ÂM THANH , CARD HÌNH ẢNH

- Aristor AM-608BX
  - Slot 1 for Pentium II 233.. 450MHz
  - Flash BIOS for 5V and 12V
  - Ultra-DMA/33
  - PCI bus Master IDE
  - AGP version 1.0
  - USB
- AZZA PT-6IB ATX mainboard
  - 1 PCI Local bus
  - Ultra-DMA/33
  - 1 port FDD
  - 2 Serial port (UART)
  - 1 Parallen port
  - 1 IrDA (Cong hong ngoai)
  - Support Pentium II 233..450mhz
  - Voltage for chip 1,8V to 3,5V
- DFI P2XBL
  - Support Pentium II 266..450mhz
  - 3DIMM
  - 3 PCI , 2 ISA, 1chung cho ISA and PCI
  - 2 Serial port ND16C550
  - 1 Parallen port SPP/ECP/EPPDB-25
  - 1 FDD ,IrDA
  - 2 USB
- P6BX-A+
  - 3 DIMM
  - 2 ISA 16 bit
  - 5 PCI 32 bit
  - 1 AGP
- GA-686BX
  - 1 AGP
  - Slot 1 for Pentium II 233..450mhz
  - 4 DIMM
  - Flash BIOS 2Mbit
  - UltraDMA/33

- GA-BX2000 Intel ® 440BX AGPset:
  - Supports Intel ® Pentium ® II/III and Celeron™ Processors
  - Slot1 with AGP Slot
  - Supports 66/75/83/100/112/124/133mhz system bus
  - Clock multiplier 3.0/3.5/.../6.5
  - 4 DIMM up to 1GB DRAM
  - Suspend - to -RAM (STR)
  - Includes DualBIOS™ technology
  - Power-on by K/B,P/S2 Mouse ,LAN & Modem
  - 3 Fan Power & Speed Detection Connectors
  - Intel ® LDCM ® Utility
  - TREN MICRO PC-cillin 98 anti-Virus utility (scan only)
- GA-6L7 Intel 440LX AGPset:
  - Supports Intel ® PPGA Celeron™ Processors
  - Socket 370 with AGP slot
  - Supports 66/75/83MHz system bus
  - Clock multiplier 4.5/5.0/6.0/.../6.5
  - 3 DIMM up to 768MB DRAM
  - 3 Fan Power & Speed Detection Connectors
  - TREN MICRO PC-cillin 98 anti-Virus utility (scan only)
- GA-5SMM SiS 530 AGPset:
  - Supports Intel ®MMX ™ AMD K6 ® - III IBM®/Cyrix® IDT ™ Processors
  - Supports 66/75/83/90/95/100/105/112/124/133MHz system bus
  - Clock multiplier 1.5/2.0/.../5.5
  - 3 DIMM up to 768MB DRAM
  - Support Ultra DMA 33/66 IDE Device
  - Power-on by K/B,P/S2 Mouse ,LAN & Modem
  - Chipset with built - Enhanced Graphics Engine(share or 4/8MB SDRAM)
  - TREN MICRO PC-cillin 98 anti-Virus utility (scan only)
- GA- 630 3Dfx Voodoo Banshee™
  - 128 bit 3D/2D Graphics Engine Voodoo Banshee from 3Dfx
  - 16MB 125 Mhz SGRAM for ultra high resolution

- 250mhz RAMDAC for Direct3D,Glide,and OpenGL under Windows 95/98
- Optimized driver for Glide,and OpenGL under Windows NT4.0

• GA-6BXE :
- Intel 440BX AGPset (AGP : Cổng đồ hoạ tăng tốc)
- Slot 1 & AGP slot support
- Supports 66/75*/83/100/112*/124*/133* MHz
- AC Recovery ON/OFF Control Build in BIOS
- Support 3 Fan Detection Connectors
- System Health Check & Report by BIOS during boot
- Intel LDCM^{R} Utility

• GA-6EX :
- Intel 440EX AGPset (AGP : Cổng đồ hoạ tăng tốc)
- Slot 1 & AGP slot support
- Supports 66/75*/83/100 MHz
- AC Recovery ON/OFF Control Build in BIOS
- Power on by Keyboard PS/2 Mouse,LAN,Modem & Others
- AUTO H/W detect CPU voltage

• GA-6EMMP :
- Slot 1 support
- Supports 66/75*/83 System speet
- 2 DIMM up to 256 MB DRAM
- Power on by keybord ,PS/2 Mouse,LAN,Modem & others
- 3 level ACPILED
- Ati RAGE Pro on board (2 oder 4MB SGRAM)
- YAMAHA PCI Sound 740 onboard
- AUTO H/W detect CPU Voltage
- Micro ATX From Factor (24,5x20cm)

• GA-5SG100 :
- SiS 5591 & 5595 AGPset
- Intel;,AMD,IBM,Cyrix Processor 90-550 MHz Ready
- Socket 7 & AGP Slot Support
- 60/66/75/83/100 Mhz main clock
- CPU Temperature sensor by Thermistor (Very close to CPU)

- System Healt Check & Report by BIOS When boot

• GA-612 : (Card man hinh)
  - Optimized drives for OpenGL 1.1 , DirectX 5.0 GDI and much more
  - Full 2X AGP Support(AGP VxD)
  - Full Sideband AGP Initiator Support
  - Full - Featured 3D Function Integrated 24bit 220 Mhz RAMDAC
  - 8 MB 100MMHz SDRAM on board

• SV-401 (Card màn hình)
  S3 Trio 64 VGA Card
  -PCI Bus 64 bit video memory hight ferfomance Windows accelerator
  -64 bit S3 Trio 64 VGA Chipset
  - Supports VGA feature connector for video ovelay on graphics and genlock capability
  - MediPro control panel sofware for changing Windows resolution,refresh rates and color depth
  - Support VESA VGA standard version 1.0a vertical refresh rates of 72Hz for flicker-free display
  - Provided 64 bit graphics engine to enhance Bit Block Transfers(BITBLT) withROPs
  point line draws ,trapezsoidal,polygon fills, clipping and hardware cusor

• EC505 (Card màn hình)
  - Using Trident 3dimage 9850 Chipset
  - AGP Bus (Rev 1.0) Compable
  - Support VESA DPMS ,VESA DDC
  - Support up to 83Mhz SGRAM
  - 2MB or 4MB Display Memory
  - Max.Resolution up to 1600x1200
  - Max Color Depht up to 24 bit
  - Integrated True Color 230 Mhz RAMDAC
  - NTSC/PAL TV Out Composite & Video (EC505V ver.only)

- 64 bit GUI Engine

- Servex VGA Card (Card màn hình):
  - 64 bit S3 Trio 64 VGA Chipset
  - PCI bus 16 bit video memory high peformance Windows Accelarator
  - MediPro control panel sofware for changing Windowsresolution,refresh rates and color depth
  - Supports VESA VGA standard version 1.0a vertical refresh rates of 72Hz for flicker-free display
- VGA Card (Card màn hình):

  EC237V S3TM Trio 64V2/DX Chipset PCI VGA Card
  - Using S3 Trio64V2/DX Chipset
  - PCI BUS (Rev 2.1) Compatible
  - Support DPMS which is idal for Enegy Start Program
  - Optimize Peformance by using EDO DRAM
  - 1MB or 2MB Display Memory(Optinal)
  - Max.Resolution up to 1600x1200
  - Max Color/Gradation: 16,7 milion (24bit)
- Servex Sound Card (Card âm thanh)
  - 16 bit YAMAHA OPL719 Chipset
  - Supports Plug and Play ISA 1.0a
  - Compatible with Sound Blaster , Sound Blaster Pro, Windows Sound System
- SV-204 (Card âm thanh)

  Yamaha - 718 Sound Card
  - 16 bit Yamaha OPL718 chipset
  - Supports Plug and play ISA 1.0a
  - Support software Wavetable ,General MIDI files compliant(32 voice channels /128 instruments)
  - Programable sample rate of up to 48khz(under Win95)
  - Built in high quality Sigma-Delta CODEC (Crystal 4231)
  - Compatible with Sound Blaster , Sound Blaster Pro,Windows 95
  - Support Industry standard PC game compatiblity
  - Build I Phone connector

- LXA800D:

Processor : - Intel Pentium II
- 233/266/300/333mhz
- Intel 440LX BIOS Chipset
- 1MB Flash EPROM (Support PnP,APM,ACPI,ATAPI,DMI&Windows 95)
- IDE AUTO LBA Mode Supports HDDs over 8.4GB
- Anti virus Protection Cache
- - L2 Cache is CPU Built inMemory
- - Up to 512 MB SDRAM or 1GB EDO
- Four 168 pins DIMMs(64bits)
- Synchronous DRAM (SARAM)& 3.3V Extended Data out (EDO)
- DIMMs Depth of 1MB,2MB,4MB,8MB,16MB and 32 MB
- Support ECC using parity DRAM Modules On bord I/O
- Support Tow PCI Enhandced IDEs PIO Mode3,Mode4 and Ultra DMA33 HDDs.Twin Headers for four IDE Devices Including IDE HDDs and CD ROMs Plug and Play
- Plug and Play Specification 1.1
- Plug and Play fos DOS ,Windows 3.X,and windows 95AGP
- AGP V1.0

- TXB820DS
- Intel Pentium P54C/Cyrix TM6x86,6x86MX/AMDTM K5,K6 90/100/120/133/150/166/180/200/233 Mhz
- Intel Pentium Processors with MMX TM Technology
- Socket 7 CPU Upgradability
- 125/150/166/180/200/233 Mhz with Penntium OverDrive Processor

- IDE AUTO LBA Mode supports HDDs up to 8,4 GB
- Direct mapped L2 write back Cache
- 256/512KB on-board Synchrounous Pipelined Burst SRAM 8MB to 256MB
- Plug and Play fos DOS ,Windows 3.X,and windows 95

• W-P6KL

- Slot 1 support Pentium®II Processor,233-300MHz
- AGP 66/133mhz slot x 1,PCI slot x 4 168 pins DIMM x 4 Upto 512MB ECC support
- Modem Ring - Power on/Soft Power Off,Hard Monitoring(CPU Volt /Smart Fan Status/System Temperature)-LM78,S/W LANDesk®Client Manager,Switching Volt Regulators
- ATX Double Decker(2lrDA/PS/2 Mouse & Keyboard/2USB/2S/1P/FDD 3Mode Upto 2,88)
-AWARD PnP PCI BIOS with ACPM , GreenPC ,DMI&Bootable from CD,SCSI, LS-120 & ZIP
- ATX Power Connector/ATX From factor 12” by 7,7 “

• W-P55TX2

- CPU : INTEL,Pentium P55C/MMX,AMD K5/K6,Cyrix/IBM M1-M2-75 -266mhz
- Chipset : INTEL 82439TX PCIset
- Modules : 72 pins SIMM x 4 for FP/EDO RAM & 168 pin DIMM x 2 for SDRAM
- Cache Memory : Onboard 512 KB Pipeline Burst SRAM
- Onboard I/O : Super Multi I/O for 1 FDD/2S/1P(ECP,EPP)/USB/FastIR
- Connector : Supports P2/2 Mouse & Standar AT style Keyboard
- BIOS : Award PnP & Flash EPROM System
- From Factor : 220mm x 220 mm Half AT size
- Function : Ultra 33 synchronous DMA Mode /ACPI/PC97. Supports SCSI/CD-ROM/LS 120/zip driver Boot-Up.An AT/ATX power connector & thermal sensor

• W-P55VT2-Rev.E

- CPU : INTEL,Pentium P55C/MMX,AMD K5/K6,Cyrix/IBM M1-M2-75 -266mhz
- Chipset : VIA 82585VPX97 PCIset(Intel TX level)
- Modules : 72 pins SIMM x 4 for FP/EDO RAM & 168 pin DIMM x 2 for SDRAM
- Cache Memory : Onboard 512 KB Pipeline Burst SRAM
- Onboard I/O : Super Multi I/O for 1 FDD/2S/1P(ECP,EPP)/USB/FastIR
- Connector : Supports P2/2 Mouse & Standar AT style Keyboard
- BIOS : Award PnP & Flash EPROM System.Supports DMI & Green PC
- From Factor : 220mm x 220 mm Half AT size
- Function : Ultra 33 synchronous DMA Mode /ACPI/PC98. Supports SCSI/CD-ROM/LS 120/zip driver Boot-Up.An AT/ATX power connector & thermal sensor

• W-P6KF
- CPU : INTEL,Pentium Pro 150-233MHz&PentuimII(Klamath)233-266MHz Slot 1 for P6/MMX & Pentium Pro
- Chipset : INTEL 8244FX PCIset
- Modules : 72 pins SIMM x 4 for FP/EDO RAM up to 512MB
- Onboard I/O : Super Multi I/O for 1 FDD/2S/1P(ECP,EPP)/2USB/FastIR
- Connectors : Supports P2/2 Mouse & PS/2 style Keyboard
- BIOS : Award PnP & Flash EPROM System and SCSI BIOS
- Power Management : Support hardware Sleep /Resum & SMM
- From Factor : 178mm x 305 mm ATX size
- Expansion Slot : 32 bit PCI Master bus x5,16 bit ISA busx3

• W-P55TX
- CPU : INTEL,Pentium /MMX,AMD K5/K6,Cyrix/IBM M2
- Chipset : INTEL 82430TX PCIset
- Modules : 72 pins SIMM x 4 for FP/EDO RAM & 168 pin DIMM x 2 for SDRAM
- Cache Memory : Onboard 512 KB Pipeline Burst SRAM

- Onboard I/O : Super Multi I/O for 1 FDD/2S/1P(ECP,EPP)/USB/FastIR
- Connector : Supports P2/2 Mouse & Standar AT style Keyboard
- BIOS : Award PnP & Flash EPROM System
- From Factor : 220mm x 280 mm Half AT size
- Function : Ultra 33 synchronous DMA Mode /ACPI/PC97/ECC Parity Check

• W-P55TV2
- CPU : INTEL,Pentium /MMX Technology,AMD K5/K6,Cyrix/IBM M1-M2-
- Chipset : INTEL 82430VX PCIset
- Modules : 72 pins SIMM x 4 for FP/EDO RAM & 168 pin DIMM x 2 for SDRAM
- Cache Memory : Onboard 512 KB Pipeline Burst SRAM
- Onboard I/O : Super Multi I/O for 1 FDD/2S/1P(ECP,EPP)/USB Connector
- Connectors : Supports P2/2 Mouse & Standar AT style Keyboard
- Expansion Slot : 32 bit PCI Master bus x3,16 bit ISA busx4
- BIOS : Award PnP & Flash EPROM System
- From Factor : 220mm x 260 mm Half AT size

• MS-5156 Intel 430TX PCIset B.AT mainboard
- Support Intel 75 to 233mhz Pentium Processor with MMX Technology,or AMD K6 and Cyrix M2 Processor
- Vcore Voltage :2.0V to 3,52V for future 266/300mhz CPU
- PC97/ACPI Compliant
- Modem ring Wake-Up ,Soft Power Off
- Support AT/ATX power supply connector
- Optional Hardware Monitor
- System Voltage Monitor
- CPU fan Speed Monitor
- 4SIMM+2DIMM Support SDRAM,EDO ,FPDRAM,Max 256 MB Memory on board
- Support Ultra DMA33Synchronous DMA mode

- System boottable from LS-120 ,ZIP,CD ROM through BIOS setup
- 3 ISA and 5 PCI slots. 25cmx22cm BabyAT

• MS-5145 B.AT TX1 mainboard
- Support Intel 75 to 233mhz Pentium Processor and Pentium
- Processor with MMX Technology,or AMD K6
and Cyrix M2 Processor
- Support SDRAM,EDO ,FPDRAM,Max 256 MB Memory on board
- Peripherals protection through Poly fuse
- CPU fan on/off Control
- Support Ultra DMA33Synchronous DMA mode transfer up to 33MB/sec
- Support PIO Mode4 and Bus master IDE ,transfer up to 14MB/sec
- Dual channel ,support independent Timing of up to 4 IDE HDD/CD ROM Devices
- 26,5cmx22cm BabyAT

• MS-5146
Baby SI13 Mainbord
SIS 5571 Trinity Chipset

• MS-5164
Baby AL7 Mainboard
Ali Aladdin 4+chipset

• MS-6111 ATX LX1 Mainboard
- Support Intel 233 to 266MHz or even faster Intel Pentium II Processor with MMX Technology
- Modem ring Wake-Up ,Soft Power Off
- Peripherals protection through Poly fuse
- With NS LM-78 Controller on board(Intel Soft Ware LDCM) Hardware Monitor
- Processor Temperature Monitor
- Chassis Intruction Monitor (Reserved)
- Fan Speed Monitor

- fan Speed Control on CPU Power supply and System fans
- Supports a maximum memory size of 512 MB with SDRAM , or 1GB with EDO/FP DRAM
- System boottable from LS-120 ,ZIP
through BIOS setup
- Support Ultra DMA33Synchronous DMA mode transfer up to 33MB/sec
- Provide 1 AGP slot ,support 66/133mhz 3.3V devices(30% to 50% faster than PCI)
- Keyboard Password Wake-up (for Winbond 977TF V.C or later version)
- MSI Super Package CD (3 in 1)

SƠ ĐỒ CHUYỂN CÁC CẦU NỐI (JUMPER) THIẾT LẬP CÁC CHẾ ĐỘ LÀM VIỆC CHO CPU CỦA 1 SỐ MAINBOARD

Bo mạch 80486 PCI Green Mainboard
Bo mạch TX98
Bo mạch SP-586 TB
Bo mạch 486 PCI/ISA MP060
Bo mạch Pentium II /Celeron 440EX

(Các sơ đồ nối Jumper)

CHƯƠNG TRÌNH NẠP MASTER BOOT KHI BỊ VIRUS XÂM NHẬP

Để tránh những điều đáng tiếc ,trong khi máy còn đang "sạch" , bạn nên sao lưu master boot và partition của đĩa cứng ra 1 đĩa mềm khác để khi cần có thể nạp lại.

Việc nạp lại master boot trong bất kể tình huống nào cũng có tác dụng làm "tươi master boot và loại đI mọi "rác rưởi" dính vào master boot .

Tiến hành khi máy còn đang sạch :

Bước 1 :

Vào chương trình Debug để nạp Sector = 1,Track=0,Side=0,mã ổ đĩa =80h vào 1 vùng nhớ nào đó dành cho người sở dụng (chẳng hạn chọn 4000:0) bằng các lệnh Debug sau:

C:>\ debug ↵
A 100 ↵
MOV AX,0202 ↵
MOV DX,0080 ↵
MOV CX,0001 ↵
MOV DI,4000 ↵
MOV ES,DI ↵
MOV BX,0 ↵
INT 13 ↵
INT 20 ↵
↵
G ↵

Sau lệnh G(Go) 512 bytes của master boot và bảng Partition sẽ được nạp vào bộ nhớ từ địa chỉ 4000:0

Có thể xem 512 bytes này bằng lệnh D(Dump) :

D 4000:0 L200 ↵

Bước 2 :

Chuyển 512 bytes từ vùng nhớ 4000:0 sang vùng CS:100 bằng lệnh M(Move):

M 4000:0 L200 ↵

Bước 3:

Lưu 512bytes trên vào tệp MastBoot.luu bằng lệnh :

N MastBoot.luu ↵
R CX ↵
: 200 ↵
W ↵

(Lệnh R (Register) để nhập vào thanh ghi CX số hexa 200h = 512 bytes)
Sau 3 bước này trên đĩa C: đã có file MastBoot.luu lưu trữ bảng Partition . Đưa file này vào đĩa "cứu hộ "cùng với file debug.exe . Khi gặp trục trặc ở master boot ta dùng đĩa cứu hộ để khởi động .Sau đó chạy debug ↵ . Rồi nạp lại master boot như sau:

```
A 100 ↵
MOV AX,0301 ↵
MOV DX,0080 ↵
MOV CX,0001 ↵
MOV DI,4000 ↵
MOV ES,DI ↵
MOV BX,0 ↵
INT 13 ↵
INT 20 ↵
↵
G ↵
```

Sau lệnh này master boot lại được ghi từ file MastBoot.luu .Trở lại "sạch sẽ".

Chương trình này đã được thử nghiệm để phục hồi masterboot có hiệu quả

khi bị vi rut nhiễm vào boot sector .

CHƯƠNG TRÌNH CẤT GIỮ VÀ PHỤC HỒI THÔNG TIN TRONG CMOS

```
Program CMOS_Save_Restore;
uses crt,dos;
const FileName='CMOSINFO.SAV';
Var giatri,temp,i:byte;
    F:file of byte;
    {Thu tuc in thong bao loi}
 Procedure Err_Handle;
```

```
  Begin
    Write('Error ! Program Terminated');
    Halt(1);
  End;
  { Ham doc va ghi CMOS vao file}
Function Read_CMOS:integer;
  Begin
    Assign(F,FileName);
    {$I-} {Tat kiem tra vao ra de bat loi }
    Rewrite(F);
    {$I+}
    If IOResult<>0 then Read_CMOS:=-1
    Else
      Begin
        For i:=0 to 127 do
          begin
            Port[$70]:=i;
            giatri:=Port[$71];
            Write(F,giatri);
          end;
        Read_CMOS:=0;
        Close(F);
      End;
   End;
   { Ham doc du lieu tu file & ghi vao CMOS}
Function Write_CMOS:integer;
   Begin
    Assign(F,FileName);
    {I-}
    Reset(F);
    {I+}
    If IOResult<>0 then Write_CMOS:=-1
    Else
      Begin
        For i:=0 to 127 do
          begin
```

```
            Read(F,temp);
            giatri:=temp;
            Port[$70]:=i;
            Port[$71]:=giatri;
            Port[$70]:=i+128;
            Port[$71]:=giatri;
          end;
        Close(F);
        Write_CMOS:=0;
      End;
    End;
    {Thu tuc hien man hinh tro giup}
Procedure Help;
  Begin
    Writeln('Usage:CMOS  S|s  -  Save  CMOS  Infor  to  file
CMOSINFO.SAV');
    Writeln('   CMOS W|w  - Restore CMOS Info');
    Writeln(' Press any key to continue ...!');
    Readln;
  End;
    {Main Program}
BEGIN
  Clrscr;
  Writeln('CMOS Info - Save & Restore Utility');
  Writeln('Cread Date 09/10/1999');
  Writeln('--------------------');
  If (ParamCount=0) then Help;
  If  (ParamCount=1)  and   (ParamStr(1)='W')or  (ParamStr(1)='w')
then
    Begin
      Write_CMOS;
      If Write_CMOS=-1 then Err_Handle
      Else Writeln('CMOS Updated . OK !');
    End;
     If  (ParamCount=1)  and   (ParamStr(1)='S')or  (ParamStr(1)='s')
then
```

```
    Begin
     Read_CMOS;
     If Read_CMOS=-1 then Err_Handle
     Else Writeln('CMOS Saved . OK !');
    End;
END.
```

Cấu Trúc Máy Tính & ASM

Created by mercury

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

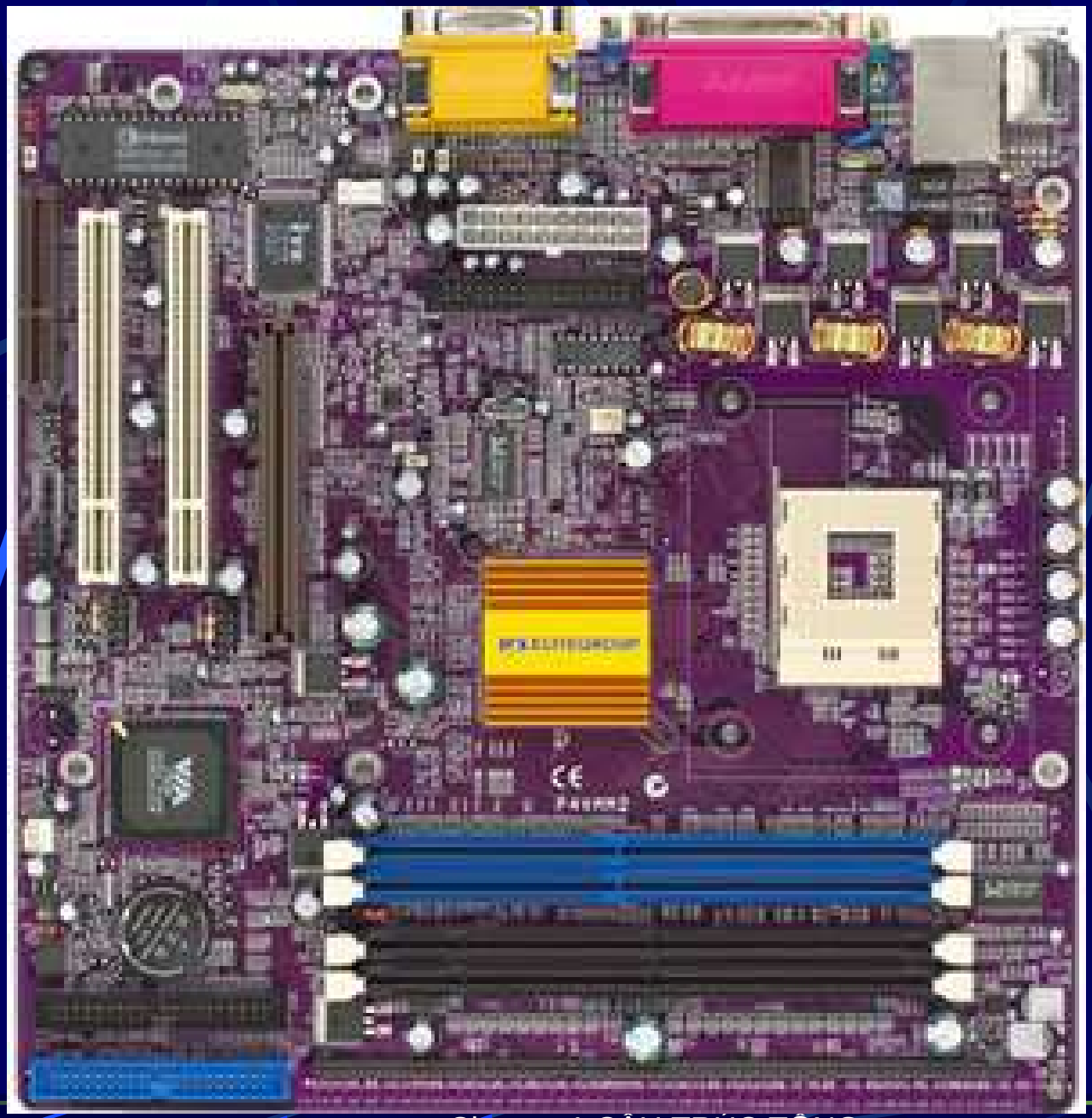
Mục tiêu :

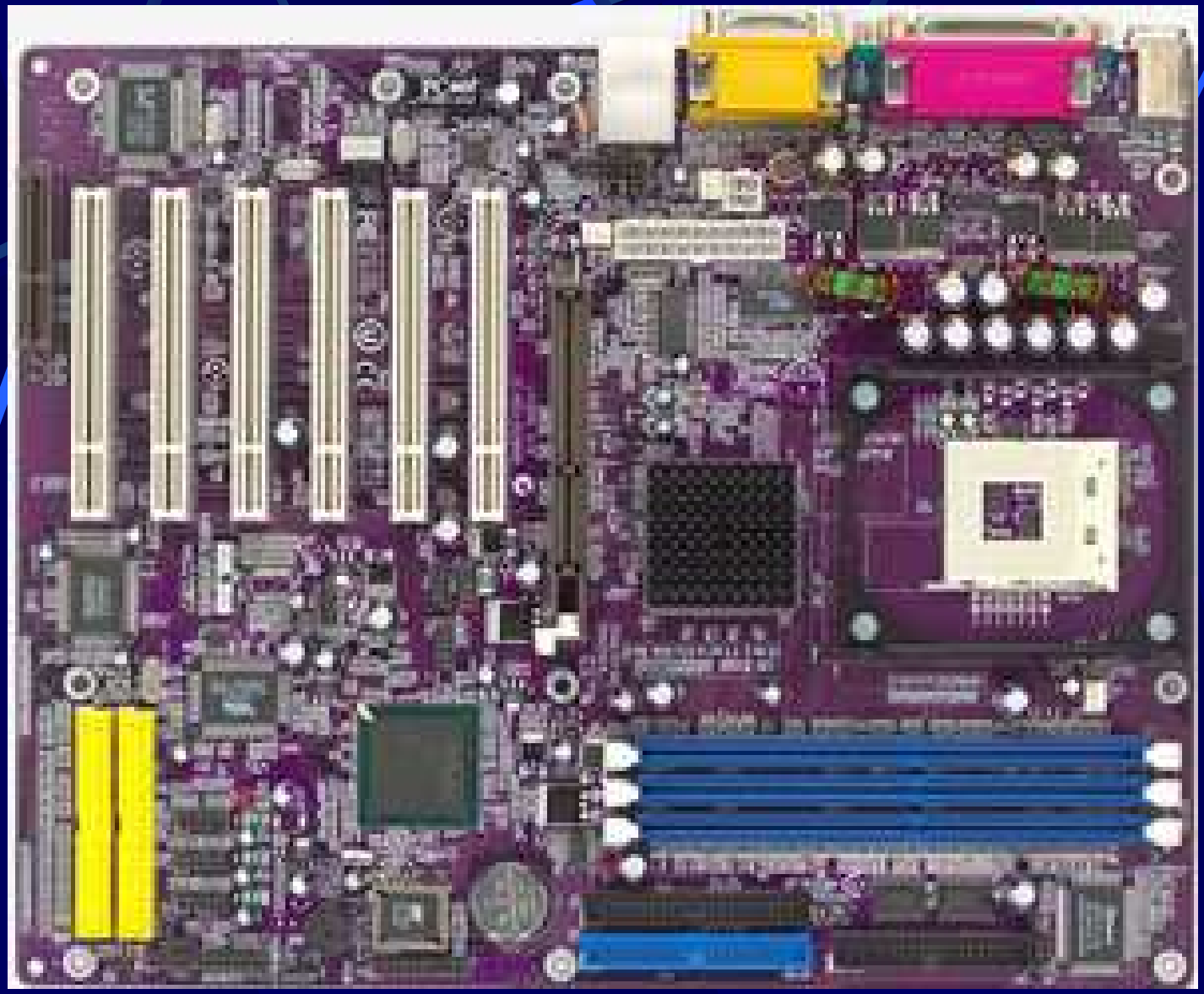
- Nắm được tổng quan về cấu trúc máy tính.
- Hiểu về Máy Turing & Nguyên lý Von Neumann
- Biết sơ đồ khối chi tiết của máy tính
- Nắm nguyên lý hoạt động máy tính
- Biết các component của máy tính : Processors,Memory,Input/Output devices,Bus

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

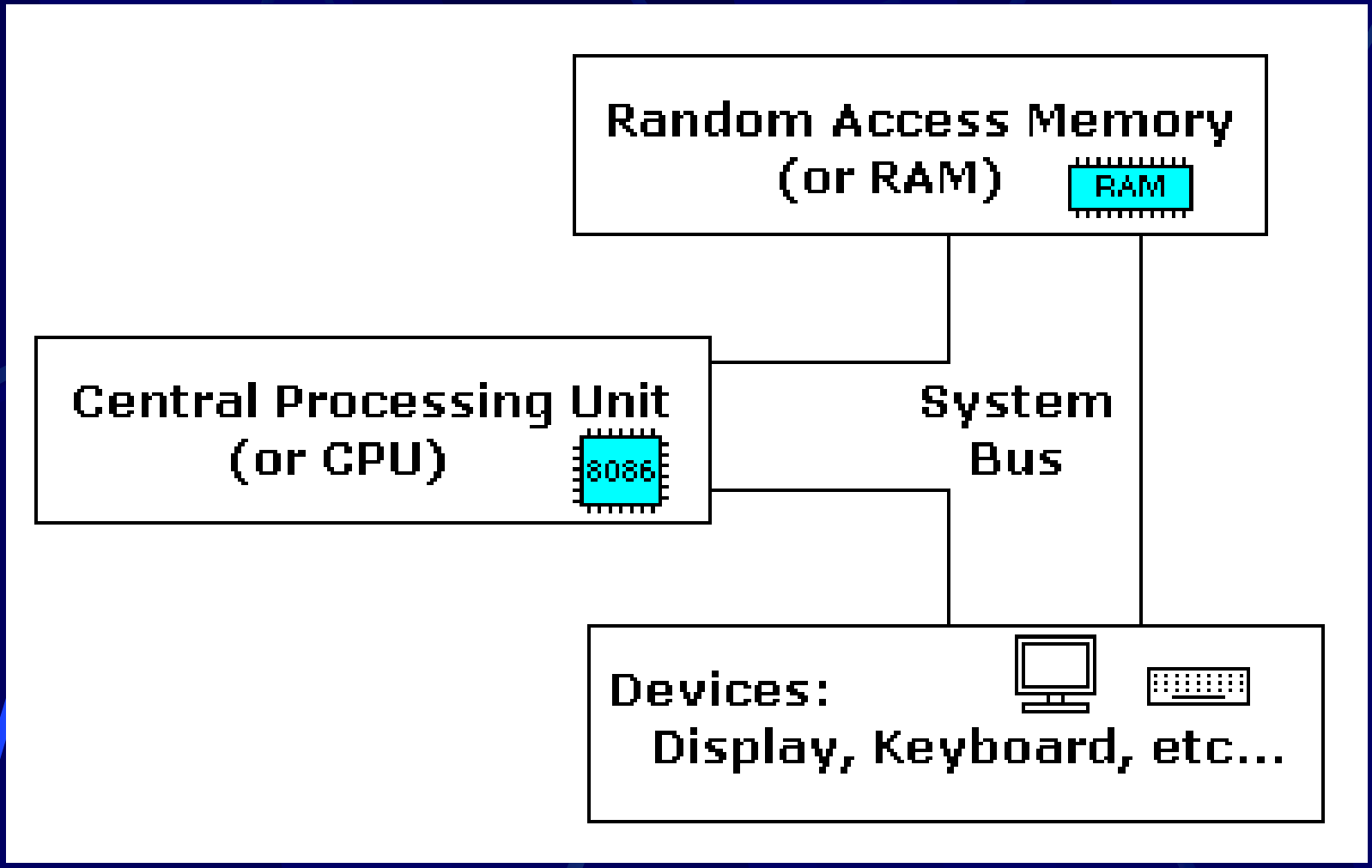
2+3/4*3-5=?

……

……..

……

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**

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

Nguyên lý xây dựng MT

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

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

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

Mạch điện trong MT

Trong MT có những loại mạch điện nào ?

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

Ex : Mạch giải mã địa chỉ

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

Ex : mạch cộng, trừ, nhân , chia

Nguyên lý Turing

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.

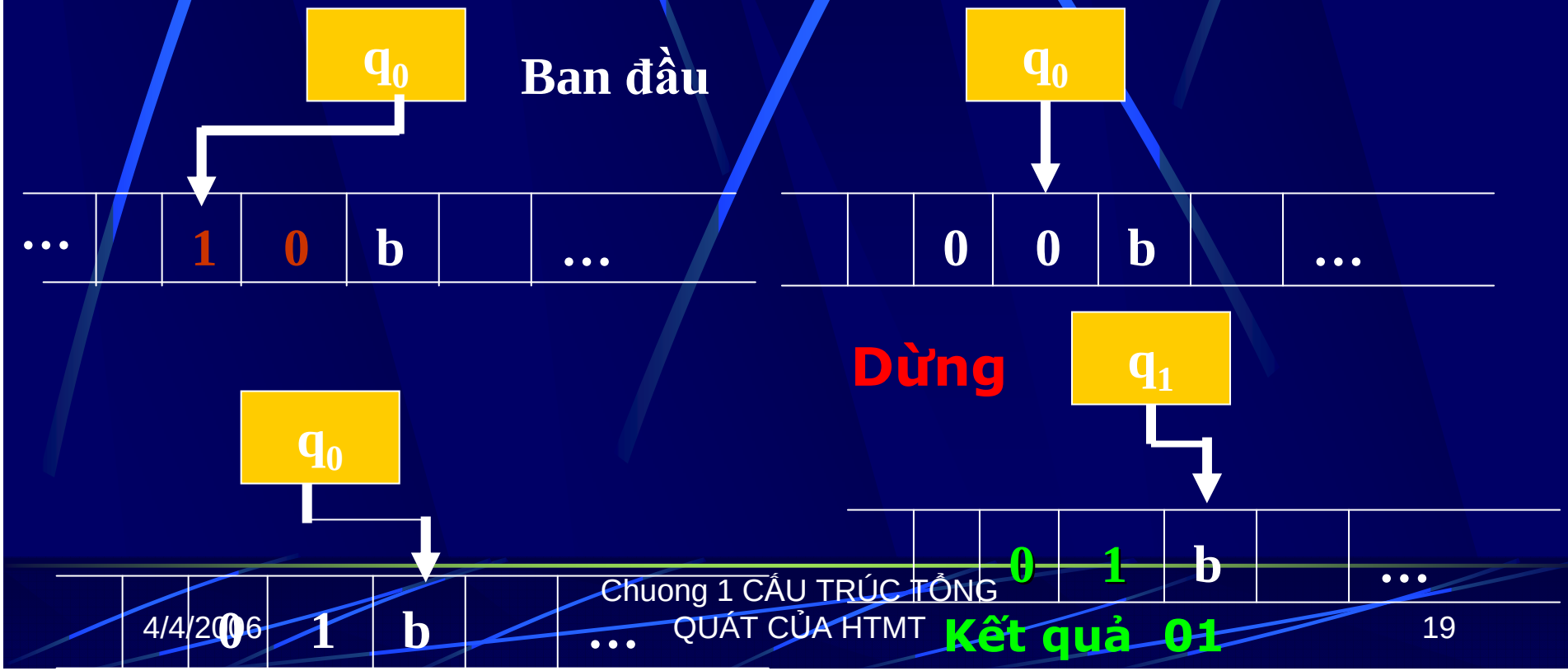
- Trạng thái trong ban đầu của máy là q_0 .
  - Đầu đọc/ghi ở ô chứa ký hiệu đầu tiên của chuỗi ký hiệu nhập. Trong quá trình hoạt động, sự thay đổi dữ liệu trên băng, sự dịch chuyển đầu đọc ghi và sự biến đổi trạng thái trong của máy sẽ diễn ra tuân theo các lệnh thuộc tập lệnh của máy tùy theo trạng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

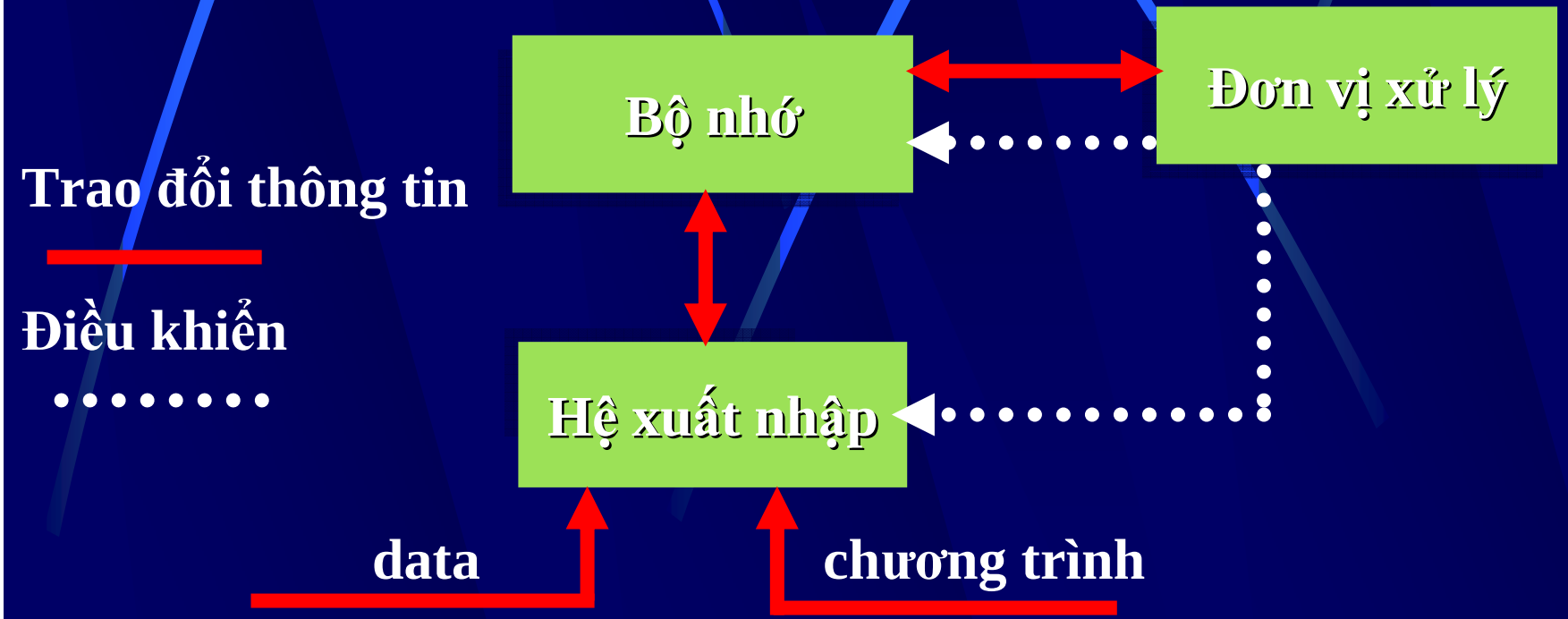


Nguyên lý VonNeumann

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.

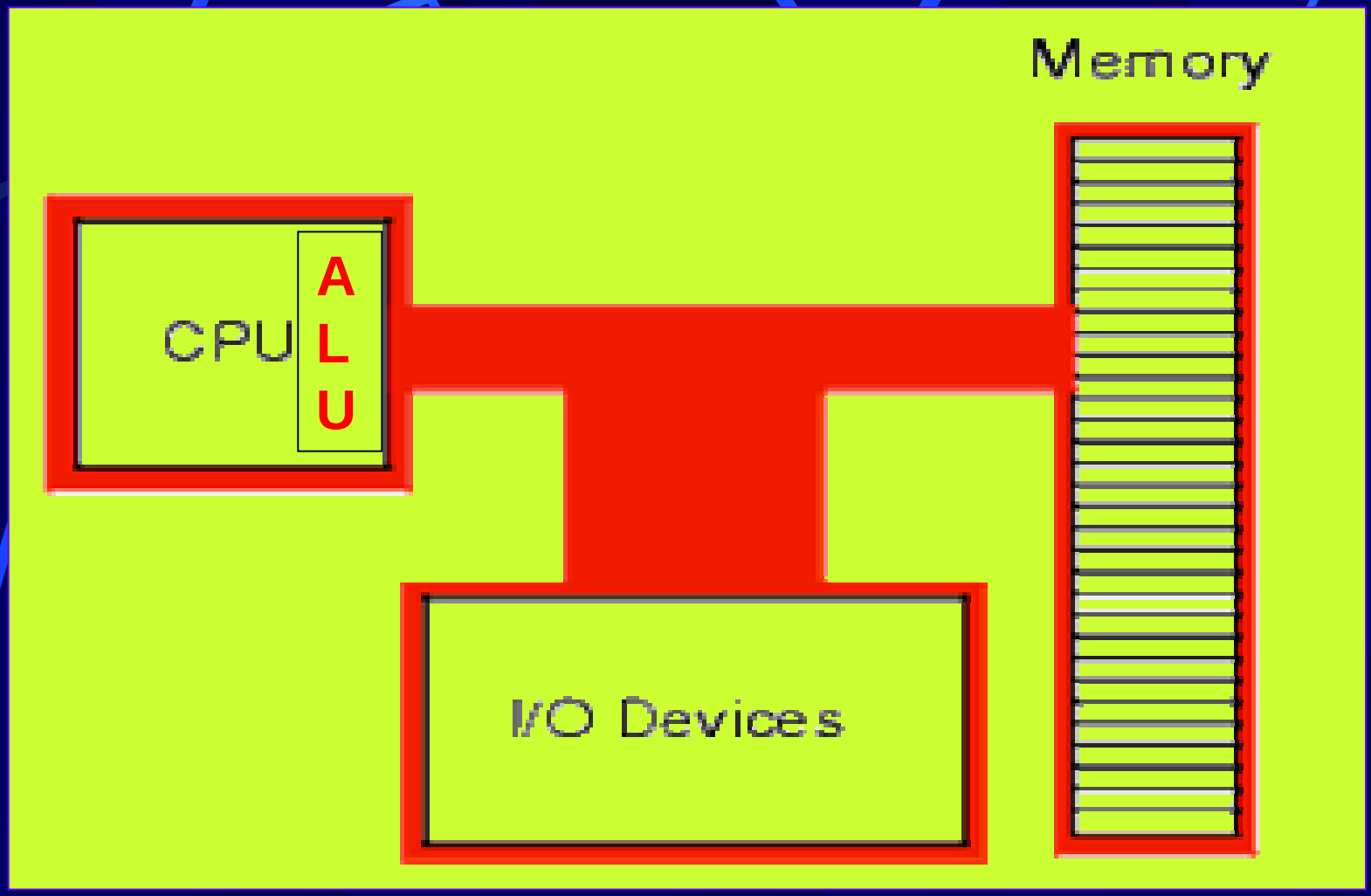
- Bộ nhớ chia làm nhiều ô, mỗi ô có 1 địa chỉ (đánh số thứ tự) để có thể chọn lựa ô nhớ trong quá trình đọc ghi dữ liệu. (nguyên lý định địa chỉ)

Nguyên lý Von Neumann (cont)

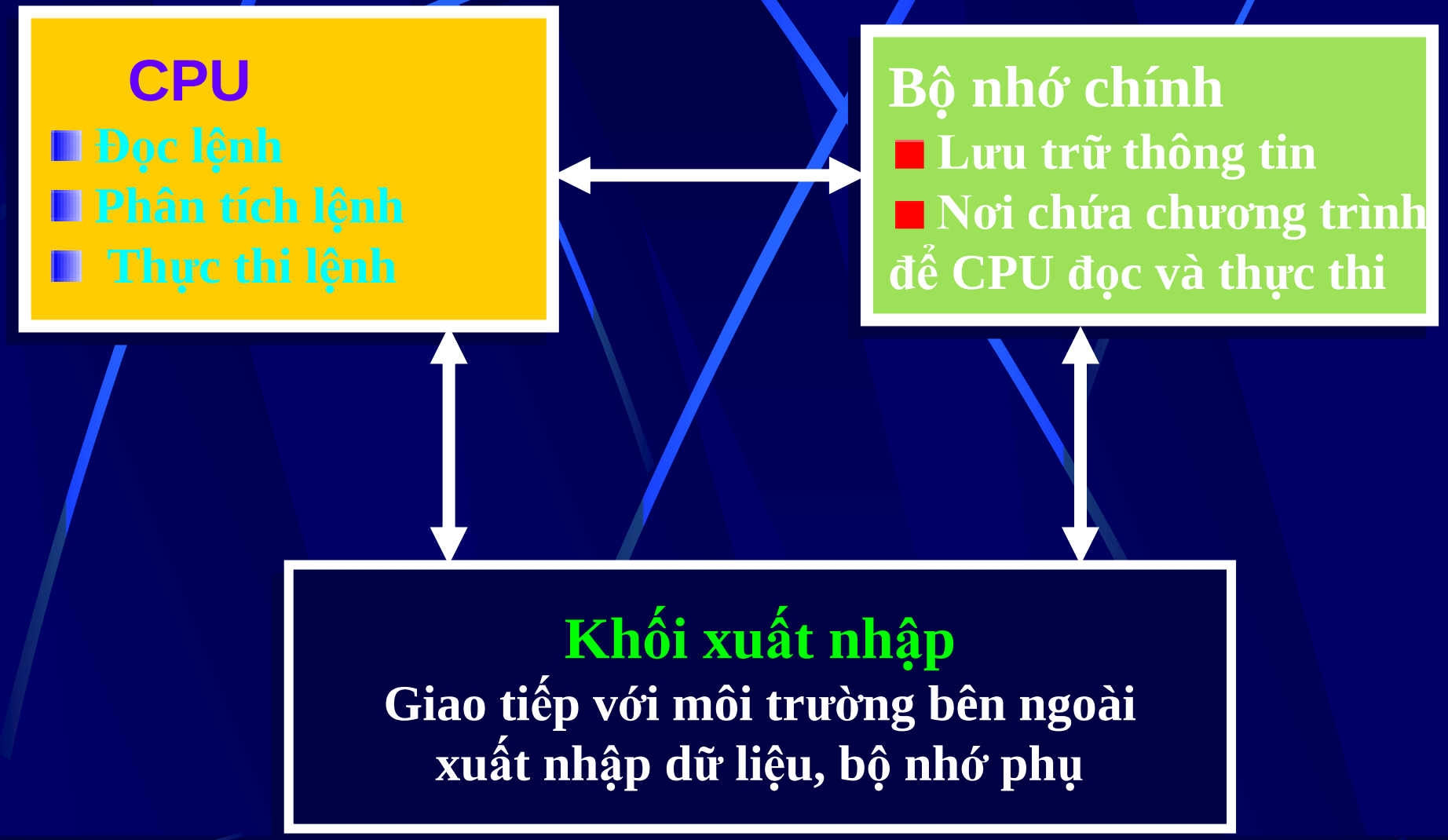
- Các lệnh được thực hiện tuần tự nhờ 1 bộ đếm chương trình (thanh ghi lệnh) nằm bên trong đơn vị xử lý.

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

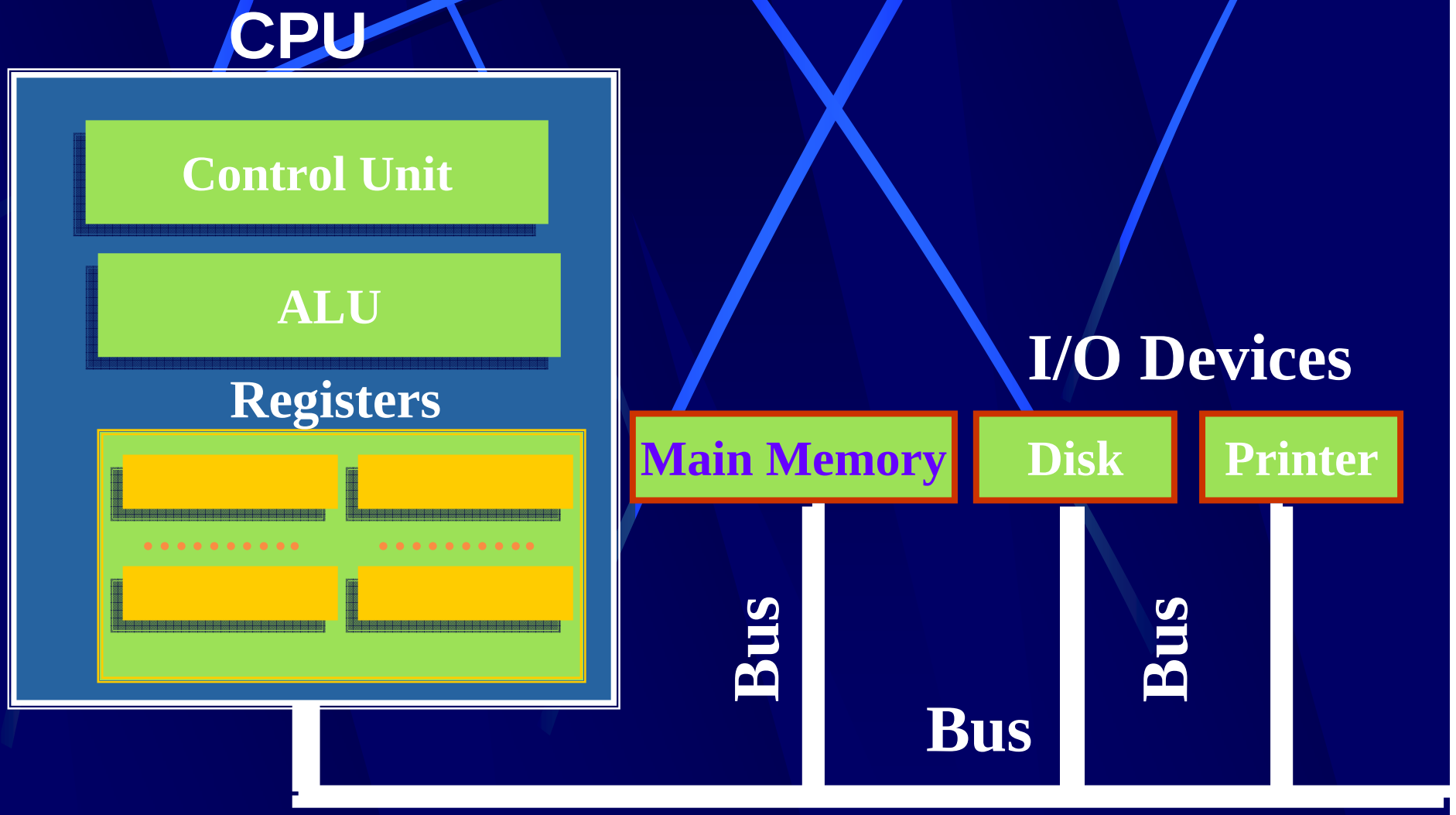
Typical Von Neumann Machine



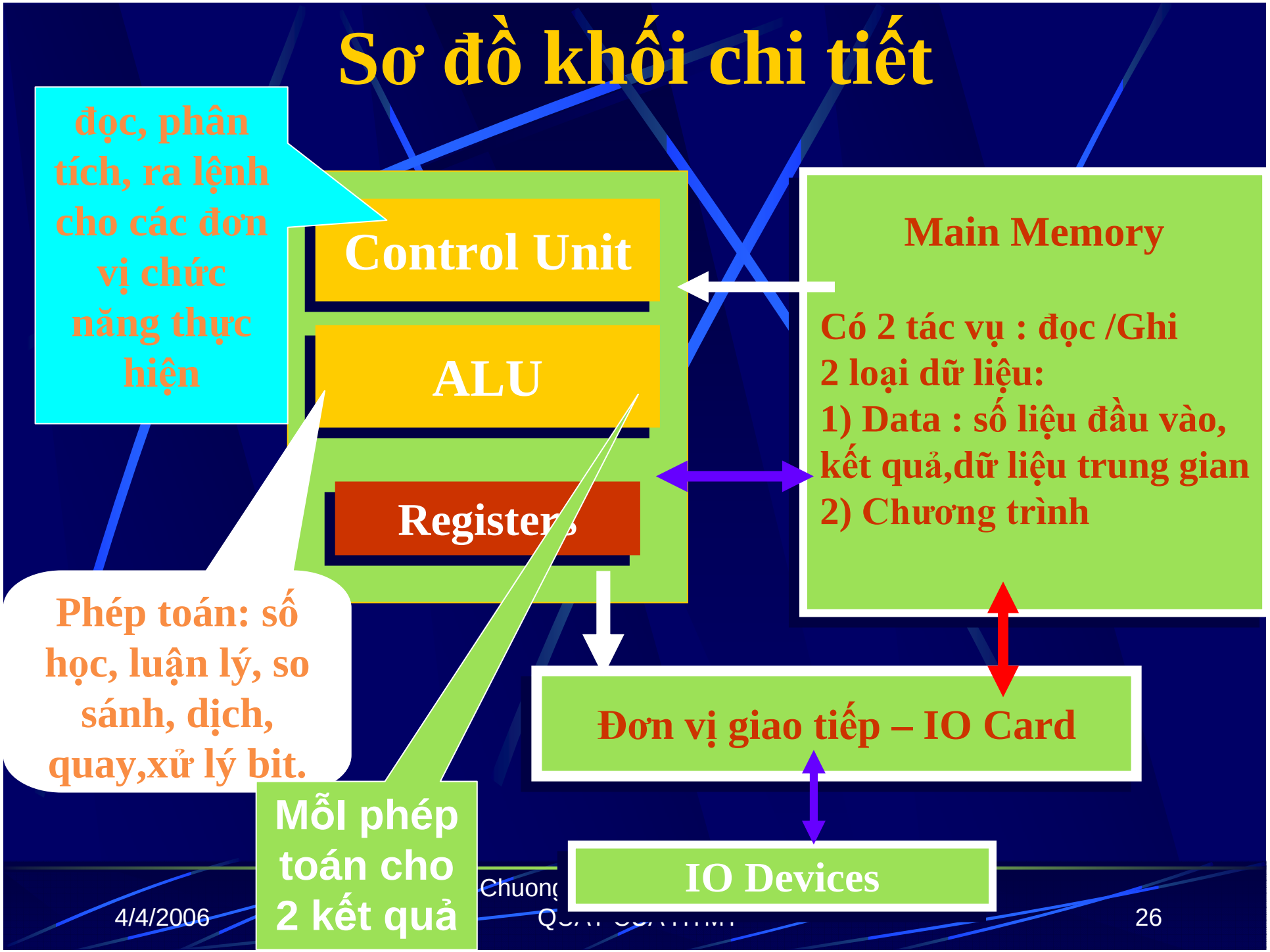
Nguyên lý hoạt động MT



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.

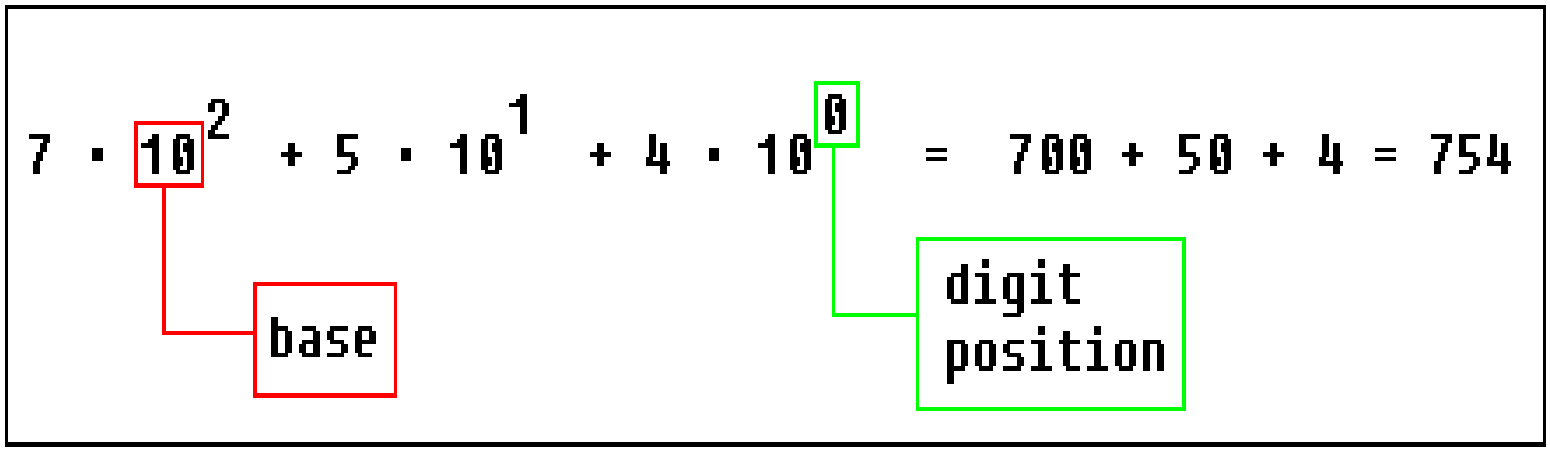
Hệ thống số

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 :

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

base

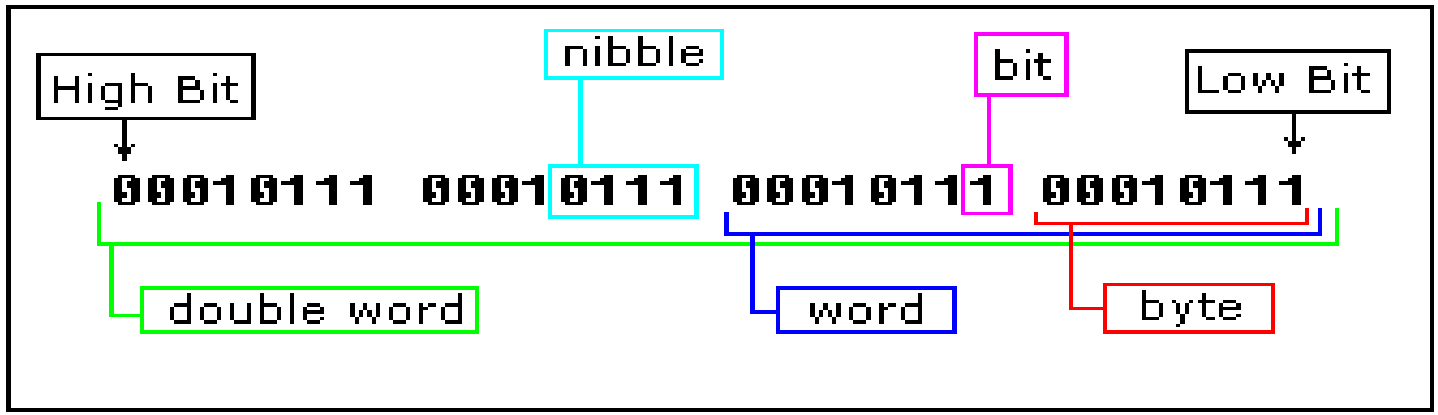
digit position

Binary System

MT không thông minh như con ngườI,nó dùng trạng thái của điện tử :
on and off, or 1 and 0.
MT dùng binary system, binary system có 2 digits:
0, 1

Như vậy cơ số (base) là 2.

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



Hexadecimal System

Hexadecimal System

Hexadecimal System dùng 16 digits:

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

do đó cơ số (**base)** là **16**.

Hexadecimal numbers are compact and easy to read.
Ta dễ dàng biến đổI các số từ binary system sang hexadecimal system and và ngược lại, mỗi nibble (4 bits) có thể biến thành 1 hexadecimal digit :

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

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 từ 10 ➔ hệ 2

Đổi từ hệ 10 ➔ hệ 2 :

Ex : 12d = 1100b

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

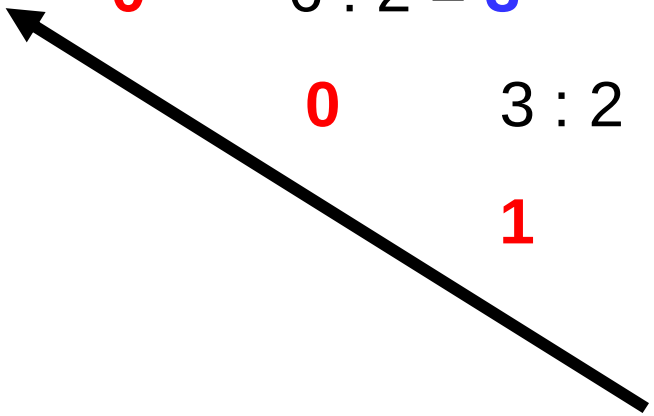
12 : 2 = **6**

0 6 : 2 = **3**

0 3 : 2 = **1**

1 1 : 2 = **0** **dừng**

1



Chuyển hệ từ hệ 2 ➔ hệ 10

Đổi từ hệ 2 ➔ hệ 10 :

Ex : $1100_b = ?_d$

Cách đổi : $\Sigma a_i * 2^i$ **với i ∈ 0...n**

a là ký số của số cần đổi.

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

↓

a

Chuyển hệ từ hệ 10 ➔ hệ 16

Đổi từ hệ 10 ➔ hệ 16 :

Ex : $253_d = ?_h$

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

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

↓

Chuyển hệ từ hệ 2 ➔ hệ 16

Đổi từ hệ 2 ➔ hệ 16 :

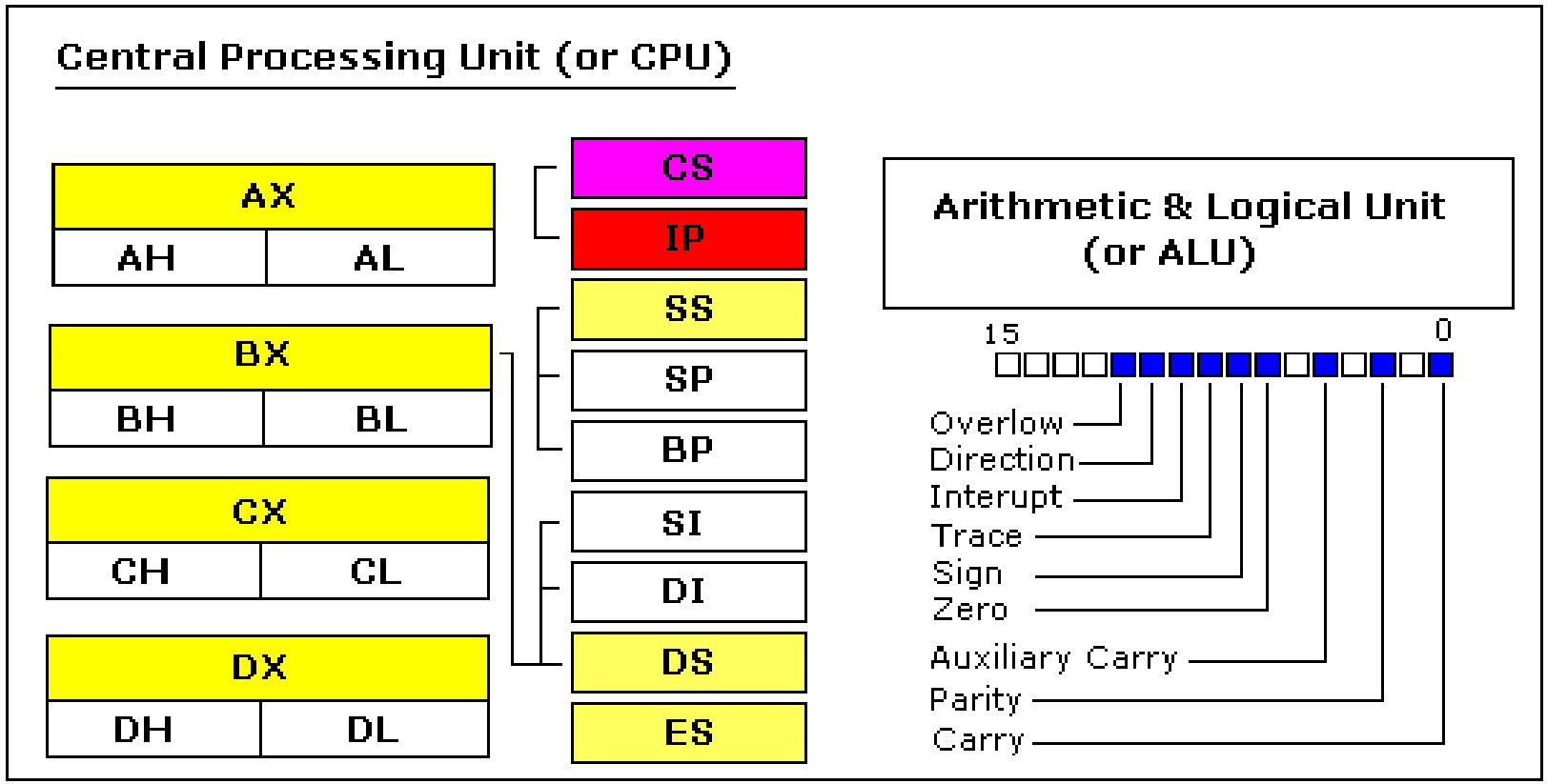
Ex : $101011010_b = ?_h$

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

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

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 lî trung ŧm –

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

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

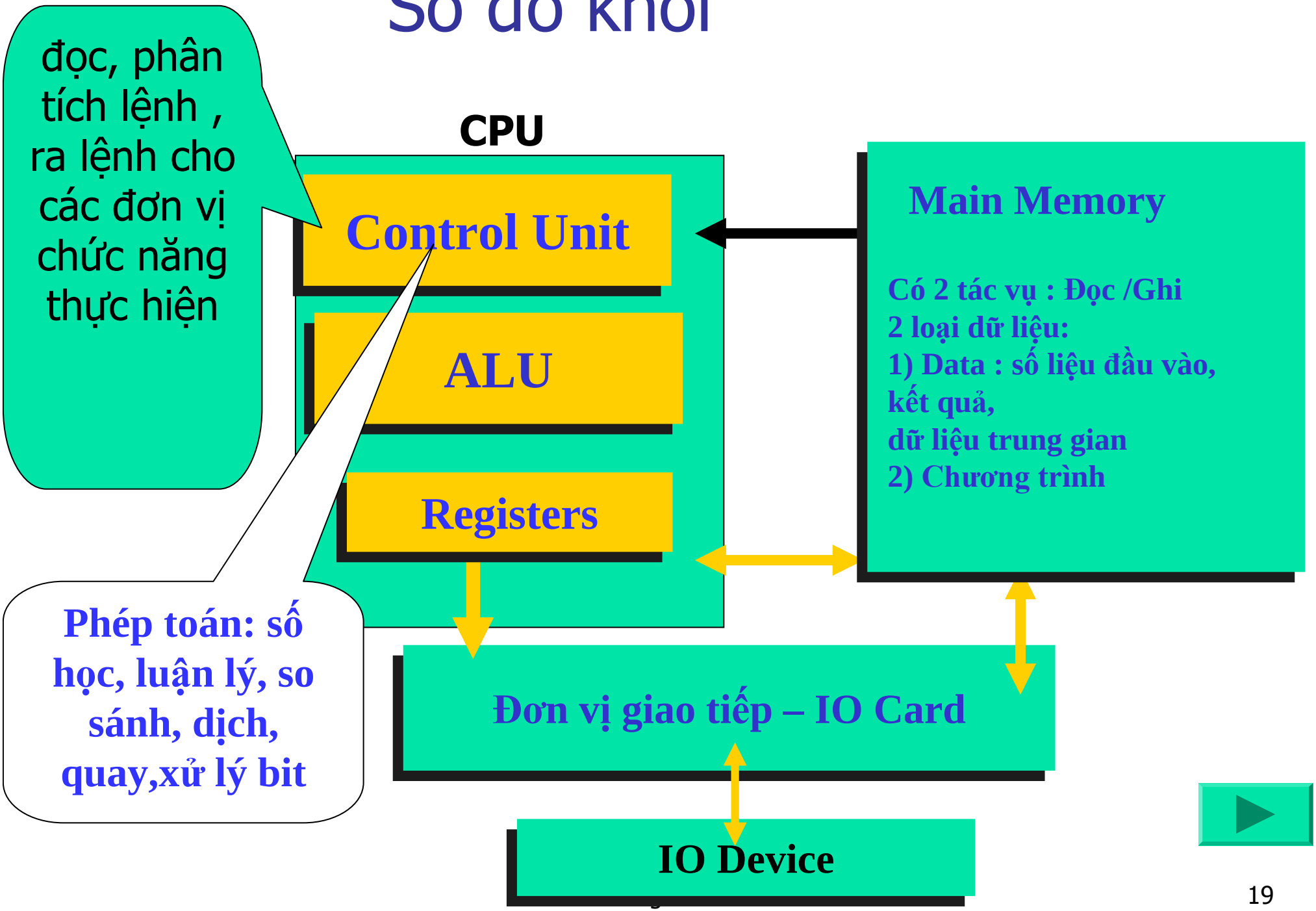
CPU gồm 1 số bộ phận tách biệt :

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

CPU (cont)

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

Sơ đồ khối


đọc, phân tích lệnh , ra lệnh cho các đơn vị chức năng thực hiện
CPU
Control Unit
ALU
Registers
Main Memory
Có 2 tác vụ : Đọc /Ghi
2 loại dữ liệu:
1) Data : số liệu đầu vào, kết quả, dữ liệu trung gian
2) Chương trình
Phép toán: số học, luận lý, so sánh, dịch, quay,xử lý bit
Đơn vị giao tiếp – IO Card
IO Device

Chu kỳ lệnh

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

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

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

Chu kỳ máy

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

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

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

Thực hiện lệnh

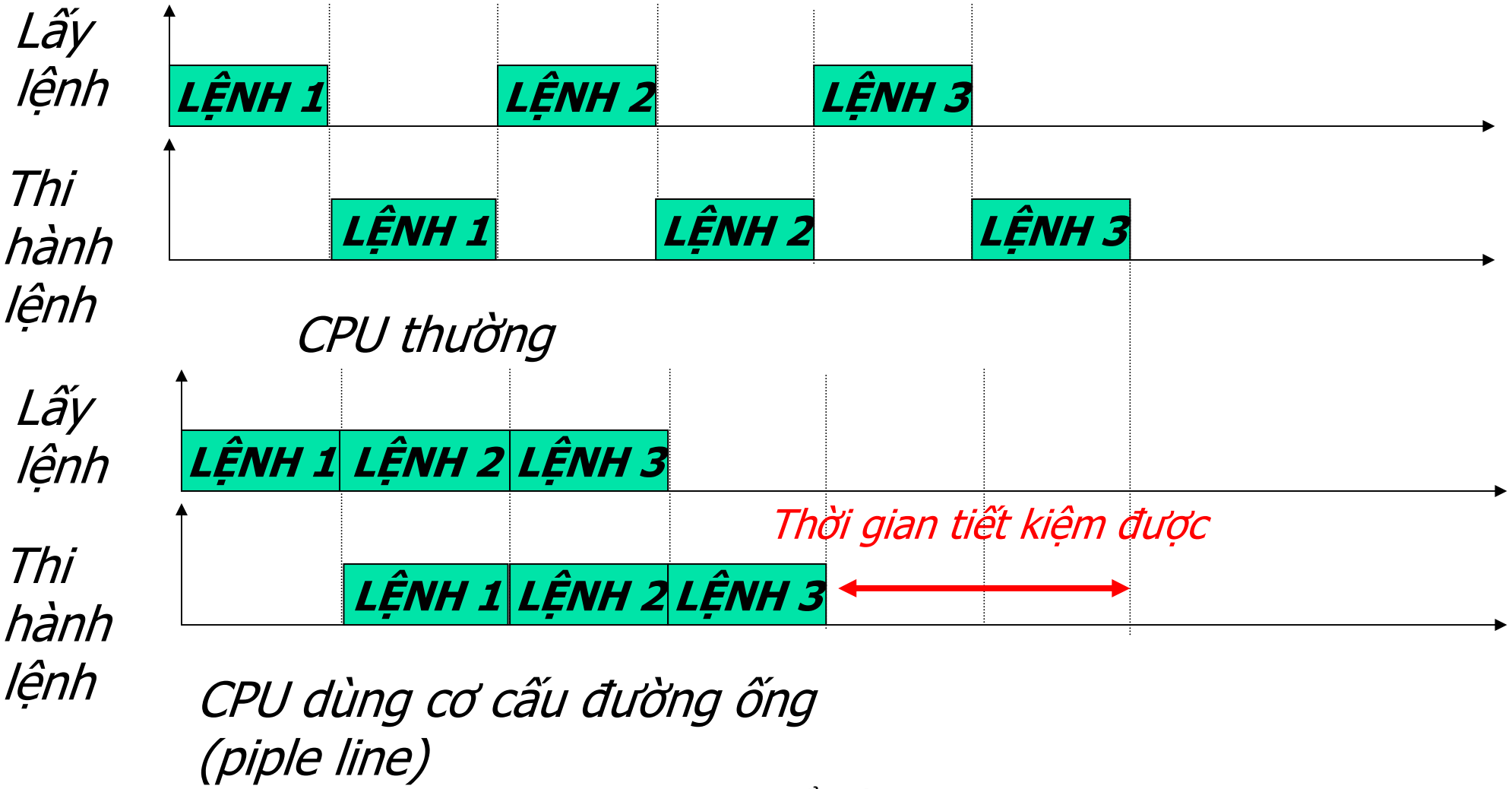
CPU thực hiện lệnh tuần tự theo chuỗi các bước :

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

Thực hiện lệnh (cont)

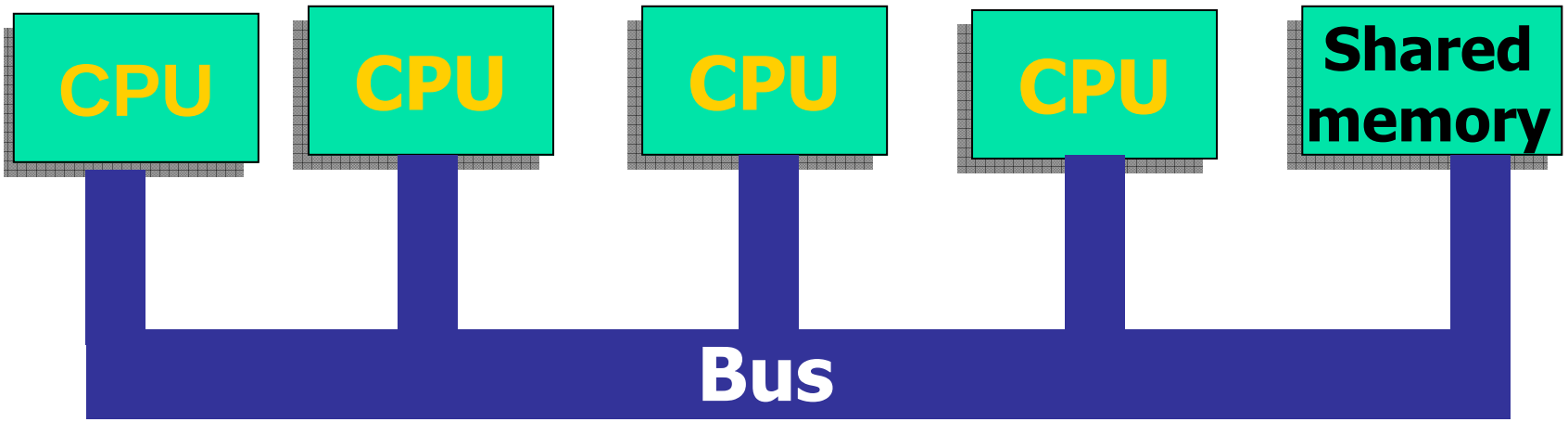
- Thực hiện lệnh..
- Lưu kết quả ở nơi thích hợp. .
- Trở về bước 1 để thực hiện lệnh kế.

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



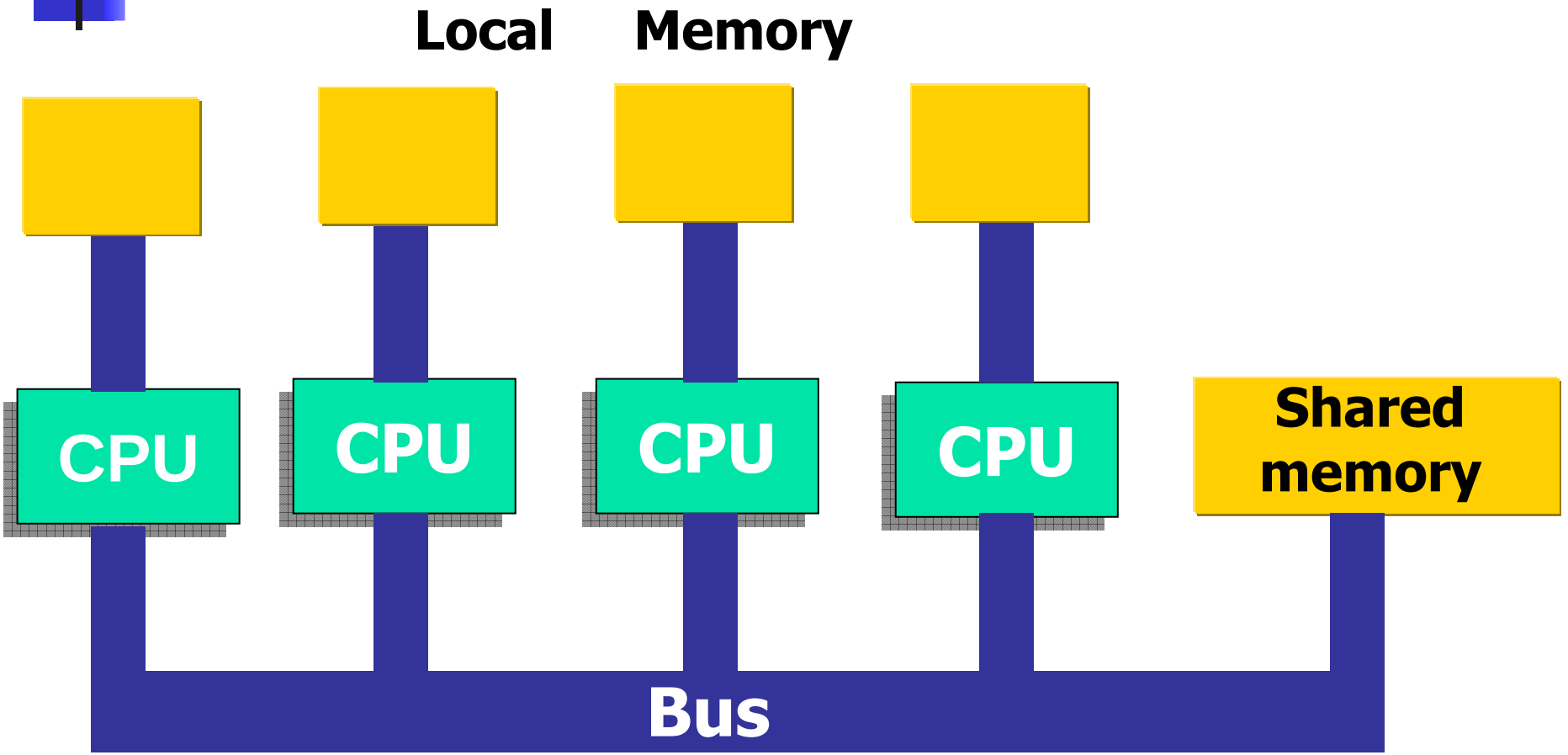
CPU dùng cơ cấu đường ống (piple line)

Hệ đa bộ xử lý (MultiProccessor)

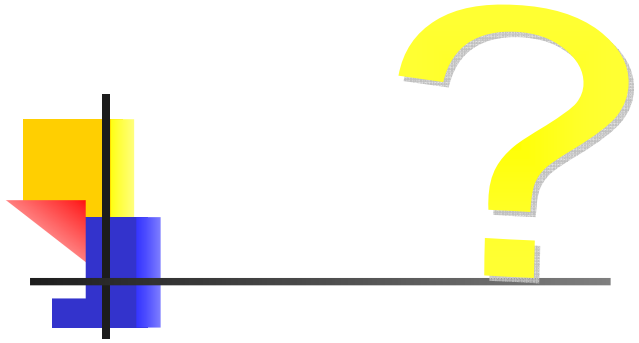


Hệ MultiProccessor sử dụng 1 đường Bus

Hệ đa bộ xử lý (MultiProccessor)

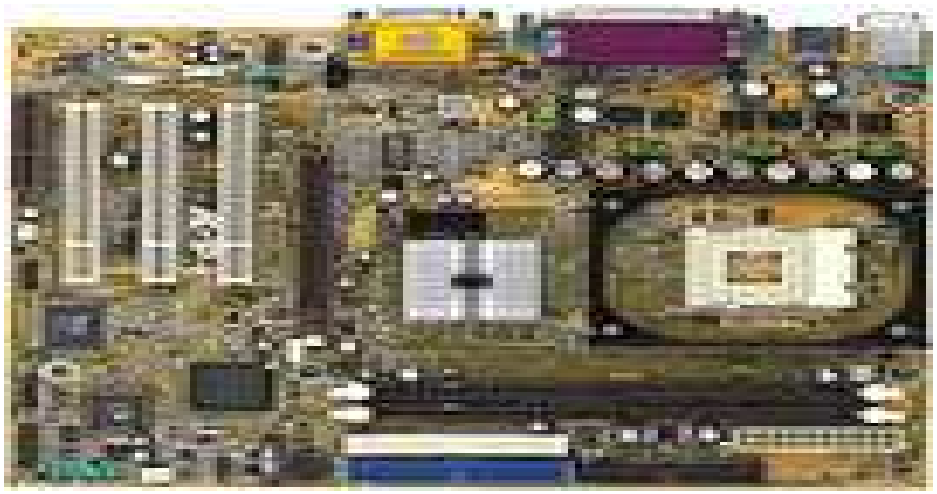


Hệ MultiProccessor sử dụng nhiều bộ nhớ cục bộ



Bus

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



2.3 Hệ thống Bus

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

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

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

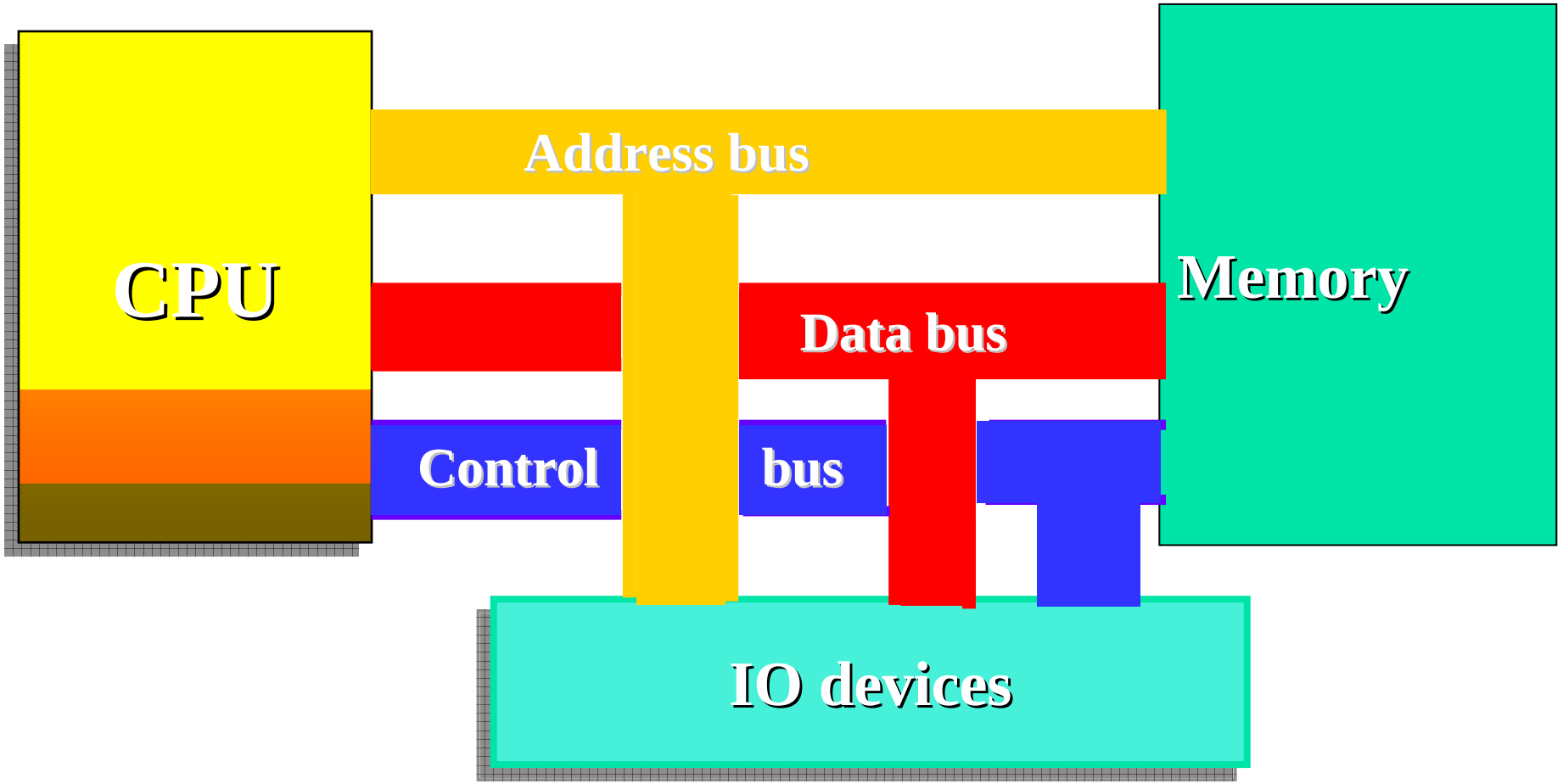
Các loại Bus

- **Address Bus : nhóm đường truyền nhận diện vị trí truy xuất trong thiết bị đích : thông tin được đọc từ đâu hoặc ghi vào đâu.**

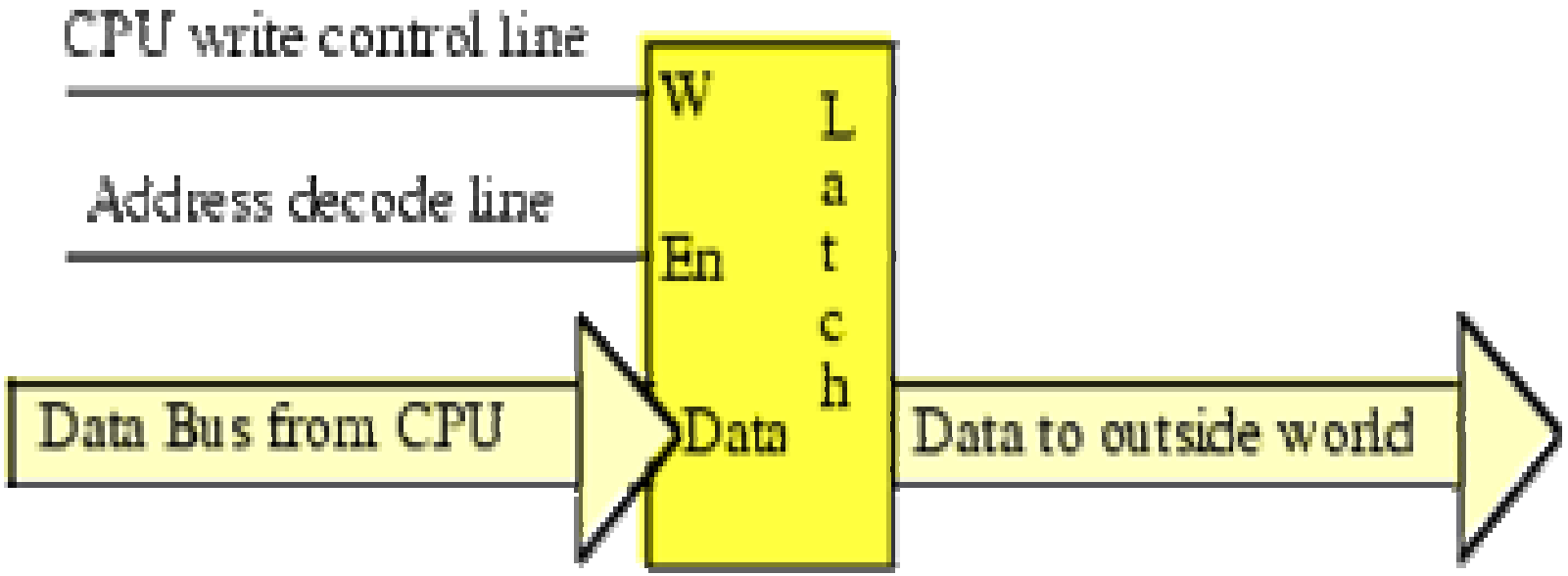
- **Data Bus : nhóm đường truyền để tải data thực sự giữa các thiết bị hệ thống do địa chỉ trên address bus đã xác định. Độ rộng của data bus (số đường dây dẫn) xác định data trong mỗi lần truyền là bao nhiêu.**

- **Control Bus : nhóm đường truyền cho các tín hiệu điều khiển như : tác vụ là đọc hay ghi, tác vụ thực thi trên bộ nhớ hay trên thiết bị ngoại vi, nhận dạng chu kỳ bus và khi nào thì hoàn tất tác vụ…**

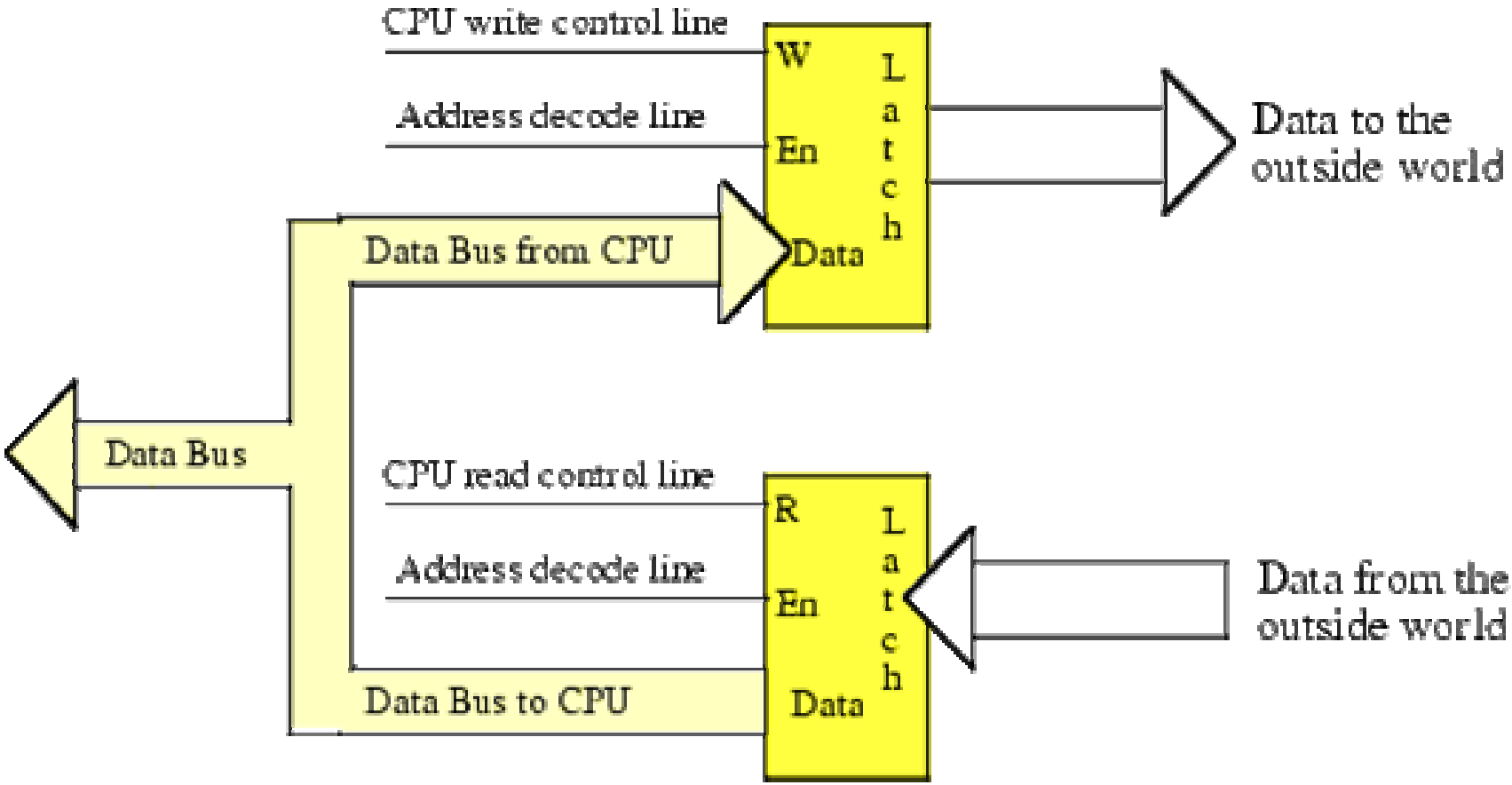
Minh họa hệ thống Bus



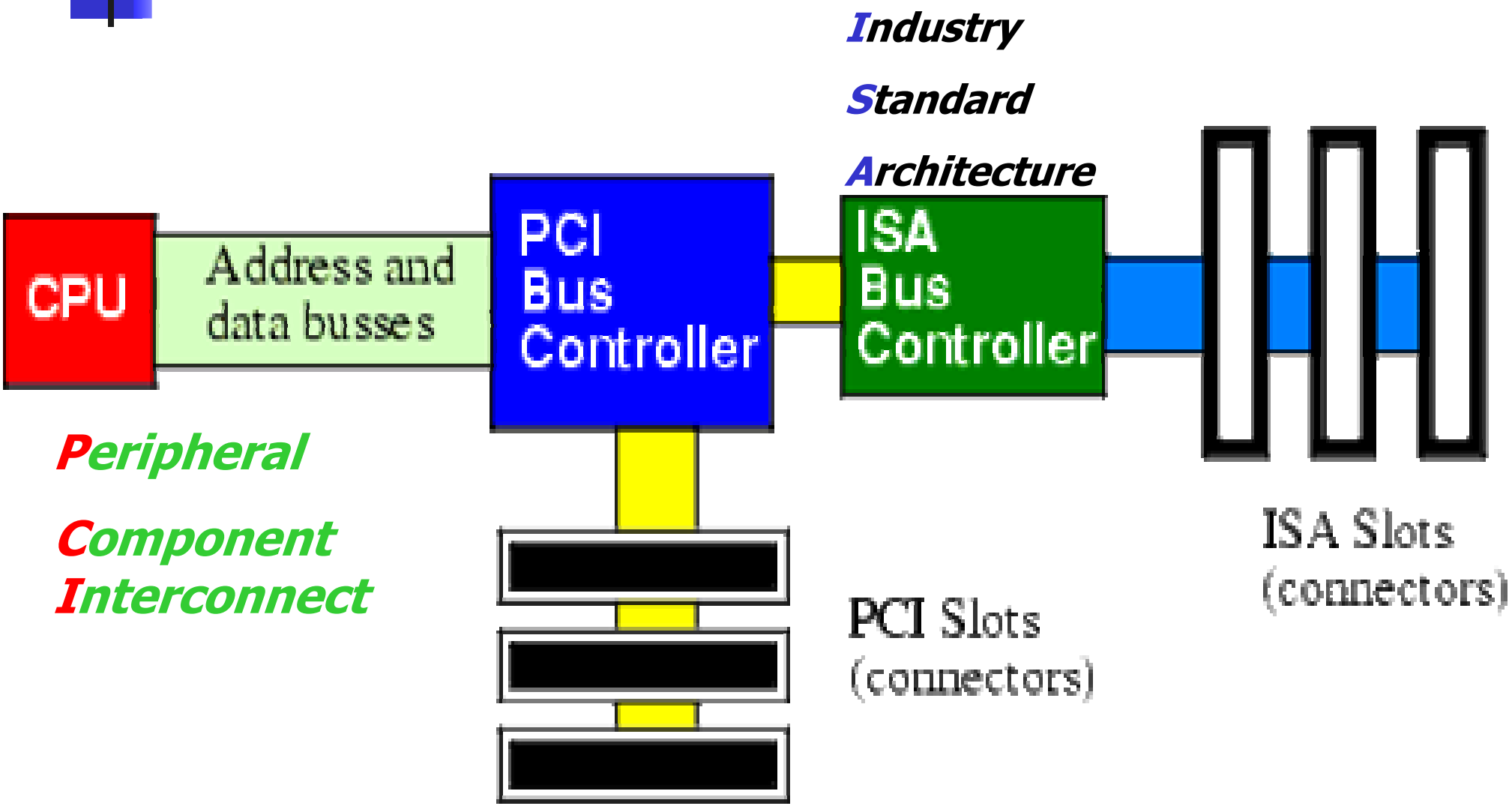
A Typical Output Port



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



Connection of the PCI and ISA Busses in a Typical PC



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

Bus PCI

PCI chuẩn nốI ghép các thiết bị ngọai 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*

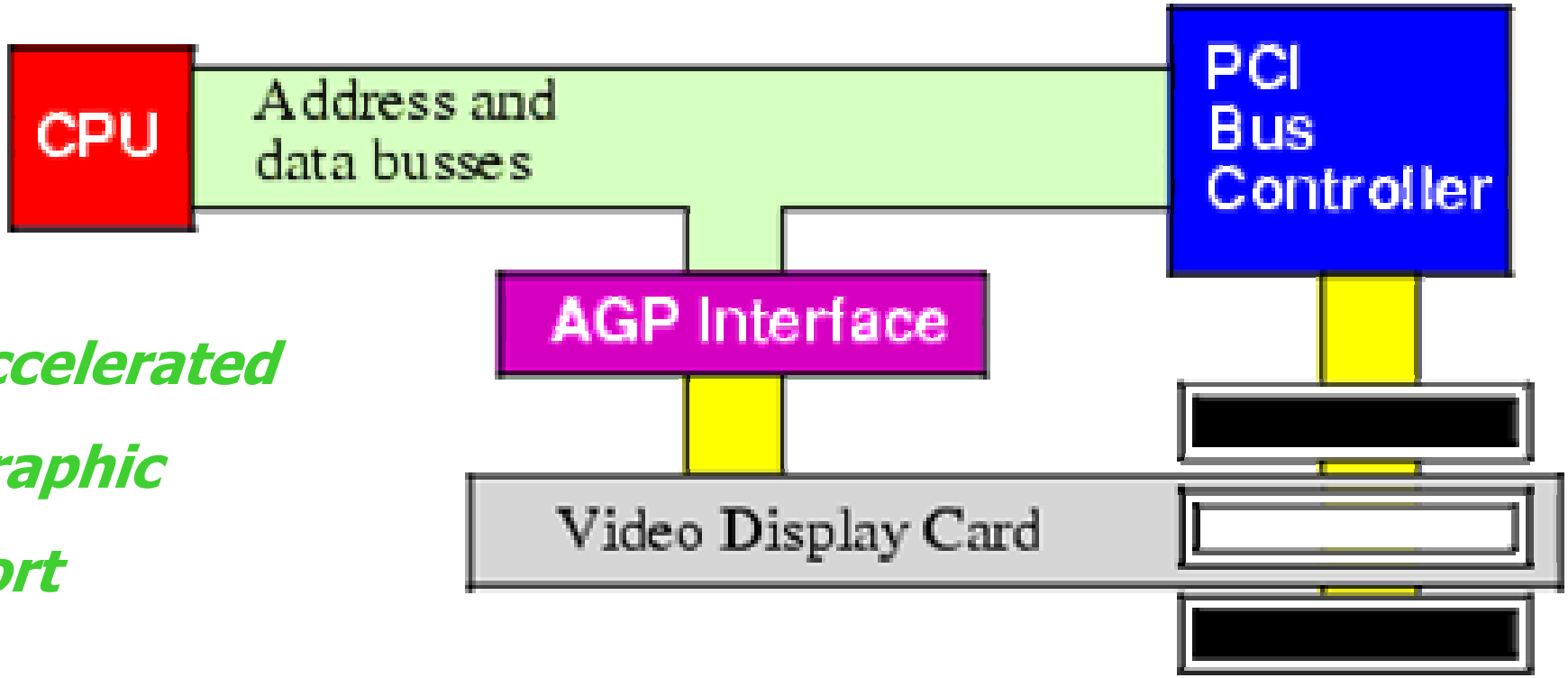
Plug and Play

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 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.

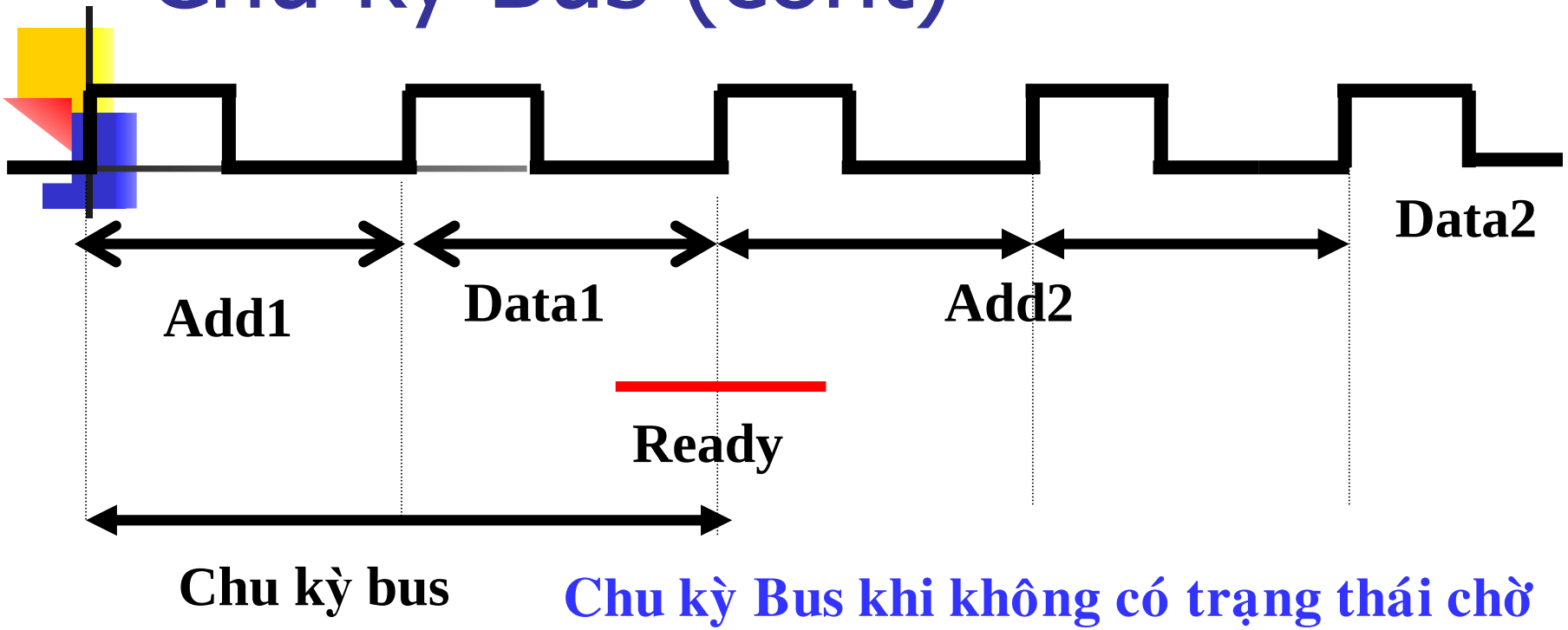
Đây là mốc tham chiếu theo thời gian để đồng bộ hoá tất cả các tác vụ bên trong máy tính. Xung đầu tiên gọi là Address time , địa chỉ truy xuất sẽ được gửi đi cùng với tín hiệu xác định loại tác vụ sẽ được thực thi (đọc/ghi/đến mem/đến I/O).

Chu kỳ Bus

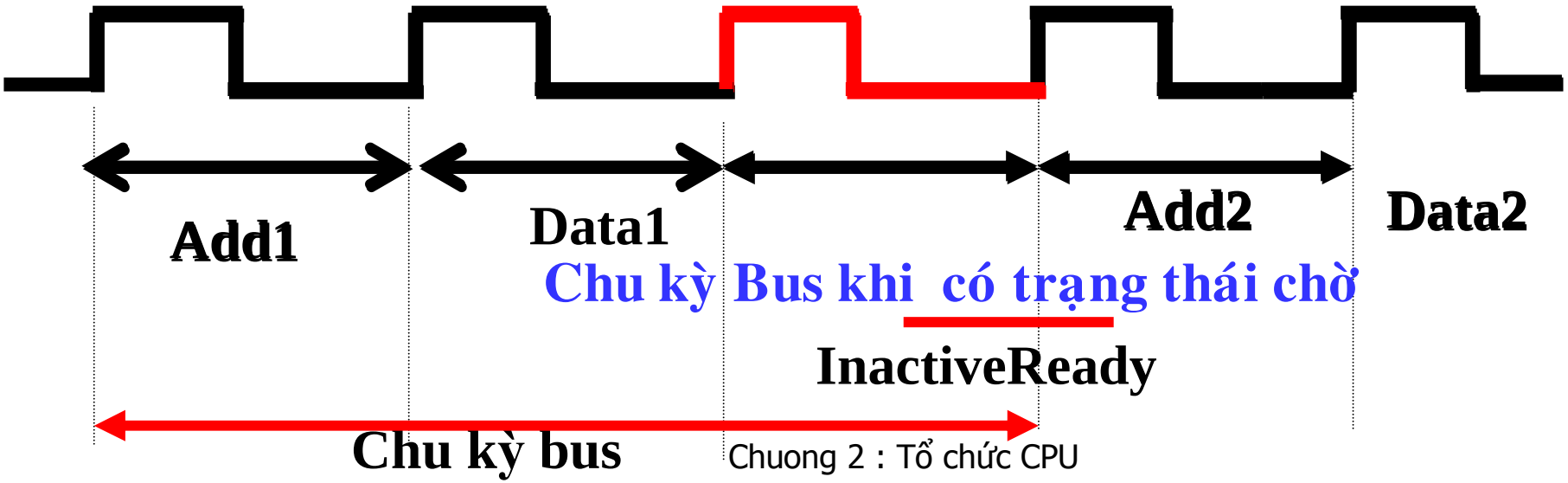
Cuối xung thứ 2, CPU sẽ kiểm tra đường tín hiệu Ready. Nếu thiết bị cần truy xuất sẵn sàng đáp ứng tác vụ, thiết bị này sẽ kích 1 tín hiệu lên đường Ready để báo cho CPU biết và chu kỳ bus hoàn tất.

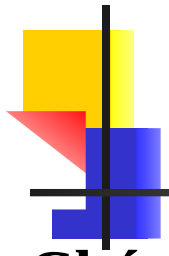
Khi 1 thiết bị không sẵn sàng, không có tín hiệu trên đường Ready, CPU phải chờ, có thể phải tiêu tốn thêm 1 hay nhiều xung clock.

Chu kỳ Bus (cont)



Chu kỳ Bus khi không có trạng thái chờ





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

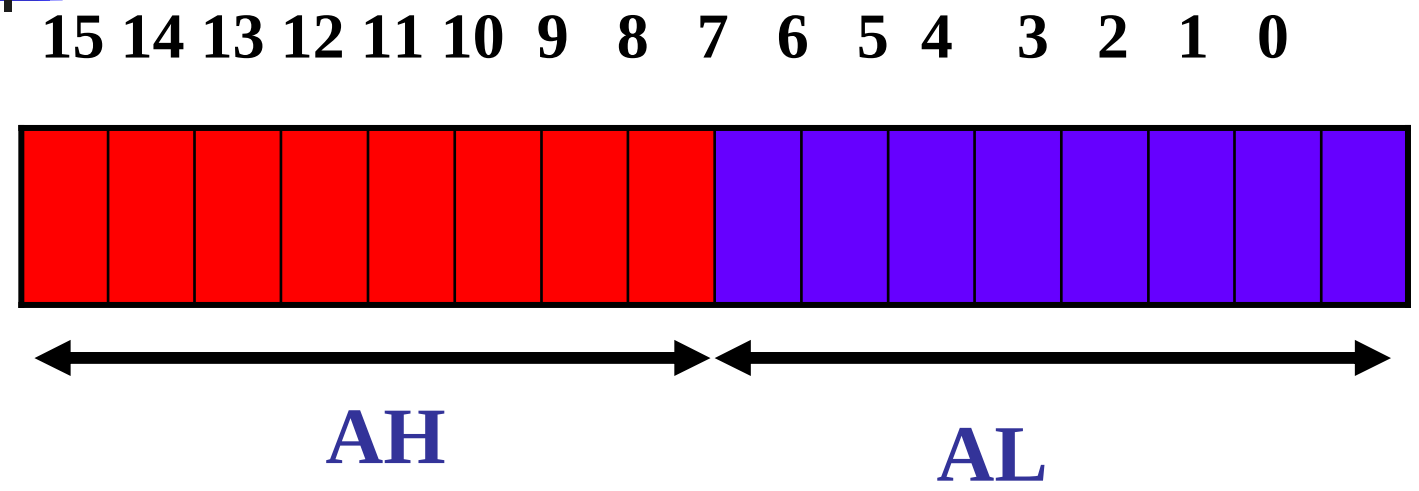
2.4 Hệ thống thanh ghi

- **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.**

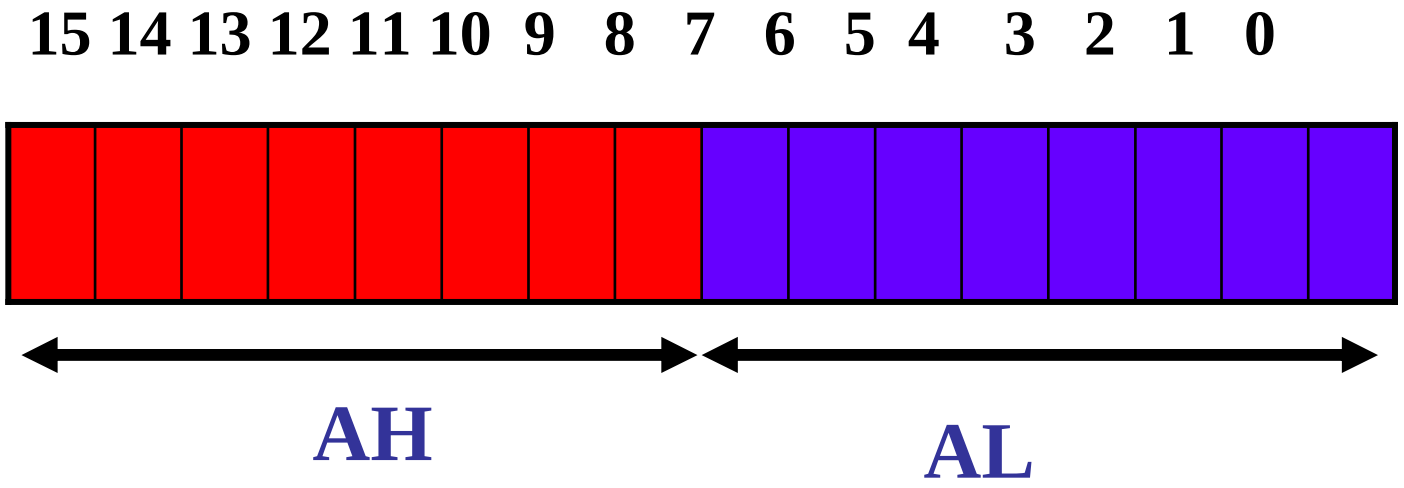
AX Register



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

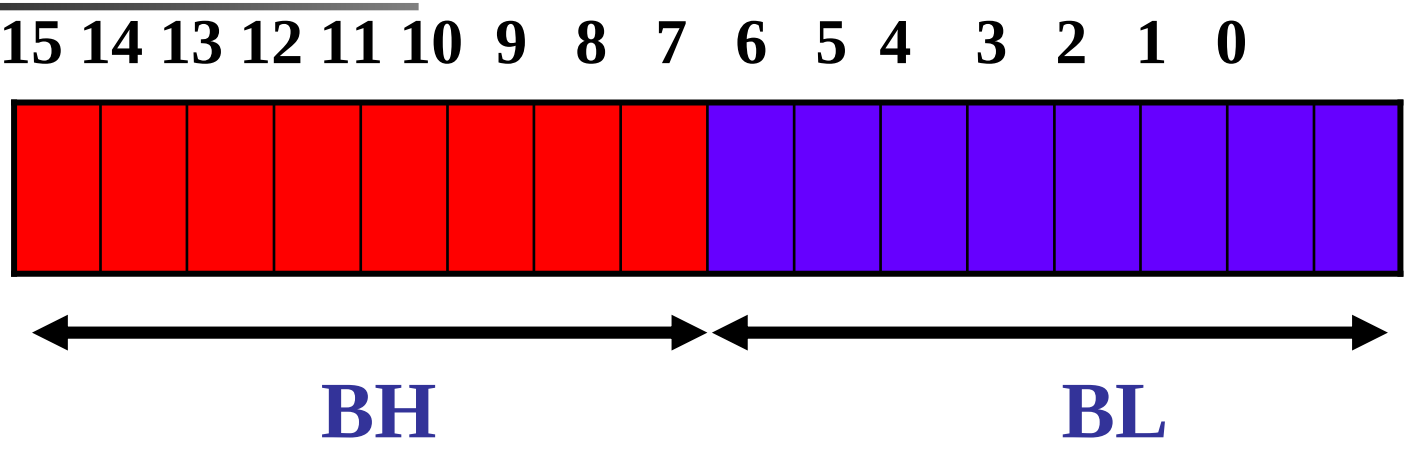


Thanh ghi AH là nửa cao của thanh ghi AX

Thanh ghi AL là nửa thấp của thanh ghi AX

Thí dụ nếu AX=1234h thì AH=12H AL=34h

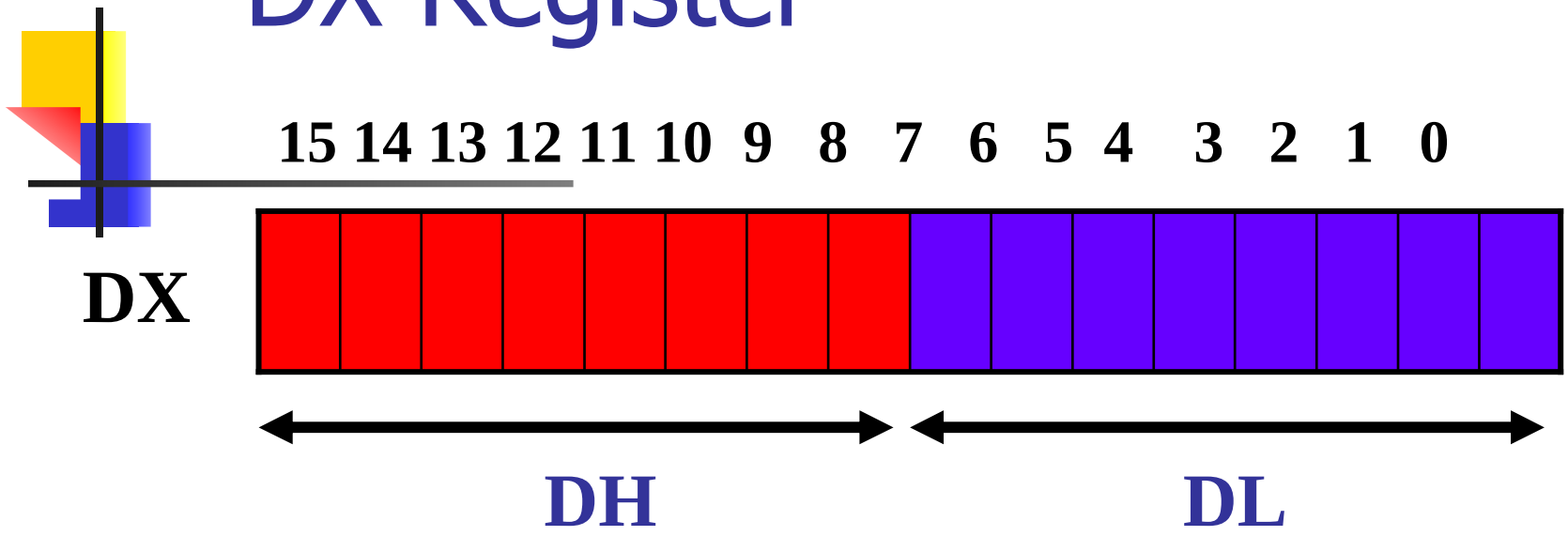
BX Register



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

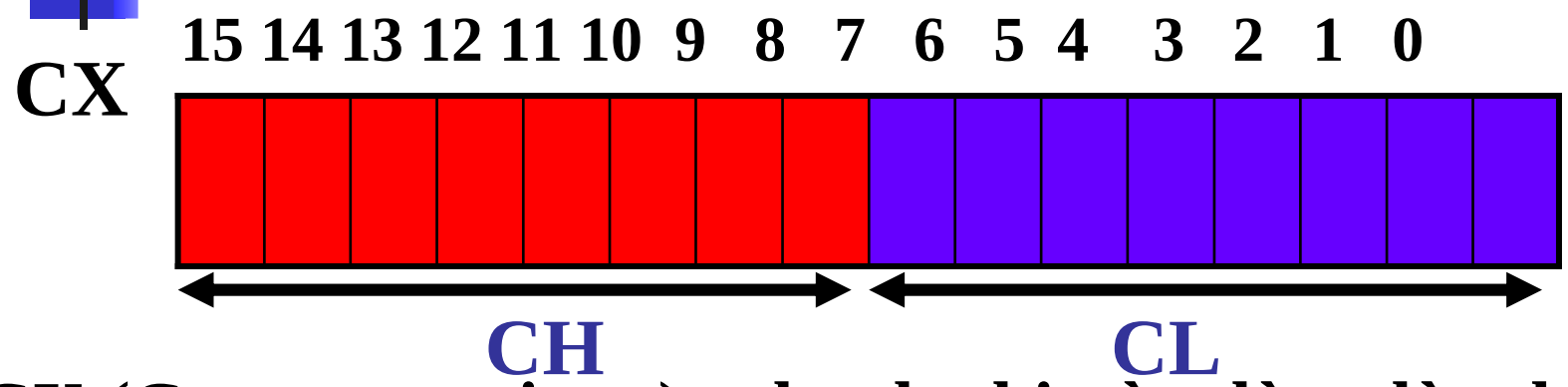


Thanh ghi DX (Data register) : dài 16 bit nhưng nó cũng có thể chia làm 2 thanh ghi 8 bit DH và DL

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.

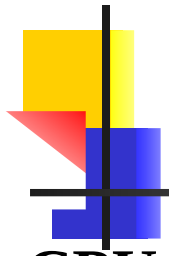
CX Register



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

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

  EAX, EBX, ECX, EDX ....

2.5 Thanh ghi đoạn và sự hình thành địa chỉ

- **8088 sử dụng 20 bit để đánh địa chỉ bộ nhớ → quản lý trên 1Mb bộ nhớ. Nhưng 8088 lại không có thanh ghi nào 20 bit, tất cả là 16 bit do đó 1 thanh ghi chỉ có thể đánh địa chỉ tối đa là 64 kB bộ nhớ.**

- **Như vậy phải kết hợp 2 thanh ghi mới địa chỉ hoá toàn bộ bộ nhớ. 8088 sử 1 trong các thanh ghi dùng chung và 1 trong các thanh ghi đoạn (CS,DS,SS,ES) để tạo thành 1 địa chỉ 20 bit.**

SỰ PHÂN ĐOẠN BỘ NHỚ

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

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

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

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

2.5 Địa chỉ vật lý & địa chỉ luận lý

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

Địa chỉ vật lý dùng như thế nào ?

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

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

Địa chỉ luận lý

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

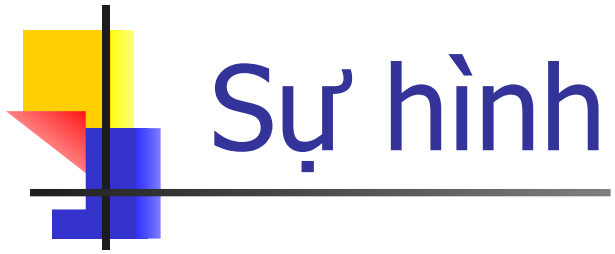
Segment : offset → *Địa chỉ trong đoạn (độ dời)*

Địa chỉ đoạn

Ex : B001:1234

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

Segment : offset



Sự hình thành địa chỉ

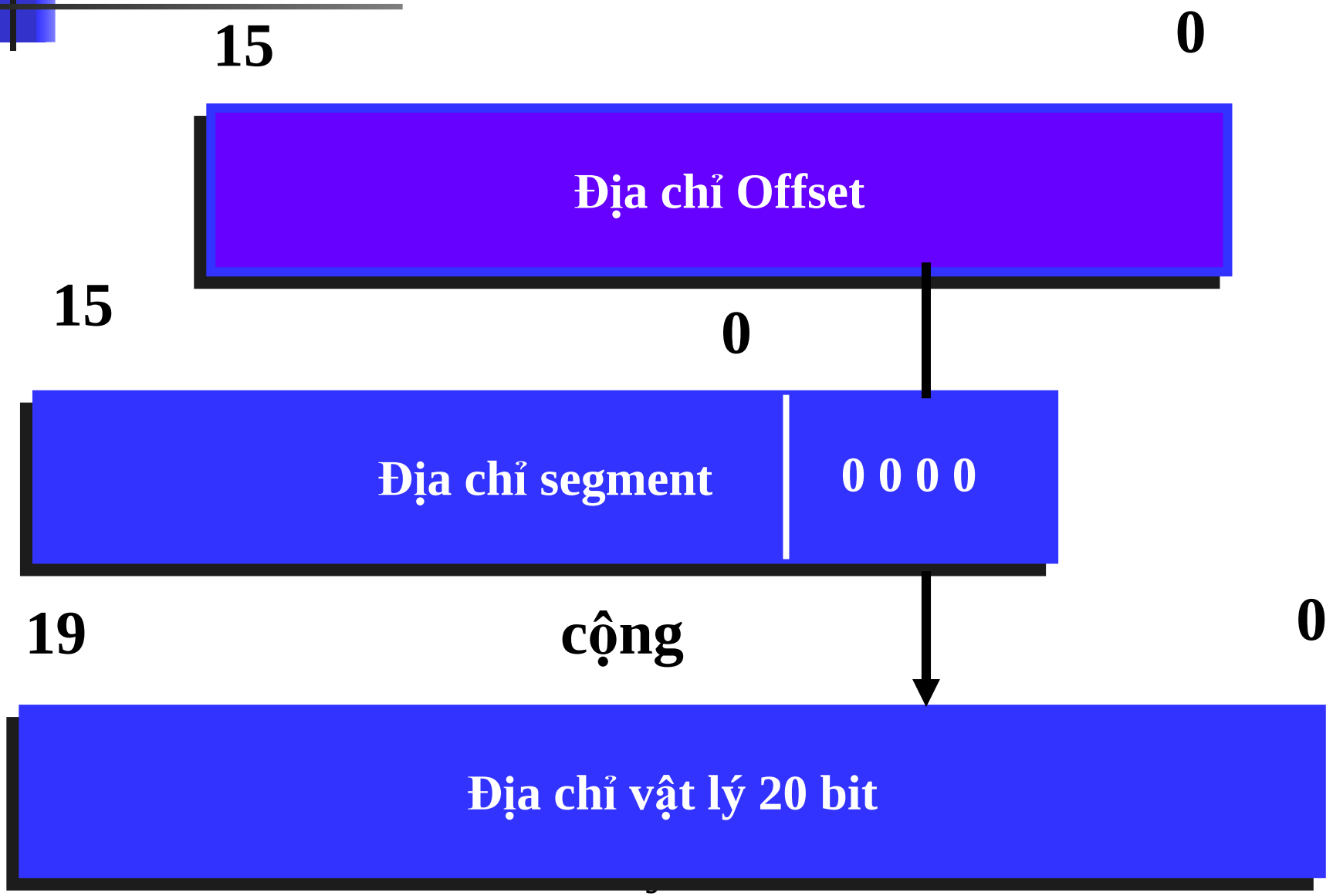
Hãng Intel đề xuất 1 phương pháp để hình thành địa chỉ.

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

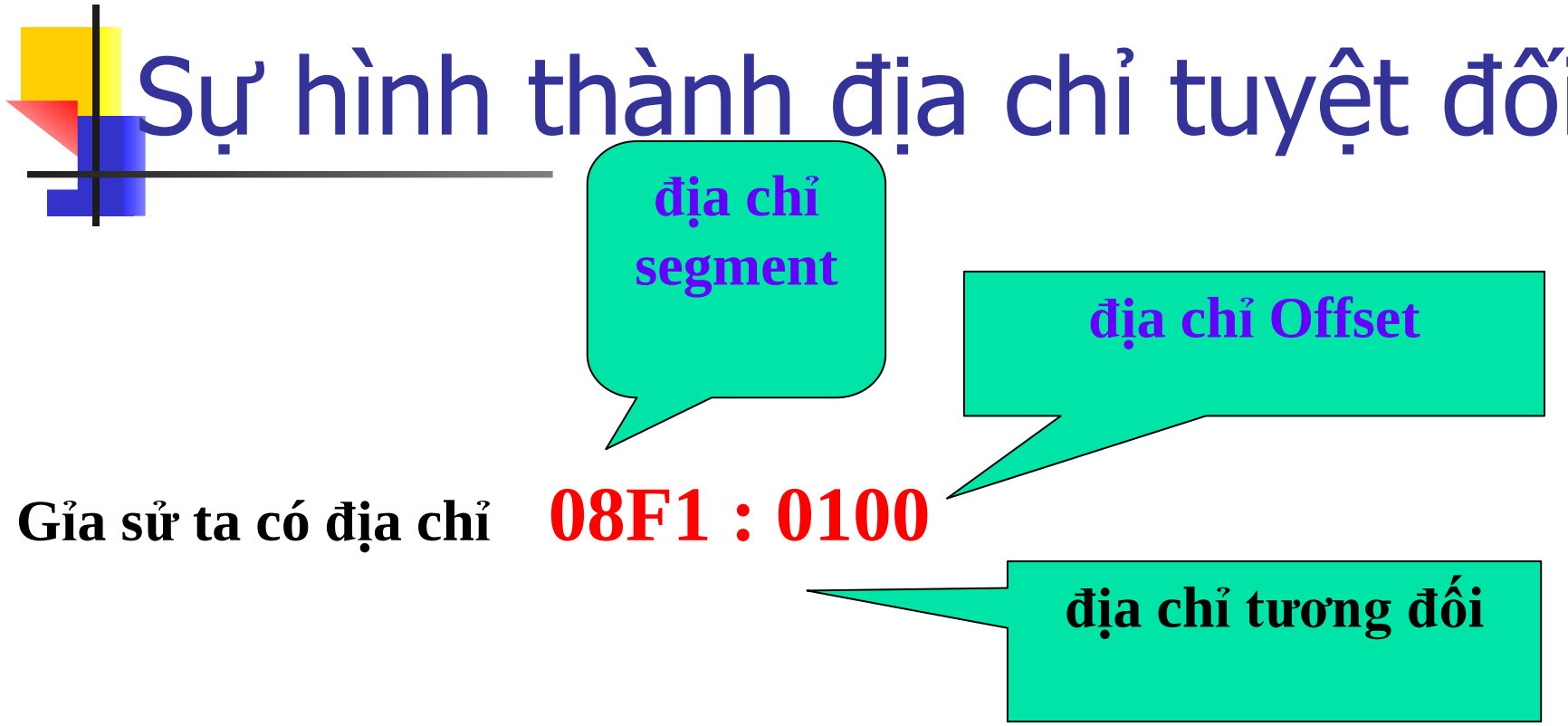
Địa chỉ cơ sở lưu trong 1 thanh ghi segemnt, còn địa chỉ offset nằm trong 1 thanh ghi chỉ số hay thanh ghi con trỏ.

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



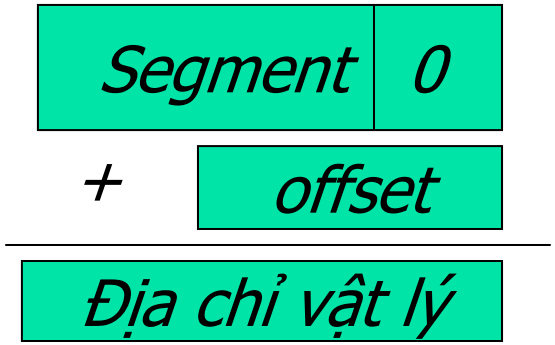
CPU tự động lấy địa chỉ segment x 10 (hệ 16) thành 08F10

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

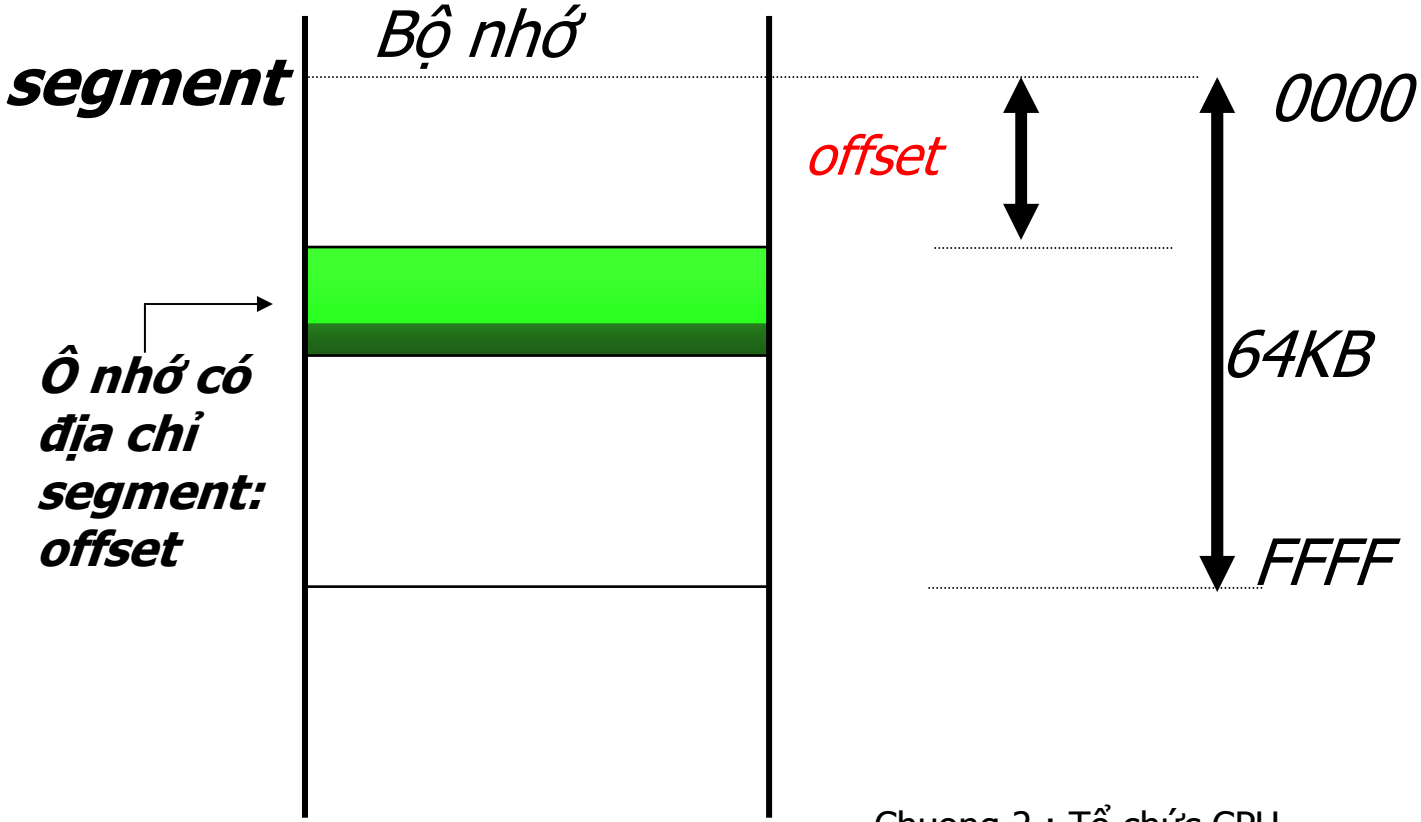
Địa chỉ segment hay còn gọi là địa chỉ nền của đoạn. Nó cho biết điểm bắt đầu của đoạn trong bộ nhớ.

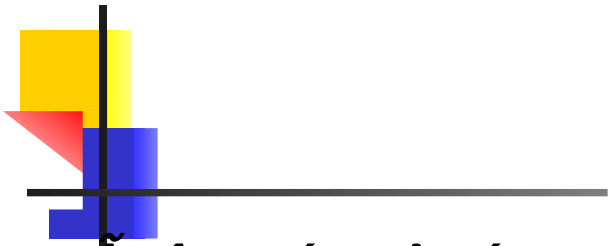
Địa chỉ offset thể hiện khoảng cách kể từ đầu đoạn của ô nhớ cần tham khảo.

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

Sự chồng chất các đoạn

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





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

Ex : 1234:1234

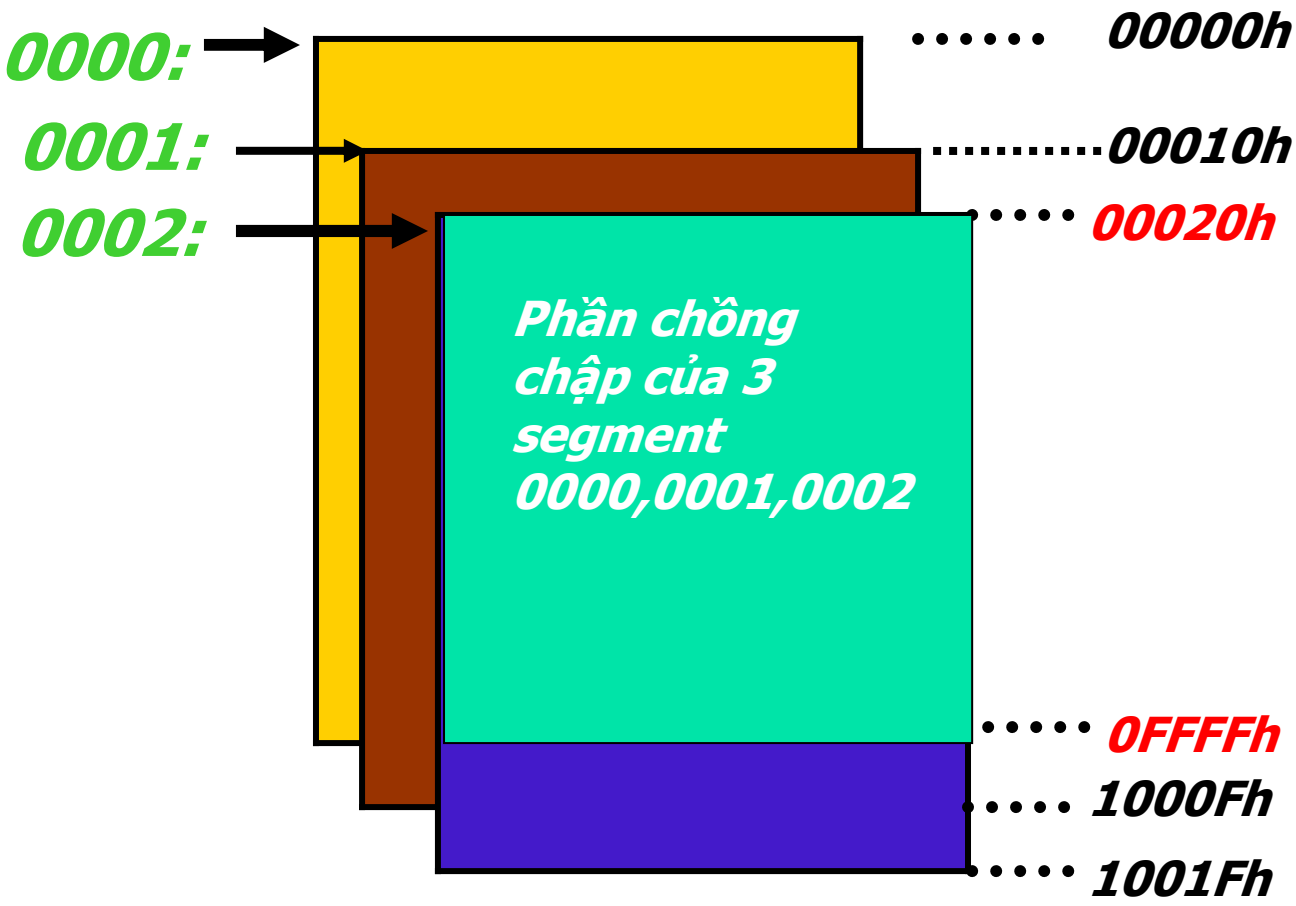
1334:0234

1304:0534

Đều có chung địa chỉ vật lý 13574h

Tại sao ?

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



Giải thích

0000:0000 ➔ 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.....

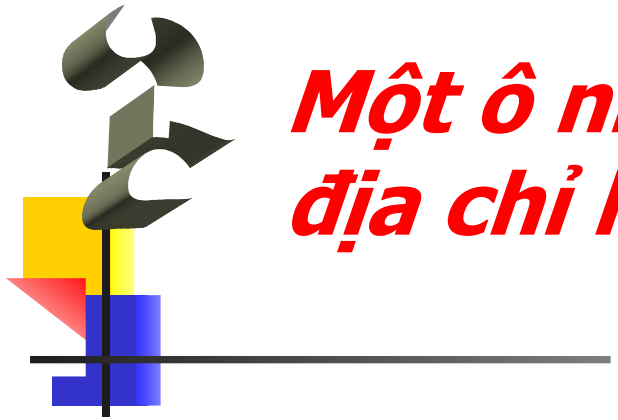
Phần chồng chập 3 segment 0000,0001,0002 trên hình vẽ là vùng bộ nhớ mà bất kỳ ô nhớ nào nằm trong đó (địa chỉ vật lý từ 00020h đến 0FFFFh) đều có thể có địa chỉ luận lý tương ứng trong cả 3 segment.

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 đoạn CS, DS, SS, ES

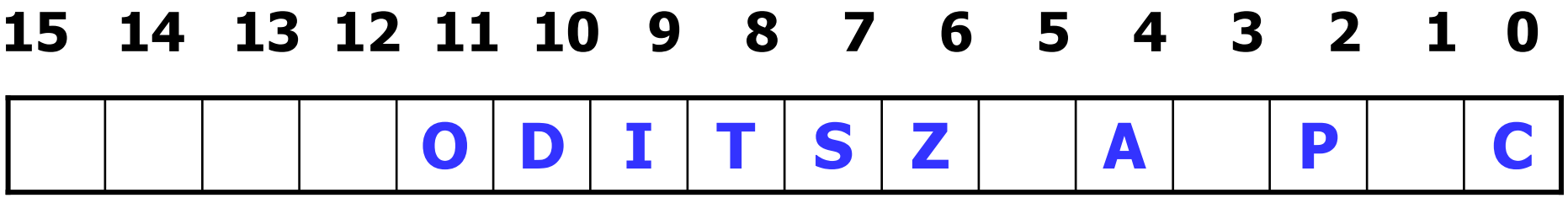
- 3 trong 4 thanh ghi đoạn được dùng trong các mục đích đặc biệt sau
- **CS** : xác định đoạn lệnh – nơi chứa chương trình được thi hành.
- **DS** : xác định đoạn dữ liệu – nơi chứa chương trình được thi hành.
- **SS** : xác định đoạn stack – vùng làm việc tạm thời dùng để theo dõi các tham số và các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

Các thanh ghi đoạn CS, DS, SS, ES

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

Thanh ghi trạng thái (thanh ghi cờ)

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



O OverFlow flag D : Direction flag I : Interrupt flag

T : Trap flag S : Sign flag Z : Zero flag

A : Auxiliary flag P : Parity flag C : Carry flag

Thanh ghi trạng thái (thanh ghi cờ)

Giải thích :

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

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

Nếu số bit 1 là số lẻ thì PF là Off.

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

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

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

Thanh ghi trạng thái (thanh ghi cờ)

Giải thích :

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

3 bit còn lại là 3 bit điều khiển :

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

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

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

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

Thanh ghi chỉ số (Index)

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

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

Thanh ghi chỉ số (Index)

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

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.

Các đặc tính của CPU Intel

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

Các đặc tính của CPU Intel

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

Các đặc tính của CPU Intel

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

Tóm tắt CPU họ Intel

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

- **Nếu muốn có khả năng thực hiện các phép toán dấu chấm động phải gắn CoProccessor 8087.**

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

Tóm tắt CPU họ Intel

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

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

Tóm tắt CPU họ Intel

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

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

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

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

RISC & CISC

- **Nguyên lý CISC** :

Complex Intruction Set Computer

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

quá nhiều lệnh ➔ nạp lâu ➔ làm chậm hệ thống

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

Chỉ có hơn 20% lệnh thường dùng tới

RISC & CISC

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

Reduce Intruction Set Computer

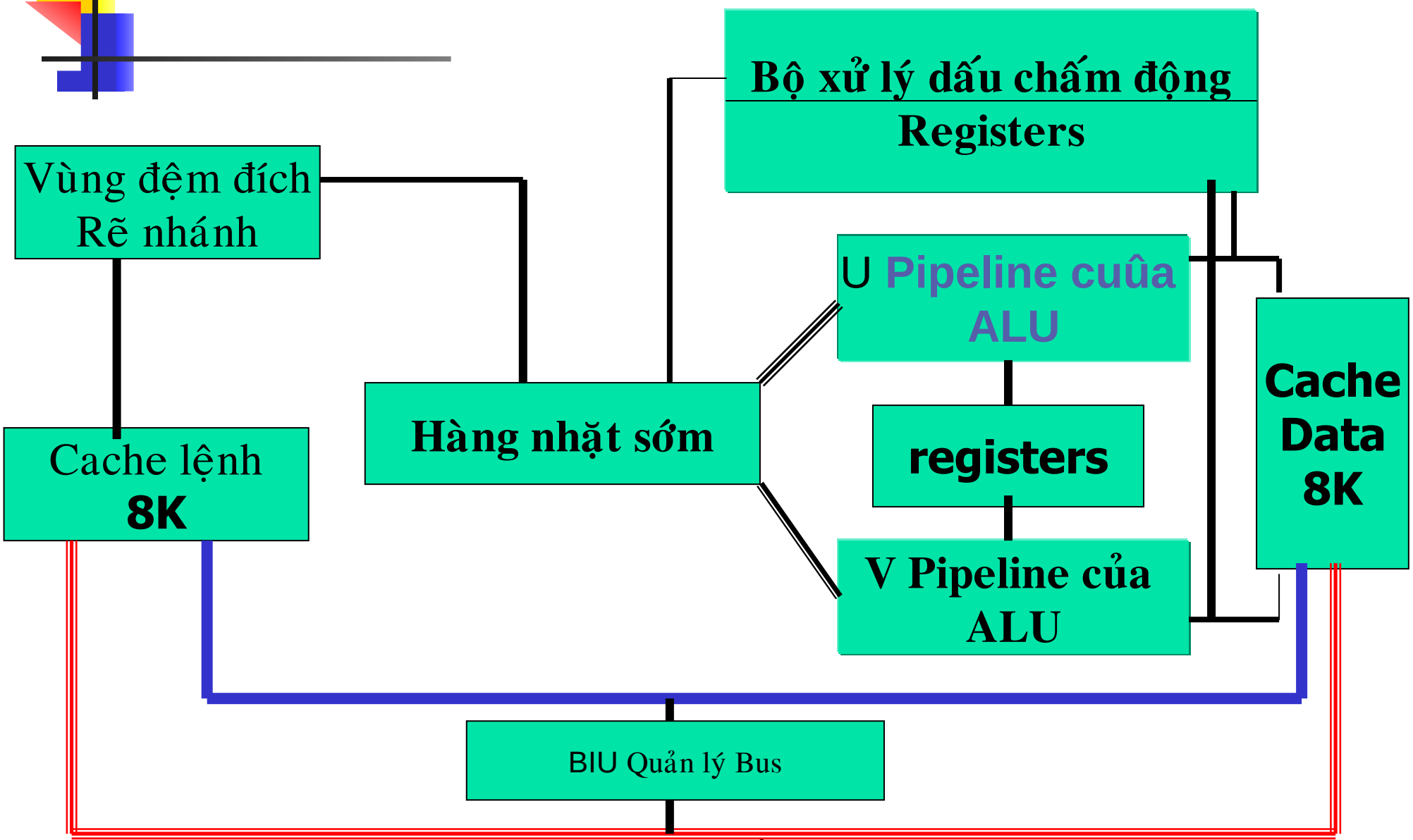
tập lệnh nhỏ ➔ thi hành ngay không cần giải mã.

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

CPU Pentium

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

SƠ ĐỒ KHỐI PENTIUM



Câu hỏi ôn tập

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

Câu hỏi ôn tập

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

Câu hỏi ôn tập

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

Bus ISA-8 bits :

a. *chạy ở tốc độ đồng hồ là 8 MHz truyền tải dữ liệu tối đa 8 MB/s.*

b. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 6MB/s.*

c. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 1MB/s.*

d. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 12MB/s.*

Bus ISA-16 bits :

a. *chạy ở tốc độ đồng hồ là 8➔12 MHz truyền tải dữ liệu tối đa 8 MB/s.*

b. *chạy ở tốc độ đồng hồ là 32 MHz truyền tải dữ liệu tối đa 12MB/s.*

c. *chạy ở tốc độ đồng hồ là 4.77 MHz truyền tải dữ liệu tối đa 12MB/s.*

d. *chạy ở tốc độ đồng hồ là 16MHz truyền tải dữ liệu tối đa 12MB/s.*

Bus PCI :

a. *truyền tải dữ liệu tối đa 528 MB/s.*

b. *truyền tải dữ liệu tối đa 128MB/s.*

c. *truyền tải dữ liệu tối đa 512MB/s.*

d. *truyền tải dữ liệu tối đa 64MB/s.*

Dẫn đầu về Chipset hiện có trên thị trường là :

a.AMD

b.ALI

c.Intel

d.Mac

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

a.thông tin giữa các máy tính

b.dữ liệu giữa các thiết bị 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

MOV AX,1234

MOV BH,AL

MOV BL,AH

INT 20H

Hãy cho biết giá trị của các thanh ghi AX,BX ?

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.

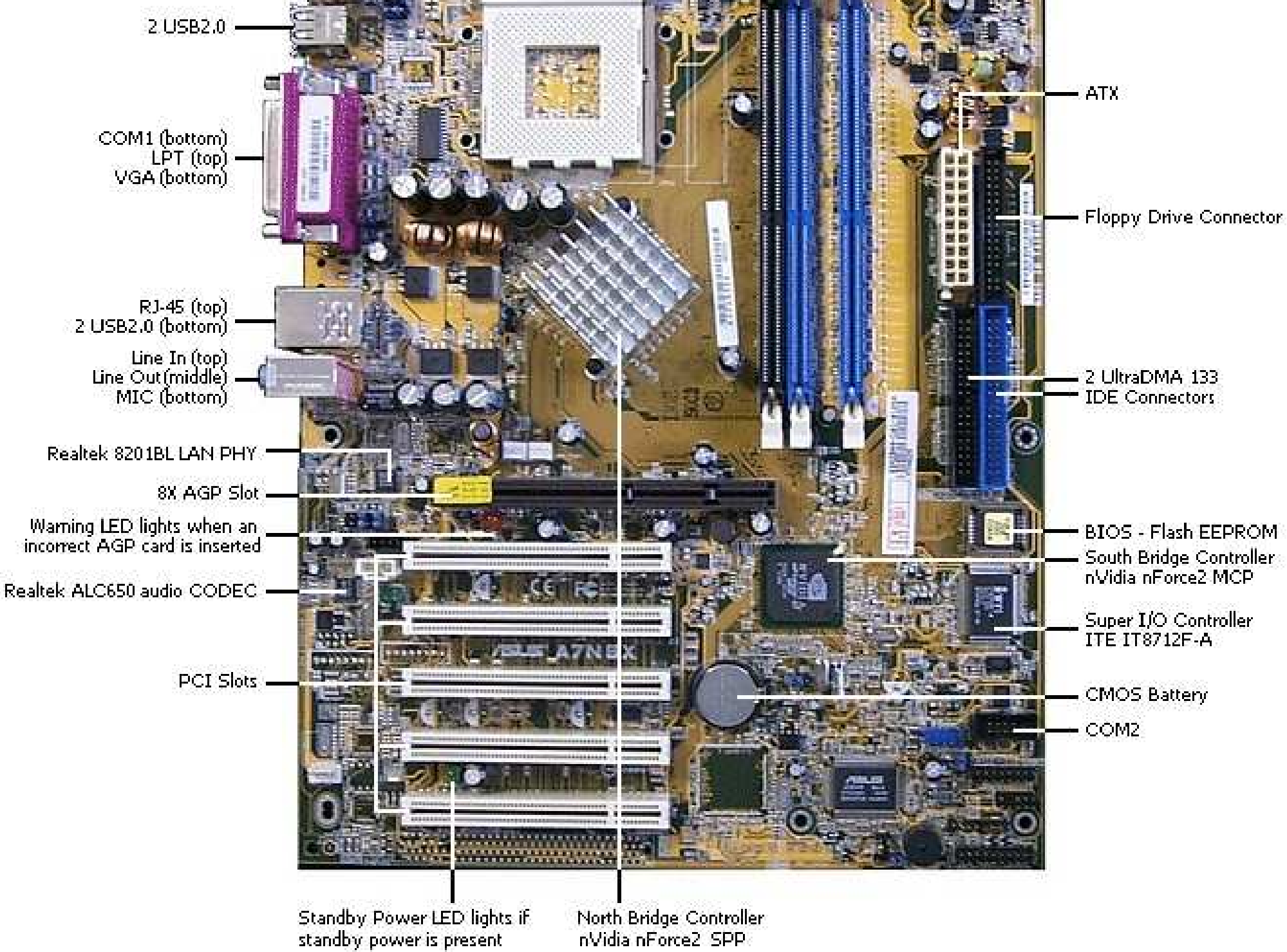
BÀI TẬP

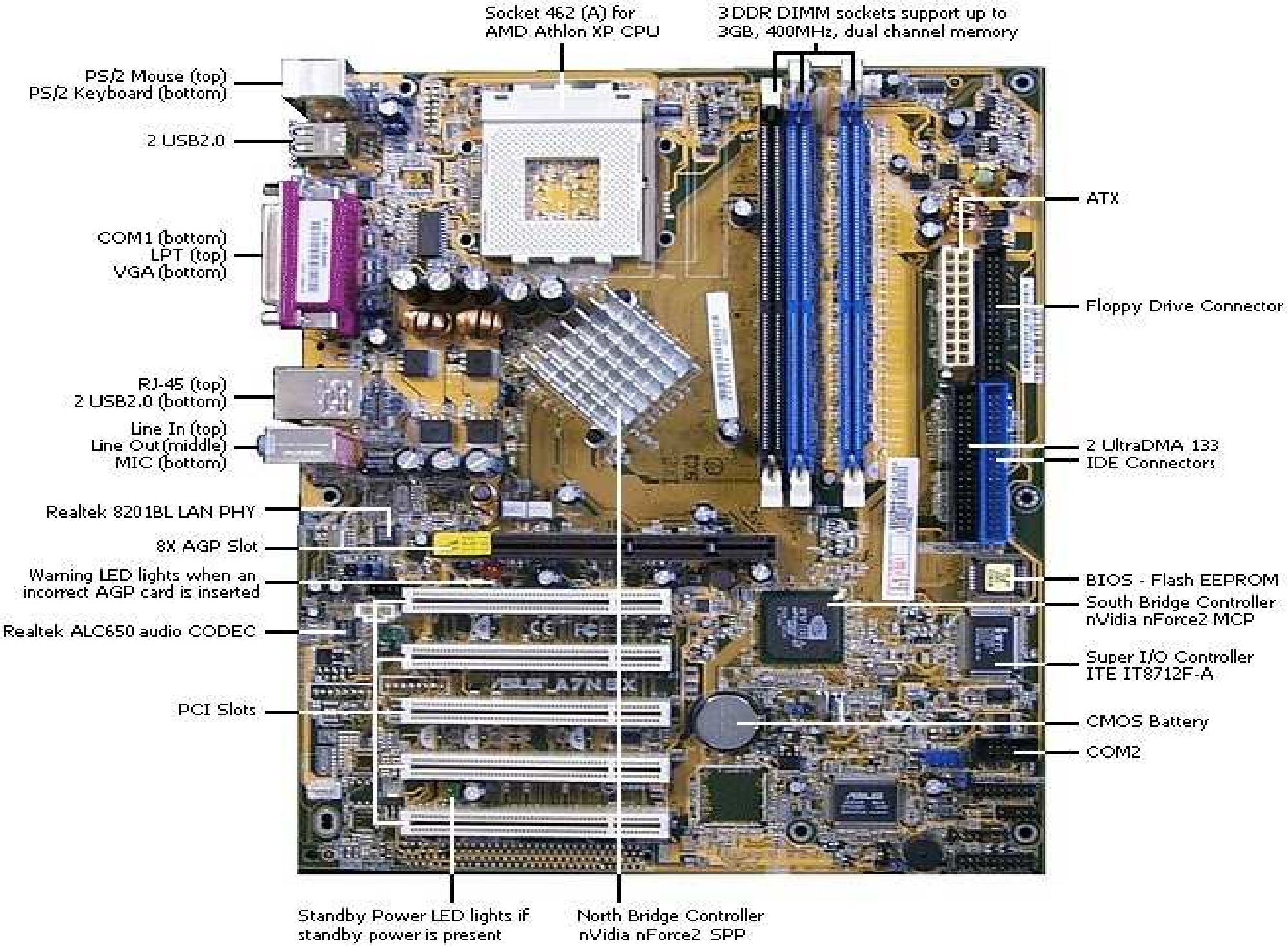
Bài 5: Giả sử thanh ghi trong MT của bạn dài 24 bits, cho biết giá trị của số dương lớn nhất mà thanh ghi này có thể chứa ở 2 hệ 2 và hệ 16?

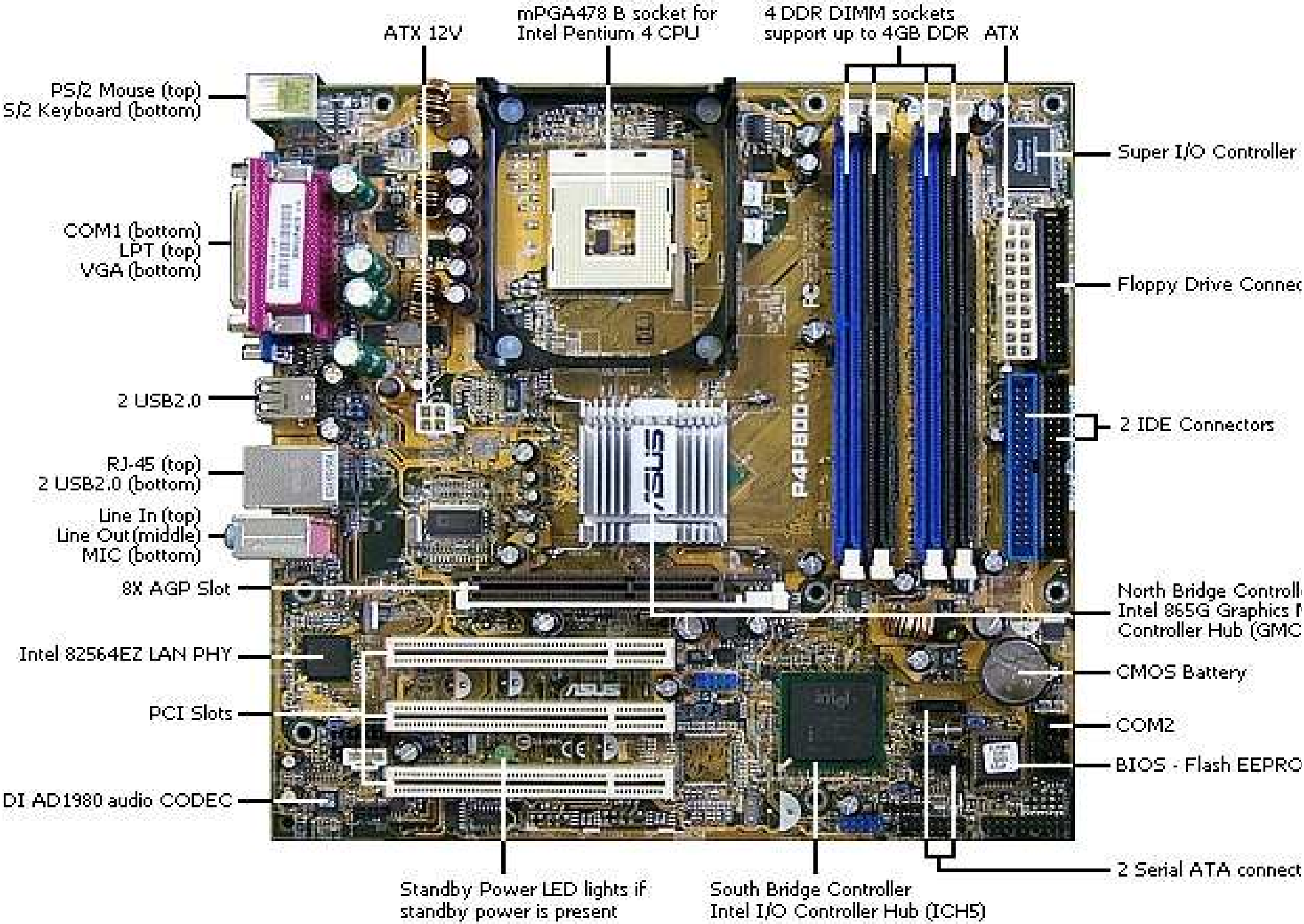
Bài 6 : Biến đổI địa chỉ sau thành địa chỉ tuyệt đối

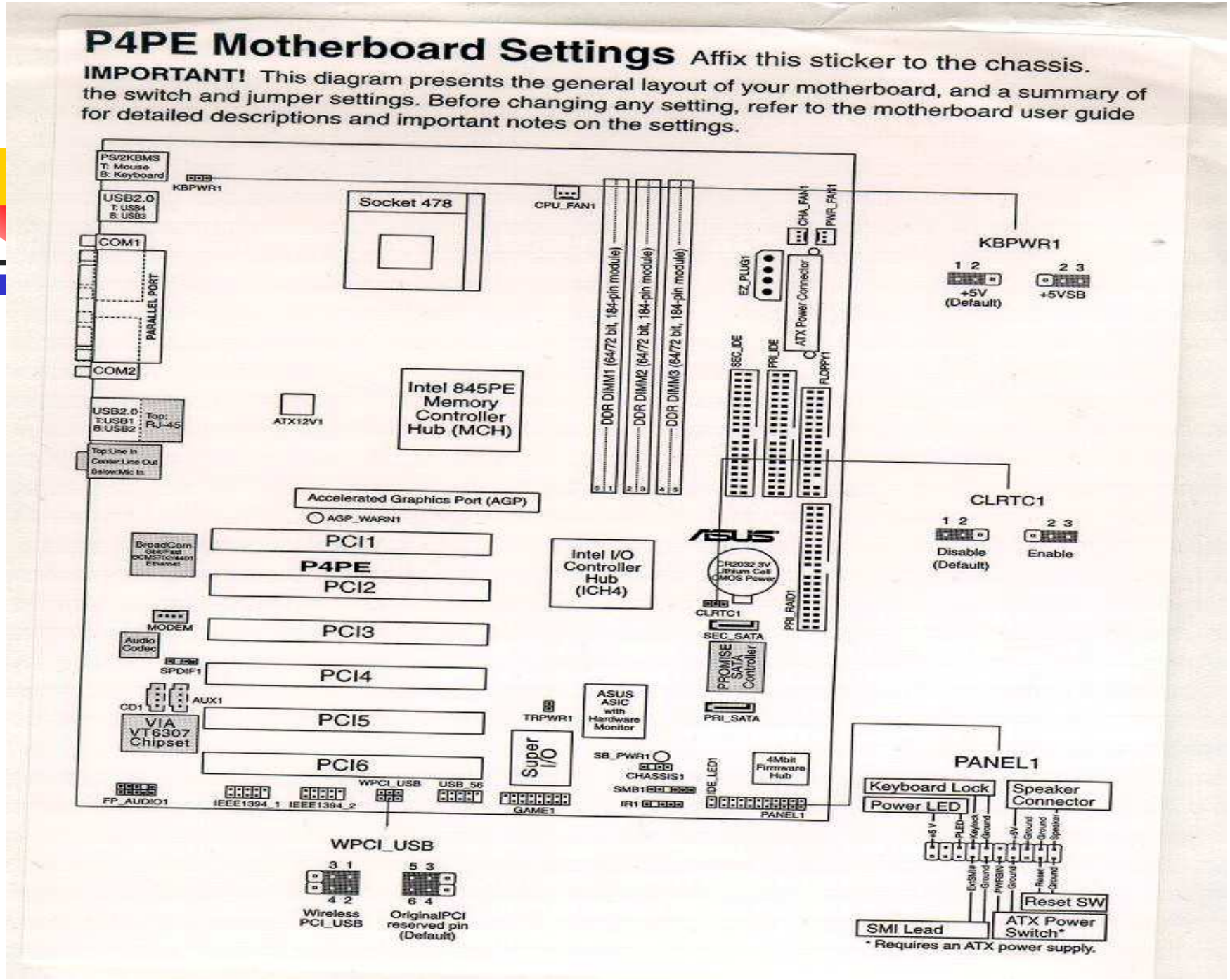
a. 0950:0100

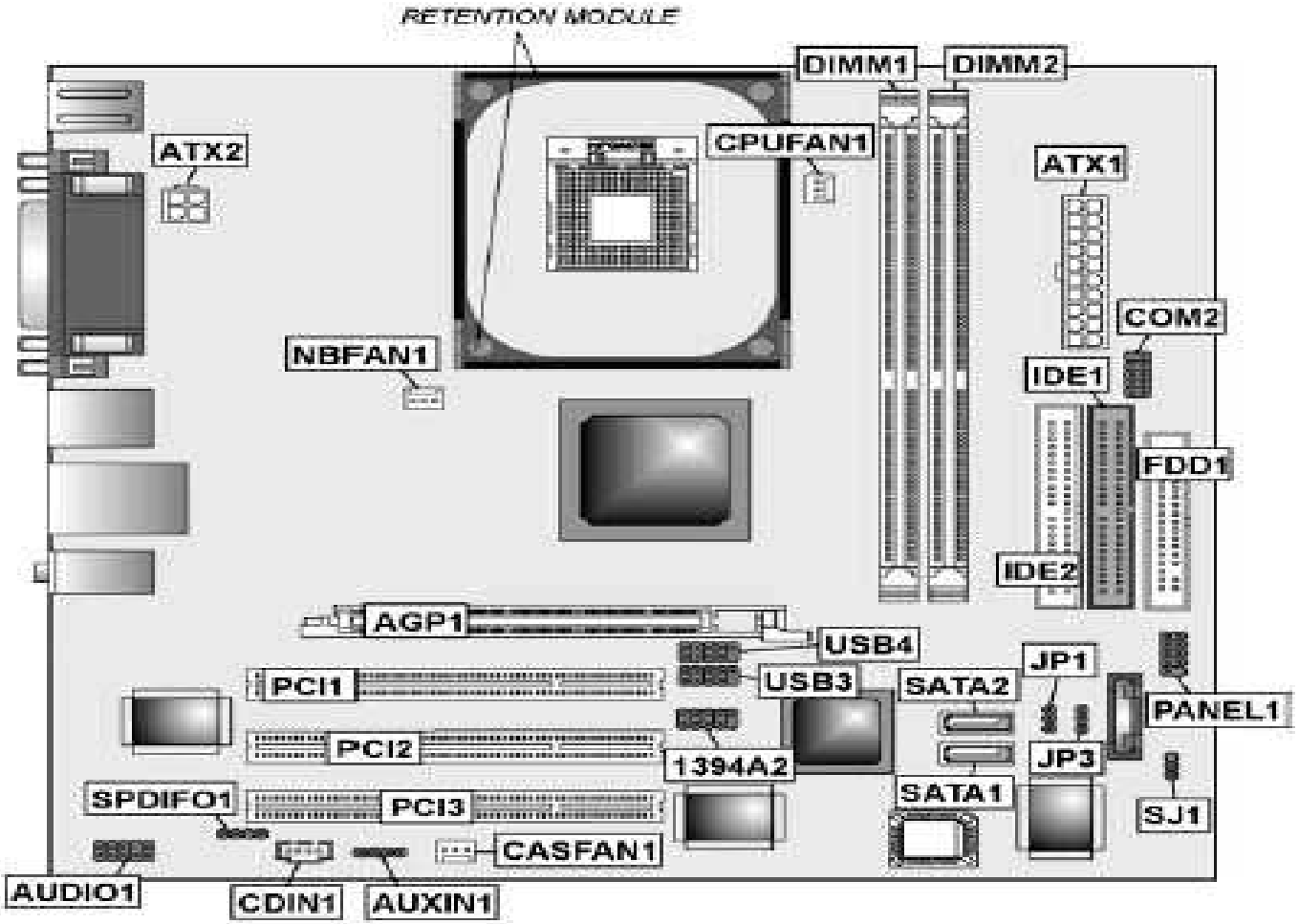
b. 08F1:0200


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

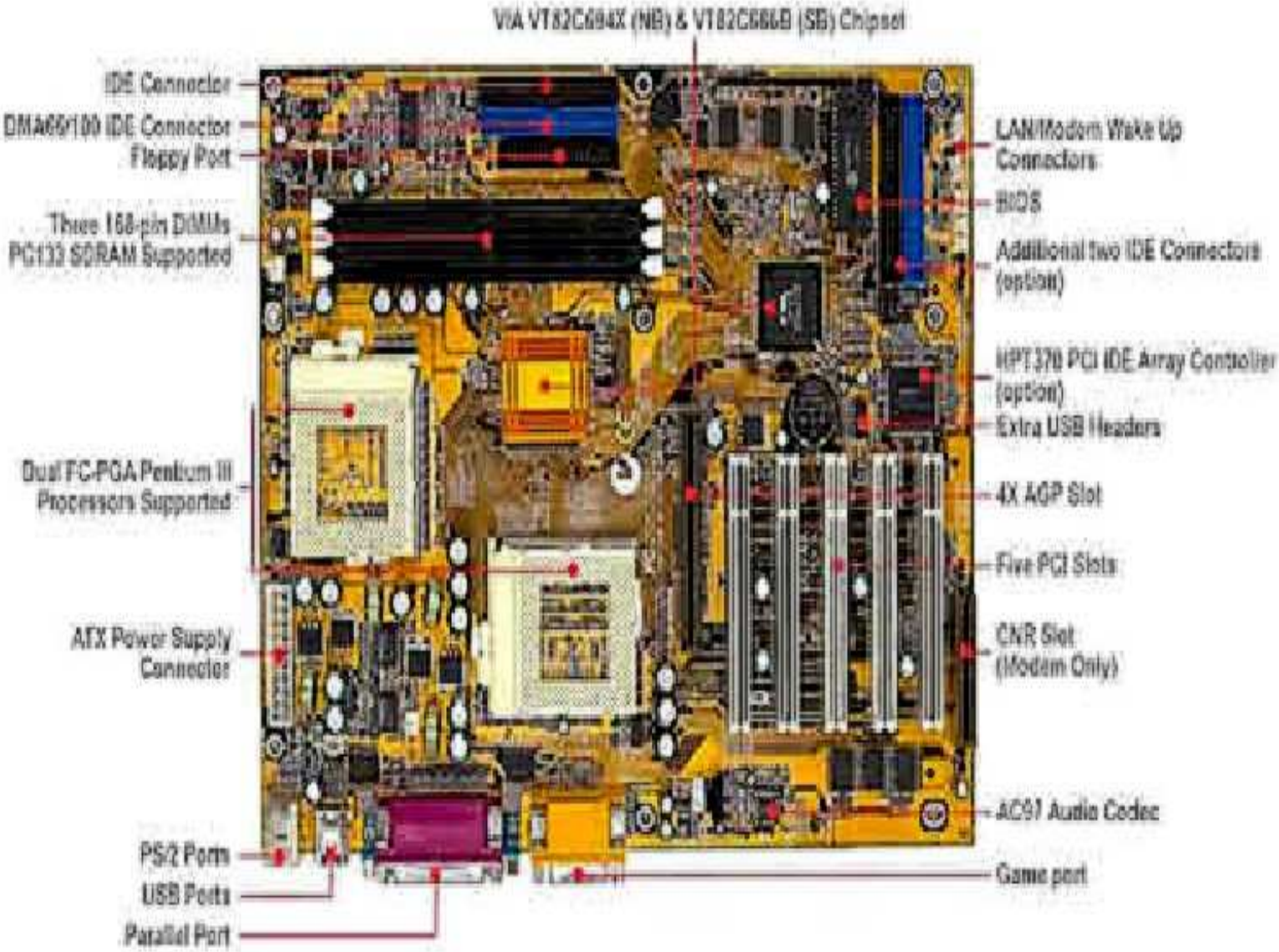

Socket 462 (A) for
AMD Athlon XP CPU
3 DDR DIMM sockets support up to
3GB, 400MHz, dual channel memory
PS/2 Mouse (top)
PS/2 Keyboard (bottom)
2 USB2.0
COM1 (bottom)
LPT (top)
VGA (bottom)
RJ-45 (top)
2 USB2.0 (bottom)
Line In (top)
Line Out (middle)
MIC (bottom)
Realtek 8201BL LAN PHY
8X AGP Slot
Warning 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

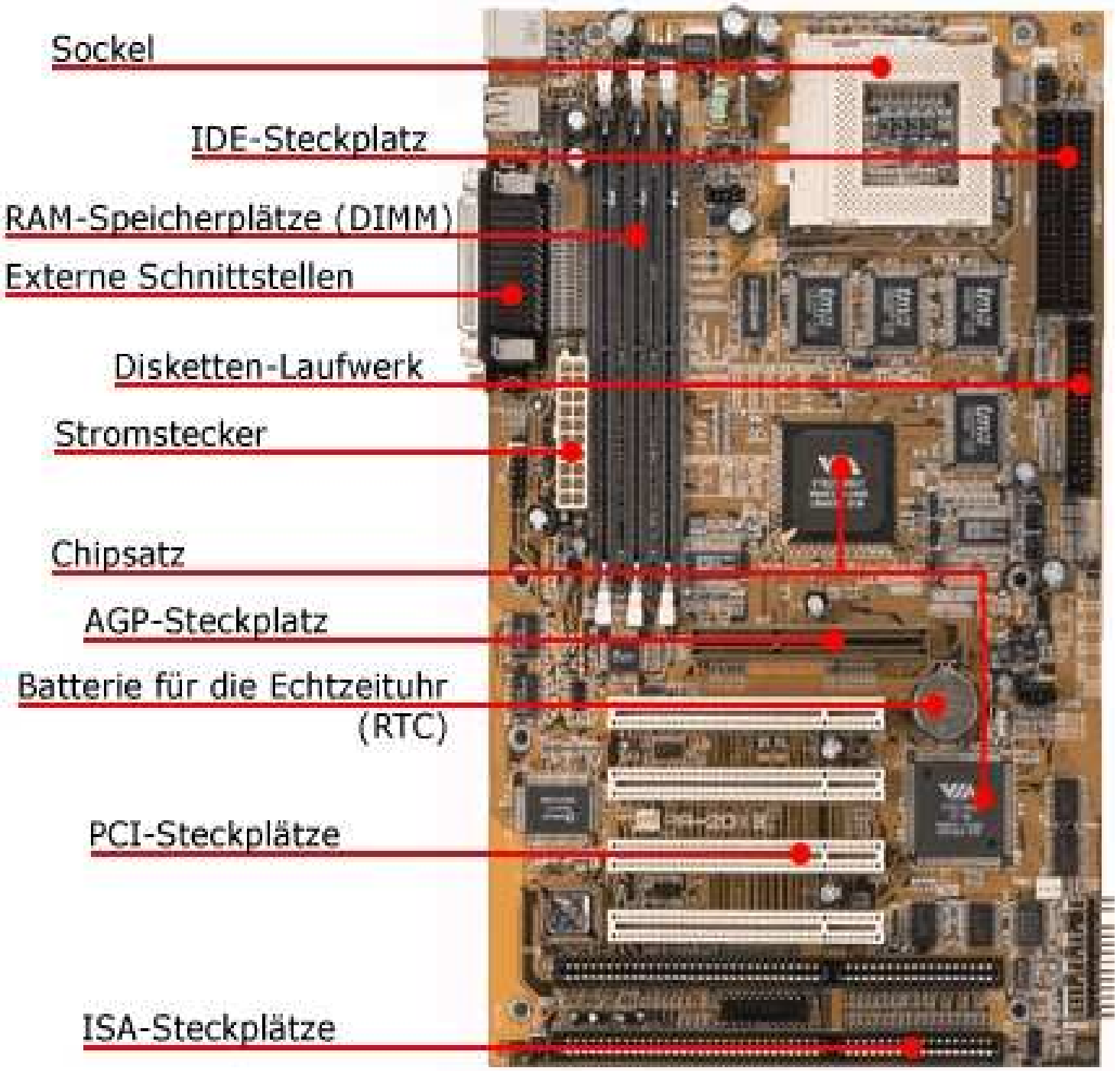

ATX 12V
mPGA478 B socket for
Intel Pentium 4 CPU
4 DDR DIMM sockets
support up to 4GB DDR
ATX
PS/2 Mouse (top)
S/2 Keyboard (bottom)
Super I/O Controller
COM1 (bottom)
LPT (top)
VGA (bottom)
Floppy Drive Connec
2 USB2.0
2 IDE Connectors
RJ-45 (top)
2 USB2.0 (bottom)
Line In (top)
Line Out(middle)
MIC (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)
P4P800-VM


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.
PS/2KBMS
T: Mouse
B: Keyboard
KBPWR1
USB2.0
T: USB4
B: USB3
Socket 478
CPU_FAN1
CHA_FAN1
PWR_FAN1
COM1
PARALLEL PORT
COM2
EZ_PLUG1
ATX Power Connector
SEC_IDE
PRI_IDE
FLOPPY1
USB2.0
T:USB1
B:USB2
Top: RJ-45
ATX12V1
Intel 845PE Memory Controller Hub (MCH)
DDR DIMM1 (64/72 bit, 184-pin module)
DDR DIMM2 (64/72 bit, 184-pin module)
DDR DIMM3 (64/72 bit, 184-pin module)
Top:Line In
Center:Line Out
Below:Mic In
Accelerated Graphics Port (AGP)
AGP_WARN1
PCI1
P4PE
PCI2
PCI3
PCI4
PCI5
PCI6
Intel I/O Controller Hub (ICH4)
BroadCom
MODEM
Audio Codec
SPDIF1
CD1
AUX1
VIA VT6307 Chipset
TRPWR1
ASUS ASIC with Hardware Monitor
Super I/O
SB_PWR1
CHASSIS1
SMB1
IR1
4Mbit Firmware Hub
IDE_LED1
PANEL1
CR2032 3V Lithium Cell CMOS Power
CLRTC1
SEC_SATA
PRI_RAID1
PROMISE SATA Controller
PRI_SATA
FP_AUDIO1
IEEE1394_1
IEEE1394_2
WPCI_USB
USB_56
GAME1
KBPWR1
1 2
+5V (Default)
2 3
+5VSB
CLRTC1
1 2
Disable (Default)
2 3
Enable
PANEL1
Keyboard Lock
Power LED
Speaker Connector
Reset SW
SMI Lead
ATX Power Switch*
* Requires an ATX power supply.
WPCI_USB
3 1
4 2
Wireless PCI_USB
5 3
6 4
OriginalPCI reserved pin (Default)


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 leänh cuûa 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 chuổi 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.

Caùch goïi : D <khoảng>

Ex : - D F000 : 0

- D ES : 100

- D 100

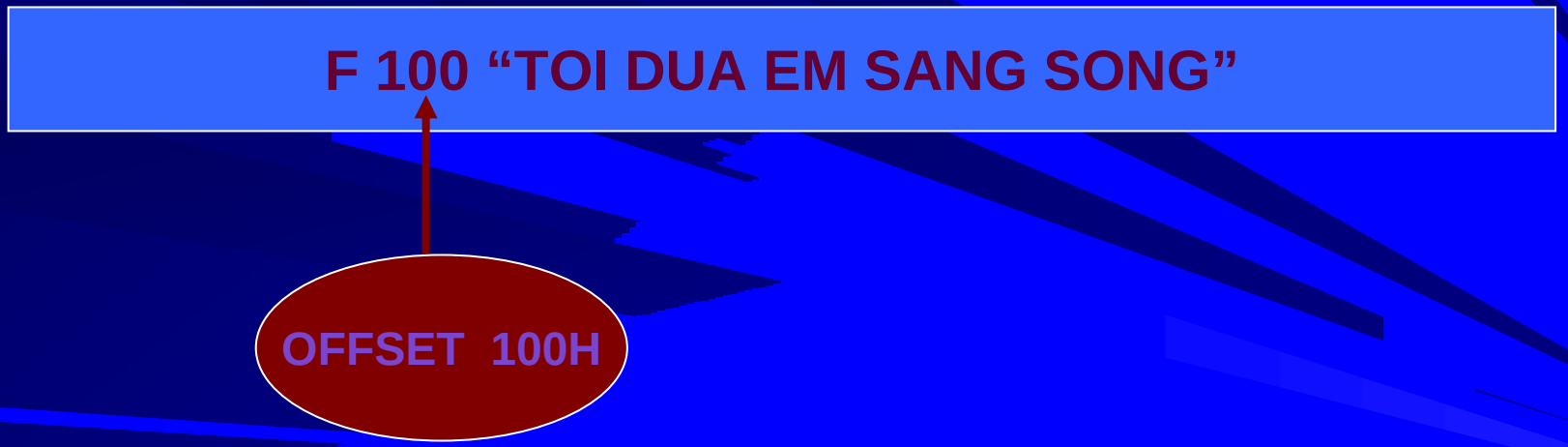
Lệnh F (Fill)

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

Trị nhập vào từng byte một theo hệ 16
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 chuổi “ 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]

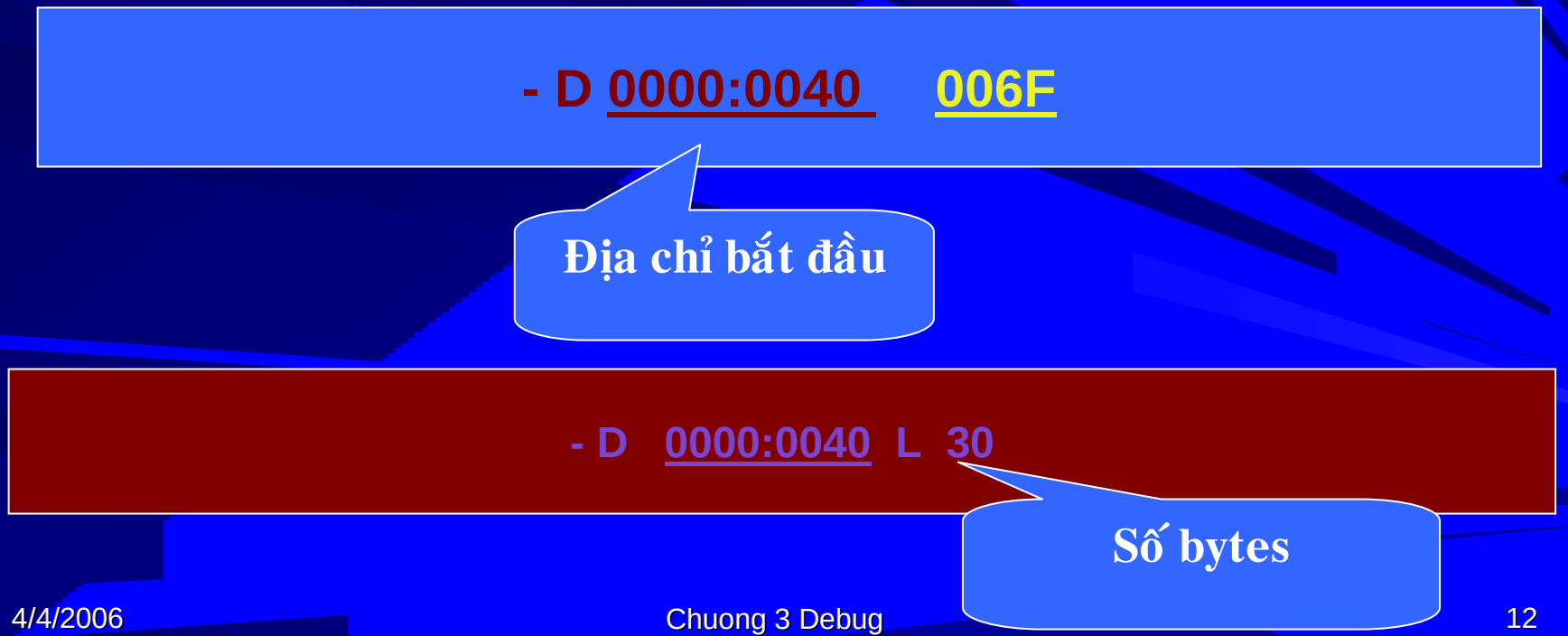
Ex : in nội dung vùng nhớ đã lắp đầy ở ví dụ trước ở địa chỉ 100h

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

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



E (ENTER)

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

Caùch goïi :

- E <địa chỉ> <danh sách>

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

Minh họa lệnh E

- Mục 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

Leänh 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.**

- Thoaùt khoûi Debug vaø goïi laïi taäp tin theo caùch sau :
  C :\> Debug Love.txt
  tìm xem Debug ñaõ naïp taäp tin Love.txt vaøo choã naøo trong boä nhôù.

Lệnh W (Write)

Cú pháp : W [address]

Thöôøng ñöôïc söû duïng chung vôùi leänh N

Ex : taïo taäp tin coù teân Love.txt

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

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

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

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

Ở đây số byte cần ghi là 1Eh.

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

Leänh T (Trace)vaø P

- cuù phaùp : - T [= <ñiaï chæ>][soá laàn]

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

Ex : - T = 3000:1000

Ex : - T = 3000:1000 <soá laàn>

Leänh L (Load)

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

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

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

Taäp tin luoân luoân ñöôïc gaùn ôû ñòa chæ offset 100h

Daïng 2 : neáu coù ñaày ñuû caùc thoâng soá , duøng ñeå ñoïc sector luaän lyù treân ñóa vaøo boä nhôù.

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

Leänh H (Hex Arithmethic)

- thöïc hieän pheùp coäng vaø tröø heä 16

Cuù phaùp : - H <trò 1> <trò 2>

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

Lệnh S (Search)

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

  18AF : 0154

  18AF : 0823

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

Lệnh M (Move)

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

Chép 5 bytes từ DS:100 đến DS:200

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

Chép 50 bytes từ CS:100 đến ES:300

Lệnh I (Input)

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

Lệnh O (Output)

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

Summary

- Dùng lệnh D để xem nội dung vùng nhớ tại địa chỉ của ROM BIOS F000:0000.
- Tương tự xem nội dung vùng nhớ RAM màn hình ở địa chỉ B800:0000; bảng vector ngắt quãng 0000:0000
- 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.

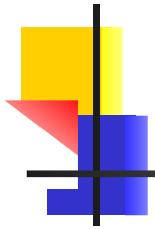
All right reserved, including the of reproduction in whole or in part, in any form.

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

BỘ NHỚ (Memory)

Mục tiêu :

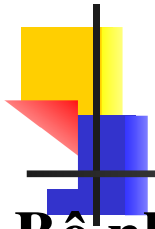
1. Hiểu được cấu tạo của bộ nhớ, chức năng và hoạt động của bộ nhớ.
2. Nắm được quá trình đọc bộ nhớ & ghi bộ nhớ.
3. Vai trò của bộ nhớ Cache trong máy tính.



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

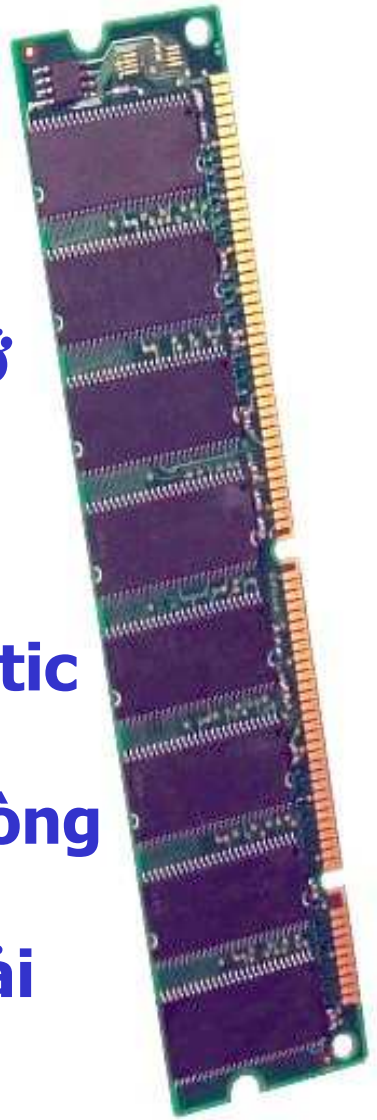
RAM

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 □iểm của RAM :

- Cho phép □ọc/ 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



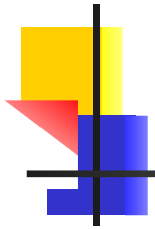
Ram là vùng nhớ làm việc ➔ nếu vùng nhớ này trở nên nhỏ so với nhu cầu sử dụng thì ta tăng thêm Ram (gắn thêm Ram).

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

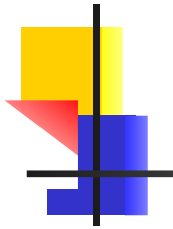


ROM

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

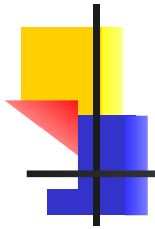
Là bộ nhớ phụ nằm ngoài hộp CPU.

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

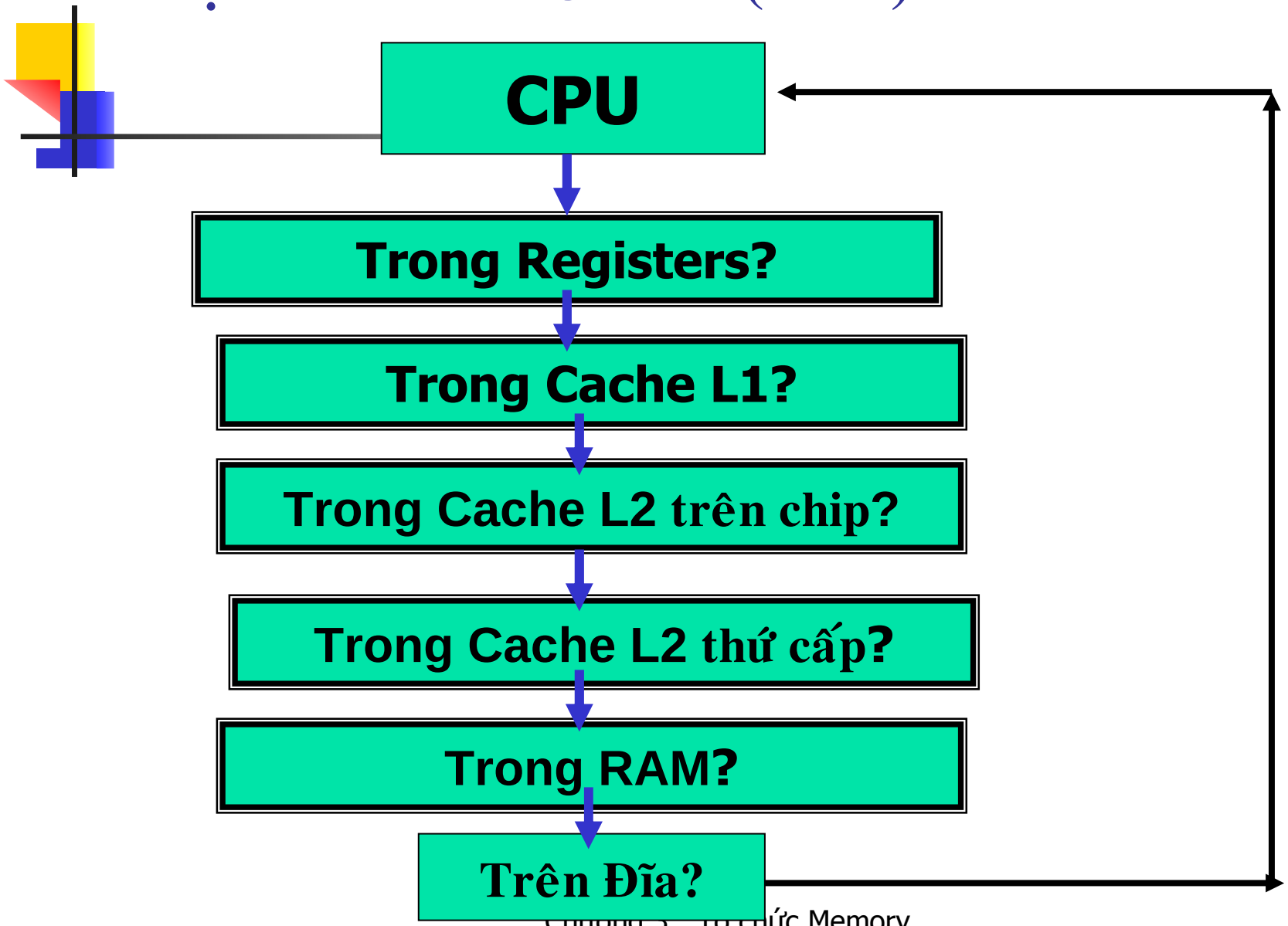
Tỉ lệ thất bại (miss ratio) (1-h)

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

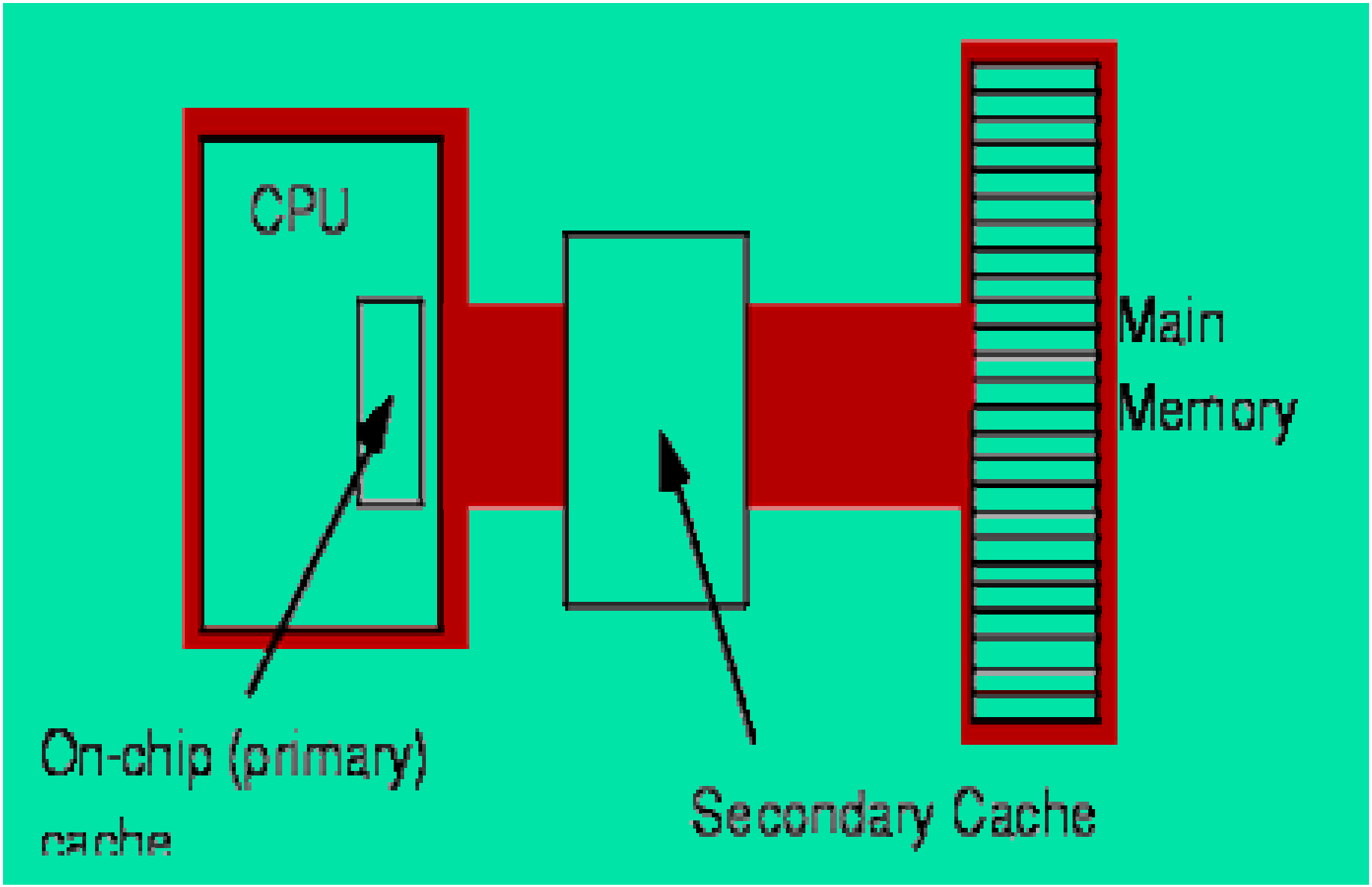
Khi h→ 1, tất cả truy xuất đều tham chiếu tới Cache, thời gian truy xuất trung bình → c.

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



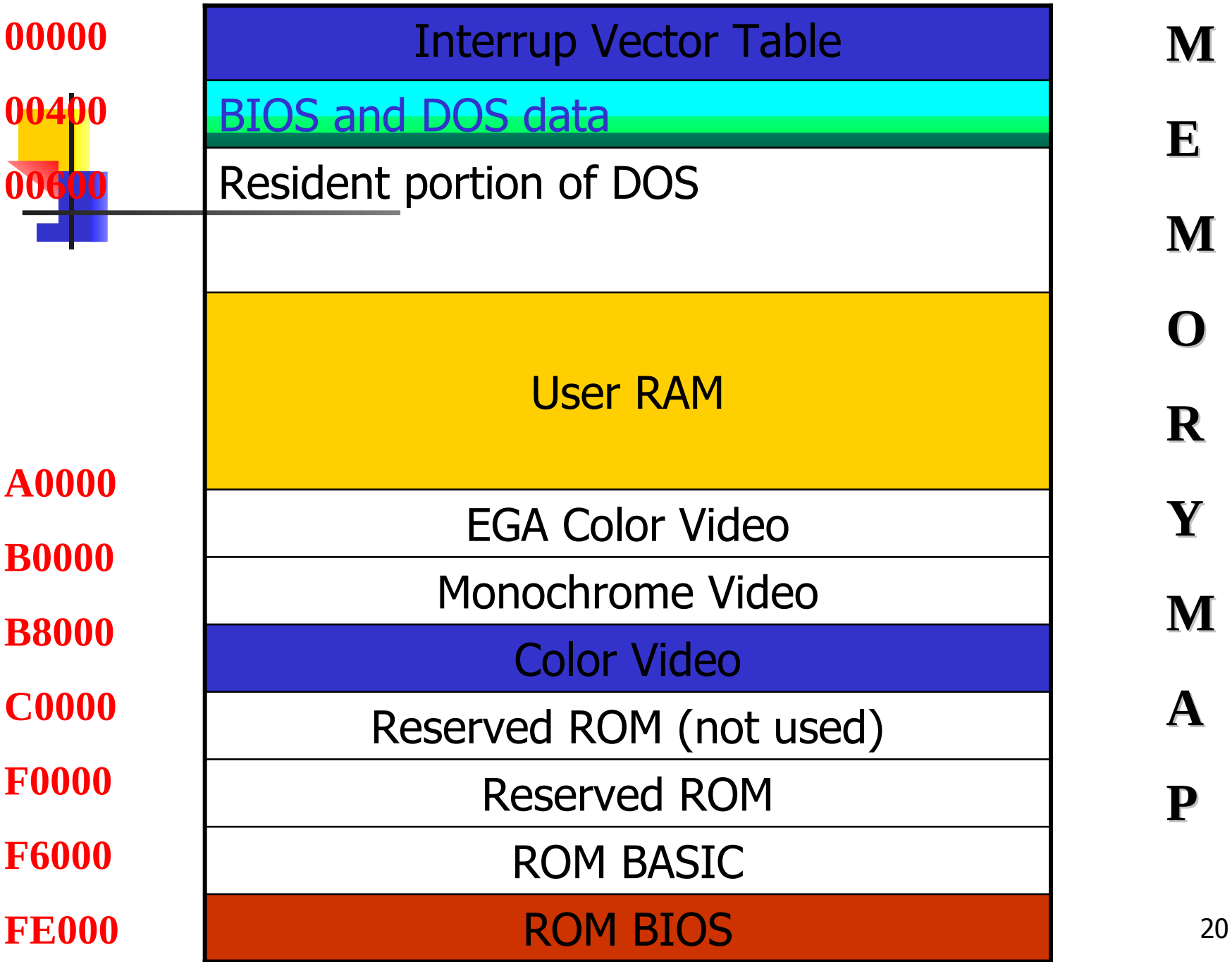
Các chiến lược trữ đệm trong Cache

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

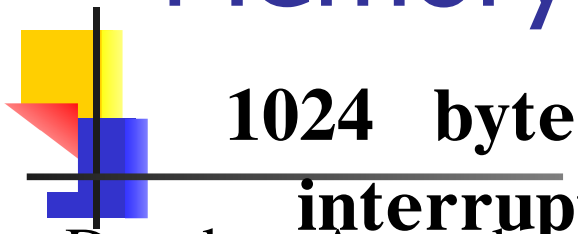
- **Khi CPU đọc từ bộ nhớ qui ước thì WTC và WBC đều như nhau : sẽ đọc 1 đoạn nội dung trong bộ nhớ vào cache.**
- **Khi CPU ghi ra bộ nhớ qui ước :**

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
MEMORY MAP

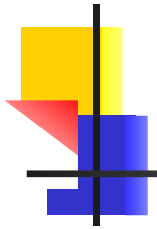
Memory Map



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...

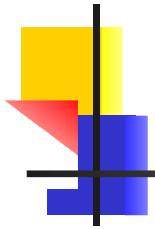


Memory Map

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..

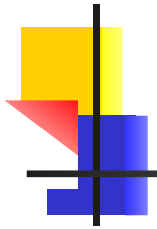


Quá trình Boot máy

- **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.



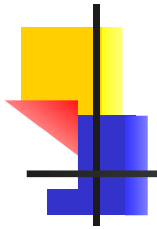
Trình tự tác vụ đọc ô nhớ

✓ 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

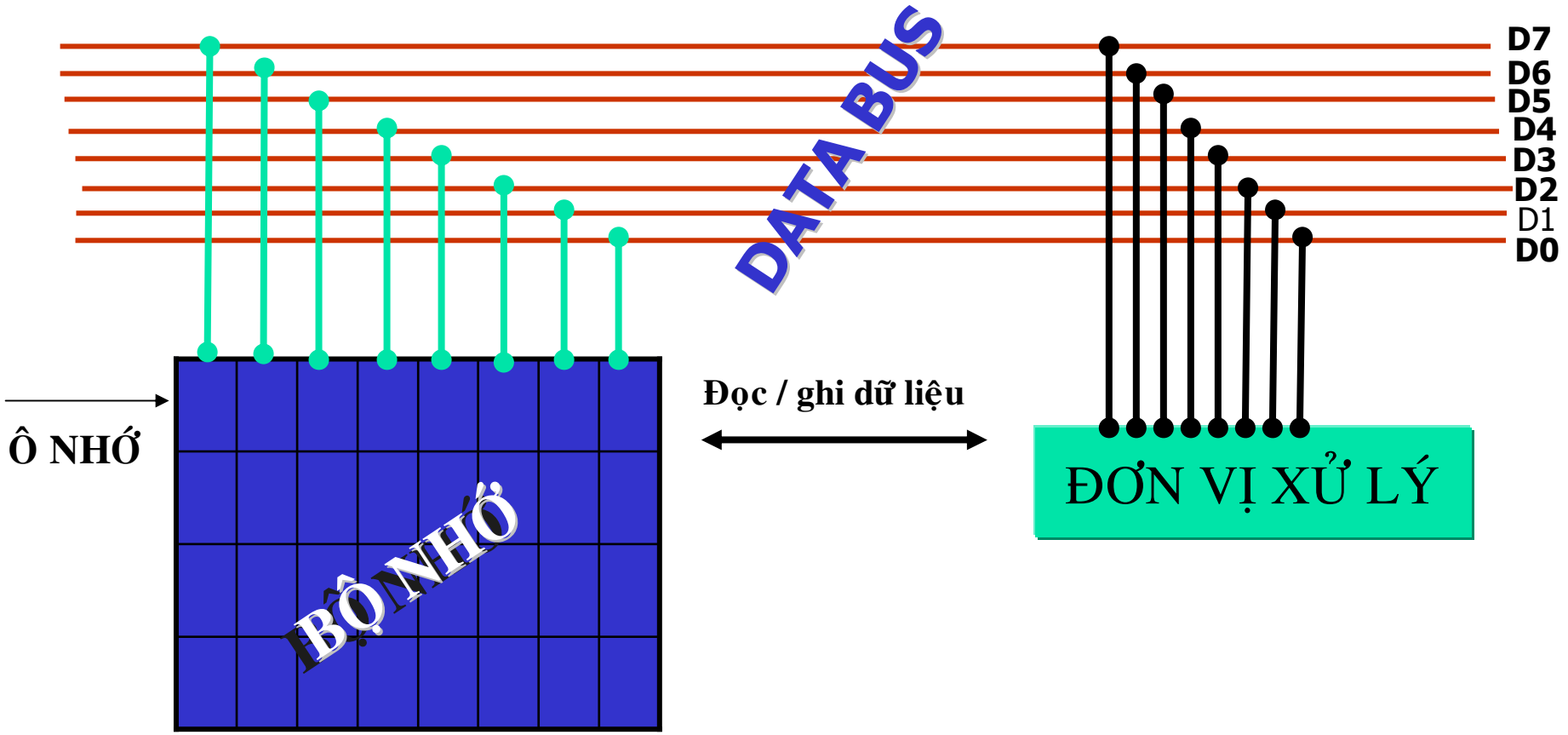
Mạch điện có nhiệm vụ xác định đúng ô nhớ cần truy xuất đang có địa chỉ lưu trong thanh ghi địa chỉ.

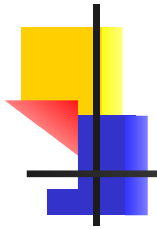
Bộ nhớ làm việc được chia thành nhiều ô nhớ.

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)

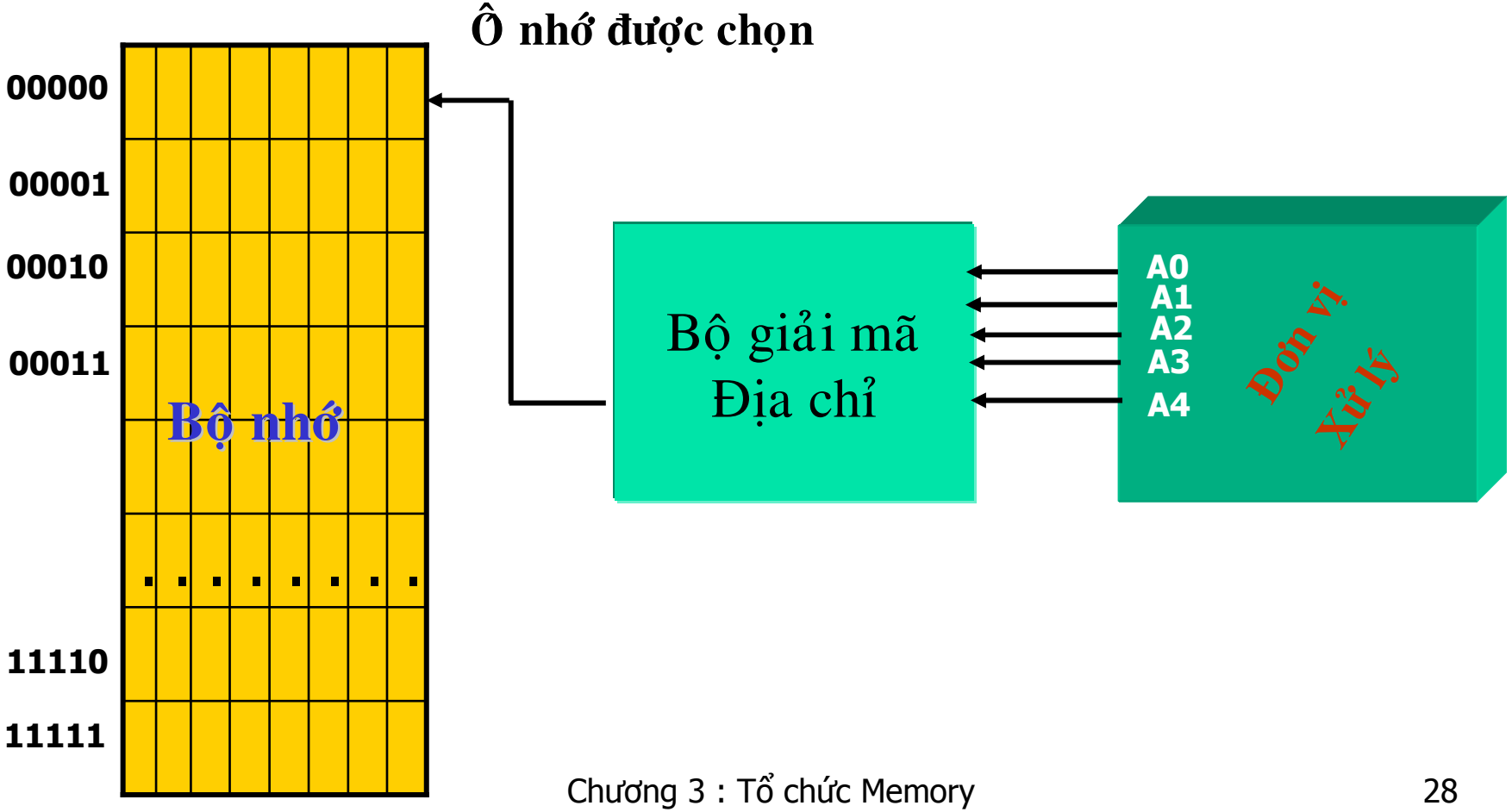


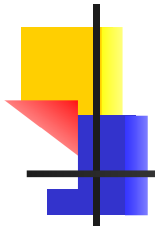


Trình tự tác vụ ghi ô nhớ

- **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



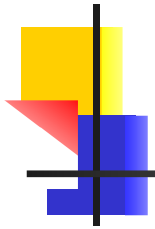


Stack

- **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.)

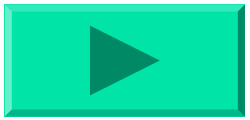


Stack

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

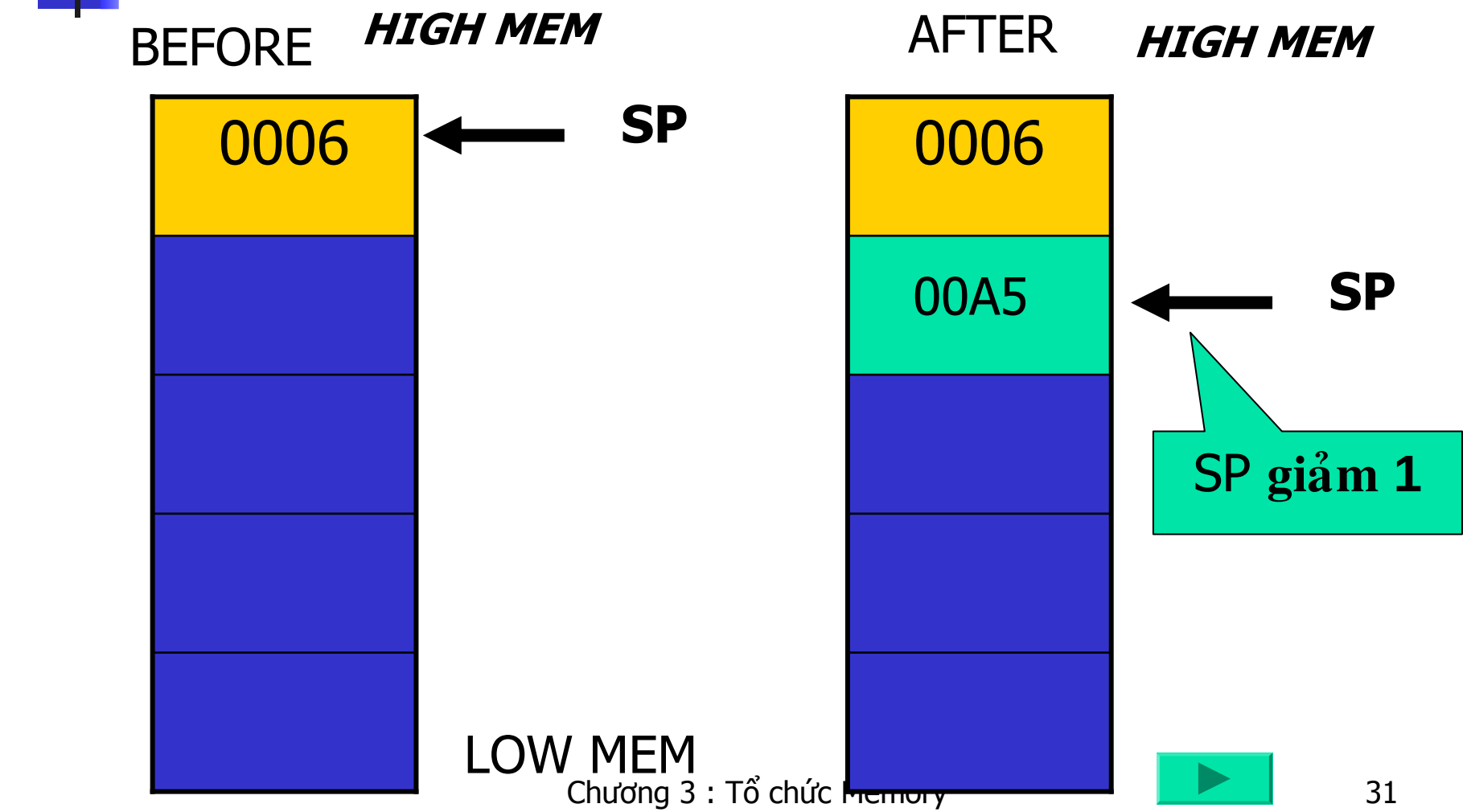
➔ Stack làm việc theo cơ chế LIFO (Last In First Out).

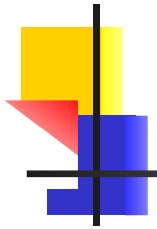
Xét ví dụ sau : giả sử stack đang chứa 1 giá trị 0006



Sau đó ta đưa 00A5 vào stack

Stack


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

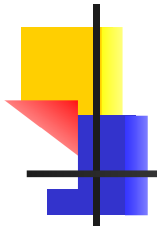


Công dụng của Stack

- Dùng để lưu trữ dữ liệu tạm cho thanh ghi nếu ta cần sử dụng các dữ liệu này.

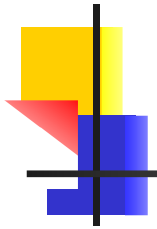
- Khi 1 chương trình con được gọi, stack sẽ lưu trữ địa chỉ trở về ngay sau khi chương trình con thực hiện xong.

- 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.



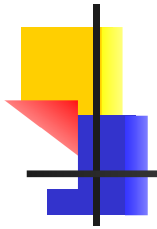
Summary Slide

- 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.



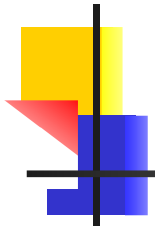
Câu hỏi ôn tập

- 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

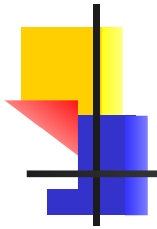
Bộ nhớ Cache nằm giữa :

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 I 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

h2

Chương 5 : Nhập môn Assembly

Mục tiêu

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

Slide 1

h1 shjsahjsa
huh, 13/10/2004

h2 ssasasasas
huh, 13/10/2004

Giới thiệu ngôn ngữ Assembly

- Giúp khám phá bí mật phần cứng cũng như phần mềm máy tính.
- Nắm được cách phần cứng MT làm việc với hệ điều hành và hiểu được bằng cách nào 1 trình ứng dụng giao tiếp với hệ điều hành.
- Một MT hay một họ MT sử dụng 1 tập lệnh 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 lệnh cho CPU thực hiện tác vụ.
  - Tác vụ đó có thể là :
    di chuyển 1 số từ vị trí nhớ này sang vị trí nhớ khác.
    Cộng 2 số hay so sánh 2 số.

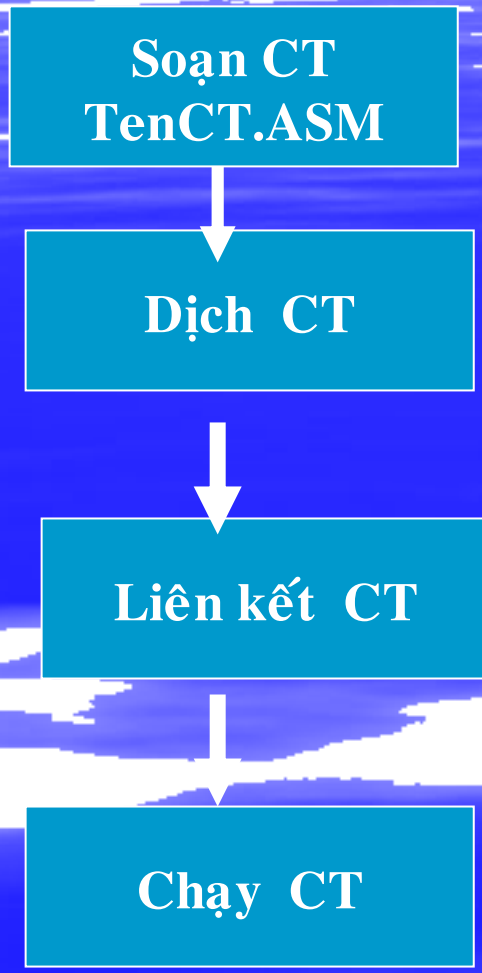
00000100	Add a number to the AL register
10000101	Add a number to a variable
10100011	Move the AX reg to another reg

Lệnh máy (cont)

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

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

Cách viết 1 chương trình Assembly



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

CT có phần mở rộng là .ASM

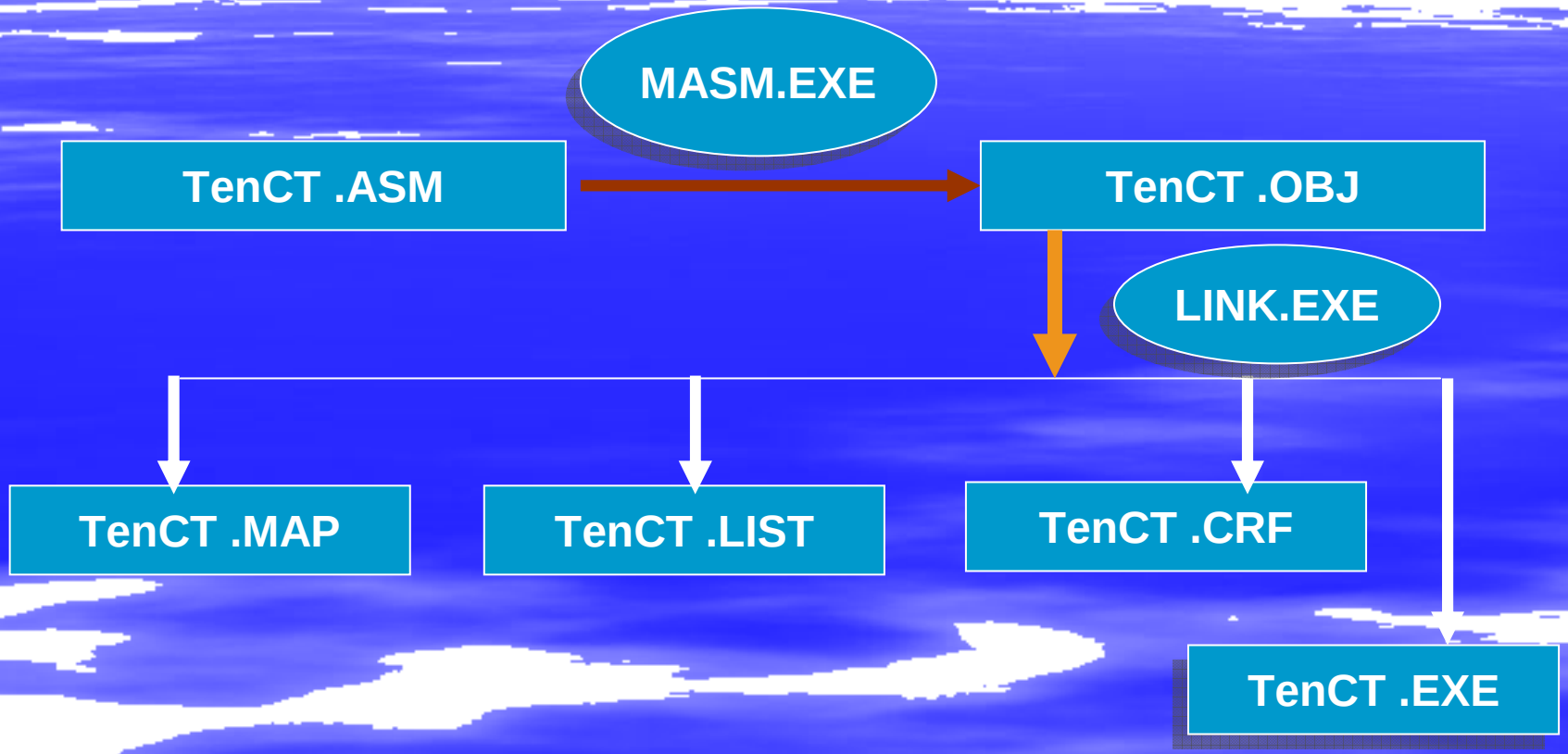
dùng MASM để dịch chương trình nguồn .ASM

➔ 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

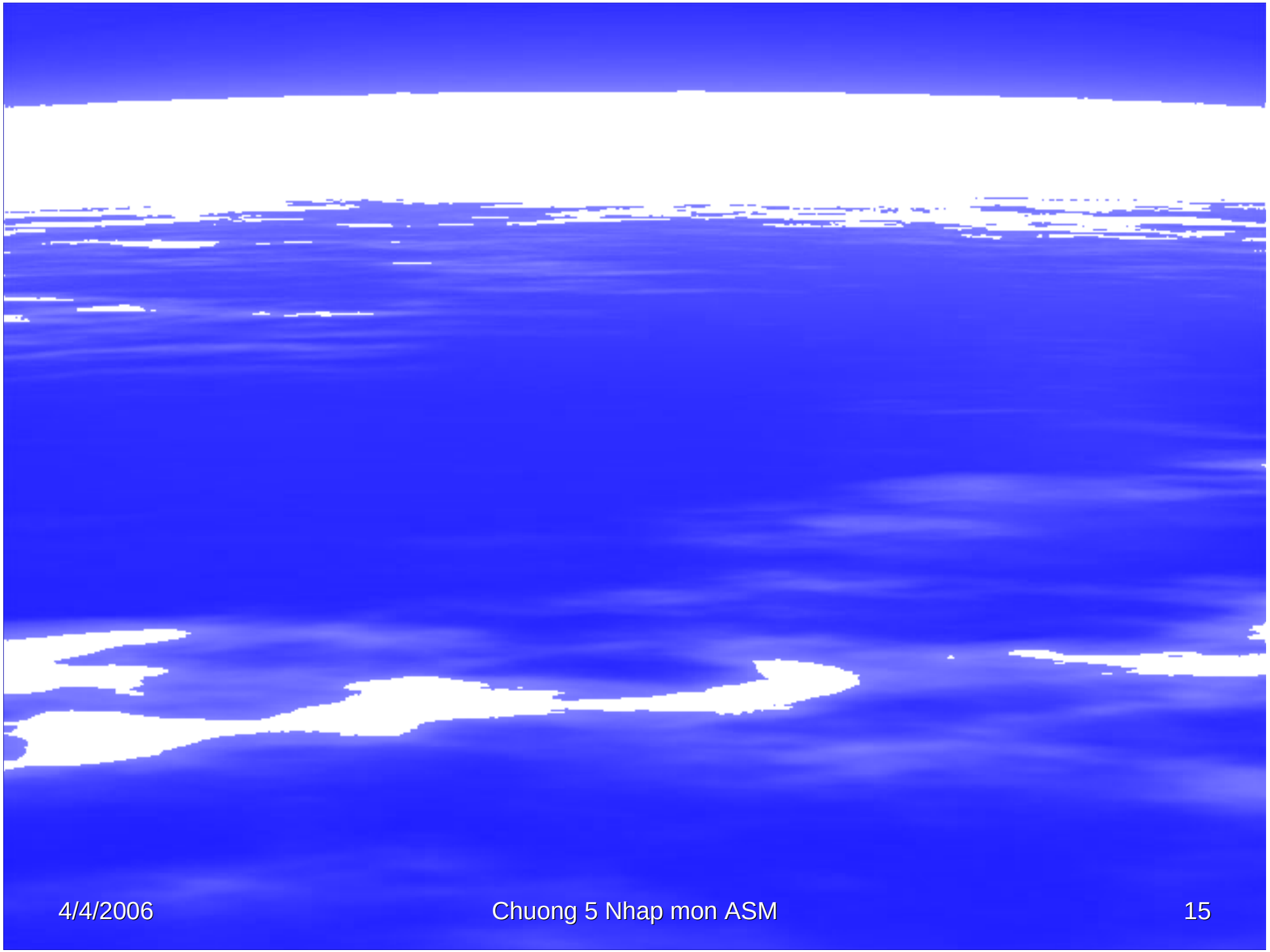
- Program entry point at 0000:0010

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 chuổi ký tự ra màn hình.
- Thoát khỏi CT .
- Main endp : đánh dấu kết thúc thủ tục
- End main : chấm dứt chương trình



Các chế độ bộ nhớ

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

Dạng lệnh

Chuù thích

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

Nhãn, tên biến
Tên thủ tục

Mã lệnh dạng
gợi nhớ

Register, ô nhớ
Trị, hằng

Ex: MOV CX , 0

LAP : MOV CX, 4

DB 1,2,3,4

Mỗi dòng chỉ chứa 1 lệnh và mỗi lệnh
hải nằm t 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 khiển được ấn

Hàm 2 : Hiển thị 1 ký tự ra màn hình

Input : AH =2
DL = Mã ASCII của ký tự hiển thị hay ký tự điều khiển

Thí dụ minh họa

```
DOSSEG
.MODEL SMALL
.STACK 100H
.CODE
  MAIN PROC
     MOV AH , 2
     MOV DL , '?'
     INT 21H
     MOV AH ,1
     INT 21H
     MOV BL,AL
   MOV  AH,2
   MOV DL, 0DH
   INT 21H
   MOV DL , 0AH
   INT 21H
   MOV DL , BL
   INT 21H
   MOV AX , 4C00H
   INT 21H
   MAIN ENDP
END MAIN
```

KẾT QUẢ

```
? N
N
```

Thí dụ minh họa các hàm của INT 21

- **In dấu ? ra màn hình :**

```
MOV AH, 2
MOV DL, '?'
INT 21H
```

- **Nhập 1 ký tự từ bàn phím :**

```
MOV AH, 1
INT 21H
```

Biến

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

```
.data
   aList  db "ABCD"
```

Biến (cont)

Lúc đó :

Offset	Biến
0000	A
0001	B
0002	C
0003	D

Khai báo biến

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

Minh họa khai báo biến

KIỂU BYTE

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

Minh họa khai báo biến

KIỂU WORD

DW 3 DUP (?)

DW 1000h, ‘AB’, 1024

DW ?

DW 5 DUP (1000h)

DW 256*2

DẠNG LƯU TRỮ DỮ LIỆU KIỂU WORD :

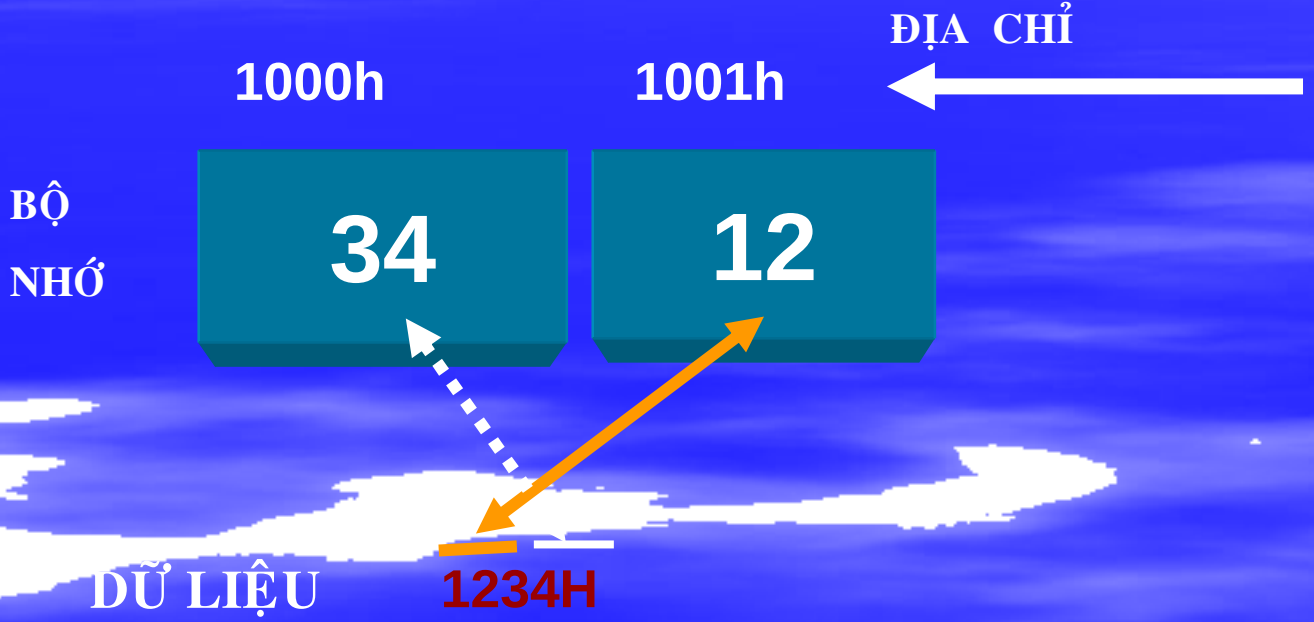
Trình hợp dịch đảo ngược các byte trong 1 giá trị kiểu WORD khi lưu trữ trong bộ nhớ :

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))

Taïo 1 ma traän 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 chuổi.

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 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
  N ENDP
     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
```

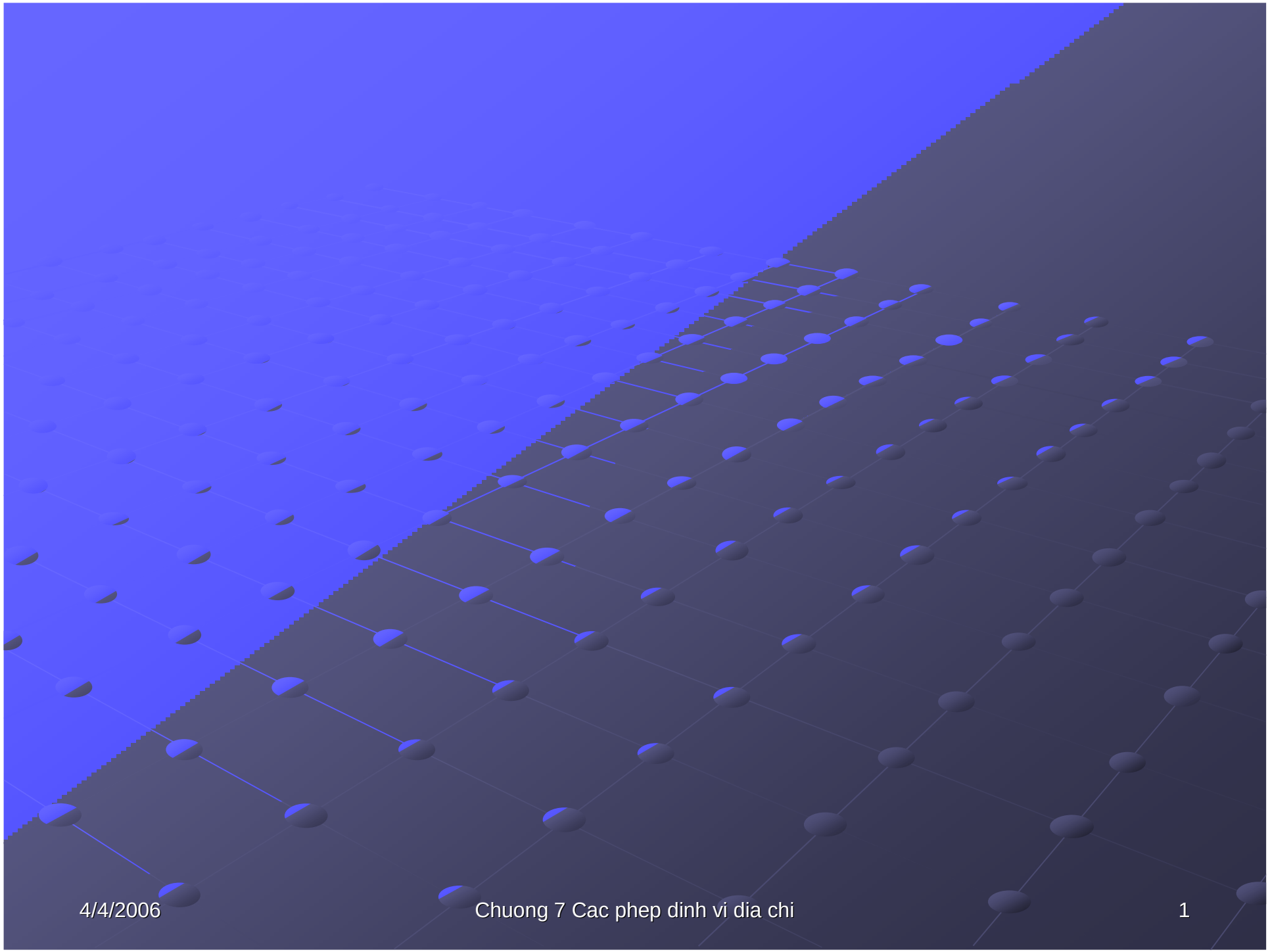
Bài tập lập trình

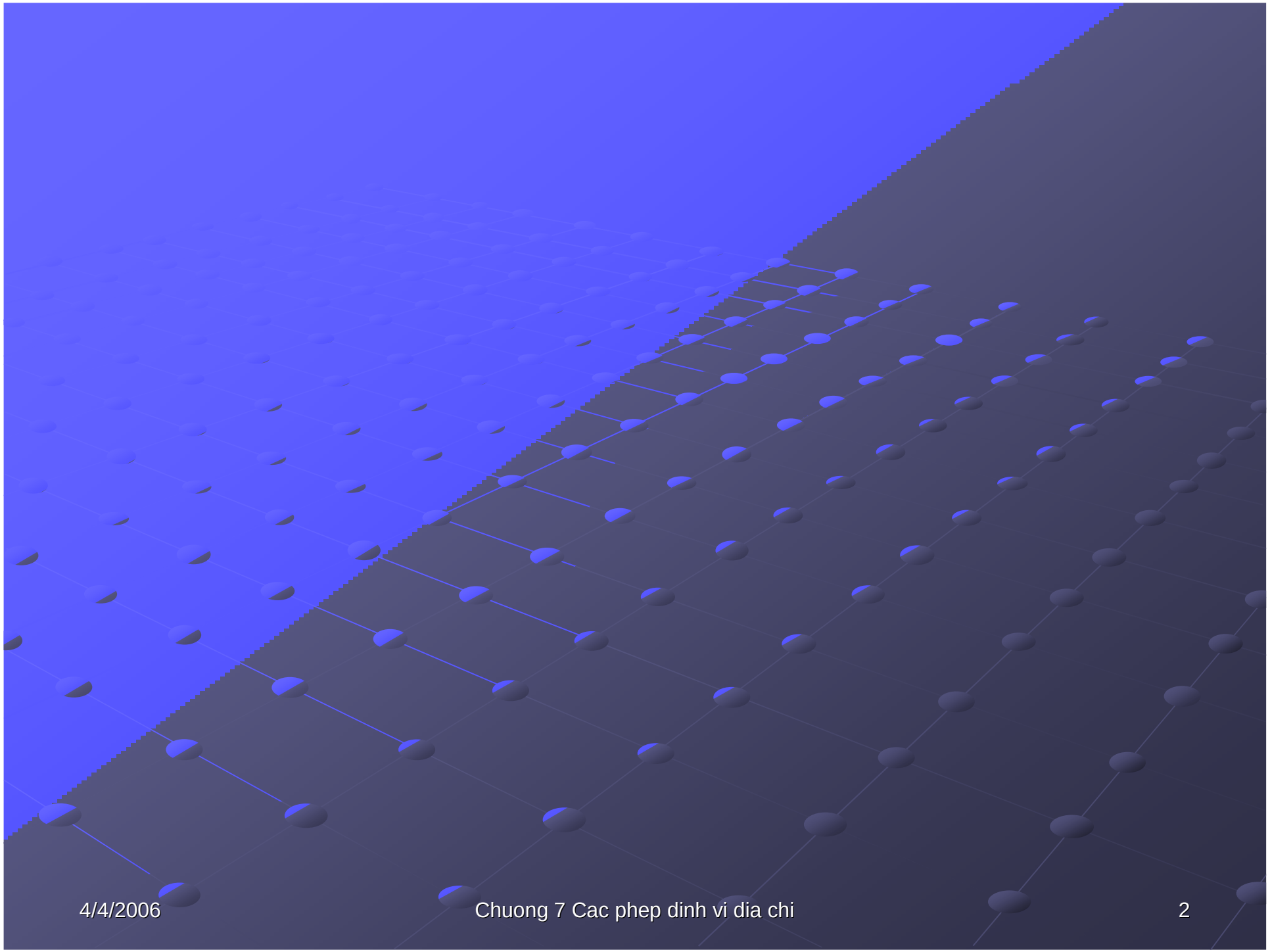
Bài 1 : Viết chương trình nhập 1 ký tự thường , in ra ký tự hoa tương ứng.

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 số học

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

Toán tử logic

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

Ex : MOV AH , 8 OR 4 AND 2

MOV AL, NOT (20 XOR 0011100B)

Toán Tử Quan Hệ

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

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

ĐỘ ƯU TIÊN TOÁN TỬ

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

Toán tử SEG

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

Toán tử OFFSET

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

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

```
TABLE  DB ?

MOV AX, SEG TABLE

MOV DS, AX

MOV DX, OFFSET  Table
```

TOÁN TỬ $

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

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.

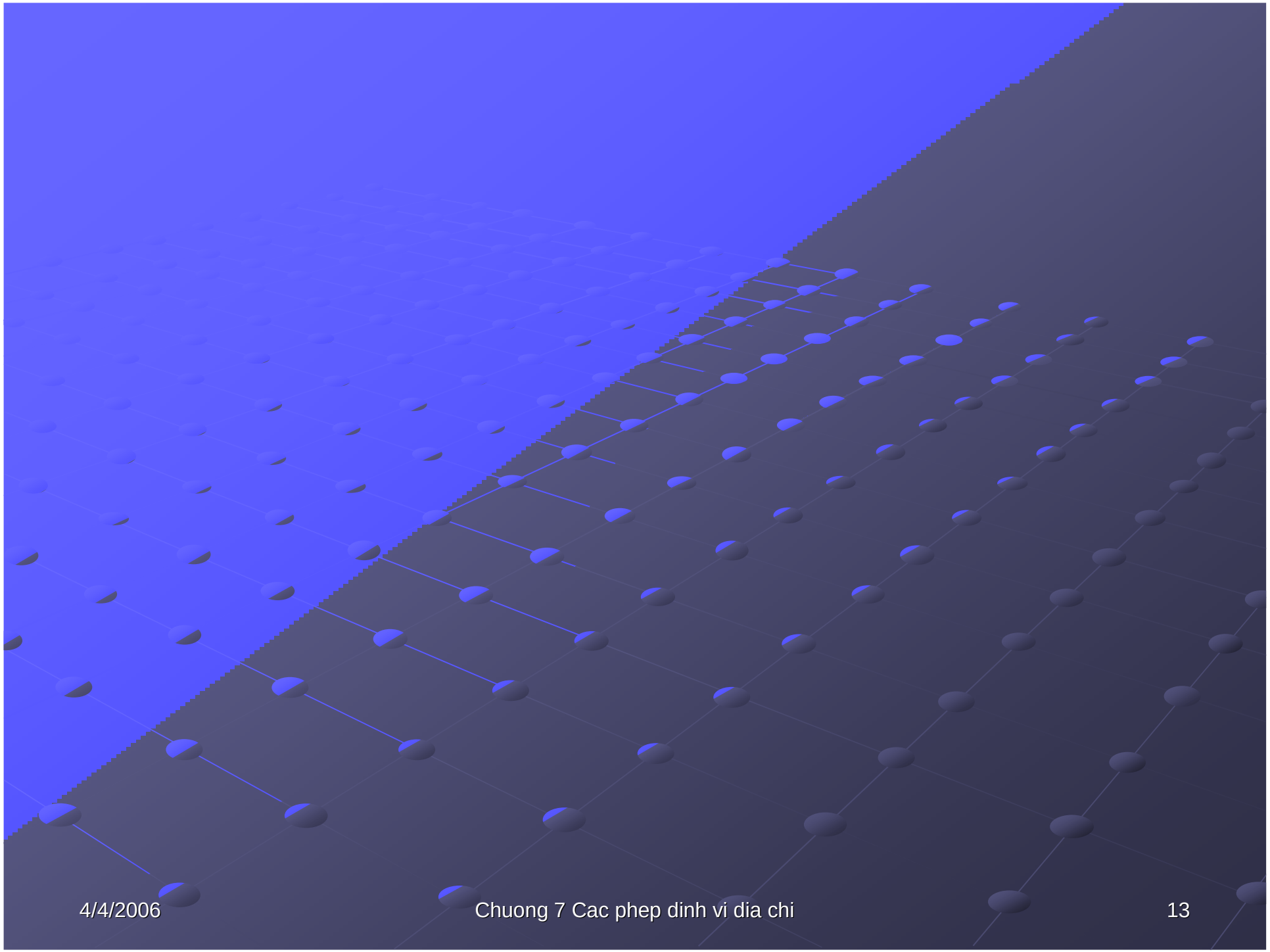
Với các lệnh 2 toán hạng thì toán hạng thứ 2 là toán hạng nguồn (source) – chứa dữ liệu hoặc địa chỉ của dữ liệu.

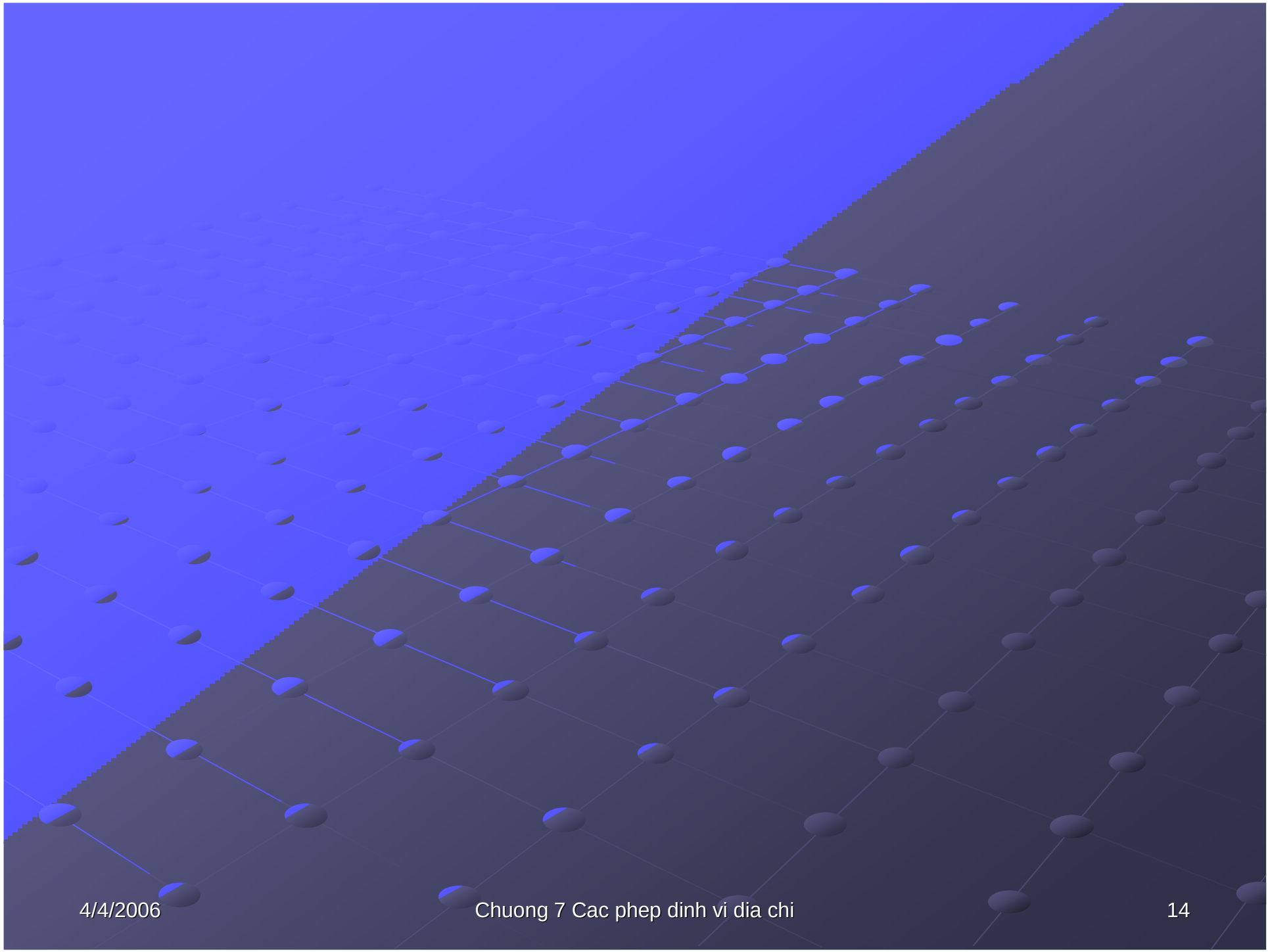
Toán hạng (Operand)

- 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)

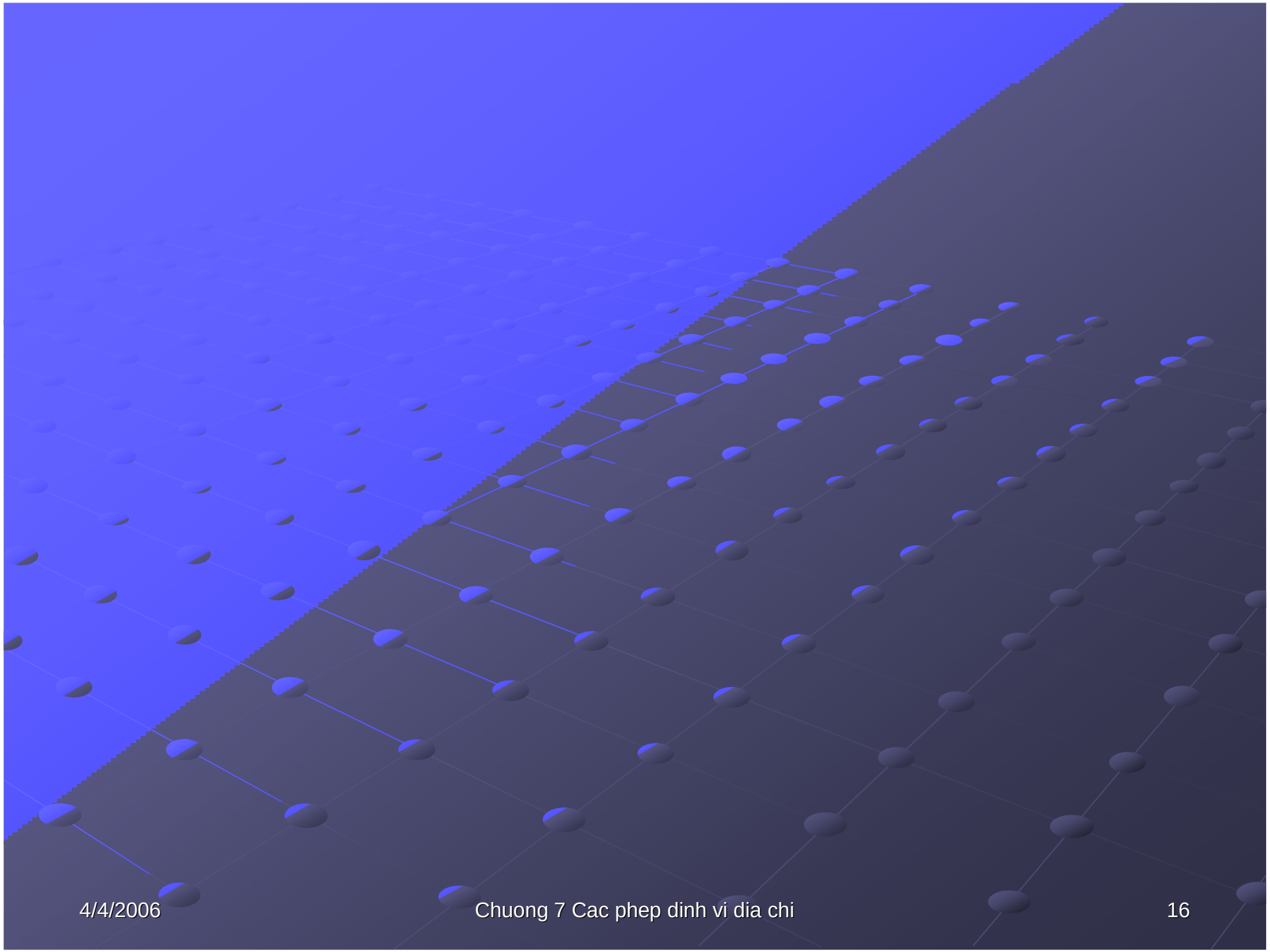




ĐỊNH VỊ THANH GHI

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

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

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

Nhóm thanh ghi chỉ số : SI , DI
Nhóm thanh ghi nền : BX, BP
Địa chỉ trực tiếp : số 16 bit

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

ĐỊA CHỈ HIỆU DỤNG

Một số thí dụ

Địa chỉ hiệu dụng hợp lệ :

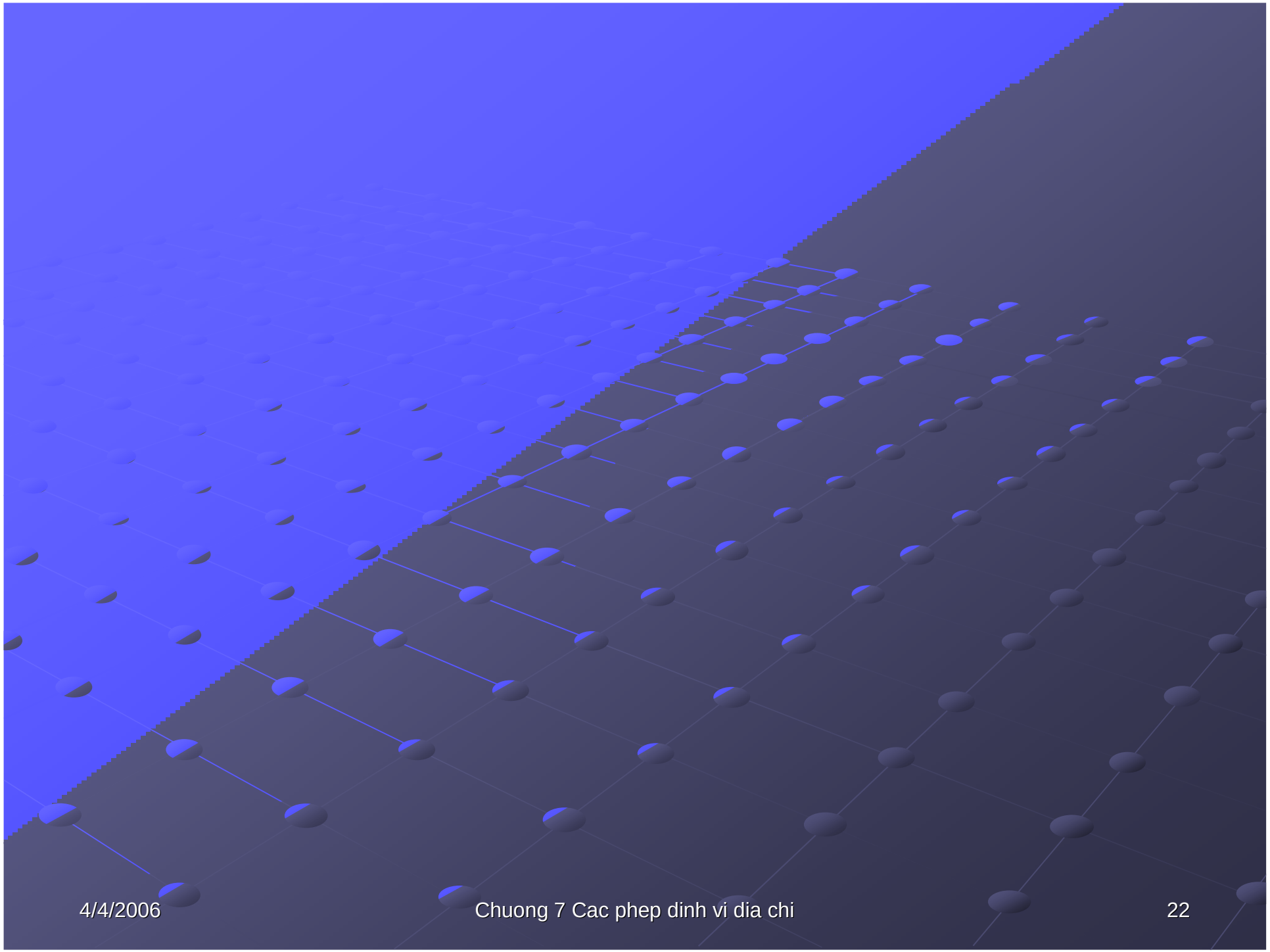
[1000h] [SI], [DI] , [BX] , [BP]

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

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

Địa chỉ hiệu dụng không hợp lệ :

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



Địa chỉ hiệu dụng (tt)

Qui ước

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

Dữ liệu 8 bit bộ nhớ : [địa chỉ]

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

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

8 bit : BYTE PTR [1000H]

Tham khảo 1 byte bộ nhớ ở địa chỉ 1000h

16 bit : WORD PTR [1000H]

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

Ex : Tính tổng 1 array có 5 phần tử

```
MOV BX, OFFSET LIST
MOV AX, 0
MOV AL, [BX]
ADD AL , [BX+1]
ADD AL , [BX+2]
ADD AL , [BX+3]
ADD AL , [BX+4]
MOV  SUM , AX
.....
LIST  DB  10h, 20h, 40h, 2h, 5h
SUM  DW  0
```

Cách thực hiện :

Lấy địa chỉ của List vào BX

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

Khi tính tổng xong, đưa tổng vào biến SUM.

CHẠY CT này bằng DEBUG

Ex : Tính tổng 1 array có 5 phần tử

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

Tập lệnh

Lệnh MOV :
Ý nghĩa : copy giá trị từ toán hạng nguồn → toán hạng đích
Cú pháp : MOV dest , source
Yêu cầu : Dest và source cùng kiểu
Dạng lệnh :

MOV reg , reg
MOV mem , reg
MOV reg, mem
MOV reg16, segreg
MOV segreg, reg16
MOV reg, immed
MOV mem, immed
MOV mem16, segreg
MOV segreg, mem16

Minh 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 70 sang vùng nhớ có địa chỉ bắt đầu là 1000h. Biết chiều mỗi vùng nhớ là 9 bytes và dữ liệu đang khảo sát trong đoạn được chỉ bởi D

Cho vùng nhớ MEM có chiều dài 9 bytes gồm các ký tự 'abcdefghi' trong đoạn chỉ bởi DS.
Viết chương trình đảo ngược vùng nhớ MEM.

Lệnh LEA (Load Effective Address)

Cú pháp : LEA REG | MEM
ý nghĩa : nạp địa chỉ Offset vào thanh ghi để khởi động Reg.
Ex : MOV DX, OFFSET MES
Tương đương với LEA DX, MES

Ex : LEA BX, [1000h] ; BX ⟵ 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.

Cac 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.

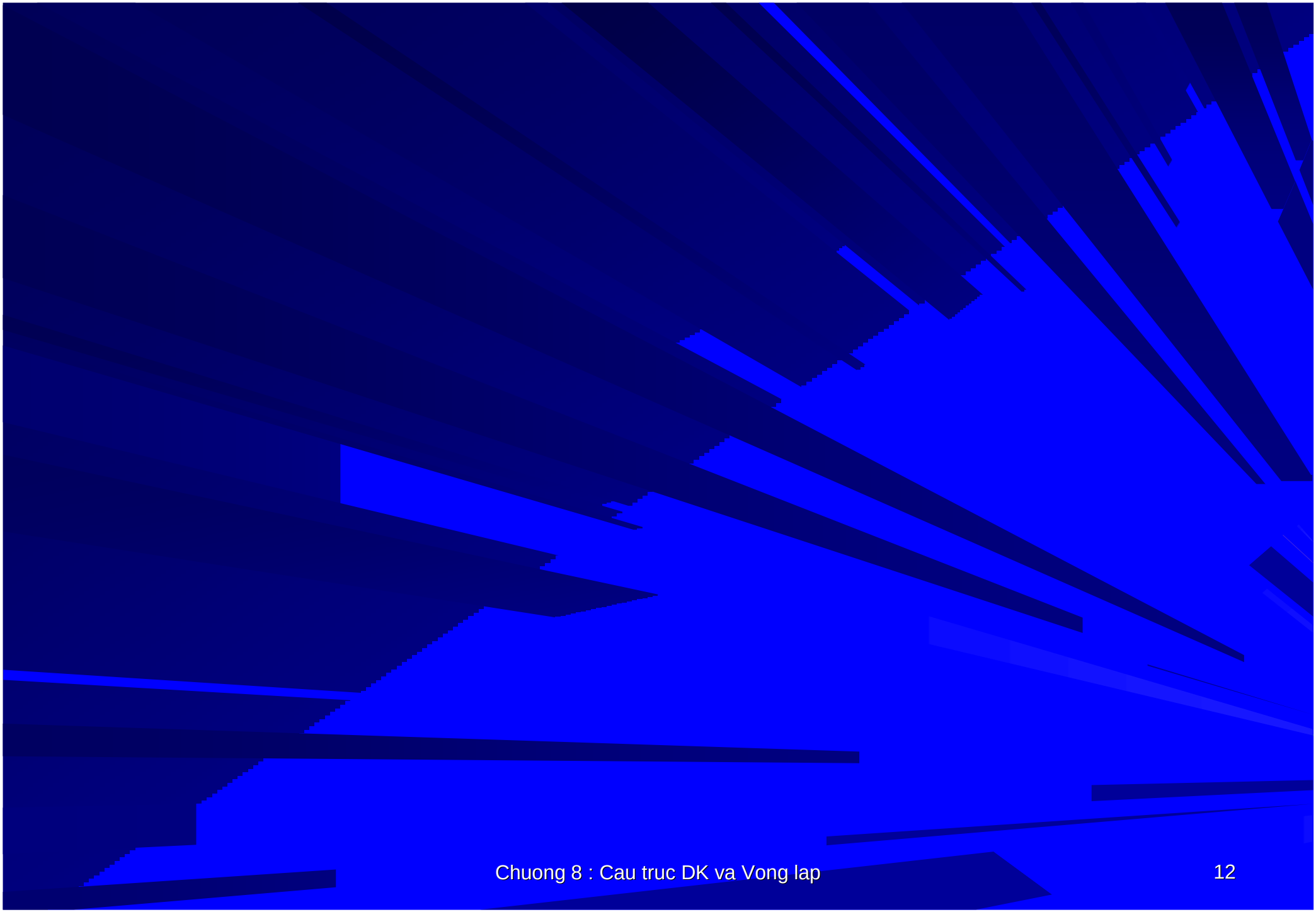
Ex 3:

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 = 11000100b

XOR AL, 11111111b ; AL = 00111011b

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 chẳn
JNP	Nhảy nếu PF =0

CÁC VỊ DỤ MINH HỌA LỆNH NHẢY CÓ ĐK

Ex1 : tìm số lớn hơn trong 2 số
chứa trong thanh ghi AX và BX .
Kết quả để trong DX

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

CÁC VÍ DỤ MINH HỌA LỆNH NHẢY CÓ ĐK

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

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

; giả sử AL nhỏ nhất

; nếu SMALL <= BL thì

Nhảy đến L1

; nếu SMALL <= CL thì

; Nhảy đến L2

; CL là số nhỏ nhất

Các lệnh dịch và quay bit

SHL (Shift Left) : dịch các bit của toán hạng 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.

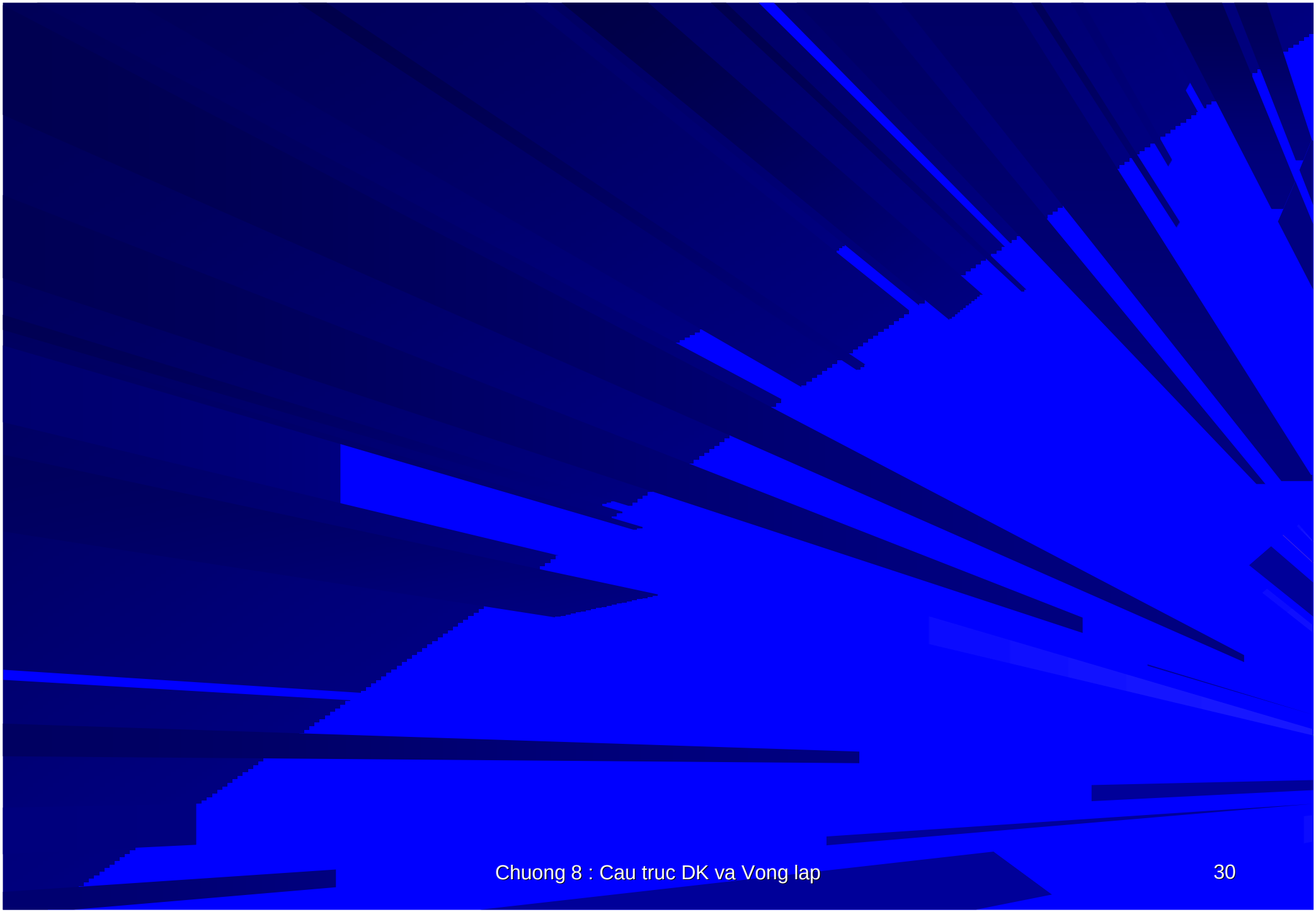
MT thực hiện phép chia bằng dịch phải

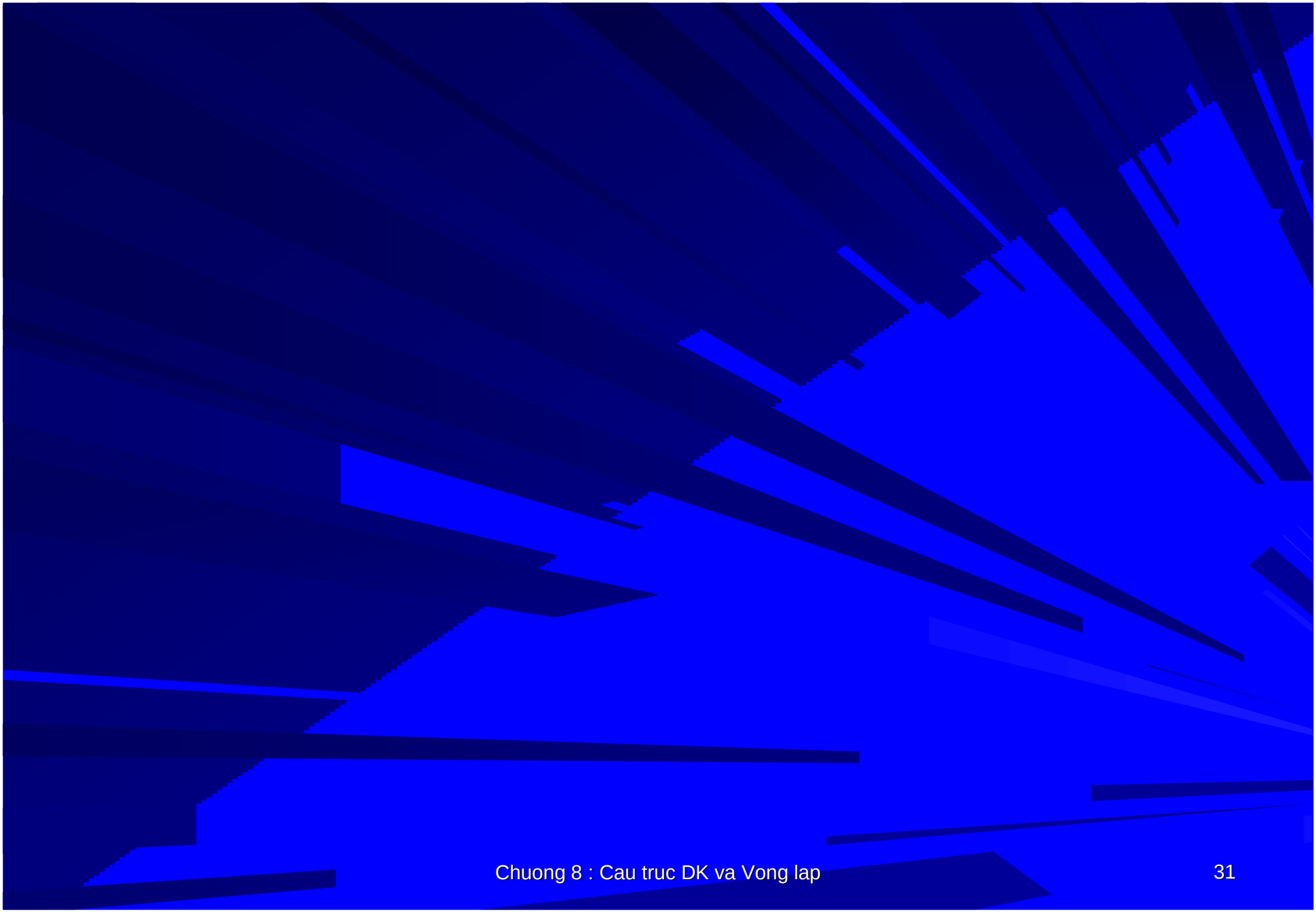
lệnh dịch phải SHR

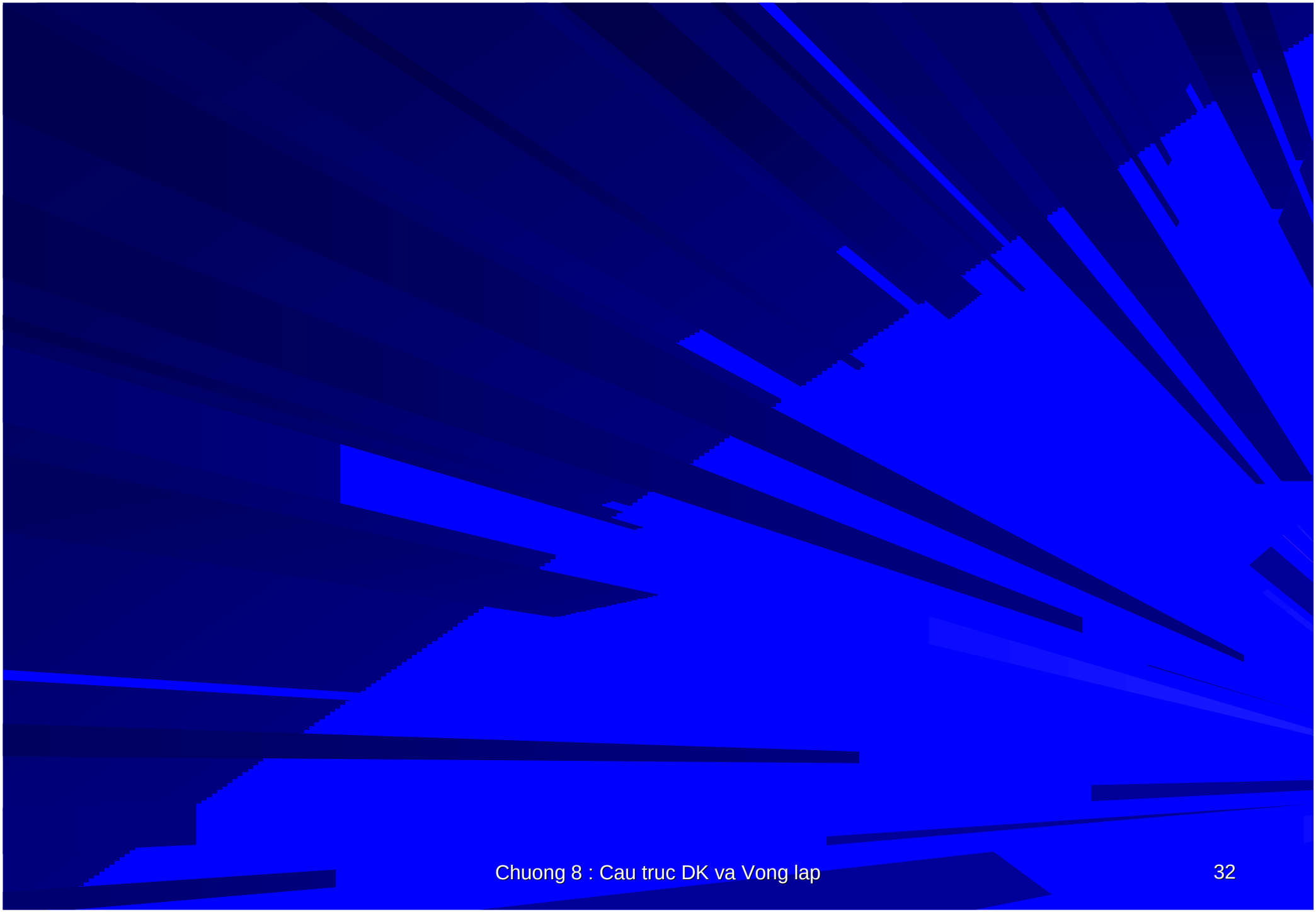
Ex : shr 0100b, 1 ; 0010b = 2

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 : .....
```

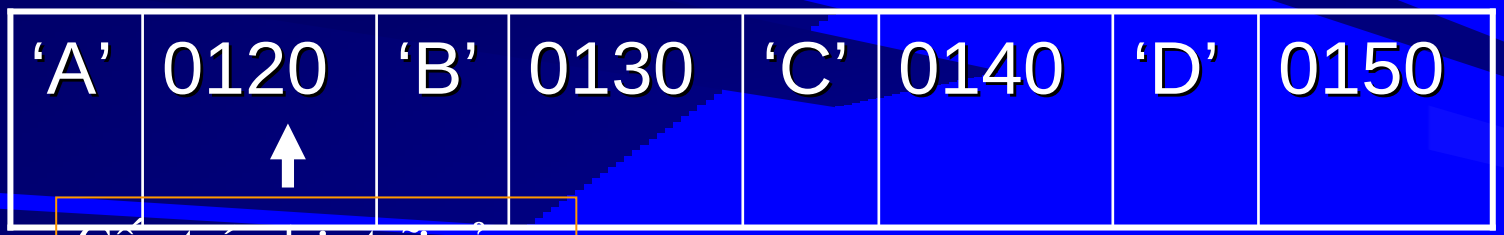
LooKup Table

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

LooKup Table

Case :

```
MOV AL, INPUT
MOV BX, OFFSET CASE_TABLE
MOV CX, 4 ; lặp 4 lần số entry của table
TEST :
  CMP  AL, [BX]  ; kiểm tra Input
 JNE  TESTAGAIN ; không thỏa kiểm tra tiếp
CALL WORD  PTR [BX+1]  ; gọi thủ tục tương ứng
JMP  ENDCASE
TESTAGAIN : ADD BX , 3 ; sang entry sau của CaseTable
LOOP TEST
ENDCASE : ..........
```

Chương trình con

Cấu trúc CTC :

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

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

CTC được gọi bằng lệnh CALL <TenCTC>.

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

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

Kỹ thuật lập trình

- Hãy tổ chức chương trình ➔ các chương trình con ➔ đơn giản 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 chuổi ký tự.

In chuổi đã nhập theo thứ tự ngược.

Ex : nhập ký tự : abcdef

5 ký tự kế tiếp : fedcba

MACRO

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

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

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

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

```
@NewLine Macro
        MOV DL,13
        MOV AH,2
        INT 21H
        MOV DL,10
        MOV AH,2
        INT 21H
ENDM
```

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

```
@NewLine
```

MACRO (tt)

- Khi hợp dịch nội dung nhóm lệnh này mà ta đã gán cho macro sẽ được thay thế vào những nơi có tên macro trước khi CT được hợp dịch thành file 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.

GỌI 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 GOI..

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 toán USCLN như sau :

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

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

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

Bài 3 :Viết macro 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

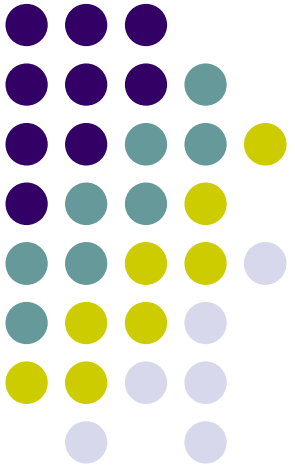
Bài 5 : Viết Macro in số hecxa chứa trong BL ra dạng binary

;Input: BL chứa số cần in

;Output: Nothing

Chương 9
STACK &
CHƯƠNG TRÌNH CON

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



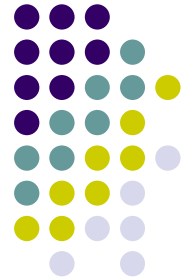
GiỚI THIỆU STACK

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

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

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

LẬP TRÌNH VỚI STACK



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

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

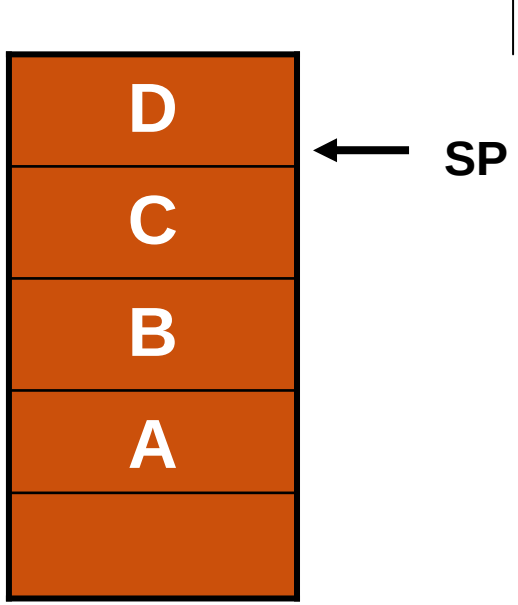
- **Khi thêm một phần tử vào STACK ta thêm từ đỉnh, khi lấy một phần tử ra khỏi STACK ta cũng lấy ra từ đỉnh → địa chỉ của ô nhớ đỉnh STCAK luôn luôn bị thay đổi.**

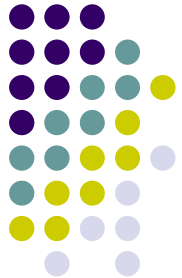
SS dùng để lưu địa chỉ segemnt của đoạn bộ nhớ dùng làm STACK
SP để lưu địa chỉ của ô nhớ đỉnh STACK (trỏ tới đỉnh STACK)

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
```

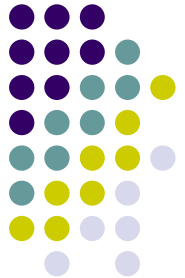




Để lưu 1 phần tử vào Stack ta dùng lệnh PUSH
Để lấy 1 phần tử ra từ Stack ta dùng lệnh POP

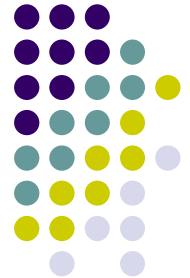
PUSH nguồn : đưa nguồn vào đỉnh STACK
PUSHF : cất nội dung thanh ghi cờ vào STACK

- **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

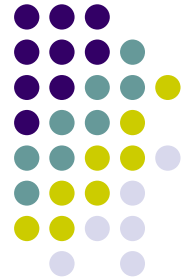


- Khắc phục các hạn chế của lệnh MOV
  Ex : MOV CS,DS ; sai
  PUSH DS
  POP CS ; đúng

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

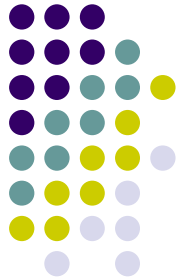
- **Lưu tạm thời giá trị thanh ghi hay biến.**

THÍ DỤ 2



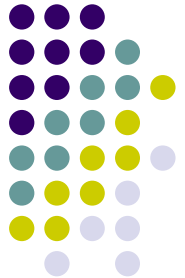
- Nhập vào 1 chuỗi, in chuỗi đảo ngược
  Ex : nhập : Cong nghe thong tin
  xuất : int gnoht ehgn gnoC

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

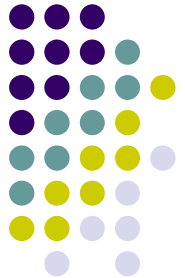


- **CTC là 1 nhóm các lệnh được gộp lại dưới 1 cái tên mà ta có thể gọi từ nhiều nơi khác nhau trong chương trình thay vì phải viết lại các nhóm lệnh này tại nơi cần đến chúng.**

Lợi ích

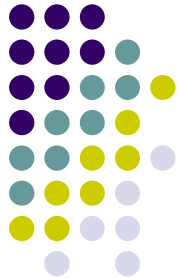
CTC làm cho cấu trúc logic của của CT dễ kiểm soát hơn, dễ tìm sai sót hơn và có thể tái sử dụng mã ➔ tiết kiệm được công sức và thời gian lập trình.

CẤU TRÚC CỦA CTCON



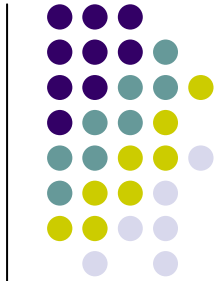
```
TÊNCTC  PROC  [NEAR|FAR]
    CÁC LỆNH CỦA CTC
   RET
TÊNCTC  ENDP
```

MINH HỌA



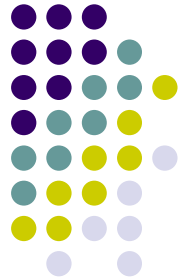
- Viết chương trình nhập 1 số n (n nguyên dương và <9). Tính giai thừa của n và xuất ra màn hình dưới dạng số hex (giới hạn kết quả 16 bit).
- Viết chương trình tìm số hoàn thiện (giới hạn 2 chữ số) và in nó ra màn hình.

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

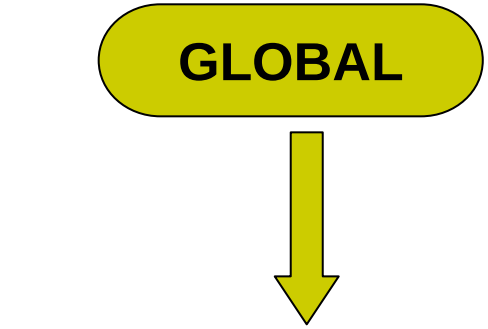
TÊN ĐI TRƯỚC NHÃN

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

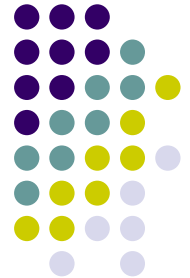
- BYTE
- DATAPTR
- WORD
- NEAR
- DWORD
- FAR
- PROC



THAY THẾ PUBLIC VÀ EXTRN.

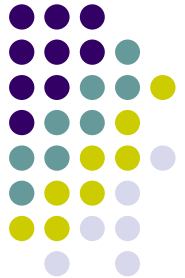
Viết chương trình nằm trên 2 file (2 module) với sự phân công như sau :
Module của chương trình chính (Main.ASM) có nhiệm vụ xác định Offset của 2 chuỗi ký tự và gọi CTC nối 2 chuỗi này và cho hiện kết quả ra màn hình.
Module CTC (Sub.ASM) làm nhiệm vụ nối 2 chuỗi và đưa vào bộ nhớ.

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



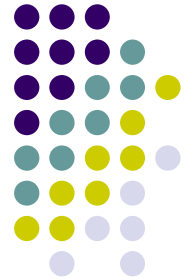
```
PrintNum10    PROC
; số nguyên N nằm trong AX
              PUSH   BX     CX    DX
              MOV    CX, 0          ; so lan push (so ky tu)
laysodu:
          XOR     DX, DX       ;  cho DX = 0 trước khi chia
                  MOV    BX, 10
                  DIV     BX             ; số dư trong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

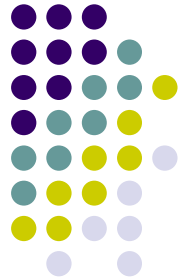
Có 3 cách truyền tham số

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

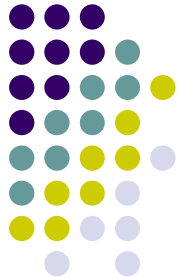
- KHAI BÁO BiẾN TOÀN CỤC.
- DÙNG NÓ ĐỂ CHUYỂN CÁC GIÁ TRỊ GiỮA CT GỌI VÀ CT ĐƯỢC GỌI.

CÁCH NÀY THƯỜNG ĐƯỢC DÙNG :

TRONG 1 CT ViẾT THUẦN TÚY BẰNG ASM

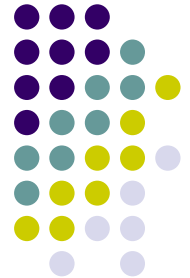
ViẾT HỖN HỢP GiỮA ASM VÀ 1 NGÔN NGỮ CẤP CAO

TRUYỀN THAM SỐ QUA STACK



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

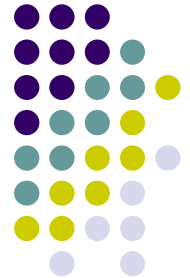
CHUYỂN GIÁ TRỊ TỪ CTCON LÊN CT CHÍNH.



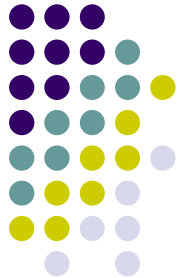
- **CŨNG THÔNG QUA CÁC THANH GHI,BỘ NHỚ VÀ STACK.**

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.



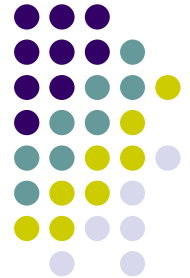
- **NẾU GIÁ TRỊ QUAY LẠI LÀ 32 BIT (CHO KHAI BÁO LONG, CON TRỎ XA) THÌ GIÁ TRỊ ĐÓ PHẢI ĐƯỢC ĐẶT TRONG THANH GHI DX,AX CỦA HÀM TRƯỚC KHI QUAY VỀ CT CHÍNH.**



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

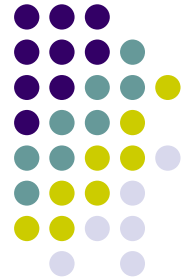
- **NEAR : lấy địa chỉ OFFSET (16BIT) trong STACK và gán vào thanh ghi IP.**
- **FAR : lấy địa chỉ OFFSET và SEGMENT trong STACK nạp vào thanh ghi CS:IP.**

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



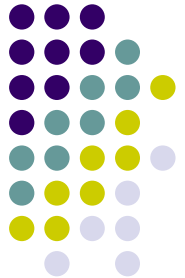
NHẬP VÀO 1 SỐ HỆ HEX. IN RA SỐ ĐÃ NHẬP VỚI YÊU CẦU SAU :

ViẾT CTCON NHẬP SỐ

ViẾT CTCON XUẤT SỐ

CTCHÍNH GỌI 2 CTCON TRÊN.

LUYỆN TẬP LẬP TRÌNH C10

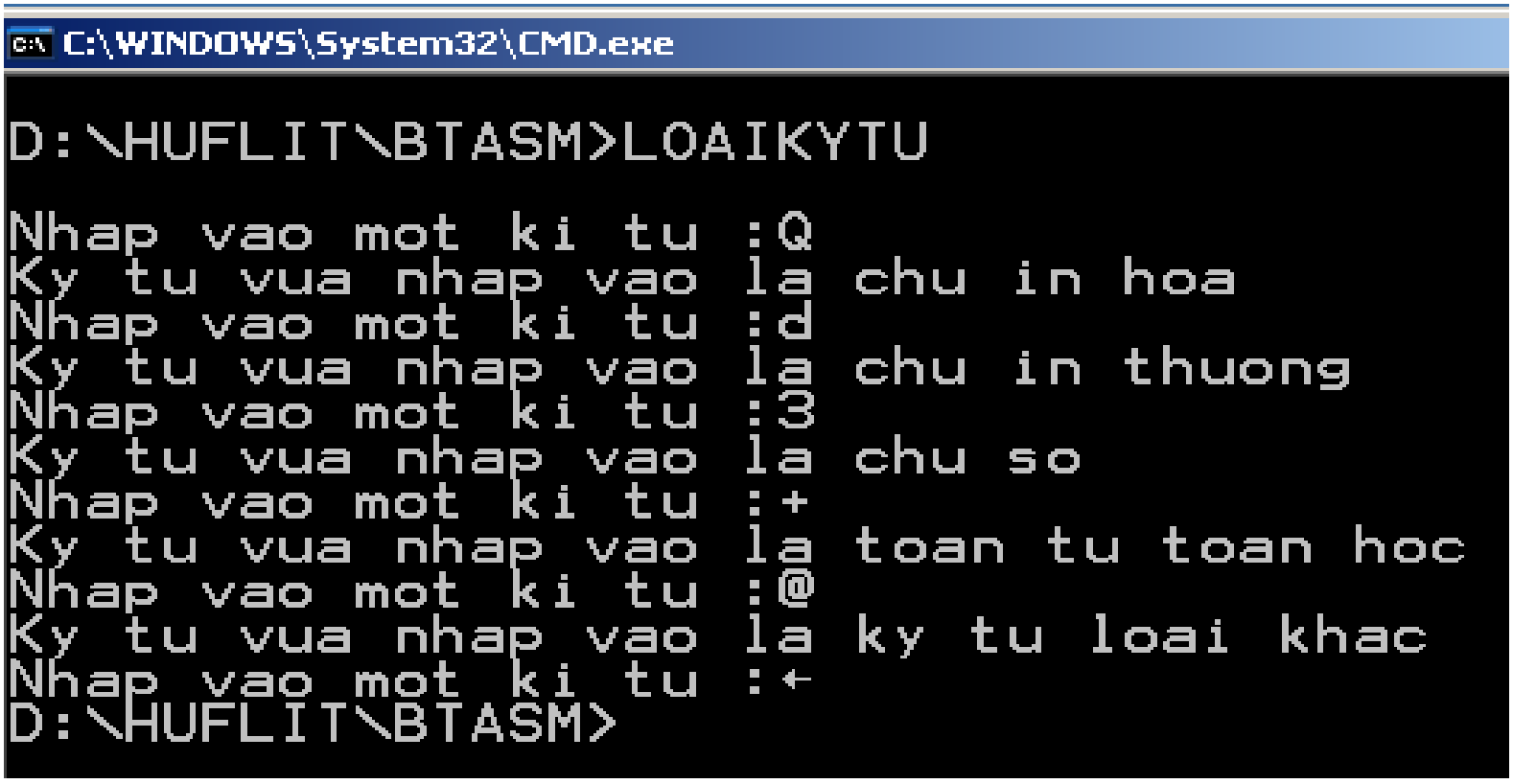


Bài 1 : Viết chương trình nhập 1 số nguyên n (n<9). Tính giai thừa của n và xuất kết quả ra màn hình dưới dạng số Hex (giới hạn 16 bits).

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.

LẬP TRÌNH XỬ LÝ MÀN HÌNH & BÀN PHÍM

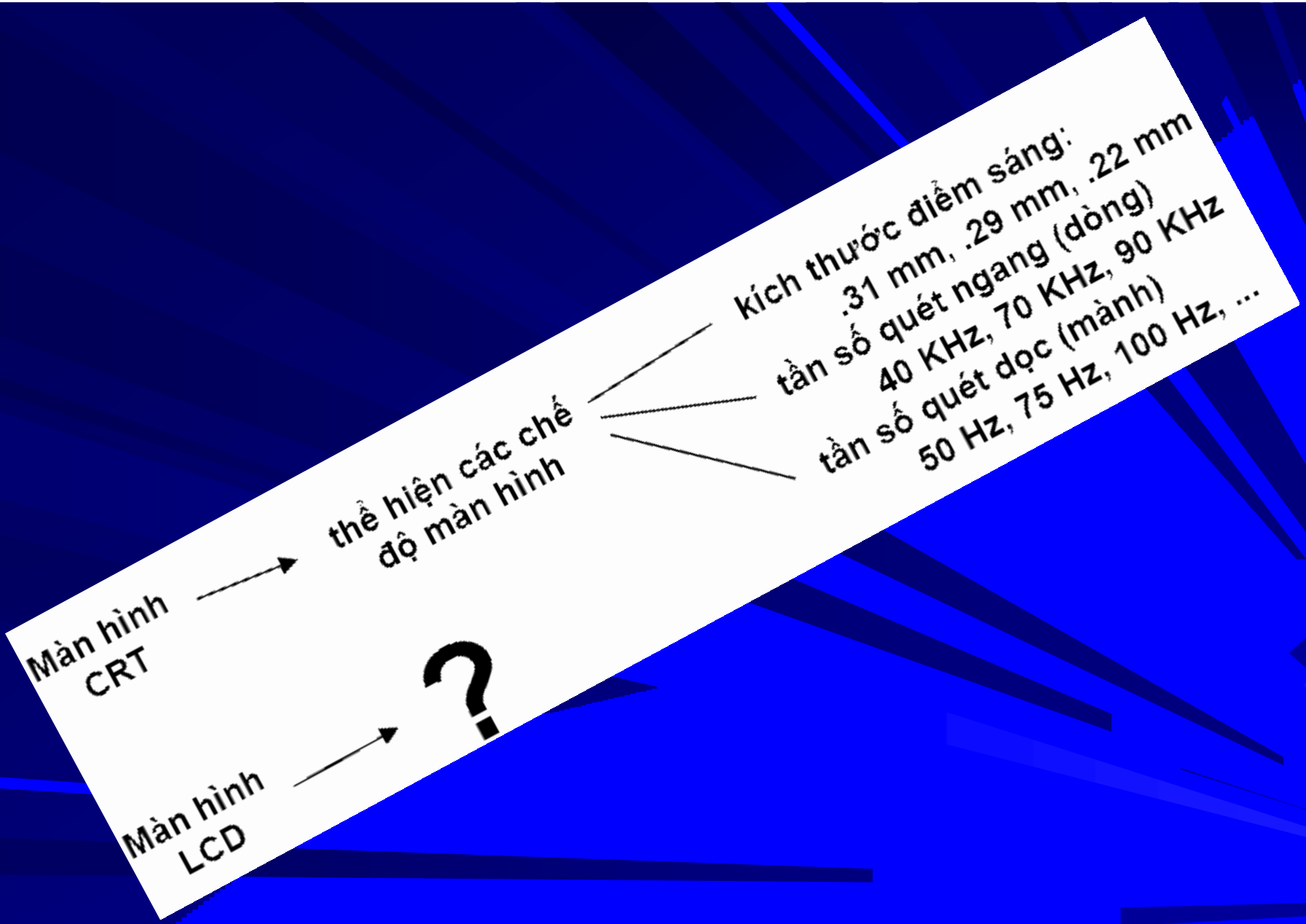
- Giới thiệu màn hình & việc quản lý màn hình
- Hiểu được tổ chức của màn hình.
- So sánh chức năng điều khiển màn hình của INT 10h của ROM BIOS với chức năng của INT 21h.
- Biết cách lập trình quản lý màn hình trong ASM.
- Biết cách lập trình xử lý phím và 1 số ứng dụng của nó.

MÀN HÌNH

ĐẶC TRƯNG CỦA MÀN HÌNH

ĐỘ PHÂN GIẢI

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


Màn hình CRT
thể hiện các chế độ màn hình
kích thước điểm sáng: .31 mm, .29 mm, .22 mm
tần số quét ngang (dòng) 40 KHz, 70 KHz, 90 KHz
tần số quét dọc (mành) 50 Hz, 75 Hz, 100 Hz, ...
Màn hình LCD
?

CARD MÀN HÌNH

Cung cấp các chế độ MH

Độ phân giải

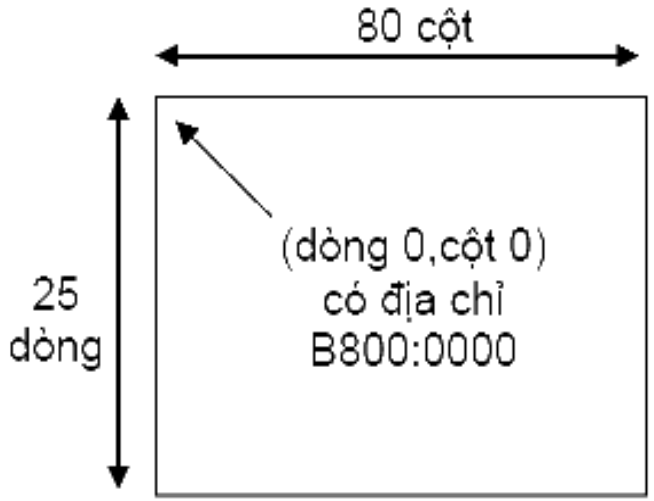
Số điểm ngang x số điểm dọc x số màu (số bit màu)

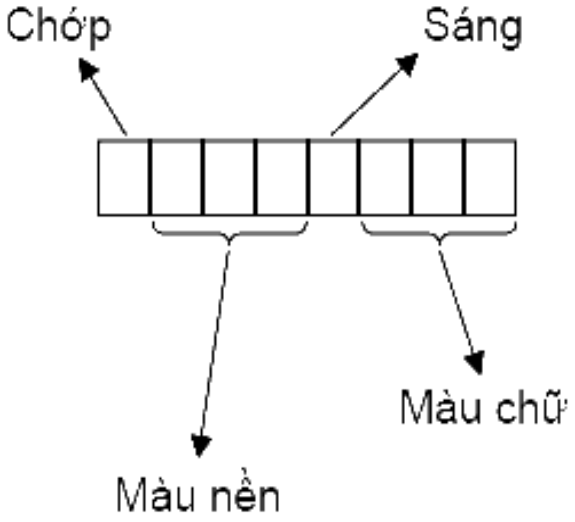
CHẾ ĐỘ ĐỒ HỌA

800x600x16 bits = 960 000 bytes ➔ 1Mb
1024x 768x32 bits ➔3.145.728 bytes ➔ 4Mb

RAM MÀN HÌNH

Dung lượng



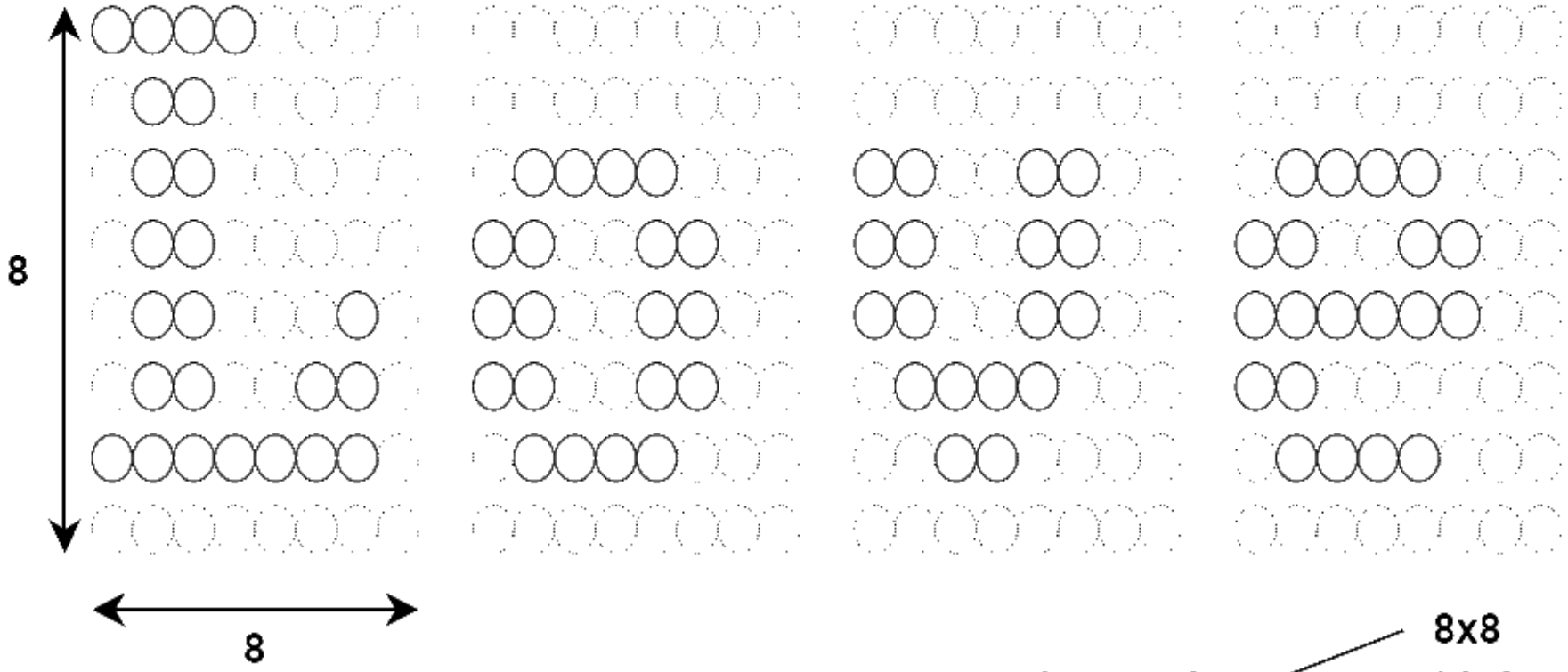


- Mỗi ký tự lưu bằng 2 byte.
- Byte địa chỉ thấp chứa mã ASCII.
- Byte địa chỉ cao chứa mã màu

Địa chỉ (i,j) = B800:0000+ (i*160+j*2)

Ma trận ký tự trên màn hình

Thiết bị ngoại vi



Kích thước
- 8x8
- 14x8
- 16x8

QUẢN LÝ MÀN HÌNH

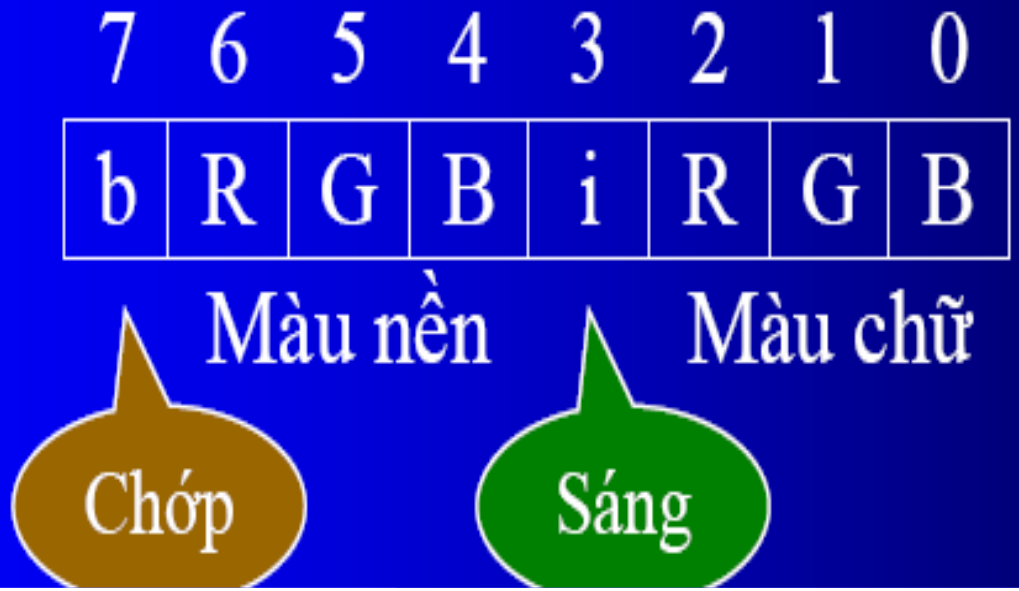
- Màn hình được điều khiển hiệu quả nhờ các chức năng của **INT 10H** trong Rom Bios. Các chức năng này quản lý màn hình tốt hơn các chức năng của INT 21h của Dos.

- Bên cạnh 1 số chức năng do INT 21h của Dos cung cấp, 1 số tác vụ được thực hiện trên màn hình nhờ các chức năng trong INT 10h như xoá màn hình, định vị con trỏ, thiết lập màn hình ...

- IBM PC hỗ trợ 3 loại màn hình cơ bản có tên tùy thuộc vào loại Card màn hình cắm trên Bus mở rộng trên Mianboard như : **Monochrome** chỉ hiển thị text đơn sắc; **CGA** (Color Graphic Adaptor) cho phép hiển thị text và đồ họa; **EGA** (Enhanced Graphics Adaptor) hiển thị text và đồ họa với độ phân giải cao hơn. Ngoài ra còn có card **VGA** (Video Graphics Array), **SVGA .. .**

THUỘC TÍNH MÀN HÌNH

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



THUỘC TÍNH MÀN HÌNH

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

- Thuộc tính của ký tự được lưu trong 1 byte gọi là byte thuộc tính.

THUỘC TÍNH MÀN HÌNH

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

BLINK = 10000000B

RED = 100B

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

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

VÙNG HIỂN THỊ MÀN HÌNH

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

THUỘC TÍNH MÀN HÌNH

- **Các thuộc tính chuẩn của màn hình Monochrome :**

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

THUỘC TÍNH MÀN HÌNH (tt)

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

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

Bit 6,5,4 : màu nền

Bit 2,1,0 : màu chữ

Bit 3 : độ sáng

BẢNG MÀU (COLOR PALETTE)

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

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

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

CÁC MODE MÀN HÌNH

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

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

CÁC MODE MÀN HÌNH

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

Màn hình là hình ảnh của Video Ram.

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

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

25X80X2 = 4000BYTES RAM VIDEO

VÙNG NHỚ NÀY NẰM TRÊN CARD MH

CÁC MODE MÀN HÌNH

Các Video mode thông dụng :

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

TRANG MÀN HÌNH (VIDEO PAGE)

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

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

TRANG MÀN HÌNH (VIDEO PAGE)

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

THÍ DỤ VỀ TRANG MH

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

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

CHẾ ĐỘ ĐỒ HỌA

```
MOV     AH, 0
MOV     AL, CheDo
INT     10h
```

CheDo=

4	320 x 200	4 màu CGA
6	640 x 200	2 màu CGA
D	320 x 200	16 màu EGA
E	640 x 200	16 màu EGA
10	640 x 350	16 màu EGA
11	640 x 480	2 màu VGA
12	640 x 480	16 màu VGA
13	640 x 480	256 màu VGA

Truy xuất thiết bị xuất chuẩn (màn hình)

. 1. Chọn chế độ hiển thị :

Chức năng AH = 0, ngắt 10H

Vào : AH = 0, AL = kiểu

Ví dụ : thiết lập chế độ văn bản màu

```
XOR     AH, AH
MOV     AL, 3     ; chế độ văn bản màu 80 x 25
INT     10H
```

THAY ĐỔI SIZE CON TRỎ MÀN HÌNH

Chức năng AH = 1, ngắt 10H

Vào : AH = 1,

CH = dòng quét đầu, CL = dòng quét cuối

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

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

DỊCH CHUYỂN CON TRỎ

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

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

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

LẤY VỊ TRÍ KÍCH THUỚC CON TRỎ HiỆN HÀNH

Chức năng AH = 3, ngắt 10H

Vào : AH = 3, BH = số hiệu trang

Ra : DH = dòng, DL = cột,

CH = dòng quét đầu, CL = dòng quét cuối

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

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

CUỘN MÀN HÌNH

Chức năng AH = 6, ngắt 10H

Vào : AH = 6,

AL = số dòng cuốn (= 0 là toàn màn hình)

Ra : BH = thuộc tính các dòng trống ,

CH, CL = dòng, cột góc trái trên

DH, DL = dòng, cột góc phải dưới của cửa sổ

Ví dụ : Xoá đen màn hình 80 x 25

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

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

. Lập chế độ hiển thi màu 80 x 25

. Xoá cửa sổ tại góc trái trên : cột 26 dòng 8 và góc phải dưới tại cột 52 dòng 16 thành màu đỏ. .

Sau đó hiển thị kí tự A màu cam tại vị trí con trỏ.

CÁC HÀM XỬ LÝ MÀN HÌNH

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

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

CÁC HÀM XỬ LÝ MÀN HÌNH

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

Chức năng (để trong AH)	*nhiệm vụ*
0Ah	*Ghi ký tự bỏ qua thuộc tính ký tự vào vị trí con trỏ hiện hành.*
0Bh	*Chọn palette màu*
0Ch	*Ghi 1 điểm graphic trong graphics mode.*
0Dh	*Đọc giá trị màu của 1 pixel có vị trí đã biết.*
0Eh	*Ghi ký tự ra màn hình và cập nhật con trỏ sang phải 1 vtrí.*
0Fh	*Lấy mode màn hình hiện hành để xem đang ở chế độ text hay graphics.*

HÀM 0H INT 10H

- Thiết lập video mode.
  AH = 0
  AL = mode.
  Nếu bit cao của AL =0 sẽ tự động xoá màn hình.
  Nếu bit cao của AL = 1 không xoá màn hình.

- Ex : thiết lập 80x25 color text mode
  MOV AH, 0
  MOV AL, 3 ; mode 3 , có xoá màn hình
  INT 10h

LƯU Ý : Không muốn xoá màn hình thì AL = 83H

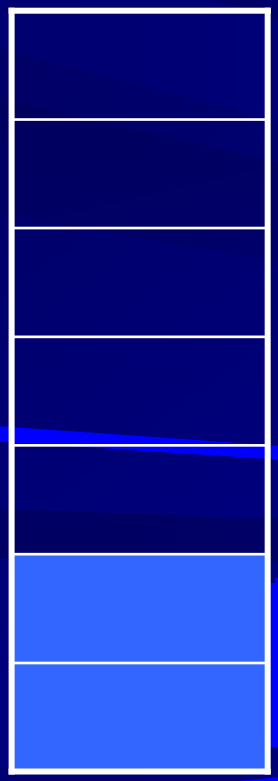
HÀM 0H INT 10H

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

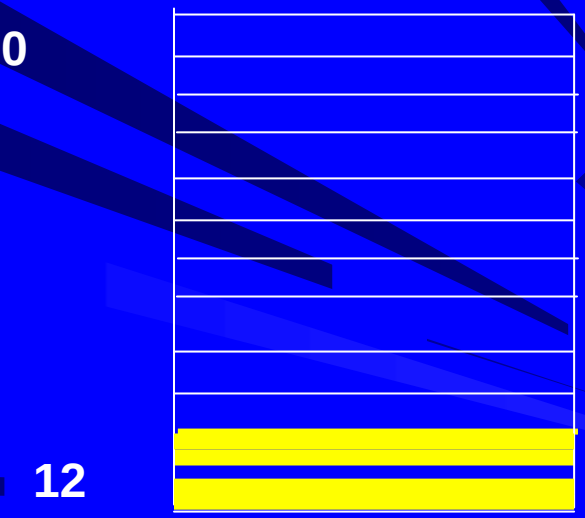
```
MOV AH, 0 ; set video mode
MOV AL, 6 ; 640x200 color graphics mode
INT 10h
MOV AH, 1 ; đợi gỏ 1 phím
INT 21H
MOV AH, 0 ; set video mode
MOV AL, 3 ; color text mode
INT 10H
```

HÀM 01
INT 10H

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



CGA/EGA



MONOCHROME

Ex: Minh họa

Thiết lập con trỏ có hình khối đặc.

```
MOV AH , 1
MOV CH, 0
MOV CL,0CH
INT 10H
```

Để thay đổi dạng con trỏ :
AH = 1
CH = TOP (dòng đầu)
CL = BOTTOM (dòng cuối)

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

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

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

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

HÀM 02H
INT 10H

Thiết lập vị trí hiện hành

SET CURSOR POSITION

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

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

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

HÀM 03H
INT 10H

Lấy vị trí cursor hiện hành

GET CURSOR POSITION

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

HÀM 03H
INT 10H

Lấy vị trí cursor hiện hành

GET CURSOR POSITION

Ex : lấy vị trí của cursor lưu vào biến.

Thường dùng trong các tác vụ menu.

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

HÀM 05H
INT 10H

THIẾT LẬP TRANG MÀN HÌNH

SET VIDEO PAGE

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

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

```
DOSSEG
.MODEL SMALL
.STACK 100H
.CODE
MAIN PROC
MOV AX, @DATA
MOV DS,AX
MOV DX, OFFSET  TRANG0
INT 21H
MOV AH , 1
INT 21H
SANG_TRANG_1 :
MOV AH, 5
MOV AL, 1
```

HÀM 05H
INT 10H

THIẾT LẬP TRANG MÀN HÌNH

SET VIDEO PAGE

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

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

HÀM 06H, 07H
INT 10H

CUỘN MÀN HÌNH

SCROLL WINDOW UP AND DOWN

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

CÁC DÒNG DỮ LIỆU BỊ CUỐN SẼ ĐƯỢC THAY THẾ BẰNG CÁC DÒNG TRỐNG.

TA ĐỊNH NGHĨA WINDOWS NHỜ HỆ TỌA ĐỘ HÀNG CỘT VỚI GỐC TỌA ĐỘ LÀ GÓC TRÊN TRÁI CỦA MÀN HÌNH.

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

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

TA CÓ THỂ CUỘN 1 VÀI DÒNG HOẶC CẢ WINDOWS.

TOÀN BỘ WINDOWS BỊ CUỘN → MÀN HÌNH BỊ XOÁ.

HÀM 06H, 07H INT 10H

CUỘN MÀN HÌNH

SCROLL WINDOW UP AND DOWN

CÁC THÔNG SỐ :

CUỘN LÊN AH =6 ; CUỘN XUỐNG AH = 7

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

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

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

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

HÀM 06H, 07H
INT 10H

CUỘN MÀN HÌNH

SCROLL WINDOW UP AND DOWN

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

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

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

HÀM 06H, 07H
INT 10H

CUỘN MÀN HÌNH

SCROLL WINDOW UP AND DOWN

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

```
MOV AX, 0702H
MOV CX,0A14H
MOV DX, 0F3CH
MOV BH, 70H
INT 10H
```

HÀM 08H
INT 10H

ĐỌC 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.

READ CHARACTER AND ATTRIBUTE

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

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

```
LOCATE :
MOV AH, 2
MOV BH, 0
MOV DX, 0501H
INT 10H
```

```
GETCHAR :
MOV AH, 8
MOV BH, 0
INT 10H
MOV CHAR, AL
MOV ATTRIB , AH
```

HÀM 09H
INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.

WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 09H INT 10H :

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

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;
BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT
CX = HỆ SỐ LẶP

HÀM 0AH
INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.

WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 0AH INT 10H :

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

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;
BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT
CX = HỆ SỐ LẶP

HÀM 0AH
INT 10H

GHI 1 KÝ TỰ VÀ THUỘC TÍNH KÝ TỰ.

WRITE CHARACTER AND ATTRIBUTE

CHỨC NĂNG 0AH INT 10H :

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

AH =9 ; BH = TRANG VIDEO

AL = KÝ TỰ SẼ XUẤT ;
BL = THUỘC TÍNH CỦA KÝ TỰ SẼ XUẤT
CX = HỆ SỐ LẶP

HÀM 0FH
INT 10H

LẤY VIDEO MODE

GET VIDEO MODE

CHỨC NĂNG 0FH INT 10H : LẤY VIDEO MODE

AH =0F ;
BH = TRANG HIỆN HÀNH

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

```
EX : MOV AH,0FH ; Get Video Mode Function
     INT 10H ; gọi BIOS
     MOVE VIDEO_MODE, AL ; lưu Video Mode vào biến bộ nhớ
     MOV PAGE, BH ; lưu trang hiện hành.
```

LẬP TRÌNH XỬ LÝ PHÍM

1. Đọc phím nhấn :

Chức năng AH = 0, ngắt 16H

Vào : AH = 0

Ra : AL = mã ASCII nếu một phím ASCII được nhấn

= 0 nếu phím điều khiển được nhấn

AH = mã scan của phím nhấn

LẬP TRÌNH XỬ LÝ PHÍM

BÀN PHÍM

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

BÀN PHÍM

INT 21h, AH = 1 (DOS)

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

BÀN PHÍM

INT 21h, AH = 8 (DOS)

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

BÀN PHÍM

INT 16h, AH = 0 (BIOS)

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

BÀN PHÍM

INT 16h, AH = 1 (BIOS)

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

BÀN PHÍM

INT 16h, AH = 2 (BIOS)

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

LẬP TRÌNH XỬ LÝ PHÍM

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

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

Cách đọc trực tiếp

```
XOR AX, AX
MOV ES, AX
MOV AL, ES:[417H]
TEST AL, 01H                ; kiểm tra phím Shift
JNZ  SHIFT_DANGNHAN
TEST AL, 04H                ; kiểm tra phím Ctrl
JNZ  CTRL_DANGNHAN
TEST AL, 08H                ; Kiểm tra phím Alt
JNZ  ALT_DANGNHAN
```

3. Kiểm tra và thiết lập trạng thái các phím Caps/Num/Scroll Lock tương tự nhưng với mã scan khác Scroll = 10H, Num = 20H, Cap = 40H.

4. Đặt lại các trạng thái đèn Caps/Num/Scroll Lock, ta chỉ cần đặt lại giá trị ở địa chỉ 0:0417.
Vd, để bật đèn Caps Lock và đổi trạng thái đèn Num Lock ta sẽ làm như sau:

```
XOR      AX,  AX
MOV      ES,  AX              ; ES  = 0
MOV      AL,  ES:[417H]    ; đọc trạng thái đèn
OR       AL,  40H           ; bật đèn Caps Lock
XOR      AL, 20H            ; đảo đèn Nums Lock
MOV     ES:[417H],  AL    ;
MOV     AH ,  2H
INT      16H
```

BÀI TẬP LẬP TRÌNH

BÀI 1 : VIẾT ĐOẠN CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

- **CUỘN WINDOW TỪ HÀNG 5, CỘT 10 TỚI HÀNG 20 CỘT 70 VỚI THUỘC TÍNH MÀN HÌNH ĐẢO.**
- **ĐỊNH VỊ CURSOR TẠI HÀNG 10, CỘT 20**
- **HIỂN THỊ DÒNG TEXT " DAY LA 1 DONG TEXT TRONG WINDOW "**
- **SAU KHI XUẤT TEXT ĐỢI NHẤN 1 PHÍM.**
- **CUỘN WINDOW TỪ HÀNG 5, CỘT 15 TỚI HÀNG 18 CỘT 68 VỚI THUỘC TÍNH THƯỜNG.**
- **XUẤT KÝ TỰ A VỚI THUỘC TÍNH NHẤP NHÁY TẠI GIỮA WINDOW.**
- **ĐỢI GỎ 1 PHÍM, XÓA TOÀN BỘ MÀN HÌNH..**

BÀI TẬP LẬP TRÌNH

BÀI 2 : VIẾT CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

- **XUẤT CHUỔI "GO VAO 1 KY TU THUONG : ' .**
- **KHI USER GỎ 1 KÝ TỰ (KHI GỎ KHÔNG HIỂN THI KÝ TỰ GỎ RA MÀN HÌNH) ,ĐỔI KÝ TỰ NÀY THÀNH CHỮ HOA RỒI XUẤT RA MÀN HÌNH.GIẢ SỬ CHỈ NHẬP CÁC KÝ TỰ HỢP LỆ.**
- **KHI GỎ KÝ TỰ MỞ RỘNG SẼ THOÁT VỀ DOS, NHƯNG CÓ LẼ BẠN CÒN NHÌN THẤY 1 KÝ TỰ XUẤT THÊM TRÊN MÀN HÌNH. GIẢI THÍCH.**

1. Viết chương trình để :
a. Xoá màn hình, tạo kích thước to nhất cho con trỏ và di chuyển nó đến góc trái trên
b. Nếu nhấn phím Home : chuyển con trỏ đến góc trái trên, End : chuyển đến góc trái dưới, Page Dn : chuyển con trỏ đến góc phải dưới, Esc : kết thúc chương trình.
2. Dịch chuyển con trỏ đến góc trái trên màn hình nếu phím F1 được nhấn, góc trái dưới nếu phím F2 được nhấn. Chương trình sẽ bỏ qua các kí tự thông thường.
3. Viết chương trình soạn thảo văn bản như sau :
a. Xoá màn hình, định vị con trỏ tại đầu dòng 12
b. Để người sử dụng đánh vào các kí tự. Con trỏ dịch chuyển đi sau khi hiển thị kí tự nếu nó không ở tại lề phải của màn hình
c. Phím mũi tên trái , phải, lên , xuống dịch con trỏ tương ứng
d. Phím Insert : chèn kí tự, Delete : Xoá một kí tự , Esc : kết thúc chương trình.

BÀI TẬP LẬP TRÌNH

BÀI 3 : VIẾT CHƯƠNG TRÌNH LÀM CÁC VIỆC SAU :

CHO PHÉP VẼ ĐƠN GIẢN NHỜ CÁC PHÍM MỦI TÊN TRÊN BÀN PHÍM ĐỂ DI CHUYỂN THEO HƯỚNG MONG MUỐN.

PHẢI BẢO ĐẢM XUẤT CÁC KÝ TỰ GÓC THÍCH HỢP.

BIẾT RẰNG MÃ ASCII CỦA 1 SỐ KÝ TỰ : xem bảng mã ASCII

MÃ SCAN CODE CỦA CÁC PHÍM MŨI TÊN :

TRÁI 4BH PHẢI 4DH LÊN 48H XUỐNG 50H

LẬP TRÌNH XỬ LÝ ĐĨA&FILE

- CƠ BẢN VỀ LƯU TRỮ TRÊN ĐĨA TỪ.
- MỘT ỨNG DỤNG HIỂN THỊ SECTOR
- MỘT ỨNG DỤNG HIỂN THỊ CLUSTER.
- CÁC CHỨC NĂNG VỀ FILE Ở MỨC HỆ THỐNG.
- QUẢN LÝ ĐĨA VÀ THƯ MỤC.
- TRUY XUẤT ĐĨA VỚI INT 13H CỦA ROMBIOS
- BÀI TẬP
- GiỚI THIỆU FILE VÀ LẬP TRÌNH XỬ LÝ FILE

CƠ BẢN VỀ LƯU TRỮ TRÊN ĐĨA TỪ

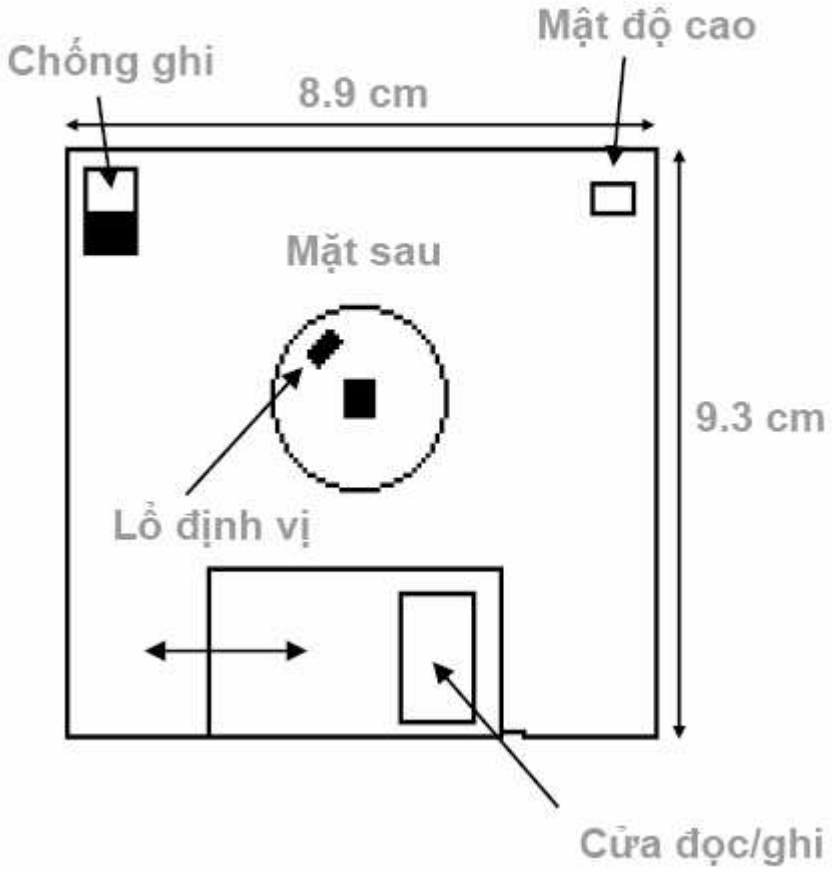
Ngôn ngữ ASM vượt trội hơn các ngôn ngữ khác về khả năng xử lý đĩa.

Ta xem xét việc lưu trữ thông tin trên đĩa theo 2 mức độ : mức phần cứng/BIOS và mức phần mềm/DOS.

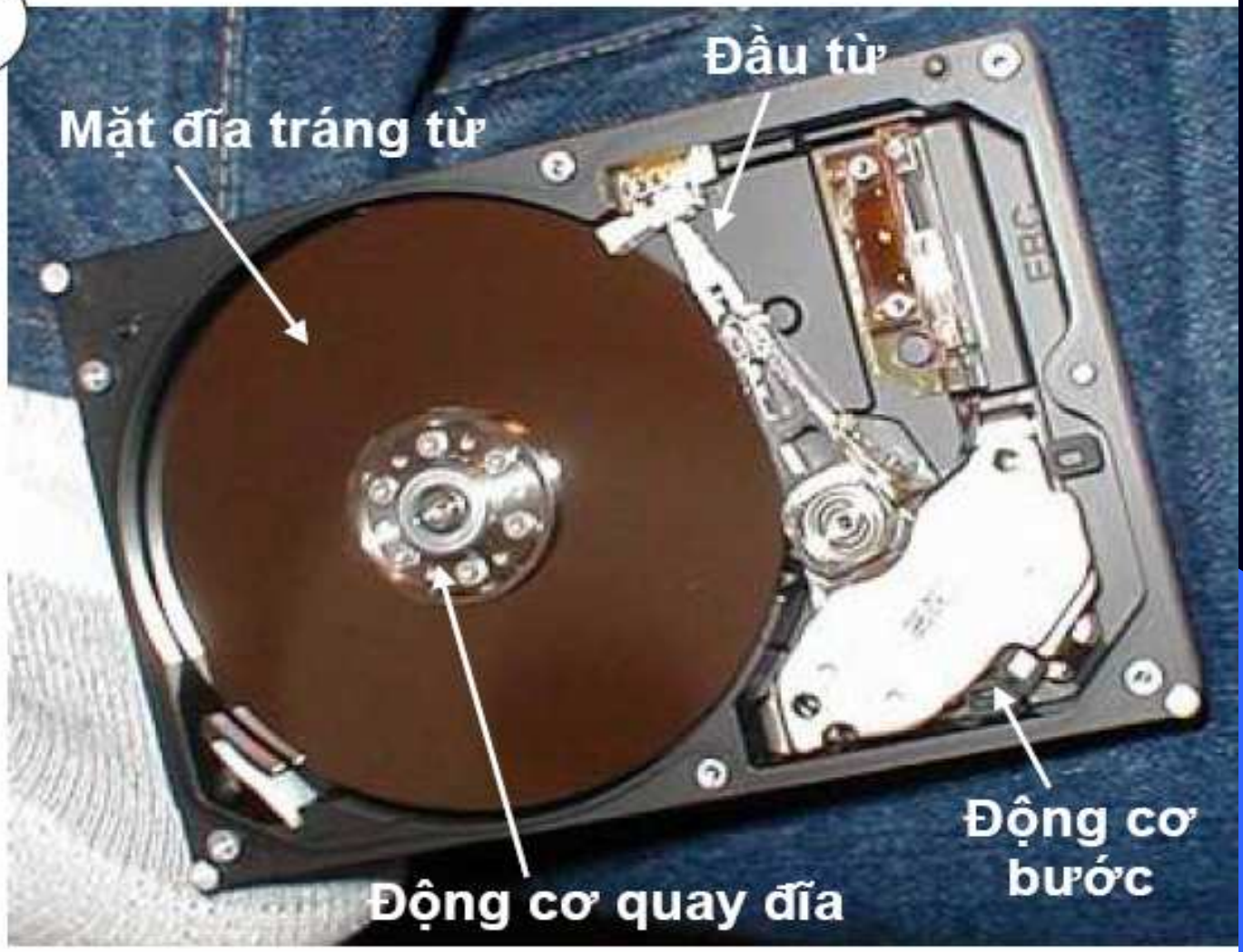
- mức phần cứng :lưu trữ thông tin liên quan đến cách dữ liệu được lưu trữ 1 cách vật lý như thế nào trên đĩa từ?

- mức phần mềm : việc lưu trữ được quản lý bởi tiện ích quản lý File của HĐH DOS.

Đĩa mềm 3.5”



Đĩa cứng

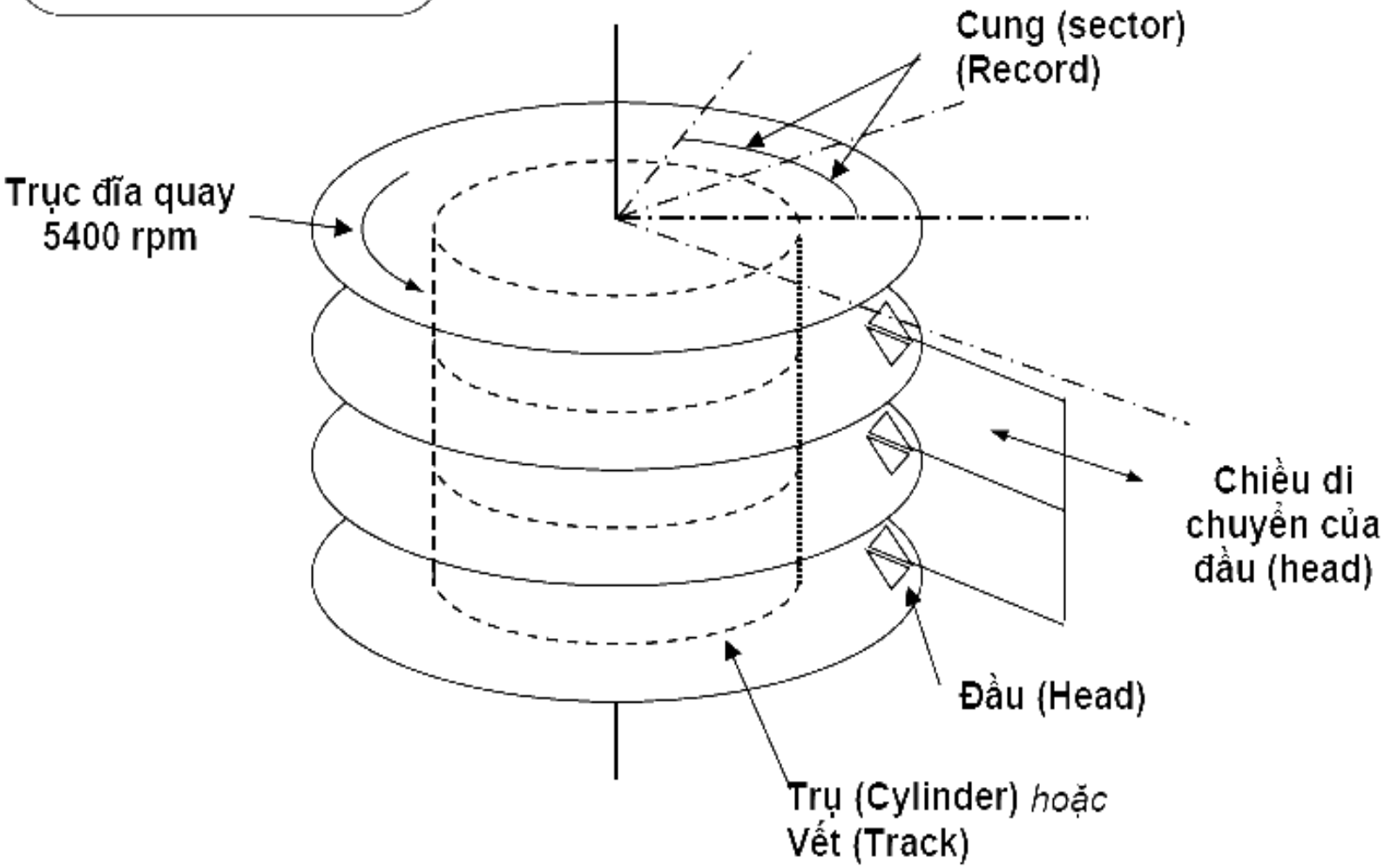


CÁC ĐẶC TÍNH LUẬN LÝ & VẬT LÝ CỦA ĐĨA TỪ

Ở mức vật lý : đĩa được tổ chức thành các Tracks, Cylinders, Sectors.

➔ Khả năng lưu trữ của đĩa được mô tả bằng 3 thông sô :
C (cylinder number)
H (Head side)
R (sector number)

Phân chia đĩa vật lý



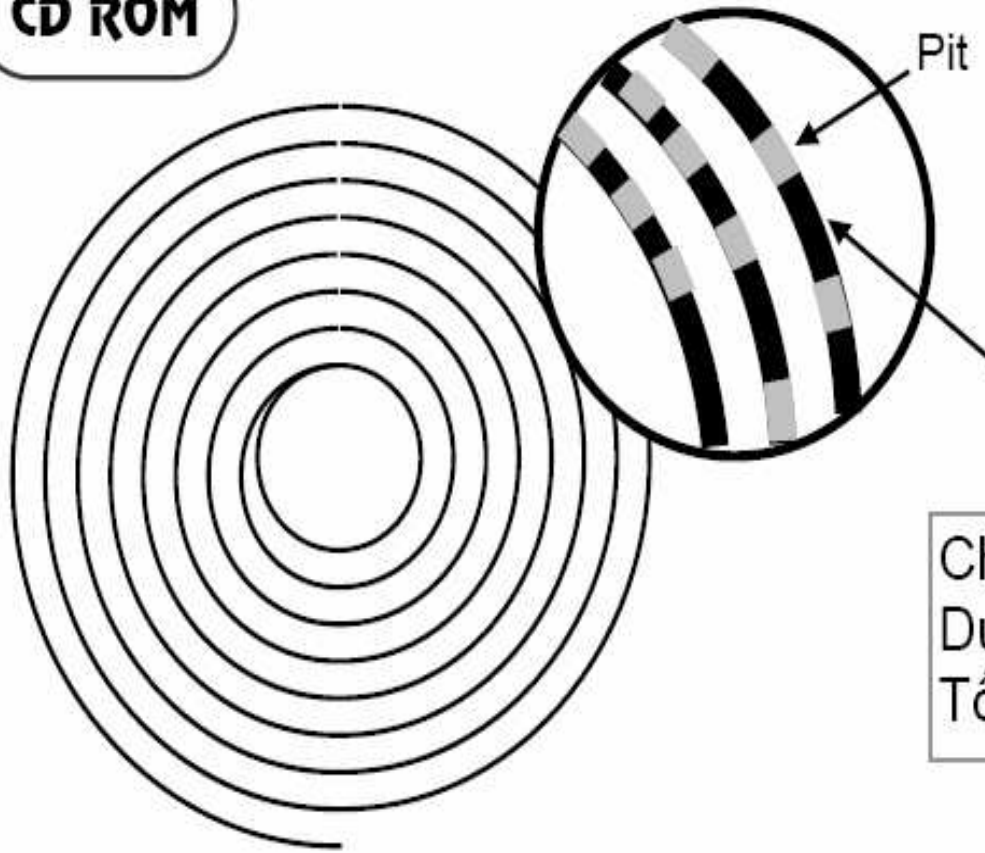
CÁC KHÁI NIỆM TRACK, CYLINDER, SECTOR

Tracks : là các vòng tròn đồng tâm được tạo ra trên bề mặt đĩa.

Cylinder : tập các tracks cùng bán kính trên 1 chồng đĩa. Mặt đĩa có bao nhiêu track thì sẽ có bấy nhiêu Cylinder.

Sector : là 1 đọan của track (cung từ) có khả năng lưu trữ 512 bytes dữ liệu. Các sector được đánh số bắt đầu từ 1 trên mỗi track → trên 1 đĩa tồn tại nhiều sector cùng số hiệu.

CD ROM



Chứa 330.000 khối dữ liệu.
Dung lượng 650 MB / 74 min
Tốc độ x1 = 153.60 KByte/s

Thông tin ghi theo rãnh (track) hình xoắn ốc
Dùng tia laser đục lỗ 1 μm trên rãnh gọi là Pit.
Phần không bị đục lỗ trên rãnh gọi là Land.

Ở mức luận lý : đĩa được tổ chức thành các Clusters, các files mà DOS sẽ dùng để cấp phát vùng lưu trữ cho dữ liệu cần lưu trữ.

Cluster : là 1 nhóm gồm 2,4,6 các sector kề nhau. Đó chính là đơn vị cấp phát vùng lưu trữ cho dữ liệu (file). Các cluster được đánh số bắt đầu từ 0.

Nếu dữ liệu cần lưu trữ chỉ 1 byte thì hệ điều hành cũng cấp phát 1 cluster.

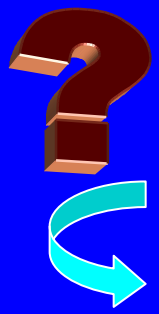
số bytes/cluster hay sector/cluster tùy thuộc vào từng loại đĩa.

TƯƠNG QUAN GIỮA SECTOR VẬT LÝ VÀ SECTOR LOGIC TRÊN ĐĨA MỀM

MẶT ĐĨA	TRACK	SECTOR	SECTOR LOGIC	THÔNG TIN
0	0	1	0	BOOT RECORD
0	0	2-5	1-4	FAT
0	0	6-9	5-8	Thư mục gốc
1	0	1-3	9-11	Thư mục gốc
1	0	4-9	12-17	Dữ liệu
0	1	1-9	18-26	Dữ liệu

BAD SECTOR

Trên bề mặt đĩa có thể tồn tại các sector mà HĐH không thể ghi dữ liệu vào đó hoặc không thể đọc dữ liệu từ đó. Các sector này gọi là Bad Sector.



Làm sao biết sector nào là bad sector

Kiểm tra giá trị của các phần tử (entry) trong bảng FAT, phần tử nào chứa giá trị (F)FF7H thì cluster tương ứng bị Bad

BẢNG FAT
FILE ALLOCATION TABLE

DOS quản lý các File nhờ vào 1 bảng gọi là bảng FAT.

Trong bảng FAT có ghi cluster bắt đầu của File này ở đâu ? Và đĩa còn bao nhiêu Clusters trống chưa cấp phát.

tổ chức luận lý của đĩa được mô tả như hình sau :

Thí dụ về bảng FAT

Đĩa mềm 3.5”” 360K thì :

Sector 0 : boot sector

Sector 1-4 : bảng FAT

Sector 5 – 11 : thư mục gốc

Sector 12-719 : vùng chứa data

BOOT RECORD

- Còn được gọi là Boot Sector. Ổ đĩa cứng gọi là Master boot, là Sector đầu tiên khi đĩa được format.
- chứa 1 chương trình nhỏ cho biết dạng lưu trữ trên đĩa và tên hệ thống MT, kiểm tra xem có các file hệ thống IO.SYS, MSDOS.SYS, COMMAND.COM hay không ?
- nếu có thì nạp chúng vào bộ nhớ (gọi là chương trình mồi của HĐH)

BOOT RECORD (tt)

- **Tọa độ vật lý :**

C=0, H=0, R =1 (C0H0R1) tức ở tại sector đầu tiên của track đầu tiên, mặt trên của đĩa đầu tiên trong ổ đĩa cứng.

- **Trong Master boot có chứa bảng PARTITION TABLE cho biết tầm địa chỉ vật lý (dung lượng) của ổ đĩa luận lý.**

Master boot không thuộc Partition nào

BOOT RECORD (tt)

- BOOT RECORD được ROM BIOS nạp vào địa chỉ 0000:7C00H.
- Nếu máy không bị Virus thì lệnh đầu tiên của chương trình BOOT là JMP 7C3EH, nghĩa là nhảy đến chương trình nạp mồi.
- chương trình nạp mồi (Bootstrap Loader) nạp thành phần cốt lõi của DOS lên RAM trong quá trình khởi động MT.

THÔNG TIN TRONG MASTER BOOT

BYTEBĐẦU	SỐ BYTES	THÔNG TIN
00H	3	chỉ thị nhảy về nới chứa CT nạp mồi
03H	8	Tên nhà sản xuất và hệ điều hành
0BH	2	Bytes/sector
0DH	1	Sector/block (mỗi block >=1 sector)
0EH	2	Số lượng Sectors không dùng đến kể từ sector 0.
10h	1	Số lượng bảng FAT

THÔNG TIN TRONG MASTER BOOT

BYTE BĐẦU	SỐ BYTES	THÔNG TIN
11H	2	Số Entry của thư mục gốc ổ đĩa.
13H	2	Tổng số sector của ổ đĩa logic này.
15H	1	Byte mô tả
16H	2	Số sector cho 1 bảng FAT
18H	2	Số Sectors trong 1 track.
1AH	2	Số lượng đầu đọc
1CH	4	Số lượng sector ẩn
20H	4	Tổng số sectors

THÔNG TIN TRONG MASTER BOOT

BYTE BĐẦU	SỐ BYTES	THÔNG TIN
3EH		Bootstrap
….		
1BEH	64	PARTITION TABLE
……		
1FEH	1	Giá trị 55H
1FFH	1	Giá trị 0AAH

THÔNG TIN TRONG MASTER BOOT

Từ thông tin trong bảng FORMAT, ta tính được địa chỉ của bảng FAT1, FAT2, Thư mục gốc ổ đĩa, địa chỉ bắt đầu của vùng dữ liệu.

BẢNG FAT

- Bảng chứa các danh sách liên kết các clusters. Mỗi danh sách trong bảng cho DOS biết rằng các clusters nào đã cấp phát, các clusters nào chưa dùng.
  - tùy theo ổ đĩa có thể có 1 hay 2 bảng FAT, bảng FAT2 để dự phỏng.
- có 2 loại bảng FAT :

bảng có Entry 12 bit cho đĩa mềm.

bảng có Entry 16 bit cho đĩa cứng.

PARTITON TABLE

64 Bytes của Partiton table được chia làm 4, mỗi phần 16 bytes mô tả cho 1 partition các thông tin sau :

Bytes	Mô tả
00H	active flag (=0 Non bootable =80H Bootable)
01H	starting head – Nơi bắt đầu Partittion
02H	starting cylinder

PARTION TABLE

Bằng FDISK của HĐH ta có thể chia không gian lưu trữ của đĩa cứng thành các phần khác nhau gọi là Partition.

DOS cho phép tạo ra 3 loại Partition :

Primary Dos, Extended Dos và None Dos

Ta có thể cài đặt các HĐH khác nhau lên các Partition khác nhau.

PARTITON TABLE

03H starting sector

04H parttition type :

- **0 Non Dos**
- **1 cho đĩa nhỏ 12 bit FAT Entry**
- **4 cho đĩa lớn 16 bit FAT Entry**
- **5 Extended Dos**

05H Ending nơi kết thúc Partition

06H Ending Cylinder

07H Ending Sector

08H, 0BH Starting sector for partition

0Ch,0FH Partition length in sectors

Một số thí dụ

kiểm tra Partition Active

- đọc sector đầu tiên của đĩa cứng lưu vào biến.
- kiểm tra offset 00 của 4 phần tử Partition trong Partition Table

```
MOV CX, 4
MOV SI, 1BEH
PACTIVE :
MOV AL, MBOOT [SI]
CMP AL, 80H
JE ACTIVE
ADD SI, 16
LOOP PACTIVE
NO_ACTIVE :
.....
ACTIVE : .....
```

Một số thí dụ

Đọc nội dung của BootSector ghi vào biến dem

- đọc sector đầu tiên của đĩa cứng lưu vào buffer.
- tìm partition active (phần tử trong bảng partition có offset 80h)
- đọc byte tại offset 01h và word tại offset 02h của phần tử partition tương ứng ở trên (head, sector, cylinder) để xác định số hiệu bắt đầu của partition active → boot sector của đĩa cứng.
- đọc nội dung của sector đọc được ở trên lưu vào buffer.

Một số thí dụ

ACTIVE :

MOV AX, 0201H ; đọc 1 sector

MOV CX, WORD PTR MBOOT [SI+2] ; sector cylinder

MOV DH, BYTE PTR MBOOT[SI+1] ; head

MOV DL, 80H ; đĩa cứng

MOV ES, CS ; trỏ về đầu vùng buffer lưu

LEA BX, BUFFER

INT 13H

THƯ MỤC GỐC (ROOT DIRECTORY)

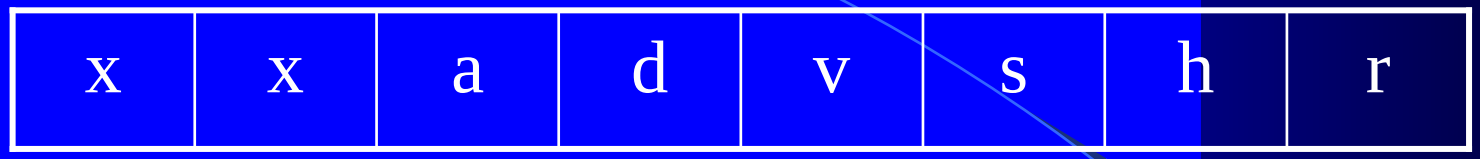
- Là danh sách tất cả các Files đã có trên đĩa, các thư mục cấp 1 đã có.
- Mỗi phần tử (32 bytes) trong bảng thư mục sẽ chứa thông tin về tên file hoặc là thư mục, kích thước, thuộc tính, cluster bắt đầu của file này hoặc cluster bắt đầu của thư mục thứ cấp (thư mục con).

mỗi bảng thư mục chứa tối đa 112 entry, mỗi entry là 32 bytes.

THƯ MỤC GỐC (ROOT DIRECTORY)

Offset	Nội dung	Kích thước
00H	tên chính của File	8 bytes
08H	phần mở rộng của tên file	3 bytes
0BH	thuộc tính của File	1 byte
0CH	dự trữ	10 bytes
16H	giờ thay đổi thông tin cuối cùng	2 bytes
18H	ngày thay đổi thông tin cuối cùng	2 bytes
1Ah	cluster đầu tiên của File	2 bytes
1CH	Kích thước File	4bytes

BYTE THUỘC TÍNH



x : không sử dụng

a : thuộc tính lưu trữ (Archive)

d : thuộc tính thư mục con (Sub – Directory)

v : thuộc tính nhãn đĩa (Volume)

s : thuộc tính hệ thống (System)

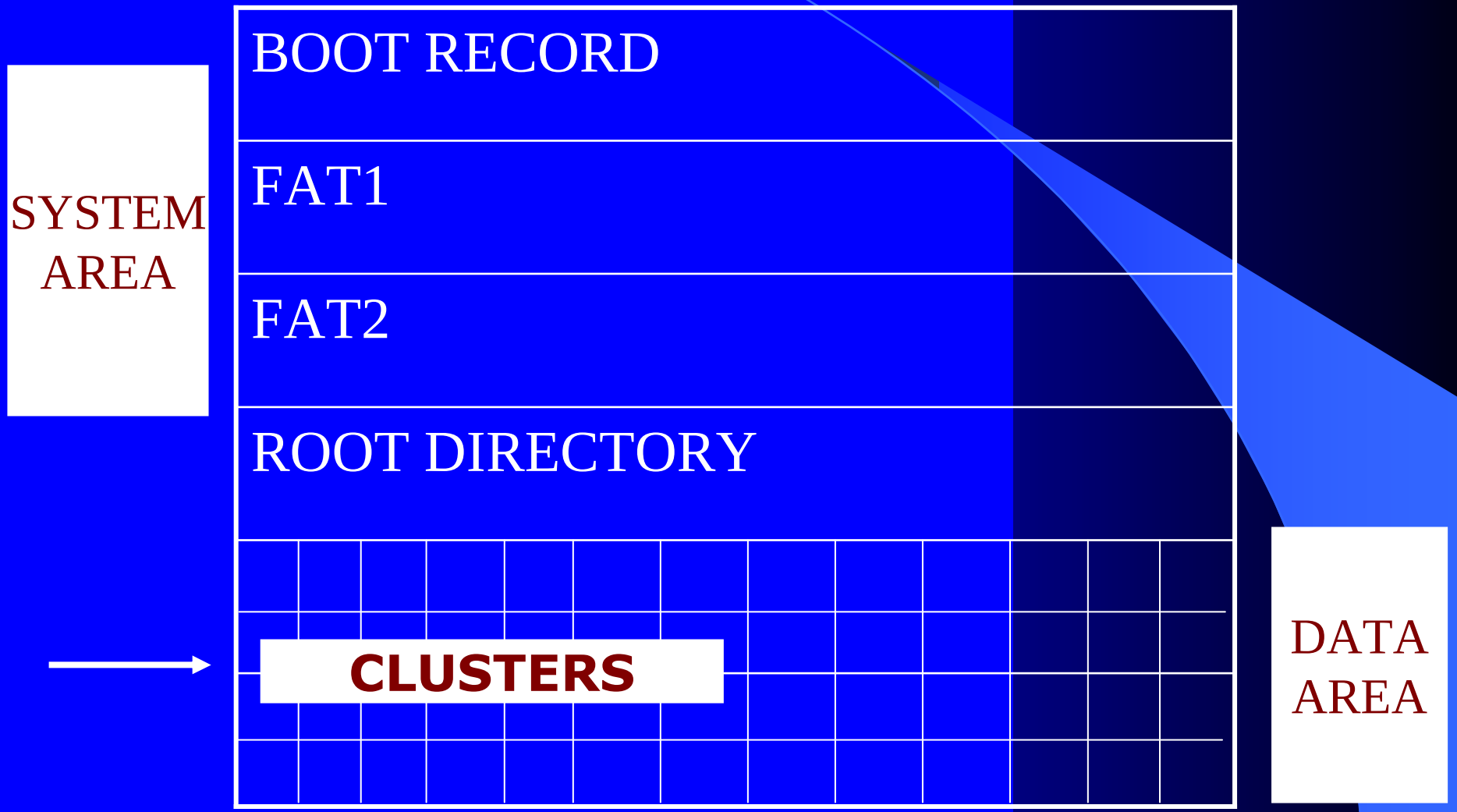
h : thuộc tính ẩn (Hidden)

r : thuộc tính chỉ đọc (Read Only)

VÙNG LƯU TRỮ

- là vùng dành cho việc lưu trữ dữ liệu.
- như vậy việc lưu trữ dữ liệu trên đĩa có cấu trúc là 1 danh sách liên kết mà bảng thư mục gốc là đầu của danh sách liên kết.
- đầu mỗi cluster luôn luôn chứa địa chỉ của cluster sau nó cho biết phần còn lại của file là cluster nào. Nếu giá trị này là 0 thì cluster này là cluster cuối cùng.

SỰ PHÂN VÙNG TRÊN ĐĨA



CÁC LOẠI ĐĨA

Disk Type	sides	track per side	sectors per track	total sector	cluster size	total bytes
360K	2	40	9	720	1,024	368,640
720K	2	80	9	1,440	512	737,280
1.2MB	2	80	15	2,400	512	1,228,800
1.4MB	2	80	18	2,880	512	1,474,560
32MB	6	614	17	62,610	2,048	32,056,832

TÍNH DUNG LƯỢNG ĐĨA

Công thức tính dung lượng đĩa :

Dung lượng đĩa (bytes) = số byte/1 sector * số sector/1 track * số track/ 1 mặt đĩa * số mặt đĩa.

MỘT SỐ HÀM THAO TÁC VỚI FILE VÀ ĐĨA INT 21H

HÀM 36H INT 21H :

Lấy số bytes còn trống trên đĩa

Input :

AH = 36H DL = O63 đĩa (0 : mặc định, 1 ổ A ....

Output :

Có lỗi AX = 0FFFFH

Không lỗi : AX = số sector / cluster

BX = số cluster còn trống

DX = tổng số cluster trên đĩa

CX = số bytes/cluster

BÀI TẬP

Viết chương trình tạo thư mục với yêu cầu tên thư mục (có thể bao gồm tên ổ đĩa, đường dẫn và tên thư mục) được nhập từ bàn phím, cho phép sửa sai khi gỏ nhầm tên thư mục.

Viết chương trình ghi dữ liệu vào file với yêu cầu :

- **Tên file nhập từ bàn phím**
- **Dữ liệu ghi vào file cũng gỏ từ bàn phím và kết thúc việc nhập bằng phím CTRL+Z**

Viết chương trình gộp nội dung 1 file vào cuối 1 file khác.

LẬP TRÌNH XỬ LÝ FILE

GiỚI THIỆU FILE
CÁC HÀM CHỨC NĂNG XỬ LÝ FILE
CỦA INT 21H CỦA DOS

GIỚI THIỆU FILE

- Trong quản lý File, Dos vay mượn khái niệm Handle trong HĐH Unix để truy xuất File và thiết bị.

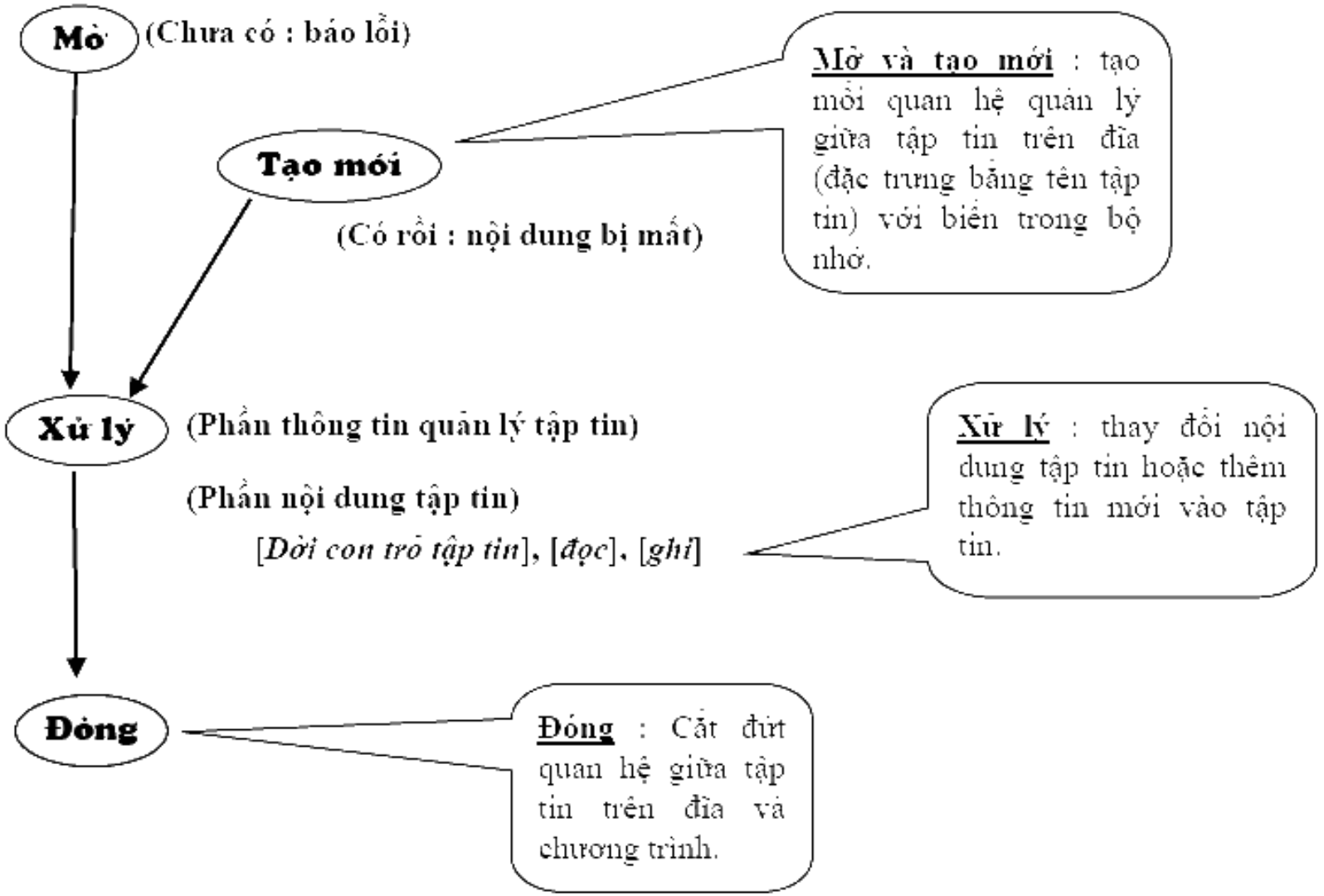
- Handle là 1 số 16 bits được Dos sử dụng để nhận biết File đã mở hoặc 1 thiết bị trong hệ thống.

GIỚI THIỆU FILE

- **Có 5 Handle thiết bị chuẩn được Dos nhận dạng.**

Handle	Thiết bị
0	Keyboard, standard input
1	Console, standard output
2	Error output thiết bị xuất lỗi – màn hình
3	Auxiliary device asynchronous
4	Printer

CÁC THAO TÁC XỬ LÝ FILE



CÁC CHỨC NĂNG CƠ BẢN VỀ XỬ LÝ FILE CỦA INT 21H

CÁC CHỨC NĂNG NÀY PHẢI ĐƯA VÀO AH

Chức năng	Tác vụ
3Ch	Tạo File mới
3Dh	Mở File đã có để xuất/nhập/vừa nhập vừa xuất
3Eh	Đóng thẻ File
3Fh	Đọc từ File hay đọc từ thiết bị 1 số bytes định trước
40h	Ghi vào File hay đọc từ thiết bị 1 số bytes định trước
42h	di chuyển con trỏ File trước khi đọc/ ghi

CHỨC NĂNG TẠO FILE 3Ch
CREATE FILE FUNCTION 3Ch

Chức năng : Mở 1 File mới để đọc ghi. Nếu file đã có thì file cũ sẽ bị xóa.

AH = 3Ch

DS:DX địa chỉ của tên File muốn mở (ASCIIZ String)

CX = thuộc tính File

(0 normal 1 ReadOnly 2 Hidden 4 System)

Xuất : không lỗi CF =0 AX = File Handle Có lỗi CF =1.

Mã lỗi trong AX (3,4,5).

CHỨC NĂNG TẠO FILE 3Ch
CREATE FILE FUNCTION 3Ch

Ex :

```
CREATE_FILE :
MOV AH, 3CH
MOV DX, OFFSET NEWFILE
MOV CX, 0
INT 21H
JC DISPLAY_ERROR
MOV NEWFILEHANDLE, AX
...
NEWFILE DB ‘ FILE1.DOC ’,0
NEWFILEHANDLE  DW ?
```

CHỨC NĂNG TẠO FILE 3Ch
CREATE FILE FUNCTION 3Ch

Ex :

CHỨC NĂNG 3Ch CÓ 1 KHUYẾT ĐIỂM LÀ NẾU CÓ 1 FILE CÙNG TÊN(CÙNG ĐƯỜNG DẪN) ĐÃ TỒN TẠI THÌ FILE CŨ SẼ BỊ XÓA.

ĐỂ BẢO VỆ FILE, CÓ 2 CÁCH :

C1 : MỞ FILE BẰNG CHỨC NĂNG 3Dh, NẾU FILE CHƯA CÓ THÌ TRẢ VỀ LỖI SỐ 2 (FILE NOT FOUND) ➔ YÊN TÂM MỞ FILE MỚI.

C2 : DÙNG CHỨC NĂNG 5Bh MỞ FILE CÓ KIỂM TRA TÊN FILE NÀY ĐÃ CÓ CHƯA.

CHỨC NĂNG 5Bh
TẠO FILE MỚI CÓ KIỂM TRA

ĐIỀU KIỆN : GIỐNG CHỨC NĂNG 3Ch

NẾU FILE NÀY ĐÃ CÓ THÌ KHÔNG MỞ FILE MỚI MÀ TRẢ VỀ LỖI 50h

```
CREATE_FILE :
MOV AH,5BH
MOV DX, OFFSET FILENAME
MOV CX, 0
INT 21H
JC  ERROR
….
FILENAME  DB ‘FILE1.DOC’ , 0
```

CÁC LỖI KHI MỞ FILE

MÃ LỖI	DIỄN GIẢI
2	FILE NOT FOUND KHÔNG TÌM THẤY FILE, CÓ THỂ ĐƯỜNG DẪN KHÔNG ĐÚNG HOẶC TÊN FILE MÔ TẢ KHÔNG HỢP LỆ.
3	PATH NOT FOUND ĐƯỜNG DẪN KHÔNG CÓ.
4	TOO MANY OPEN FILES CÓ THỂ DO LỆNH PATH XX TRONG CONFIG.SYS QUÁ NHỎ KHÔNG CHO PHÉP MỞ NHIỀU FILE.
5	ACCESS DENIED TỪ CHỐI TRUY XUẤT. CÓ THỂ TA MUỐN XOÁ FILE ĐANG MỞ, HAY FILE NÀY CÓ THUỘC TÍNH CHỈ ĐỌC.

CH Mã truy nhập không hợp lệ.

FH Ổ đĩa không hợp lệ

10h Đang tìm cách xóa thư mục hiện thời

CÁC LỖI KHI MỞ FILE

MÃ LỖI	DIỄN GIẢI
11H	Không cùng thiết bị
12H	Không tìm được thêm File nào

CHỨC NĂNG MỞ FILE ĐÃ CÓ 3Dh Int 21h
OPEN FILE

ĐIỀU KIỆN :

AH = 3DH DS:DX ĐỊA CHỈ TÊN FILE

AL = MODE

0: INPUT (MỞ CHỈ ĐỌC)
1 : OUTPUT (MỞ ĐỂ GHI)

2 : INPUT OUTPUT (MỞ VỪA ĐỌC VỪA GHI)

XUẤT :

KHÔNG LỖI CF = 0 AX = FILE HANDLE

CÓ LỖI CF = 1 AX ← mã lỗi (2,4,512)

MỞ FILE HÀM 3CH INT 21H

- Trước khi sử dụng 1 file, ta phải mở nó.
- Để tạo 1 file mới hay ghi lại 1 file cũ, ta sử dụng tên file và thuộc tính của File.
- ➔ DOS trả về thẻ file

MỞ FILE HÀM 3CH INT 21H

AH = 3CH
DS:DX địa chỉ của chuỗi ASCII
(chuỗi tên File kết thúc bằng byte 0)
CL = thuộc tính File
Nếu thành công, AX = thẻ File
Nếu CF được set thì có lỗi, mã lỗi chứa trong AX
(lỗi 3,4,5)

Viết code mở 1 File mới với thuộc tính chỉ đọc, tên File là FILE1

```
Fname DB ‘FILE1’,0
FHANDLE DW ?
MOV AX,@DATA
MOV DS,AX
MOV AH,3CH
MOV CL,1
LEA DX,FNAME
INT 21H
MOV FHANDLE, AX
JC OPEN_ERROR
……
```

CHỨC NĂNG MỞ FILE ĐÃ CÓ SẴN
HÀM 3Dh INT 21H
OPEN FILE

AH = 3DH

DS:DX = địa chỉ của chuỗi ASCII
(chuỗi tên File kết thúc bằng byte 0)

AL = mã truy cập

- 0 : mở để đọc
- 1 : mở để ghi
- 2 : mở để đọc và ghi

➔Thành công, AX = Fhandle

➔Có lỗi. Mã lỗi chứa trong AX (2,4,5,12)

CHỨC NĂNG MỞ FILE ĐÃ CÓ SẴN
HÀM 3Dh INT 21H
OPEN FILE

```
MOV AH, 3DH
MOV AL, 0
MOV DX, OFFSET FILENAME
INT 21H
JC DISPLAY_ERROR
MOV  INFILEHANDLE, AX
 …..
INFILE  DB ‘ D:\FILE1.DOC’, 0
INFILEHANDLE  DW  ?
```

CHỨC NĂNG 3EH ĐÓNG FILE

ĐIỀU KIỆN :

AH = 3EH BX = FILE HANDLE CẦN ĐÓNG

XUẤT :

KHÔNG LỖI CF = 0 CÓ LỖI CF = 1

EX :

```
MOV AH, 3EH
MOV BX, INFILEHANDLE
INT 21H
JC DISPLAY_ERROR
.....
INFILE  DB 'D:\FIEL1.DOC', 0
INFILEHANDLE  DW ?
```

LỖI SỐ 6 : INVALID HANDLE

FILE HANDLE TRONG BX KHÔNG PHẢI LÀ THẺ FILE CỦA FILE ĐÃ MỞ.

CHỨC NĂNG 3FH ĐỌC FILE

ĐỌC 1 SỐ BYTES TỪ FILE LƯU VÀO BỘ NHỚ

ĐIỀU KIỆN :

AH = 3FH BX = FILE HANDLE , CX = SỐ BYTES CẦN ĐỌC

DS:DX : ĐỊA CHỈ BỘ ĐỆM.

XUẤT :

AX = SỐ BYTES ĐỌC ĐƯỢC, NẾU AX = 0 HAY AX<CX FILE ĐÃ KẾT THÚC.

NẾU CỜ CF ĐƯỢC LẬP → CÓ LỖI, MÃ LỖI CHỨA TRONG AX(5,6)

CHỨC NĂNG 3FH ĐỌC FILE

EX : ĐỌC 1 SECTOR 512 BYTES TỪ FILE

```
.DATA
HANDLE DW ?
BUFFER DB 512 DUP(?)
MOV AX, @DATA
MOV DS, AX
MOV AH, 3FH
MOV CX, 512
MOV BX, HANDLE
MOV CX, 512
INT 21H
JC READ_ERROR
```

NẾU CẦN ĐỌC HẾT CÁC SECTOR CHO ĐẾN HẾT FILE ➔ EOF

```
CMP AX, CX
JL  EXIT
JMP READ_LOOP
```

CHỨC NĂNG 40H GHI FILE

GHI 1 SỐ BYTES LÊN FILE HAY THIẾT BỊ

INPUT :

AH =40H BX = THẺ FILE CX = SỐ BYTES CẦN GHI

DS:DX : ĐỊA CHỈ VÙNG ĐỆM.

OUTPUT :

AX : SỐ BYTES GHI ĐƯỢC, NẾU AX<CX, CÓ LỖI (ĐĨA ĐẦY).NẾU CF ĐƯỢC LẬP → CÓ LỖI, MÃ LỖI TRONG AX (5,6).

HÀM 40H CŨNG CÓ THỂ DÙNG ĐỂ ĐƯA DỮ LIỆU RA MÀN HÌNH

CON TRỎ FILE

- DÙNG ĐỂ ĐỊNH VỊ TRONG FILE.
- KHI FILE ĐƯỢC MỞ, CON TRỎ FILE NẰM Ở ĐẦU FILE.
- SAU MỖI THAO TÁC ĐỌC, CON TRỎ FILE SẼ DI CHUYỂN ĐẾN BYTE KẾ.
- SAU KHI GHI 1 FILE MỚI CON TRỎ CHỈ ĐẾN CUỐI FILE (EOF).
- ĐỂ DI CHUYỂN CON TRỎ FILE HÀM 42H

MINH HỌA LẬP TRÌNH FILE

Viết chương trình cho phép User gỏ vào tên File (có thể có kèm theo tên ổ đĩa, thư mục chứa fiel), chương trình sẽ đọc và hiển thị nội dung File ra màn hình.

DỊCH CHUYỂN CON TRỎ FILE
HÀM 42H INT 21H

AH = 42H AL = PHƯƠNG THỨC TRUY NHẬP

0 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI ĐẦU FILE.

1 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI VỊ TRÍ HIỆN THỜI CỦA CON TRỎ.

2 DỊCH CHUYỂN TƯƠNG ĐỐI SO VỚI CUỐI FILE.

BX = THẺ FILE.

CX : DX SỐ BYTES CẦN DỊCH CHUYỂN.

OUTPUT :

DX:AX : VỊ TRÍ MỚI CỦA CON TRỎ FILE TÍNH BẰNG BYTE TỪ ĐẦU FILE.

NẾU CF =1 MÃ LỖI TRONG AX (1, 6).

DỊCH CHUYỂN CON TRỎ FILE HÀM 42H INT 21H

CX : DX CHỨA SỐ BYTES ĐỂ DI CHUYỂN CON TRỎ. NẾU LÀ SỐ DƯƠNG → CHUYỂN VỀ CUỐI FILE.

NẾU LÀ SỐ ÂM →CHUYỂN VỀ ĐẦU FILE.

DI CHUYỂN CON TRỎ FILE ĐẾN CUỐI FILE VÀ XÁC ĐỊNH KÍCH THƯỚC FILE

```
MOV AH, 42H          ; DI CHUYỂN CON TRỎ FILE
MOV BX, HANDLE   ; LẤY THẺ FILE
XOR DX, DX
XOR CX, CX   ; DỊCH CHUYỂN 0 BYTE
MOV AL, 2     ; TÍNH TỪ CUỐI FILE
INT 21H  ; CHUYỂN CON TRỎ ĐẾN CUỐI FILE, DX:AX KÍCH THƯỚC FILE
JC MOVE_ERROR
```

THAY ĐỔI THUỘC TÍNH FILE HÀM 43H INT 21H

INPUT :

AH = 43H DS :DX = ĐỊA CHỈ CHUỖI ASCII STRING

AL = 0 ĐỂ LẤY THUỘC TÍNH FILE AL =1 ĐỂ THAY ĐỔI THUỘC TÍNH FILE, CX = THUỘC TÍNH FILE MỚI (NẾU AL =1)

OUTPUT :

NẾU THÀNH CÔNG, CX = THUỘC TÍNH HIỆN THỜI

NẾU CF ĐƯỢC LẬP → CÓ LỖI, MÃ LỖI TRONG AX (2,3,5).

Ex : thay đổi thuộc tính File thành hidden file

```
MOV AH, 43H              ; Hàm lấy / đổi thuộc tính File
MOV AL, 1                ; tuỳ chọn thay đổi thuộc tính
LEA DX, FILENAME         ; lấy tên file kế cả đường dẫn.
MOV CX, 1                I; thuộc tính Hideen
INT 21H                  ; đổi thuộc tính
JC ATT_ERROR             ; thoát nếu có lỗi, mã lỗi trong AX
```

LẬP TRÌNH FILE

1. Viết chương trình chép một file nguồn đến một file đích trong đó thay chữ thường bằng chữ hoa.
2. Viết chương trình đọc 2 file và hiển thị chúng bên cạnh nhau trên màn hình. Chú ý có chức năng dừng từng trang màn hình nếu file quá dài.
3. Viết chương trình ghép nội dung 1 file vào cuối 1 file khác đã có.
4. Viết chương trình tạo 1 thư mục, tên thư mục được gỏ từ bàn phím (tên thư mục có thể bao gồm tên ổ đĩa, đường dẫn).

Chương 13 :LẬP TRÌNH XỬ LÝ MẢNG & CHUỖI

- GiỚI THIỆU
- CỜ HƯỚNG DF
- CÁC LỆNH THIẾT LẬP VÀ XÓA CỜ HƯỚNG
- CÁC LỆNH THAO TÁC TRÊN CHUỖI
- MỘT SỐ THÍ DỤ MINH HỌA
- THƯ ViỆN LIÊN QUAN ĐẾN CHUỖI

GiỚI THIỆU CHUỖI

Trong ASM 8086 khái niệm chuỗi bộ nhớ hay chuỗi là 1 mảng các byte hay word.

➔ Các lệnh thao tác với chuỗi cũng được thiết kế cho các thao tác với mảng.

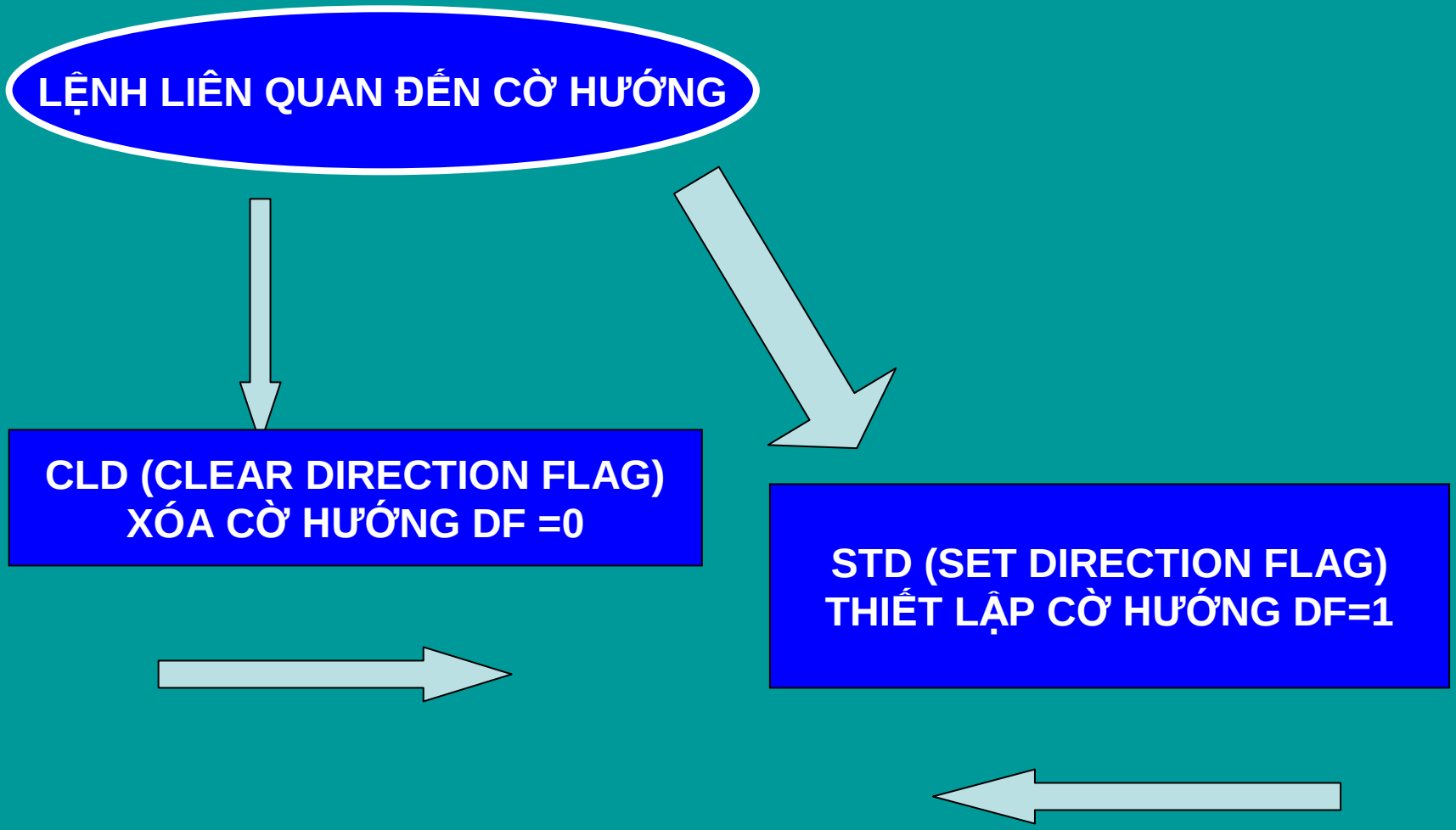
Cờ hướng DF

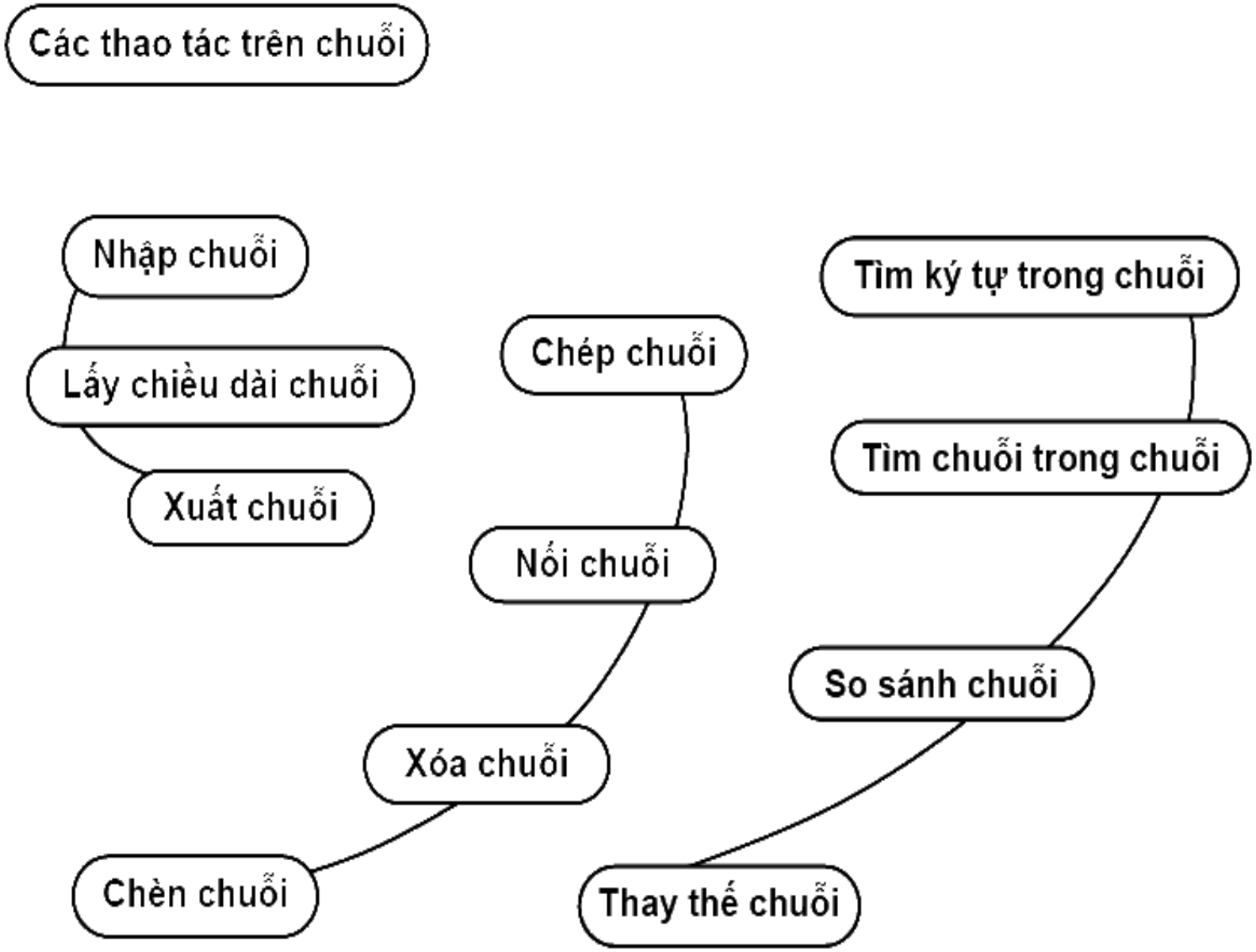
Cờ định hướng (Direction Flag) : xác định hướng cho các thao tác chuỗi.

DF=0 chuỗi được xử lý theo chiều tăng tức địa chỉ vùng nhớ chứa chuỗi tăng dần.
(chuỗi được xử lý từ trái qua phải).

DF=1 chuỗi được xử lý theo chiều tăng tức địa chỉ vùng nhớ chứa chuỗi giảm dần.
(chuỗi được xử lý từ phải qua trái).

Trong DEBUG DF=0 ký hiệu là UP DF=1 ký hiệu là DN


LỆNH LIÊN QUAN ĐẾN CỜ HƯỚNG
CLD (CLEAR DIRECTION FLAG)
XÓA CỜ HƯỚNG DF =0
STD (SET DIRECTION FLAG)
THIẾT LẬP CỜ HƯỚNG DF=1

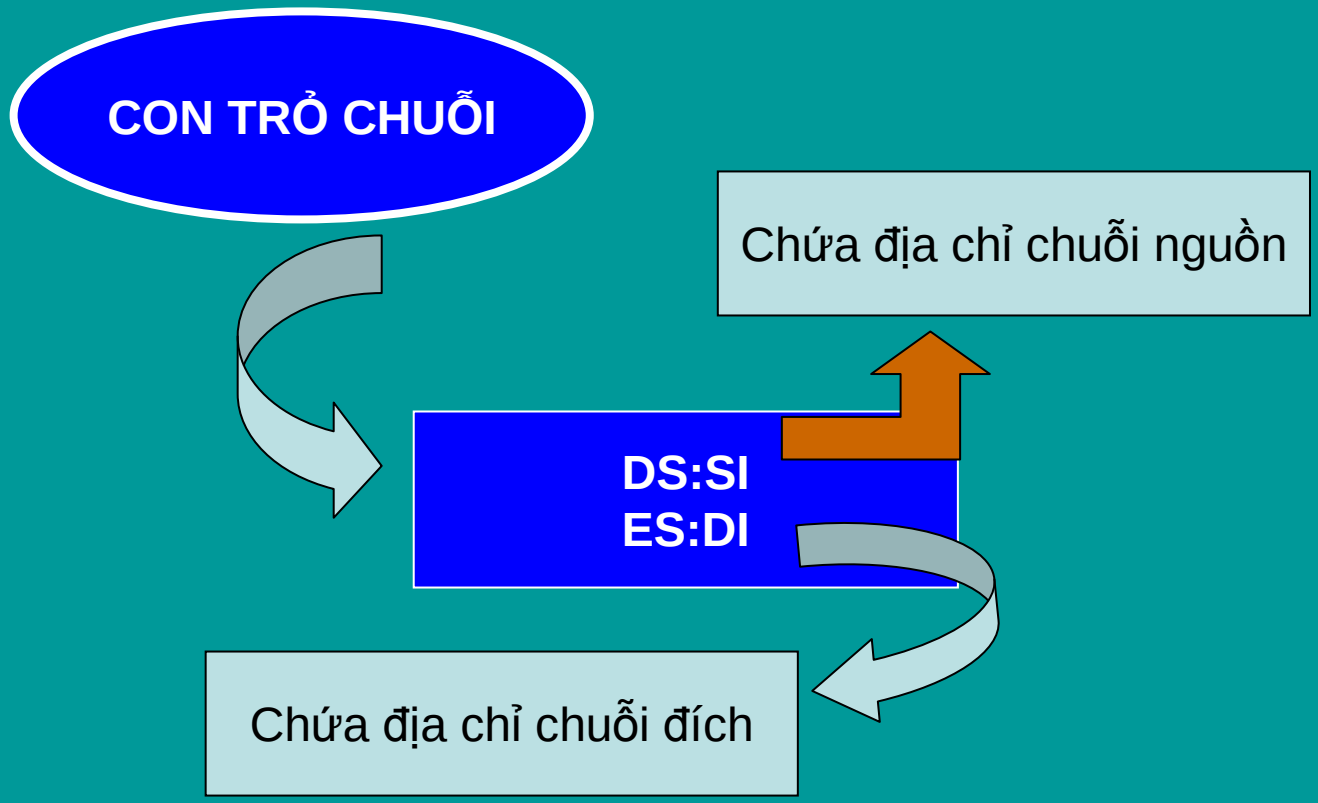

Các thao tác trên chuỗi
Nhập chuỗi
Lấy chiều dài chuỗi
Xuất chuỗi
Chép chuỗi
Nối chuỗi
Xóa chuỗi
Chèn chuỗi
Tìm ký tự trong chuỗi
Tìm chuỗi trong chuỗi
So sánh chuỗi
Thay thế chuỗi

➔ Trước khi sử dụng các lệnh xử lý chuỗi, ta phải xác định hướng xử lý chuỗi bằng cách set hay clear cờ hướng.

Lệnh đặt cờ hướng :

CLD : xóa cờ hướng, chuỗi được xử lý từ trái → phải

STD : đặt cờ hướng, chuỗi được xử lý từ phải → trái

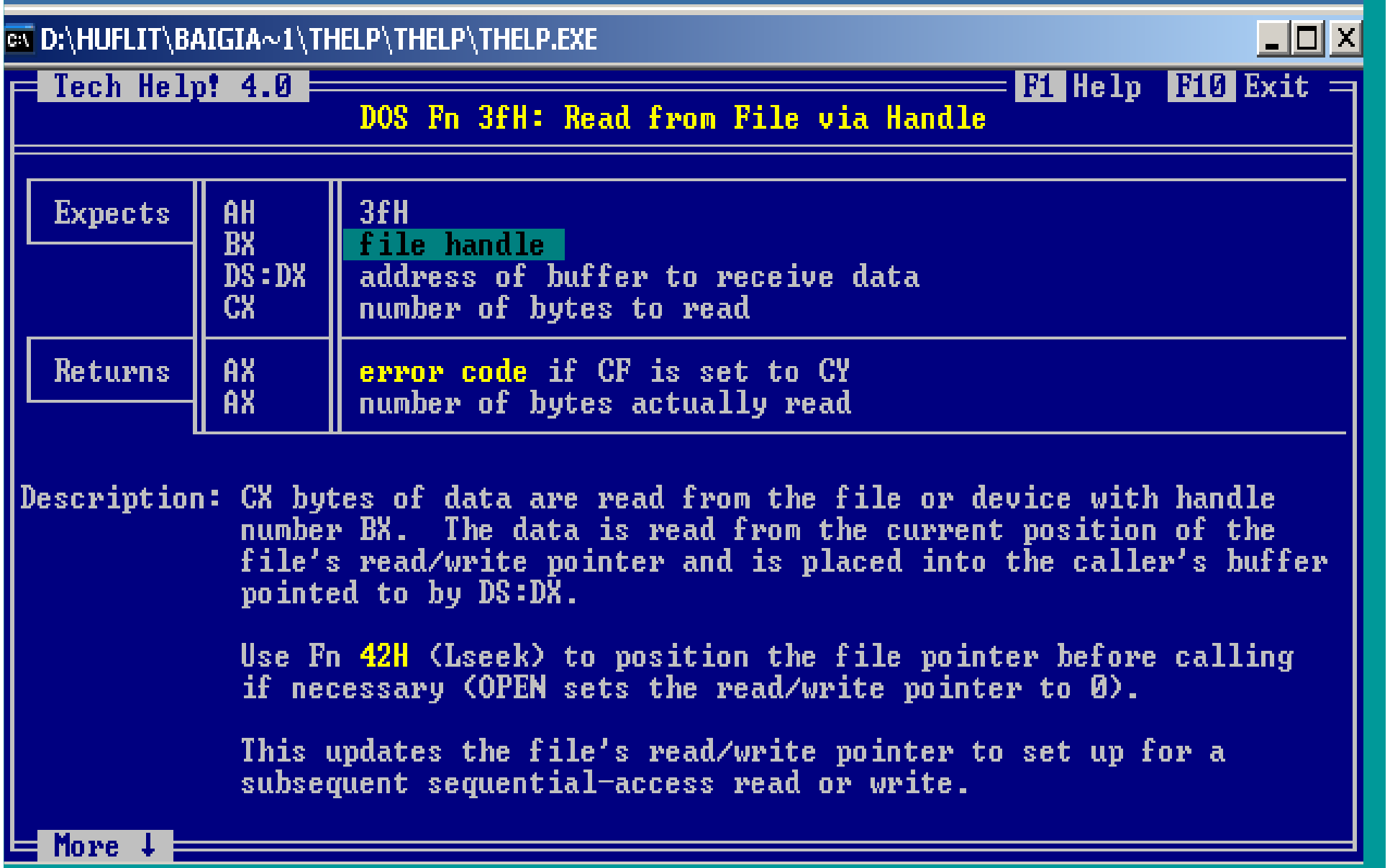


CÁC THAO TÁC XỬ LÝ CHUỖI

NHẬP CHUỖI

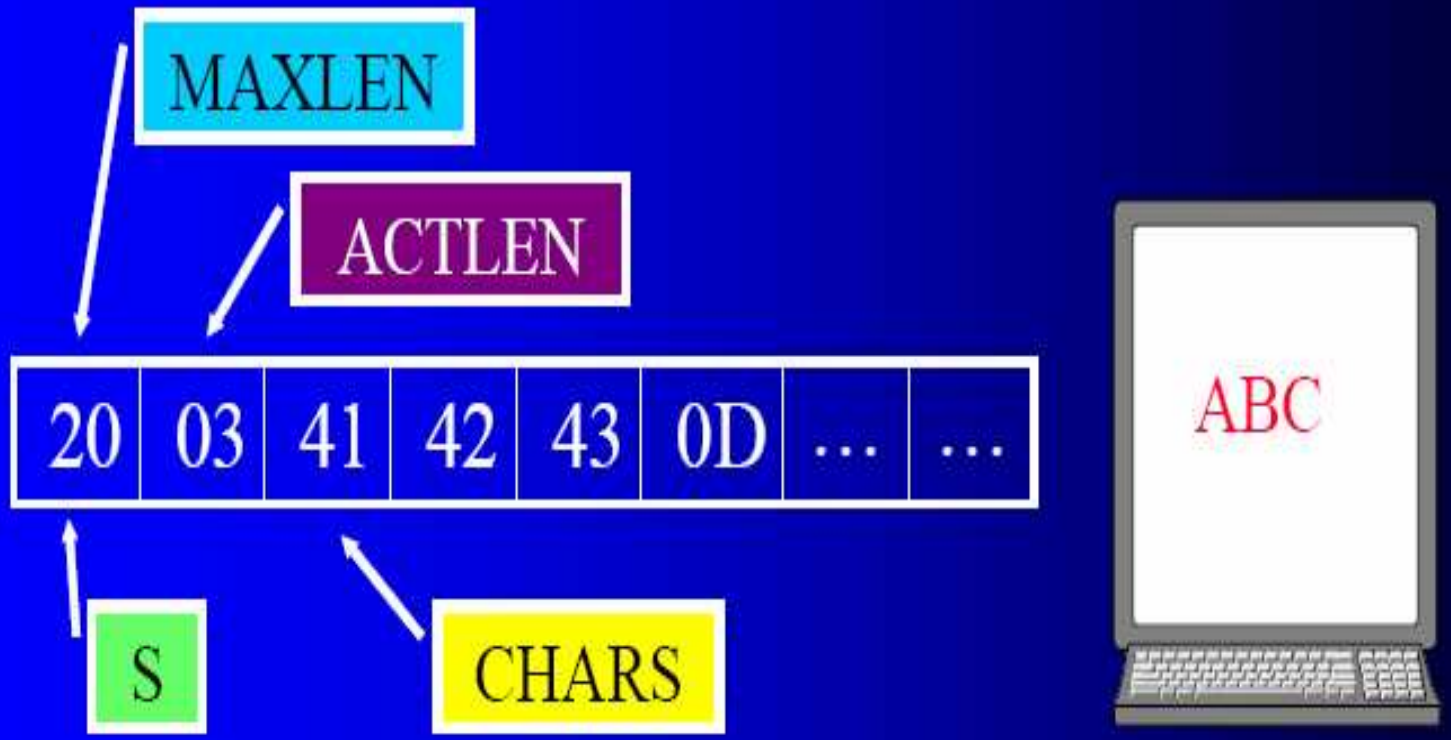
Input : AH = 0AH, ngắt 21H
DS:DX = địa chỉ của buffer, trong đó buffer[0] là kích thước tối đa của chuỗi,
buffer[1] sẽ là kích thước dữ liệu nhập.
Output : Chuỗi buffer chứa nội dung nhập vào từ buffer[2] trở đi

Yêu cầu xem thêm các chức năng AH = 3FH và AH = 40H của ngắt 21H.


D:\HUFLIT\BAIGIA~1\THELP\THELP\THELP.EXE
Tech Help! 4.0
F1 Help F10 Exit
DOS Fn 3fH: Read from File via Handle
Expects
AH 3fH
BX file handle
DS:DX address of buffer to receive data
CX number of bytes to read
Returns
AX error code if CF is set to CY
AX number of bytes actually read
Description: CX bytes of data are read from the file or device with handle number BX. The data is read from the current position of the file's read/write pointer and is placed into the caller's buffer pointed to by DS:DX.
Use Fn 42H (Lseek) to position the file pointer before calling if necessary (OPEN sets the read/write pointer to 0).
This updates the file's read/write pointer to set up for a subsequent sequential-access read or write.
More ↓

NHẬP CHUỖI

- Hàm 0Ah, INT 21h: Nhập chuỗi từ bàn phím, kết thúc Enter



NHẬP CHUỖI

Ta cũng có thể dùng hàm 1 INT 21h đọc 1 ký tự từ bàn phím để nhập 1 chuỗi bằng cách dùng vòng lặp và lưu chuỗi bằng lệnh STOSB.

NHẬP CHUỖI

Ta cũng có thể dùng hàm 1 Int 21h đọc 1 ký tự từ bàn phím để nhập 1 chuỗi bằng cách dùng vòng lặp và lưu chuỗi bằng lệnh STOSW.

STOSW (STORE STRING WORD)

CHUYỂN NỘI DUNG AX ĐẾN WORD ĐƯỢC TRỎ BỞI ES:DI.
SAU KHI LỆNH ĐƯỢC THỰC HiỆN DI TĂNG HAY GiẢM 2 TÙY VÀO DF.

LƯU CHUỖI CÁC WORD

THÍ DỤ

```
.MODEL SMALL
.STACK 100H
.DATA
  STRING1 DB 'HELLO'
  .CODE
 MAIN PROC
  MOV AX,@DATA
  MOV ES,AX
  LEA DI, STRING1      ; khởi tạo ES
  CLD                  ; xử lý từ trái ➔ phải
 MOV AL,'A'            ; AL chứa ký tự cần lưu
 STOSB                 ; lưu ký tự 'A'
 STOSB                 ; lưu ký tự thứ 2
MOV AH,4CH
INT 21H
MAIN ENDP
END MAIN
```

THÍ DỤ

```
READSTR PROC
PUSH AX
PUSH DI
CLD
XOR BX,BX
MOV AH,1
INT 21H
LAP:
        CMP AL,0DH
        JE ENDLAP
        CMP AL,8H
        JNE ELSE1
        DEC DI
        DEC BX
        JMP READ
ELSE1 :
        STOSB
        INC BX
READ :
        INT 21H
        JMP LAP
ENDLAP :
        POP DI
        POP AX
RET
READSTR ENDP
```

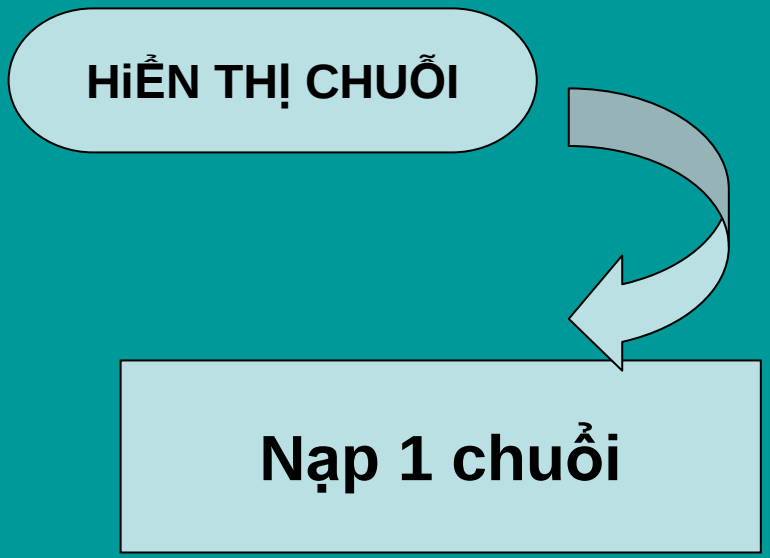
Giải thích :
DI chứa offset của chuỗi
BX chứa số ký tự nhập
8H mã ASCII của Backspace
không → lưu nó vào chuỗi
tăng số ký tự lên 1
Đúng → lùi con trỏ DI
giảm số ký tự nhập được

NHẬP XUẤT CHUỖI

HiỂN THỊ CHUỖI

AH = 09, ngắt 21H
Vào : DX = địa chỉ offset của chuỗi.
Chuỗi phải kết thúc bằng kí tự '$'.
Chú ý : thay vì dùng lệnh MOV OFFSET ta có thể dùng lệnh LEA.

CÁC THAO TÁC XỬ LÝ CHUỖI



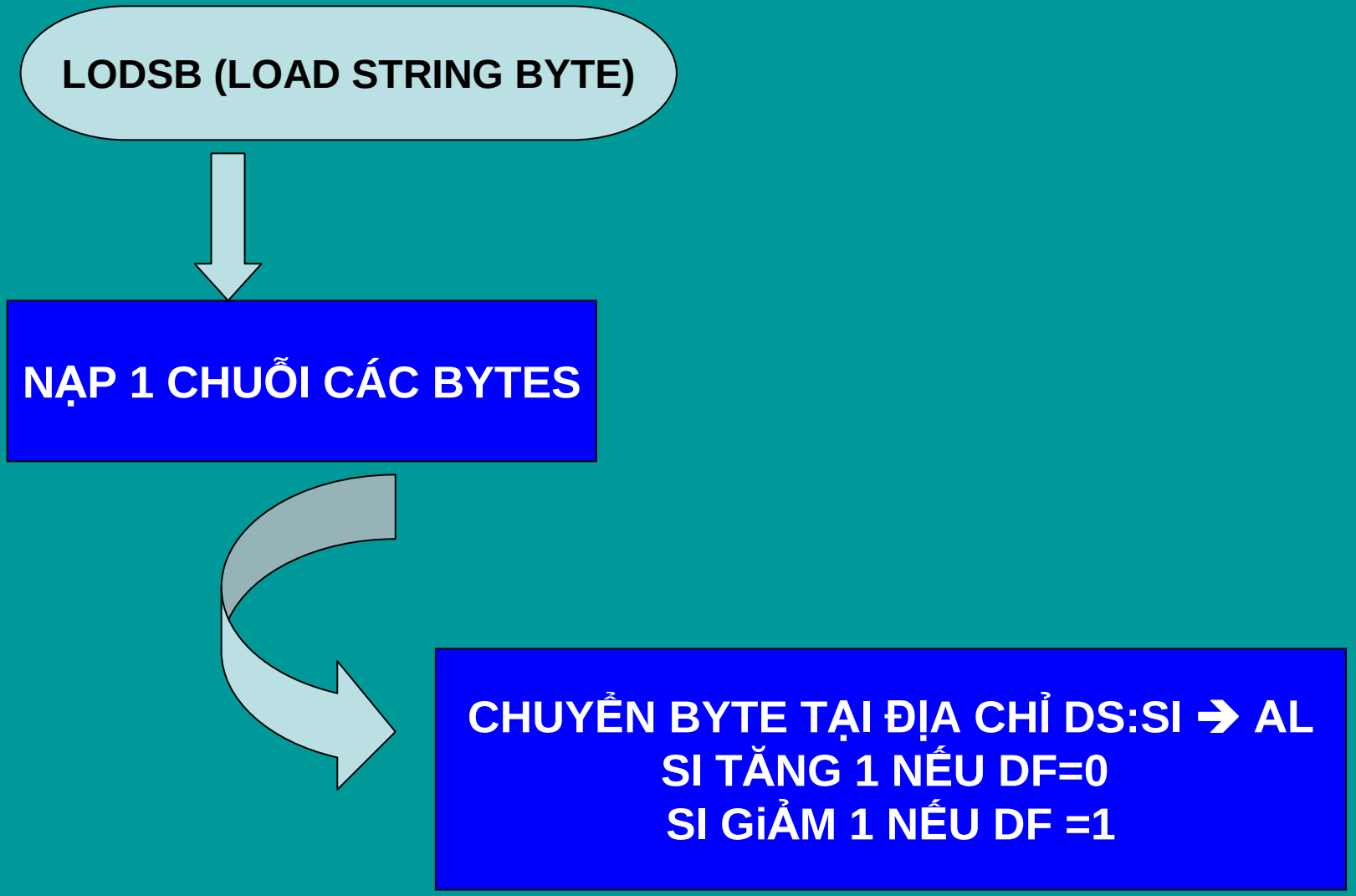
For counter Do

Nạp chuỗi cần hiển thị vào AL

Chuyển vào DL

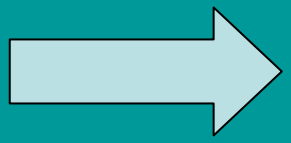
Hiển thị ký tự

EndFor

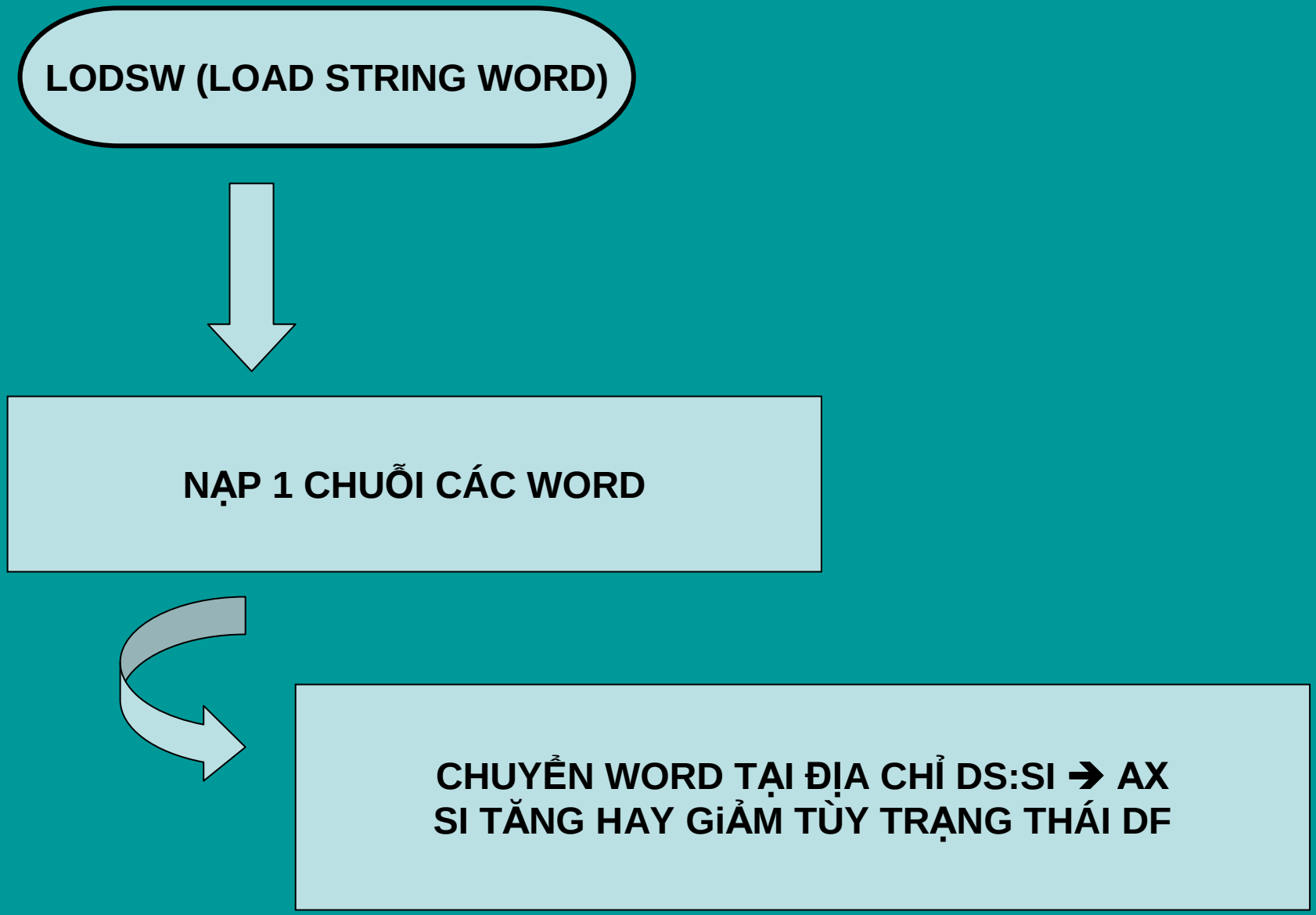

LODSB (LOAD STRING BYTE)
NẠP 1 CHUỖI CÁC BYTES
CHUYỂN BYTE TẠI ĐỊA CHỈ DS:SI ➔ AL
SI TĂNG 1 NẾU DF=0
SI GiẢM 1 NẾU DF =1

THÍ DỤ

```
STRING1 DB ‘ABC’
MOV AX,@DATA
MOV DS,AX
LEA SI, STRING1
CLD
LODSB
LODSB
…….
```



NẠP BYTE THỨ 1 VÀ THỨ 2 ➔ AL


LODSW (LOAD STRING WORD)
NẠP 1 CHUỖI CÁC WORD
CHUYỂN WORD TẠI ĐỊA CHỈ DS:SI ➔ AX
SI TĂNG HAY GiẢM TÙY TRẠNG THÁI DF

THÍ DỤ

Hiển thị chuỗi nhập

```
DISPSTR PROC
PUSH AX
PUSH BX
PUSH CX
PUSH DX
PUSH SI
MOV CX, BX
JCXZ EXIT
CLD
MOV AH,2
LAP :
LODSB
MOV DL, AL
INT 21H
LOOP LAP
```

```
EXIT :
POP SI
POP DX
POP CX
POP BX
POP AX
RET
DISPSTR ENDP
```

CHƯƠNG TRÌNH HÒAN CHỈNH

Viết chương trình nhập 1 chuỗi ký tự tối đa 80 ký tự, hiển thị 15 ký tự của chuỗi đã nhập ở dòng kế.

```
.MODEL SMALL
.STACK 100H
.DATA
STRING1 DB 80 DUP(0)
XDONG DB 0DH,0AH,'$'
.CODE
MAIN PROC
MOV AX,@DATA
MOV DS,AX

MOV ES,AX
LEA DI, STRING1
CALL READSTR
LEA DX,XDONG
MOV AH,9
INT 21H
```

```
LEA SI, STRING1
MOV BX, 15
CALL DISPSTR
MOV AX,4C00H
INT 21H
MAIN ENDP
; READSTR PROC
   ……..
; DISPSTR PROC
……..
END MAIN
```

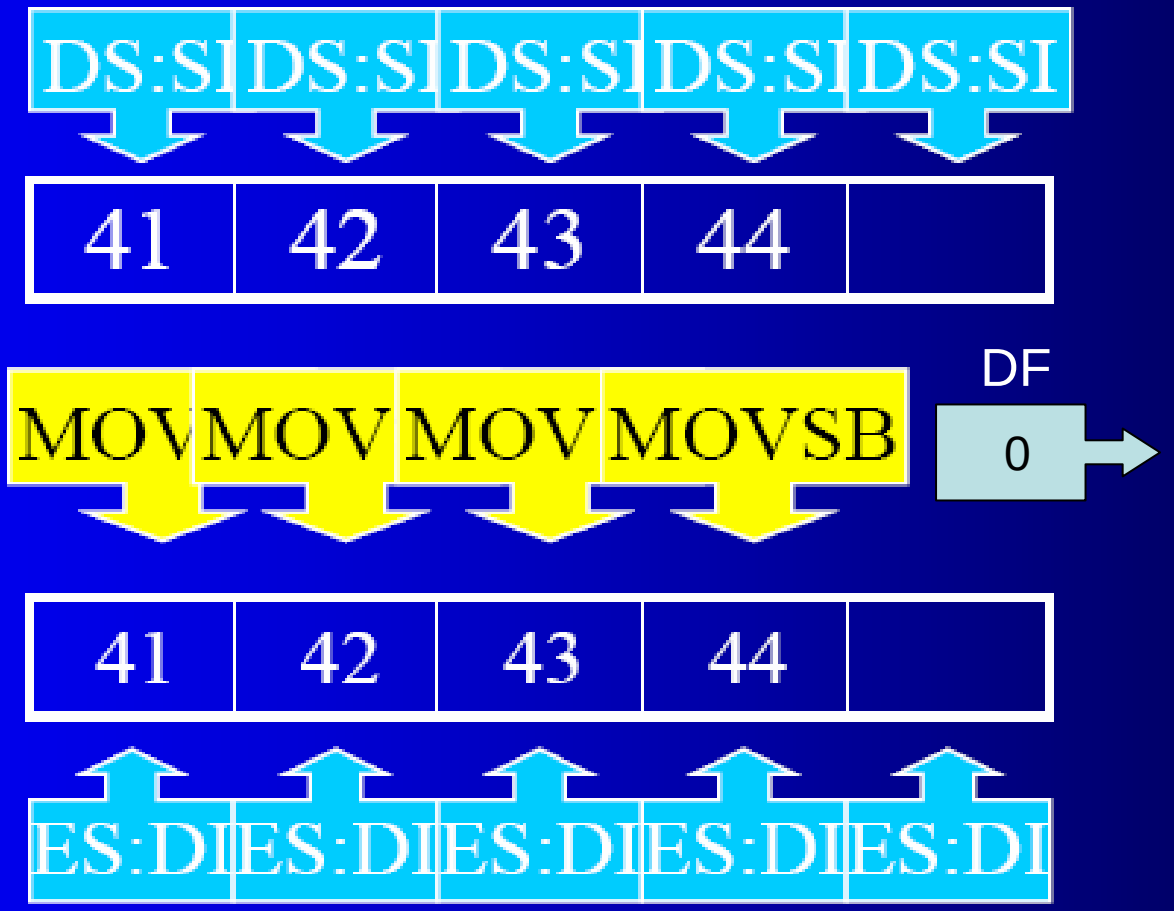
CÁC THAO TÁC XỬ LÝ CHUỖI

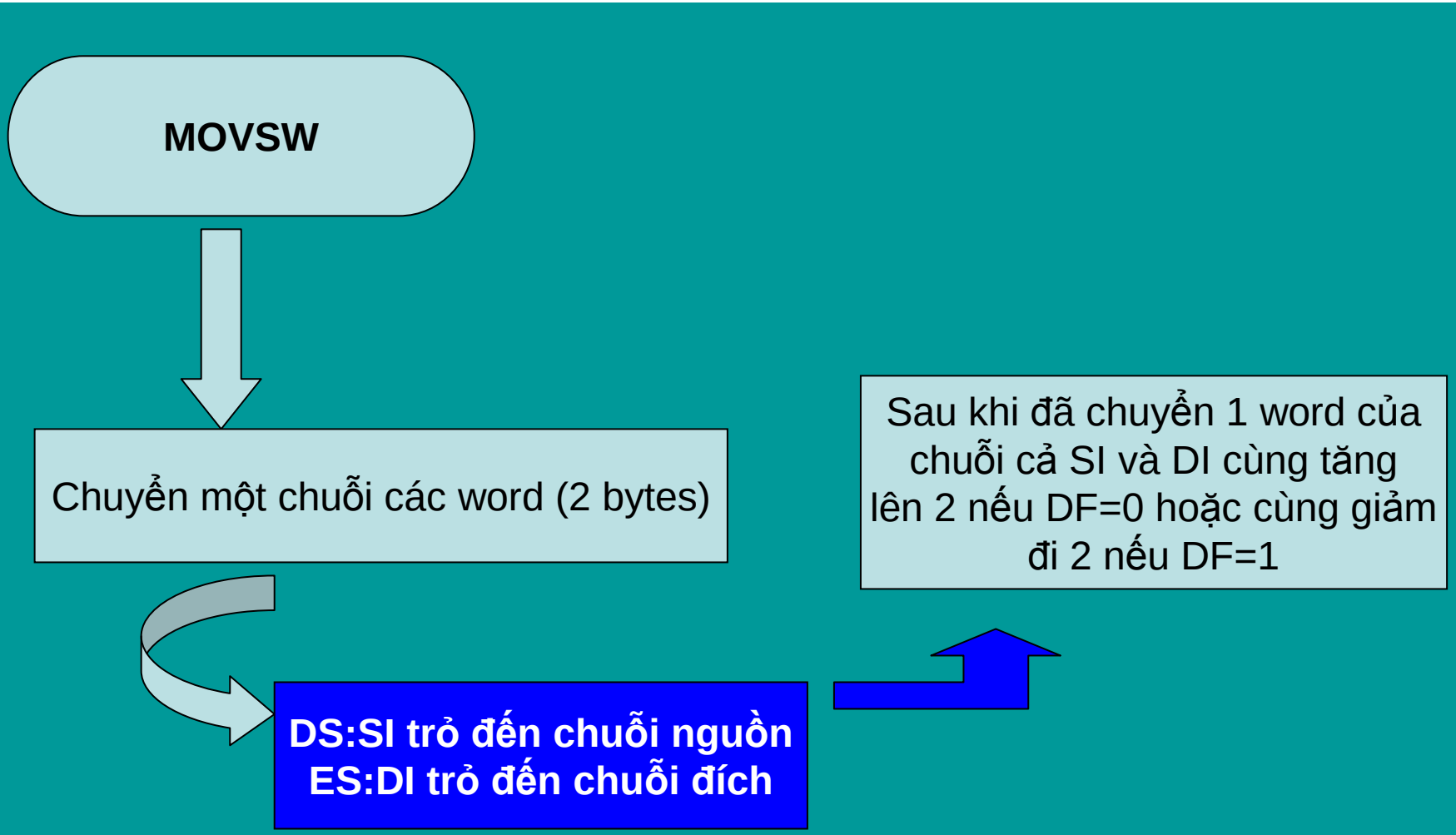
Chuyển một BYTE : MOVSB

chuyển nội dung của byte được định bởi DS:SI đến byte được chỉ bởi ES: DI.
Sau đó SI và DI tự động tăng lên 1 nếu cờ DF = 0
hay giảm 1 nếu DF = 1.

MOVSB chỉ chuyển 1 byte. Vậy cả chuỗi ta làm thế nào ?

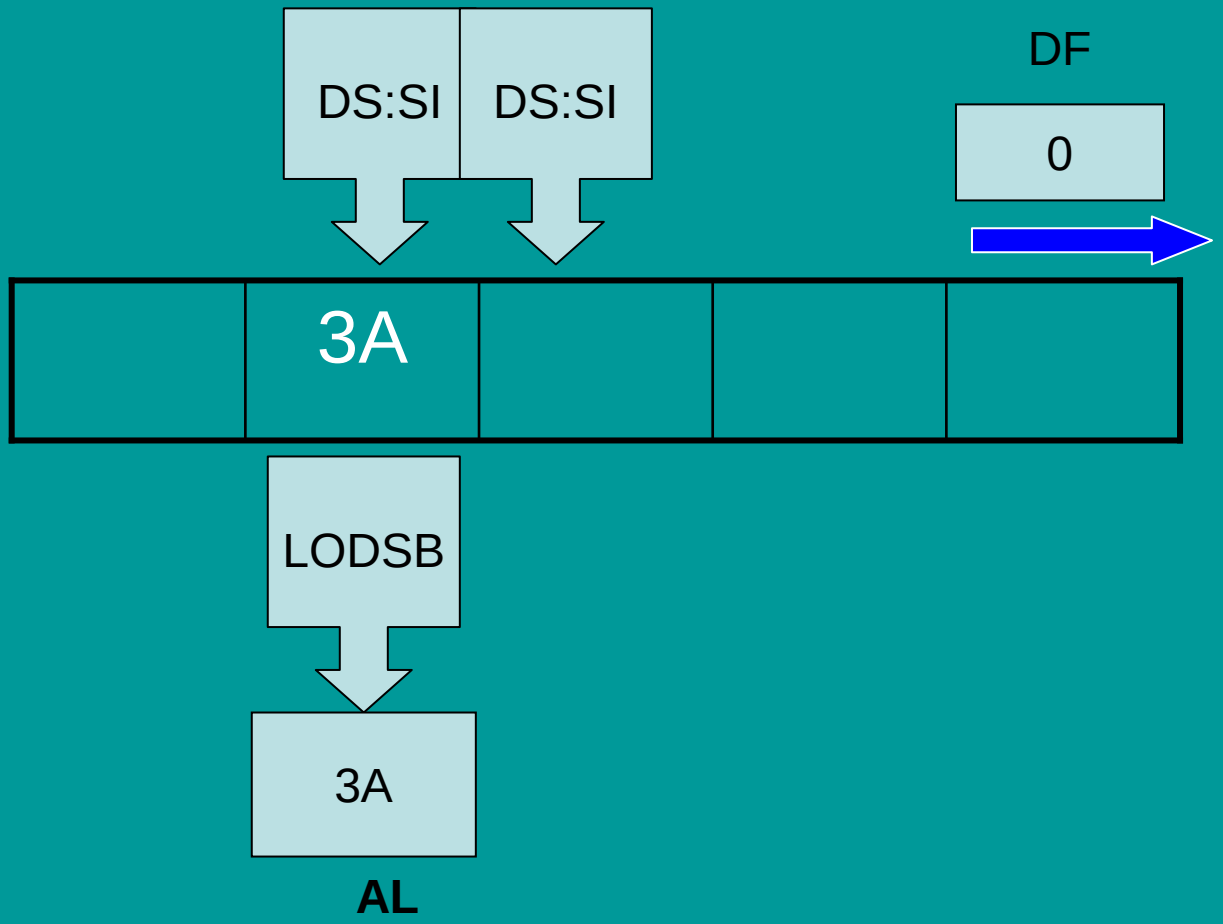
- MOVSB


DS:SI DS:SI DS:SI DS:SI DS:SI
41 42 43 44
MOV MOV MOV MOVSB
DF
0
41 42 43 44
ES:DI ES:DI ES:DI ES:DI ES:DI

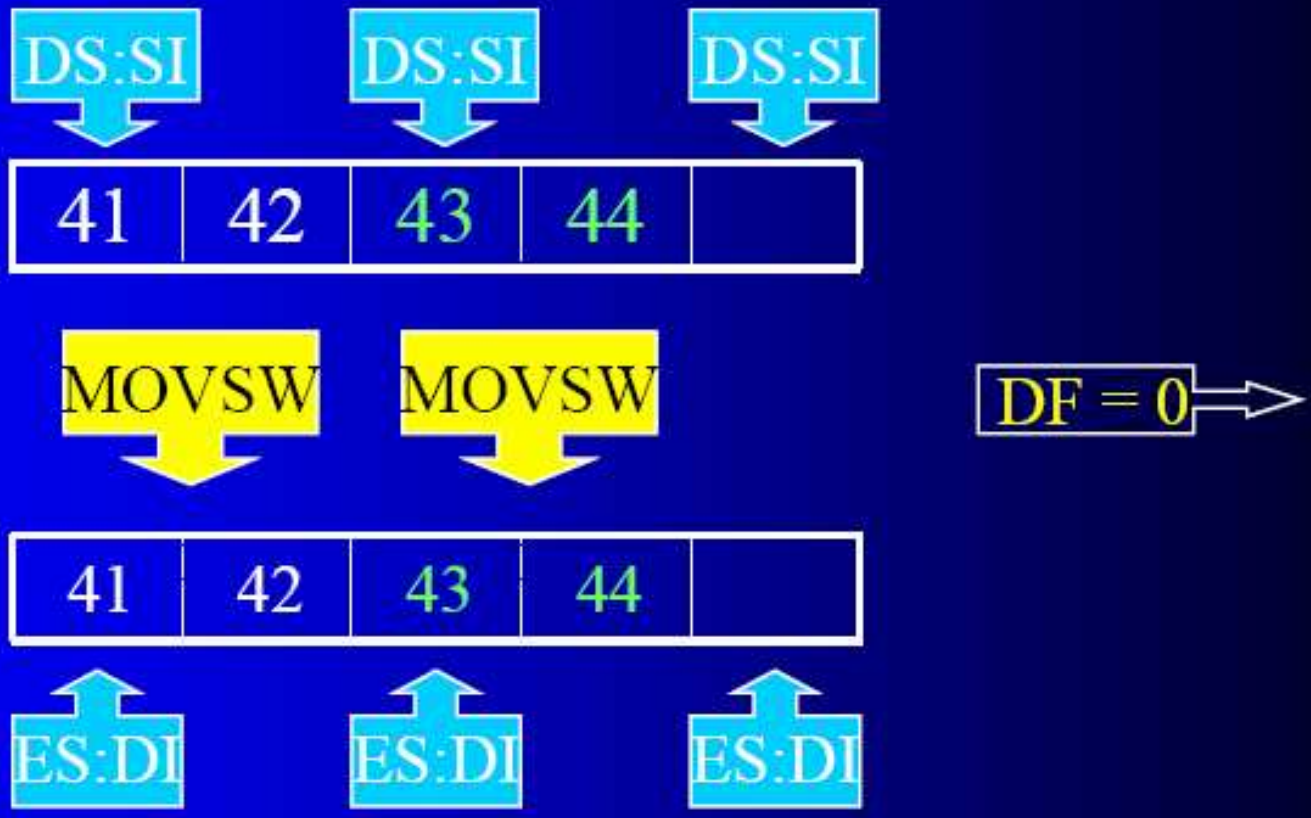

MOVSW
Chuyển một chuỗi các word (2 bytes)
DS:SI trỏ đến chuỗi nguồn
ES:DI trỏ đến chuỗi đích
Sau khi đã chuyển 1 word của chuỗi cả SI và DI cùng tăng lên 2 nếu DF=0 hoặc cùng giảm đi 2 nếu DF=1

LODSB (Load String Byte)

Chuyển byte chỉ bởi DS:SI ➔ AL
tăng SI lên 1 nếu DF=0
giảm SI xuống 1 nếu DF=1

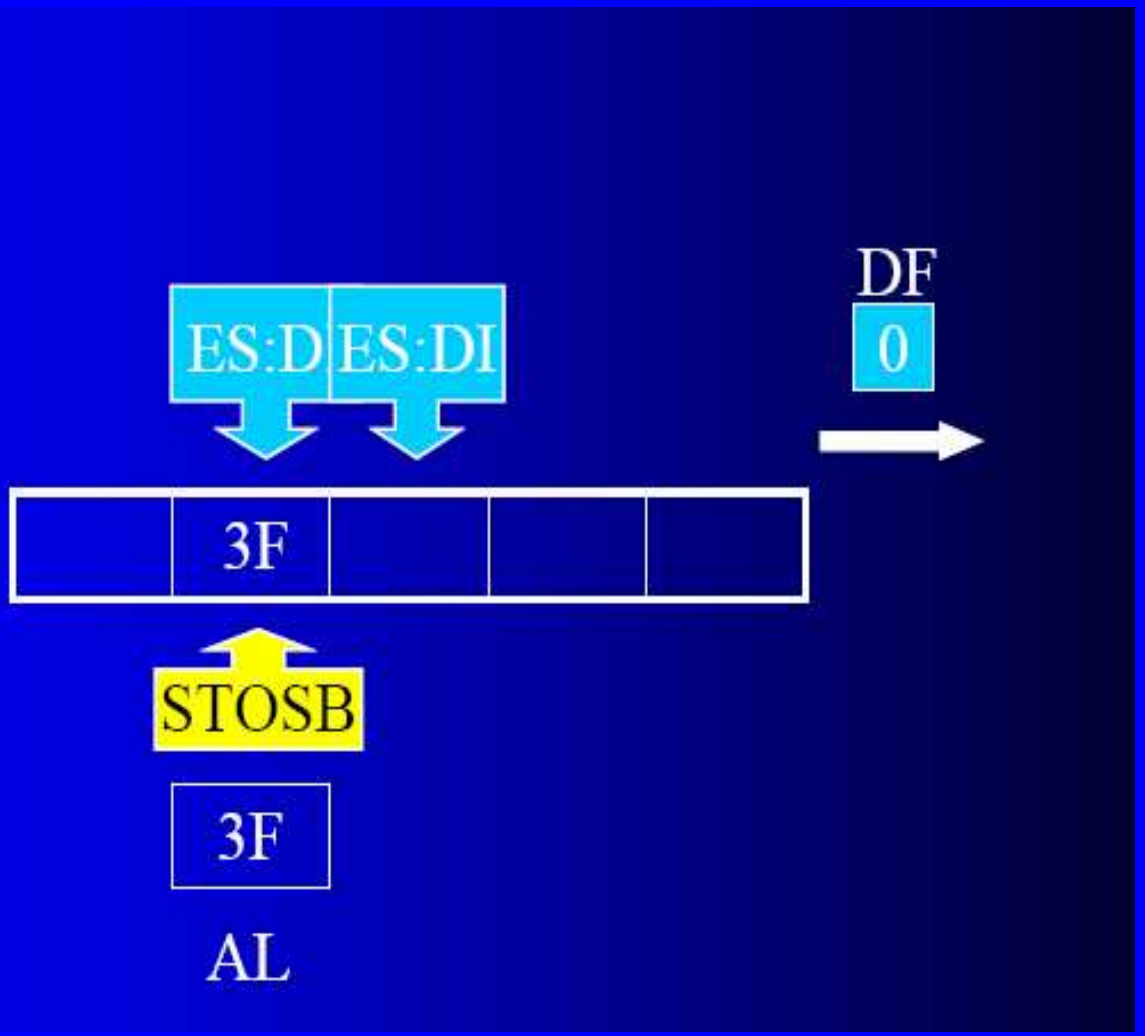

DS:SI
DS:SI
DF
0
3A
LODSB
3A
AL

- MOVSW


DS:SI
DS:SI
DS:SI
41
42
43
44
MOVSW
MOVSW
DF = 0
41
42
43
44
ES:DI
ES:DI
ES:DI

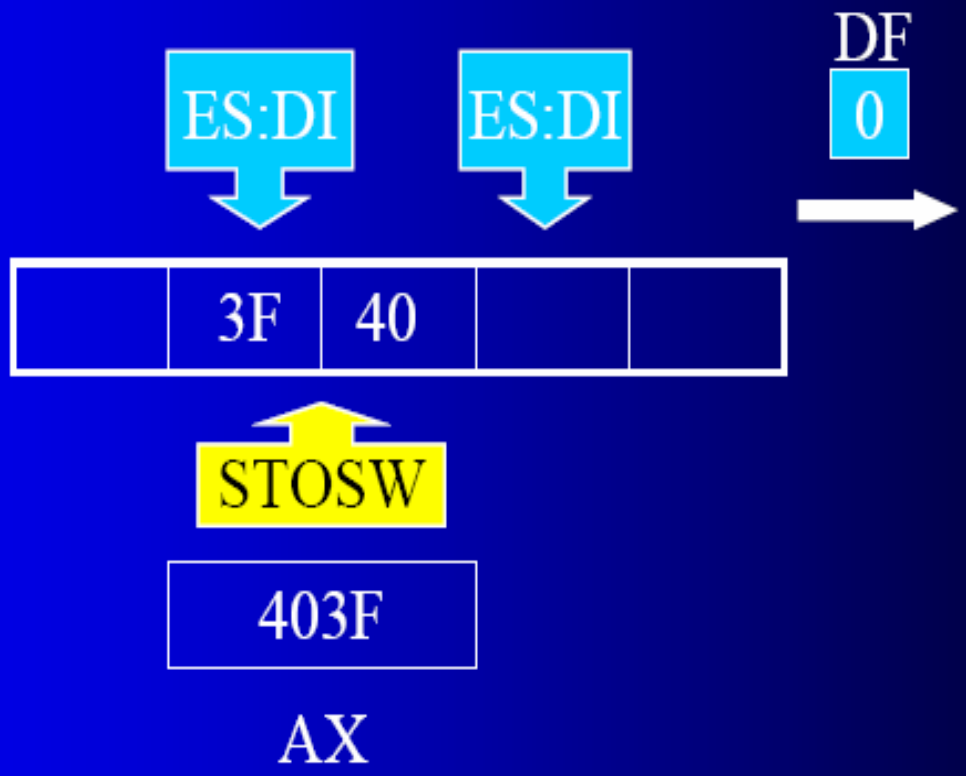
STOSB (LƯU CHUỖI BYTE)

- STOSB

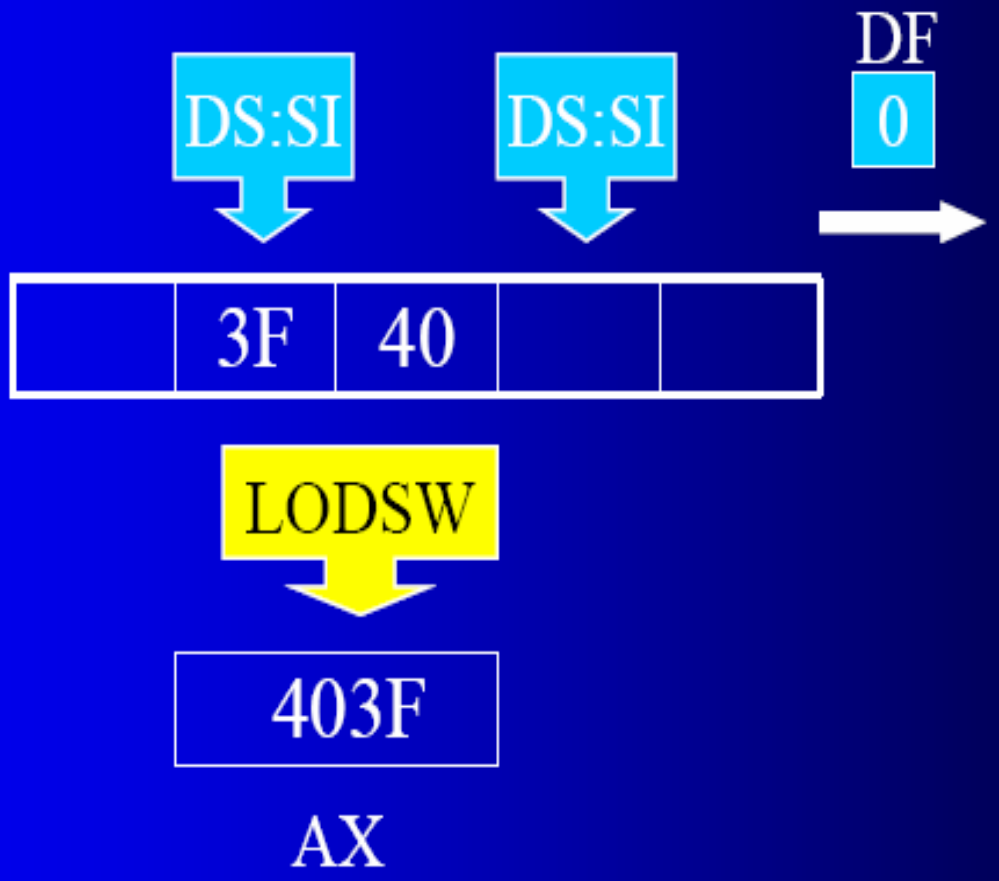


STOSW (LƯU CHUỖI WORD)

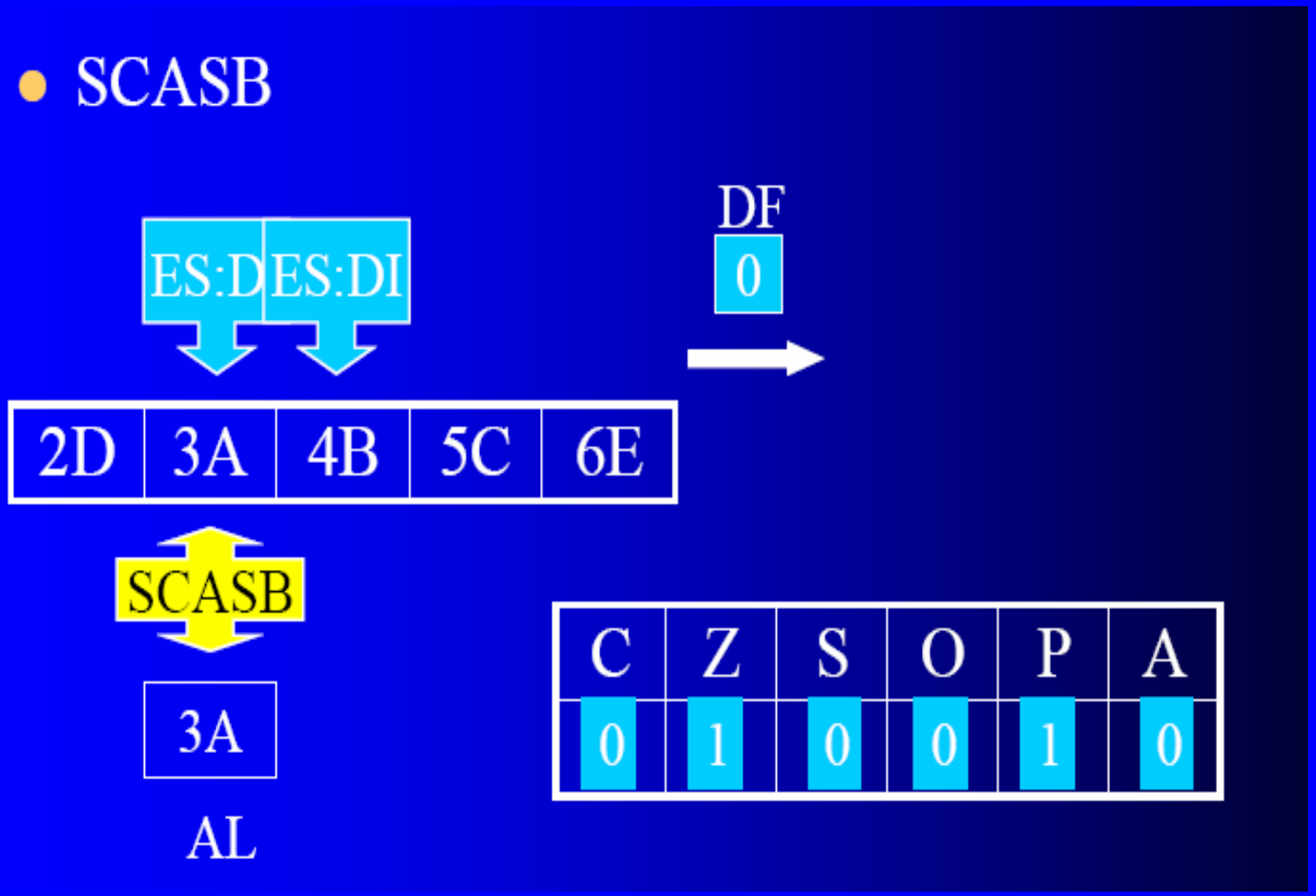
- STOSW



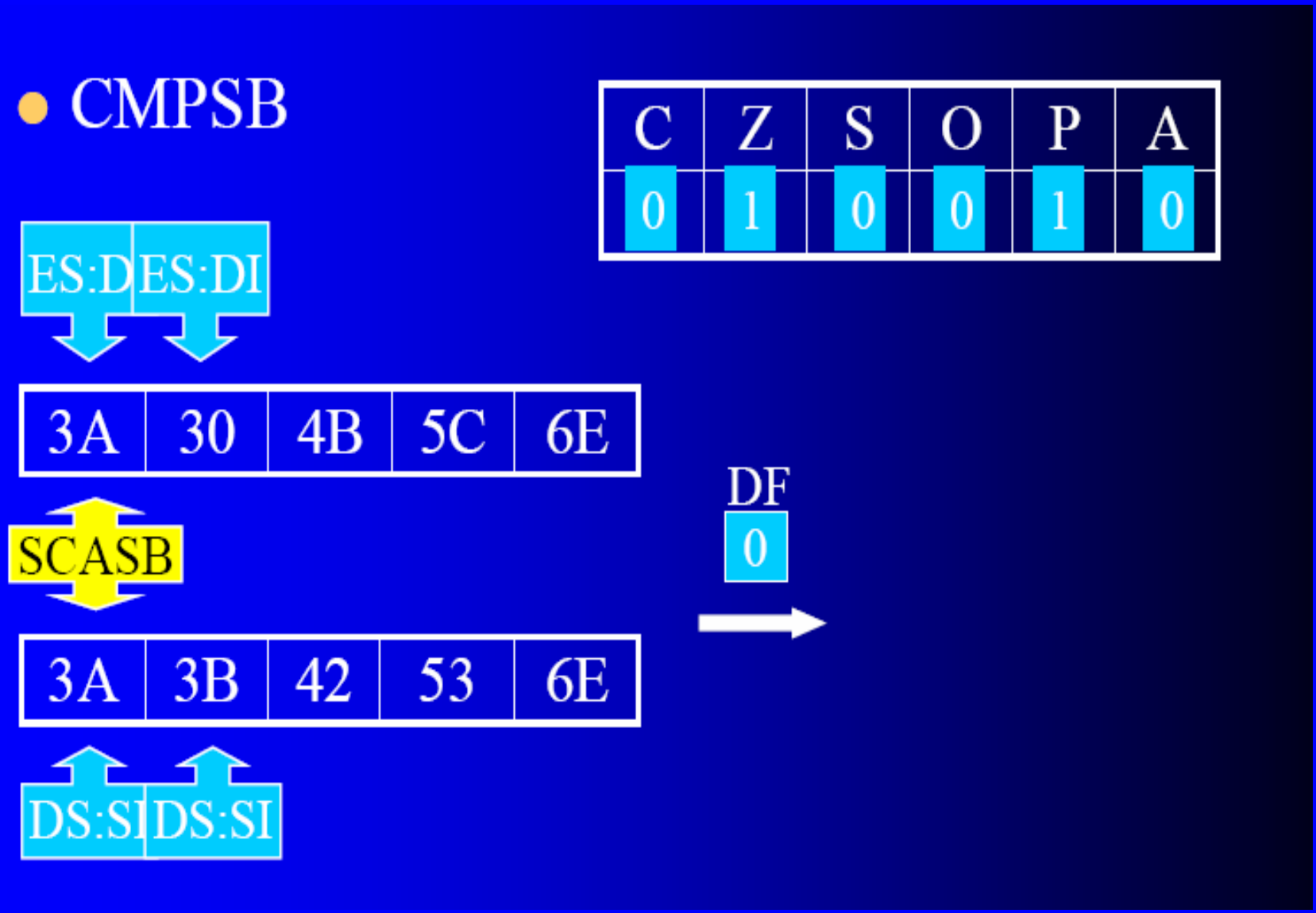
- LODSW



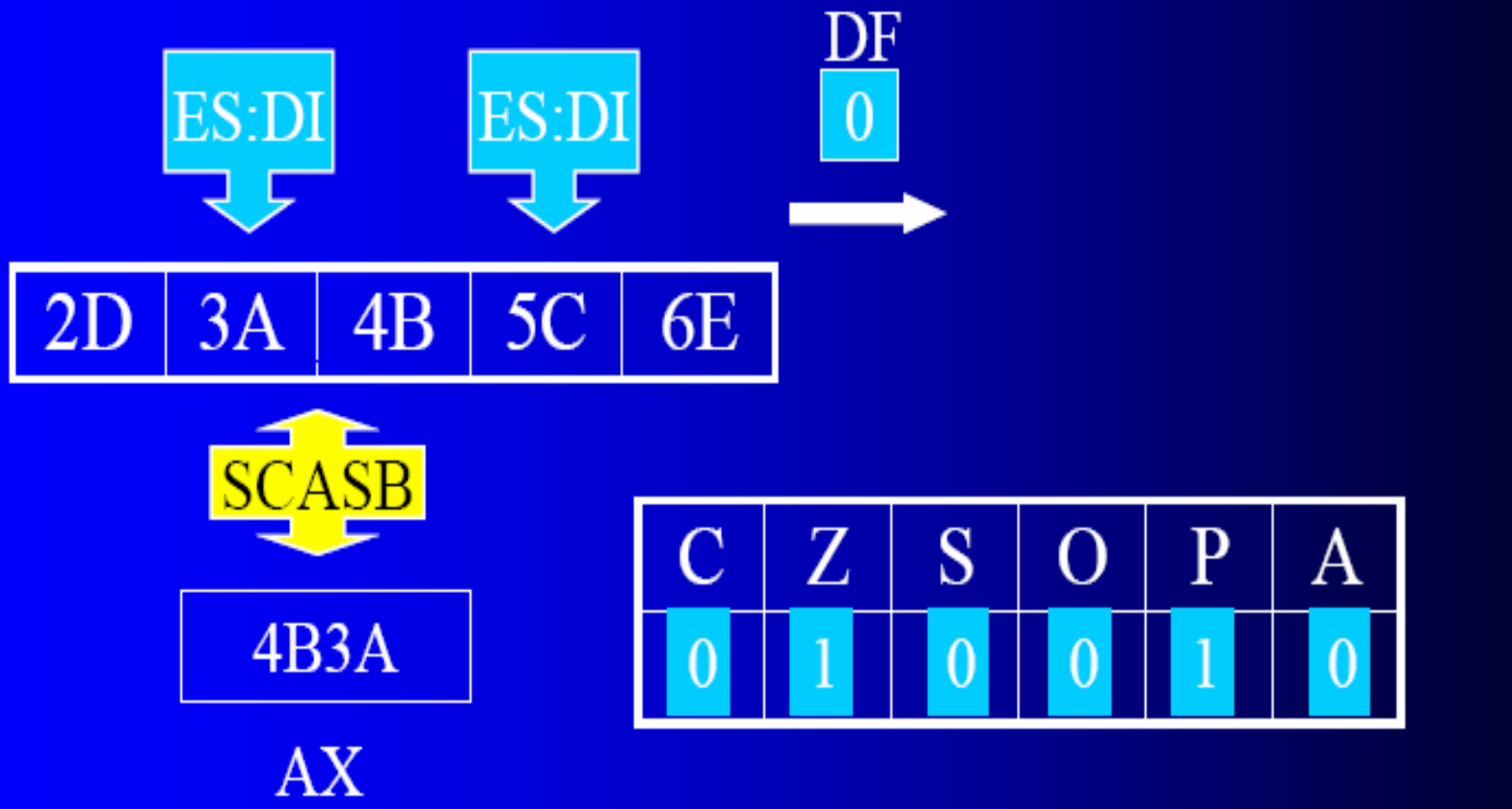
- SCASB



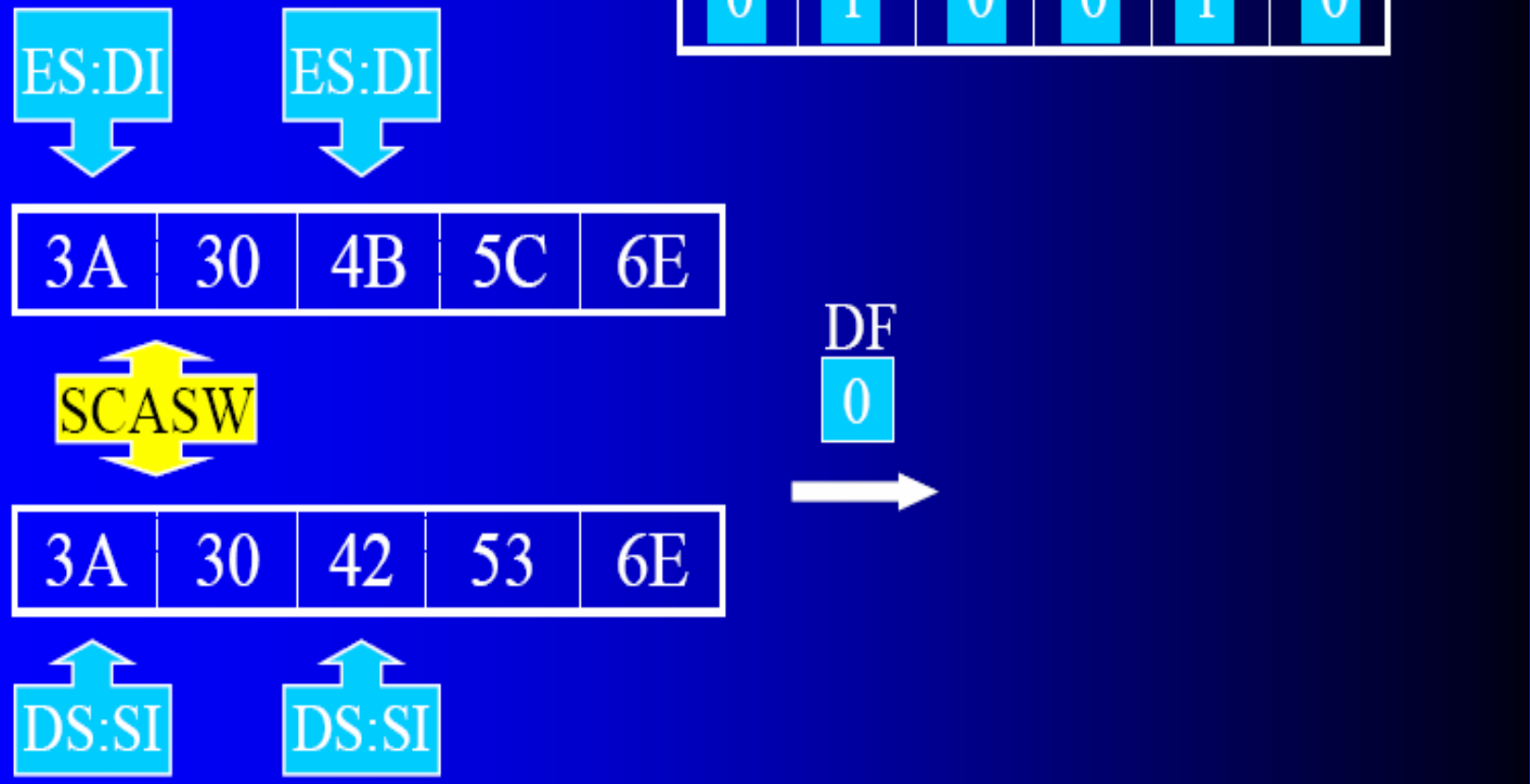
- CMPSB

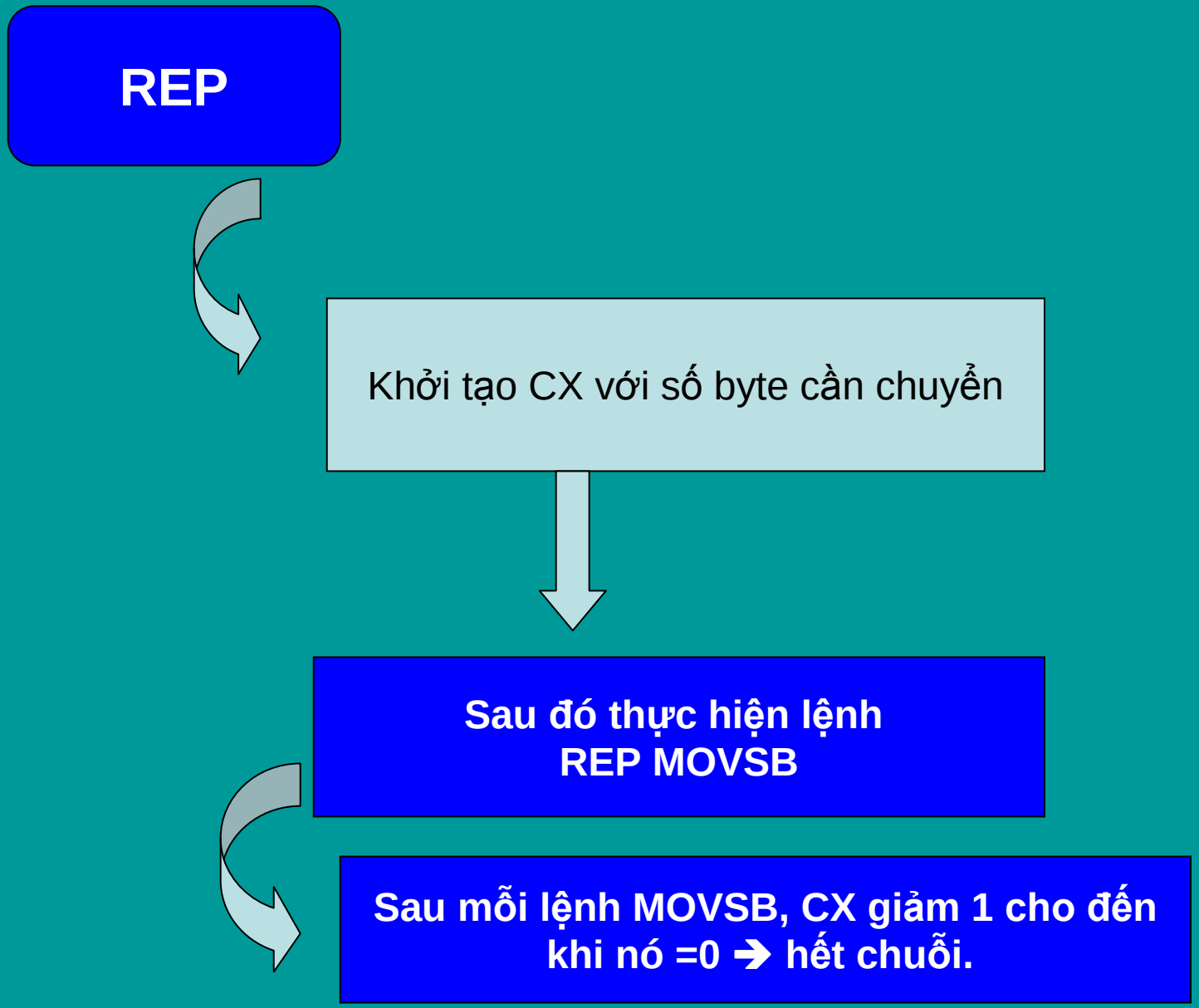

C Z S O P A
0 1 0 0 1 0
ES:D ES:DI
3A 30 4B 5C 6E
SCASB
DF
0
3A 3B 42 53 6E
DS:SI DS:SI

- SCASW



- CMPSW


C Z S O P A
0 1 0 0 1 0
ES:DI
ES:DI
3A 30 4B 5C 6E
SCASW
DF
0
3A 30 42 53 6E
DS:SI
DS:SI


REP
Khởi tạo CX với số byte cần chuyển
Sau đó thực hiện lệnh
REP MOVSB
Sau mỗi lệnh MOVSB, CX giảm 1 cho đến
khi nó =0 ➔ hết chuỗi.

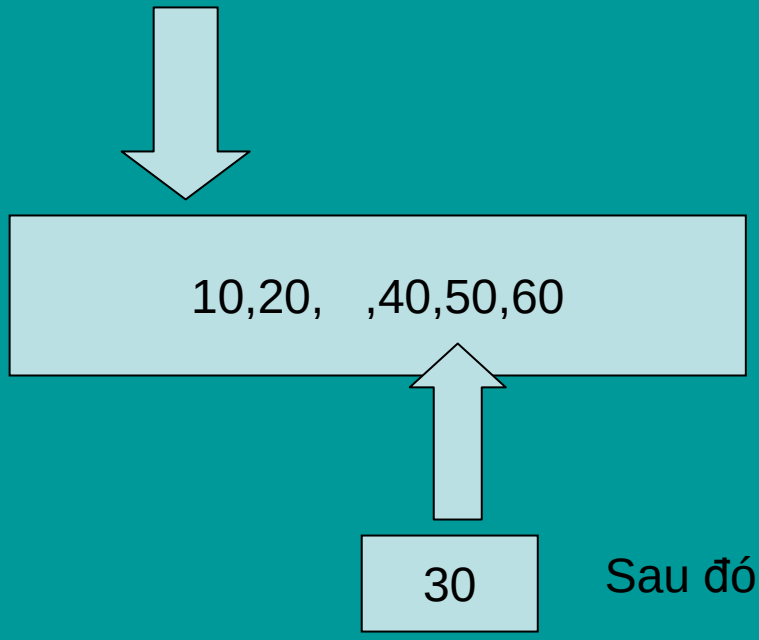
THÍ DỤ MINH HỌA

```
.DATA
STRING1 DB ‘HELLO’
STRING2 DB 5 DUP(?)
…..
CLD
LEA SI, STRING1
LEA DI, STRING2
MOV CX, 5
REP MOVSB
……..
```

Bài tập :
Viết đoạn chương trình chép chuỗi STRING1 ở thí dụ trước vào chuỗi STRING2 nhưng theo thứ tự ngược lại.

THÍ DỤ MINH HỌA

Cho mảng sau
ARR DW 10,20,40,50,60,?
Viết các lệnh để chèn 30 vào giữa 20 và 40 (giả sử rằng DS và ES đã chứa địa chỉ đoạn dữ liệu)



Dời 40,50,60 ra sau 1 vị trí

Sau đó chèn 30 vào

```
STD
LEA SI, ARR+8H
LEA DI, ARR+AH
MOV CX, 3
REP MOVSW
MOV WORD PTR[DI],30
```

MẢNG 1 CHIỀU

Một dãy các phần tử có cùng kiểu dữ liệu, có cùng 1 tên gọi.

Khai báo

MKT DB ‘abcdef’ ; mảng ký tự

MNB Dw 10h,20h,30h,40h,50h,60h ; mảng số

ArrA DB 100 DUP(0) ; khai báo mảng có 100 phần tử có giá trị khởi tạo bằng 0.

Các chương trình con

READSTRING - (Đọc tập tin bàn phím)
Vào : DS:DX = địa chỉ đệm nhận;
CX = số ký tự nhận tối đa.
Ra : DS:DX = địa chỉ chuỗi ASCIIZ.

STRLEN - Lấy chiều dài chuỗi ASCIIZ.
Vào : ES:DI = địa chỉ chuỗi ASCIIZ.
Ra : AX = chiều dài chuỗi không kể số 0.

WRITESTRING - Xuất một chuỗi ra màn hình.
Vào : DS:DX= địa chỉ chuỗi ASCIIZ.
Ra : Không.

STRCOPY - Chép chuỗi nguồn sang chuỗi đích.
Vào : DS:SI = địa chỉ chuỗi nguồn.
ES:DI = địa chỉ chuỗi đích.
Ra : Không.

Các chương trình con (tt)

STRCOMP - So sánh chuỗi và lập cờ.
Vào : DS:SI = địa chỉ chuỗi nguồn.
ES:DI = địa chỉ chuỗi đích.
Ra : thay đổi cờ.
CY = 1 : chuỗi nguồn < chuỗi đích.
ZF = 1 : chuỗi nguồn = chuỗi đích.

STRCHR - tìm ký tự trong chuỗi.
Vào : ES:DI = địa chỉ chuỗi.
AL = ký tự cần tìm.
Ra : CY = 0 : tìm thấy ký tự.
ES:DI = địa chỉ vị trí xuất hiện đầu tiên.
CY = 1 : không tìm thấy ký tự trong chuỗi.

STRSTR - tìm chuỗi trong chuỗi.
Vào : DS:SI = địa chỉ chuỗi nguồn.
DX = chiều dài chuỗi nguồn.
ES:DI = địa chỉ chuỗi đích.
BX = chiều dài chuỗi đích.
Ra : CY = 0 : tìm thấy
ES:DI = địa chỉ vị trí xuất hiện đầu tiên.
CY = 1 : không tìm thấy.

BÀI TẬP

Bài 1 : Viết chương trình nhập 1 số từ 1-12, in ra tên tháng tương ứng.

Bài 2 : Viết chương trình nhập 1 số từ 1-7, in ra tên thứ tương ứng.

MỘT SỐ BÀI TẬP MINH HỌA LẬP TRÌNH XỬ LÝ CHUỖI

Nhập 1 chuỗi dài tối đa 255 ký tự từ bàn phím. Cho phép dùng phím BackSpace để sửa khi nhập sai và kết thúc nhập khi gỏ phím Enter.

Hướng dẫn :

Dùng hàm 0AH INT 21H để nhập chuỗi
DS:DX địa chỉ của buffer đệm lưu chuỗi.
Byte 0 : số byte tối đa có thể nhập.
Byte 1 : chứa giá trị 0
Byte 2 trở đi : để trống (lưu các ký tự sẽ nhập)

Để nhập 1 chuỗi ký tự vào Buffer đệm ta khai báo như sau :

```
.DATA
BUFFERN DB 80,0,80 DUP(?)
```

B1. Viết chương trình nhập vào 1 từ, sau đó in từng ký tự trong từ theo chiều dọc.

Thí dụ Nhập CONG

Xuất : C
O
N
G

B2. Viết chương trình nhập vào 1 chuỗi, sau đó đổi tất cả chuỗi thành chữ hoa và in chuỗi ra màn hình ở dòng kế.

B3.Viết chương trình nhập hai chuỗi ký tự , kiểm tra xem chuỗi thứ hai có xuất hiện trong chuỗi thứ nhất hay không.

Ví dụ : Nhập chuỗi thứ nhất : computer information

Nhập chuỗi thứ hai : compute

Xuất: Chuỗi thứ hai có xuất hiện trong chuỗi thứ nhất.

B4. Viết chương trình nhập 1 chuỗi ký tự viết hoa các ký tự nguyên âm, viết thường các ký tự phụ âm.

Ví dụ : Nhập chuỗi : “aBcdE”

Xuất chuỗi: “AbCdE”

B5. Viết chương trình nhập vào 2 chuỗi ký tự s1, s2 và 1 số nguyên dương n. Chèn chuỗi s2 vào chuỗi s1 ở vị trí ký tự thứ n trong chuỗi s1 .

Ví dụ : Nhập chuỗi s1 : “abcde”

Nhập chuỗi s2 : “fgh”

Nhập n = 3

Xuất kết quả : “abcfghde”

B6. Viết chương trình nhập vào từ bàn phím 1 chuỗi và tính số lần xuất hiện của các nguyên âm (a,e,i,o,u, y), cac phu am, cac khoang trang, trong chuỗi tương ứng.

Ví dụ : Nhập chuỗi : “dai hoc khoa hoc tu nhien thanh pho ho chi minh”

Xuất : Số lần xuất hiện của các nguyên âm là : 14 , phu am la: 24, khoang trang la: 9

B7. Viết chương trình nhập vào từ bàn phím 1 chuỗi gồm các ký tự trong bảng chữ cái. Đếm xem trong chuỗi có bao nhiêu từ.

Ví dụ : Nhập chuỗi : “ hO Chi mINh ”

Xuất : chuỗi gồm có 3 từ

B8. Viết chương trình nhập vào từ bàn phím 4 số . Xuất ra màn hình 4 số đó theo thứ tự tăng dần .

Ví dụ : Nhập : 14 7 26 11

Xuất : 7 11 14 26

B9. Viết chương trình nhập vào từ bàn phím 4 số và sau đó xuất số lớn nhất và nhỏ nhất ra màn hình.

Ví dụ : Nhập : 13 21 1 49

Xuất : Số lớn nhất : 49

Số nhỏ nhất : 1

Viết chương trình nhập vào từ bàn phím chuỗi 1 (chuỗi dài), chuỗi 2 (chuỗi ngắn) và một ký tự. Sau đó, làm các công việc sau :

- Tìm chuỗi 2 trong chuỗi 1 và in ra vị trí xuất hiện đầu tiên của chuỗi 2 trong chuỗi 1 nếu tìm thấy. Ngược lại in ra không tìm thấy.
- Tìm ký tự đã nhập trong chuỗi 1 và in ra vị trí xuất hiện đầu tiên của ký tự nếu tìm thấy.Ngược lại in ra không tìm thấy.
- Thay chuỗi 2 trong chuỗi 1 bằng ký tự (nếu được).

Bài giảng

Cấu trúc máy tính và ghép nối

CHƯƠNG 1: GIỚI THIỆU CHUNG

1.1. Tổng quan về cấu trúc máy tính

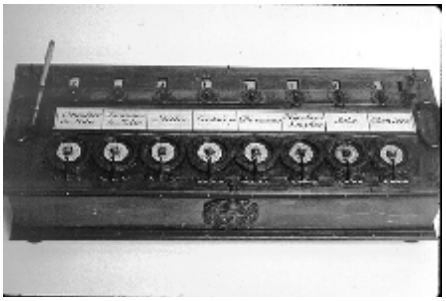
Cấu trúc máy tính là một mảng kiến thức nghiên cứu về cách xử lý của một hệ thống máy tính dưới cách nhìn của một lập trình viên. Cách nhìn này thực tế cũng có nhiều khía cạnh, ví dụ như máy tính có độ rộng dữ liệu khác nhau sẽ có cấu trúc phần cứng và hoạt động khác nhau, hoặc máy tính có hỗ trợ các phép toán nào (cộng, trừ, nhân, chia, hỗ trợ các chương trình con,...)

Trong cấu trúc máy tính xuất hiện khái niệm "mức máy". Ý tưởng cơ bản của nó là trong mỗi một máy tính có nhiều mức khác nhau, từ mức độ cao nhất (người sử dụng có thể chạy chương trình, sử dụng máy tính...) cho đến mức thấp nhất (máy tính chỉ là tập hợp các phần tử là transistor và các dây nối...). Giữa mức máy cao đến thấp còn có các mức máy trung gian. Trước khi thảo luận về các mức của máy tính, chúng ta xem xét lịch sử phát triển của máy tính để có được một quan điểm về cách xây dựng một máy tính

1.2. Lịch sử phát triển của máy tính

Các thiết bị cơ khí được sử dụng để điều khiển các thiết bị phức hợp đã xuất hiện ít nhất từ những năm 1500. Vào thời điểm đó, người ta sử dụng trục quay cố định để làm những hộp nhạc. Và hộp nhạc đó chỉ có thể hoạt động đơn giản là lặp đi lặp lại một giai điệu nhất định

Blaisa Pascal (1623 - 1662) đã phát triển một máy tính toán cơ khí để giúp người cha trong công việc tính thuế. Máy tính Pascal (Pascaline) bao gồm 8 con số được kết nối trên một trống xoay. Một số sẽ tăng 1 giá trị (xoay một góc nhất định) khi số thấp hơn quay đủ một vòng. Một số máy tính Pascal khác được ông xây dựng năm 1642 vẫn còn đến tận ngày nay.



Hình 1.1. Máy tính Pascal

Đến những năm 1800, một người đã phát triển các thành phần cơ khí của máy tính Pascal thành một máy mà chúng ta nhận thấy rằng đó là thành phần cơ bản của một máy tính số. Người đó là Charles Babbage

Charles Babbage được coi là "ông nội" của máy tính hơn là cha đẻ của máy tính, bởi vì ông chưa bao giờ xây dựng một máy tính mà ông thiết kế. Babbage sống tại nước Anh, tại thời điểm đó, người ta thường sử dụng bàn tính để tính toán. Để tránh việc tính toán có nhiều lỗi, Babbage đã tạo ra một máy tính hoạt động bằng cách quay các bánh răng. Máy của ông thậm chí còn có khả năng tạo ra những đĩa dữ liệu có thể sử dụng ngay trong máy in, do đó tránh được lỗi do sắp chữ trong khi thiết kế bản in. Máy tính của Babbage đã có chức năng đọc dữ liệu, lưu trữ và biểu diễn dữ liệu. Các chức năng cơ bản của nó gần giống với các chức năng của máy tính hiện đại. Sự thành công của các máy tính Babbage đã giúp ông giành được sự hỗ trợ của chính phủ trong việc thiết kế những máy phân tích cỡ lớn, những máy có ý nghĩa rất lớn trong việc lập trình sử dụng các thẻ đục lỗ theo mô hình Jacquard

Những máy phân tích của Babbage đã thiết kế nhưng đã không được xây dựng bởi Babbage vì tại thời điểm đó, những máy cơ khí không đạt được độ chính xác theo thiết kế. Một phiên bản khác của máy tính Babbage cuối cùng cũng đã được làm ra tại Bảo tàng khoa học London năm 1991, và tồn tại cho đến tận ngày nay

Trải qua hàng thế kỷ cho đến Thế chiến thứ II, xuất hiện một động lực lớn cho việc phát triển máy tính. Tại Anh, tàu ngầm của Đức đã bị thiệt hại nặng nề trong khi vận chuyển. Chiếc tầu ngầm đã nhận và giải mã các tín hiệu từ các tầu khác của Đức và đã bị điều khiển sai. Việc mã hóa tín hiệu của Đức được tạo ra bằng cách sử dụng một đoạn mã được tạo ra bởi một chiếc máy do Siemens AG tạo ra dưới cái tên ENIGMA

Quá trình tạo ra các mã thông tin đã được biết đến từ lâu, thậm chí Tổng thống Hoa Kỳ Thomas Jefferson (1743 - 1826) đã thiết kế một máy được coi là tiền thân của ENIGMA, mặc dù ông đã không chế tạo nó. Quá trình giải mã diễn ra phức tạp hơn rất nhiều. Nó là động lực để Alan Turning (1912 - 1954) và một số nhà khoa học nước Anh khác tạo ra máy phá mã. Trong suốt Thế chiến II, Turning là người giải mã hàng đầu ở Anh và là một trong những người đã biến khoa học mật mã từ chức năng là dịch các ngôn ngữ cổ đại thành một khoa học tính toán

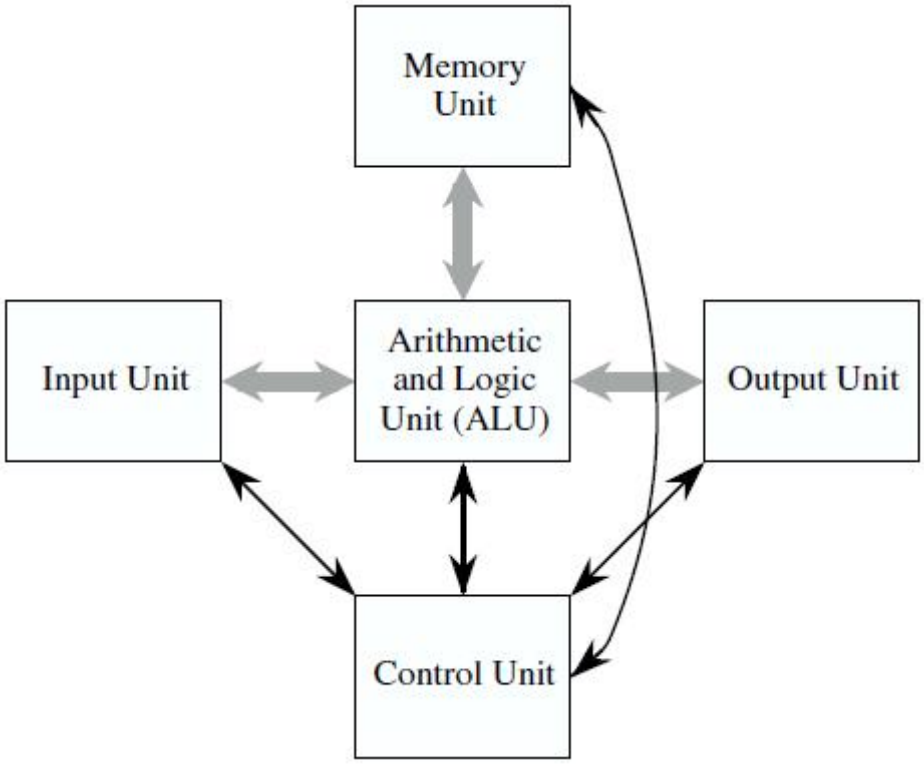
Colossus là máy giải mã đầu tiên được chế tạo tại Bletchley Park, Anh quốc, nơi Turning làm việc. Những ống chân không được sử dụng để lưu trữ nội dung giống như trên giấy và được đưa vào chiếc máy, việc tính toán diễn ra sau đó được thực hiện trên những ống chân không đó cho đến khi những ống khác lại được tiếp tục đưa vào máy. Việc lập trình được diễn ra trên những bảng cắm các ống chân không.

Cùng thời với Turning, J.Presper Eckert và John Mauchly đã chế tạo ra một chiếc máy dùng để tính toán quỹ đạo của đường đạn sử dụng cho quân đội Hoa kỳ. Kết quả sự nỗ lực của Eckert và Mauchly là chiếc máy điện tử số tích hợp máy tính (Electronic Numerical Intergrator And Computer - ENIAC). Máy ENIAC bao gồm 18.000 ống chân không tạo nên phần tính toán của máy. Việc lập trình và nhập liệu được thực hiện bằng cách thay đổi trạng thái các công tắc và các đường cable. Máy không có khả năng lưu trữ dữ liệu hay chương trình nhưng nó không phải là hạn chế lớn nhất của máy bởi vì chức năng của máy EMIAC là tính toán quỹ đạo của đường đạn. Thậm chí chiếc máy này không hoạt động được cho đến tận năm 1946, sau chiến tranh Thế giới II nhưng nó được coi là một thành công và đã được sử dụng trong suốt 9 năm

Sau thành công của máy ENIAC, Eckert và Mauchly (làm việc tại Đại học Pennsylvania) được John Von Neumann (1903 - 1957) mời cộng tác làm việc tại Viện nghiên cứu cao cấp tại Princeton. Cùng với nhau, họ đã thiết kế một máy tính có khả năng lưu trữ được gọi là EDVAC. Mâu thuẫn nảy sinh, hai nhóm người tại Đại học Pennsylvania và Princeton chia tách nhau. Tuy nhiên, mô hình máy tính mà họ thiết kế phát triển mạnh mẽ, hình thành nên máy tính EDSAC được tạo ra bởi Maurice Wilkes tại Đại học Cambridge năm 1947

1.3. Mô hình máy tính Von Neumann

Máy tính kỹ thuật số thông thường được chế tạo dựa trên mô hình được cho là của Von Neumann. Mô hình Von Neumann bao gồm 5 thành phần chính được chỉ ra trên hình 1.2. Khối nhập liệu sẽ đưa lệnh và dữ liệu vào hệ thống và lưu trữ tuần tự ở khối bộ nhớ chính. Lệnh và dữ liệu sẽ được thực thị tại khối Số học và logic (ALU) dưới sự điều khiển bởi khối điều khiển. Kết quả sẽ được đưa ra khối hiển thị dữ liệu. Khối ALU và bộ điều khiển thường được gọi chung là bộ xử lý trung tâm (CPU). Hầu hết các máy tính thông thường có thể phân chia thành các khối cơ bản như trên

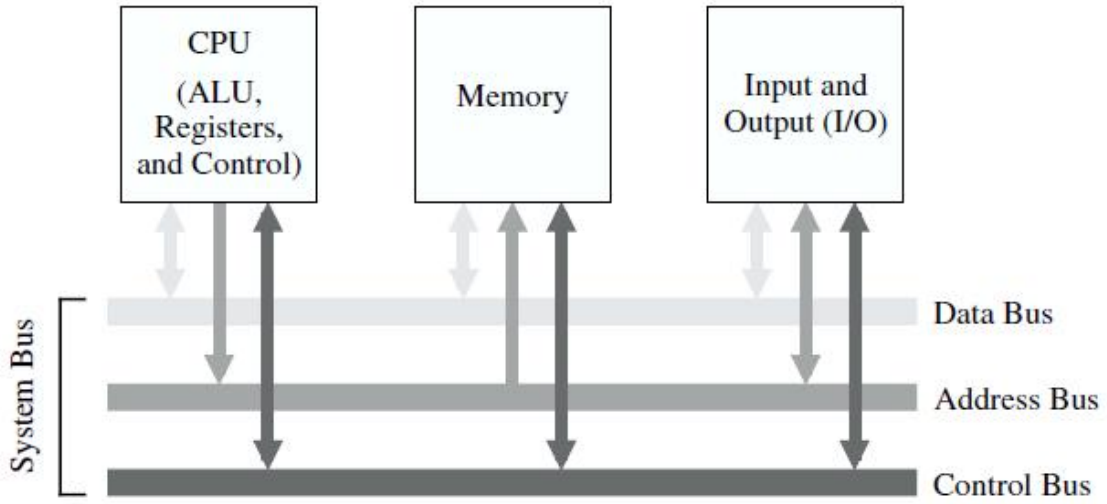


Hình 1.2. Mô hình máy tính Von Neumann

Chương trình lưu trữ là nội dung quan trọng nhất của mô hình Von Neumann. Chương trình được lưu trữ tại bộ nhớ chính cùng với dữ liệu chương trình được xử lý. Việc lưu trữ được thực hiện theo từng cấp độ khác nhau cùng với sự phát triển của công nghệ. Ví dụ như trước kia, chương trình và dữ liệu được lưu trữ dưới dạng thẻ đục lỗ hay băng từ,... Trong máy tính, chương trình hay các lệnh của chương trình được thao tác như thể là dữ liệu. Điều này dẫn đến việc xuất hiện các trình biên dịch và hệ điều hành, và làm cho máy tính trở nên rất linh hoạt

1.4. Mô hình hệ thống bus

Mặc dù kiến trúc Von Neumann được sử dụng rộng rãi trong các máy tính hiện đại, nhưng nó đã được biến đổi. Hình 1.3. thể hiện một mô hình hệ thống bus của một hệ thống máy tính. Mô hình này chia máy tính ra làm 3 khối: CPU, bộ nhớ và các cổng vào ra I/O. Điều tinh tế của mô hình này là đã kết hợp khối ALU và khối điều khiển thành một khối có chức năng duy nhất là CPU. Khối nhập dữ liệu và hiển thị dữ liệu được kết hợp thành khối các cổng ngoại vi



Hình 1.3. Mô hình hệ thống bus

Điều quan trọng nhất của mô hình hệ thống bus là sự kết nối giữa các khối được gọi chung là hệ thống bus, được cấu thành từ các **bus dữ liệu** mang thông tin bằng cách truyền thông, **bus địa chỉ** có chức năng chỉ ra nơi dữ liệu sẽ được chuyển tới và **bus điều khiển** sẽ chỉ ra các khía cạnh thông tin đang được gửi đi và trong những phương thức gì. Tất nhiên hệ thống còn có hệ thống bus công suất là các đường dây cung cấp năng lượng điện cho toàn bộ hệ thống. Hệ thống bus công suất thường không được chỉ ra trong mô hình nhưng sẽ được ngầm hiểu trong mọi mô hình. Một số kiến trúc còn có các hệ thống bus I/O riêng biệt

Về mặt vật lý, các hệ thống bus thực chất là các đường dây được nhóm lại với nhau theo chức năng. Hệ thống bus dữ liệu 32 bit bao gồm 32 đường dây riêng biệt, mỗi dây sẽ truyền tải 1 bit dữ liệu (địa chỉ hoặc các thông tin điều khiển). Trong cách hiểu này, bus hệ thống là các nhóm bus được phân chia theo chức năng

Bus dữ liệu có chức năng chuyển dữ liệu giữa các khối. Một số hệ thống có hệ thống bus dữ liệu riêng để chuyển dữ liệu tương tác với khối CPU được gọi là các hệ thống bus dữ liệu và và bus dữ liệu ra. Thông thường bus dữ liệu và và ra được thực hiện trên cùng một hệ thống dây dẫn và tại cùng một thời điểm sẽ chỉ truyền dữ liệu theo một hướng

Vì hệ thống bus được sử dụng để kết nối giữa các khối, do đó các khối chức năng cần có đặc điểm nhận dạng riêng, chính là địa chỉ. Trong một số máy tính, tất cả các địa chỉ được giả định là địa chỉ ô nhớ, nhưng thực tế sẽ bao gồm cả

địa chỉ ô nhớ và địa chỉ các cổng ngoại vi I/O. Mỗi cổng I/O sẽ có một địa chỉ hoàn toàn riêng biệt. Chủ đề này sẽ được bàn kỹ hơn trong chương 8

Địa chỉ ô nhớ, hay được hiểu là vị trí lưu trữ dữ liệu tương tự như phương thức đánh địa chỉ thư tín xác định nơi gửi và nhận thư. Trong suốt quá trình đọc/ghi dữ liệu, bus địa chỉ sẽ chứa địa chỉ mà dữ liệu sẽ được đọc hoặc ghi. Thuật ngữ đọc và ghi được hiểu với chủ thể là CPU, tức là CPU sẽ đọc dữ liệu từ bộ nhớ hoặc ghi dữ liệu lên bộ nhớ. Nếu dữ liệu được đọc từ bộ nhớ thì bus dữ liệu sẽ chứa nội dung đọc được ở địa chỉ ô nhớ được chỉ ra trên bus địa chỉ. Nếu dữ liệu được ghi vào bộ nhớ thì bus dữ liệu sẽ chứa dữ liệu cần được ghi vào ô nhớ tương ứng trong bộ nhớ

Bus điều khiển có phần phức tạp và sẽ được thảo luận trong những chương tiếp theo. Để dễ hiểu, ta có thể coi bus điều khiển được sử dụng để cho phép truy cập vafp hệ thống bus dữ liệu và bus địa chỉ, phối hợp các tương tác giữa các khối chức năng

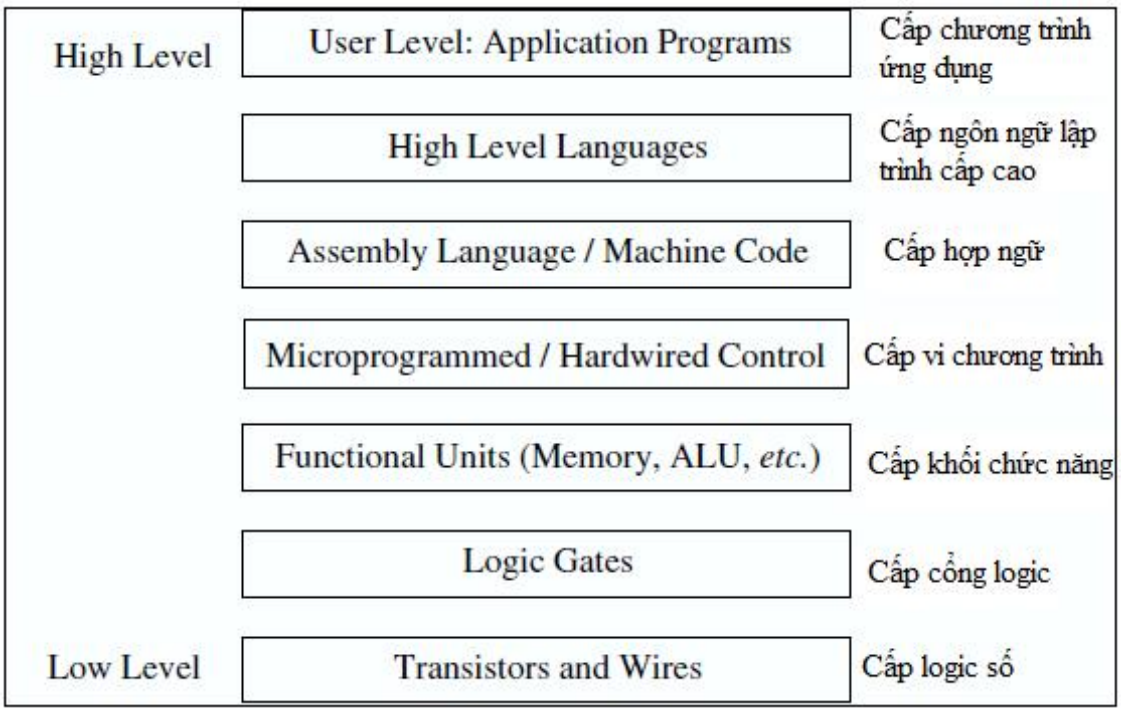
1.5. Mức máy tính

Trong một hệ thống phức hợp, máy tính có thể được nhìn nhận thành các mức máy khác nhau, từ mức cao nhất, mức "người sử dụng" đến mức thấp nhất là mức "transistor". Mỗi một mức thể hiện một mức độ trừu tượng khác nhau về máy tính. Có lẽ một trong những nguyên nhân của sự thành công của máy tính số là sự phân chia các mức trừu tượng một cách rõ ràng, độc lập với nhau. Điều hiển nhiên có thể nhận thấy là một người sử dụng máy tính để gõ văn bản không cần hiểu biết về lập trình. Đồng thời một lập trình viên cũng không cần quan tâm đến các thành phần cấu tạo nên máy tính. Một điều thú vị là việc phân chia máy tính thành các cấp máy đã được khai thác để phát triển các dòng máy tính có chức năng khác nhau

Mức cổng logic, transistor, dây dẫn

Cấp thấp nhất ở bất cứ một máy tính cấp cao chính là cấp cổng logic, transistor và dây dẫn. Cấp này được tạo ra bởi các cổng logic được thiết kế để thực hiện một chức năng nhất định, thực hiện một thuật toán nhất định. Ở cấp độ này, máy tính bao gồm các phần tử điện như transistor, dây dẫn,... Cũng tại cấp độ này, chức năng của máy tính chưa được thể hiện rõ vì hoạt động của nó chỉ thể hiện

thông qua các tín hiệu điện áp, dòng điện, các tín hiệu trễ, tín hiệu lượng tử và các vấn đề ở cấp độ thấp hơn



Hình 1.4. Các cấp máy tính

Cấp khối chức năng

Trong cấp khối chức năng, các thanh ghi thực hiện việc dịch chuyển dữ liệu vào và ra khỏi các "các khối chức năng" dưới sự kiểm soát của các khối điều khiển. Các khối chức năng này thể hiện một số chức năng quan trọng của sự hoạt động của máy tính. Các khối chức năng này bao gồm các thanh ghi bên trong CPU, khối ALU và bộ nhớ chính của máy tính

Cấp vi chương trình

Đây là cấp độ thể hiện sự tác động của khối điều đến việc dịch chuyển dữ liệu từ thanh ghi đến các thanh ghi và đến các khối chức năng khác ra làm sao. Khối điều khiển sẽ nạp lần lượt mã lệnh và thực thi từng lệnh theo một chương trình đã được định sẵn bởi nhà sản xuất các chip vi xử lý được gọi là các vi chương trình. Thực ra, người lập trình không cần quan tâm lắm đến sự hoạt động của cấp độ này bởi vì các vi chương trình là cố định, chỉ có người thiết kế phần cứng mới tác động được đến các vi chương trình này

Cấp hợp ngữ

Từ cấp độ này, các lập trình viên có thể tự viết các chương trình để bắt máy tính thực hiện các yêu cầu của mình. Tuy nhiên, máy tính chỉ có thể hiểu được mã máy bao gồm các chuỗi số 0 và 1. Việc lập trình kiểu như vậy rất dễ bị lỗi. Do đó việc xuất hiện một ngôn ngữ gần với ngôn ngữ con người là điều tất yếu – ngôn ngữ hợp dịch. Một trình biên dịch sẽ chuyển ngôn ngữ hợp dịch sang ngôn ngữ máy và máy tính có thể hiểu được. Tập hợp các lệnh của ngôn ngữ hợp dịch được gọi là tập lệnh

Ngôn ngữ cấp cao

Bất cứ một lập trình viên nào đã sử dụng một trong những ngôn ngữ như C, Pascal, Fortran, hay Java đều đã tương tác với máy tính ở cấp độ ngôn ngữ cấp cao. Tại cấp độ này, lập trình viên tương tác với dữ liệu và mã lệnh chương trình thông qua ngôn ngữ cấp cao, rất giống với ngôn ngữ hàng ngày mà không cần quan tâm tới việc dữ liệu và mã lệnh đó được máy tính xử lý như thế nào

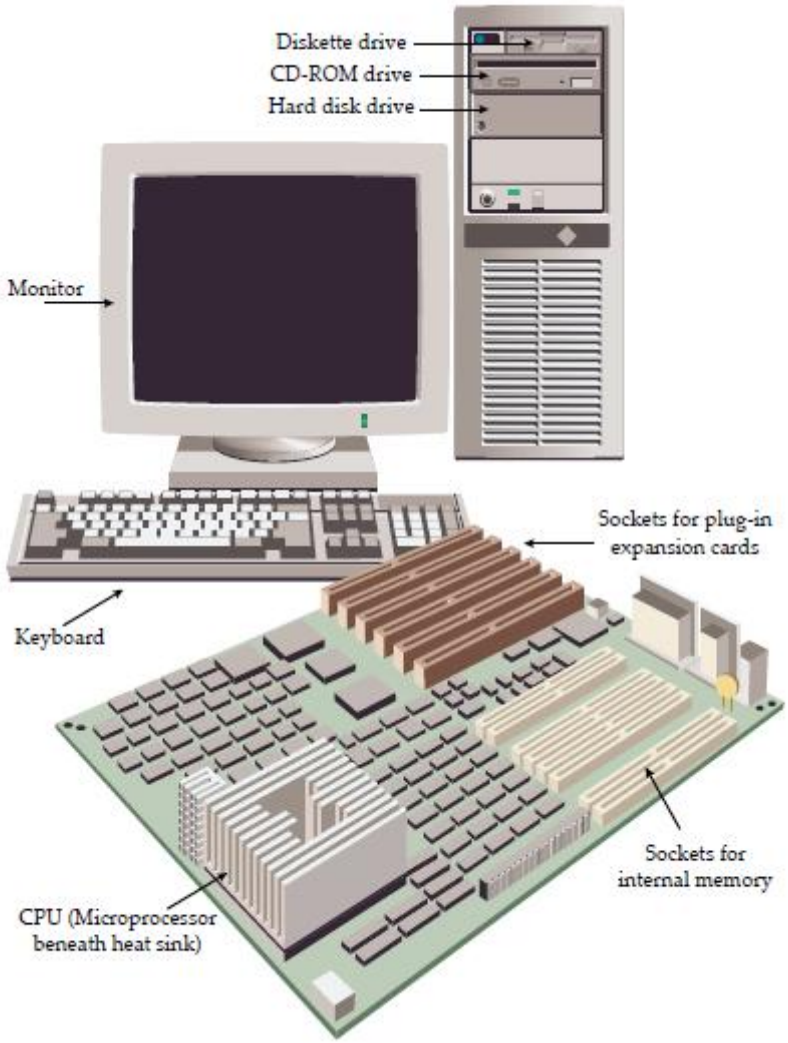
Thực tế, để máy tính có thể hiểu được các lệnh được viết bằng ngôn ngữ cấp cao, máy tính phải thực hiện quá trình chuyển đổi từ ngôn ngữ cấp cao thành ngôn ngữ máy thông qua một trong hai quá trình biên dịch hoặc thông dịch. Biên dịch (Compiler) là quá trình chuyển đổi mã lệnh của toàn bộ chương trình từ ngôn ngữ cấp cao thành ngôn ngữ cấp thấp rồi máy tính mới thực thi chương trình. Thông dịch (Interpreter) là quá trình chuyển đổi từng câu lệnh từ ngôn ngữ cấp cao thành ngôn ngữ cấp thấp, thực thi lệnh rồi chuyển đổi tiếp câu lệnh kế tiếp

Cấp chương trình ứng dụng

Ở cấp độ này, người sử dụng tương tác với máy tính bằng cách chạy các chương trình như soạn thảo văn bản, các bảng tính hay game. Người sử dụng sẽ sử dụng máy tính thông quá các chương trình chạy trên nó

1.6. Hệ thống máy tính điển hình

Mẫu máy tính hiện đại được phát triển từ những năm 1950 đến 1960 và càng ngày càng có kích thước nhỏ gọn và càng ngày càng mạnh mẽ. Mặc dù đã có rất nhiều cải tiến nhưng 5 thành phần cơ bản trong mô hình Von Neumann vẫn không thể thay đổi trong máy tính hiện đại

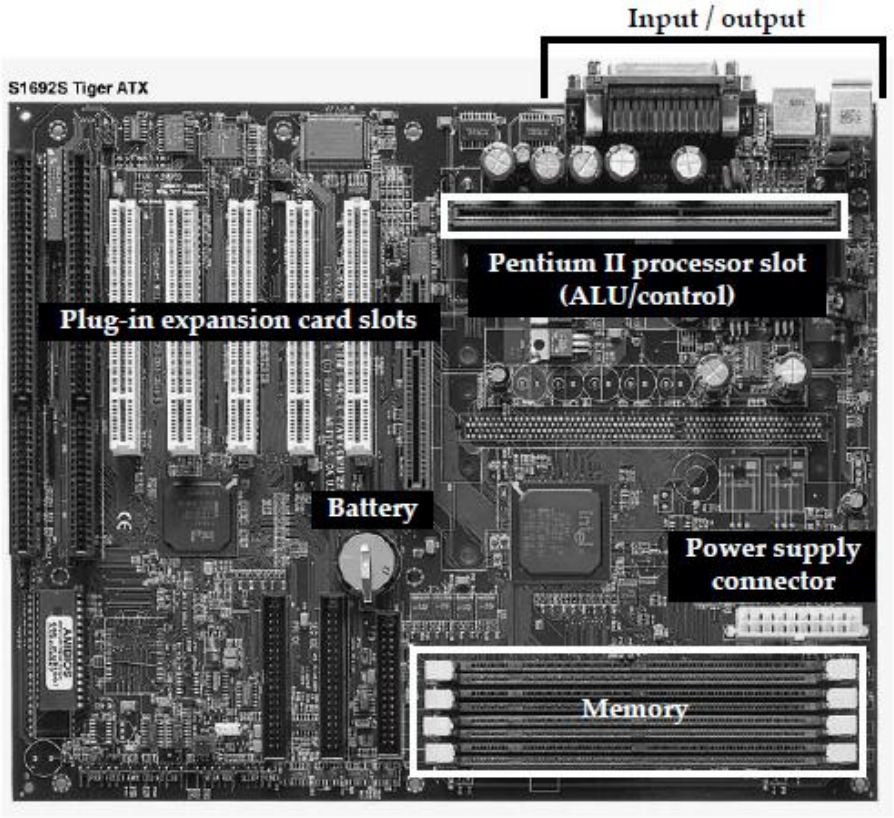


Hình 1.5. Các thành phần của máy tính hiện đại

Hình 1.5 thể hiện các thành phần điển hình của một máy tính để bàn. Khối nhập liệu chính là bàn phím, thông qua nó, người sử dụng sẽ nhập các dữ liệu và các dòng lệnh vào hệ thống. Màn hình là nơi hiển thị các dữ liệu ra bên ngoài. Khối ALU và khối điều khiển được chế tạo trên một chip điện tử được gọi là CPU. Khối bộ nhớ bao gồm các mạch điện độc lập và các khối ổ đĩa cứng, đĩa mềm và các ổ CD-ROM,...

Nhìn sâu hơn vào hệ thống, chúng ta có thể thấy được thành phần quan trọng nhất của hệ thống đó là bản mạch chính (mainboard), hình 1.6. Bản mạch chính

bao gồm các mạch điện tích hợp Ics, các khe cắm các card mở rộng, các dây nối để kết nối các mạch tích hợp và các khe cắm mở rộng. Trên hình 1.6. các vị trí gắn các khối nhập liệu input, khối hiển thị dữ liêu output, bộ nhớ và CPU được khoanh vùng và chỉ rõ trên hình



Hình 1.6. Cấu trúc mainboard

TỔNG KẾT CHƯƠNG

Cấu trúc của máy tính cùng với các cấp độ của nó cần được thể hiện một cách rõ ràng dưới con mắt của lập trình viên để lập trình viên có thể tương tác với máy tính một cách dễ dàng. Tuy nhiên, yêu cầu đó lại không được dễ dàng thực hiện được. Trong lịch sử phát triển, các lập trình viên phải tương tác với máy tính thông qua các khía cạnh khác nhau. Ví dụ như Babbage phải lập trình thông qua các bánh răng cơ khí,...Cùng với sự phát triển của công nghệ, các cấp độ của máy tính cũng trở nên rõ ràng hơn, cho phép máy tính có nhiều hơn các tương tác với con người. Một mô hình phát triển nhất chính là mô hình máy tính Von Neumann, đây là mô hình thông dụng nhất trong các máy tính ngày nay.

CHƯƠNG 2: BIỂU DIỄN DỮ LIỆU

2.1. Giới thiệu chung

Trong những ngày đầu tiên, có một số quan niệm sai lầm về máy tính. Một trong những quan niệm đó là máy tính chỉ là một cái máy kích thước lớn có khả năng tính toán. Máy tính đã có thể làm được nhiều hơn thế ngay cả ở trong giai đoạn khởi đầu. Một quan niệm sai lầm khác là máy tính có thể làm tất cả. Chúng ta đều biết rằng có những vấn đề rất phức tạp mà ngay cả máy tính mạnh nhất hiện nay cũng không có khả năng thực hiện được. Quan điểm đúng đắn tất nhiên nằng ở khoảng nào đó giữa 2 quan niệm trên

Chúng ta đã rất quen thuộc với những hoạt động của máy tính mà không liên quan tới các con số như đồ họa, âm thanh kỹ thuật số, thậm chí cả những cái nhấp chuột...Vậy làm thế nào mà máy tính có thể xử lý được tất cả các loại thông tin đó, khi mà tất cả các thông tin mà máy tính xử lý được chỉ được biểu diễn bằng các con số 0 và 1. Ngay lập tức trong chúng ta sẽ có một câu hỏi là các thông tin đã được máy tính biểu diễn hay mã hóa như thế nào ?

Chúng ta chắc hẳn đã nghĩ rằng máy tính đã sử dụng số thập phân, nhưng máy tính thực tế đã sử dụng mã nhị phân để biểu diễn thông tin. Chương này sẽ giới thiệu một vài cách biểu diễn thông tin phổ biến và quan trọng nhất là số dấu phẩy tĩnh có dấu và không dấu, số thực và các ký tự

Liên quan đến việc biểu diễn dữ liệu, có một khía cạnh mà chúng ta cũng cần quan tâm tới đó là độ rộng của dữ liệu mà máy tính biểu diễn. Ví dụ như máy tính 32 bit có thể xử lý các dữ liệu có độ rộng là 32 bit. Khi đó, kết quả phép toán có thể không còn là 32 bit nữa mà có thể là một con số chỉ có thể biểu diễn bằng một dữ liệu lớn hơn 32 bit. Điều này dẫn tới hiện tượng tràn số - overflow. Do đó, chúng ta cần phải tìm hiểu giới hạn của mỗi một phương pháp biểu diễn dữ liệu mà chúng ta xem xét trong những phần tiếp theo dưới đây.

2.2. Số dấu phẩy tĩnh

Trong hệ thống sử dụng số dấu phẩy tĩnh, mỗi con số được biểu diễn bằng một số chính xác các con số, và có một dấu “phẩy” được sử dụng để phân cách giữa phần nguyên và phần lẻ được đặt ở một vị trí chính xác. Một ví dụ về số dấu phẩy tĩnh thể hiện trong số thập phân là 0.23, 5.12 hay 9.11. Trong ví dụ này,

mỗi số được biểu diễn bằng 3 số và có 1 dấu phẩy đặt ở vị trí thứ 2 từ bên phải sang. Điều khác biệt quan trọng giữa việc biểu diễn số dấu phẩy tĩnh trên giấy và trên máy tính đó là "dấu phẩy tĩnh" trong máy tính không được lưu trữ ở bất kỳ đâu. Người ta nói rằng, "dấu phẩy tĩnh" đó chỉ tồn tại trong đầu của lập trình viên

Để nghiên cứu về số dấu phẩy tĩnh, trước hết chúng ta nên tìm hiểu về phạm vi sử dụng cũng như độ chính xác của số dấu phẩy tĩnh trong hệ số thập phân. Sau đó, chúng ta sẽ đi vào bản chất của các hệ số chẳng hạn như số thập phân và số nhị phân và phương pháp chuyển đổi giữa các hệ số. Và với nền tảng này, ta sẽ tiếp tục tìm hiểu về cách biểu diễn số âm trong số dấu phẩy tĩnh.

2.2.1. Phạm vi và độ chính xác của số dấu phẩy tĩnh

Một số dấu phẩy tĩnh có thể được phân chia theo phạm vi biểu diễn các con số (đó là phạm vi biểu diễn của con số lớn nhất và nhỏ nhất) và độ chính xác của nó. Với ví dụ về số dấu phẩy tĩnh ở trên, phạm vi biểu diễn số của nó là các con số từ 0.00 đến 9.99 với mỗi bước nhảy là 0.01. Do đó, sai số được tính là 0.01

Cần lưu ý rằng phạm vi và độ chính xác phụ thuộc vào vị trí của dấu phẩy tĩnh. Với dấu phẩy tĩnh dịch chuyển về phía bên phải, phạm vi sẽ là [000,999] với độ chính xác là 1.0 và nếu dấu phẩy dịch chuyển về phía trái, phạm vi biểu diễn số sẽ là [0.000,0.999] và sai số sẽ là 0.001. Trong các trường hợp trên, các số được biểu diễn bằng 3 con số, phạm vi của nó từ 000 đến 999 hoặc .000 đến .999, tức là nó có thể biểu diễn chỉ 1000 giá trị không hơn không kém, không phụ thuộc vào phạm vi và độ chính xác.

Ngoài ra, cũng không có một lý do nào bắt buộc chúng ta phải bắt đầu dãy số cần biểu diễn bằng giá trị 0. Một số thập phân có 2 chữ số có thể là dãy [00,99] hoặc [-50,49], thậm chí là [-99,00]. Việc biểu diễn số âm sẽ được trình bày kỹ ở mục sau

2.2.2. Luật kết hợp đại số không phải lúc nào cũng có thể thực hiện trên máy tính

Trong toán học chúng ta đã biết rằng

$$a + (b + c) = (a + b) + c$$

Bây giờ chúng ta sẽ phân tích tại sao luật kết hợp này lại không phải lúc nào cũng có thể thực hiện được trên máy tính. Nếu máy tính chúng ta đang xét là máy có thể thực hiện biểu diễn 1 con số, giả sử phạm vi biểu diễn của nó là [-9,9] với a = 7, b = 4 và c = -3. Vế trái a + (b + c) = 7 + (4 + -3) = 7 + 1 = 8. Nhưng vế phải (a + b) + c = (7 + 4) + -3 = 11 + -3 nhưng số 11 nằm ngoài phạm vi biểu diễn của hệ thống này. Chúng ta có hiện tượng tràn số trong tính toán mặc dù kết quả cuối cùng vẫn nằm trong khoảng có thể biểu diễn được.

Ví dụ trên đã thể hiện luật kết hợp đại số sẽ không được áp dụng cho các số có độ dài hữu hạn. Điều này là không thể tránh khỏi bởi vì chính bản thân cách biểu diễn dữ liệu. Để khắc phục nhược điểm này, hệ thống sẽ ngừng ngay việc tính toán khi phát hiện hiện tượng tràn số, thông báo cho người sử dụng hoặc giải quyết bằng phương pháp khác là lặp lại tính toán với độ rộng dữ liệu lớn hơn

2.2.3. Hệ thống cơ số bất kỳ

Trong phần này, chúng ta sẽ nghiên cứu việc biểu diễn các con số và chuyển đổi giữa các hệ số thường được sử dụng trong máy tính: số nhị phân (binary), số bát phân (octal) và số hecxa (hecxadecimal)

Cơ số của hệ thống số là khoảng giá trị có thể biểu diễn được bởi các con số trong hệ thống đó. Ví dụ cơ số thập phân có 10 con số được sử dụng để biểu diễn dữ liệu là 0,1,2,3,4,5,6,7,8,9. Công thức tổng quát để biểu diễn một số dấu phẩy tĩnh cơ số k là

$$Value = \sum_{i=-m}^{n-1} b_i \cdot k^i$$

Giá trị của con số ở vị trí thứ i được thể hiện bởi b_i. Đồng thời n và m là số lượng các con số ở bên trái và bên phải dấu phẩy tĩnh. Với cấu trúc này, mỗi một con số có trọng số nhất định. Giả sử với giá trị $(541.25)_{10}$ được thể hiện dưới cơ số 10. Ta sẽ có n = 3, m = 2 và k = 10

$$5 \times 10^2 + 4 \times 10^1 + 1 \times 10^0 + 2 \times 10^{-1} + 5 \times 10^{-2} =$$

$$(500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10} = (541.25)_{10}$$

Xem xét cơ số 2 $(1010.01)_2$ với n = 4, m = 2 và k =2

$$1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} \quad =$$

$$(8)_{10} + (0)_{10} + (2)_{10} + (0)_{10} + (0/2)_{10} + (1/4)_{10} \quad = (10.25)_{10}$$

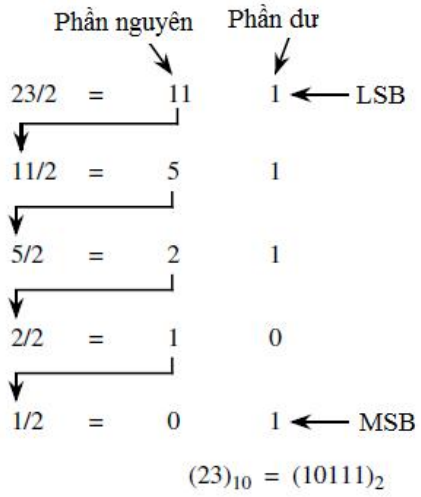
Ví dụ trên gợi ý cho ta phương pháp chuyển đổi một con số có cơ số bất kỳ sang cơ số 10. Tư tưởng của phương pháp là nhân mỗi một con số với cơ số lũy thừa trọng số và sau đó tính tổng tất cả các giá trị đó lại. Chú ý rằng số có trọng số lớn nhất được ký hiệu là MSB và số có trọng số nhỏ nhất được ký hiệu là LSB

Chuyển đổi giữa các cơ số

Trong phần trên, chúng ta đã tìm hiểu cách chuyển từ cơ số bất kỳ sang cơ số 10. Trong phần này chúng ta sẽ tìm hiểu phương pháp chuyển ngược lại từ cơ số 10 sang cơ số bất kỳ. Ví dụ ta cần chuyển số $(23.375)_{10}$ sang cơ số 2. Chúng ta bắt đầu bằng cách chia số đó thành phần nguyên và phần thập phân

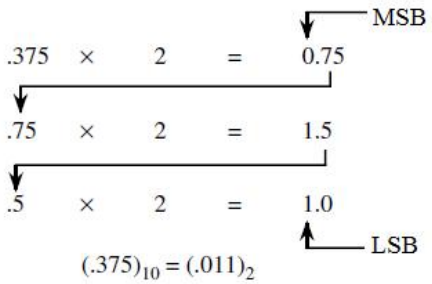
$$(23.375)_{10} = (23)_{10} + (.375)_{10}.$$

Với phần nguyên



$(23)_{10} = (10111)_2$

Với phần thập phân



$(.375)_{10} = (.011)_2$

Kết quả ta sẽ được

$$(23.375)_{10} = (10111.011)_2.$$

2.2.4. Biểu diễn các số nhị phân, bát phân, và hecxa

Binary (base 2)	Octal (base 8)	Decimal (base 10)	Hexadecimal (base 16)
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

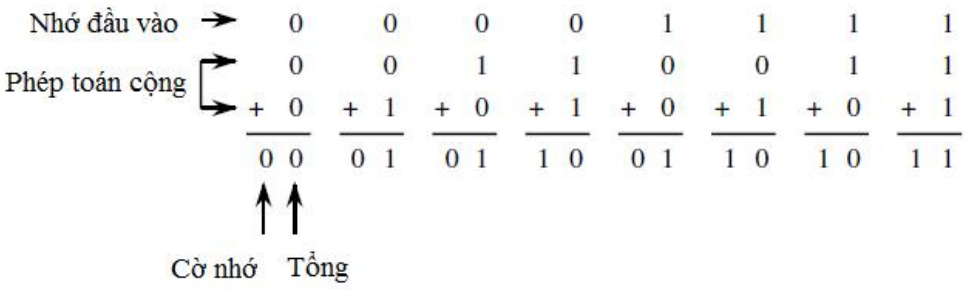
Chuyển đổi qua lại từ cơ số 2 sang cơ số 8 và cơ số 16

$$(10110)_2 = (010)_2\,(110)_2 = (2)_8\,(6)_8 = (26)_8$$

$$(10110110)_2 = (1011)_2\,(0110)_2 = (B)_{16}\,(6)_{16} = (B6)_{16}$$

2.2.5. Phép toán cơ bản trong máy tính

Phép toán cơ bản trong máy tính là phép cộng. Từ phép toán này, ta sẽ phát triển các phép toán khác (chúng ta sẽ nói rõ vấn đề này trong chương 3). Nguyên tắc thực hiện phép cộng nhị phân giống nguyên tắc phép cộng số thập phân mà ta đã thực hiện từ phổ thông



Ví dụ

Nhớ		1	1	1	1	0	0	0	0	
Toán hạng A		0	1	1	1	1	1	0	0	$(124)_{10}$
Toán hạng B	+	0	1	0	1	1	0	1	0	$(90)_{10}$
Tổng		1	1	0	1	0	1	1	0	$(214)_{10}$

Lưu ý rằng số lớn nhất có thể biểu diễn được bằng số 8 bit là $(11111111)_2 = (255)_{10}$ và số nhỏ nhất là $(00000000)_2 = (0)_{10}$. Với cách biểu diễn như vậy, tổ hợp nằm giữa khoảng 11111111 và 00000000 là các con số có giá trị trong khoảng 0 đến 255 là các con số có giá trị dương. Đây là quy ước của số không dấu. Nếu quy ước số có dấu, khoảng một nửa các trạng thái bit sẽ được sử dụng để biểu diễn số dương và một nửa trạng thái còn lại biểu diễn số âm. 4 phương pháp biểu diễn số có dấu sẽ được chúng ta đề cập trong mục kế tiếp ngay sau đây

2.2.6. Số có dấu

Như đã nói ở mục trên, số có dấu có thể được biểu diễn bằng 1 trong 4 cách. Đó là phương pháp dùng bit MSB làm bit dấu (sign-magnitude), số bù 1 (one's complement), số bù 2 (two's complement) và số thừa (Excess). Mối quan hệ giữa các phương pháp biểu diễn được mô tả bởi bảng dưới đây

Decimal	Unsigned	Sign–Mag.	1's Comp.	2's Comp.	Excess 4
7	111	–	–	–	–
6	110	–	–	–	–
5	101	–	–	–	–
4	100	–	–	–	–
3	011	011	011	011	111
2	010	010	010	010	110
1	001	001	001	001	101
+0	000	000	000	000	100
-0	–	100	111	000	100
-1	–	101	110	111	011
-2	–	110	101	110	010
-3	–	111	100	101	001
-4	–	–	–	100	000

Số dùng MSB làm bit dấu

Phương pháp này có cách biểu diễn số âm và dương giống như đối với cơ số 10 sử dụng dấu + và dấu –. Điểm khác biệt đó là nó sẽ dùng 1 bit có ở tận cùng phía trái làm bit dấu. Giá trị 1 tương ứng với số âm và giá trị 0 tương ứng với số dương. Các con số còn lại trong chuỗi số biểu diễn độ lớn của số theo mã nhị phân thông thường. Với cách biểu diễn này, chúng ta sẽ có 2 số mang giá trị 0 là +0 và -0

$$(+12)_{10} = (00001100)_2$$

$$(-12)_{10} = (10001100)_2$$

Chúng ta thường sử dụng phương pháp này để biểu diễn số đối với số dấu phẩy động (xem tiếp mục 2.3)

Số bù 1

Số bù 1 là phương pháp cũng ít khi được sử dụng. Phương pháp này đảo tất cả các bít từ 0 lên 1 và từ 1 về 0 của tất cả các bit trong số cần biểu diễn. Với cách biểu diễn này, nếu bit tận cùng bên trái có giá trị là 1 thì đây là một số âm

$$(+12)_{10} = (00001100)_2$$

$$(-12)_{10} = (11110011)_2$$

Phương pháp sử dụng số bù 1 rất hiếm khi được sử dụng bởi vì khi sử dụng phương pháp này, việc so sánh giá trị của các con số thực hiện tương đối phức tạp vì có 2 trạng thái bit biểu diễn giá trị 0. Hơn nữa, việc thực hiện các phép toán cũng khó thực hiện

Số bù 2

Số bù 2 được tạo ra từ số bù 1 cộng thêm 1 đơn vị. Số tận cùng bên trái thể hiện số đó là âm (giá trị 1) hoặc dương (giá trị 0). Phương pháp biểu diễn này khắc phục được nhược điểm của 2 phương pháp biểu diễn trình bày phía trên là chỉ sử dụng 1 cách biểu diễn cho số 0.

$(+12)_{10} = (00001100)_2$

$(-12)_{10} = (11110100)_2$

Bàn thêm về phạm vi của các con số mà nó có thể biểu diễn được. Giả sử số chúng ta cần biểu diễn là các con số 8 bit. Như vậy chúng ta sẽ có tất cả 256 trạng thái khác nhau để biểu các giá trị. Sử dụng 1 trạng thái để biểu diễn số 0, chúng ta sẽ còn 127 trạng thái để biểu diễn các giá trị dương và 128 trạng thái biểu diễn giá trị âm

Số bù 2 được sử dụng rộng rãi trong các hệ thống máy tính. Chúng ta sẽ sử dụng phương pháp biểu diễn này trong toàn bộ phần sau của tài liệu này

Số thừa

Phương pháp này đánh lừa máy tính, nó làm cho máy tính không phân biệt được là nó đang tương tác với số có dấu. Số thừa được tạo ra bằng cách xác định một giá trị thừa (hay còn gọi là giá trị dịch chuyển gốc 0) để làm mốc giá trị 0 giả sử là 128. Khi đó, giá trị giả sử là +12 sẽ được tạo ra bằng cách tính toán $(128 + 12 = 140)_{10}$. Và giá trị 140 khi biểu diễn trong cơ số 2 sẽ đại diện cho số +12

$(+12)_{10} = (10001100)_2$

$(-12)_{10} = (01110100)_2$

Cách biểu diễn trên là số thừa 128 của các giá trị +12 và -12. Chúng ta có thể tham khảo thêm về số thừa với các ví dụ trong bảng (số thừa 4)

2.2.7. Số BCD (Binary Coded Decimal)

Mã BCD sử dụng các tổ hợp 4 bit nhị phân để biểu diễn các giá trị từ 0 đến 9 của số thập phân

(a)	0000 $(0)_{10}$	0011 $(3)_{10}$	0000 $(0)_{10}$	0001 $(1)_{10}$	$(+301)_{10}$	Nine's and ten's complement
(b)	1001 $(9)_{10}$	0110 $(6)_{10}$	1001 $(9)_{10}$	1000 $(8)_{10}$	$(-301)_{10}$	Nine's complement
(c)	1001 $(9)_{10}$	0110 $(6)_{10}$	1001 $(9)_{10}$	1001 $(9)_{10}$	$(-301)_{10}$	Ten's complement

Để biểu diễn số âm trong mã BCD, ta có thể sử dụng một trong hai phương pháp là mã bù 9 và mã bù 10. Trong mã bù 9 và mã bù 10, số dương được biểu diễn bình thường. Với số âm trong mã bù 9, tổ hợp BCD cao nhất thể hiện là số âm nếu có giá trị lớn hơn hoặc bằng 5, và biểu thị số dương nếu có giá trị nhỏ hơn 5. Các giá trị còn lại được tính bằng cách lấy giá trị 9 trừ đi giá trị dương tương ứng tại vị trí đó. Trong ví dụ trên, tại phần (b) ta thấy, số -301 được tổ hợp bởi

(b)	1001	0110	1001	1000	$(-301)_{10}$
	$(9)_{10}$	$(6)_{10}$	$(9)_{10}$	$(8)_{10}$	
Bit dấu có giá trị bằng 9 - 0 = 9		(3=9-6)	0=9-9	(1=9-8)	

Tương tự như số bù 9, số bù 10 được tạo thành số bù 9 cộng thêm 1 đơn vị.

2.3. Số dấu phẩy động

2.3.1. Hạn chế của số dấu phẩy tĩnh

Số dấu phẩy tĩnh có số lượng chính xác các bit để biểu diễn số. Để biểu diễn số 1 tỉ, ta cần khoảng 40 bit ở phía bên trái của dãy số. Để biểu diễn độ chính xác 1 phần tỉ, ta cũng cần phải sử dụng khoảng 40 bit nữa ở phía bên phải "sau dấu chấm". Như vậy, ta cần khoảng 80 bit để biểu diễn số trên.

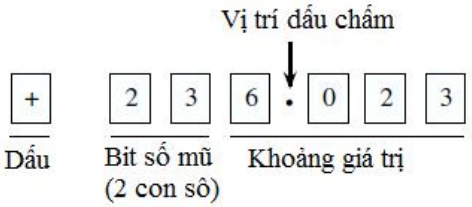
Trong thực tế, rất nhiều trường hợp ta lại cần tính toán các con số lớn hơn 1 tỉ thậm chí lớn hơn khả năng tính toán của máy tính. Để tính toán con số càng lớn, phần cứng của máy tính đòi hỏi phải mạnh hơn để lưu trữ và tính toán. Trong lúc đó, độ chính xác cao của phép toán lại tỏ ra là không cần thiết đối với các phép tính số lớn. Ngược lại, đôi khi ta không cần biểu diễn số lớn nếu nó được tạo ra từ các số nhỏ.

2.3.2. Độ lớn và độ chính xác của số dấu phẩy động

Cách biểu diễn của số dấu phẩy động cho phép biểu diễn số có giá trị lớn bằng một dãy các bit có số lượng hữu hạn bằng cách chia một số lượng bit để biểu diễn độ chính xác và số lượng bit còn lại sẽ biểu diễn độ lớn của con số

Ví dụ $+6.023 \times 10^{23}$

Trong ví dụ trên, độ lớn mà con số được biểu diễn lên tới khoảng 10^{23} còn độ chính xác của con số được biểu diễn bằng một số dấu phẩy tĩnh. Để biểu diễn giá trị này, số dấu phẩy động cần có 3 thành phần: phần dấu âm dương, số mũ cơ số 10 và phần giá trị



Để có thể biểu diễn số có giá trị lớn hơn nữa, ta phải hi sinh 1 bit trong phần khoảng giá trị tức là hi sinh độ chính xác của con số đang biểu diễn để tăng bit đó vào phần số mũ. Việc thay đổi các bit tại các trường này sẽ làm thay đổi độ lớn và độ chính xác của con số cần biểu diễn

2.3.3. Chuẩn hóa và những bit đã được giấu đi

Một vấn đề nổi cộm của số dấu phẩy động là một con số có thể được biểu diễn bằng các quy chuẩn khác nhau. Điều này gây khó khăn cho việc so sánh và thực hiện các phép toán. Ví dụ

$$3584.1 \times 10^0 = 3.5841 \times 10^3 = .35841 \times 10^4$$

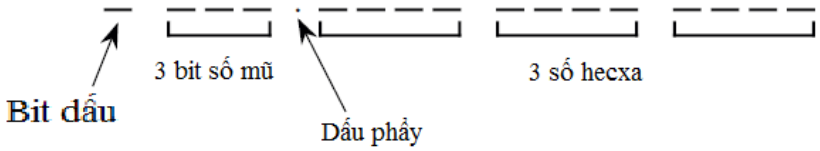
Để tránh việc cùng một con số được biểu diễn bằng nhiều cách như ví dụ trên, các số dấu phẩy động cần được chuẩn hóa. Theo đó, dấu phẩy được đẩy sang trái hoặc sang phải và số mũ được điều chỉnh tương ứng cho đến khi số đầu tiên về phía bên phải sau dấu chấm không phải là một số 0. Với cách chuẩn hóa như vậy, số tận cùng bên phải ở ví dụ trên là số đã được chuẩn hóa.

Đối với trường hợp số cần biểu diễn là số nhị phân. Sau khi chuẩn hóa, số cần biểu diễn sẽ luôn bắt đầu bằng số 1, khi đó không nhất thiết cần phải biểu diễn số 1 đó. Thực tế, con số 1 đó sẽ không được lưu lại và luôn được bỏ đi. Kết quả là sẽ có thêm 1 bit để biểu diễn con số và điều này làm tăng độ chính xác của con số mà ta cần biểu diễn. Bit mà ta bỏ đi được gọi là **những bit được dấu đi**

2.3.4. Biểu diễn số dấu phẩy động trong máy tính

Để biểu diễn số dấu phẩy động, chúng ta sử dụng cấu trúc sau

- Bit dấu được đặt ở tận cùng phía bên trái
- Tiếp theo là 3 bit thể hiện số mũ cơ số 10. 3 bit này được biểu diễn bằng mã thừa 4 với cơ số 16
- Tiếp theo là dấu phẩy đã được chuẩn hóa và 3 số hecxa biểu diễn độ lớn của số



Bây giờ, ta sẽ thử chuyển đổi số $(358)_{10}$ sang số dấu phẩy động. Trước hết, ta chuyển con số ta cần biểu diễn sang số có cơ số 16.

	Phần nguyên	Nhớ
358/16 =	22	6
22/16 =	1	6
1/16 =	0	1

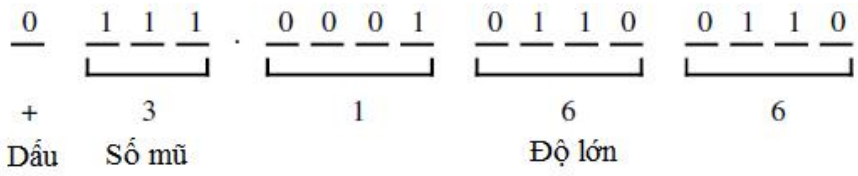
Khi đó ta được $(358)_{10} = (166)_{16}$. Bước tiếp theo là chuyển sang số dấu phẩy động

$$(166)_{16} = (166.)_{16} \times 16^0$$

Chú ý rằng $(16^0)_{10} = (10^0)_{16}$. Khi đó ta có

$$(166.)_{16} \times 16^0 = (.166)_{16} \times 16^3$$

Số mũ ta được là 3, số này cần được chuyển sang số thừa 4. Tức là nó sẽ có giá trị trong cơ số 10 là 3 + 4 = 7 = $(111)_2$. Cuối cùng ta được



Chúng ta cần lưu ý rằng dấu chấm ở trong chuỗi số trên chỉ là mang tính chất tượng trưng, nó không hề tồn tại trong máy tính.

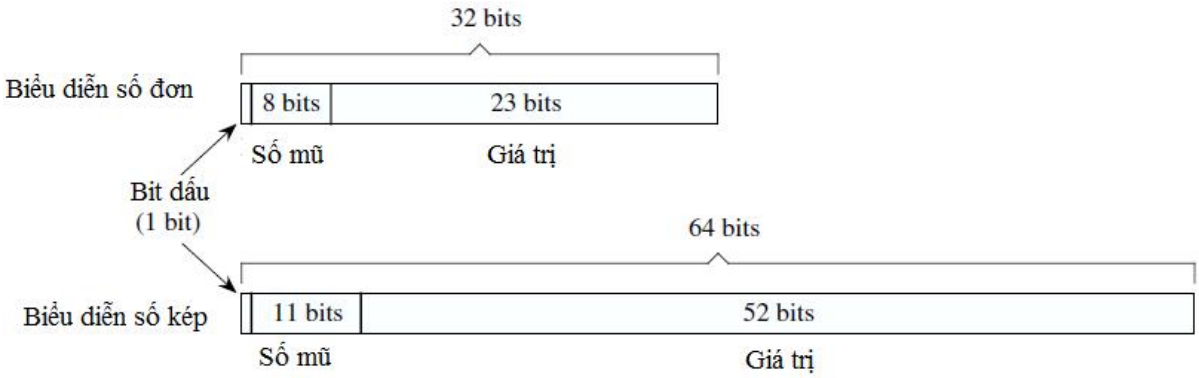
Với 3 bit số mũ, ta sử dụng mã thừa 4 chứ không phải là mã bù hay mã có bít dấu là bởi vì khi sử dụng mã thừa 4, việc tính toán (cộng trừ) trở nên rất thuận tiện. Để thực hiện việc cộng hay trừ 2 số, số có số mũ nhỏ hơn sẽ được phi chuẩn hóa bằng cách tăng số mũ đến giá trị bằng với số mũ của số lớn hơn. Sau khi thay đổi, vì mã thừa không còn sự phân biệt của số có dấu và không dấu nên ta chỉ cần so sánh các bit từ cao đến thấp là ta có thể so sánh được số nào lớn hơn

Bàn luận thêm một chút về sự so sánh 2 số mã thừa 4. Trong mã thừa 4, số nhỏ nhất là -4 được biểu diễn bằng 3 ký tự 000, số lớn nhất là +3 được biểu diễn bởi 111. Các con số -3, -2, -1, 0, 1 và 2 lần lượt được biểu diễn là 001, 010, 011, 100, 101, và 110. Rõ ràng số lớn hơn trong dãy có các bít từ lớn đến nhỏ là lớn hơn.

2.3.5. Biểu diễn số dấu phẩy động theo chuẩn IEEE 754

Trong ví dụ ở phần trên, số dấu phẩy động được biểu diễn bởi 12 bit, vậy máy tính sẽ sử dụng bao nhiêu bit để biểu diễn. Về mặt lý thuyết là không có một giá trị cố định. Do vậy, để tất cả các máy tính có thể hiểu được dữ liệu của nhau, và để tạo sự thống nhất cho các lập trình viên khi xử lý dữ liệu, việc hình thành nên một tiêu chuẩn thống nhất là một tất yếu. Đây là nguyên nhân của sự hình thành nên chuẩn IEEE 754

Các phương pháp biểu diễn



Một vài minh họa

	Giá trị	Chuỗi bit thu được		
		Dấu	Số mũ	Giá trị
(a)	$+1.101 \times 2^5$	0	1000 0100	101 0000 0000 0000 0000 0000
(b)	-1.01011×2^{-126}	1	0000 0001	010 1100 0000 0000 0000 0000
(c)	$+1.0 \times 2^{127}$	0	1111 1110	000 0000 0000 0000 0000 0000
(d)	+0	0	0000 0000	000 0000 0000 0000 0000 0000
(e)	−0	1	0000 0000	000 0000 0000 0000 0000 0000
(f)	$+\infty$	0	1111 1111	000 0000 0000 0000 0000 0000
(g)	$+2^{-128}$	0	0000 0000	010 0000 0000 0000 0000 0000
(h)	+NaN	0	1111 1111	011 0111 0000 0000 0000 0000
(i)	$+2^{-128}$	0	011 0111 1111	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000

Có 1 quy ước biểu diễn giá trị vô cùng là số mũ là 1111 1111 và khoảng giá trị là các con số 0, bit dấu có thể là 0 hoặc 1

Kết quả của phép toán $0/0$ là một số bất định. Khi đó, kết quả được trả về là NaN (Not a Number). Chuỗi quy ước có số mũ là 1111 1111, bit dấu là 0 hoặc 1, còn giá trị là một số khác 0

2.4. Mã ký tự

Khác với các con số có khoảng giá trị là vô hạn, số lượng các ký tự là một số hữu hạn và có thể được biểu diễn bởi môt số lượng các bit. Có 3 phương pháp biểu diễn các ký tự thông dụng là mã ASCII, mã EBCDIC và mã Unicode

2.4.1. Mã ASCII

ASCII là viết tắt từ American Standard Code for Information Interchange. Phương pháp biểu diễn này sử dụng 7 bit để mã hóa ký tự, trong đó các ký tự có mã từ 00 đến 1F và ký tự có mã 7F là các ký tự điều khiển các chức năng đặc biệt. Các ký tự đó được sử dụng cho việc truyền dữ liệu, điều khiển việc in ấn và một số quá trình khác. Các ký tự có mã còn lại đều là các ký tự có thể in được bao gồm các ký tự chữ cái, các con số, các ký hiệu thông thường và khoảng trống

00	NUL	10	DLE	20	SP	30	0	40	@	50	P	60	`	70	p
01	SOH	11	DC1	21	!	31	1	41	A	51	Q	61	a	71	q
02	STX	12	DC2	22	"	32	2	42	B	52	R	62	b	72	r
03	ETX	13	DC3	23	#	33	3	43	C	53	S	63	c	73	s
04	EOT	14	DC4	24	$	34	4	44	D	54	T	64	d	74	t
05	ENQ	15	NAK	25	%	35	5	45	E	55	U	65	e	75	u
06	ACK	16	SYN	26	&	36	6	46	F	56	V	66	f	76	v
07	BEL	17	ETB	27	'	37	7	47	G	57	W	67	g	77	w
08	BS	18	CAN	28	(	38	8	48	H	58	X	68	h	78	x
09	HT	19	EM	29	)	39	9	49	I	59	Y	69	i	79	y
0A	LF	1A	SUB	2A	*	3A	:	4A	J	5A	Z	6A	j	7A	z
0B	VT	1B	ESC	2B	+	3B	;	4B	K	5B	[	6B	k	7B	{
0C	FF	1C	FS	2C	´	3C	<	4C	L	5C	\	6C	l	7C	|
0D	CR	1D	GS	2D	-	3D	=	4D	M	5D	]	6D	m	7D	}
0E	SO	1E	RS	2E	.	3E	>	4E	N	5E	^	6E	n	7E	~
0F	SI	1F	US	2F	/	3F	?	4F	O	5F	_	6F	o	7F	DEL

NUL	Null	FF	Form feed	CAN	Cancel
SOH	Start of heading	CR	Carriage return	EM	End of medium
STX	Start of text	SO	Shift out	SUB	Substitute
ETX	End of text	SI	Shift in	ESC	Escape
EOT	End of transmission	DLE	Data link escape	FS	File separator
ENQ	Enquiry	DC1	Device control 1	GS	Group separator
ACK	Acknowledge	DC2	Device control 2	RS	Record separator
BEL	Bell	DC3	Device control 3	US	Unit separator
BS	Backspace	DC4	Device control 4	SP	Space
HT	Horizontal tab	NAK	Negative acknowledge	DEL	Delete
LF	Line feed	SYN	Synchronous idle		
VT	Vertical tab	ETB	End of transmission block		

2.4.2. Mã EBCDIC

Có một vấn đề của mã ASCII là nó chỉ có thể biểu diễn được 128 ký tự khác nhau bởi vì hạn chế các ký tự có trong bàn phím. Mã EBCDIC (Extended Binary Coded Decimal Interchange Code) được sử dụng để mở rộng thành mã 8 bit. Mã này được sử dụng rộng rãi trong các máy chủ mainframe của IBM. Từ các mã 7 bit của mã ASCII, ta thêm vào đó 1 bit 0 hoặc 1 để thu được mã EBCDIC

Việc sử dụng mã này không làm thay đổi kích thước của vùng nhớ trong máy tính. Tuy nhiên, khi thực hiện truyền dữ liệu, việc truyền mã 8 bit tốn thời gian hơn việc truyền số 7 bit

00	NUL	20	DS	40	SP	60	–	80		A0		C0	{	E0	\
01	SOH	21	SOS	41		61	/	81	a	A1	~	C1	A	E1	
02	STX	22	FS	42		62		82	b	A2	s	C2	B	E2	S
03	ETX	23		43		63		83	c	A3	t	C3	C	E3	T
04	PF	24	BYP	44		64		84	d	A4	u	C4	D	E4	U
05	HT	25	LF	45		65		85	e	A5	v	C5	E	E5	V
06	LC	26	ETB	46		66		86	f	A6	w	C6	F	E6	W
07	DEL	27	ESC	47		67		87	g	A7	x	C7	G	E7	X
08		28		48		68		88	h	A8	y	C8	H	E8	Y
09		29		49		69		89	i	A9	z	C9	I	E9	Z
0A	SMM	2A	SM	4A	¢	6A	‘	8A		AA		CA		EA	
0B	VT	2B	CU2	4B		6B	,	8B		AB		CB		EB	
0C	FF	2C		4C	<	6C	%	8C		AC		CC		EC	
0D	CR	2D	ENQ	4D	(	6D	_	8D		AD		CD		ED	
0E	SO	2E	ACK	4E	+	6E	>	8E		AE		CE		EE	
0F	SI	2F	BEL	4F	|	6F	?	8F		AF		CF		EF	
10	DLE	30		50	&	70		90		B0		D0	}	F0	0
11	DC1	31		51		71		91	j	B1		D1	J	F1	1
12	DC2	32	SYN	52		72		92	k	B2		D2	K	F2	2
13	TM	33		53		73		93	l	B3		D3	L	F3	3
14	RES	34	PN	54		74		94	m	B4		D4	M	F4	4
15	NL	35	RS	55		75		95	n	B5		D5	N	F5	5
16	BS	36	UC	56		76		96	o	B6		D6	O	F6	6
17	IL	37	EOT	57		77		97	p	B7		D7	P	F7	7
18	CAN	38		58		78		98	q	B8		D8	Q	F8	8
19	EM	39		59		79		99	r	B9		D9	R	F9	9
1A	CC	3A		5A	!	7A	:	9A		BA		DA		FA	|
1B	CU1	3B	CU3	5B	$	7B	#	9B		BB		DB		FB	
1C	IFS	3C	DC4	5C	·	7C	@	9C		BC		DC		FC	
1D	IGS	3D	NAK	5D	)	7D	'	9D		BD		DD		FD	
1E	IRS	3E		5E	;	7E	=	9E		BE		DE		FE	
1F	IUS	3F	SUB	5F	¬	7F	"	9F		BF		DF		FF	

STX	Start of text	RS	Reader Stop	DC1	Device Control 1	BEL	Bell
DLE	Data Link Escape	PF	Punch Off	DC2	Device Control 2	SP	Space
BS	Backspace	DS	Digit Select	DC4	Device Control 4	IL	Idle
ACK	Acknowledge	PN	Punch On	CU1	Customer Use 1	NUL	Null
SOH	Start of Heading	SM	Set Mode	CU2	Customer Use 2		
ENQ	Enquiry	LC	Lower Case	CU3	Customer Use 3		
ESC	Escape	CC	Cursor Control	SYN	Synchronous Idle		
BYP	Bypass	CR	Carriage Return	IFS	Interchange File Separator		
CAN	Cancel	EM	End of Medium	EOT	End of Transmission		
RES	Restore	FF	Form Feed	ETB	End of Transmission Block		
SI	Shift In	TM	Tape Mark	NAK	Negative Acknowledge		
SO	Shift Out	UC	Upper Case	SMM	Start of Manual Message		
DEL	Delete	FS	Field Separator	SOS	Start of Significance		
SUB	Substitute	HT	Horizontal Tab	IGS	Interchange Group Separator		
NL	New Line	VT	Vertical Tab	IRS	Interchange Record Separator		
LF	Line Feed	UC	Upper Case	IUS	Interchange Unit Separator		

2.4.3. Mã Unicode

Bảng mã ASCII và mã EBCDIC hỗ trợ các ký tự Latin được sử dụng trong máy tính. Tuy nhiên, trên thế giới có rất nhiều các bộ ký tự khác nhau, và do đó, ký tự ASCII không thể biểu diễn được tất cả các ngôn ngữ này. Vì nguyên nhân đó, một tiêu chuẩn mới được xây dựng để hỗ trợ các ngôn ngữ trên toàn thế giới, đó là chuẩn Unicode

Unicode là một chuẩn có tính chất mở. Nó sẽ luôn cập nhật các ký tự mới vào nó và các ký tự này sẽ được chuẩn hóa, tinh chế để việc sự dụng được thuận lợi. Trong phiên bản 2.0, đã có 38.885 ký tự được mã hóa bao gồm các ký tự của ngôn ngữ cơ bản của Châu Mỹ, Châu Âu, Trung đông, Châu Phi, Ấn độ, Châu Á Thái Bình Dương,...

Chuẩn Unicode sử dụng mã 16 bit để biểu diễn ký tự, trong đó có một sự tương ứng 1-1 giữa mã 16 bit và ký tự biểu diễn. Giống như mã ASCII, Unicode không có một mô hình phức hợp. Mặc dù Unicode hỗ trợ nhiều ký tự hươn ASCII hay EBCDIC nhưng đây cũng không chắc chắn là chuẩn cuối cùng mà chúng ta sử dụng. Thực tế, chuẩn Unicode 16 bit chỉ là một thành phần của chuẩn ký tự 32 bit ISO 10646 (UCS-4)

Bảng mã dưới đây liệt kê 256 ký tự đầu tiên của Unicode 2.1 trong đó có 128 ký tự giống như trong bảng mã ASCII

TỔNG KẾT CHƯƠNG

Tất cả các dữ liệu trong máy tính được biểu diễn bởi một chuỗi các bit. Các bit đó có thể được định nghĩa để đại diện cho số nguyên, số có dấu phẩy tĩnh, số dấu phẩy động hoặc là một ký tự

0000	NUL	0020	SP	0040	@	0060	`	0080	Ctrl	00A0	NBS	00C0	À	00E0	à
0001	SOH	0021	!	0041	A	0061	a	0081	Ctrl	00A1	¡	00C1	Á	00E1	á
0002	STX	0022	"	0042	B	0062	b	0082	Ctrl	00A2	¢	00C2	Â	00E2	â
0003	ETX	0023	#	0043	C	0063	c	0083	Ctrl	00A3	£	00C3	Ã	00E3	ã
0004	EOT	0024	$	0044	D	0064	d	0084	Ctrl	00A4	¤	00C4	Ä	00E4	ä
0005	ENQ	0025	%	0045	E	0065	e	0085	Ctrl	00A5	¥	00C5	Å	00E5	å
0006	ACK	0026	&	0046	F	0066	f	0086	Ctrl	00A6	¦	00C6	Æ	00E6	æ
0007	BEL	0027	'	0047	G	0067	g	0087	Ctrl	00A7	§	00C7	Ç	00E7	ç
0008	BS	0028	(	0048	H	0068	h	0088	Ctrl	00A8	¨	00C8	È	00E8	è
0009	HT	0029	)	0049	I	0069	i	0089	Ctrl	00A9	©	00C9	É	00E9	é
000A	LF	002A	*	004A	J	006A	j	008A	Ctrl	00AA	ª	00CA	Ê	00EA	ê
000B	VT	002B	+	004B	K	006B	k	008B	Ctrl	00AB	«	00CB	Ë	00EB	ë
000C	FF	002C	´	004C	L	006C	l	008C	Ctrl	00AC	¬	00CC	Ì	00EC	ì
000D	CR	002D	-	004D	M	006D	m	008D	Ctrl	00AD	–	00CD	Í	00ED	í
000E	SO	002E	.	004E	N	006E	n	008E	Ctrl	00AE	®	00CE	Î	00EE	î
000F	SI	002F	/	004F	O	006F	o	008F	Ctrl	00AF	¯	00CF	Ï	00EF	ï
0010	DLE	0030	0	0050	P	0070	p	0090	Ctrl	00B0	°	00D0	Đ	00F0	¶
0011	DC1	0031	1	0051	Q	0071	q	0091	Ctrl	00B1	±	00D1	Ñ	00F1	ñ
0012	DC2	0032	2	0052	R	0072	r	0092	Ctrl	00B2	²	00D2	Ò	00F2	ò
0013	DC3	0033	3	0053	S	0073	s	0093	Ctrl	00B3	³	00D3	Ó	00F3	ó
0014	DC4	0034	4	0054	T	0074	t	0094	Ctrl	00B4	´	00D4	Ô	00F4	ô
0015	NAK	0035	5	0055	U	0075	u	0095	Ctrl	00B5	µ	00D5	Õ	00F5	õ
0016	SYN	0036	6	0056	V	0076	v	0096	Ctrl	00B6	¶	00D6	Ö	00F6	ö
0017	ETB	0037	7	0057	W	0077	w	0097	Ctrl	00B7	·	00D7	×	00F7	÷
0018	CAN	0038	8	0058	X	0078	x	0098	Ctrl	00B8	¸	00D8	Ø	00F8	ø
0019	EM	0039	9	0059	Y	0079	y	0099	Ctrl	00B9	¹	00D9	Ù	00F9	ù
001A	SUB	003A	:	005A	Z	007A	z	009A	Ctrl	00BA	º	00DA	Ú	00FA	ú
001B	ESC	003B	;	005B	[	007B	{	009B	Ctrl	00BB	»	00DB	Û	00FB	û
001C	FS	003C	<	005C	\	007C	|	009C	Ctrl	00BC	1/4	00DC	Ü	00FC	ü
001D	GS	003D	=	005D	]	007D	}	009D	Ctrl	00BD	1/2	00DD	Ý	00FD	Þ
001E	RS	003E	>	005E	^	007E	~	009E	Ctrl	00BE	3/4	00DE	ý	00FE	þ
001F	US	003F	?	005F	_	007F	DEL	009F	Ctrl	00BF	¿	00DF	ß	00FF	ÿ

NUL	Null	SOH	Start of heading	CAN	Cancel	SP	Space
STX	Start of text	EOT	End of transmission	EM	End of medium	DEL	Delete
ETX	End of text	DC1	Device control 1	SUB	Substitute	Ctrl	Control
ENQ	Enquiry	DC2	Device control 2	ESC	Escape	FF	Form feed
ACK	Acknowledge	DC3	Device control 3	FS	File separator	CR	Carriage return
BEL	Bell	DC4	Device control 4	GS	Group separator	SO	Shift out
BS	Backspace	NAK	Negative acknowledge	RS	Record separator	SI	Shift in
HT	Horizontal tab	NBS	Non-breaking space	US	Unit separator	DLE	Data link escape
LF	Line feed	ETB	End of transmission block	SYN	Synchronous idle	VT	Vertical tab

CHƯƠNG 3: CÁC PHÉP TOÁN SỐ HỌC

3.1. Tổng quan

Trong chương trước, chúng ta đã tìm hiểu các phương pháp biểu diễn các con số trong máy tính số nhưng chúng ta mới chỉ làm quen với các phép toán cơ bản mà chưa tìm hiểu kỹ máy tính thực hiện công việc đó như thế nào. Trong chương này, chúng ta sẽ tìm hiểu 4 phép toán cơ bản trong máy tính là cộng, trừ, nhân, và chia, tìm hiểu xem máy tính sử lý các phép toán đó được thực hiện như thế nào đối với các trường hợp các con số trong phép toán là các số dấu phẩy tĩnh, dấu phẩy động.

3.2. Phép cộng và trừ với số dấu phẩy tĩnh

Phép cộng số nhị phân và ý nghĩa của hiện tượng tràn overflow đã được chúng ta điểm qua ở trong chương 2. Trong chương này, chúng ta sẽ xem xét kỹ phép cộng và trừ đối với số không dấu và có dấu một cách kỹ càng. Từ khi số bù 2 được sử dụng để biểu diễn một con số đến nay, nó đã được sử dụng rất phổ biến, do vậy, chúng ta sẽ tập trung tìm hiểu các phép toán trên số bù 2

3.2.1. Phép toán cộng và trừ với số bù 2

Trong phần này, chúng ta sẽ tìm hiểu phép cộng đối với số bù 2 có dấu. Tại sao lại chỉ là phép cộng mà không phải là phép trừ ? Phép trừ bản chất cũng là phép cộng, chúng ta có thể coi phép trừ hai số a và b là một phép cộng như sau

$$a - b = a + (-b).$$

Với cách biểu diễn như trên, chúng ta có được số âm từ một số bất kỳ bằng cách lấy số bù 2 của chúng (tìm số bù 1 rồi cộng thêm 1 đơn vị), và để thực hiện phép trừ thì ta sẽ thực hiện 1 phép cộng.

Xem xét phép cộng trong các trường hợp các số là 2 số dương, 1 số âm và 1 số dương và 2 số âm. Giả sử các số được biểu diễn là số 8 bit

```
  00001010    (+10)10                    00000101    (+5)10
+ 00010111    (+23)10                  + 11111110    (−2)10
------------                           ------------
  00100001    (+33)10    Cờ cần loại bỏ →(1) 00000011    (+3)10

                           11111111    (−1)10
                         + 11111100    (−4)10
                           --------
Cờ cần loại bỏ →(1)        11111011    (−5)10
```

Với 3 ví dụ về phép cộng ở trên, ta thấy nguyên tắc chung của phép cộng là rất quen thuộc vì đã được trình bày trong chương 2. Ở đây chỉ có một vấn đề duy nhất cần bàn luận đó là bit cờ được loại bỏ trong phép cộng giữa 1 số âm với 1 số dương và giữa 2 số âm. Mặc dù bit cờ đã được set up lên 1, nhưng điều này không có nghĩa là phép toán của chúng ta đã thực hiện sai. 2 ví dụ ở trên đã chứng minh rất rõ rằng mặc dù bit cờ là 1 nhưng kết quả lại hoàn toàn đúng.

Trong trường hợp nào kết quả là chấp nhận được khi bit cờ bằng 1

Khi tiến hành phép cộng 2 số dương có giá trị lớn và cùng dấu, hiện tượng tràn dấu sẽ xảy ra khi số lượng các bit dữ liệu biểu diễn con số không đủ lớn. Ví dụ khi ta cộng 2 số +80 và +50 được biểu diễn bởi số 8 bit có dấu. Về mặt toán học ta sẽ có kết quả là +130 nhưng máy tính sẽ cho ta kết quả là -126

```
  01010000    (+80)10
+ 00110010    (+50)10
------------
  10000010    (−126)10
```

Điều này hoàn toàn không ngạc nhiên vì số 8 bit có dấu chỉ có giá trị lớn nhất là +127. Mặc dù chuỗi số 10000010 có giá trị đúng là 130 nhưng kết quả không được chấp nhận bởi lập trình viên.

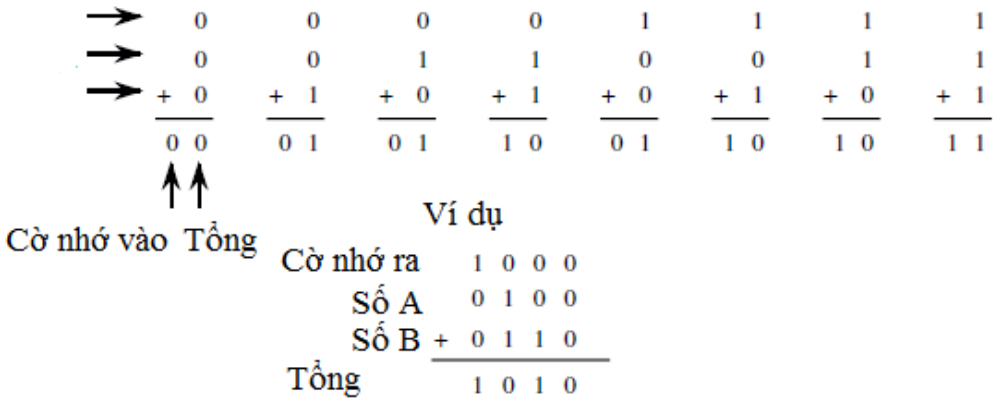
Để phân biệt khi nào thì kết quả có thể được chấp nhận, chúng ta có 2 định nghĩa sau đây

- *Khi cộng 2 số cùng dấu, nếu kết quả có dấu ngược lại đồng thời có hiện tượng tràn dấu thì kết quả đó là sai. Nếu cộng 2 số khác dấu, hiện tượng kết quả sai do tràn dấu sẽ không bao giờ diễn ra*

- *Nếu một số âm trừ đi một số dương và kết quả là dương hoặc một số dương trừ đi một số âm và kết quả là âm thì hiện tượng tràn dấu đã xảy ra và kết quả là không chính xác*

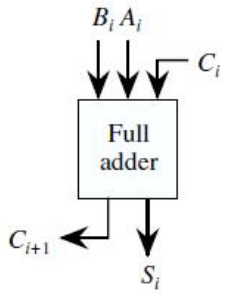
3.2.2. Mạch phần cứng của phép cộng và trừ

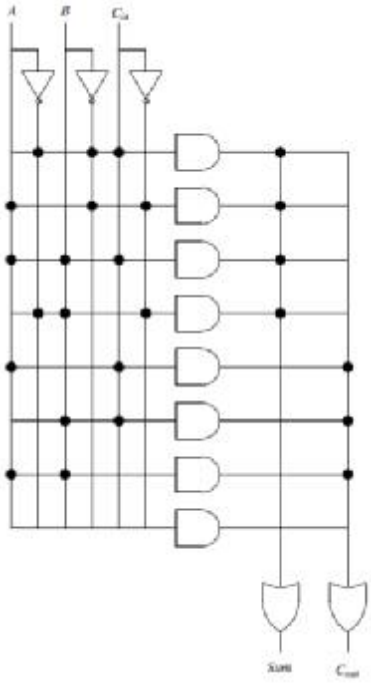
Nguyên lý của phép cộng



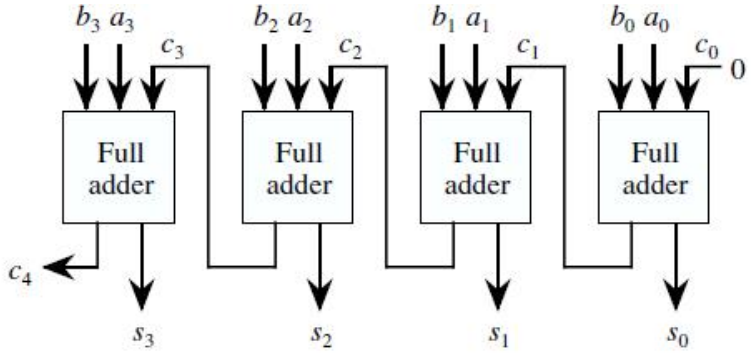
Bảng chân lý và mạch mô tả dùng PLA

A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

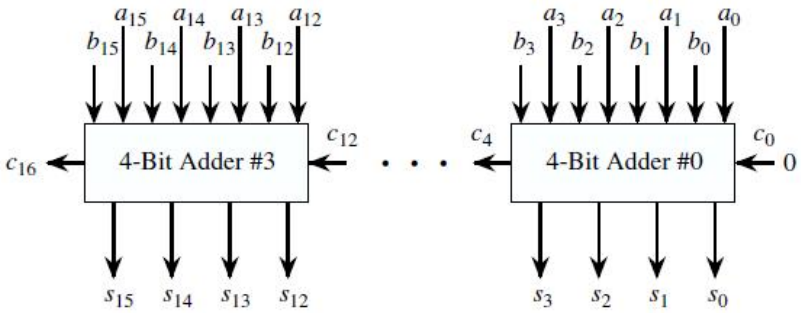




Mạch cộng 4 bit có nhớ

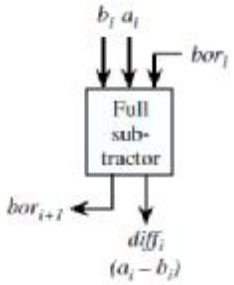


Tổng hợp mạch cộng 16 bit từ mạch cộng 4 bit

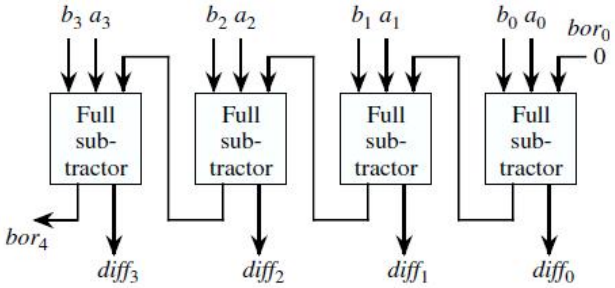


Tương tự ta có thể thiết kế được mạch trừ với bảng chân lý và mô hình như sau

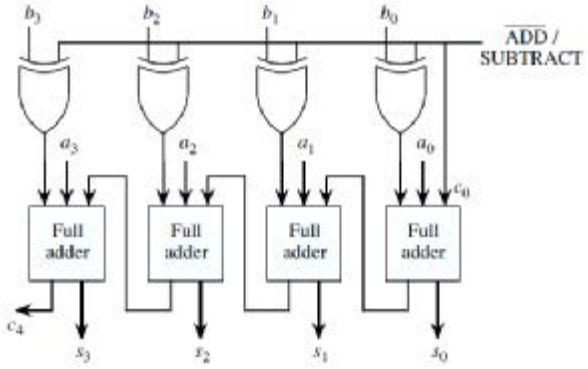
a_i	b_i	bor_i	$diff_i$	bor_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1



Bộ trừ 4 bit có nhớ



Với phân tích về mạch trừ với bản chất là phép cộng với số trừ được chuyển sang số âm, ta có thể thiết kế được mạch vừa thực hiện cả phép cộng và trừ như dưới đây



Trong mạch trên, khi thực hiện phép cộng, tín hiệu chân $\overline{ADD}/SUBTRACT$ có tín hiệu là 0. Khi đó 4 phần tử XOR không có tác dụng trong mạch. Các chân b_i lần sẽ đưa trực tiếp vào các bộ cộng có nhớ. Phần tử nhớ C_0 là 0. Khi thực hiện phép tính trừ, tín hiệu chân $\overline{ADD}/SUBTRACT$ có tín hiệu bằng 1. 4 phần tử XOR có chức năng đảo các bit của các tín hiệu bi hình thành nên số bù 1. Tín hiệu C_0 là 1 được đưa vào bộ nhớ có tác dụng để chuyển số bù 1 sang số bù 2 tức là số âm của B. Như vậy, mạch sẽ thực hiện phép toán A + (-B)

3.2.3. Phép cộng và trừ đối với số bù 1

Mặc dù không còn được sử dụng một cách rộng rãi nhưng phép toán với số bù 1 đã từng được sử dụng trong những ngày đầu tiên của máy tính điện tử. Phép toán đối với số bù 1 khác một chút đối với số bù 2 là bit tràn không bị loại bỏ mà nó được sử dụng trong phép toán. Cách sử dụng bit nhớ này được gọi là "làm tròn bit nhớ". Phương pháp làm tròn bit nhớ được trình bày với 2 ví dụ dưới đây, áp dụng cho 2 phép toán là số nguyên và số không nguyên

```
  10011   (–12)10
+ 01101   (+13)10
-------
 100000
      └──► Làm tròn bit nhớ
+     1
-------
  00001   (+1)10
```

```
   0101.1   (+5.5)10
+  1110.0   (–1.0)10
-----------
   10011.1
+      └──► 1.0
-----------
   0100.1   (+4.5)10
```

Tại sao lại cần phải có bước "làm tròn bit nhớ". Để trả lời câu hỏi này chúng ta lật lại 2 vấn đề mà chúng ta đã biết trong các chương trước.

- Số bù 1 có 2 giá trị chuỗi số biểu diễn số +0 và -0. Về mặt giá trị chúng hoàn toàn bằng nhau nhưng về cách biểu diễn thì chúng cách nhau 1 đơn vị. Do đó ta cân phải bù giá trị này trong phép toán
- Với phép cộng 2 số cùng dấu dương và kết quả chưa vượt qua khả năng biểu diễn của chuỗi số nhị phân, hiện tượng tràn sẽ không xảy ra, kết quả vẫn đúng. Nếu hiện tượng tràn xảy ra, kết quả sai. Khi cộng 2 số âm, hoặc giữa 1 số âm và 1 số dương, hiện tượng tràn xảy ra, việc làm tròn bít nhớ khắc phục hiện tượng 2 số +0 và -0

3.3. Phép nhân và chia với số dấu phẩy tĩnh

Phép nhân và chia với số dấu phẩy tĩnh có thể được phân tích thành tổ hợp các phép toán cộng, trừ và dịch. Trong phần này, chúng ta sẽ phân tích kỹ phương pháp thực hiện phép toán nhân và chia đối với số dấu phẩy tĩnh, đầu tiên là với số không dấu rồi sau đó là số có dấu

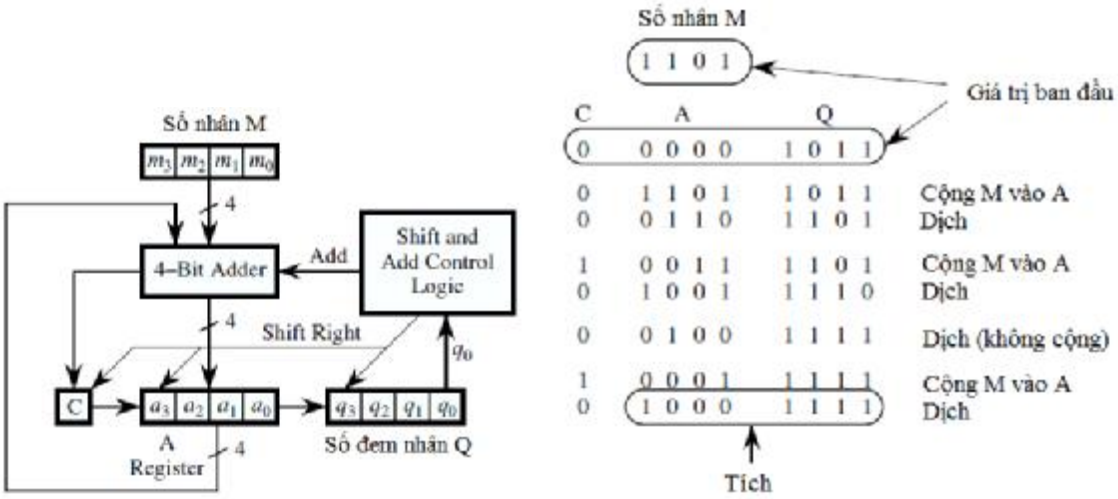
3.3.1. Phép nhân số không dấu

Quá trình thực hiện nhân số không dấu được thực hiện tương tự như đối với số thập phân

1 1 0 1	$(13)_{10}$	Số nhân M
× 1 0 1 1	$(11)_{10}$	Số đem nhân Q
1 1 0 1		
1 1 0 1		Quá trình nhân
0 0 0 0		
1 1 0 1		
1 0 0 0 1 1 1 1	$(143)_{10}$	Tích P

Phép nhân được thực hiện bằng cách lần lượt dịch sang trái số nhân M, các kết quả của phép dịch này được thực hiện cộng với nhau hay không tùy thuộc vào giá trị của bit tương ứng trong số đem nhân Q. Kết quả cuối cùng P là kết quả của phép nhân. Hai số n bit nhân với nhau sẽ cho ta kết quả là một số 2n bit. Khi thực hiện phép nhân đối với 2 số n bit có dấu, kết quả có độ lớn là số 2n-1 bit trong đó có 1 bit dấu

Sơ đồ nguyên lý mạch phần cứng

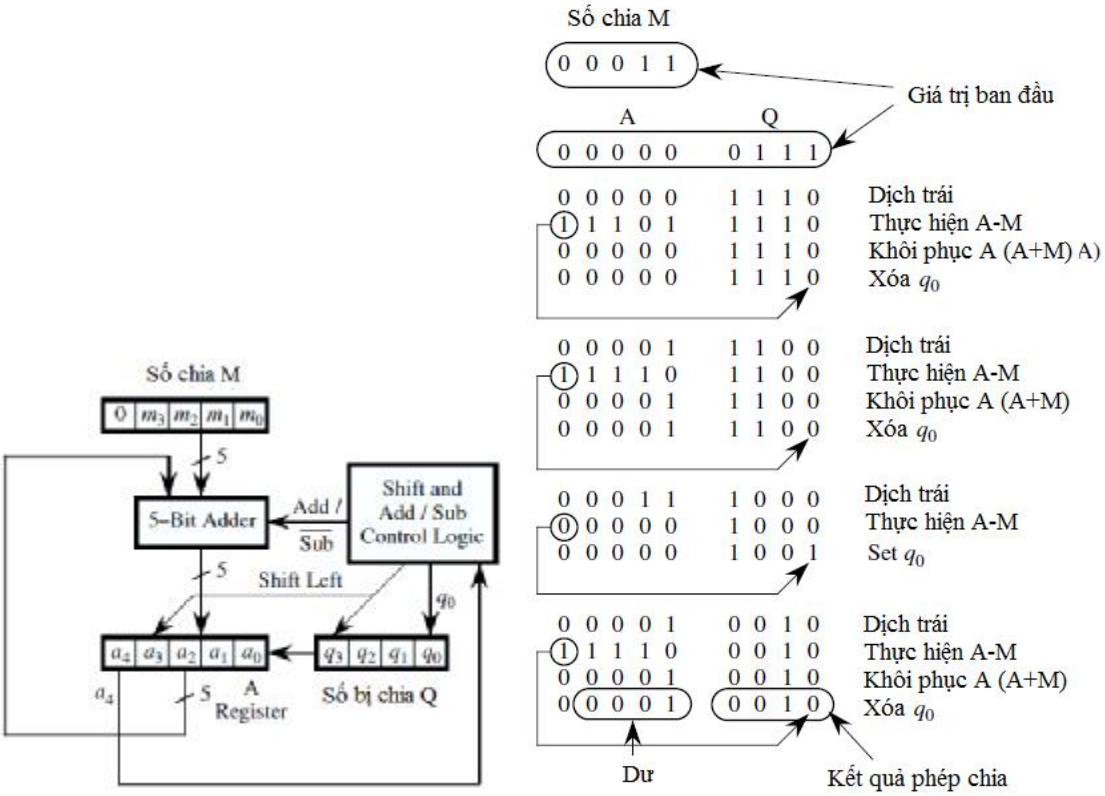


Mạch phần cứng thực hiện phép tính cộng tương tự như nguyên tắc thực hiện phép toán cộng. Phần cứng của mạch bao gồm 1 bộ cộng 4 bit và 3 thanh ghi có độ dài 4 bit và 1 thanh ghi tràn 1 bit. Để thực hiên phép nhân, số nhân M được đưa vào thành ghi M, số đem nhân được đưa vào thanh ghi Q, nội dung các thanh ghi A và C còn lại được đặt ở giá trị 0. Trong quá trình thực hiện phép cộng, giá trị của bit tận cùng bên phải của thanh ghi Q sẽ quyết định có thực hiện phép cộng giá trị M vào bộ cộng hay không. Sau khi thực hiện phép cộng đó, cờ nhớ C, số đem nhân Q và thanh ghi A được dịch sang phải 1 bit. Điều này có hiệu ứng tương ứng như đẩy số nhân M dịch sang phải tương ứng với quá trình nhân 2 số bằng tay, đồng thời đưa các bit tiếp theo trong thanh ghi Q vào bộ điều khiển dịch và cộng để thực hiện phép toán

3.3.2. Phép chia số không dấu

Trong quá trình thực hiện phép chia số nhị phân, ta phải cố gắng trừ số chia M từ số bị chia Q bằng cách sử dụng ít nhất số bit trong số bị chia. Nguyên lý của phép chia ngược lại hoàn toàn với phép nhân. Trong phép chia, thay vì dịch sang phải trong phép nhân, ta sẽ thực hiện dịch sang trái; thay vì thực hiện phép cộng, ta sẽ thực hiện phép trừ. Nếu kết quả của phép trừ là âm (do tràn số), ta sẽ thực hiện khôi phục lại giá trị trước khi trừ, dịch tiếp rồi tiếp tục phép trừ đề thực hiện tiếp phép chia.

Sơ đồ mạch phần cứng



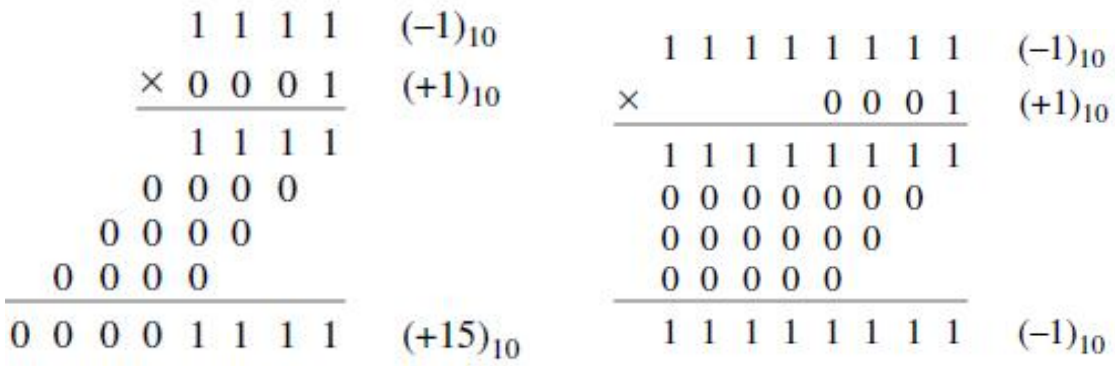
Cấu tạo của bộ chia bao gồm 1 bộ cộng 5 bit, bộ điều khiển, thanh ghi 4 bit lưu trữ số bị chia Q, 2 thanh ghi 5 bit lưu trữ số chia M và phần dư kết quả phép chia A. Trong mạch này, A và M là các thanh ghi 5 bit vì ta cần sử dụng bit có trọng số lớn nhất để xác định dấu của kết quả phép trừ, tức là kiểm tra xem phép trừ có được thực hiện hay không

Để thực hiện chia 2 số, số bị chia được đặt trong thanh ghi Q, số chia đặt trong thanh ghi M, thanh ghi A và các bit cao nhất của M được xóa về 0. A và Q được dịch sang trái 1 bit rồi lấy giá trị đó trừ đi M. Kết quả có bit tận cùng bên trái có giá trị là 1, A sẽ được khôi phục và bit tận cùng bên phải của Q bị xóa về 0 (tức là kết quả phép chia của bit này là 0 vì phép trừ không hoàn thành). Nếu bit tận cùng bên trái của A có giá trị 0, phép trừ đã thực hiện thành công, tức là phép chia sẽ có kết quả là 1, khi đó q_0 được set lên 1. Ta có thể xem xét các bước đã diễn ra ở ví dụ thực hiện phép chia với số chia là 3, số bị chia là 7 ở lưu đồ trên

3.3.3. Phép nhân và chia số có dấu

Trong trường hợp số thực hiện phép nhân đều là số dương, việc thực hiện phép toán sẽ không có gì khác như trong trường hợp số không dấu.

Trường hợp nhân một số âm với một số dương, ta hãy xét ví dụ nhân số +1 với số -1-



Kết quả đúng phải là -1 nhưng ở đây, kết quả có giá trị là +15. Nguyên nhân của hiện tượng này là bit dấu không được mở rộng về phía trái đủ để biểu diễn bit dấu. Để khắc phục hiện tượng này, một phần của các phần tử thực hiện phép nhân sẽ được mở rộng độ lớn bằng với kết quả. Ta thấy ngay giá trị của phép nhân đã đúng trong hình biểu diễn bên phải.

Phép chia được thực hiện phức tạp hơn rất nhiều. Chúng ta sẽ không tìm hiểu thêm về nó nữa. Tuy nhiên, về mặt nguyên tắc chung, chúng ta thực hiện phép chia số có dấu bằng cách chuyển nó sang số dương, thực hiện phép chia rồi chuyển trở lại số có dấu.

3.4. Phép toán với số dấu phẩy động

Phép toán đối với số có dấu phẩy động có thể được suy luận từ phép toán với số có dấu phẩy tĩnh mà thuật toán được tìm hiểu ở mục 3.3. Ở mục tiếp theo này, chúng ta sẽ tìm hiểu làm thế nào để có thể thực hiện phép toán dấu phẩy tĩnh với các số cơ số 2 và cơ số 10

3.4.1. Phép cộng và trừ số dấu phẩy động

Phép toán số dấu phẩy động khác với phép toán số nguyên vì biểu diễn số dấu phẩy động có 2 trường quan trọng là số mũ và trường độ lớn. Để thực hiện được phép toán, trường số mũ cần có giá trị bằng nhau ở cả 2 toán hạng. Với 2 số cơ số 10 thông thường, số mũ ở 2 toán hạng được chuẩn hóa sao cho có giá trị

bằng nhau. Sau đó, việc thực hiện phép toán cộng và trừ được thực hiện trong trường độ lớn. Phương pháp này đơn giản, tuy nhiên hạn chế của nó là kết quả có thể có những sai số. Để phân tích hiện tượng này, ta hãy tính phép tính (.101 x 2^3 + .111 x 2^4) với độ lớn của trường giá trị là 3 con số. Đầu tiên chúng ta điều chỉnh số mũ của số bé hơn cho bằng với số mũ của số lớn hơn. Do đó chúng ta có .101 x 2^3 = .010 x 2^4. Việc này làm mất giá trị .001 x 23 của toán hạng này. Kết quả phép cộng

$$(.010 + .111) \times 2^4 = 1.001 \times 2^4 = .1001 \times 2^5$$

Việc làm tròn tiếp tục diễn ra vì trường độ lớn chỉ có 3 con số, và chúng ta lại mất đi 0.001 x 2^4 nữa

3.4.2. Phép nhân và chia số dấu phẩy động

Phép nhân và chia số dấu phẩy động được thực hiện tương tự như phép cộng và trừ, ngoại trừ các trường dấu, độ lớn và trường số mũ sẽ được thực hiện một cách độc lập. Nếu các toán hạng có cùng dấu, kết quả thu được sẽ là số dương. Nếu khác dấu, kết quả là số âm. Số mũ sẽ được thực hiện chuẩn hóa bằng cách thực hiện phép cộng đối với phép nhân và thực hiện phép trừ đối với thực hiện phép chia. Độ lớn của kết quả có được bằng phép nhân hoặc chia phụ thuộc vào phép toán mà ta thực hiện. Giá trị này được thực hiện một cách bình thường như là số có dấu phẩy tĩnh.

Ví dụ như với hệ thống máy tính có trường độ lớn được biểu diễn bằng 3 bit. Ta thực hiện phép toán (+.101 x 2^2) x (-.110 x 2^{-3}). 2 toán hạng này có dấu khác nhau nên kết quả có dấu là số âm. Bây giờ ta sẽ thực hiện phép cộng số mũ để thực hiện phép chia, và do đó ta có số mũ của kết quả là 3 + -3 = -1. Tiếp theo ta sẽ nhân trường độ lớn với nhau, kết quả là .01111. Sau đó chuẩn hóa đối với hệ thống 3 bit, ta sẽ có kết quả phép nhân là $-.111 \times 2^{-2}$.

TỔNG KẾT CHƯƠNG

Các phép toán trong máy tính có thể được thực hiện tương tự như cách mà chúng ta tính toán trong cơ số 10 bằng tay, chỉ khác là chúng ta sẽ thực hiện đối với cơ số mà máy tính sử dụng. Trong máy tính, số bù 2 và số bù 10 được sử dụng để biểu diễn các phép toán với số nguyên. Trong khi đó, số có dấu được sử dụng để biểu diễn độ lớn các số âm và dương trong các phương pháp biểu diễn chuẩn

CHƯƠNG 4: NGÔN NGỮ MÁY VÀ HỢP NGỮ

Trong chương này chúng ta sẽ giải quyết nội dung trọng tâm của cấu trúc máy tính, đó là loại ngôn ngữ mà máy tính có thể hiểu được thường được gọi là *ngôn ngữ máy*. Ngôn ngữ máy thường được thảo luận nhiều nhất là *hợp ngữ*. Đây là loại ngôn ngữ mà chức năng của nó tương đương với ngôn ngữ máy ngoại trừ việc nó sử dụng các lệnh gần gũi với ngôn ngữ thông thường. Chẳng hạn như lệnh cộng nội dung thanh ghi r0 với thanh ghi r1 và ghi kết quả vào thanh ghi r2 "`ADD r0, r1, r2`" sẽ dễ hiểu và trực quan hơn chuỗi lệnh tương ứng "0110101110101101"

Chúng ta sẽ bắt đầu bằng việc mô tả tập lệnh ISA (**Intruction Set Architecture**) bằng cách xem xét các lệnh và toán hạng của nó. Ngôn ngữ ISA này tương ứng với cấp hợp ngữ được thể hiện trong hình 1.4 chương 1. Nó là cấp ở giữa cấp ngôn ngữ cấp cao (tại cấp này, lập trình viên không cần quan tâm đến cấu trúc phần cứng máy tính)và cấp điều khiển hoạt động (tại cấp này, nội dung của các thanh ghi điều khiển tác động trực tiếp đến hoạt động của máy tính)

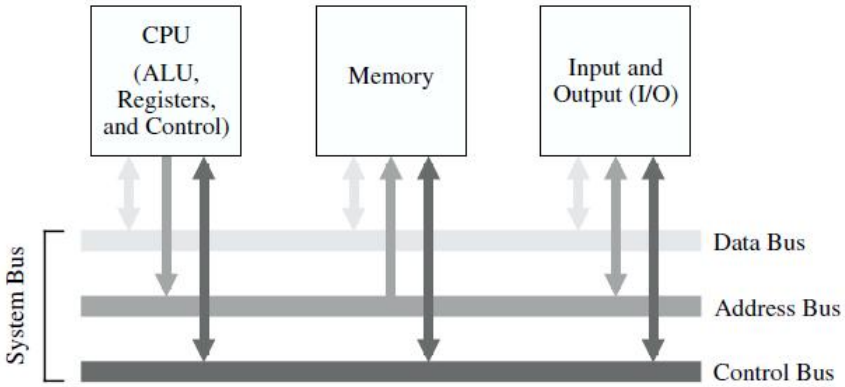
Để mô tả bản chất của hợp ngữ, chúng ta sẽ nghiên cứu mô hình máy ARC, một mô hình đơn giản của kiến trúc máy tính SPARC là mô hình thông dụng của máy tính Sun

Chúng ta sẽ minh họa các chức năng khác nhau của các lớp câu lệnh bằng các ví dụ về chương trình hợp ngữ

4.1. Các thành phần cứng của cấu trúc tập lệnh

Một máy tính theo mô hình ISA dưới cái nhìn của một lập trình viên bao gồm tất cả các phần cứng, các lệnh và các dữ liệu mà chúng ta có thể truy cập được. Trong phần này, chúng ta sẽ xem xét cấu trúc phần cứng của máy tính dưới con mắt của một lập trình viên. Chúng ta sẽ bắt đầu bằng việc thảo luận về cấu trúc của máy tính, đó là CPU và tương tác của nó với bộ nhớ chính, đó là sự xuất nhập dữ liệu của máy tính với bên ngoài

4.1.1. Nhắc lại về mô hình cấu trúc bus



Hình 4.1. Mô hình cấu trúc bus

Hình 4.1. nhắc lại cho chúng ta cấu trúc của mô hình cấu trúc bus mà chúng ta đã xem xét trong chương 1. Mục đích của hệ thống bus là giảm thiểu các liên kết giữa CPU và các thành phần của nó. Thay vì việc sử dụng các mối liên kết riêng rẽ giữa bộ nhớ và cổng ngoại vi I/O, CPU được kết nối với các thành phần đó bằng hệ thống bus. Trong một vài cấu trúc khác phức tạp hơn, CPU sẽ sử dụng các hệ thống bus riêng biệt để kết nối các thành phần đó

Không phải là tất cả các thành phần hệ thống được kết nối với nhau theo cùng một phương thức. CPU tạo ra các địa chỉ và được đưa lên bus địa chỉ, và bộ nhớ nhận tín hiệu địa chỉ này từ bus địa chỉ đó. Bộ nhớ không tạo ra các tín hiệu địa chỉ và CPU không bao giờ nhận các tín hiệu địa chỉ, và do đó không có một sự tương ứng trong các đường bus đó

Thông thường, người sử dụng sẽ viết các chương trình bằng ngôn ngữ cấp cao, sau đó được chuyển sang hợp ngữ. Chương trình biên dịch sẽ chuyển tiếp chương trình hợp ngữ này thành mã máy được lưu trữ trên ổ đĩa. Trước khi thực thi chương trình, mã máy sẽ được nạp từ ổ đĩa lên bộ nhớ chính dưới sự điều khiển của hệ điều hành

Trong quá trình thực thi các lệnh của chương trình, từng lệnh sẽ được nạp vào ALU từ bộ nhớ một cách lần lượt cùng với dữ liệu tương ứng cần được xử lý của lệnh đó. Kết quả của chương trình sẽ được đưa đến các thiết bị như màn hình hay ổ đĩa. Tất cả các hoạt động đó được phối hợp bởi bộ điều khiển mà chúng ta sẽ nói kỹ hơn trong chương 6.

Điều quan trọng chúng ta cần nhớ rằng các lệnh được xử lý bởi ALU mặc dù tất cả các lệnh và dữ liệu của nó được lưu trữ trên bộ nhớ. Điều đó có nghĩa là các lệnh và dữ liệu cần được nạp từ bộ nhớ vào các thanh ghi của ALU, và kết quả sẽ được lưu ngược trở lại vào bộ nhớ.

4.1.2. Bộ nhớ

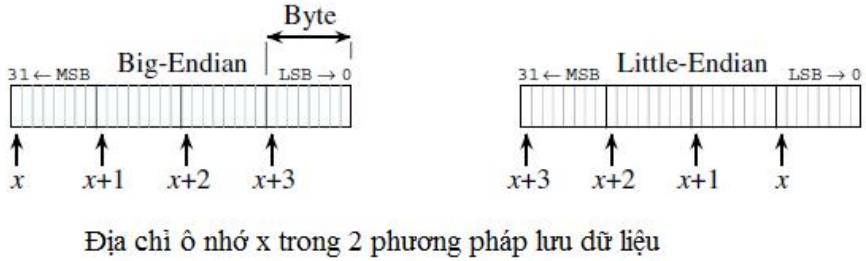
Bộ nhớ của máy tính bao gồm một tập hợp một số các thanh ghi được đánh số thứ tự hay còn gọi là đánh địa chỉ, mỗi địa chỉ có thể lưu trữ 1 byte có độ lớn 8 bit. Một nibble hay đôi khi còn được gọi là nybble được định nghĩa là 4 bit liền kề nhau. Ý nghĩa của các khái niệm bit, byte, và nibble nói chung không phụ thuộc và cấu trúc của máy tính nhưng ý nghĩa của word sẽ phụ thuộc vào từng cấu trúc cụ thể. Một word thông thường có thể là 16, 32, 64 và 128 bit, với word 32 bit được xem là dạng chuẩn của máy tính. Hình 4.2 liệt kê các dạng dữ liệu của máy tính

Bit	0
Nibble	0110
Byte	10110000
16-bit word (halfword)	11001001 01000110
32-bit word	10110100 00110101 10011001 01011000
64-bit word (double)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101
128-bit word (quad)	01011000 01010101 10110000 11110011 11001110 11101110 01111000 00110101 00001011 10100110 11110010 11100110 10100100 01000100 10100101 01010001

Hình 4.2. Các dạng dữ liệu máy tính

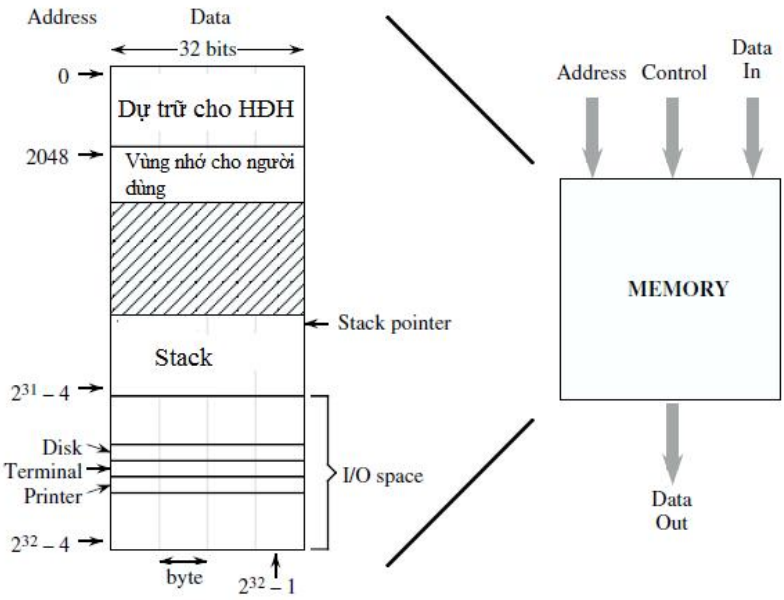
Trong máy tính đánh địa chỉ theo byte, dữ liệu nhỏ nhất mà nó có thể tác động trực tiếp trong bộ nhớ là byte. Tuy nhiên, thông thường các lệnh sẽ đọc và ghi một vài byte liên tiếp. Những cụm byte như vậy sẽ được lưu trữ liên tiếp trong bộ nhớ, được đánh địa chỉ bởi byte có địa chỉ thấp nhất. Hầu hết các máy tính ngày nay có thể truy cập theo byte, half word, và double word.

Khi sử dụng cụm byte, có 2 phương pháp chính để lưu dữ liệu vào trong bộ nhớ: byte có trọng số lớn được lưu vào địa chỉ thấp, được gọi là **big-endian**, hoặc là byte có trọng số nhỏ được lưu vào địa chỉ thấp, được gọi là **little-endian.**



Hình 4.3. Phương pháp lưu dữ liệu trong bộ nhớ

Các ô nhớ trong bộ nhớ được đánh địa chỉ tuyến tính như thể hiện trên hình 4.3. Mỗi địa chỉ duy nhất tương ứng với một từ đặc biệt gồm 4 byte. Các địa chỉ được đánh tăng dần bắt đầu từ 0 và địa chỉ cao nhất được xác định là dung lượng của bộ nhớ trừ 1 đơn vị. Địa chỉ cao nhất của bộ nhớ 2^{32} byte là 2^{32}-1.

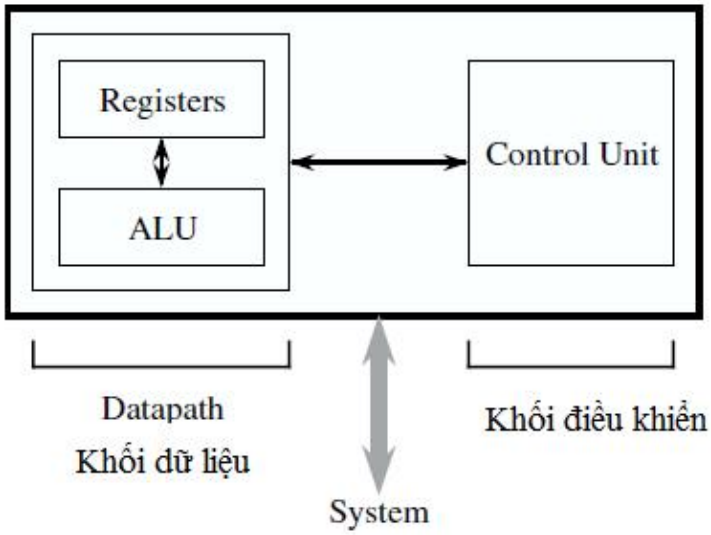


Hình 4.4. Tổ chức bộ nhớ trên máy tính

Hình 4.4. mô tả tổ chức bộ nhớ trên máy tính. Máy tính trong ví dụ này có không gian địa chỉ 32 bit, có nghĩa là chương trình có thể truy cập vào bất cứ ô nhớ nào trong không gian địa chỉ từ 0 đến 2^{32}-1. Không gian địa chỉ trong máy tính sẽ được chia thành các vùng có chức năng riêng biệt đó là các vùng dành cho hệ điều hành, các cổng ngoại vi I/O, người sử dụng và stack. Cách phân vùng giữa các máy tính khác nhau cũng khác nhau. Điều đó lý giải tại sao chương trình thích hợp cho những dạng vi xử lý này lại không thích hợp đối với những dạng vi xử lý khác.

4.1.3. CPU

Chúng ta vừa tìm hiểu cấu trúc cơ bản nhất của máy tính là hệ thống bus, bộ nhớ, bây giờ chúng ta sẽ tìm hiểu cấu trúc bên trong của CPU. Một cách đơn giản nhất, CPU bao gồm khối dữ liệu bao gồm các thanh ghi chứa và bộ ALU, và một khối điều khiển được sử dụng để biên dịch các lệnh và tác động đến các thanh ghi dịch chuyển



Hình 4.5. Cấu trúc đơn giản của CPU

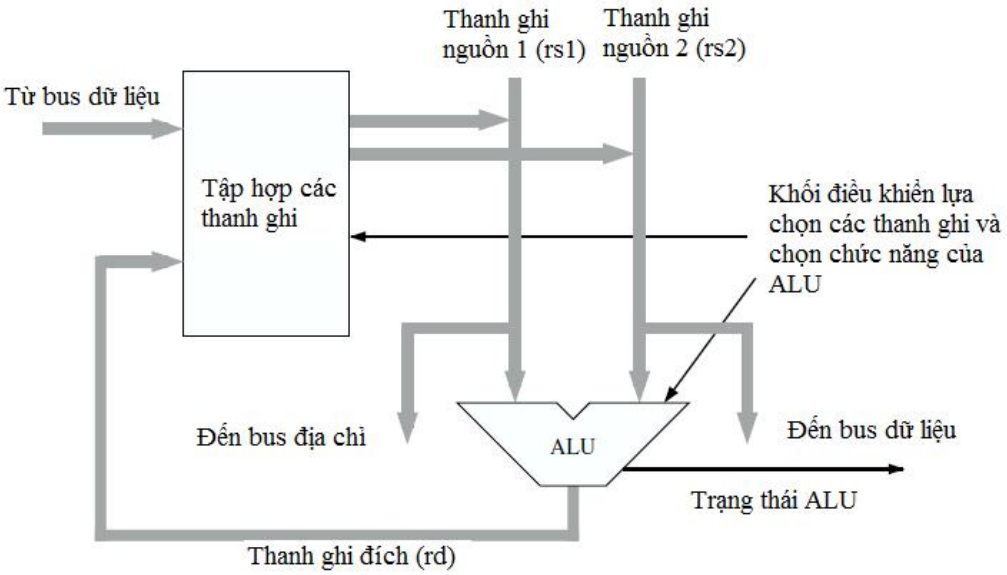
Khối điều khiển của máy tính có trách nhiệm thực thi các lệnh của chương trình được lưu trên bộ nhớ (ta sẽ tìm hiểu kỹ hơn trong chương 9). Có 2 thanh ghi liên kết giữa khối điều khiển và khối dữ liệu là thanh ghi đếm chương trình PC (Program Counter) và thanh ghi lệnh IR (Instruction Register). Thanh ghi PC chứa địa chỉ của lệnh đang được thực thi trong ALU. Lệnh này sẽ được nạp từ bộ nhớ, được lưu trong thanh ghi IR và được biên dịch tại đó. Các bước thực thi 1 lệnh được chỉ ra bởi các bước dưới đây

1. Nạp lệnh kế tiếp sẽ được thực thi từ bộ nhớ
2. Giải mã lệnh
3. Đọc các toán hạng từ bộ nhớ chính nếu cần
4. Thực thi lệnh và lưu trữ kết quả
5. Quay về bước đầu tiên

Quá trình thực hiện trên được biết đến bằng cái tên **chu kỳ lệnh**.

Ngoài ra, khối điều khiển còn có trách nhiệm phối hợp các thành phần khác nhau để thực thi lệnh. Có thể coi rằng có một "máy tính" nhỏ bên trong máy tính lớn để đảm nhiệm mọi hoạt động của CPU. Chúng ta sẽ tìm hiểu kỹ hơn vấn đề này trong chương 6

Mô hình điều khiển luồng dữ liệu



Hình 4.6. Một mô hình luồng dữ liệu

Luồng dữ liệu là mối liên kết giữa các thanh ghi trong CPU và khối ALU. Hình 4.6. là một mô hình luồng dữ liệu của máy tính. Các thanh ghi trong mô hình này có thể được coi là một vùng nhớ truy cập nhanh, độc lập với vùng nhớ của hệ thống, được sử dụng để lưu trữ tạm thời trong quá trình tính toán. Kích cỡ của hệ thống thanh ghi này có thể từ vài thanh ghi cho tới vài nghìn thanh ghi. Cũng giống như bộ nhớ, các thanh ghi được đánh địa chỉ tăng dần bắt đầu từ 0, tuy nhiên không gian địa chỉ của các thanh ghi nhỏ hơn nhiều so với không gian bộ nhớ. Sự khác biệt lớn nhất giữa hệ thống thanh ghi và bộ nhớ hệ thống là hệ thống thanh ghi nằm trên CPU, nên hoạt động của nó nhanh hơn. Dữ liệu dịch chuyển giữa các thanh ghi nhanh hơn khoảng 10 lần so với dịch chuyển dữ liệu với bộ nhớ hệ thống.

Khối ALU có thể thực thi các lệnh có 1 hoặc 2 toán hạng. Các toán hạng được sử dụng sẽ được lựa chọn bởi khối điều khiển. Các toán hạng này được nạp từ các luồng dữ liệu được đánh dấu bởi nhãn “Register Source 1 (rs1)” và “Register Source 2 (rs2)”. Đầu ra từ ALU là luồng dữ liệu được đánh nhãn “Register Destination (rd)” được đưa ngược trở về hệ thống các thanh ghi. Trong hầu hết các hệ thống, các liên kết bao gồm cả các luồng dữ liệu đều có thể truy cập được

Tập lệnh

Tập lệnh là tập hợp các câu lệnh mà vi xử lý có thể thực thi được, do đó nó phải được định nghĩa bởi vi xử lý. Tập lệnh cho mỗi họ vi xử lý là hoàn toàn khác nhau. Chúng khác nhau ở số lượng các câu lệnh, các toán tử, toán hạng, thậm chí cả kết quả thu được từ vi xử lý cũng khác nhau. Sự không tương thích này lại hoàn toàn trái ngược với khả năng tương thích của các ngôn ngữ bậc cao như C, Pascal,...Chương trình viết bằng ngôn ngữ cấp cao có thể chạy được trên hầu hết các hệ thống vi xử lý khác nhau nếu được biên dịch lại cho phù hợp với hệ vi xử lý tương ứng.

Chúng ta sẽ xem xét một tập lệnh cụ thể trong chương này

Phần mềm tạo ra ngôn ngữ máy

Trình biên dịch (Compiler) là một chương trình máy tính được sử dụng để chuyển một chương trình được viết bằng ngôn ngữ cấp cao như C, Pascal, hay Fortran sang ngôn ngữ máy. Các trình biên dịch ở cùng một mức ngôn ngữ có cùng một dạng đầu vào nhưng chúng sẽ có những đầu ra khác biệt phụ thuộc vào chủng loại vi điều khiển. Ngoài ra, khi biên dịch cùng một chương trình cho cùng một loại vi xử lý, ta sẽ nhận được các kết quả khác nhau phụ thuộc vào các trình biên dịch khác nhau.

Việc sử dụng các trình biên dịch đã tạo thuận lợi lớn cho các lập trình viên. Đó là việc lập trình trở nên dễ dàng vì ngôn ngữ sử dụng gần với ngôn ngữ đời thường, không phải là chuỗi các con số 0 và 1 dễ nhầm lẫn. Tuy nhiên, khi lập trình bằng ngôn ngữ cấp cao, tốc độ xử lý của hệ thống bị ảnh hưởng do hệ thống tốn thời gian chuyển chương trình từ ngôn ngữ cấp cao thành ngôn ngữ hợp ngữ, sau đó lại phải chuyển tiếp sang ngôn ngữ máy. Các ngôn ngữ lập trình hiện nay cũng hỗ trợ việc lập trình kết hợp cả ngôn ngữ cấp cao và ngôn ngữ máy trong cùng một chương trình để tận dụng những ưu điểm của cả 2 loại ngôn ngữ cấp cao và cấp thấp.

Ngôn ngữ cấp cao cho phép chúng ta không cần quan tâm đến cấu trúc của máy tính trong quá trình lập trình. Ở cấp độ ngôn ngữ máy, để lập trình, sự hiểu biết về cấu trúc máy tính là điều thiết yếu. Nhưng nếu ta lập trình bằng ngôn ngữ cấp cao như C, Pascal, hay Fortran, chúng ta không cần quan tâm đến điều đó vì đã có trình biên dịch chuyển các chương trình đó thành các chương trình tương ứng với mỗi một hệ thống vi xử lý.

4.2. Máy tính ARC (A RISC Computer)

Để hiểu rõ hơn các nội dung của chương này, chúng ta sẽ nghiên cứu một mô hình cấu trúc dựa trên một kiến trúc vi xử lý mở rộng đã được thương mại hóa SPARC (Scalable Processoer Architecture) được phát triển bởi Sun Microsystems vào những năm 1980. SPARC nhanh chóng trở thành kiến trúc phổ biến từ khi được giới thiệu bởi đây là một hệ thống mở. Kiến trúc đầy đủ của SPARC được hoàn thiện vào khoảng năm 1992. Trong nội dung của chương này, chúng ta sẽ xem xét một thành phần của SPARC là máy tính dựa trên công nghệ RISC, máy tính ARC. ARC là chức năng quan trọng nhất của kiến trúc SPARC nhưng chức năng này lại không có trong các vi xử lý ngày nay

4.2.1. Bộ nhớ của ARC

Máy tính ARC là máy tính 32 bit có bộ nhớ đánh địa chỉ theo byte, tức là máy xử lý các dữ liệu có độ rộng 32 bit, nhưng dữ liệu được lưu trữ là dạng byte. Đồng thời, độ rộng của tín hiệu địa chỉ là 32 bit được đánh từ thấp đến cao. Hình 4.4 chính là tổ chức của bộ nhớ trong máy tính ARC, trong đó

- Dải địa chỉ 2^{11} = 2048 thấp nhất trong không gian nhớ được sử dụng làm vùng nhớ dự trữ cho hệ điều hành
- Vùng nhớ dành cho người sử dụng được dùng để nạp chương trình hợp ngữ. Vùng nhớ này bắt đầu từ địa chỉ 2048 đến khoảng stack của hệ thống
- Stack hệ thống bắt đầu từ địa chỉ 2^{31}-4 và giảm dần về những địa chỉ thấp hơn. Độ lớn của stack là không xác định trước vì thông thường nó phụ thuộc vào lập trình viên
- Dải địa chỉ từ 2^{31} đến 2^{32}-1 được sử dụng để đánh địa chỉ các cổng ngoại vi I/O, với mỗi cổng có địa chỉ hoàn toàn riêng biệt

Máy tính ARC có một vài dạng dữ liệu khác nhau như byte, half word, integer,... nhưng hiện tại, chúng ta sẽ chỉ xem xét đến dạng dữ liệu số nguyên 32 bit. Mỗi số nguyên được lưu trữ trong 4 byte trong bộ nhớ. ARC dựa trên kiến trúc **big-endian** tức là những byte có trọng số cao sẽ được đánh địa chỉ thấp. Địa chỉ lớn nhất có thể mà ARC có được là 2^{32}-1, do đó tín hiệu địa chỉ lớn nhất mà CPU đưa ra là 2^{32}-4

4.2.2. Tập lệnh của ARC

Trước khi tìm hiểu về tập lệnh của ARC, chúng ta sẽ xem xét kỹ hơn về các chức năng của CPU

- Máy tính ARC có 32 thanh ghi 32 bit đa mục đích, cùng với PC và IR
- Thanh ghi trạng thái vi xử lý PSR (Processor Status Register) chứa các thông tin về trạng thái hoạt động của CPU, bao gồm cả các thông tin về kết quả của phép toán trong ALU.
- Mỗi một lệnh được lưu trữ trên 1 word (32 bit)
- ARC là máy tính loại **load-store** tức là tại 1 thời điểm, nó chỉ cho phép nạp một giá trị vào 1 thanh ghi hoặc lưu 1 giá trị vào một địa chỉ trên bộ nhớ. Các toán hạng trước và sau khi được xử lý bởi ALU đều được lưu trữ trên các thanh ghi. Kiến trúc SPARC có khoảng 200 lệnh, tập lệnh của ARC cũng dựa trên nền lệnh đó. Hình 4.7 mô tả một tập con gồm 15 lệnh của ARC. Các lệnh được thể hiện dưới dạng biểu tượng với ý nghĩa riêng biệt

	Biểu tượng lệnh	**Ý nghĩa**
Tương tác bộ nhớ	**Ld**	Nạp nội dung thanh ghi từ bộ nhớ
	St	Cất nội dung thanh ghi vào bộ nhớ
Phép toán logic	**Sethi**	Set 22 bit cao của một thanh ghi
	Andcc	Thực hiện phép AND từng bit
	orcc	Thực hiện phép OR từng bit
	Orncc	Thực hiện phép NOR từng bit
	Srl	Dịch sang phải
Phép toán số học	**Addcc**	Cộng
Điều khiển luông chương trình	**Call**	Gọi chương trình con
	Jmpl	Nhảy và liên kết trở về từ chương trình con
	Be	Rẽ nhánh nếu bằng
	Bneg	Rẽ nhánh nếu nhỏ hơn 0
	Bcs	Rẽ nhánh nếu có cờ carry
	Bvs	Rẽ nhánh nếu tràn dữ liệu
	Ba	Luôn luôn rẽ nhánh

Hình 4.7. Một vài lệnh cơ bản của ARC

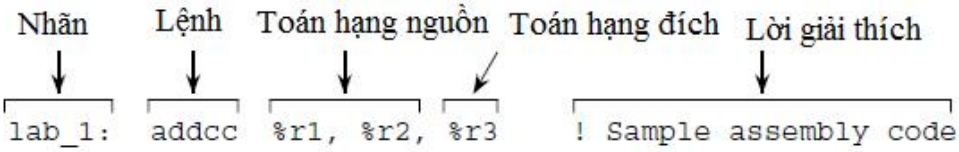
Nhóm lệnh dịch chuyển dữ liệu bao gồm 2 lệnh `ld` và `st` là 2 lệnh duy nhất cho phép truy cập vào bộ nhớ hệ thống. Lệnh `sethi` sẽ định lại giá trị của 22 bit cao trong 1 thanh ghi theo một giá trị định trước.

Nhóm lệnh toán học và lệnh logic gồm các lệnh `andcc, orcc, orncc` được sử dụng để thực hiện các phép toán AND, OR, NOR theo từng bit của các toán hạng. Một trong 2 toán hạng bắt buộc phải là 1 thanh ghi. Kết quả phép toán sẽ được đặt vào 1 thanh ghi. Lệnh dịch chuyển dữ liệu `srl` được sử dụng để dịch phải nội dung của một thanh ghi đồng thời sẽ thêm chuỗi các con số 0 vào những bit tận cùng phía bên trái. Lệnh `sra` (không được mô tả trong hình 4.7) sẽ dịch nội dung các bit trong thanh ghi sang bên phải, đưa nội dung các bit LSB quay trở về vị trí tương ứng thành các bit MSB.

Nhóm lệnh điều khiển bao gồm 2 lệnh chủ yếu là lệnh `call` và `jmpl`. Đây là cặp lệnh được sử dụng để gọi chương trình con và nhảy từ chương trình con trở về chương trình chính. Những lệnh như `be, bneg, bcs, bvs` và `ba` còn có tên là nhóm lệnh rẽ nhánh. Những lệnh này sẽ kiểm tra nội dung của thanh ghi trạng thái PSR và rẽ chương trình sang các nhánh tương ứng. Điều này có tác dụng giống như các lệnh `goto, if-then-else` hay `do-while` trong các ngôn ngữ bậc cao

4.2.3. Cú pháp lệnh của ARC

Mỗi một ngôn ngữ hợp ngữ đều có cú pháp riêng của nó. Chúng ta sẽ tìm hiểu cú pháp của hợp ngữ SPARC, thể hiện trên hình 4.8

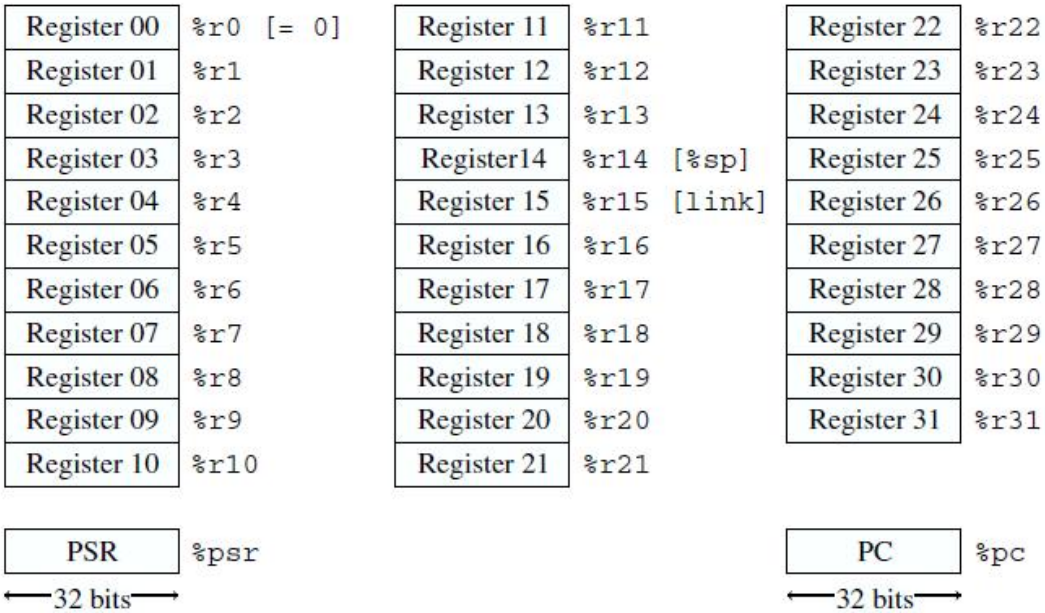


Hình 4.8. Cú pháp của ARC

Cú pháp bao gồm 4 thành phần: trường nhãn lệnh có thể không có, trường tên lệnh, các toán hạng nguồn và toán hạng đích nếu có, và cuối cùng là trường lời giải thích cho lệnh đó. Trường giải thích có ý nghĩa chủ yếu là để giúp người lập trình ghi nhớ các lệnh đã viết nên nó có thể có hoặc không.

Nhãn trong cú pháp của ARC có thể là một chuỗi ký tự bất kỳ, tuy nhiên nó phải thỏa mãn là không chứa các ký tự điều khiển, ký tự đặc biệt, hay trùng với một lệnh nào đó

Các thanh ghi toán hạng nguồn và đích trong ARC bao gồm 32 thanh ghi được ký hiệu từ r0 đến r31. 32 thanh ghi này có độ rộng dữ liệu là 32 bit trong đó



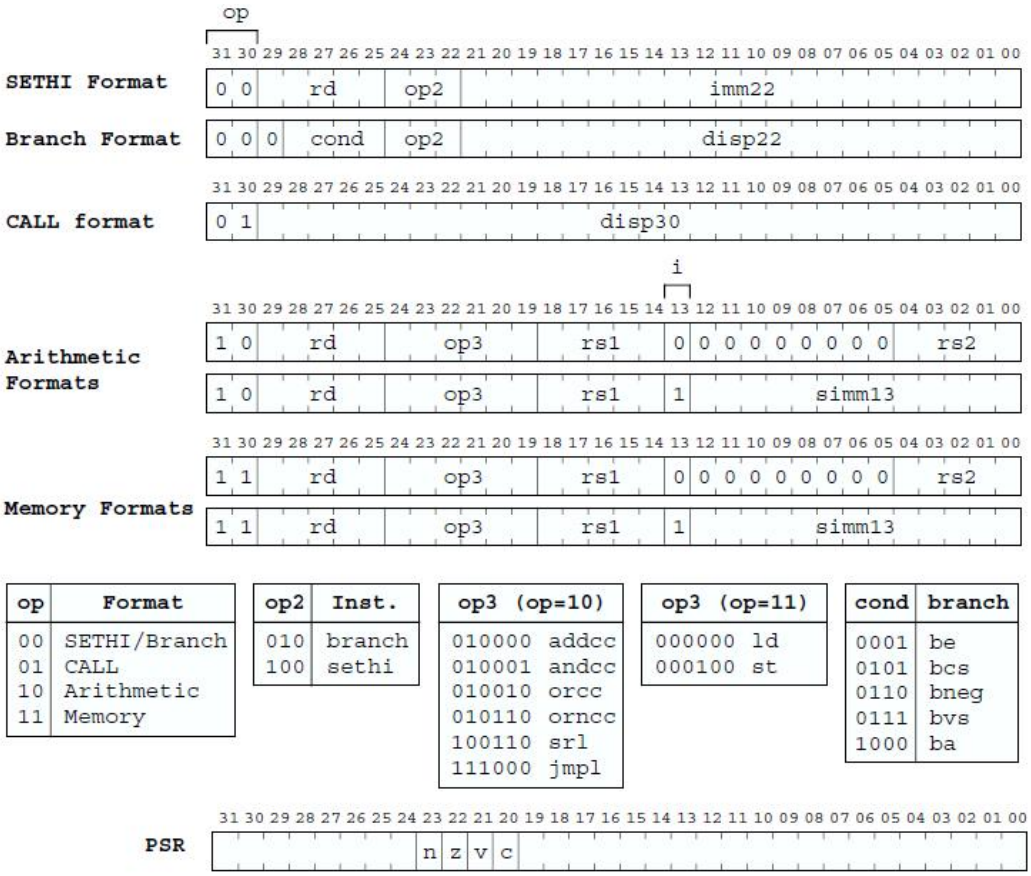
Hình 4.9. Các thanh ghi trong ARC mà người sử dụng có thể truy cập

thanh ghi số 0 `%r0` luôn có giá trị là 0 và giá trị này luôn luôn không đổi. Thanh ghi `%r14` và `%r15` có thêm chức năng là thanh ghi con trỏ stack và thanh ghi liên kết. Ngoài 32 thanh ghi này, trong ARC còn có thêm các thanh ghi PC là thanh ghi đếm chương trình, thanh ghi trạng thái PSR.

Các toán hạng trong cú pháp lệnh ARC được phân cách nhau bởi dấu phẩy. Toán hạng đích luôn là toán hạng đứng tận cùng phía bên tay phải.

4.2.4. Cấu trúc ngôn ngữ máy của ARC

Cấu trúc ngôn ngữ máy định nghĩa ý nghĩa của từng bit trong câu lệnh. Tức là nó định nghĩa từng trường trong câu lệnh, độ lớn của từng trường dữ liệu đó và tác động của từng bit lên hệ thống. Có tất cả 5 cấu trúc trong ARC là **SETHI, branch, call, arithmetic** và **memory**



Hình 4.10. Cấu trúc lệnh trong ARC

Mỗi một lệnh được biểu diễn bằng một tên riêng và một mã lệnh (Opcode). Mỗi một cấu trúc lệnh có thể có nhiều hơn một trường mã lệnh với mục đích là để thể hiện rõ ràng hơn nội dung, nhiệm vụ, và hoạt động của lệnh.

Cụ thể như sau

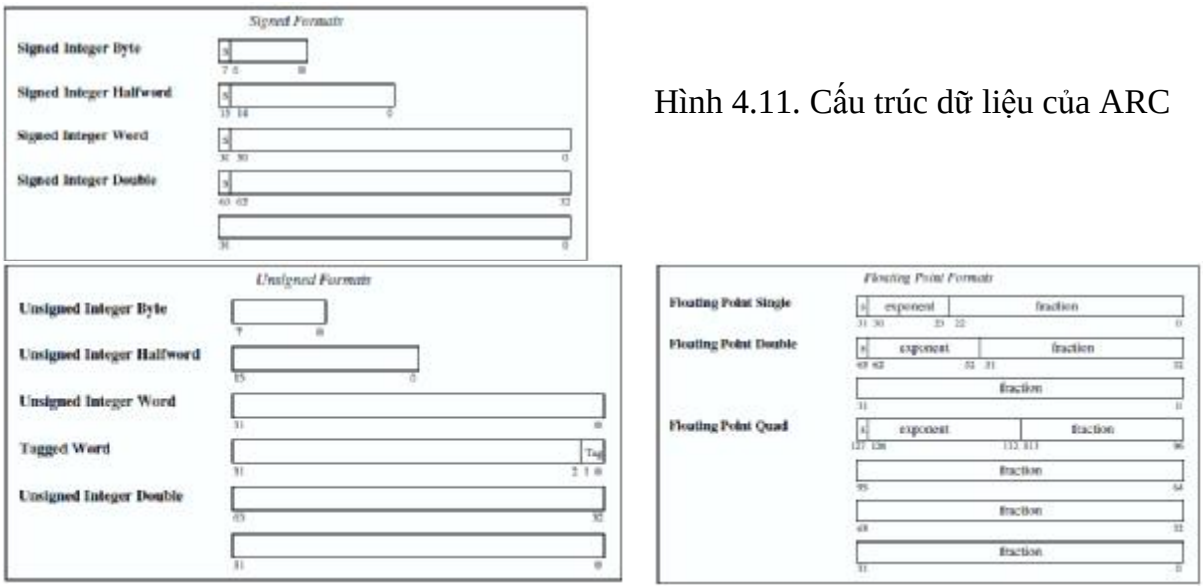
- 2 bit tận cùng phía bên trái của mỗi một lệnh là trường `op` (opcode) được sử dụng để nhận diện cấu trúc lệnh. Cấu trúc SETHI và Branch có cùng giá trị là 00 trong trường này. Cấu trúc SETHI và Branch được phân biệt với nhau ở trường `op2` trong đó giá trị `010` tương ứng với Branch và `100` tương ứng với cấu trúc SETHI. Ngoài ra bit thứ 29 trong cấu trúc Branch luôn luôn là 0. Trong khi đó, 5 bit `rd` trong cấu trúc SETHI chứa thanh ghi đích của lệnh.
- Trường `cond` trong cấu trúc Branch nhận diện loại lệnh rẽ nhánh, dựa trên mã điều kiện như `z, n, v,` và `c` trong thanh ghi trạng thái PSR. Đối với những lệnh có biểu tượng tận cùng có 2 ký tự "`cc`", nếu kết quả của phép toán là số âm thì bit `n` sẽ có giá trị là 1.

Tương tự, bit `z = 1` nếu kết quả bằng 0 và `v=1` nếu tràn. Những lệnh có biểu tượng kết thúc không bằng 2 ký tự "`cc`" sẽ không có hiệu ứng rẽ nhánh. Trường `imm22` và `disp22` lưu trữ 22 bit được sử dụng như là toán hạng đối với cấu trúc SETHI và để làm cơ sở tính toán cho lệnh rẽ nhánh trong cấu trúc Branch

- Cấu trúc CALL chỉ bao gồm 2 trường duy nhất là trường `op` có nội dung `01` và trường `disp30` được sử dụng là nơi chứa địa chỉ của chương trình mà nó gọi tới.
- Cấu trúc Arithmetic có `op = 10` và cấu trúc Memmory có `op = 11` cùng có trường `rd` để nhận dạng thanh ghi nguồn với lệnh `st`, hoặc để nhận dạng thanh ghi đích trong các trường hợp lệnh còn lại. `Rs1` và `rs2` là các thanh ghi nguồn số 1 và 2. Trường `op3` được sử dụng để phân biệt các lệnh (được trình bày như hình 4.10)
- Trường `simm13` có 13 bit giá trị tức thì có dấu có thể mở rộng lên 32 bit khi trường `i` có giá trị bằng 1. Để làm được việc này, bit dấu tức là bit tận cùng bên phải được chép vào một số 32 bit có giá trị bằng số trong trường simm13 tương ứng. Ví dụ, giá trị trường trường `simm13` là $(1111111110011)_2 = (-13)_{10}$. Khi đó số nguyên 32 bit tương ứng được tạo ra khi trường i = 1 là $(11111111111111111111111110011)_2 = (-13)_{10}$.
- Cấu trúc lệnh Arithmetic có 2 toán hạng nguồn và 1 toán hạng đích, tổng cộng là 3 toán hạng. Cấu trúc lệnh Memmory có 2 toán hạng: một dành cho địa chỉ, một dành cho dữ liệu. Thông thường toán hạng nguồn là địa chỉ

4.2.5. Cấu trúc dữ liệu của ARC

ARC hỗ trợ 12 cấu trúc dữ liệu khác nhau được mô tả như hình 4.11. được chia thành 3 nhóm: số nguyên có dấu, số nguyên không dấu và số dấu phẩy động. Với các loại cấu trúc dữ liệu này, độ rộng dữ liệu cho phép là 1 byte (8 bit), halfword (16 bit), tagged word (32 bit nhưng chỉ sử dụng 30 bit có trọng số lớn nhất để chứa giá trị), doubleword (64 bit) và quadword (128 bit)



Hình 4.11. Cấu trúc dữ liệu của ARC

Trong thực tế sử dụng, ARC không phân biệt số có dấu và số không dấu. Cả 2 loại này được lưu trữ và tính toán như là số nguyên bù 2. Chúng chỉ khác nhau ở phương pháp biểu diễn. Trong từng trường hợp cụ thể, một giá trị sẽ được giả định là có dấu hoặc không dấu. Khi đó các bit `v` và `c` sẽ được sử dụng. Bit `c = 1` khi xảy ra hiện tượng tràn của số không dấu, bit `v = 1` khi tràn số có dấu.

Trong số tagged word, 2 bit có trọng số nhỏ nhất được sử dụng để biểu thị hiện tượng tràn số

Số dấu phẩy động được sử dụng trong máy tính ARC tuân theo chuẩn IEEE 754-1985 (xem chương 2)

4.2.6. Mô tả các lệnh của ARC

Chúng ta vừa tìm hiểu xong cấu trúc một lệnh. Bây giờ chúng ta sẽ mô tả kỹ hơn 15 lệnh được liệt kê trong hình 4.7. Trong phần nêu chi tiết này, ta sẽ đưa ra chi tiết mã lệnh (Object code) với mục đích chính là để tham khảo. Việc chuyển mã lệnh như thế nào sẽ được chúng ta tìm hiểu chi tiết trong chương kế tiếp

Tên lệnh	`ld`
Mô tả	Nạp nội dung của một thanh ghi từ bộ nhớ. Các địa chỉ bộ nhớ phải có giá trị chia hết cho 4 tức là có ranh giới từ. Giá trị địa chỉ sẽ được tính toán bằng cách cộng nội dung của thanh ghi trong trường `rs1` hoặc giá trị trong trường `simm13` tùy theo từng trường hợp
Ví dụ	`Ld [x], %r1`
	Hoặc `Ld [x], %r0, %r1`
	Hoặc `Ld %r0+x, %r1`
Ý nghĩa	Copy nội dung của bộ nhớ địa chỉ là `x` vào thanh ghi `r1`
Mã lệnh	`11000010000000000010100000010000` `(x = 2064)`

Tên lệnh `st`
Mô tả Cất nội dung của thanh ghi vào bộ nhớ. Các địa chỉ bộ nhớ phải có giá trị chia hết cho 4 tức là có ranh giới từ. Địa chỉ sẽ được tính toán bằng cách cộng nội dung của thanh ghi trong trường rs1 với nội dung của thanh ghi trong trường rs2 hoặc nội dung trong trường simm13. Trường rd trong lệnh này được sử dụng để chứa thanh ghi nguồn
Ví dụ `st %r1, [x]`
Ý nghĩa Copy nội dung của thanh ghi `r1` vào bộ nhớ địa chỉ là `x`
Mã lệnh `11000010001000000010100000010000 (x = 2064)`

Tên lệnh `sethi`
Mô tả Set nội dung của 22 bit cao của thanh ghi lên một giá trị nhất định và đưa 10 bit thấp của thanh ghi đó về 0. Nếu toán hạng là 0 và thanh ghi được sử dụng là `r0` thì lệnh này tương đương với lệnh không làm gì cả NOP
Ví dụ `sethi 0x304F15, %r1`
Ý nghĩa Nạp nội dung (304F15)16 vào 22 bit cao của thanh ghi `r1` còn 10 bit thấp xóa về 0
Mã lệnh `00000011001100000100111100010101`

Tên lệnh `andcc`
Mô tả Tính lệnh AND từng bit của 2 toán hạng nguồn và đưa kết quả về toán hạng đích
Ví dụ `andcc %r1, %r2, %r3`
Ý nghĩa Tính lệnh AND nội dung thanh ghi `r1` với thanh ghi `r2` và đưa kết quả vào thanh ghi `r3`
Mã lệnh `10000110100010000100000000000010`

Tên lệnh `orcc`
Mô tả Tính lệnh OR từng bit của 2 toán hạng nguồn và đưa kết quả về toán hạng đích
Ví dụ `orcc %r1, 1, %r1`
Ý nghĩa Set tất cả các bit trong thanh ghi `r1` lên giá trị 1 và đưa kết quả về thanh ghi `r1`
Mã lệnh `10000010100100000110000000000001`

Tên lệnh `Srl`
Mô tả Thực hiện phép dịch sang phải từ 0 tới 31 bit. Những bit ở phía bên trái sẽ được điền đầy bởi giá trị 0
Ví dụ `srl %r1, 3, %r2`
Ý nghĩa Dịch nội dung thanh ghi `r1` sang phải 3 bit, lưu kết quả vào thanh ghi `r2`
Mã lệnh `10000101001100000110000000000011`

Tên lệnh `addcc`
Mô tả Cộng 2 toán hạng nguồn và đưa kết quả về toán hạng đích với phép cộng 2 số bù 2
Ví dụ `addcc %r1, 5, %r1`
Ý nghĩa Cộng 5 vào thanh ghi `r1`
Mã lệnh `10000010100000000110000000000101`

Tên lệnh `call`
Mô tả Gọi chương trình con, lưu địa chỉ của lệnh đang thực hiện vào thanh ghi `%r15`. Trong mã lệnh, trường `disp30` trong cấu trúc `call` chứa 30 bit địa chỉ mà nó trỏ tới. Địa chỉ mà máy tính xử lý kế tiếp là địa chỉ có giá trị `4 x disp30`. Chú ý rằng `disp30` cũng có thể là số âm

Ví dụ `call sub_r`
Ý nghĩa Gọi chương trình con `sub_r`
Mã lệnh `01000000000000000000000000011001`

Tên lệnh `jmpl`
Mô tả Nhảy về chương trình chính từ chương trình con và cất địa chỉ của lệnh đang thực hiện về thanh ghi đích
Ví dụ `jmpl %r15 + 4, %r0`
Ý nghĩa Trở về từ chương trình chính. Địa chỉ trước kia được lưu ở thanh ghi `%r15` sẽ được khôi phục và lệnh kế tiếp được thực thi là ở địa chỉ `%r15+4`. Địa chỉ hiện tại sẽ được bỏ đi
Mã lệnh `10000001110000111110000000000100`

Tên lệnh `be`
Mô tả Nếu bit `z = 1` thì chương trình thì chương trình nhảy tới địa chỉ `4 x disp22` trong cấu trúc rẽ nhánh. Nếu `z = 0` thì lệnh nhảy tới lệnh kế tiếp sau lệnh `be`
Ví dụ `be label`
Ý nghĩa Nhảy tới nhãn `label` nếu `z = 1`
Mã lệnh `00000010100000000000000000000101`

Tên lệnh `bneg`
Mô tả Nếu bit `n = 1` thì chương trình thì chương trình nhảy tới địa chỉ `4 x disp22` trong cấu trúc rẽ nhánh. Nếu `n = 0` thì lệnh nhảy tới lệnh kế tiếp sau lệnh `bneg`
Ví dụ `bneg label`
Ý nghĩa Nhảy tới nhãn `label` nếu `n = 1`
Mã lệnh `00001100100000000000000000000101`

Tên lệnh `bcs`
Mô tả Nếu bit `c = 1` thì chương trình thì chương trình nhảy tới địa chỉ `4 x disp22` trong cấu trúc rẽ nhánh. Nếu `c = 0` thì lệnh nhảy tới lệnh kế tiếp sau lệnh `bcs`
Ví dụ `bcs label`
Ý nghĩa Nhảy tới nhãn `label` nếu `c = 1`
Mã lệnh `00001010100000000000000000000101`

Tên lệnh `bvs`
Mô tả Nếu bit `v = 1` thì chương trình thì chương trình nhảy tới địa chỉ `4 x disp22` trong cấu trúc rẽ nhánh. Nếu `v = 0` thì lệnh nhảy tới lệnh kế tiếp sau lệnh `bvs`
Ví dụ `bvs label`
Ý nghĩa Nhảy tới nhãn `label` nếu `v = 1`
Mã lệnh `00001110100000000000000000000101`

Tên lệnh `ba`
Mô tả Nhảy không điều kiện đến địa chỉ `4 x disp22` trong cấu trúc rẽ nhánh
Ví dụ `ba label`
Ý nghĩa Nhảy không điều kiện tới nhãn `label`. Trong mã lệnh dưới đây, nhãn là là vùng nhớ trước vùng nhớ hiện tại 5 word
Mã lệnh `00010000101111111111111111111011`

4.3. Toán tử giả

Ngoài những lệnh được liệt kê ở trên, ARC còn có các toán tử giả hay còn gọi là lệnh giả để hỗ trợ các lập trình viên xây dựng cấu trúc chương trình. Các lệnh giả được liệt kê chi tiết trong hình 4.12. Lưu ý rằng các lệnh giả không phải là các lệnh sẽ được xử lý bởi máy tính. Các lệnh giả này sẽ được xử lý bởi trình biên dịch tức là sẽ được thực hiện bởi phần mềm

Pseudo-Op	Usage	Meaning
.equ X	.equ #10	Treat symbol X as $(10)_{16}$
.begin	.begin	Start assembling
.end	.end	Stop assembling
.org	.org 2048	Change location counter to 2048
.dwb	.dwb 25	Reserve a block of 25 words
.global	.global Y	Y is used in another module
.extern	.extern Z	Z is defined in another module
.macro	.macro M a, b, ...	Define macro M with formal parameters a, b, ...
.endmacro	.endmacro	End of macro definition
.if	.if <cond>	Assemble if <cond> is true
.endif	.endif	End of .if construct

Hình 4.12. Các lệnh giả

Lệnh `.equ` được sử dụng để gán một giá trị hay một chuỗi ký tự cho một biểu tượng. Lệnh `.begin` và `.end` được sử dụng để thông báo cho trình biên dịch biết điểm bắt đầu và kết thúc của chương trình. Tất cả các lệnh trước `.begin` và sau `.end` sẽ bị bỏ qua. Một chương trình có thể có nhiều hơn 1 cặp `.begin/.end` nhưng một lệnh `.end` sẽ sẽ kết thúc tất cả các `.begin` và trong chương trình phải có ít nhất một lệnh .`begin`.

Lệnh `.org` sẽ làm cho các lệnh được viết sau nó sẽ được lưu trữ tại địa chỉ bắt đầu từ địa chỉ mà nó chỉ ra. Ví dụ trong hình 4.12. lệnh sau `.org 2048` sẽ được lưu bắt đầu từ địa chỉ 2048. Lệnh `.dwb` sẽ tạo ra những khối dữ liệu mà mỗi phần tử của nó có độ rộng 4 byte. Lệnh này hay được sử dụng để tạo ra các khối mảng dữ liệu.

Lệnh `.global` và `.extern` được sử dụng với các biến và địa chỉ mà các biến và địa chỉ đó được viết trong 1 modul hợp ngữ nhưng lại được sử dụng trong

một modul hay chương trình khác. Lệnh `.global` cho phép các biến mà nó định nghĩa được sử dụng trong nhiều chương trình khác nhau. Lệnh `.extern` chỉ ra rằng biến mà nó đang sử dụng được định nghĩa trong một modul khác

Lệnh `.macro, .endmacro, .if` và `.endif` sẽ được tìm hiểu trong chương sau

4.4. Ví dụ về một chương trình hợp ngữ

Quá trình viết chương trình hợp ngữ không khác lắm so với việc viết một chương trình bậc cao ngoại trừ một số chi tiết mà ta phải tuân thủ. Trong phần này ta sẽ tìm hiểu 2 chương trình mẫu được viết bằng hợp ngữ

a, Chương trình cộng hai số nguyên

```
! This programs adds two numbers
        .begin
        .org 2048
prog1:  ld      [x], %r1          ! Load x into %r1
        ld      [y], %r2          ! Load y into %r2
        addcc   %r1, %r2, %r3     ! %r3 ← %r1 + %r2
        st      %r3, [z]          ! Store %r3 into z
        jmpl    %r15 + 4, %r0     ! Return
x:      15
y:      9
z:      0
        .end
```

Hình 4.13. Chương trình cộng 2 số nguyên

Chúng ta nghiên cứu một ví dụ về chương trình hợp ngữ viết cho máy tính ARC như hình 4.13. Đây là một chương trình đơn giản để cộng số 15 và 9. Chương trình được bắt đầu và kết thúc bởi cặp lệnh giả `.begin/.end`. Lệnh giả `.org` chỉ ra rằng các lệnh hợp ngữ sẽ được nạp vào máy tính bắt đầu từ địa chỉ 2048. Hai toán hạng 15 và 9 sẽ được lưu vào 2 biến x và y. Chúng ta chỉ có thể thực hiện phép toán cộng với các số được lưu trữ trong thanh ghi ở máy tính ARC bởi vì chỉ các lệnh `ld` và `st` mới được truy cập bộ nhớ. Do đó chương trình sẽ bắt đầu bằng việc nạp các giá trị x và y vào thanh ghi `%r1` và `%r2`. Lệnh `addcc` sẽ thực hiện phép toán cộng và đưa kết quả vào thanh ghi `%r3`. Lệnh

`st` sẽ cất kết quả phép toán từ thanh ghi `%r3` vào biến z đã định trước. Lệnh jmpl với các toán tử `%r15+4,%r0` có tác dụng quay trở về chương trình chính. Ở đây ta giả định chương trình thực hiện phép toán cộng này là một chương trình con và sẽ được một chương trình ở cấp cao hơn gọi tới.

b, Chương trình tính tổng một chuỗi số nguyên

```
! This program sums LENGTH numbers
! Register usage:       %r1 - Length of array a
!                       %r2 - Starting address of array a
!                       %r3 - The partial sum
!                       %r4 - Pointer into array a
!                       %r5 - Holds an element of a
            .begin                      ! Start assembling
          .org  2048            ! Start program at 2048
a_start   .equ  3000              ! Address of array a
            ld    [length], %r1 ! %r1 ← length of array a
            ld    [address],%r2 ! %r2 ← address of a
            andcc %r3, %r0, %r3 ! %r3 ← 0
loop:       andcc %r1, %r1, %r0 ! Test # remaining elements
            be    done          ! Finished when length=0
            addcc %r1, -4, %r1  ! Decrement array length
            addcc %r1, %r2, %r4 ! Address of next element
            ld    %r4, %r5      ! %r5 ← Memory[%r4]
            addcc %r3, %r5, %r3 ! Sum new element into r3
            ba    loop          ! Repeat loop.

done:     jmpl  %r15 + 4, %r0 ! Return to calling routine

length:             20          ! 5 numbers (20 bytes) in a
address:            a_start
            .org  a_start       ! Start of array a
a:                   25         ! length/4 values follow
                    -10
                     33
                     -5
                      7
            .end                        ! Stop assembling
```

Hình 4.14. Chương trình tính tổng chuỗi số nguyên

Chương trình chúng ta xem xét ở đây phức tạp hơn ví dụ trước, mã chương trình được thể hiện trên hình 4.14. Cũng như chương trình ở phía trên, chương trình này cũng bắt đầu bằng cặp lệnh giả `.begin/.end` và cũng sử dụng lệnh

`.org` để lưu mã lệnh vào bộ nhớ bắt đầu từ địa chỉ 2048. Biến `a_start` được tạo ra với giá trị mặc định là 3000

Chương trình bắt đầu bằng việc nạp độ dài của mảng a vào thanh ghi `%r1`, địa chỉ bắt đầu của mảng vào `%r2` và xóa nội dung thanh ghi `%r3` để làm nơi chứa tổng cần tính (bằng cách thực hiện phép toán AND từng bit với thanh ghi `%r0`)

Vòng lặp loop sẽ thực hiện lần lượt cộng các thành phần của mảng vào tổng trong thanh ghi `%r3`. Vòng lặp loop bắt đầu bằng việc kiểm tra xem số phần tử còn lại cần phải cộng vào tổng có phải là 0 hay không. Việc này được thực hiện bằng cách thực hiện phép toán AND giữa `%r1` với chính nó. Nếu `%r1` = 0 thì kết quả là 0, việc thực hiện phép cộng sẽ dừng lại vì chương trình nhảy đến nhãn `done`. Nếu `%r1` khác 0 thì kết quả của phép AND cũng khác 0, chương trình thực hiện phép cộng.

Để thực hiện phép cộng, chương trình sẽ

- Tính lại số phần tử cần cộng vào tổng bằng cách trừ `%r1` đi giá trị tương ứng 1 phần tử (tức là cộng với -4). Có thể coi `%r1` chính là địa chỉ tương đối của mảng
- Xác định địa chỉ của phần tử cần cộng vào tổng tại vòng lặp đang xét và đưa địa chỉ này vào thanh ghi `%r4.` Việc này được thực hiện bằng cách cộng địa chỉ tuyệt đối của mảng (`%r2`) với địa chỉ tương đối của mảng (`%r1`)
- Nạp phần tử mảng vào `%r5`
- Cộng phần tử vừa đọc được từ `%r5` vào `%r3`
- Nhảy về nhãn `loop`

4.5. Truy cập dữ liệu bộ nhớ - Các chế độ địa chỉ

Bảng 4.1. dưới đây chỉ ra các chế độ địa chỉ sử dụng trong ARC. Các quy ước trong bảng này cũng rất phổ biến trong lập trình hợp ngữ. Ký hiệu M[x] có ý nghĩa dữ liệu được truy cập theo kiểu dữ liệu mảng với giá trị x được cho trong ngoặc là chỉ số địa chỉ được cho dưới kiểu byte là địa chỉ của phần tử trong mảng.

Chế độ địa chỉ	Ký hiệu	Ý nghĩa
Địa chỉ tức thì	#K	K

Địa chỉ trực tiếp	K	M[k]
Địa chỉ gián tiếp	(K)	M[M[K]]
Địa chỉ thanh ghi	(Rn)	M[Rn]
Địa chỉ chỉ số thanh ghi	(Rm+Rn)	M[Rm+Rn]
Địa chỉ thanh ghi cơ sở	(Rm+x)	M[Rm+x]
Địa chỉ thanh ghi tương đối cơ sở chỉ số	(Rm+Rn+x)	M[Rm+Rn+x]

Bảng 4.1. Các chế độ địa chỉ

- Chế độ địa chỉ tức thì cho phép truy xuất một hằng số trong một chu kỳ máy
- Chế độ địa chỉ trực tiếp được sử dụng để truy cập dữ liệu với địa chỉ đã biết trước trong một chu kỳ máy
- Chế độ địa chỉ gián tiếp được sử dụng để truy cập vào một con trỏ của một biến mà đã biết địa chỉ của biến đó. Chế độ địa chỉ này hiếm khi được sử dụng trong các vi xử lý hiện đại vì nó cần tham chiếu tới 2 ô nhớ để truy cập toán hạng. Lập trình viên khi sử dụng phương pháp truy cập này cần phải thực hiện bằng 2 lệnh, một để truy cập con trỏ và một lệnh khác để truy cập giá trị cần truy cập.
- Chế độ địa chỉ thanh ghi được sử dụng khi ta không biết địa chỉ của toán hạng cho đến khi máy thực hiện lệnh.
- Chế độ địa chỉ chỉ số thanh ghi, chế độ thanh ghi cơ sở và chế độ thanh ghi tương đối cơ sở chỉ số thường được sử dụng để truy cập vào các dữ liệu kiểu mảng như ở ví dụ trong mục 4.4.b

4.6. Các mối liên kết chương trình con và stack

Chương trình con hay còn có tên gọi là hàm (function) hay thủ tục (procedure) bản chất là một chuỗi các câu lệnh được đại diện bởi một tên gọi ở một cấp lập trình cao hơn. Khi một chương trình gọi chương trình con, chương trình sẽ chuyển quyền điều khiển từ chương trình chính sang chương trình con để chương trình con thực hiện các lệnh của nó. Khi các lệnh của chương trình con đã thực hiện xong, chương trình con chuyển ngược quyền điều khiển trở về chương trình đã gọi tới nó. Việc thực hiện liên kết giữa chương trình chính và chương trình con có thể được thực hiện bởi một trong những phương pháp sau

a, Liên kết thông qua thanh ghi

```
! Calling routine                ! Called routine
      :                          ! %r3 ← %r1 + %r2
      :
     ld     [x], %r1
     ld     [y], %r2            add_1: addcc  %r1, %r2, %r3
     call   add_1                      jmpl   %r15 + 4, %r0
     st     %r3, [z]
      :
      :
x:   53
y:   10
z:    0
```

Hình 4.15. Liên kết chương trình con qua thanh ghi

Trong chương trình trên, 2 đối số sẽ được nạp vào 2 thanh ghi `%r1` và `%r2`, sau đó chương trình con `add_1` sẽ được gọi và kết quả được trả ngược lại thanh ghi `%r3`. Chương trình con `add_1` sẽ nhận toán hạng thông qua 2 thanh ghi `%r1` và `%r2` và trả kết quả vào `%r3` trước khi quay trở về chương trình chính thông qua lệnh `jmpl`. Phương pháp này hoạt động nhanh và đơn giản, nhưng nó không thể áp dụng trong trường hợp số lượng các đối số lớn hơn số lượng các thanh ghi có thể sử dụng hoặc trong trường hợp các chương trình con lồng vào nhau

b, Liên kết thông qua dữ liệu

Trong phương pháp liên kết thông qua dữ liệu, địa chỉ của dữ liệu liên kết sẽ được nạp từ thanh ghi. Như trên hình 4.16. lệnh giả .dwb trong chương trình chính sẽ tạo ra một vùng dữ liệu liên kết gồm 3 từ, địa chỉ là x, x + 4 và x + 8. Trước khi gọi chương trình con, chương trình chính nạp 2 toán tử vào x và x +4. Chương trình con sẽ lưu kết quả mà nó xử lý vào vùng nhớ x + 8. Để xem xét kỹ hơn về ví dụ này, ta sẽ phân tích từng bước mà chương trình đã thực hiện

- Chương trình lưu nội dung của thanh ghi `%r1` và `%r2` vào x và x + 4
- Nạp giá trị x vào 22 bit cao của thanh ghi `%r5`
- Dịch sang trái `%r5` 10 bit, điều này đồng nghĩa với việc chuyển toàn bộ giá trị địa chi x vào `%r5`

```
! Calling routine
    .
    .
    .
    st     %r1, [x]
    st     %r2, [x+4]
    sethi  x, %r5
    srl    %r5, 10, %r5
    call   add_2
    ld     [x+8], %r3
    .
    .
    .
! Data link area
x:  .dwb   3
```

```
! Called routine
! x[2] ← x[0] + x[1]

add_2:  ld     %r5, %r8
        ld     %r5 + 4, %r9
        addcc  %r8, %r9, %r10
        st     %r10, %r5 + 8
        jmpl   %r15 + 4, %r0
```

Hình 4.16. Liên kết thông qua dữ liệu

- Gọi chương trình `add_2` để thực hiện các lệnh sau
  - Nạp `%r5` vào `%r8`, tức là nạp x vào `%r8`
  - Nạp `%r5 + 4` vào `%r9` tức là x + 4 vào `%r9`
  - Tính tổng và đưa kết quả vào `%r10`
  - Lưu kết quả vào `%r5 + 8` tức x + 8
  - Quay về chương trình chính
- Lưu kết quả từ x+ 8 vào `%r3`

c, Liên kết thông qua stack

```
  ! Calling routine
      .
      .
%sp .equ   %r14
    addcc  %sp, -4, %sp
    st     %r1, %sp
    addcc  %sp, -4, %sp
    st     %r2, %sp
    call   add_3
    ld     %sp, %r3
    addcc  %sp, 4, %sp
      .
      .
```

```
! Called routine
! Arguments are on stack.
! %sp[0] ← %sp[0] + %sp[4]
    %sp  .equ   %r14
add_3:  ld     %sp, %r8
        addcc  %sp, 4, %sp
        ld     %sp, %r9
        addcc  %r8, %r9, %r10
        st     %r10, %sp
        jmpl   %r15 + 4, %r0
```

Hình 4.17.Liên kết dữ liệu thông qua stack

Phương pháp liên kết chương trình con thứ 3 là liên kết thông qua stack. Về mặt bản chất, phương pháp này không khác lắm so với liên kết dữ liệu thông qua bộ nhớ. Với phương pháp này, ta cũng khai báo một vùng nhớ trong bộ nhớ để làm

stack và sử dụng bình thường. Điểm khác biệt giữa phương pháp này và phương pháp liên kết thông qua bộ nhớ là truy xuất các ô nhớ liên tiếp trong stack thì giảm dần còn các ô nhớ liên tiếp trong bộ nhớ thì truy xuất tăng dần. Ta phân tích ví dụ trong hình 4.17

- Khai báo stack (thanh ghi `%r14` luôn luôn chứa địa chỉ của stack)
- Xác định địa chỉ của phần đỉnh stack bằng cách trừ đáy stack đi 4
- Lưu giá trị `%r1` vào đỉnh stack vừa xác định
- Tiếp tục xác định địa chỉ đỉnh stack
- Lưu giá trị `%r2` vào đỉnh stack vừa xác định
- Gọi chương trình `add_3` để thực hiện các nội dung sau
    - Xác định đỉnh stack
    - Nạp giá trị đỉnh stack vào `%r8`, tức là nạp `%r2` vào `%r8`
    - Tăng giá trị stack
    - Nạp đỉnh stack vào `%r9` tức là nạp `%r1` vào `%r9`
    - Tính tổng và đưa vào `%r10`
    - Lưu vào stack
    - Quay về chương trình chính
- Trả kết quả từ stack vào `%r3`

Phương pháp liên kết thông qua stack được sử dụng rất phổ biến bởi vì phương pháp này có thể được sử dụng trong mọi trường hợp, kể cả trong trường hợp gọi các chương trình con lồng vào nhau. Phương pháp liên kết này được áp dụng rất rộng rãi cả trong ngôn ngữ lập trình bậc cao như ngôn ngữ C.

Trong ví dụ trong hình 4.18, hai chương trình con là `func_1` và `func_2` sử dụng các toán hạng được chương trình chính khai báo giá trị và lưu vào trong stack trước khi thực hiện chương trình con thông qua dấu ngoặc đơn.

Với chương trình con `func_1` và `func_2`, sau khi tính toán các giá trị, kết quả được lưu vào stack thông qua lệnh return. Kết quả này sẽ được bảo lưu trong stack và được gán vào các biến w và z tương ứng trong chương trình chính

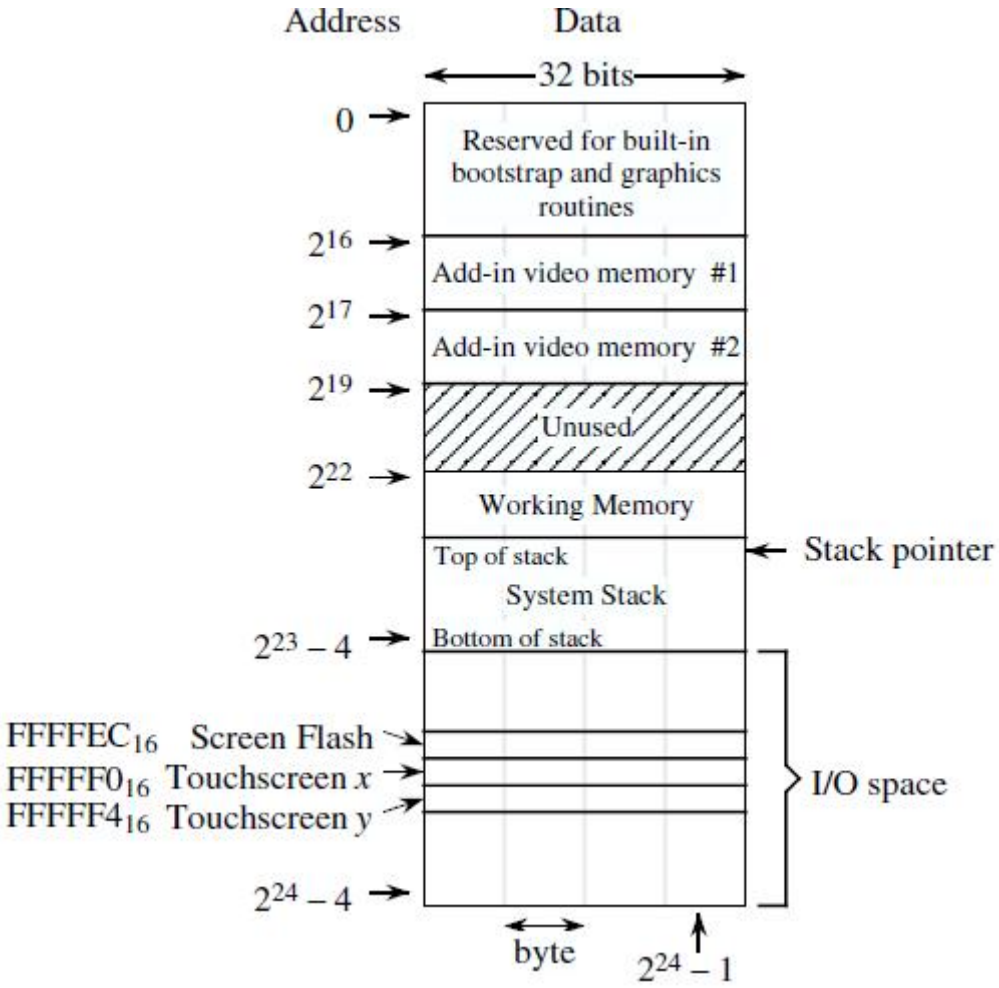
```
Line No.  /* C program showing nested subroutine calls */
00  main()
01  {
02      int w, z;          /* Local variables */
03      w = func_1(1,2);   /* Call subroutine func_1 */
04      z = func_2(10);    /* Call subroutine func_2 */
05  }                      /* End of main routine */

06  int func_1(x,y)        /* Compute x * x + y */
07  int x, y;           /* Parameters passed to func_1 */
08  {
09      int i, j;         /* Local variables */
10      i = x * x;
11      j = i + y;
12      return(j);    /* Return j to calling routine */
13  }

14  int func_2(a)        /* Compute a * a + a + 5 */
15  int a;               /* Parameter passed to func_2 */
16  {
17      int m, n;        /* Local variables */
18      n = a + 5;
19      m = func_1(a,n);
20      return(m);       /* Return m to calling routine */
21  }
```

Hình 4.18. Liên kết chương trình thông qua stack trong ngôn ngữ bậc cao C

4.7. Nhập và xuất dữ liệu trong hợp ngữ

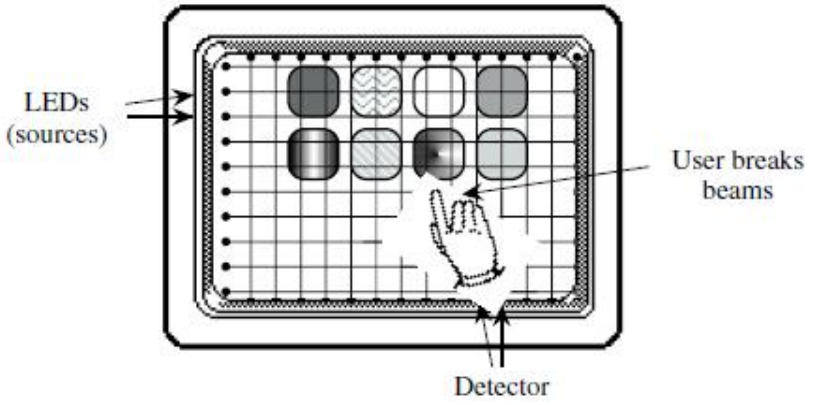
Trong phần này, chúng ta sẽ nghiên cứu cách thức một chương trình hợp ngữ liên kết với bên ngoài thông qua các cổng ngoại vi. Một cách để thiết bị ngoại vi liên kết với máy tính là sử dụng những lệnh giao tiếp đặc biệt và với hệ thống bus vào/ra được thiết kế riêng cho mục đích này. Một phương pháp khác được sử dụng là tương tác thông qua việc ánh xạ không gian bộ nhớ vào /ra. Cụ thể, mỗi một thiết bị vào /ra sẽ chiếm một địa chỉ nhất định trong không gian nhớ. Khi đó, máy tính tương tác với thế giới bên ngoài được thực hiện không khác gì việc tương tác với bộ nhớ máy tính

Một ví dụ về ánh xạ vào/ra được sử dụng trong máy tính ARC được trình bày trong hình 4.20. Trên hình, ta có thể nhìn thấy trong máy tính có một vài vùng nhớ, trong đó, hệ thống đã giành ra 2 vùng add-in video memory và cho vùng



Hình 4.20. Ánh xạ bộ nhớ máy tính ARC

touchscreen. Màn hình cảm ứng có 2 dạng lượng tử và điện. Một minh họa về màn hình cảm ứng dạng lượng tử được mô tả trên hình 4.21. Một ma trận các chùm tia bao phủ các chiều ngang và dọc màn hình. Nếu chùm tia bị ngắt quãng bởi ngón tay chạm vào màn hình thì vị trí đó sẽ được tính toán bởi vị trí bị ngắt quãng



Hình 4.21. Màn hình touchscreen

Bộ nhớ chỉ thực sự bị chiếm dụng trong khoảng địa chỉ từ 2^{22} đến 2^{23}-1 (lưu ý rằng 2^{23}-4 là địa chỉ ở tận cùng bên trái trong định dạng big-endian). Phần còn lại của không gian địa chỉ được sử dụng cho các mục đích khác. Phần không gian địa chỉ từ 0 đến 2^{16}-1 được sử dụng để chứa các chương trình được xây dựng sẵn cho việc quản lý hệ điều hành và các chương trình con, chương trình ngắt. Địa chỉ từ 2^{16} đến 2^{19}-1 được dành riêng cho 2 vùng nhớ add-in video memory với mục đích là lưu các dữ liệu xử lý video. Lưu ý rằng các dữ liệu video sẽ chỉ được xử lý khi các module video được chèn vào tương ứng. Cuối cùng khoảng địa chỉ 2^{23} đến 2^{24}-1 được sử dụng cho các thiết bị ngoại vi

Trở lại bài toán màn hình touchscreen, tọa độ x và y sẽ được tự động cập nhật vào các thanh ghi tương ứng trong khoảng không gian bộ nhớ. Các thanh ghi này được truy cập một cách đơn giản thông qua việc đọc ô nhớ ánh xạ đến nó. Vùng nhớ “Screen Flash” là nơi chứa chương trình để đọc các giá trị x và y bất cứ khi nào màn hình được chạm vào

TỔNG KẾT CHƯƠNG

Trong chương này, chúng ta đã nghiên cứu máy tính ARC và một số thành phần cơ bản của ARC. Thông qua đó, ta đã giải quyết các nội dung cơ bản sau

- Nghiên cứu về thành phần cơ bản của máy tính như CPU, cấu trúc bộ nhớ, phương pháp truy cập bộ nhớ.
- Nghiên cứu về phương pháp truy cập bộ nhớ, phân tích các chế độ địa chỉ bộ nhớ
- Tìm hiểu về phương pháp lập trình top-down trong lập trình vi xử lý

CHƯƠNG 5: NGÔN NGỮ BẬC CAO VÀ MÁY TÍNH

Trong chương trước, chúng ta đã xem xét mối quan hệ giữa máy tính ISA, ngôn ngữ hợp ngữ và ngôn ngữ máy. Bây giờ chúng ta sẽ xem xét kỹ hơn về tác động của các câu lệnh lên các thanh ghi, về sự dịch chuyển dữ liệu giữa bộ nhớ và CPU, và tất nhiên cả quá trình liên kết giữa hợp ngữ và chương trình. Cũng trong chương này, chúng ta sẽ mở rộng hơn khái niệm về mối quan hệ giữa ngôn ngữ máy tính và máy tính

Chúng ta sẽ bắt đầu với việc thảo luận về biên dịch, quá trình chuyển đổi từ ngôn ngữ lập trình bậc cao sang chương trình hợp ngữ. Tiếp đó, chúng ta sẽ thảo luận quá trình chuyển đổi từ ngôn ngữ hợp ngữ sang ngôn ngữ máy. Phần tiếp theo, chúng ta sẽ thảo luận về vấn đề liên kết chương trình, đây là quá trình liên kết dữ liệu kiểu modul để trở thành một chương trình hoàn chỉnh, quá trình nạp, đây là quá trình dịch chuyển chương trình đến bộ nhớ và quá trình chuẩn bị thực thi. Chúng ta cũng sẽ nghiên cứu về macro, đây là những đoạn chương trình được sử dụng giống như chương trình thủ tục

5.1. Quá trình biên dịch

Như trong chương trước chúng ta đã nghiên cứu, quá trình chuyển một chương trình từ ngôn ngữ hợp ngữ sang ngôn ngữ máy được thực hiện một cách rất đơn giản, bởi vì đó chỉ là quá trình từng bước chuyển 1 lệnh hợp ngữ sang tổ hợp nhị phân lệnh tương ứng. Tuy nhiên đối với ngôn ngữ bậc cao, quá trình này được thực hiện phức tạp hơn nhiều

5.1.1. Các bước thực hiện quá trình biên dịch

Để xem xét các bước thực hiện quá trình biên dịch, ta sẽ xem xét quá trình chuyển một câu lệnh đơn giản được viết từ một ngôn ngữ bậc cao thành một đoạn chương trình hợp ngữ

Câu lệnh A = B + 4;

Trình biên dịch phải đối mặt với một số nhiệm vụ phức tạp trong việc chuyển câu lệnh trên thành một hay một vài lệnh hợp ngữ, đó là các vấn đề

- Giảm lượng chữ viết trong câu lệnh, chuyển nó thành các biểu tượng của câu lệnh của ngôn ngữ mà ta đang nghiên cứu. Ví dụ trong câu lệnh trên, ta phải định nghĩa cac biến A và B, cũng như phải nhận dạng và định nghĩa các phép toán + và toán tử =. Quá trình này được gọi là quá trình **phân tích ngôn từ**
- Phân tích các ký hiệu để nhận biết cấu trúc cơ bản của chương trình. Trong ví dụ trên, việc phân tích cú pháp phải nhận ra được các cấu trúc cơ bản như là cấu trúc cơ bản của lệnh gán được biểu hiện bằng ký tự "=", hay là cấu trúc của phép toán cộng được biểu hiện bằng ký tự "+". Quá trình trên còn được gọi là **phân tích cú pháp**
- **Quá trình phân tích tên biến**: đây là quá trình kết hợp các biến A và B của chương trình với một vị trí nhất định trong bộ nhớ để lưu trữ các giá trị đó
- **Quá trình phân tích dạng dữ liệu**. Trong ví dụ trên, các biến A và B và hằng số 4 sẽ được nhận diện thành dạng biến kiểu số nguyên `int`. Quá trình phân tích tên biến và quá trình phân tích dạng dữ liệu đôi khi được gọi tên chung là phân tích ngữ nghĩa
- Tạo bản đồ mã và tạo mã lệnh, đây là quá trình kết nối các câu lệnh trong chương trình với ngôn ngữ hợp ngữ tương ứng. Với câu lệnh trên, chương trình hợp ngữ tương ứng có thể là

```
ld     [B],%r0, %r1  ; Nạp nội dung của ô nhớ [B] vào %r1
add      %r1,4, %r2  ; Cộng giá trị của %r1 với 4, đưa kết quả vào %r2
st     %r2,%r0, [A]  ; Lưu kết quả từ %r2 vào ô nhớ [A]
```

- Một số bước tiếp theo mà trình biên dịch phải thực hiện là quản lý biến và quản lý các thanh ghi, theo dõi việc sử dụng các thanh ghi, tối ưu chương trình...

5.1.2. Các đặc trưng của quá trình biên dịch

Trong khi quá trình biên dịch đang thực hiện nhiệm vụ của nó, các thông tin về kiến trúc tập lệnh ISA sẽ được nhúng vào nó. (Chú ý rằng kiến trúc tập lệnh ISA của trình biên dịch không nhất thiết phải giống kiến trúc tập lệnh ISA mà nó tạo ra, quá trình này còn có tên là **biên dịch chéo**). Các thông tin ISA nhúng vào trình biên dịch còn được gọi là các đặc trưng của quá trình biên dịch. Ví dụ, trình biên dịch sẽ quyết định gán cho các biến và các hằng số trong chương trình dạng dữ liệu tương ứng tùy thuộc vào loại ngôn ngữ máy tính mà hệ thống đang

sử dụng. Đối với ngôn ngữ lập trình C, kiểu dữ liệu `int` có thể là 16 bit, 32 bit hoặc hơn. Đối với ngôn ngữ Java, các số kiểu `int` luôn là 32 bit.

Trình biên dịch cũng phải tính tới các tính năng và hạn chế của máy tính khi nhúng các đặc trưng của máy tính vào hợp ngữ. Ví dụ trong máy tính ARC, tập lệnh của ARC yêu cầu toán hạng của các phép toán phải là số tức thì hoặc là thanh ghi. Do đó trình biên dịch phải tạo ra các lệnh để chuyển toán hạng cần thực hiện phép toán vào các thanh ghi trước khi tạo ra lệnh để thực hiện phép toán. Đây cũng là nguyên nhân chính trình biên dịch thực hiện biên dịch lệnh

A = B + 4

Phải có lệnh

```
ld      [B],%r0, %r1  ; Nạp nội dung của ô nhớ [B] vào %r1
```

Trước khi có lệnh cộng

```
add       %r1,4, %r2  ; Cộng giá trị của %r1 với 4, đưa kết quả vào %r2
```

Nói tóm lại, trong quá trình biên dịch, chuyển một chương trình bậc cao thành chương trình được viết bằng ngôn ngữ thấp hơn, trình biên dịch có một số đặc trưng sau

- Dạng dữ liệu của các biến số hay hằng số sẽ được trình biên dịch tự động gán cho một kiểu dữ liệu phụ thuộc vào đặc trưng của kiến trúc tập lệnh ISA, tức là kiến trúc ISA được nhúng vào trình biên dịch
- Quá trình tạo mã lệnh cấp thấp cũng tính toán tới đặc trưng của máy tính sử dụng lệnh mà nó tạo ra, để mã lệnh được tạo ra tương thích với máy tính sẽ sử dụng nó

5.1.3. Biên dịch các nhóm lệnh sang hợp ngữ

Trong phần này, chúng ta sẽ nghiên cứu kỹ hơn vấn đề chuyển 3 nhóm lệnh: nhóm lệnh dịch chuyển dữ liệu, nhóm lệnh toán học, và nhóm lệnh điều khiển luồng chương trình từ ngôn ngữ bậc cao sang hợp ngữ. Trong ví dụ dưới đây, chúng tôi sẽ sử dụng ngôn ngữ lập trình C để mô tả quá trình này bởi vì ngôn ngữ C rất phổ biến, thông dụng, cấu trúc câu lệnh cũng như ý nghĩa của câu

lệnh rất gần gũi với đời sống hàng ngày làm cho người không biết về ngôn ngữ C cũng có thể tạm hiểu được vấn đề. Trước khi tìm hiểu về 3 nhóm lệnh trên, chúng ta sẽ nghiên cứu trước 1 vấn đề chung, đó là việc lưu trữ các biến trong bộ nhớ máy tính

Trong các ví dụ phía trên trong tài liệu này, chúng ta luôn sử dụng các biến được truy xuất trực tiếp thông qua tên biến, tức là nó đã được định nghĩa tại vùng nhớ xác định trong bộ nhớ và nó sẽ được truy xuất ngay tại thời điểm thực thi câu lệnh liên quan đến nó. Ví dụ như trong lệnh A = B + 4 ở trên, các biến A và B được định nghĩa trước khi câu lệnh đó được thực hiện. Thực tế, chỉ có toàn cục **global variable** hay là biến **static variable** trong C mới được truy xuất ngay tại thời điểm thực thi. Những biến được khai báo và sử dụng trong các hàm hoặc khối hàm lại hoàn toàn khác, nó chỉ tồn tại khi hàm đó hoặc khối hàm đó được truy cập, các biến đó sẽ mất đi khi hàm hoặc khối hàm kết thúc truy cập. Các biến này được gọi là các biến địa phương **local** hoặc trong C có tên là automatic **variable**. Trong các chương trình, các biến địa phương được sử dụng thông dụng hơn rất nhiều so với biến toàn cục

Vì bản chất không được sử dụng bền vững của biến địa phương, để thực hiện việc quản lý chúng, ta có thể sử dụng phương pháp last-in-first-out giống như stack được mô tả trong chương 4. Các biến được lưu trong stack (bản chất vẫn là lưu trong bộ nhớ) và sẽ được sử dụng khi hàm được gọi tới. Khi kết thúc hàm, các biến sẽ được stack xóa đi. Trong chương trước, chúng ta đã biết hệ thống ARC sử dụng thanh ghi con trỏ `%sp` để truy cập vào stack. Thực tế, người ta hay sử dụng phương pháp copy nội dung của %sp vào một thanh ghi khác để sử dụng, đó là thanh ghi frame pointer `%fp` (thanh ghi này có thể được hiểu là thanh ghi cơ sở).Lúc đó, ta sẽ truy cập `%fp` để sử dụng trong hàm như là một biến. Ta phải làm như thế vì stack ngoài việc lưu trữ các biến địa phương, nó còn phải lưu trữ các biến tạm thời hoặc các giá trị trung gian trong quá trình tính toán. Ngoài việc sử dụng trực tiếp `%fp`, ta cũng có thể sử dụng các ô nhớ offset xung quanh `[%fp + n]`. Ví dụ nếu ta muốn nạp một biến được lưu tại byte số 12 phía trước `%fp` vào thanh ghi `%r1`, ta có thể sử dụng 1 trong 2 câu lệnh sau

```
ld   %fp, -12, %r1
ld   [%fp – 12], %r1
```

5.1.4. Nhóm lệnh dịch chuyển của dữ liệu

Thực chất vấn đề biên dịch câu lệnh của nhóm lệnh dịch chuyển dữ liệu chính là việc chuyển việc sử dụng các biến số và hằng số có trong ngôn ngữ bậc cao sang vị trí một ô nhớ trong hợp ngữ. Vấn đề này đã được giải quyết trong phần 5.1.3. bằng cách sử dụng stack và thanh ghi `%fp` để lưu trữ các biến số. Trong phần này, chúng ta sẽ bàn kỹ hơn một chút về vấn đề sử dụng các biến mà kiểu biến được định nghĩa bởi người sử dụng

Biến dạng cấu trúc

Trong ngôn ngữ C, ta có kiểu biến mà người sử dụng định nghĩa, đó là kiểu `struct` hay là kiểu `record` trong Pascal

Xem xét kiểu `struct` trong C, ta lấy ví dụ về kiểu `struct` này với dữ liệu được định nghĩa là kiểu `point` với cấu trúc như sau

```
Struct point {
    Int x;
    Int y;
    Int z;
}
```

Biến (có tên là điểm A) sử dụng cấu trúc trên được khai báo trong C như sau

```
Struct point diemA;
```

Với cách định nghĩa trên, người dùng có thể truy cập vào từng phần tử của biến `diemA` bằng toán tử dấu chấm quen thuộc `diemA.x.`

Đối với việc biên dịch sang ngôn ngữ hợp ngữ, trình biên dịch sẽ phải tạo ra các ô nhớ tương ứng với cấu trúc `point`. Với việc khai báo biến diemA có cấu trúc trên, trình biên dịch sẽ tạo ra một "biến"-một ô nhớ có tên là `diemA`. Biến `diemA.x` sẽ tương ứng với ô nhớ `[diemA],` biến `diemA.y` tương ứng `[diemA + 4],` biến `diemA.z` tương ứng `[diemA + 8]` . Do đó, câu lệnh để nạp biến y vào thanh ghi %r1 sẽ là

```
ld   [diemA + 4], %r1
```

Biến dạng mảng

Hầu hết các ngôn ngữ lập trình đều cung cấp kiểu dữ liệu mảng array. Trong ngôn ngữ C, để định nghĩa mảng A gồm 10 phần tử kiểu `int` ta sử dụng câu lệnh

```
Int A[10];
```

Khi đó, C sẽ tạo ra một chuỗi phần tử kiểu `int` có cùng tên là A và được gán các chỉ số từ 0 đến 9 để phân biệt

Hợp ngữ truy cập các phần tử trong mảng hoàn toàn tương tự như đối với cấu trúc `struct`. Có nghĩa là phần tử `A[3]` sẽ được lưu tại vị trí `[A + 12]` (Lưu ý rằng phần tử đầu tiên của mảng A có chỉ số là 0)

Trong ngôn ngữ Pascal, mảng được khai báo

```
A : array [-10..10] of integer ;
```

Cách khai báo như trên được chấp nhận. Khi đó, Pascal tạo ra mảng A gồm 21 phần tử kiểu `int` có chỉ số từ -10 đến 10

Việc biên dịch ngôn ngữ Pascal do đó phức tạp hơn so với C bởi hợp ngữ cũng có chỉ số bắt đầu từ 0. Để tính được phần tử `A[i]` bất kỳ, ta sử dụng công thức chuyển đổi

Địa chỉ phần tử i = Địa chỉ cơ sở + (Chỉ số - Chỉ số bắt đầu) * Size

Với khai báo như trên, để nạp phần tử `A[5]` vào thanh ghi `%r1` ta phải biên dịch thành chuỗi lệnh sau

```
Ld   %r0, 5, %r3
Ld   %r0, -10, %r4
Sub  %r3, %r4, %r6
Sll  %r6, 2, %r6
Ld   [A + %r6], %r1
```

5.1.5. Các lệnh toán học

Các lệnh toán học được biên dịch rất đơn giản vì các phép toán được thể hiện trong ngôn ngữ bậc cao đều có câu lệnh hợp ngữ tương ứng. Quá trình biên dịch chỉ có một chút phức tạp đối với máy tính dạng load-store như máy tính ARC

hoặc các máy tính dựa trên kiến trúc RISC. Với máy tính này, phép toán trước khi được thực hiện, các toán hạng phải được đưa vào các thanh ghi. Mặc dù số lượng các thanh ghi trong máy tính luôn luôn tăng lên trong lịch sử phát triển, tuy nhiên vẫn luôn có trường hợp số lượng thanh ghi cần sử dụng lớn hơn số lượng thanh ghi mà máy tính có. Trong trường hợp này, trình biên dịch sẽ tạm thời lưu trữ các biến trong bộ nhớ stack, kỹ thuật này còn có tên là ***register spill***. Khi đó, trình biên dịch sẽ sử dụng một kỹ thuật phức tạp có tên là ***register coloring*** để quyết định sẽ tiếp tục lưu trữ thanh ghi nào, và nội dung thanh ghi nào sẽ phải được lưu trữ trên bộ nhớ stack để nhường thanh ghi đó cho một biến khác, thậm chí, kỹ thuật register coloring còn có khả năng xác định xem có cần thiết phải lưu lại một nội dung của một thanh ghi nữa hay không

5.1.6. Luồng chương trình

Để điều khiển luông chương trình, tất cả các ngôn ngữ lập trình đều sử dụng các lệnh rẽ nhánh có điều kiện hoặc không có điều kiện, kết hợp với việc sử dụng các cờ flags thông báo trạng thái. Trong phần này, chúng ta sẽ nghiên cứu việc biên dịch việc điều khiển luồng chương trình thông dụng

Lệnh nhảy rẽ nhánh không điều kiện `goto`

Đây là dạng rẽ nhánh đơn giản nhất, lệnh nhảy `goto label`. Câu lệnh ngôn ngữ bậc cao này được thực hiện một cách đơn giản trong ARC bằng lệnh

```
ba label
```

Cấu trúc rẽ nhánh `if-then-else`

Trong ngôn ngữ C, cấu trúc `if-then-else` có cấu trúc

```
If (expr) stmt1 else stmt2;
```

Hoạt động của cấu trúc như sau. Nếu mệnh đề `expr` có giá trị là `true`, hành động `stmt1` sẽ được thực thi, nếu không thì hành động `stmt2` sẽ được thực thi. Do đó, trình biên dịch phải đánh giá được giá trị của `expr` và thực hiện một trong hai hành động phụ thuộc vào `expr.` Giả sử điều kiện `expr` được mô tả rất đơn giản là kiểm tra nội dung của 2 thanh ghi `%r1==%r2`. Khi đó cấu trúc này được thể hiện bởi các lệnh sau

```
Subcc  %r1, %r2, %r0 ! Kiểm tra sự bằng nhau của %r1 và %r2
```

```
Bne Hdong2          ! Nếu %r1 khác %r2 thì nhảy đến hdong2
...                 ! Các lệnh thuộc hành động 1
Ba ketthuc
Hdong2:
...                 ! Các lệnh thuộc hành động 2
Ketthuc:
```

***Cấu trúc vòng lặp* `while`**

Trong ngôn ngữ C, cấu trúc này được thể hiện

```
While (expr) stmt;
```

Cấu trúc này có nghĩa là nếu mệnh đề `expr` là `true` thì hành động `stmt` sẽ được thực hiện. Quá trình này sẽ được lặp đi lặp lại cho đến khi mệnh đề `expr` chuyển sang `false`. Để việc tìm hiểu về cấu trúc này đơn giản hơn, ta xem xét một ví dụ là tính tổng của 10 số lẻ liên tiếp bắt đầu từ số 5. Nếu viết bằng ngôn ngữ "giả C", đoạn chương trình này sẽ là

```
%r1=5;
%r2=10;
%r3=0;
%r4=0;
While (%r2!=%r3) {    %r4=%r4+%r1 ;
                      %r1=%r1+2;
                      %r3=%r3+1; }
```

Trình biên dịch sẽ chuyển sang hợp ngữ thành đoạn chương trình tương ứng như sau

```
Ld %r0, 5, %r1
Ld %r0, 10, %r10
```

```
    Andcc %r3, %r0, %r3
    Andcc %r4, %r0,%r4
    Ba kiemtra
True: add %r4, %r1, %r4
    Add %r1, 2, %r1
    Add %r3, 1, %r3
Kiemtra: subcc %r2, %r3, %r0
    Bneg True
```

Việc đặt mệnh đề kiểm tra `expr` ở trên hay ở dưới hành động `stmt` tùy thuộc vào lập trình viên. Người đọc có thể tự kiểm tra. Tác giả khuyến khích đặt mệnh đề kiểm tra ở dưới (như ví dụ trên) vì cách làm này có hiệu quả hơn nếu ta đặt mệnh đề kiểm tra ở trên

Cấu trúc `do-while`

Cấu trúc do-while được mô tả trong C như sau

```
    Do stmt while (expr)
```

Đây chỉ là cách biểu diễn khác của cấu trúc `while` ngoại trừ việc kiểm tra điều kiện sẽ được thực hiện sau khi hành động `stmt` được thực hiện tại chu kỳ đầu tiên. Việc biên dịch cấu trúc này cũng được thực hiện tương tự như cấu trúc `while` nhưng ta phải có biện pháp để loại bỏ ảnh hưởng của lệnh `ba` đầu tiên

Cấu trúc vòng lặp `for`

Trong C, cấu trúc này được biểu diễn bởi

```
    For (expr1; expr2; expr3) stmt;
```

Cấu trúc này có hiệu ứng hoàn toàn tương đương với

```
    Expr1;
    While (expr2) {  stmt;
                Expr3; }
```

Do đó, việc biên dịch cấu trúc này được thực hiện bằng cách kết hợp các cấu trúc trên

5.2. Quá trình chuyển sang ngôn ngữ máy

Quá trình chuyển từ ngôn ngữ hợp ngữ sang ngôn ngữ máy được gọi là quá trình assembly. Quá trình này được thực hiện rất đơn giản vì nó được thực hiện như việc tra bảng từng lệnh. Công việc này rất đơn giản nhưng rất dễ nhầm lẫn nếu thực hiện bằng tay. Thông thường, quá trình này được thực hiện bởi một phần mềm được gọi là ***assembler***

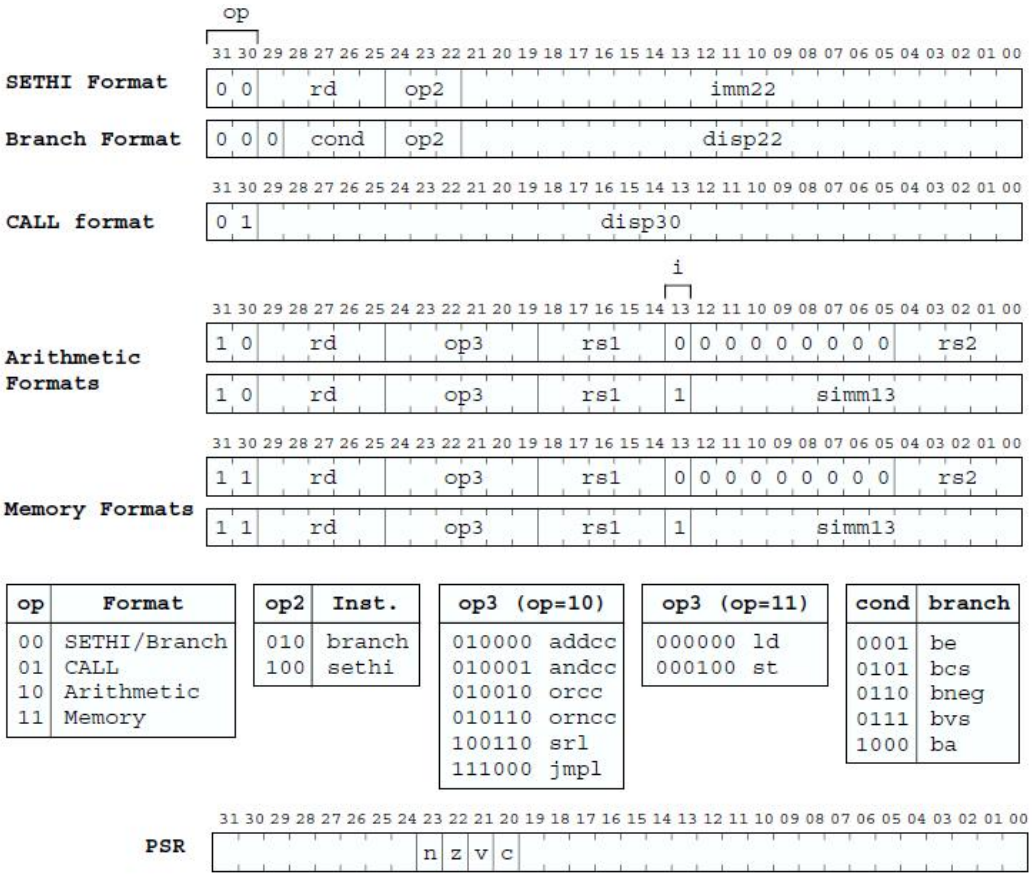
Một phần mềm assemble ít nhất phải có những khả năng sau

- Cho phép lập trình viên xác định chính xác nơi lưu trữ dữ liệu và chương trình tại thời điểm chạy chương trình
- Cung cấp phương tiện để lập trình viên khởi tạo giá trị dữ liệu trước khi chạy chương trình
- Cung cấp các phương án chuyển từ ngôn ngữ hợp ngữ sang ngôn ngữ máy cho tất cả các lệnh hợp ngữ và ở tất cả các chế độ địa chỉ
- Cho phép sử dụng các nhãn biểu tượng để biểu diễn địa chỉ và hằng số
- Cung cấp khả năng cho người lập trình xác định địa chỉ bắt đầu của chương trình
- Có cơ chế cho phép sử dụng các biến được định nghĩa trong 1 chương trình này được sử dụng trong một chương trình độc lập khác
- Cung cấp khả năng thực hiện chương trình con

Để tìm hiểu thêm về vấn đề này, chúng ta sẽ thử chuyển mã chương trình bằng tay một chương trình đơn giản đã được nêu ở trong chương trước, đó là chương trình tính tổng 2 số nguyên

```
! This programs adds two numbers
        .begin
        .org 2048
prog1:  ld      [x], %r1          ! Load x into %r1
        ld      [y], %r2          ! Load y into %r2
        addcc   %r1, %r2, %r3     ! %r3 ← %r1 + %r2
        st      %r3, [z]          ! Store %r3 into z
        jmpl    %r15 + 4, %r0     ! Return
x:      15
y:      9
z:      0
        .end
```

Hình 5.1. Chương trình cộng hai số nguyên



Hình 5.2. Cấu trúc lệnh trong ARC

Hình 5.2 cung cấp cho người đọc phương pháp mã hóa lệnh của máy tính ARC. Đây chính là những đặc tính của các trường nhị phân trong ngôn ngữ máy giúp người đọc chuyển hợp ngữ sang ngôn ngữ máy.

Quá trình chuyển ngữ kép

Hầu hết chương trình biên dịch đều thực hiện chuyển ngữ 2 lần, hay còn có tên là “two-pass assemblers”. Bước đầu tiên là xác định địa chỉ, dữ liệu và loại mã lệnh, đồng thời chọn những mã lệnh sẽ chuyển sang ngôn ngữ máy (nhưng không tạo mã máy)

Địa chỉ của dữ liệu và mã máy được chỉ ra bởi một bộ đếm được biết đến bởi tên gọi **location counter**. Bộ đếm này sẽ lưu lại địa chỉ của câu lệnh đang được xử lý hoặc dữ liệu đang được nạp vào vi xử lý. Bộ đếm này bằng 0 khi khởi tạo lúc chương trình bắt đầu chạy và tăng dần theo mỗi lệnh được thực thi. Lệnh giả `.org` có thể xác định được một giá trị của bộ đếm này bằng giá trị đi kèm với nó. Ví dụ như câu lệnh sau

```
.org 1000
```

Câu lệnh này sẽ xác định bộ đếm có giá trị là 1000 và lệnh kế tiếp nó sẽ có địa chỉ trong máy là các giá trị tiếp theo số 1000. Bộ đếm này sẽ tạo thành một bảng các con số tăng dần. Trong suốt bước 1 này, chương trình hợp ngữ cũng sẽ kiểm tra hoạt động của các lệnh, chèn thêm các định nghĩa các biến hay hằng số vào bảng số đó, quá trình này có tên là **symbol table**

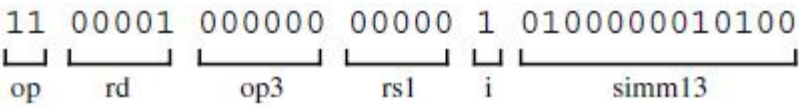
Nguyên nhân đòi hỏi phải thực hiện chuyển mã 2 lần là ta sẽ phải thực hiện việc "đưa" những biến số, hằng số mà ta đã thực hiện symbol table vào chương trình mà ta cần chuyển mã

Bây giờ ta sẽ thực hiện việc chuyển mã lệnh đầu tiên trong hình 5.1.

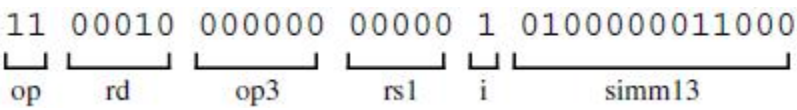
```
Ld   [x], %r1
```

Khi bộ đếm đếm tới lệnh đầu tiên (thương được biết tới là quá trình nạp câu lệnh). Câu lệnh này có ý nghĩa là nạp nội dung của ô nhớ có tên là `x`, tức là ô nhớ `%r0 + x` từ bộ nhớ vào thanh ghi %r1. Lệnh này tương ứng với cấu trúc Memory format trong hình 5.2. Nghiên cứu loại mã lệnh này ta thấy trường đầu tiên trong mã lệnh là `op` có giá trị là 11. Mã thanh ghi đích trong cấu trúc lệnh này được chứa trong trường `rd`, trong ví dụ trên sẽ là `00001` vì đây là thanh ghi `%r1`. Trường op3 có giá trị là `000000` đối với lệnh `ld` (đối chiếu với hình 5.2.). Trường `rs1` trong trường hợp này tương ứng với thanh ghi `%r0`. Giá trị `x` sẽ được cung cấp thông qua trường simm13. Bit i sẽ là bit tiếp theo. Bít này được sử dụng để phân biệt giữa các định dạng nhớ khác nhau. Trong trường hợp này `i=1`. Trường `simm13` tương ứng với địa chỉ x, vậy x phải la bao nhiêu ? Nhãn x xuất hiện sau 5 word sau lệnh đầu tiên có địa chỉ 2048. Mỗi word là 4 byte, do đó x sẽ có địa chỉ là 2048 + 5 * 4 = 2068. Biểu diễn bằng số nhị phân sẽ là `0100000010100`

Như vậy, mã lệnh của lệnh đầu tiên sẽ là



Tương tự, lệnh tiếp theo là



Tiếp tục thực hiện, ta có

```
ld [x], %r1          1100 0010 0000 0000 0010 1000 0001 0100
ld [y], %r2          1100 0100 0000 0000 0010 1000 0001 1000
addcc %r1,%r2,%r3    1000 0110 1000 0000 0100 0000 0000 0010
st %r3, [z]          1100 0110 0010 0000 0010 1000 0001 1100
jmpl %r15+4, %r0     1000 0001 1100 0011 1110 0000 0000 0100
15                   0000 0000 0000 0000 0000 0000 0000 1111
9                    0000 0000 0000 0000 0000 0000 0000 1001
0                    0000 0000 0000 0000 0000 0000 0000 0000
```

Nhìn chung, quá trình chuyển mã sẽ được thực hiện tuần tự từ đầu tới cuối, mã lệnh sẽ được tạo ra lần lượt theo từng câu lệnh. Sự khó khăn sẽ chỉ xuất hiện nếu trong chương trình xuất hiện các lệnh gọi chương trình con với từ khóa `call`.

```
                .
                .
                .
            call  sub_r      ! Subroutine is invoked here
                .
                .
                .
sub_r:      st    %r1, [w]   ! Subroutine is defined here
                .
                .
                .
```

Hình 5.3. Chương trình khung của lệnh call

Khi chương trình chuyển mã lệnh `call`, nó hoàn toàn chưa biết được `sub_r` là gì cho đến khi `sub_r` được tìm thấy. Do đó, chương trình sẽ đưa `sub_r` vào bảng **symbol table** và đánh dấu là chưa được giải quyết. Khi tìm thấy `sub_r`, chương trình sẽ thực hiện quá trình khắc phục.

Hoạt động của symbol table

Trong bước đầu của quá trình chuyển mã 2 bước, symbol table đã được tạo ra. Một nhãn hay một cái tên bất kỳ sẽ tương ứng với một giá trị được sử dụng trong suốt quá trình chuyển mã. Chúng ta sẽ tìm hiểu lại ví dụ trong hình 4.14 để làm rõ hơn hoạt động của symbol table

Bắt đầu từ lệnh giả `.begin` và bộ đếm sẽ có giá trị 2048 với lệnh giả `.org 2048.` Lệnh đầu tiên được tính đến là `a_start .equ 3000`. Với lệnh này, một biểu tượng `a_start` sẽ được thiết lập và nạp giá trị là 3000. Lưu ý

rằng lệnh này không tạo ra bất kỳ mã lệnh nào nên sẽ không chiếm địa chỉ trong bộ nhớ

```
! This program sums LENGTH numbers
! Register usage:       %r1 - Length of array a
!                       %r2 - Starting address of array a
!                       %r3 - The partial sum
!                       %r4 - Pointer into array a
!                       %r5 - Holds an element of a
            .begin                      ! Start assembling
          .org  2048          ! Start program at 2048
a_start   .equ  3000            ! Address of array a
            ld    [length], %r1 ! %r1 ← length of array a
            ld    [address],%r2 ! %r2 ← address of a
            andcc %r3, %r0, %r3 ! %r3 ← 0
loop:       andcc %r1, %r1, %r0 ! Test # remaining elements
            be    done          ! Finished when length=0
            addcc %r1, -4, %r1  ! Decrement array length
            addcc %r1, %r2, %r4 ! Address of next element
            ld    %r4, %r5      ! %r5 ← Memory[%r4]
            addcc %r3, %r5, %r3 ! Sum new element into r3
            ba    loop          ! Repeat loop.

done:     jmpl  %r15 + 4, %r0 ! Return to calling routine

length:             20          ! 5 numbers (20 bytes) in a
address:            a_start
            .org  a_start       ! Start of array a
a:                   25         ! length/4 values follow
                    -10
                     33
                     -5
                      7
            .end                        ! Stop assembling
```

Hình 5.4. Ví dụ hình 4.14

Quá trình chuyển mã sẽ bắt đầu với lệnh đầu tiên

```
Ld   [length], %r1
```

Lệnh này sẽ được chuyển mã tại địa chỉ đầu tiên được xác định bởi các lệnh giả tức là 2048. Bộ đếm sau đó sẽ tăng lên giá trị kế tiếp tương ứng với 4 byte, 2052 để thực hiện chuyển mã lệnh kế tiếp. Nhưng tại thời điểm đó, biến length không được chương trình nhận ra nên nó chưa biết giá trị là bao nhiêu. Trong bảng symbol, giá trị của nó tương ứng được ký hiệu “----”. Lệnh tiếp theo sẽ được xử lý tương tự như vậy.

Cho đến khi trình biên dịch gặp lại `length`, khi đó giá trị của bộ đếm là 2092. Giá trị của `length` sẽ được gán với giá trị đếm 2092, tức là nội dung của ô nhớ `length` sẽ là nội dung của ô nhớ có địa chỉ là 2092. Như vậy, ô nhớ `address` và `done` sẽ có giá trị tương ứng là 2096 và 2088. Nhãn done có giá trị là 2088 vì có 10 lệnh tương ứng 40 byte từ khi có lệnh đầu tiên cho đến khi gặp nhãn `done`

Sau khi bảng symbol được tạo ra, bước 2 của quá trình chuyển mã sẽ bắt đầu. Chương trình sẽ được đọc lại một lần nữa bắt đầu từ `.begin`, đồng thời mã lệnh sẽ được tạo ra. Lệnh đầu tiên được tạo ra là lệnh `ld` sẽ có địa chỉ là 2048. Khi tạo mã lệnh này, ô nhớ `length` sẽ được đối chiếu với bảng symbol để truy ra giá trị 2096. Với đầy đủ các yếu tố đó, lệnh `ld` này sẽ được đối chiếu là loại lệnh Memory format và tạo mã tương ứng. Quá trình chuyển mã sẽ được thực hiện tuần tự vào có kết quả tương ứng như hình 5.5

Location counter	Instruction		Object code
	.begin		
	.org	2048	
a_start	.equ	3000	
2048	ld	[length],%r1	11000010 00000000 00101000 00101100
2052	ld	[address],%r2	11000100 00000000 00101000 00110000
2056	andcc	%r3,%r0,%r3	10000110 10001000 11000000 00000000
2060 loop:	andcc	%r1,%r1,%r0	10000000 10001000 01000000 00000001
2064	be	done	00000010 10000000 00000000 00000110
2068	addcc	%r1,-4,%r1	10000010 10000000 01111111 11111100
2072	addcc	%r1,%r2,%r4	10001000 10000000 01000000 00000010
2076	ld	%r4,%r5	11001010 00000001 00000000 00000000
2080	ba	loop	00010000 10111111 11111111 11111011
2084	addcc	%r3,%r5,%r3	10000110 10000000 11000000 00000101
2088 done:	jmpl	%r15+4,%r0	10000001 11000011 11100000 00000100
2092 length:		20	00000000 00000000 00000000 00010100
2096 address:		a_start	00000000 00000000 00001011 10111000
	.org	a_start	
3000 a:		25	00000000 00000000 00000000 00011001
3004		-10	11111111 11111111 11111111 11110110
3008		33	00000000 00000000 00000000 00100001
3012		-5	11111111 11111111 11111111 11111011
3016		7	00000000 00000000 00000000 00000111
	.end		

Hình 5.5. Kết quả chuyển mã chương trình

5.3. Liên kết chương trình và nạp chương trình

Hầu hết các chương trình ứng dụng đều được cấu tạo từ một vài modul. Mỗi một modul này được xây dựng từ các ngôn ngữ lập trình khác nhau và chúng có thể được cung cấp bởi các thư viện khác nhau, trên các môi trường lập trình khác nhau, thậm chi trên các hệ điều hành khác nhau. Mỗi một modul có các thông tin đặc trưng khác nhau cho nên chúng cần được kết nối để nạp chương trình và thực thi

Chương trình kết nối linker là chương trình phần mềm để kết hợp các chương trình hợp ngữ khác nhau thành 1 chương trình độc lập. Chương trình linker sẽ phải giải quyết tất cả các vấn đề như các biến toàn cục, biến địa phương, giải quyết các địa chỉ các biến trên các chương trình hoàn toàn độc lập. Chương trình cuối cùng sẽ được nạp vào bộ nhớ bằng chương trình loader

Thư viện kết nối động (dynamic link libraries - DLLs) được xây dựng bởi Microsoft trên hệ điều hành Windows được sử dụng rất phổ biến.Chúng ta sẽ tìm hiểu thêm về các thư viện liên kết này ở những phần sau trong mục này

5.3.1. Liên kết chương trình

Để kết hợp các chương trình độc lập với nhau, chương trình kết nối cần phải

- Giải quyết vấn đề các địa chỉ ở các modul khác nhau phải được liên kết lại
- Xác định lại vị trí của các modul trong bộ nhớ bằng cách kết hợp. Trong suốt quá trình này, rất nhiều địa chỉ ở trong các modul phải được thay đổi tương ứng với chương trình mới được tạo ra
- Xem xét lại các biểu tượng được sử dụng trong các modul khác nhau
- Nếu các modul nằm trong các phân đoạn khác nhau, chương trình linker phải nhận diện và thâm nhập vào các phân đoạn khác nhau

Liên kết các địa chỉ ở các modul

Để giải quyết vấn đề địa chỉ của các biến và hằng số ở các modul khác nhau, chương trình liên kết cần phải phân biệt các biểu tượng ở các biến địa phương với các biểu tượng ở các biến toàn cục. Điều này được thực hiện bởi các lệnh giả `.global` và `.extern`. Lệnh giả `.global` sẽ gán cho các biểu tượng trong modul là các biến toàn cục và cho phép các modul khác truy cập. Lệnh giả `.extern` chỉ ra rằng nhãn mà nó trỏ tới được sử dụng trong 1 modul nhưng lại được định nghĩa trong một modul khác. Do đó, lệnh giả `.global` được sử

dụng trong modul mà các biểu tượng được định nghĩa ở đó còn lệnh giả `.extern` được sử dụng ở những nơi mà các modul cần sử dụng các biến trong `.global`. Có một điều chú ý là địa chỉ các nhãn có thể là toàn cục hoặc là địa phương

Tất cả các nhãn được định nghĩa trong một chương trình nhưng sẽ sử dụng trong một chương trình khác, ví dụ như chương trình con sẽ được khai báo với cấu trúc

```
.global symbol1, symbol2,...
```

Tất cả các nhãn ở các modul địa phương, có cùng tên nhãn và được sử dụng trong lớn hơn một modul khác sẽ được khai báo

```
.extern symbol1, symbol2,...
```

Ví dụ dưới đây mô tả cách sử dụng của .global và .extern

```
! Main program
      .begin
      .org   2048
      .extern sub
main: ld     [x], %r2
      ld     [y], %r3
      call   sub
      jmpl   %r15 + 4, %r0
   x: 105
   y:  92
      .end
```

```
! Subroutine library
      .begin
ONE   .equ      1
      .org    2048
      .global sub
sub: orncc   %r3, %r0, %r3
     addcc   %r3, ONE, %r3
     jmpl    %r15 + 4, %r0
     .end
```

Hình 5.6. Ví dụ minh họa về `.global` và `.extern`

Xác định lại vị trí các modul trong bộ nhớ

Trong ví dụ ở hình 5.6, ta thấy rằng cả 2 chương trình đều được bắt đầu ở cùng một địa chỉ là 2048, do vậy chúng không thể tồn tại ở cùng một thời điểm bởi nếu như vậy sẽ dẫn tới hiện tượng xung đột bộ nhớ. Để giải quyết vấn đề này các biểu tượng phải được đặt lại địa chỉ trong quá trình liên kết. Ý tưởng để giải quyết là chương trình thay vì biên dịch ở địa chỉ 2048 thì nó sẽ biên dịch ở địa chỉ 3000 chẳng hạn, và do đó độ lệch địa chỉ sẽ là 3000 – 2048 = 925. Việc này

sẽ được chương trình liên kết quản lý. Do đó sẽ không có một chương trình con nào được biên dịch ở cùng một địa chỉ so với chương trình chính.

5.3.2. Nạp chương trình

Chương trình nạp loader là chương trình được sử dụng để nạp chương trình lên bộ nhớ. Nhiệm vụ của loader là nạp các phân vùng bộ nhớ khác nhau với các đặc thù như các giá trị thanh ghi điều khiển ví dụ như `%sp`, bộ đếm chương trình, `%pc` chuyển nó thành giá trị tương ứng để chương trình có thể hoạt động bình thường

Nếu tại một thời điểm chỉ có 1 chương trình được nạp, việc thực thi chương trình hoạt động rất đơn giản. Tuy nhiên, trong các hệ điều hành hiện đại, một vài chương trình có thể được nạp cùng lúc, do đó không có cách nào để linker hay loader có thể biết được địa chỉ mà nó đang xử lý. Chương trình nạp cần xác định lại địa chỉ của các modul bằng cách nạp vào địa chỉ offset đối với toàn bộ các lệnh trong modul. Phương pháp này có tên là **relocating loader**. Phương pháp nạp relocating không lặp lại công việc của chương trình liên kết linker. Chương trình liên kết linker sẽ kết hợp một vài modul thành một modul duy nhất, trong khi đó, bộ nạp loader đánh lại địa chỉ của một chương trình duy nhất để nhiều chương trình có thể đồng thời nạp vào bộ nhớ chương trình. Phương pháp nạp thứ 2 có tên **linking loader**. Phương pháp này có chức năng của cả chương trình kết nối linker và chương trình nạp loader: xác định lại địa chỉ các biến, đánh lại địa chỉ các modul và nạp vào bộ nhớ

Chương trình liên kết sẽ tạo ra file chứa các thông tin đặc tả lại quá trình liên kết như địa chỉ bắt đầu của chương trình, các thông tin đánh lại địa chỉ và các điểm bắt đầu của các chương trình con

Một cách khác để nạp bộ nhớ chương trình là sử dụng chương trình quản lý bộ nhớ **memory managemet unit MMU**. Chương trình này sẽ phân chia bộ nhớ vật lý thành từng phân vùng (segment). Việc truy cập một ô nhớ sẽ là truy cập vào phân vùng tương ứng và địa chỉ offset. Các chương trình khác nhau có thể có cùng địa chỉ offset nhưng sẽ có địa chỉ segment khác nhau

Thư viện liên kết động DLL

Quay trở lại về thư viện liên kết động DLL, khái niệm này có một số tính năng hấp dẫn. Các biến của các chương trình con thường xuyên được sử dụng ví dụ như chương trình quản lý bộ nhớ hay quản lý đồ họa sẽ được lưu vào một vị trí, đó là thư viện DDL. Điều này làm cho kích thước chương trình nhỏ hơn bởi vì mỗi chương trình không cần phải lưu lại một bản sao DDL mà thông thường phải có. Tất cả các chương trình chia sẻ các đoạn mã của mình ngay cả trong lúc thực hiện. Hơn nữa, thư viện DDL còn được nâng cấp khả năng phân tích lỗi hoặc các chức năng khác. Như thế, các chương trình sử dụng tới nó không cần phải chuyển mã hay liên kết lại

5.4. Macro

Trong quá trình sử dụng chương trình con, chúng ta nhận thấy là một số lượng các thanh ghi sẽ được lưu vào và lấy ra từ stack trong quá trình gọi chương trình con và quay về chương trình chính. Để cất thanh ghi %r15 vào stack, hệ thống phải thực hiện 2 quá trình được thể hiện bởi 2 dòng lệnh sau

```
Addcc   %r14, -4- %r14  ! Giảm giá trị thanh ghi stack pointer
St   %r15, %r14         ! Cất nội dung %r15 vào stack
```

Trong trường hợp này, hành động xảy ra chỉ bao bồm 2 lệnh. Tuy nhiên nếu hành động xảy ra lặp đi lặp lại nhiều lần bao gồm nhiều lệnh thì việc viết đi viết lại đoạn chương trình lặp lại trở nên rất nhàm chám. Do đó, ta có thể thay thế đoạn chương trình đó bằng chương trình macro. Trong ví dụ trên, ta có thể thay thế 2 câu lệnh bằng một lời gọi macro

```
Push   %r15
```

Trong đó push là một macro với nội dung sau được thể hiện trong hình 5.9

```
! Macro definition for 'push'
.macro    push arg1         ! Start macro definition
addcc     %r14, -4, %r14    ! Decrement stack pointer
st        arg1, %r14        ! Push arg1 onto stack
.endmacro                   ! End macro definition
```

Hình 5.9. Nội dung macro

Một macro được định nghĩa bởi lệnh giả `.macro` và kết thúc bởi `.endmacro`. Ngay sau lệnh giả `.macro` là tên của macro do người sử dụng đặt ra và các toán tử là các biến địa phương được sử dụng trong thân chương trình macro.

TỔNG KẾT CHƯƠNG

Trong chương này, chúng ta đã tìm hiểu một số vấn đề quan trọng sau

- Quá trình biên dịch chương trình từ ngôn ngữ cấp cao như C hay Pascal sang hợp ngữ bao gồm các nhóm lệnh dịch chuyển dữ liệu, nhóm lệnh toán học và nhóm lệnh điều khiển luồng chương trình
- Quá trình chuyển hợp ngữ sang ngôn ngữ máy – quá trình chuyển mã 2 lần
- Kết nối các modul chương trình thành 1 chương trình lớn
- Sử dụng macro

CHƯƠNG 6: ĐIỀU KHIỂN LUỒNG DỮ LIỆU

Trong những chương trước, chúng ta đã tìm hiểu về máy tính ở các mức ứng dụng, mức ngôn ngữ lập trình bậc cao, mức hợp ngữ. Trong chương 4, chúng ta đã giới thiệu nội dung của kiến trúc tập lệnh ISA, tác động của tập lệnh lên các thanh ghi và bộ nhớ. Trong chương này, chúng ta sẽ tìm hiểu một bộ phận của máy tính chịu trách nhiệm thực thi các lệnh, đó là bộ điều khiển của CPU. Với nội dung đó, chúng ta sẽ tìm hiểu máy tính ở mức vi chương trình. Cấu trúc vi chương trình bao gồm các bộ điều khiển và các thanh ghi lập trình mà người sử dụng không được phép tương tác, các khối chức năng như ALU, các thanh ghi có chức năng bổ xung mà các bộ điều khiển yêu cầu

Mỗi một kiến trúc tập lệnh khác nhau có thể có cấu trúc vi chương trình khác nhau. Ví dụ, cùng một kiến trúc ISA của máy tính Intel Pentium nhưng sẽ có nhiều phương thức điều khiển khác nhau. Không chỉ với Intel, một số đối thủ cạnh tranh khác như AMD hay Cyrix cũng thực hiện dựa trên kiến trúc tập lệnh ISA nhưng phương pháp thực hiện hoàn toàn khác so với Intel. Một kiến trúc vi chương trình có thể tối ưu tốc độ hoạt động, có thể tối ưu tiết kiệm năng lượng hoặc cũng có thể là tiết kiệm giá thành sản xuất. Việc kiến trúc vi chương trình thay đổi trong khi vẫn giữ nguyên kiến trúc tập lệnh ISA sẽ làm các nhà sản xuất IC có thể tận dụng được công nghệ IC và công nghệ bộ nhớ và chỉ cần người sử dụng thay đổi phần mềm

Trong chương này, chúng ta sẽ nghiên cứu 2 thái cực của kiến trúc vi chương trình là điều khiển vi chương trình bằng phần mềm và điều khiển vi chương trình bằng phần cứng và xem xét hoạt động của vi xử lý trên nền tảng của 2 kiến trúc đó

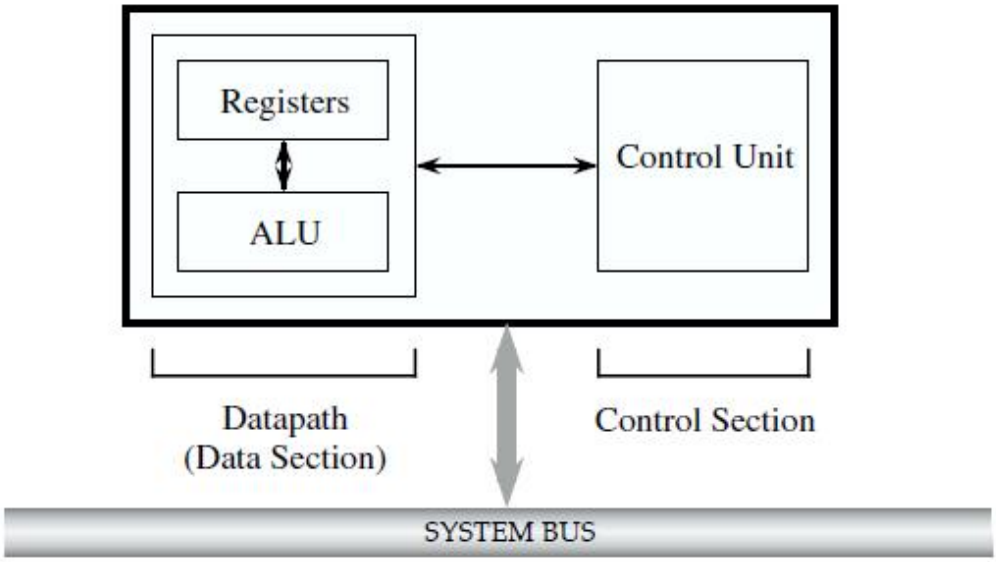
6.1. Cơ sở vi kiến trúc

Các chức năng của vi kiến trúc xoay quanh việc giải mã-thực thi lệnh, do đó được coi là “trái tim” của máy tính. Như đã thảo luận ở chương 4, các bước để thực thi một chu kỳ lệnh là

1. Nạp lệnh kế tiếp được thực thi từ bộ nhớ
2. Giải mã lệnh
3. Nạp các toán hạng từ bộ nhớ hay thanh ghi nếu cần

4. Thực thi lệnh và lưu kết quả
5. Trở về bước đầu tiên

Vi kiến trúc chịu trách nhiệm hoạt động của 5 bước trên. Vi kiến trúc điều khiển việc nạp lệnh kế tiếp sẽ được thực thi vào vi xử lý, kiểm tra các lệnh đó, nạp các toán hạng nếu cần, thực thi lệnh, lưu kết quả và lặp lại quá trình.



Hình 6.1. Cấu trúc của vi kiến trúc

Vi kiến trúc bao gồm khối dữ liệu chứa các thanh ghi và ALU và khối điều khiển như trên hình 6.1. Khối dữ liệu cũng được biết đến là **luồng dữ liệu**. Bộ điều khiển Control Unit có thể được xây dựng bởi một trong 2 thái cực như đã nói ở trên. Với thái cực sử dụng vi chương trình, việc điều khiển hoạt động của các luồng dữ liệu hoàn toàn là do chương trình, chức năng đặc biệt này có tên **microprogram**. Đây là chức năng người lập trình không can thiệp Thông thường microprogram được cấu thành từ nhiều "chương trình nhỏ" hoạt động như các macro. Ngược lại là bộ điều khiển Control Unit được xây dựng từ phần cứng. Tư tưởng của bộ điều khiển bằng phần cứng là bộ điều khiển cho rằng mỗi một lệnh sẽ được thực hiện bởi một số bước hữu hạn, bộ điều khiển sẽ tác động để hệ thống chuyển lần lượt từ trạng thái này sang trạng thái khác, mỗi trạng thái là một bước của quá trình thực hiện một bước của một lệnh.

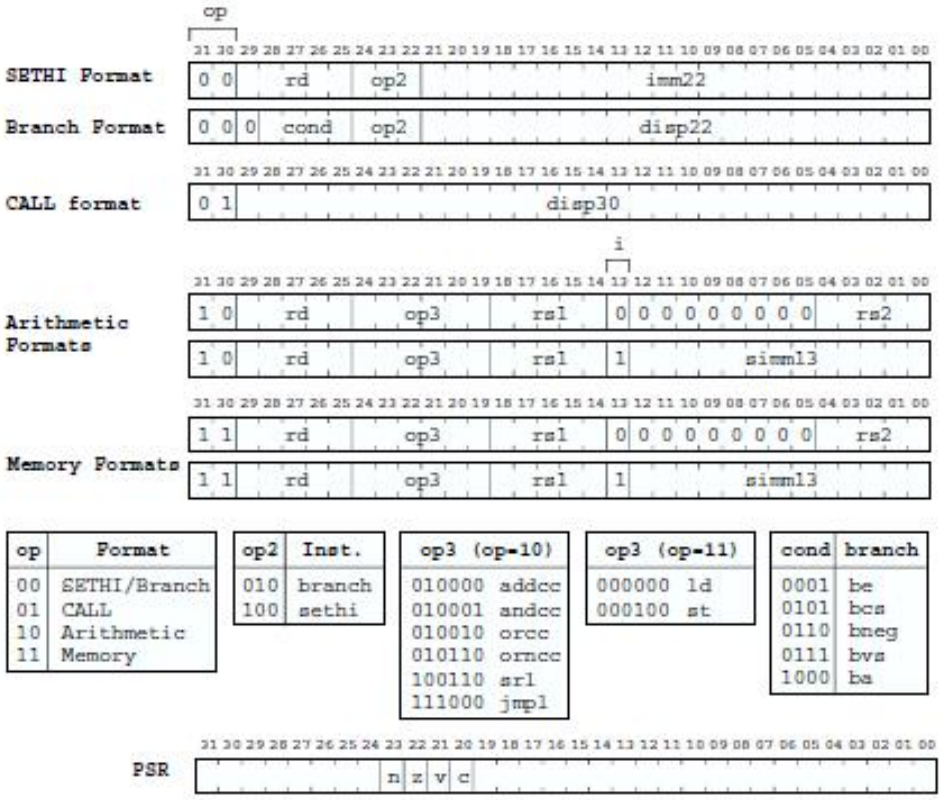
Với 2 thái cực vi kiến trúc bằng phần mềm, microprogram và vi kiến trúc bằng phần cứng hardwired, thành phần các thanh ghi và ALU sẽ có cấu tạo hoàn toàn giống nhau. Điều khác biệt chỉ là ở phần điều khiển Control Unit. Trong những phần tiếp theo, chúng ta sẽ nghiên cứu kỹ hơn về 2 kiểu kiến trúc vi chương trình này.

6.2. Kiến trúc vi mẫu của máy tính ARC

Trong phần này, chúng ta sẽ tìm hiểu bộ điều khiển Control Unit dưới cách tiếp cận bằng phần mềm, hay còn được gọi là cấu trúc vi chương trình. Chúng ta sẽ bắt đầu từ việc mô tả luồng dữ liệu và các tín hiệu điều khiển nó.

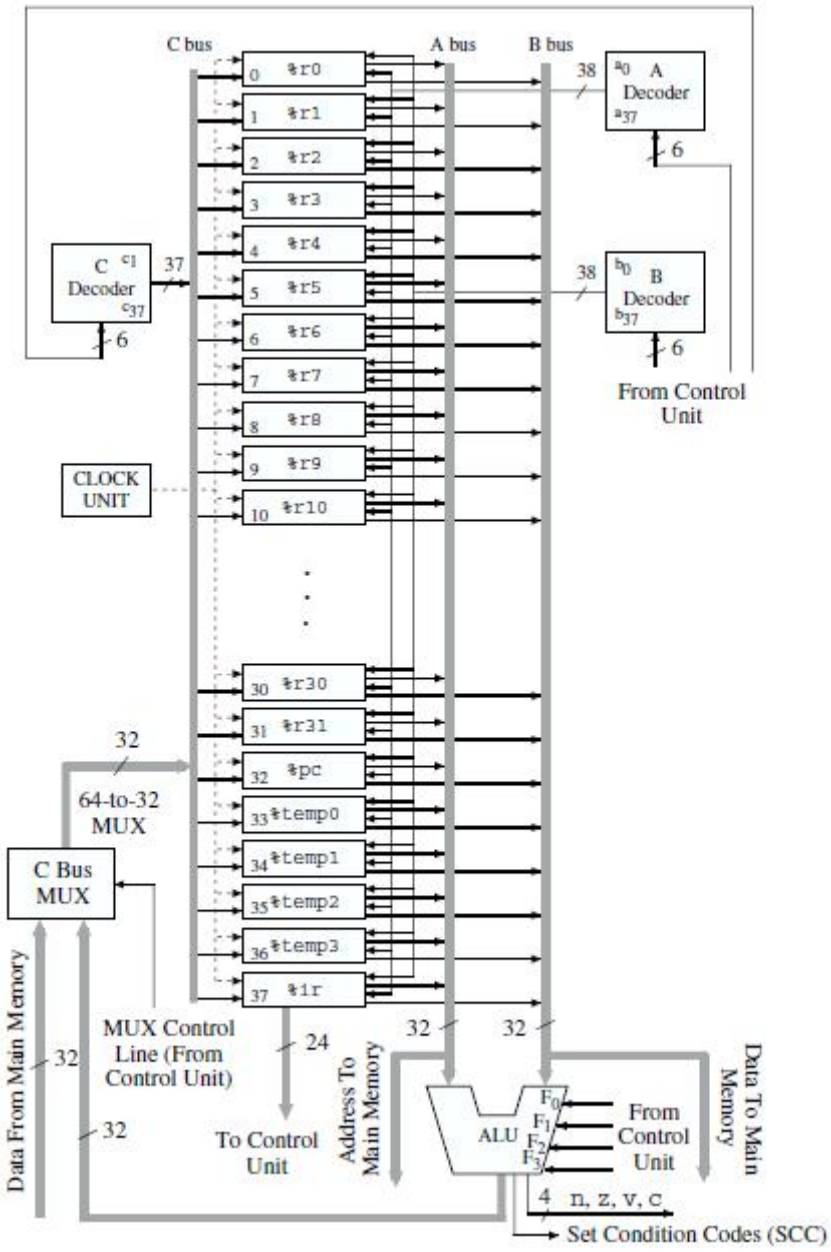
Hình 6.2 nhắc lại về tập lệnh và cấu trúc tập lệnh ARC đã được mô tả trong chương 4. Chúng ta có 15 lệnh được chia thành 4 nhóm phân biệt bởi 2 bit tận cùng bên trái. Ngoài ra chúng ta còn có thanh ghi trạng thái `%psr`

Mnemonic	Meaning
ld	Load a register from memory
st	Store a register into memory
sethi	Load the 22 most significant bits of a register
andcc	Bitwise logical AND
orcc	Bitwise logical OR
orncc	Bitwise logical NOR
srl	Shift right (logical)
addcc	Add
call	Call subroutine
jmpl	Jump and link (return from subroutine call)
be	Branch if equal
bneg	Branch if negative
bcs	Branch on carry
bvs	Branch on overflow
ba	Branch always



Hình 6.2. Tập lệnh và cấu trúc tập lệnh ARC

6.2.1. Luồng dữ liệu



Hình 6.3. Cấu trúc luồng chương trình

Cấu trúc luồng dữ liệu được thể hiện trên hình 6.3. Cấu trúc vi chương trình bao gồm 32 thanh ghi mà người dùng nhìn thấy được (`%r0` đến `%r31`), thanh ghi đếm chương trình (`%pc`), thanh ghi con trỏ lệnh (`%ir`), khối ALU, 4 thanh ghi đệm mà người dùng không nhìn thấy được (`%temp0` đến `%temp3`), và các kết nối giữa các thành phần trên

Các thanh ghi `%r0` đến `%r31` là các thanh ghi cho phép truy cập bởi người dùng. Thanh ghi `%r0` luôn có giá trị bằng 0 và không thể thay đổi được. Thanh

ghi %pc là thanh ghi đếm chương trình được sử dụng để theo dõi các lệnh đọc từ bộ nhớ. Người dùng có thể truy cập trực tiếp thanh ghi %pc thông qua các lệnh `call` và `jmpl`. Các thanh ghi đệm temp được sử dụng trong quá trình biên dịch và không được truy cập bởi người dùng. Thanh ghi %ir luôn chứa lệnh đang được thực thi và cũng không được truy cập bởi người dùng

Khối ALU

Khối ALU thực hiện 1 trong 16 lệnh với 2 toán hạng được lấy từ 2 hệ thống bus A và B. 16 lệnh này được thể hiện trên hình 6.4. Kết quả của phép toán được lưu trữ trên hệ thống bus C nếu tín hiệu điều khiển bộ dồn kênh MUX không chứa dữ liệu từ bộ nhớ

F_3	F_2	F_1	F_0	Operation	Changes Condition Codes
0	0	0	0	ANDCC (A, B)	yes
0	0	0	1	ORCC (A, B)	yes
0	0	1	0	NORCC (A, B)	yes
0	0	1	1	ADDCC (A, B)	yes
0	1	0	0	SRL (A, B)	no
0	1	0	1	AND (A, B)	no
0	1	1	0	OR (A, B)	no
0	1	1	1	NOR (A, B)	no
1	0	0	0	ADD (A, B)	no
1	0	0	1	LSHIFT2 (A)	no
1	0	1	0	LSHIFT10 (A)	no
1	0	1	1	SIMM13 (A)	no
1	1	0	0	SEXT13 (A)	no
1	1	0	1	INC (A)	no
1	1	1	0	INCPC (A)	no
1	1	1	1	RSHIFT5 (A)	no

Hình 6.4. 16 lệnh của ALU

Lệnh ANDCC và lệnh AND đều có tác động là thực hiện phép toán AND từng bit lên 2 toán hạng trên bus A và B. Tuy nhiên ta lưu ý rằng các lệnh kết thúc bởi “CC” sẽ có tác động đến điều kiện dành cho lệnh nhảy. Do đó lệnh ANDCC

sẽ tác động đến điều kiện nhảy còn lệnh AND thì không. Tương tự như vậy là lệnh ORCC và OR, lệnh NORCC và NOR, lệnh ADDCC và lệnh ADD.

Lệnh SRL (lệnh dịch phải) sẽ dịch nội dung của bus A sang phải với số bit tương ứng nằm trong bus B (từ 0 đến 31 bit). Các số 0 sẽ được nhồi vào tận cùng bên trái khi tiến hành dịch phải. Lệnh LSHIFT2 và LSHIFT10 sẽ dịch nội dung bus A sang trái 2 hoặc 10 bit tương ứng. Các số 0 sẽ được nhồi vào bên phải

Lệnh SIMM13 sẽ lấy 13 bit có trọng số nhỏ nhất của dữ liệu trên bus A và thêm 19 số 0 vào các bit có trọng số lớn nhất. Lệnh SEXT13 thực hiện việc mở rộng 13 bit có trong số nhỏ nhất trên bus A thành một từ 32 bit. Tức là nếu bit tận cùng bên trái của 13 bit là 1 thì 19 số 1 sẽ được thêm vào phần có trọng số lớn nhất để tạo thành kết quả. Lệnh INC sẽ tăng giá trị trên bus A lên 1 đơn vị. Trong khi đó lệnh INCPC sẽ tăng giá trị trên bus A lên 4. Lệnh INCPC thường được sử dụng để làm tăng thanh ghi con trỏ lệnh `%pc`. Lệnh INCPC có thể tác động đến tất cả các thanh ghi có kết nối với bus A

Lệnh RSHIFT5 dịch toán hạng trên bus A sang phải 5 bit, chép bit có trọng số lớn nhất (bit dấu) vào vị trí 5 bit phía bên trái. Lệnh này có hiệu ứng là tạo ra 5 bit mở rộng để làm bit dấu. Khi thực hiện lệnh này 3 lần với toán hạng 32 bit sẽ tạo ra hiệu ứng chuyển bit có trọng số lớn nhất của trường COND trong cấu trúc Branch sang vị trí bít số 13

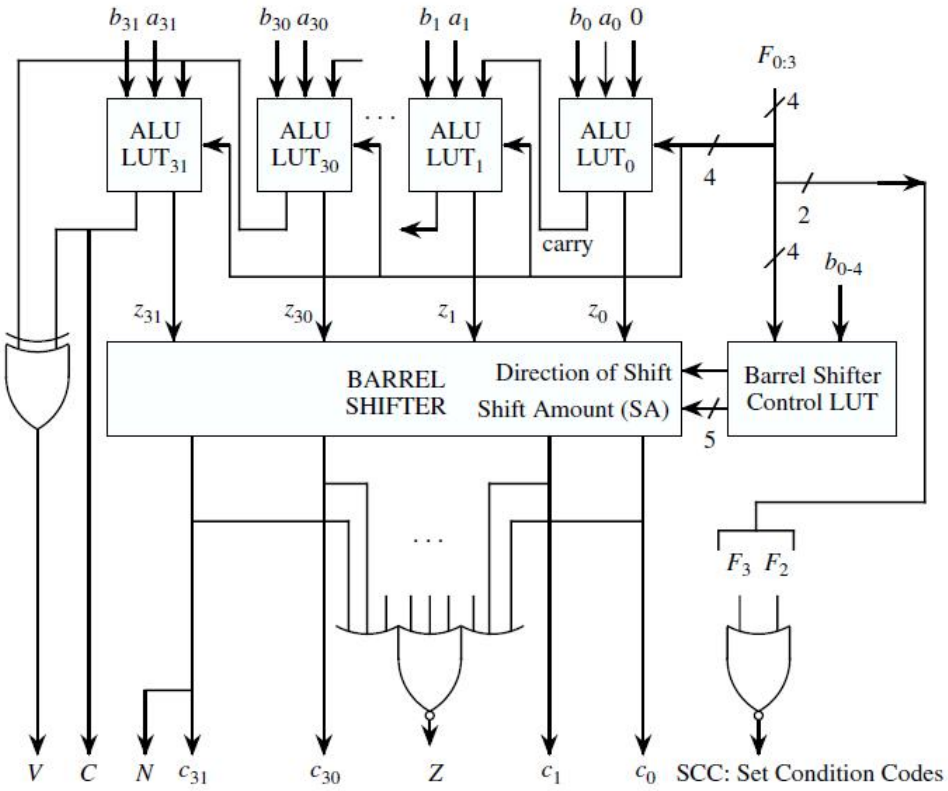
Tất cả các lệnh toán học và logic đều có thể được thực hiện bằng các lệnh trong ALU. Ví dụ như phép trừ A – B có thể được thực hiện bởi tổng của A với số -B là số bù 2 của B. Chú ý rằng trong lệnh ALU của máy tính ARC không hề có phép toán đảo. Để thực hiện phép toán đảo, ta có thể sử dụng phép toán NOR tác động với tất cả các bit của B.

ALU tạo ra các tín hiệu cờ c, n, z, và v. Các tín hiệu này tích cực nếu xảy ra các hiện tượng tương ứng

ALU có thể được cấu tạo theo nhiều cách. Một cách đơn giản nhất được gọi là phương pháp **lookup table** (LUT). Khối ALU có các đường dữ liệu vào A và B và 1 đường dữ liệu ra C. Các đường dữ liệu này có độ rộng 32 bit. Ngoài ra,ALU còn có 4 bit điều khiển F, 4 bit đưa ra các tín hiệu trạng thái và tín hiệu SCC được sử dụng để thiết lập các tín hiệu cờ trong thanh ghi `%psr`. Chúng ta có thể phân tích ALU thành 32 khối LUT, mỗi khối thực hiện các phép toán và

logic riêng biệt. Kết quả được đưa vào khối **barrel shifter** có tác dụng như bộ dịch trái phải. Sơ đồ khối của ALU được thể hiện trong hình 6.5.

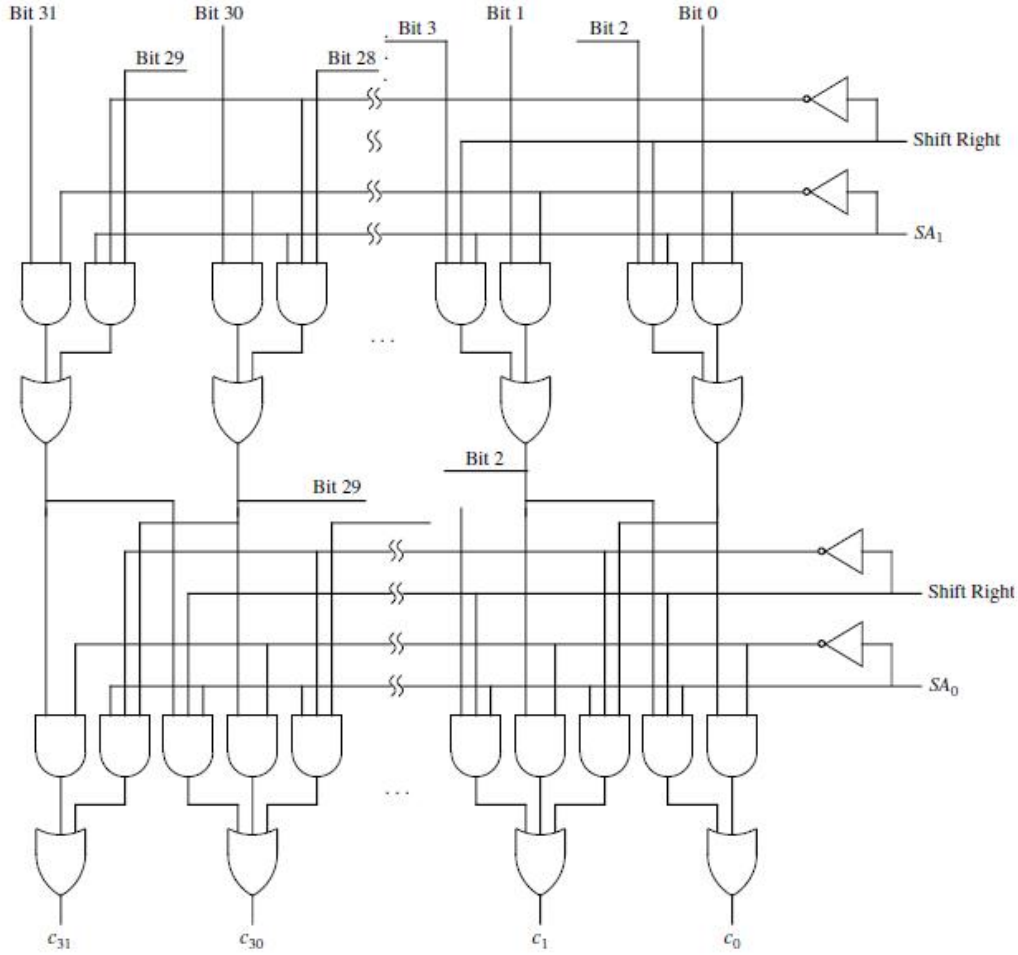
Khối barrel shifter sẽ dịch các dữ liệu đầu vào từ 0 đến 31 bit phụ thuộc vào các tính hiệu điều khiển đầu vào. Cấu tạo của bộ dịch barrel shifter được thể hiện trong hình 6.6.



Hình 6.5. Sơ đồ khối của ALU

Trên hình 6.6. chúng ta sẽ bắt đầu với phía dưới của mạch, ta sẽ thấy đầu ra của mạch sẽ tương ứng với đầu vào phía trên. Nếu tín hiệu SA_0 bằng 0 thì toàn bộ tín hiệu ở trên sẽ được chuyển xuống dưới mà không tiến hành dịch trái hay phải. Khi tín hiệu SA_0 bằng 1 thì hành động dịch mới xảy ra. Khi tín hiệu SA_0 bằng 1, tín hiệu Shift Right cũng bằng 1 thì đầu ra sẽ dịch sang phải, ngược lại sẽ dịch sang trái. Ở cụm mạch phía trên, hành động dịch cũng được tiến hành tương tự. Với nguyên lý đó, để thực hiện dịch từ 0 đến 31 bit thì chúng ta cũng có hệ thống mạch gồm 31 tầng. Số bit dịch chuyển sẽ được kích hoạt tương ứng khi hoạt động

Mỗi một khối LUT hoạt động độc lập với nhau. Chúng có cấu tạo nói chung là giống nhau ngoại trừ một số khác biệt khi thực hiện lệnh INC và INCPC. Về mặt cơ bản, ta có bảng chân lý của bộ LUT được thể hiện trong hình 6.7.



Hình 6.6. Cấu tạo bộ dịch barrel shifter

	F_3	F_2	F_1	F_0	*Carry In*	a_i	b_i	z_i	*Carry Out*
ANDCC	0	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	1	0	0
	0	0	0	0	0	1	0	0	0
	0	0	0	0	0	1	1	1	0
	0	0	0	0	1	0	0	0	0
	0	0	0	0	1	0	1	0	0
	0	0	0	0	1	1	0	0	0
	0	0	0	0	1	1	1	1	0
ORCC	0	0	0	1	0	0	0	0	0
	0	0	0	1	0	0	1	1	0
	0	0	0	1	0	1	0	1	0
	0	0	0	1	0	1	1	1	0
	0	0	0	1	1	0	0	0	0
	0	0	0	1	1	0	1	1	0
					.				.
					.				.
					.				.

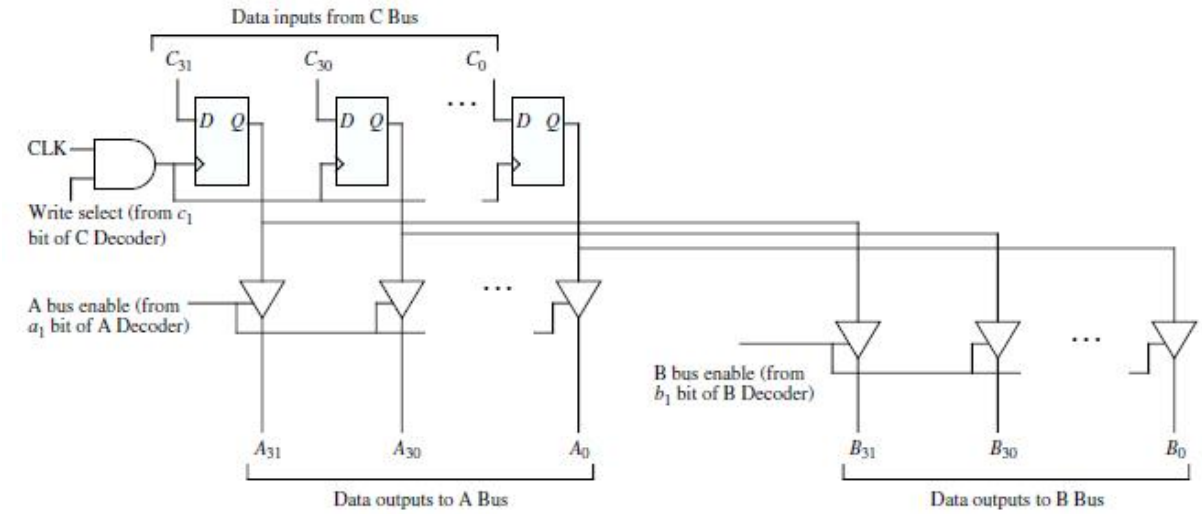
Hình 6.7. Bảng chân lý của LUT

Các tín hiệu điều khiển n, z, v và c hoạt động trực tiếp trong đó n và c được lấy ra trực tiếp từ c_{31} của bộ dịch barrel shifter và tín hiệu c từ LUT số 31

Lưu ý rằng chỉ có các lệnh kết thúc bởi "CC" mới tác động đến các điều kiện nhảy. Tín hiệu này được tạo ra và được đánh dấu bởi nhãn SCC (Set Condition Codes). Tín hiệu này là TRUE khi cả 2 tín hiệu F_3 và F_2 là FALSE

Các thanh ghi

Tất cả các thanh ghi trong ALU đều là các thanh ghi có tác động sườn xuống loại D. Điều đó có nghĩa là đầu ra của các thanh ghi không thay đổi khi tín hiệu xung nhịp chuyển trạng thái từ cao xuống thấp. Tất cả các thanh ghi đều có chung một dạng. Hình 6.8 dưới đây thể hiện thanh ghi `%r1`, các thanh ghi còn lại cũng có cấu tạo hoàn toàn như vậy



Hình 6.8. Cấu tạo của thanh ghi `%r1`

Tín hiệu xung nhịp CLK đưa vào thanh ghi `%r1` thông qua cổng AND với tín hiệu chọn thanh ghi c_1 từ bộ giải mã C decoder. Với cấu tạo của các thanh ghi `%r`$_i$ khác, ta sẽ đưa tín hiệu c_i tương ứng từ bộ giải mã. Tín hiệu vào các thanh ghi được lấy trực tiếp từ các đường dây trên hệ thống bus C. Đầu ra của thanh ghi được ghi vào hệ thống bus A hoặc bus B thông qua bộ đệm 3 trạng thái. Đầu ra của bộ đệm sẽ đưa vào bus A hoặc bus B bởi các tín hiệu điều khiển a_1 và b_1 của các bộ giải mã A decoder và B decoder, hai tín hiệu này sẽ chỉ có 1 tín hiệu được tích cực tại 1 thời điểm

Các thanh ghi còn lại có cấu tạo hoàn toàn tương tự. Tuy nhiên thanh ghi `%r0` luôn luôn mang giá trị là 0, không thay đổi được. Do đó thanh ghi `%r0` không có đầu vào từ bus C cũng như không có đầu vào từ bộ giải mã C decoder, và cũng không cần có các flip-flop. Thanh ghi `%ir` có các đầu ra đặc biệt tương ứng với các trường `rd, rs1, rs2, op, op2, op3` và trường bit 13, như hình 6.9. Các đầu ra được sử dụng bởi bộ điều khiển trong quá trình chuyển mã. Điều này sẽ được thể hiện kỹ hơn trong mục 6.2.4.

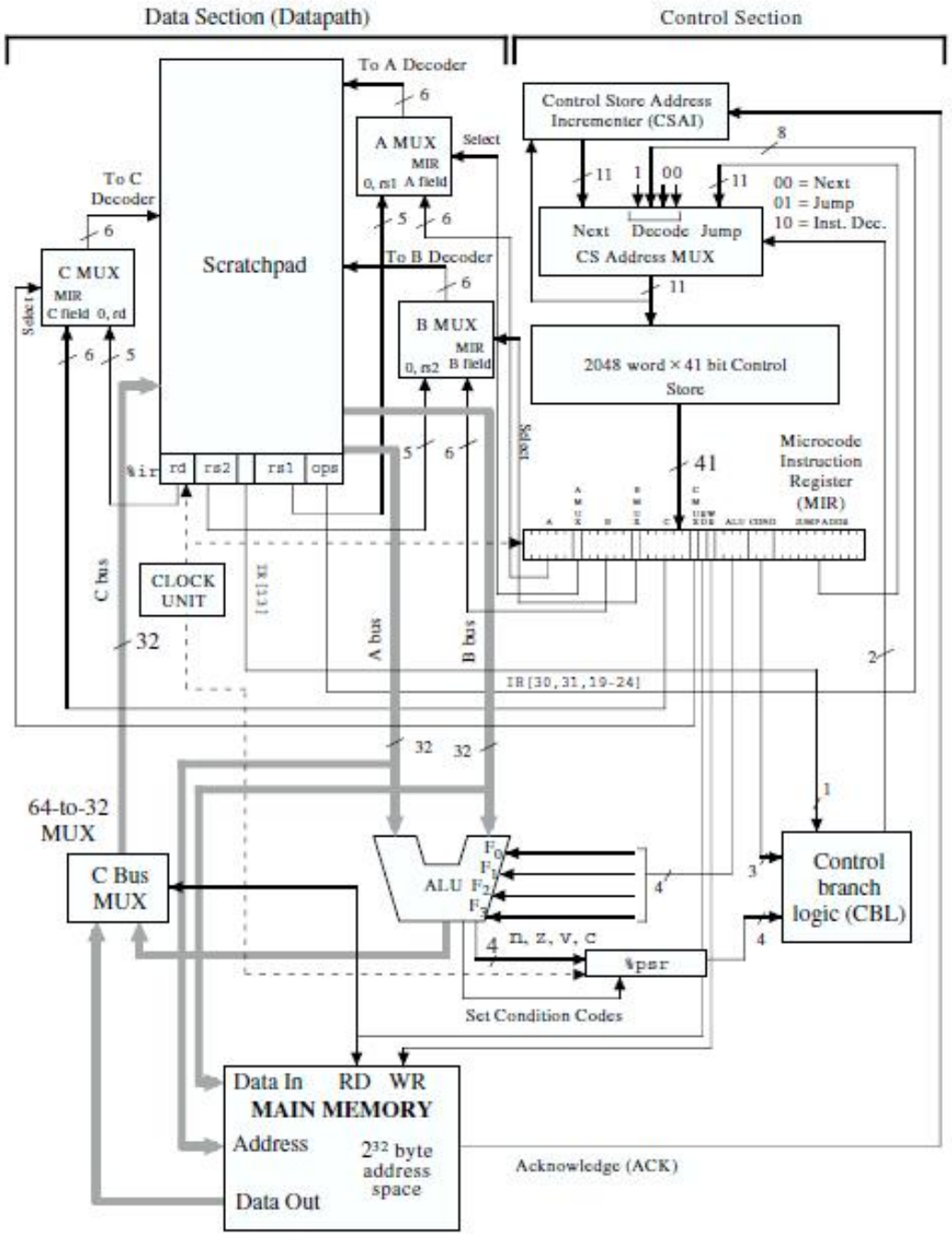
Các bộ giải mã A, B, C đơn giản là các bộ lựa chọn các tín hiệu. Mỗi bộ giải mã này có 6 bit tín hiệu vào có thể giải mã 64 tín hiệu khác nhau. Tuy nhiên chúng ta chỉ cần giải mã 38 thanh ghi, trong đó thanh ghi `%r0` không cần giải mã. Do đó, ta sẽ sử dụng tất cả 37 tín hiệu giải mã

6.2.2. Khối điều khiển Control section

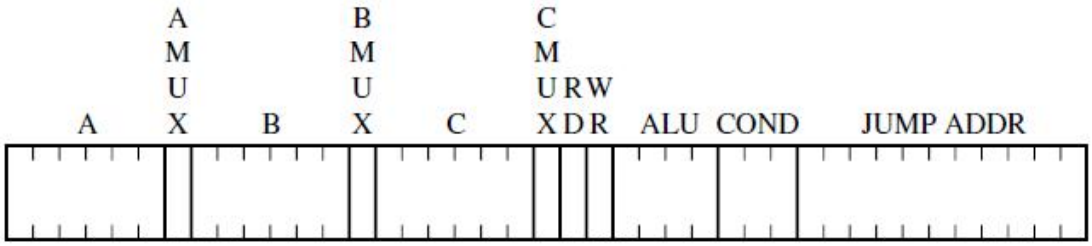
Vi kiến trúc theo cấu trúc vi chương trình được thể hiện như hình 6.10. Trên hình thể hiện cả khối luồng dữ liệu, khối điều khiển và kết nối giữa các khối. Trung tâm của khối điều khiển là vùng nhớ kích thước 2048 word x 41 bit. Vùng nhớ này chứa dữ liệu theo hàng, mỗi một hàng điều khiển một bước trong việc thực thi lệnh. Vùng nhớ ROM này sẽ có tên là **control store** trong tài liệu này. Mỗi một từ 41 bit được gọi là một vi lệnh **microinstruction**. Khối điều khiển chịu trách nhiệm nạp vi lệnh, tương tự như với các máy tính tương tự ARC khác. Hoạt động của các vi lệnh được điều khiển bởi thanh ghi vi lệnh MIR (microprogram instruction register), thanh ghi trạng thái `%psr`, và một cơ chế để xác định vi lệnh nào sẽ được thực thi tiếp theo. Cơ chế đó bao gồm khối điều khiển rẽ nhánh CBL (Control Branch Logic), và khối Control Store Address MUX. Ở đây cũng không cần thiết phải có một máy tính riêng biệt để quản lý, lưu trữ địa chỉ của vi lệnh kế tiếp bởi vì địa chỉ đó được tính toán trực tiếp trong mỗi chu kỳ lệnh

Khi vi cấu trúc bắt đầu hoạt động, giả sử là lúc bắt đầu cung cấp nguồn, mạch reset (không thể hiện trên hình) sẽ nạp vi lệnh ở địa chỉ 0 trong bộ nhớ control store vào thanh ghi vi lệnh MIR và thực thi nó. Bắt đầu từ thời điểm này, các lệnh kế tiếp được thực thi từ các khối Next, Decode hay khối Jump trong khối CS Address MUX phụ thuộc vào trường COND trong thanh ghi vi lệnh MIR và khối điều khiển rẽ nhánh và logic CBL. Mỗi khi một vi lệnh được đặt vào MIR, luồng dữ liệu tương ứng sẽ được thực hiện. Đó là luồng dữ liệu nào phụ thuộc

vào nội dung của các trường cụ thể trong thanh ghi MIR



Hình 6.10. Vi kiến trúc của ARC



Hình 6.11. Cấu trúc của một vi lệnh

Cấu trúc của một vi lệnh được thể hiện trên hình 6.11. Từ bên trái sang, bắt đầu là trường A gồm 6 bit. Tổ hợp của các bit này sẽ quyết định kích hoạt luồng dữ liệu tương ứng đi từ thanh ghi lên hệ thống bus A. Trường AMUX gồm 1 bit tiếp theo quyết định dữ liệu từ trường A decoder sẽ được nạp từ trường A trong thanh ghi MIR (AMUX = 0) hay được nạp từ trường `rs1` trong thanh ghi `%ir` (AMUX = 1). Tương tự như vậy, trường B và C sẽ chỉ ra thanh ghi được truy xuất lên hệ thống bus B và bus C. BMUX quyết định dữ liệu trên B decoder được lấy từ MIR (BMUX = 0) hay từ `rs2` trong %ir còn CMUX quyết định dữ liệu trên C decoder được lấy trên MIR (CMUX = 0) hay từ `rd` trong `%ir`

Tín hiệu RD và WR là các tín hiệu quyết định xem bộ nhớ được đọc hay ghi. Quá trình là đọc nếu RD = 1 và quá trình là ghi nếu WR = 1. Tín hiệu RD và WR sẽ không đồng thời là 1 tại cùng một thời điểm, nhưng có thể đồng thời là 0 khi không diễn ra quá trình truy cập bộ nhớ. Trong cả 2 quá trình đọc và ghi dữ liệu, địa chỉ ô nhớ sẽ được lấy trực tiếp trên hệ thống bus A, dữ liệu từ ALU đi vào bộ nhớ được lấy trên bus B còn dữ liệu từ bộ nhớ đi vào ALU được lấy trên bus C. Tín hiệu RD cũng điều khiển bộ dồn kênh 64 sang 32 (bộ MUX). Đây là bộ phận lựa chọn tín hiệu trên bus C sẽ được nạp từ bộ nhớ ngoài (RD = 1) hay được lấy từ ALU (RD = 0)

Trường ALU quyết định xem ALU sẽ thực thi hành động gì như trên hình 6.4. Có tất cả 16 lệnh mà ALU có thể thực hiện được tương ứng với 4 bit trong trường ALU. Điều này cũng có nghĩa là không có cách nào để “tắt” ALU khi không sử dụng, ví dụ như trong quá trình đọc và ghi bộ nhớ

Trường COND sẽ “chỉ dẫn” bộ vi điều khiển nạp vi lệnh kế tiếp từ bộ nhớ control store hoặc nạp từ một vị trí được chỉ ra trong trường JUMP ADDR trong thanh ghi MIR hoặc từ một mã lệnh trong thanh ghi `%ir`. Trường COND được biên dịch ra bảng thể hiện trong hình 6.12. Nếu COND = 000, sẽ không có lệnh nhảy nào được tực hiện, dữ liệu trong khối Next trong khối CS Address MUX sẽ được nạp. Khối Next này được tính toán bởi một bộ đếm CSAI (Control Store address increment) thể hiện trong hình 6.10. Trong các trường hợp COND có giá trị 001, 010, 011, 100 hay 101, lệnh nhảy có điều kiện sẽ trỏ tới vùng nhớ control store tương ứng được lưu trong trường JUMP ADDR tùy thuộc vào giá trị của các bit n, z, v, hay c hoặc bit thứ 13 trong thanh ghi `%ir`. Nếu COND = 110, lệnh nhảy không điều kiện sẽ được thực hiện. Trường COND là 111 chỉ ra rằng tại thời điểm đó đang diễn ra quá trình giải mã lệnh. Khi đó, lệnh kế tiếp

trong control store được nạp vào MIR được lấy từ khối Next trong CS Address MUX hoặc trong khối JUMP.

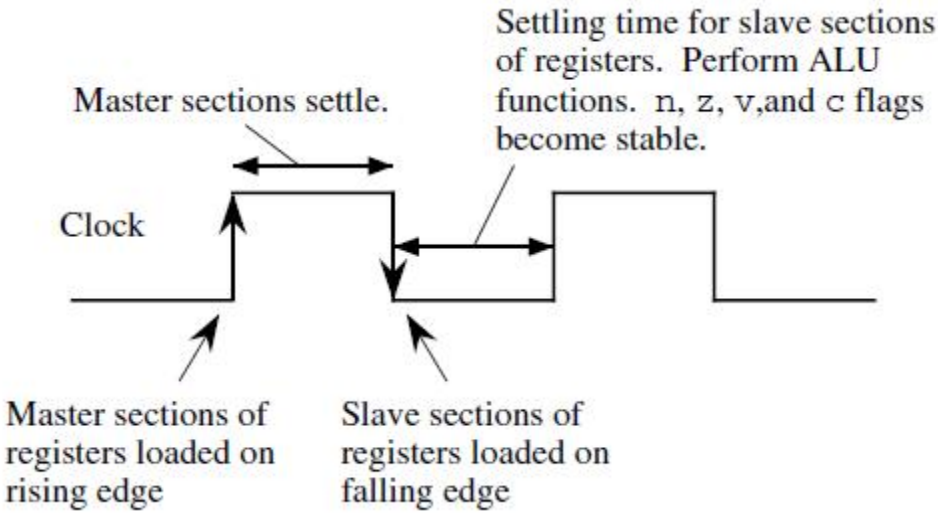
C_2 C_1 C_0	Operation
0 0 0	Use NEXT ADDR
0 0 1	Use JUMP ADDR if n = 1
0 1 0	Use JUMP ADDR if z = 1
0 1 1	Use JUMP ADDR if v = 1
1 0 0	Use JUMP ADDR if c = 1
1 0 1	Use JUMP ADDR if IR[13] = 1
1 1 0	Use JUMP ADDR
1 1 1	DECODE

Hình 6.12. Cấu hình trường COND trong vi lệnh

Cuối cùng, trường JUMP ADDR chứa 11 bit ở tận cùng bên phải của thanh ghi MIR sẽ chứa vị trí của vi lệnh chứa trong vùng nhớ control store giúp hệ thống có thể nhảy đến vị trí bất kỳ trong vùng nhớ đó.

6.2.3. Đồng bộ hoạt động

Kiến trúc vi mẫu hoạt động dưới tác động đồng bộ của 1 bộ phát xung nhịp. Các khối master của tất cả các thanh ghi thay đổi trạng thái vào thời điểm sườn lên của xung và các khối slave thay đổi trạng thái tại thời điểm sườn xuống của xung. Điều này được thể hiện như trên hình 6.14.



Hình 6.14. Đồng bộ tín hiệu với xung clock

Tất cả các thanh ghi đều cấu tạo từ flip-flop loại D ngoại trừ thanh ghi `%r0` không cấu tạo từ flip-flop nào cả. Tại sườn xuống của xung nhịp đồng hồ, dữ liệu lưu trữ trong các khối master của các thanh ghi sẽ được gửi tới các khối slave. Điều này làm cho các dữ liệu cần thiết cho hoạt động của ALU luôn luôn ở trạng thái sẵn sàng. Khi xung nhịp ở mức thấp, các khối ALU, CBL và MUX hoạt động và đến đúng thời điểm sườn lên của xung, dữ liệu trong đó đã ổn định đồng thời các dữ liệu mới trong các thanh ghi sẽ được lưu vào các khối master. Các thanh ghi sẽ dữ nguyên giá trị trong khi xung nhịp đồng hồ ở mức cao. Quá trình đó sẽ được lặp đi lặp lại

6.2.4. Bẫy và ngắt

Bẫy trap là một thủ tục tự động được gọi tới bởi một tín hiệu điện khi xảy ra một điều kiện nào đó khi ALU thực thi chương trình, ví dụ như lệnh không hợp lệ, tràn dữ liệu, hay xảy ra phép chia cho số không. Khi có yêu cầu thực thiện bẫy, quyền điều khiển được chuyển giao cho "bẫy" thực chất là một chương trình có sẵn của hệ điều hành. "Bẫy" sẽ thực hiện một chương trình nhất định nào đó tùy thuộc vào từng hệ điều hành, ví dụ như hiển thị lỗi đã xảy ra,...

Một phương pháp thực hiện bẫy là điều chỉnh vi lệnh.Ví dụ, ta có thể kiểm tra bit v để xem hệ thống có bị tràn hay không. Nếu xảy ra điều kiện bẫy, một đoạn vi lệnh sẽ được sử dụng để nạp vào PC với mục đích là xác định địa chỉ bắt đầu của bẫy.

Thông thường, người ta thường sử dụng một vùng nhớ để lưu địa chỉ của các bẫy. Vùng nhớ này có tên **branch table** được sử dụng để chuyển quyền điều khiển cho các bẫy (Hình 6.18.).

Address	Contents	Trap Handler
	⋮	
60	JUMP TO 2000	Illegal instruction
64	JUMP TO 3000	Overflow
68	JUMP TO 3600	Underflow
72	JUMP TO 5224	Zerodivide
76	JUMP TO 4180	Disk
80	JUMP TO 5364	Printer
84	JUMP TO 5908	TTY
88	JUMP TO 6048	Timer
	⋮	

Hình 6.18. Bảng ngắt branch table

Ngắt cũng tương tự như bẫy, nhưng ngắt sẽ bắt đầu khi có một tác động phần cứng như ấn một phím trên bàn phím, một biến động của nguồn cung cấp, hay một tác động của nhiệt độ lên hệ điều hành,v.v...Bẫy đồng bộ với chương trình chạy còn ngắt thì không. Do đó, bẫy sẽ được thực hiện tại cùng một vị trí trong cùng một chương trình với cùng một tập dữ liệu, còn ngắt sẽ xảy ra mà không thể dự đoán trước được

Để xem xét kỹ hơn về sự hoạt động của ngắt, ta sẽ xem xét một ví dụ hoạt động của nó sau đây. Khi tác động một phím trên bàn phím, bàn phím sẽ chuyển tín hiệu ngắt trên đường tín hiệu INT về CPU, ngay sau đó, CPU xác nhận đã nhận được tín hiệu ngắt. Bàn phím tiếp sau đó sẽ đặt vector ngắt lên đường bus dữ liệu truyền tiếp về CPU, đây chính là các dữ liệu mà chương trình ngắt sẽ sử dụng để thực hiện ngắt. Tiếp đó, CPU sẽ lưu bộ đếm chương trình và thanh ghi trạng thái vi xử lý vào stack. Vector ngắt sẽ được sử dụng để truy cập branch table để xác định địa chỉ của chương trình ngắt tương ứng để thực thi.

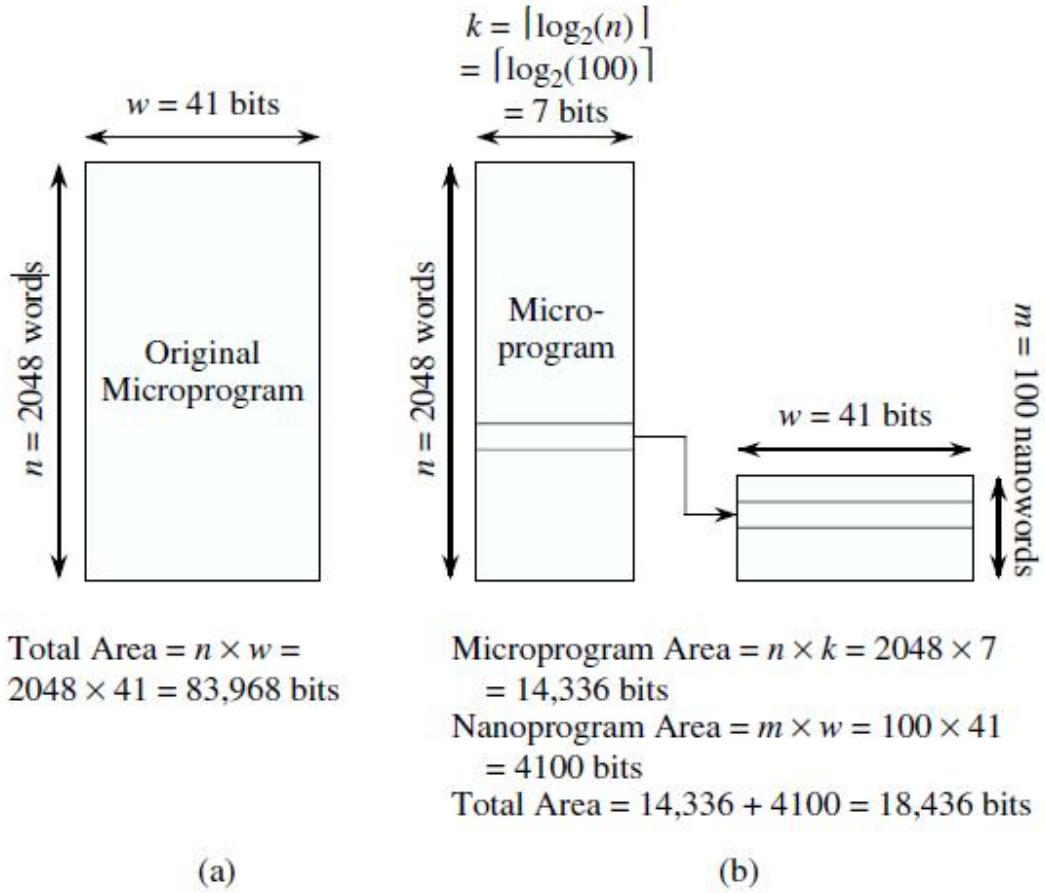
Khi bẫy hoặc ngắt được thực hiện, hệ thống sẽ tự động lưu các thanh ghi có khả năng bị thay đổi giá trị vào stack, sau đó sẽ khôi phục lại khi bẫy hoặc ngắt được thực hiện xong. Quá trình khôi phục và chuyển quyền điều khiển về chương trình chính sau khi thực hiện xong bẫy hoặc ngắt khác đối với chương trình con. Điểm khác biệt là đối với ngắt hoặc bẫy, thanh ghi `%psr` được lưu và khôi phục lại còn đối với chương trình con thì không

6.2.5. Lập trình nano

Trong hình 6.10 ở trên, chúng ta đã nghiên cứu về cấu trúc của CPU, trong đó ta thấy có một vùng nhớ control store kích thước 2048 x 41 bit. Đây là vùng nhớ chứa các vi lệnh. Các vi lệnh này sẽ được nạp vào thanh ghi điều khiển MIR để điều khiển các luồng dữ liệu trong hệ thống. Vấn đề đặt ra là ta có thể thiết kế vùng nhớ này như thế nào

Thông thường, vùng nhớ control store sẽ lưu trữ 2048 vi lệnh khác nhau, mỗi vi lệnh có kích thước 41 bit. Tuy nhiên, nếu trong trường hợp hệ thống có nhiều vi lệnh giống nhau, chúng ta có thể tiết kiệm kích thước vùng nhớ này bằng cách là lưu trữ một vi lệnh duy nhất được gọi là nanostore và sẽ đánh chỉ số để truy cập vào các nanostore này.

Hình 6.19a chỉ ra kích thước bộ nhớ trong trường hợp cơ bản là 2048 x 41 bit. Khi đó dung lượng cần thiết của bộ nhớ là 2048 x 41 = 83968 bit



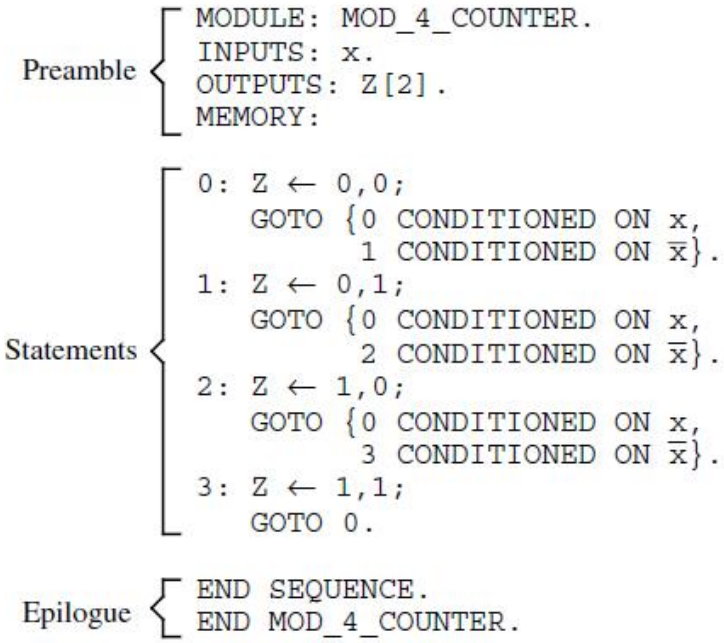
Hình 6.19. (a) Vùng nhớ control store (b) Lập trình nano

Bây giờ hãy giả sử là hệ thống của ta không sử dụng 2048 vi lệnh khác nhau mà chỉ sử dụng khoảng 100 vi lệnh. Thông thường, 100 vi lệnh này vẫn nằm trong vùng nhớ 2048 x 41 bit, trong đó có nhiều hàng có cùng một nội dung. Với mục đích sắp xếp lại vùng nhớ đó để giảm thiểu dung lượng bộ nhớ cần sử dụng (nói cách khác là lập trình lại vùng nhớ - lập trình nano), ta có thể thực hiện như sau. Vùng nhớ thứ nhất cần dùng là vùng nhớ có tên nanoprogram được sử dụng để chứa 100 vi lệnh khác nhau. Vùng nhớ thứ 2 là vùng nhớ microgram có 2048 hàng tương ứng với số hàng mặc định trong ARC, số cột trong mỗi hàng có số bit tương ứng với số vi lệnh, ở đây là 100. Do đó ta cần sử dụng ở đây là k = $\log_2(100)$ = 7. Vùng nhớ microgram này chứa 2048 giá trị mà mỗi giá trị mã hóa tương ứng 1 lệnh trong vùng nhớ nanoprogram. Lúc đó tổng vùng nhớ mà ta cần sử dụng là 18436 bit

6.3. Bộ điều khiển cứng

Một phương pháp khác để có thể điều khiển khối control unit là sử dụng phần cứng. Khi đó, ta sẽ sử dụng các mạch được tạo ra từ các cổng logic là các flip-flop để thay thế cho khối control store

Việc thiết kế mạch phần cứng để thực hiện nhiệm vụ điều khiển khối control unit rất phức tạp. Để quản lý công việc thiết kế này, người ta thường sử dụng ngôn ngữ mô tả phần cứng HDL – hardware description language. Một sự phát triển của ngôn ngữ này là VHDL, là viết tắt của VHSIC Hardware Description Language – Very High Speed Integrated Circuit.

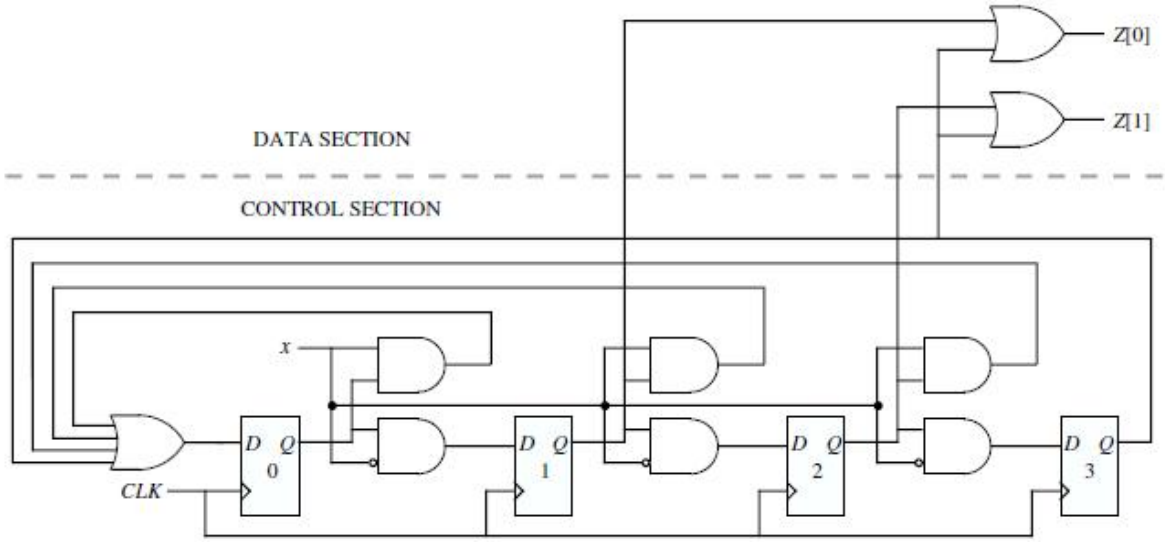


Hình 6.20. Ví dụ về ngôn ngữ HDL

Hình 6.20. thể hiện mô tả HDL của bộ đếm modul 4. Tín hiệu ra có 4 trạng thái 00, 01, 10, 11 và sau đó lặp lại cho đến khi đầu vào x là 0. Nếu đầu vào được set lên 1, bộ đếm sẽ quay lại trạng thái 0 ở chu kỳ xung nhịp kế tiếp

Một chương trình viết bằng HDL bao gồm 3 khối: Preamble, statements và epilogue. Khối Preamble sẽ đặt tên toàn bộ modul bởi từ khóa “MODULE”, định nghĩa các biến đầu vào, đầu ra bởi “INPUTS”, “OUTPUTS”, số lượng các đầu ra và tất nhiên nó cũng định nghĩa thêm dung lượng bộ nhớ mở rộng nếu cần bởi từ khóa “MEMORY”. Các trạng thái của mạch được thể hiện ngay sau đó trong phần Statements ngay sau Preamble. Khối Epilogue sẽ kết thúc modul bởi cụm từ khóa “END SEQUENCE”. Cụm từ khóa “END MOD_4_COUNTER” được sử dụng để đóng toàn bộ modul.

Việc chuyển ngôn ngữ HDL thành mạch điện tử có thể được phân tích thành 2 phần riêng biệt là khối điều khiển và khối dữ liệu. Khối điều khiển đối phó với việc chuyển trạng thái giữa các trạng thái của mạch. Khối dữ liệu quản lý các dữ liệu ra và thay đổi các giá trị đầu ra khi các giá trị đầu vào thay đổi



Hình 6.21. Mạch đếm modul 4

Trước hết ta sẽ xem xét khối điều khiển. Chúng ta có 4 trạng thái đầu ra nên sẽ sử dụng 4 flip-flop, mỗi chiếc tương ứng một trạng thái. Đầu tiên, ta sẽ vẽ 4 flip-flop và đánh số tương ứng và cung cấp cho chúng các tín hiệu xung nhịp CLK. Bước tiếp theo đơn giản là quét tất cả các trạng thái để thêm các cổng logic thích hợp cho việc chuyển trạng thái. Từ trạng thái 0, ta có 2 trạng thái mà nó có thể chuyển sang là trạng thái 0 và trạng thái 1 dưới điều kiện đầu vào x. Đầu ra của flip-flop 0 do đó sẽ được nối với đầu vào của các flip-flop số 0 và số 1. Cổng AND được sử dụng để thêm các điều kiện x vào giá trị đầu ra. Chú ý rằng cổng AND nối với flip-flop 1 có 1 vòng tròn nhỏ ở đầu vào, điều này có nghĩa là tín hiệu x được nghịch đảo trước khi được đưa vào cổng AND. Các trạng thái của các flip-flop 1 và 2 cũng hoàn toàn tương tự như đối với flip-flop 0. Với flip-flop 3 thì không cần làm vậy vì với mọi giá trị của x thì trạng thái đều được chuyển về 0.

Tiếp theo là khối dữ liệu, rất đơn giản trong trường hợp này. Các bit đầu ra chuyển trạng thái trong mỗi trạng thái của nó, do đó không cần thiết phải xác định điều kiện khi tạo các tín hiện ra. Chúng ta chỉ cần xác định đúng các giá trị của đầu ra là xong. Bit có trọng số nhỏ nhất bằng 1 ở các trạng thái 1 và 3, nên đầu ra của nó được lấy ra từ flip-flop 1 và 3 qua một cổng OR. Bit có trọng số

lớn nhất bằng 1 ở trạng thái 2 và 3, do đó cũng tương tự đầu ra này được lấy từ flip-flop 2 và 3 thông qua 1 cổng OR. Kết quả cuối cùng ta có được mạch như hình 6.21.

Bây giờ ta quay trở lại khối điều khiển của kiến trúc ARC được thể hiện trong hình 6.10. Nhắc lại một chút, ta có thể thiết kế khối điều khiển này bằng 2 cách tiếp cận. Các tiếp cận thứ nhất ta đã đề cập trong mục 6.2, đó là cách tiếp cận bằng phần mềm. Trong phần này, ta sẽ xem xét cách tiếp cận bằng mạch điều khiển phần cứng

Với mỗi lệnh được thực thi, các bước mà CPU phải thực hiện bao gồm

1. Nạp lệnh kế tiếp sẽ được thực thi từ bộ nhớ
2. Giải mã lệnh
3. Đọc các toán hạng từ bộ nhớ hoặc thanh ghi nếu cần
4. Thực thi lệnh và lưu kết quả
5. Quay về bước 1

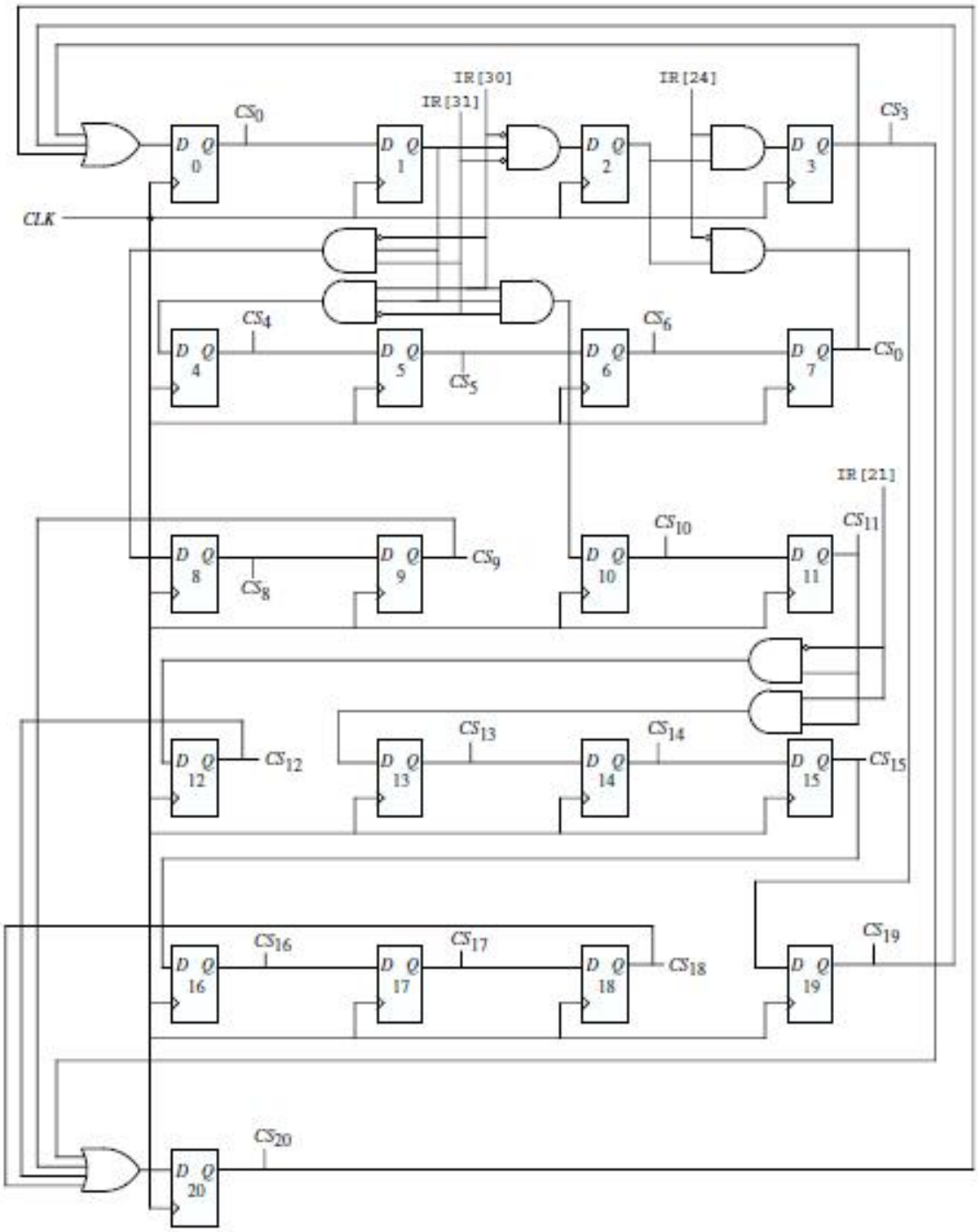
Khối điều khiển của ARC được mô tả bởi ngôn ngữ HDL được thể hiện trong hình 6.22

Hình 6.23 thể hiện khối điều khiển mô tả khối điều khiển của ARC

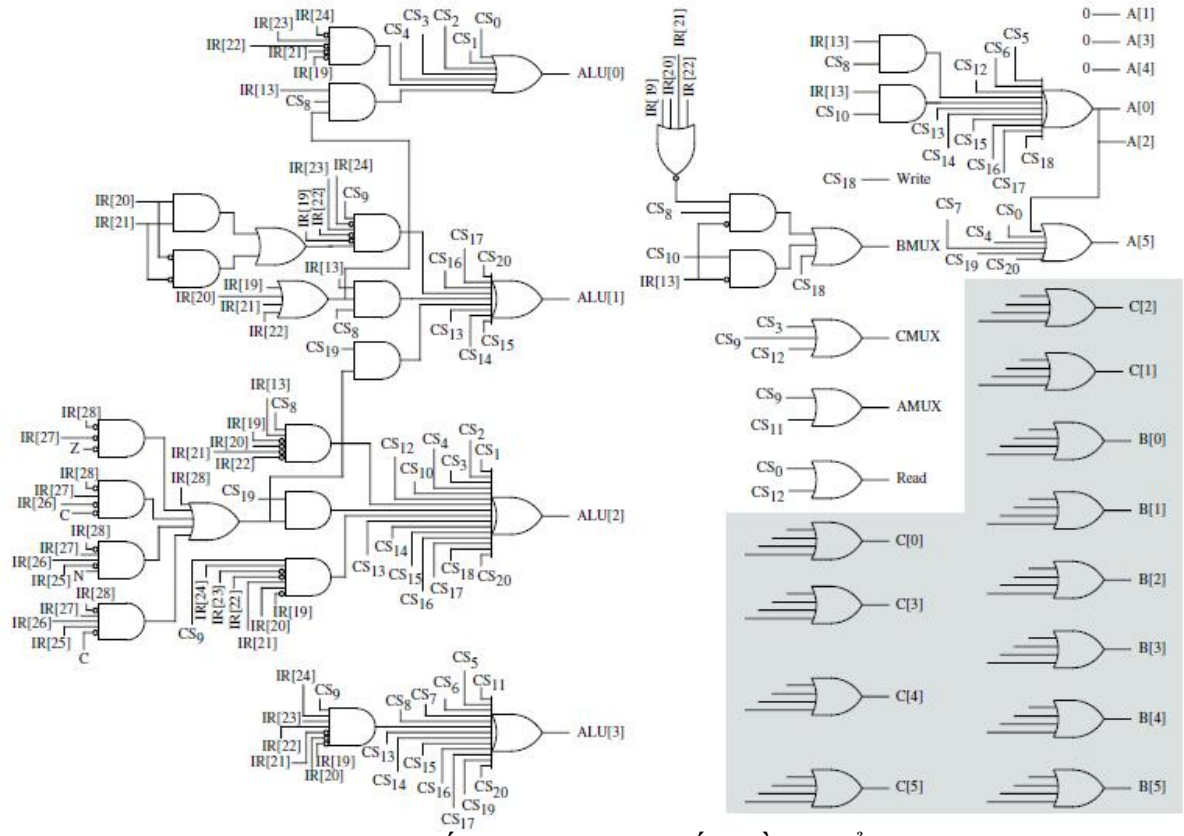
Hình 6.24. thể hiện khối dữ liệu đầu ra của khối điều khiển ARC

```
MODULE: ARC_CONTROL_UNIT.
INPUTS:
OUTPUTS: C, N, V, Z.  ! These are set by the ALU
MEMORY: R[16][32], pc[32], ir[32], temp0[32], temp1[32], temp2[32],
        temp3[32].

0: ir ← AND(pc, pc); Read ← 1;          ! Instruction fetch
    ! Decode op field
1: GOTO {2 CONDITIONED ON ir[31]×ir[30],   ! Branch/sethi format: op=00
         4 CONDITIONED ON ir[31]×ir[30],   ! Call format: op=01
         8 CONDITIONED ON ir[31]×ir[30],   ! Arithmetic format: op=10
        10 CONDITIONED ON ir[31]×ir[30]}. ! Memory format: op=11
    ! Decode op2 field
2: GOTO 19 CONDITIONED ON ir[24].          ! Goto 19 if Branch format
3: R[rd] ← ir[imm22];                      ! sethi
   GOTO 20.
4: R[15] ← AND(pc, pc).                    ! call: save pc in register 15
5: temp0 ← ADD(ir, ir).                    ! Shift disp30 field left
6: temp0 ← ADD(ir, ir).                    ! Shift again
7: pc ← ADD(pc, temp0); GOTO 0.            ! Jump to subroutine
    ! Get second source operand into temp0 for Arithmetic format
8: temp0  ← { SEXT13(ir) CONDITIONED ON ir[13]×NOR(ir[19:22]),    ! addcc
  R[rs2] CONDITIONED ON ir[13]×NOR(ir[19:22]),                    ! addcc
  SIMM13(ir) CONDITIONED ON ir[13]×OR(ir[19:22]),   ! Remaining
  R[rs2] CONDITIONED ON ir[13]×OR(ir[19:22])}. ! Arithmetic instructions
    ! Decode op3 field for Arithmetic format
9: R[rd] ← {
    ADDCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010000),  ! addcc
    ANDCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010001),  ! andcc
    ORCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010010),   ! orcc
    NORCC(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 010110),  ! orncc
    SRL(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 100110),    ! srl
    ADD(R[rs1], temp0) CONDITIONED ON XNOR(IR[19:24], 111000)};   ! jmpl
    GOTO 20.
    ! Get second source operand into temp0 for Memory format
10: temp0  ← {SEXT13(ir) CONDITIONED ON ir[13],
              R[rs2] CONDITIONED ON ir[13]}.
11: temp0  ← ADD(R[rs1], temp0).
    ! Decode op3 field for Memory format
    GOTO {12 CONDITIONED ON ir[21],                                ! ld
          13 CONDITIONED ON ir[21]}.                               ! st
12: R[rd] ← AND(temp0, temp0); Read ← 1; GOTO 20.
13: ir  ← RSHIFT5(ir).

14: ir  ← RSHIFT5(ir).
15: ir  ← RSHIFT5(ir).
16: ir  ← RSHIFT5(ir).
17: ir  ← RSHIFT5(ir).
18: r0  ← AND(temp0, R[rs2]); Write  ← 1; GOTO 20.
19: pc  ← {  ! Branch instructions
    ADD(pc, temp0) CONDITIONED ON ir[28] + ir[28]×ir[27]×Z +
        ir[28]×ir[27]×ir[26]×C + ir[28]×ir[27]×ir[26]×ir[25]×N +
        ir[28]×ir[27]×ir[26]×ir[25]×V,
    INCPC(pc) CONDITIONED ON ir[28]×ir[27]×Z +
        ir[28]×ir[27]×ir[26]×C + ir[28]×ir[27]×ir[26]×ir[25]×N +
        ir[28]×ir[27]×ir[26]×ir[25]×V};
    GOTO 0.
20: pc ← INCPC(pc); GOTO 0.
END SEQUENCE.
END ARC_CONTROL_UNIT.
```

Hình 6.22. Khối điều khiển của ARC dưới ngôn ngữ HDL



Hình 6.23. Khối control section của khối điều khiển ARC



Hình 6.24. Khối dữ liệu của khối điều khiển ARC

TỔNG KẾT CHƯƠNG

Vi kiến trúc máy tính bao gồm các luồng dữ liệu và khối điều khiển. Luồng dữ liệu bao gồm các thanh ghi, khối ALU và các kết nối giữa chúng. Khối điều khiển bao gồm thanh ghi điều khiển MIR (với cách tiếp cận bằng phần mềm) hoặc các mã điều kiện trạng thái. Tương ứng như vậy, ta có thể thiết kế bộ điều khiển bằng vi chương trình hoặc bằng phần cứng. Bộ điều khiển vi chương trình bao gồm các vi lệnh được lưu trữ trong khối bộ nhớ control store. Bộ điều khiển bằng phần cứng bao gồm tổ hợp các flip-flop mang thông tin các trạng thái, kết hợp với các cổng logic thể hiện sự chuyển trạng thái của bộ điều khiển

Cách tiếp cận bằng phần cứng làm hệ thống hoạt động với tốc độ cao và sử dụng số lượng phần cứng nhỏ hơn so với cách tiếp cận bằng phần mềm. Tuy nhiên cách tiếp cận bằng phần mềm tỏ ra linh hoạt hơn, quá trình hoạt động đơn giản hơn nên phương pháp này được sử dụng phổ biến hơn. Hỗ trợ cho cách tiếp cận phần mềm là việc thiết kế lập trình nano. Đây là thủ thuật để làm giảm thiểu bộ nhớ control store

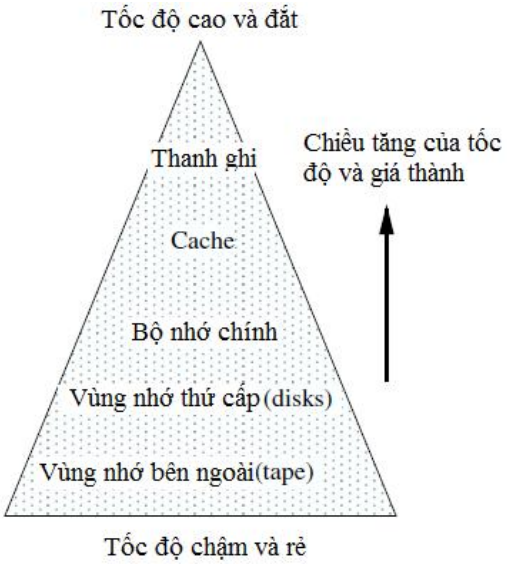
CHƯƠNG 7: BỘ NHỚ

Trong vài thập kỷ trước đây, tốc độ xử lý của CPU được tính bằng số lệnh được xử lý trong 1 giây, thông thường tăng gấp đôi trong 18 tháng. Bộ nhớ máy tính cũng tương tự như vậy, dung lượng của nó cũng tăng gấp 4 mỗi 36 tháng. Tuy nhiên, tốc độ hoạt động của bộ nhớ chỉ tăng lên với tỉ lệ dưới 10% mỗi năm. Do đó, trong khi tốc độ xử lý của CPU tăng tương ứng với dung lượng bộ nhớ, còn khoảng cách giữa tốc độ xử lý của CPU và tốc độ hoạt động của bộ nhớ ngày càng lớn.

Khi khoảng cách giữa tốc độ xử lý của CPU và tốc độ bộ nhớ tăng lên, các giải pháp kỹ thuật sẽ được sử dụng để lấp đầy các khoảng cách này. Một máy tính điển hình chứa một vài loại bộ nhớ khác nhau, từ các loại hoạt động nhanh, đắt tiền như các thanh ghi nội ở bên trong CPU, đến các loại bộ nhớ rẻ tiền, không đắt như các loại ổ nhớ gắn ngoài. Sự tương tác giữa các loại bộ nhớ này làm cho máy tính hoạt động như thể chúng sử dụng 1 hệ thống nhớ đơn giản, dung lượng lớn và tốc độ nhanh; mặc dù thực tế chúng chứa rất nhiều dạng bộ nhớ khác nhau. Chúng ta sẽ bắt đầu chương này với nội dung các tổ chức bộ nhớ trong máy tính

7.1. Phân cấp bộ nhớ

Bộ nhớ trong một máy tính số thông thường được tổ chức theo phân cấp được thể hiện như hình 7.1. Ở phía triên của hệ thống phân cấp là các thanh ghi có tốc



Hình 7.1. Phân cấp bộ nhớ

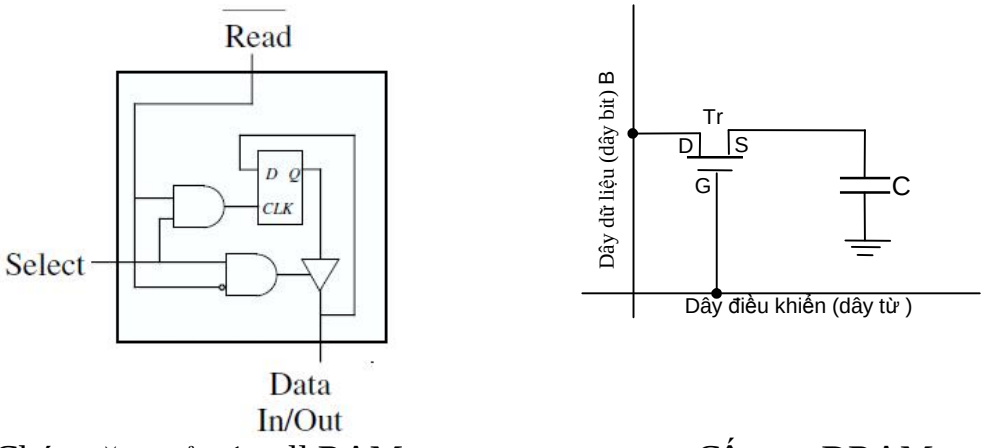
độ tương ứng với tốc độ của CPU nhưng có xu hướng là kích thước lớn và tiêu thụ điện năng lớn. Do vậy mà thông thường chỉ có một số lượng nhỏ các thanh ghi trong vi xử lý, trong một vài trường hợp có thể lên tới vài trăm thanh ghi hoặc ít hơn. Ở phía dưới hệ thống phân cấp là bộ nhớ thứ cấp và các bộ nhớ gắn ngoài như là các đĩa từ hoặc băng từ có giá thành rẻ, tiêu thụ ít điện năng nhưng thời gian truy cập lớn hơn rất nhiều so với thanh ghi. Khoảng giữa hệ thống phân cấp là một số dạng khác nhau của bộ nhớ. Nói thêm một chút về hệ thống phân cấp này, dạng bộ nhớ có tốc độ truy xuất càng cao thì có giá thành cũng càng cao. Bảng 7.1. có ta một số đặc điểm về bộ nhớ máy tính được thống kê vào những năm 90s

Loại bộ nhớ	Thời gian truy cập	Giá thành/MB	Dung lượng thường sử dụng	Giá thành
Registers	1ns	High	1KB	–
Cache	5-20 ns	$100	1MB	$100
Main memory	60-80ns	$1.10	64 MB	$70
Disk memory	10 ms	$0.05	4 GB	$200

Bảng 7.1. Một số đặc điểm về bộ nhớ máy tính

7.2. Bộ nhớ RAM – Random Access Memory

Trong mục này, chúng ta sẽ tìm hiểu cấu trúc và chức năng của bộ nhớ RAM. Thuật ngữ “random” ở đây có nghĩa là bất cứ một vùng nhớ nào cũng có thể được truy cập trong cùng khoảng thời gian. Hình 7.2 thể hiện chức năng của một cell RAM trong một máy tính điển hình và cấu tạo thực tế của một DRAM. Cấu tạo của một cell RAM bao gồm một D flip-flop cộng với phần điều khiển cho phép cell nhớ được chọn selected, đọc read, ghi writen. Cell RAM có một cổng 2 chiều được sử dụng làm đường dây input/output. Ta sẽ dựa vào cấu tạo này để thảo luận về hoạt động của bộ nhớ RAM. Chú ý rằng hình chức năng 1 cell ram chỉ là mô tả về chức năng chứ không phải là cấu tạo thực tế. Ta có rất nhiều cách khác nhau để thể hiện chức năng của RAM. Hình cấu tạo DRAM là một ví dụ.



Chức năng của 1 cell RAM Cấu tạo DRAM

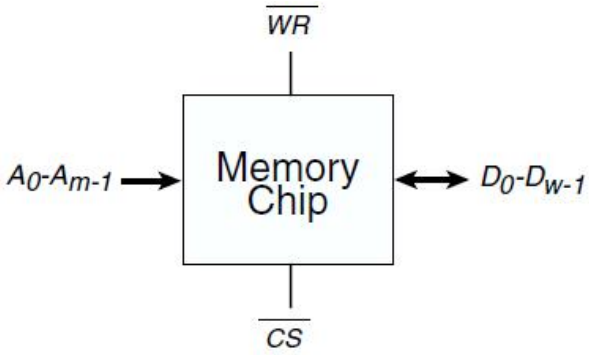
Hình 7.2. Cấu tạo của 1 cell RAM

Quá trình đọc RAM có tín hiệu điều khiển Select = 1 và $\overline{Read}$ = 0. Khi đó tín hiệu CLK đưa vào D flip-flop không tích cực, trạng thái của D Flip-flop không đổi, đồng thời tín hiệu điều khiển cổng Data in/out tích cực làm trạng thái hiện tại của D flip-flop đi qua cổng này. Quá trình ghi dữ liệu hoạt động tương tự. Khi đó Select = 1 còn $\overline{Read}$ = 1, tín hiệu cổng Data in/out không tích cực làm đầu ra của D flip-flop được cách ly, tín hiệu CLK tích cực làm D flip-flop lấy trạng thái đầu vào (lúc này là tín hiệu trên Data in/out) thành trạng thái của nó. Dữ liệu trên Data in/out được coi là được lưu trên D flip-flop.

Bộ nhớ có cấu tạo dựa trên các Flip-flop như hình 7.2 được gọi là RAM tĩnh (SRAM) bởi vì nội dung của mỗi ô nhớ sẽ kéo dài cho đến khi nào điện năng ngừng cung cấp cho ô nhớ đó. Ram động (Dynamic RAM - DRAM) có cấu tạo từ các tụ điện, có khả năng lưu được dữ liệu có mức logic 1 trong khoảng một vài phút khi ngừng cung cấp năng lượng. Do DRAM bị mất năng lượng do hiện tượng dò điện năng trên tụ nên nó cần được refresh một cách định kỳ. Cũng do hiện tượng dò điện năng mà hệ thống DRAM dễ phát sinh các tia gamma gây nhiễu lên hệ thống. Đây là hiện tượng khá hiếm khi xảy ra nhưng những hệ thống sử dụng DRAM cũng không thể hoạt động liên tục nhiều ngày. Thông thường ta nên tắt máy tính vào cuối ngày để tránh lỗi hệ thống. Một số máy tính có khả năng dò tìm lỗi hệ thống nhưng những hệ thống này có giá thành tương đối cao và được sử dụng trong các hệ thống ATM (automated teller machines) hay NFSs (network file servers)

Phần tiếp theo chúng ta sẽ tìm hiểu các cell RAM được tổ chức thành các chip nhớ như thế nào

7.3. Tổ chức bộ nhớ



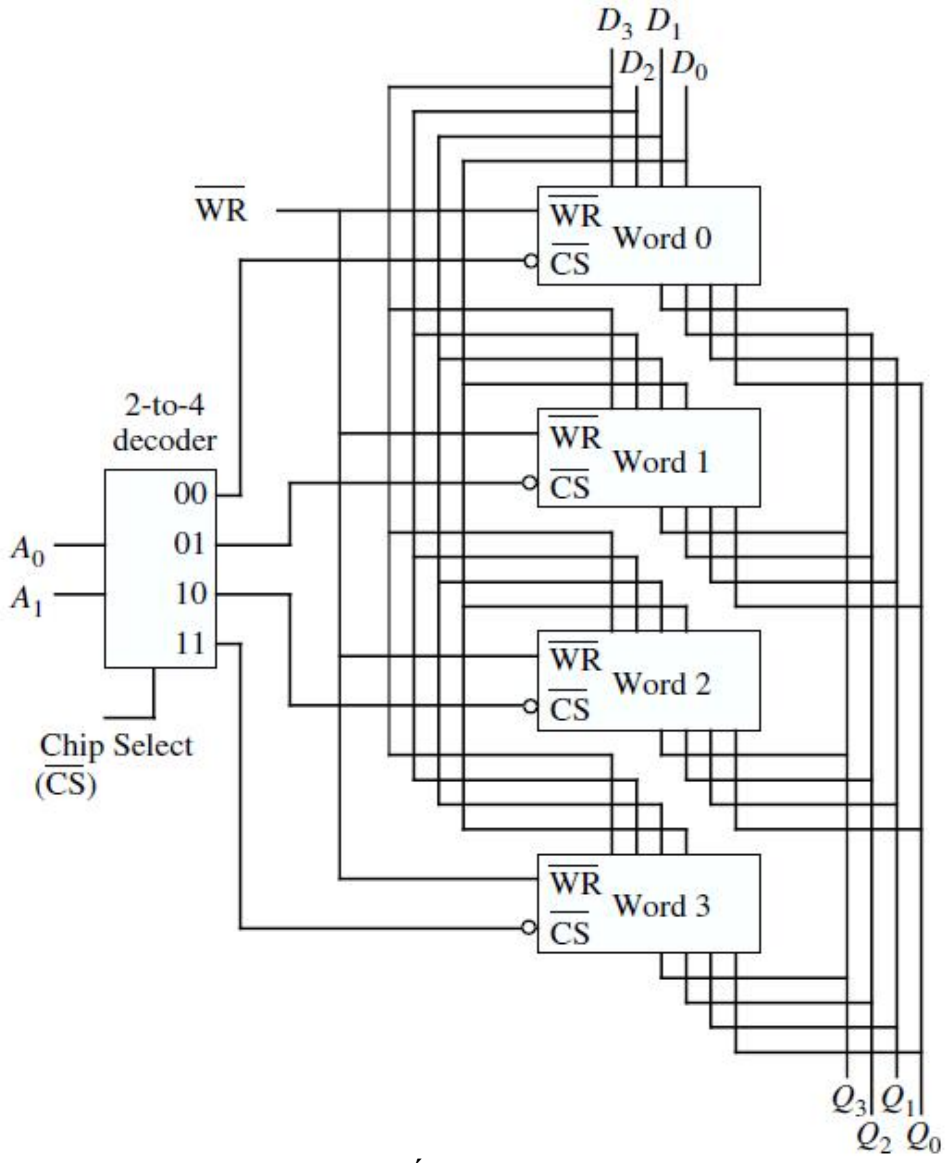
Hình 7.3. Sơ đồ khối chip RAM

Sơ đồ chân của chip Ram được mô tả một cách đơn giản như hình 7.3. Như trên hình, chip ram có m chân tín hiệu địa chỉ được đánh ký hiệu từ A_0...A_{m-1}, các tín hiệu điều khiển chọn chip $\overline{CS}$ (Chip Seclect), tín hiệu điều khiển $\overline{WR}$ ($\overline{WR}$ = 0 để ghi dữ liệu lên chip) hoặc WR(WR = 1 để đọc dữ liệu từ chip về CPU). Trong quá trình đọc dữ liệu từ chip, sau khoảng thời gian t_{AA} là khoảng thời gian trễ từ khi tín hiệu địa chỉ được giải mã và có tác dụng, tín hiệu dữ liệu w-bit mới tương tác được với các đường dây dữ liệu D_0...D_{w-1}. Trong quá trình ghi dữ liệu lên chip, đường dữ liệu cần được giữ nguyên giá trị trong suốt khoảng thời gian t_{AA}. Các đường dây dữ liệu đều là các các tín hiệu 2 chiều.

Tín hiệu địa chỉ A_0...A_{m-1} của bộ nhớ RAM thể hiện trên hình 7.3 sẽ chứa các tín hiệu địa chỉ được giải mã từ tín hiệu m-bit thành 2^m ô nhớ trong chip. Mỗi ô nhớ đó sẽ chứa w-bit. Do đó chip nhớ có dung lượng là 2^m x w bit

Vấn đề giải mã hệ thống RAM

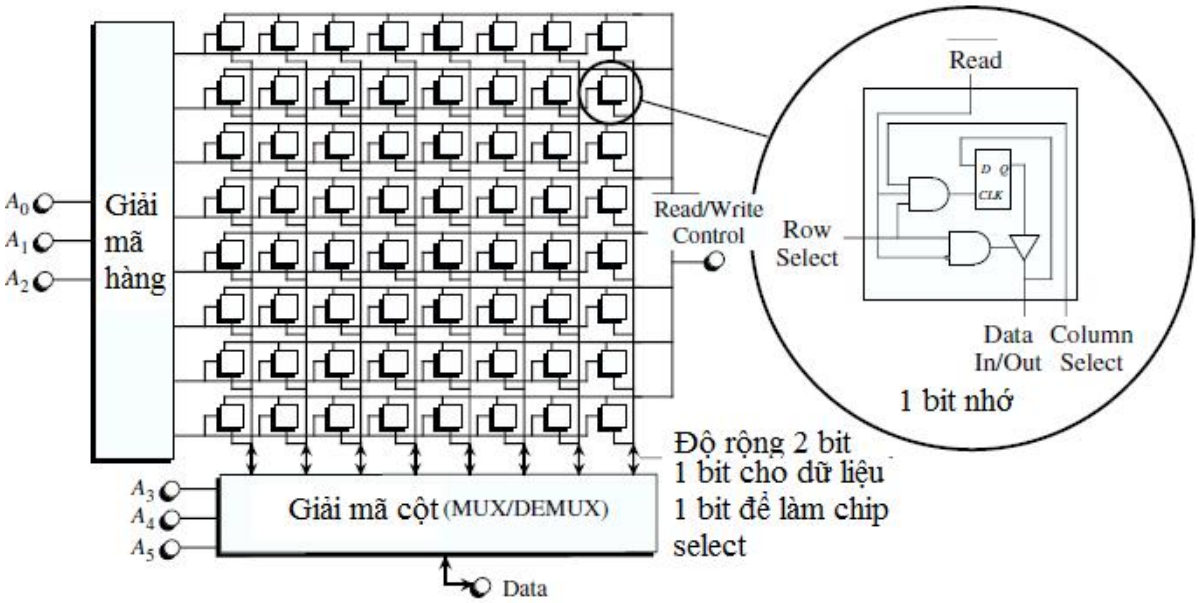
Trong mục 7.2, ta đã xem xét cấu trúc của một cell RAM. Vấn đề đặt ra là làm thế nào để có thể xây dựng hệ thống có dung lượng lớn hơn 1 cell ram. Ta hãy xem cấu trúc của một hệ thống có dung lượng 4 x 4 bit word. Hệ thống này có một bộ giải mã 2-to-4 decoder có nhiệm vụ giải mã 2 tín hiệu địa chỉ A_0 và A_1 thành 4 địa chỉ ô nhớ. Mỗi tín hiệu địa chỉ ô nhớ này được nối với một khối có 4 cell tương ứng với 4 bit thông qua các tín hiêu $\overline{CS}$. Các chân tín hiệu đầu ra của các khối này Q_0...Q_3 được nối với nhau và nối với đường dây data bus. Khi có một giá trị địa chỉ A_1A_0 đưa vào hệ thống, bộ giải mã sẽ kích hoạt một đường địa chỉ 00, 01, 10 hoặc 11. Một trong các đường dây này sẽ kích hoạt 1 khối bộ nhớ từ $Word_0$...$Word_3$ tương ứng. Các cell nhớ trong khối nhớ đó sẽ kết nối với data bus để thực hiện quá trình đọc hoặc ghi theo nhu cầu hoạt động của hệ



Hình 7.4. Hệ thống RAM 4 x 4 bit word

thống. Phương pháp giải mã như trên có tên gọi là giải mã tuyến tính hay là giải mã 1 bước. Ưu điểm của giải mã tuyến tính là thời gian truy cập bộ nhớ tương đối nhanh. Nhược điểm của cách giải mã này là tỏ ra không kinh tế với trường hợp số tín hiệu địa chỉ tăng cao. Ví dụ như chip bộ nhớ dung lượng 64M có 26 đường địa chỉ. Điều này có nghĩa là ta phải xây dựng hệ thống giải mã 26-to-2^{26} mà trong thực tế, chi phí để xây dựng bộ giải mã này sẽ chiếm phần lớn giá thành của bộ nhớ. Để giải quyết vấn đề này, ta xây dựng bộ nhớ theo phương pháp giải mã 2 bước như trên hình 7.5

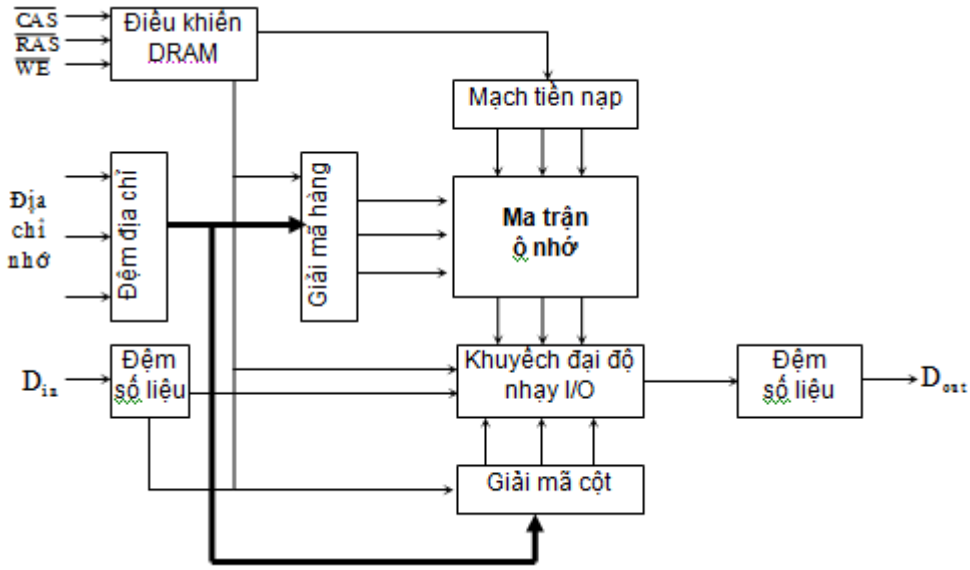
Hệ thống giải mã 2 bước được sử dụng phổ biến trong hầu hết các Ics nhớ. Hệ thống bao gồm các bộ giải mã theo hàng và theo cột hoạt động độc lập với nhau. Như trên hình 7.5. hệ thống RAM của ta có 6 đường địa chỉ được chia



Hình 7.5. Hệ thống RAM giải mã 2 bước

thành các tín hiệu hàng và cột. Bộ giải mã cột còn có chức năng làm bộ giải mã, lựa chọn tín hiệu địa chỉ hoặc dữ liệu. Một hệ thống tín hiệu 2 chiều được sử dụng để làm cổng input và output. Trong quá trình đọc, bộ giải mã hàng sẽ thực hiện chức năng của mình và kích hoạt toàn bộ một hàng của ma trận nhớ. Tiếp đó, tín hiệu giải mã cột sẽ lựa chọn cho một đầu ra duy nhất. Trong quá trình ghi dữ liệu, bit được ghi được phân phối bởi bộ giải mã DEMUX lên cột mục tiêu, đồng thời bộ giải mã hàng sẽ lựa chọn hàng thích hợp để viết

Một cải tiến của nguyên lý trên là hệ thống giải mã 2 bước kết hợp refresh dành cho DRAM được trình bày trong hình 7.6.



Hình 7.6. Bộ giải mã bộ nhớ 2 bước kết hợp refresh cho DRAM

Hoạt động của hệ thống như sau: Trong quá trình truy cập ô nhớ, đầu tên tín hiệu giải mã hàng $\overline{RAS}$ sẽ được kích hoạt tín hiệu của một hàng trong ma trận các ô nhớ, tiếp theo đó là kích hoạt tín hiệu giải mã cột $\overline{CAS}$. Trước khi xâm nhập ô nhớ, bộ tiền nạp sẽ cung cấp tín hiệu điện áp $Vcc/_2$ vào tất cả các đường

dây bit trên cùng một hàng. Khi có tín hiệu đọc ô nhớ, các đường dây bit này sẽ được kết nối với tụ điện trong ô nhớ. Tùy vào giá trị điện áp trong ô nhớ lúc đó là 0 (hay 1) mà điện áp trên tụ sẽ được kéo lên (hoặc kéo xuống) làm điện áp trên đường dây bit bị giảm (hay tăng) một lượng rất nhỏ. Bộ khuếch đại độ nhạy sẽ nhận biết sự thay đổi này và khuếch đại giá trị nạp lại cho tụ điện về mức logic 0 (hoặc 1) ở các điện áp chuẩn của nó tức là 0V (hoặc 5V). Việc này làm cho các giá trị logic trên cùng 1 hàng được làm mới – refresh một lần.

Mặc dù DRAM tỏ ra rất kinh tế, nhưng SRAM lại cung cấp một tốc độ hoạt động nhanh. Quá trình refresh, quá trình phát hiện lỗi và hoạt động với mức độ tiêu hao năng lượng thấp đã làm cho tốc độ truy cập của DRAM chêch lệch khoảng $1/_4$ so với tốc độ SRAM. Tốc độ truy xuất của cả SRAM và DRAM

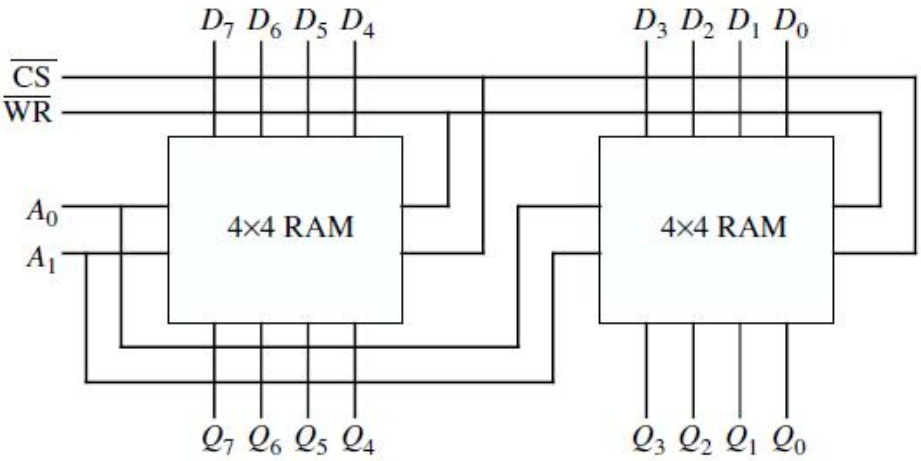
đều ngày càng được cải thiện. Một số phương pháp truy cập nhanh hay được sử dụng đó là phân trang, chế độ đan xen,...

Thiết kế hệ thống dung lượng lớn từ các chip dung lượng nhỏ

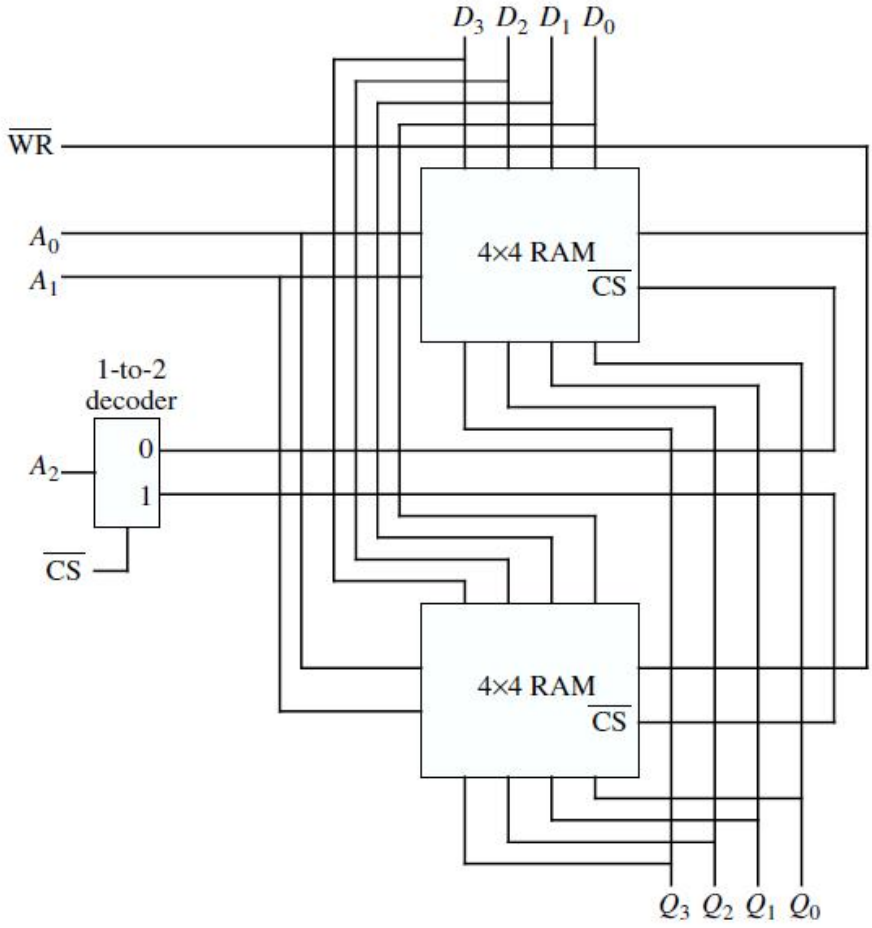
Chúng ta bây giờ có thể thiết kế một modul RAM có dung lượng lớn từ các modul dung lượng nhỏ. Ta có thể tăng dung lượng RAM theo cả 2 chiều, tức là tăng chiều dài từ nhớ và tăng số lượng từ nhớ. Ví dụ, ta có thể sử dụng 8 modul nhớ dung lượng 16M x 1bit để tạo thành một modul dung lượng 16M x 8-bit; hoặc là ta có thể sử dụng 2 modul dung lượng 32M x 8-bit để tạo thành một modul dung lượng 64M x 8-bit

Hình 7.7. cho ta nguyên tắc ghép nối bộ nhớ để tăng từ nhớ. Trên hình ta thấy từ 2 modul nhớ kích thước 4 x 4 bit, ta ghép nối để có được một hệ thống dung lượng 4 x 8 bit. Cách ghép nối này không làm thay đổi số lượng ô nhớ mà chỉ làm thay đổi kích thước của 1 địa chỉ nhớ. Nguyên tắc chung là:

- Hệ thống chung các tín hiệu điều khiển, chung các tín hiệu địa chỉ
- Các đường dây dữ liệu D và Q được đánh số lại tương ứng với modul mà ta sử dụng làm từ cao hoặc từ thấp



Hình 7.7. Ghép nối bộ nhớ tăng từ nhớ



Hình 7.8. Ghép nối bộ nhớ tăng số lượng từ nhớ

Tương tự như vậy, hình 7.8. cho ta nguyên tắc ghép nối bộ nhớ tăng số lượng từ nhớ. Các ghép nối này không làm thay đổi kích thước của 1 từ nhớ mà chỉ làm thay đổi số lượng địa chỉ mà modul truy xuất tới. Từ 2 modul nhớ 4 x 4 bit, ta có được modul 8 x 4 bit. Việc tăng số lượng từ nhớ từ 4 lên 8 làm cho số lượng

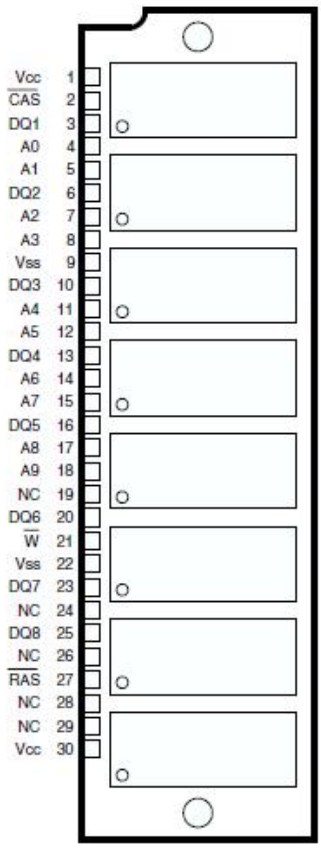
tín hiệu địa chỉ tăng từ 2 lên 3, tức là ta sẽ có 3 tín hiệu địa chỉ A_2, A_1 và A_0. Trong đó, 2 tín hiệu A_0 và A_1 được đưa trực tiếp vào 2 modul 4 x 4 bit. Tín hiệu A_2 sẽ được đưa vào một bộ giải mã để kích hoạt tín hiệu $\overline{CS}$ của 1 trong 2 modul 4 x 4 bit. Từ đó ta có thể có được nguyên tắc để ghép nối RAM trong trường hợp này là:

- Kết nối chung các đường dây dữ liệu D và Q của tất cả các modul nhớ
- Với mỗi modul nhớ, tín hiệu địa chỉ thấp được đưa tương ứng vào các đường dây địa chỉ của modul
- Các tín hiệu địa chỉ cao được đưa vào bộ giải mã tương ứng với số lương modul RAM. Đầu ra của bộ giải mã được đưa vào tín hiệu $\overline{CS}$ của mỗi modul RAM tương ứng

7.4. Modul nhớ thương mại

Modul nhớ thương mại thường được thiết kế theo một cấu trúc chuẩn. Hình 7.9. chỉ ra một cấu trúc chuẩn của chip nhớ dung lượng 2^{20} x 8 bit. Chuẩn này có tên Single-in-line Memory Module (SIMM). Các điểm kết nối điện (đánh số 1 - 30)

PIN NOMENCLATURE	
A0-A9	Address Inputs
$\overline{CAS}$	Column-Address Strobe
DQ1-DQ8	Data In/Data Out
NC	No Connection
$\overline{RAS}$	Row-Address Strobe
V_{cc}	5-V Supply
V_{ss}	Ground
$\overline{W}$	Write Enable



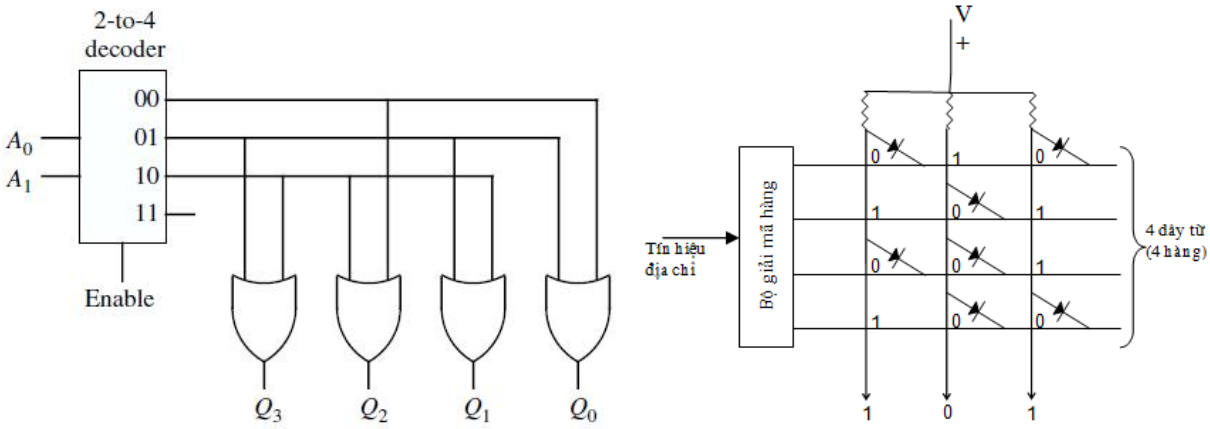
Hình 7.9. Modul nhớ SIMM

đều nằm trên một hàng đơn. Với 2^{20} địa chỉ ta cần 20 đường tín hiệu địa chỉ, nhưng cấu trúc này chỉ có 10 đường (A_0 – A_9). Các tín hiệu địa chỉ 10 bit cho các bộ giải mã hàng và giải mã cột được nạp vào modul nhớ hoàn toàn độc lập. Các tín hiệu điều khiển $\overline{RAS}$ và $\overline{CAS}$ sẽ tác động ngay khi địa chỉ 10 bit trên được đưa vào các bộ giải mã tương ứng. Mặc dù cách tổ chức này làm cho thời gian truy cập các ô nhớ riêng biệt tăng lên gấp đôi, nhưng nếu hoạt động trong chế độ phân trang thì thời gian truy xuất của nó lại giảm thiểu rất nhiều vì để truy cập các ô nhớ cạnh nhau thì chỉ cần thay đổi tín hiệu địa chỉ cột mà không cần nạp lại tín hiệu địa chỉ hàng.

Các đường tín hiệu địa chỉ dữ liệu DQ1 – DQ8 được tổ chức theo byte giúp cho việc đọc hay ghi dữ liệu được thực hiện song song. Để tổ chức một từ 32 bit, ta sẽ sử dụng 4 modul SIMM

7.5. Bộ nhớ ROM – Read Only Memory

Khi chương trình nạp vào bộ nhớ, nó sẽ tồn tại trong bộ nhớ cho đến khi bị ghi đè hoặc cho đến khi nguồn điện cung cấp cho hệ thống bị ngắt. Trong một số ứng dụng, chương trình không thay đổi trong suốt quá trình hoạt động, chương trình sẽ được ghi vào bộ nhớ chỉ đọc ROM (Read Only Memory). ROM thường được sử dụng để lưu chương trình trong các máy chơi game, máy tính, điện thoại,...



Location	Stored word
00	0101
01	1011
10	1110
11	0000

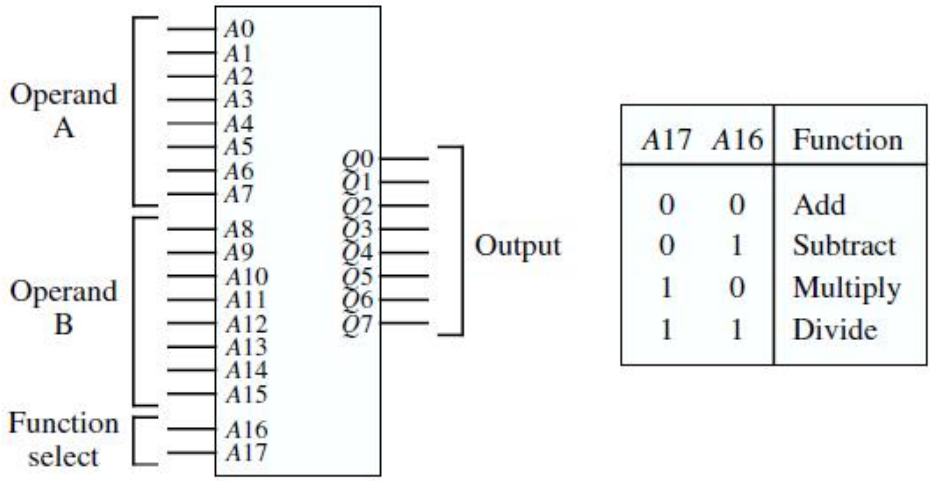
Hình 7.10. Cấu trúc của ROM

ROM là một chip được cấu tạo rất đơn giản. Tất cả cần có chỉ là một bộ giải mã, vài đường tín hiệu đầu ra và một vài cổng logic, không cần bất cứ một flip-flop hay một tụ điện nào cả. Hình 7.10 bên trái là một bộ nhớ ROM lưu trữ 4 từ độ rộng 4 bit (0101, 1011, 1110 và 0000). Mỗi một địa chỉ vào (00, 01, 10, 11) tương ứng với một địa chỉ lưu trữ. Hình 7.10 bên phải là một cách thể hiện nữa của ROM với cấu tạo từ các diode bán dẫn.

Để mở rộng ứng dụng của mình, ROM cần lập trình được. Nhu cầu này dẫn tới sự xuất hiện của PROM (Programmable ROM), cho phép người dùng ghi nội dung một lần bằng thiết bị có tên ROM bunner

PROM có nhược điểm là chỉ cho phép ghi nội dung 1 lần, do đó xuất hiện thêm ROM cho ghi dữ liệu vào phần chưa ghi, EPROM (Erasable PROM). Tiếp đó là sự xuất hiện của ROM cho phép xóa dữ liệu bằng tia cực tím UVPROM (Ultraviolet PROM) và ROM xóa dữ liệu bằng điện EEPROM (Electrically Erasable PROM)

Một ứng dụng khác của ROM được trình bày tiếp theo đây là bộ ALU cho phép toán đơn giản. Giả sử ta cần xây dựng một bộ ALU có thể thực hiện được 4 phép toán: cộng, trừ, nhân và chia với các toán hạng 8 bit. Ta sẽ tạo ra một bảng chân lý có tổng số trạng thái đầu ra là 2^{16} và 2^2 trạng thái điều khiển. Sau đó ta sẽ nạp bảng trạng thái vào ROM. Kết quả thể hiện trên hình 7.11.



A17	A16	Function
0	0	Add
0	1	Subtract
1	0	Multiply
1	1	Divide

Hình 7.11. Ứng dụng ROM để tạo thành một ALU đơn giản

Lúc đó, các tín hiệu địa chỉ được sử dụng để làm nơi nhập các toán hạng A và B và các tín hiệu điều khiển để thực hiện phép toán. Chú ý rằng các kết quả phép toán tương ứng với các tín hiệu điều khiển đã được nạp sẵn trên ROM

7.6. Bộ nhớ Cache

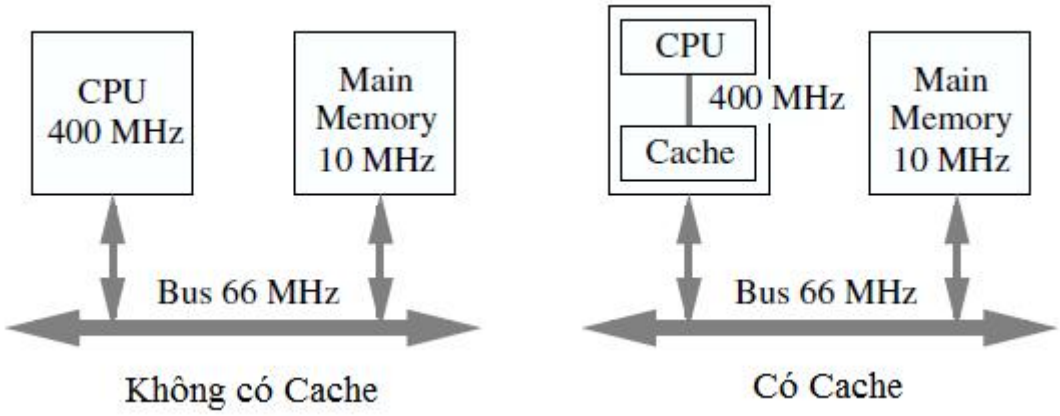
Khi chương trình được thực thi trên máy tính, hệ thống truy xuất tới một lượng nhỏ ô nhớ. Một ô nhớ vừa được truy xuất có khả năng sẽ tiếp tục được truy xuất ngay sau đó. Tốc độ truy xuất thông thường là thấp hơn rất nhiều so với tốc độ của bộ xử lý trung tâm CPU vì có một nút cổ chai ở phần kết nối giữa CPU và bộ nhớ. Do vậy, nếu hệ thống có một vùng nhớ dung lượng nhỏ nhưng tốc độ truy xuất với tốc độ cao, được sử dụng để lưu trữ nội dung ô nhớ vừa được truy xuất thì tốc độ xử lý của CPU sẽ được cải thiện rất nhiều. Đây chính là tiền đề của sự xuất hiện bộ nhớ Cache. Vùng nhớ Cache là vùng chứa nội dung của các ô nhớ thường được truy cập, được đặt ở giữa bộ nhớ chính và CPU. Khi thực thi chương trình, vùng nhớ Cache sẽ được tìm kiếm trước, nếu nội dung cần tìm có trong Cache, nó sẽ được sử dụng ngay lập tức. Nếu nội dung cần tìm không có trong Cache, hệ thống sẽ tiếp tục bằng cách truy xuất ra bộ nhớ chính bên ngoài. Nội dung này sau đó sẽ được đưa vào bộ nhớ Cache. Mặc dù quá trình này có vẻ làm cho thời gian truy cập bộ nhớ tăng lên nhưng nếu xét về tổng thể, việc sử dụng Cache đã làm giảm thời gian hoạt động của hệ thống rất nhiều.

Hệ thống bộ nhớ hiện đại có thể có vài mức Cache, đó là Cache Level 1 (L1), Level 2 (L2) và có thể có cả Level 3 (L3). Trong hầu hết các trường hợp, bộ nhớ L1 được tích hợp luôn vào CPU. Bộ nhớ Intel Pentium và IBM-Motorola PowerPC G3 đều có bộ nhớ L1 dung lượng 32Kbyte tích hợp sẵn trên chip CPU.

Bộ nhớ Cache có tốc độ truy xuất nhanh hơn bộ nhớ chính bởi một số nguyên nhân sau.

- Bộ nhớ Cache sử dụng các linh kiện điện tử có chất lượng cao, có khả năng hoạt động với tốc độ nhanh. Tất nhiên đi kèm với đặc tính này là giá thành của sản phẩm cũng như năng lượng tiêu tốn cho nó cũng tăng lên. Tuy nhiên, dung lượng của bộ nhớ Cache thường rất nhỏ nên cũng không ảnh hưởng lớn đến giá thành của sản phẩm
- Bộ nhớ Cache có dung lượng nhỏ nên bộ giải cũng đơn giản hơn, do đó giảm bớt thời gian truy cập
- Bộ nhớ Cache kết nối trực tiếp với CPU nên không phải giải quyết vấn đề tranh chấp bus

Hình 7.12 dưới đây chỉ rõ thêm về ý nghĩa của bộ nhớ Cache trong việc tăng tốc độ hoạt động của hệ thống. Tại hình phía bên trái, hệ thống không có sự tồn tại



Hình 7.12. Hệ thống máy tính không có Cache và có Cache

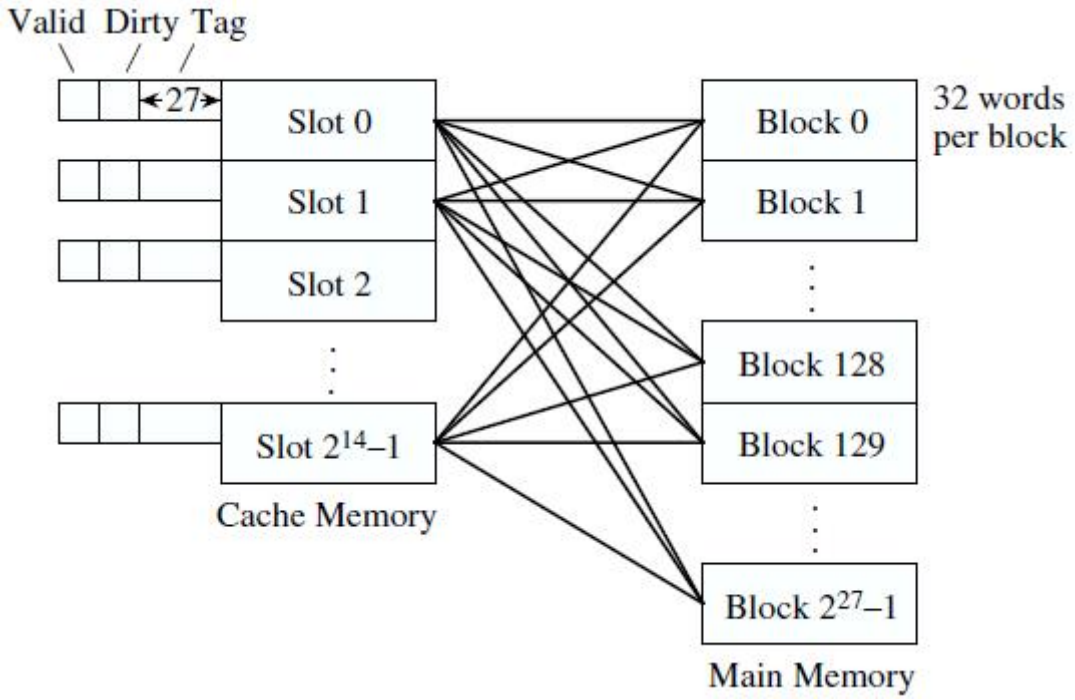
của bộ nhớ Cache. Hệ thống này có tốc độ hoạt động là 400MHz, nhưng truyền thông ở 66MHz và truy cập bộ nhớ ở tốc độ 10MHz. Một số chu kỳ bus được sử dụng để đồng bộ hoạt động của CPU và bus, do đó sự khác biệt giữa tốc độ của bộ nhớ và CPU có thể lên tới 10 lần hoặc hơn. Với hệ thống bên phải, bộ nhớ Cache được tích hợp vào CPU, do đó CPU có thể truy cập vào nó trực tiếp với tốc độ 400MHz.

7.6.1. Bộ nhớ Cache có kết hợp ánh xạ

Một số chương trình phần cứng được phát triển để chuyển tín hiệu địa chỉ truy xuất bộ nhớ chính thành địa chỉ truy xuất Cache. Điều này làm cho bộ nhớ Cache cải tiến có thể được đưa vào sử dụng tại bất cứ một máy tính nào mà không cần sử đổi phần mềm truy xuất chúng

Phương thức để chuyển đổi các tín hiệu địa chỉ ảnh hưởng rất lớn đến giá thành và tốc độ hoạt động của Cache, và không có một phương pháp nào được cho là tốt nhất. Trong phần này, chúng ta sẽ nghiên cứu phương pháp ánh xạ bộ nhớ. Phương pháp này được thực hiện theo nguyên tắc sau. Đầu tiên, giả sử hệ thống có bộ nhớ chính dung lượng 2^{32} được chia thành 2^{27} khối block, mỗi block có dung lượng là 32 word. Tất nhiên việc chia bộ nhớ chính này chỉ là về mặt logic. Thực tế, bộ nhớ chính vẫn là một khối thống nhất. Với bộ nhớ Cache giả sử trong trường hợp này sẽ bao gồm 2^{14} slot. Mỗi slot có dung lượng đủ lớn để lưu trữ 1 block bộ nhớ chính. Như vậy, 2^{27} block bộ nhớ chính đều có thể được nạp lên một slot bất kỳ của bộ nhớ Cache. Để theo dõi xem block nào của bộ nhớ chính được lưu trên 1 slot, mỗi slot có một trường **tag** có độ lớn 27 bit được sử dụng để mã hóa block tương ứng. Trường tag này chính là 27 bit có trọng số

lớn nhất trong 32 tín hiệu địa chỉ mà hệ thống đưa vào Cache. Tất cả các giá trị tag đều được lưu trên một vùng nhớ đặc biệt. Mỗi khi có một block mới được lưu vào Cache, giá trị tag của nó sẽ được cập nhật vào vùng nhớ này.

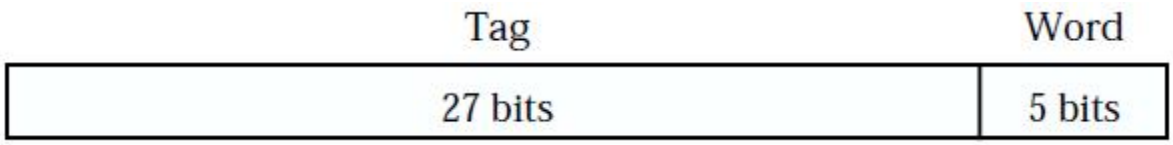


Hình 7.13. Phương pháp ánh xạ bộ nhớ

Khi một chương trình được nạp vào bộ nhớ chính lần đầu, Cache được coi là trống. Khi thực thi chương trình đó, mỗi một slot sẽ có 1 bit có tên **valid** được sử dụng để chỉ ra rằng block mà nó chứa có thuộc chương trình đang chạy hay không. Ngoài ra, mỗi slot còn có một bit **dirty** được sử dụng để theo dõi xem block nằm trên slot đó có bị thay đổi nội dung hay không. Nếu slot đó đã thay đổi nội dung thì nó cần được ghi ngược trở lại lên bộ nhớ chính trước khi slot đó được tái sử dụng để nạp một block khác.

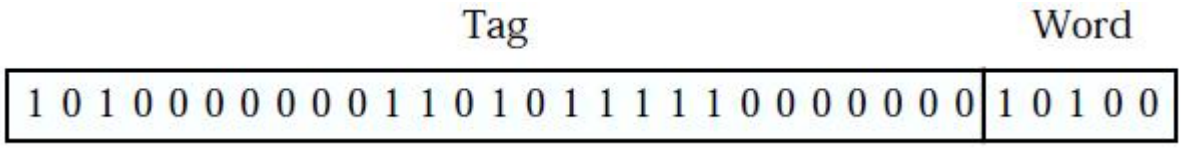
Khi CPU cần lấy nội dung từ bộ nhớ, nó sẽ kiểm tra trên Cache trước, nếu Cache có nội dung đó, kết quả kiểm tra sẽ là một **hit**, nếu không có, kết quả là một **miss**. Khi xử lý lệnh đầu tiên của một chương trình, kết quả là một miss vì chưa có một block nào được nạp lên Cache.

Trong quá trình ánh xạ Cache, mỗi block nhớ đều có thể được ánh xạ đến bất cứ một slot bất kỳ. Việc ánh xạ từ bộ nhớ chính đến Cache được thực hiện thông qua việc phân vùng địa chỉ thành 2 trường tag và word được như hình dưới đây. Khi thực hiện quá trình truy xuất bộ nhớ chính, bộ điều khiển Cache sẽ truy xuất vào Cache trước, tìm kiếm Cache tag tương ứng xem block cần truy xuất



có trong Cache hay không. Với mỗi slot, nếu bit valid bằng 1, trường tag sẽ được kiểm tra và so sánh với địa chỉ mà CPU cần truy cập. Tất cả các giá trị tag sẽ được kiểm tra cùng một lúc. Nếu có một giá trị tag nào trùng với giá trị tag mà địa chỉ ô nhớ, nội dung ô nhớ sẽ được lấy từ Cache. Nội dung của trường word chính là địa chỉ của ô nhớ này. Nếu không có giá trị tag nào được tìm thấy, khi đó, block trên bộ nhớ chính chứa nội dung cần nạp sẽ được nạp vào Cache rồi sau đó sẽ được chuyển về CPU. Các trường tag, valid, và dirty sẽ được cập nhật rồi hệ thống sẽ tiến hành thực thi.

Để hiểu rõ hơn quá trình này, ta sẽ tìm hiểu quá trình truy cập ô nhớ $(A035F014)_{16}$. Ô nhớ này coi như đã được nạp lên Cache. Địa chỉ của ô nhớ được mô tả như dưới đây



Trong địa chỉ đó, 27 bit cao của nó có giá trị tương ứng là $(501AF8)_{16}$ chính là địa chỉ tag. Nếu địa chỉ ta cần truy xuất có mặt trong Cache thì ta sẽ tìm thấy một slot có tag mang giá trị $(501AF8)_{16}$. Nếu không tìm thấy bất cứ một slot nào có tag như vậy, một miss sẽ diễn ra. Khối dữ liệu tương ứng với trường tag bằng $(501AF8)_{16}$ sẽ được nạp từ bộ nhớ chính vào một slot nào đó trên Cache.

Mặc dù phương pháp ánh xạ tỏ ra rất hiệu quả nhưng có 2 vấn đề xảy ra trong việc truy xuất bộ nhớ, đây cũng chính là hạn chế của phương pháp ánh xạ. Thứ nhất, quá trình xác định xem slot nào được giải phóng khi một block mới được đưa vào Cache là rất phức tạp. Quá trình này đòi hỏi một lượng phần cứng đáng kể gây trễ hệ thống. Vấn đề thứ hai là khi tìm một địa chỉ tag trong Cache, trường địa chỉ tag so sánh đồng thời ra làm sao với 2^{14} trường tag trong Cache. Mỗi một phương pháp giải quyết đều có hạn chế riêng của nó. Vấn đề này sẽ được nói rõ hơn trong phần 7.6.2 và 7.6.3.

Chính sách thay thế trong ánh xạ Cache

Khi một block mới cần được ánh xạ vào Cache, các slot trong Cache cần được kiểm tra xem có thể cho phép ghi dữ liệu vào hay không. Nếu có slot thích hợp (như trong trường hợp chương trình bắt đầu thực thi), slot đầu tiên có bit valid bằng 0 sẽ được sử dụng. Khi tất cả các bit valid của tất cả các slot đều bằng 1, một slot sẽ được giải phóng để dành chỗ cho block mới. Có 4 chính sách thay thế có thể được sử dụng: LRU (Least recently used), FIFO (First In First Out), LFU (Least frequently used) và random. Chính sách thứ năm được sử dụng với mục đích phân tích là phương pháp tối ưu.

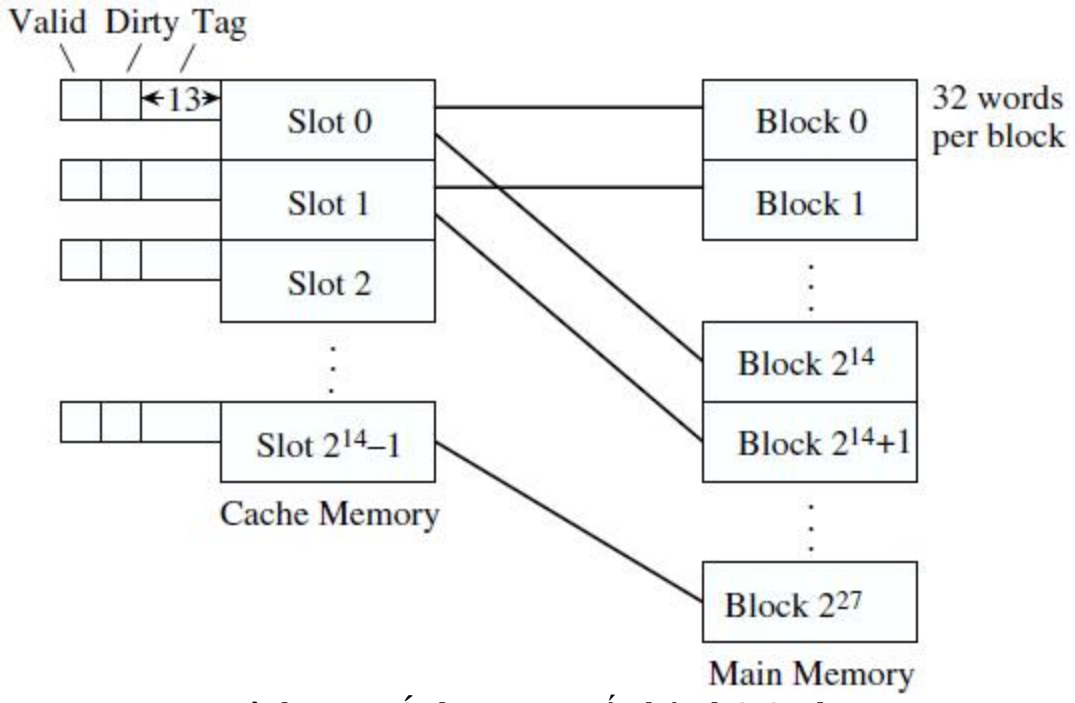
Với chính sách LRU, một biến thời gian được thêm vào mỗi slot. Biến thời gian này được cập nhật mỗi khi có một slot bất kỳ được truy cập. Khi một slot phải bị giải phóng cho một block mới, nội dung của slot mà ít được sử dụng nhất được xác định bởi biến thời gian ở trên sẽ được giải phóng và block mới sẽ được ghi vào slot này. Chính sách LFU hoạt động tương tự như vậy, ngoại trừ một điều là mỗi slot được cập nhật đều đặn bởi một bộ đếm gắn vào mỗi slot. Khi cần giải phóng một slot, slot nào được sử dụng với tần số ít nhất sẽ được giải phóng. Chính sách FIFO sẽ thay thế lần lượt các slot theo một chu trình định sẵn, slot này kế tiếp slot khác. Chính sách ramdom chọn ngẫu nhiên một slot bất kỳ để giải phóng.

Ưu nhược điểm của phương pháp ánh xạ Cache

Ánh xạ Cache có ưu điểm là bất cứ block bộ nhớ chính nào cũng có thể được nạp vào Cache slot. Điều này có nghĩa là không cần biết nội dung của block là dữ liệu hay chương trình, nếu còn slot cho bất cứ một block nào thì nó sẽ có thể được lưu trữ vào Cache. Nhưng điều này lại làm cho các nhà thiết kế phần cứng phải bổ xung thêm phần cứng để quản lý các block này. Mỗi slot phải có 27 bit tag để xác định địa chỉ trong bộ nhớ chính, và các tag phải được kiểm tra đồng thời. Điều đó có nghĩa là với ví dụ trên, vùng nhớ lưu trữ tag phải có dung lượng 27×2^{14} bit, đồng thời như đã nói ở trên, phải có một cơ chế để tìm kiếm, kiểm tra nội dung của tất cả các tag một cách đồng thời. Ta sẽ bàn về vấn đề này ở những phần sau trong chương này.

Bằng cách hạn chế vị trí mà mỗi block bộ nhớ có thể được nạp vào Cache, ta có thể loại bỏ sự cần thiết của vùng nhớ tag. Loại bộ nhớ Cache như vậy hoạt động theo nguyên tắc ánh xạ trực tiếp mà ta sẽ thảo luận ở phần kế tiếp ngay sau đây

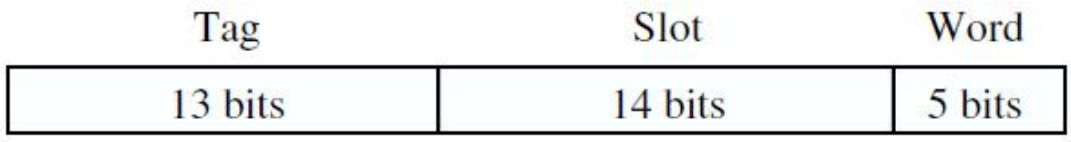
7.6.2. Ánh xạ trực tiếp bộ nhớ Cache



Hình 7.14. Ánh xạ trực tiếp bộ nhớ Cache

Hình 7.14 thể hiện nguyên lý của ánh xạ trực tiếp bộ nhớ Cache. Cũng như trong ví dụ trên, bộ nhớ được chia thành 2^{27} block có dung lượng 32 word, và Cache có 2^{14} slot. Số lượng block bộ nhớ cần được ánh xạ lên slot lớn hơn rất nhiều so với số lượng slot. Tỷ lệ là 2^{13} block/1slot. Để làm được điều này, trường tag với dung lượng 13 bit được thêm vào mỗi slot để lưu trữ một giá trị để nhận dạng một dải từ 0 đến 2^{13}-1

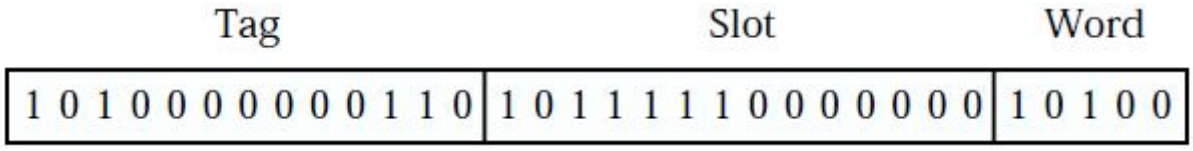
Trong quá trình ánh xạ trực tiếp, mỗi một block bộ nhớ chính được ánh xạ lên một và chỉ một slot, nhưng một slot có thể lưu trữ nhiều hơn một block tại các thời điểm khác nhau. Quá trình thực hiện ánh xạ được thực hiện theo phương thức được trình bày ở dưới đây.



Địa chỉ 32 bit mà hệ thống đưa ra được chia thành trường tag 13 bit, tiếp đó là trường slot 14 bit và cuối cùng là trường word 5 bit. Khi cần truy xuất một địa chỉ, trường slot sẽ kiểm tra xem trong 2^{14} slot mà nó quản lý có nội dung cần truy xuất hay không. Nếu bit valid bằng 1, trường tag sẽ được so sánh với các trường tag của các slot. Nếu trường tag là giống nhau, trường word sẽ chỉ ra vị

trí trên slot cần truy xuất. Nếu bit valid bằng 1 nhưng trường tag không giống nhau, slot đó sẽ được ghi ngược trở về bộ nhớ chính nếu bit dirty là 1,và block trên bộ nhớ sẽ được đọc vào slot. Khi một chương trình bắt đầu được thực thi, bit valid sẽ bằng 0, và block dữ liệu sẽ được viết vào slot một cách bình thường. Bit valid tiếp đó sẽ được set lên 1, và chương trình sẽ truy xuất vào Cache để thực thi.

Ví dụ như khi ta truy xuất vào địa chỉ nhớ $(A035F014)_{16}$, trường địa chỉ sẽ là



Nếu ô nhớ cần tìm có trên Cache, nó sẽ được tìm thấy ở địa chỉ $(14)_{16}$ của slot $(2F80)_{16}$ và tag có giá trị là $(1406)_{16}$.

Ưu nhược điểm của phương pháp ánh xạ trực tiếp

Phương pháp ánh xạ trực tiếp thực hiện tương đối đơn giản. Bộ nhớ tag trong trường hợp này có kích thước 13 x 2^{14} bit, chỉ bằng một nửa so với phương pháp ánh xạ thông thường. Hơn nữa, phương pháp ánh xạ trực tiếp không cần thực hiện công việc tìm kiếm địa chỉ tag, bởi vì địa chỉ từ CPU đưa sang được sử dụng "trực tiếp".

Việc thực hiện đơn giản này bị đánh đổi bởi cả chi phí và hiệu suất. Ta sẽ thấy được vấn đề đó khi tìm hiểu điều gì có thể xảy ra khi thực hiện truy cập bộ nhớ dung lượng 2^{19}, bằng với dung lượng của Cache. Nếu ma trận nhớ thực hiện giải mã 2 bước, tức là phải giải mã theo hàng và theo cột, sự phức tạp bắt đầu tăng lên. Mỗi ô nhớ được truy cập được trả về là một miss, hệ thống phải truy cập vào bộ nhớ chính để đọc nội dung cả môt block, mà thực tế ta chỉ cần nội dung của đúng 1 ô nhớ. Tệ hơn nữa, chỉ có một phần rất nhỏ bộ nhớ Cache được sử dụng.

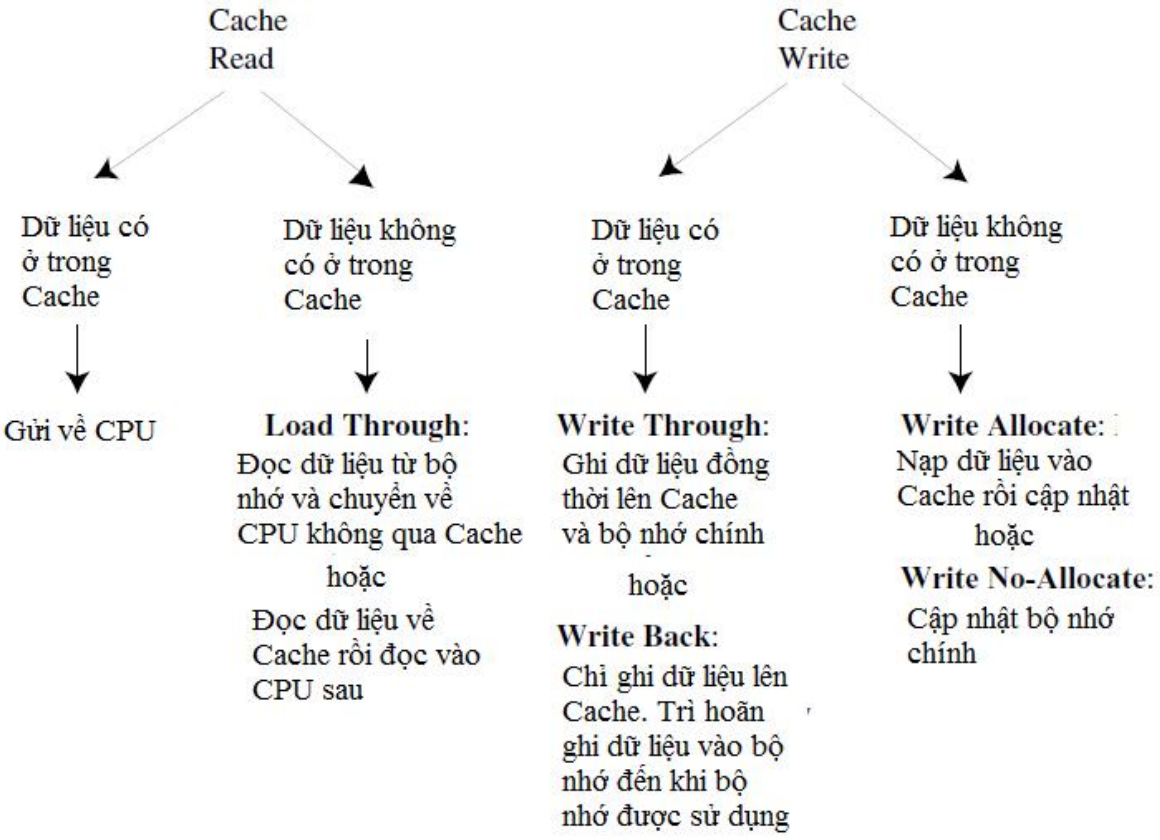
7.6.3. Hoạt động của Cache

Trong phần trên, ta đã nghiên cứu về 2 phương pháp điều khiển Cache là ánh xạ thông thường và ánh xạ trực tiếp. Việc thay đổi giữa 2 phương pháp được thực hiện mà không cần phải thay đổi một chút nào về phần cứng cũng như phần

mềm hệ thống. Việc thay đổi giữa các phương pháp chỉ ảnh hưởng đến tốc độ hoạt động của hệ thống.

Như vậy, tốc độ truy cập là mục tiêu chính của Cache, nhưng để hiểu rõ thêm về mục tiêu này, có một số vấn đề cần được giải quyết như điều gì ngăn chặn một word hay một block dịch chuyển giữa Cache và bộ nhớ chính. Các chính sách đọc và ghi hoạt động trong bộ nhớ Cache được tổng kết ở hình 7.15. Các chính sách này. Các chính sách này phụ thuộc vào việc có hay không các yêu cầu xuất phát từ Cache.

Khi có nhu cầu đọc dữ liệu từ Cache, nếu dữ liệu có ở trong Cache, một hit sẽ xảy ra và dữ liệu sẽ được đưa trực tiếp về CPU. Nếu có một miss, khối dữ liệu chứa thông tin sẽ được đọc vào Cache rồi dữ liệu đó sẽ được đưa vào CPU. Trong một số cách tổ chức, khi một miss xảy ra, nội dung cần đọc sẽ được đưa trực tiếp từ bộ nhớ chính và đưa về CPU mà không cần phải điền đầy nội dung của slot, cách thực hiện này được gọi là **load-through**.



Hình 7.15. Các chính sách đọc và ghi của Cache

Trong quá trình ghi dữ liệu lên Cache, nếu từ cần ghi có ở trong Cache, hệ thống sẽ ghi đồng thời dữ liệu đó lên Cache và bộ nhớ, quá trình **write-through**. Nếu quá trình thực hiện ghi trên Cache trước và trì hoãn quá trình ghi trên bộ nhớ, đó được gọi là **write-back**. Nếu dữ liệu cần ghi không có trong Cache, hệ thống có thể thực hiện quá trình ghi bằng cách nạp khối dữ liệu cần ghi vào Cache rồi cập nhật nó, **write-allocate**, hoặc cập nhật trực tiếp bộ nhớ chính mà không cần thông qua Cache, **write-no-allocate**.

Với câu hỏi chính sách ghi và đọc Cache nào là tốt nhất, sẽ không có một câu trả lời nào được đưa ra. Mỗi một cách tổ chức bộ nhớ Cache thường được xây dựng riêng biệt cho từng loại máy tính khác nhau. Để xây dựng một bộ nhớ Cache, người ta phải tiến hành chạy mô phỏng các thiết kế khác nhau rồi mới lựa chọn một thiết kế phù hợp

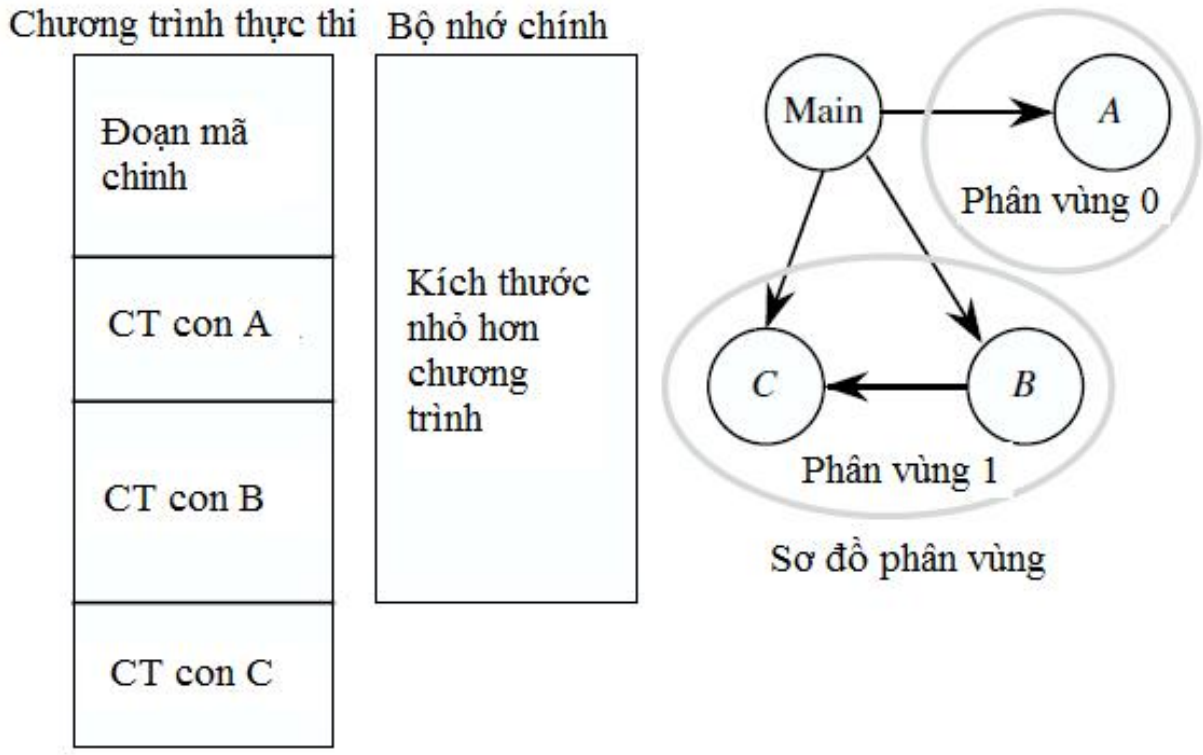
7.7. Bộ nhớ ảo

Mặc dù có tiến bộ rất lớn trong việc tăng dung lượng và giảm kích thước bộ nhớ nhưng dường như bộ nhớ vẫn được coi là hữu hạn và dường như vẫn chưa đáp ứng được nhu cầu của người sử dụng. Một giải pháp kinh tế là mở rộng không gian bộ nhớ bằng các bộ nhớ phụ bên ngoài. Việc sử dụng bộ nhớ bên ngoài với mục đích như bộ nhớ chính tỏ ra khá hiệu quả vì giá thành thấp, đồng thời hệ thống được coi như là được mở rộng “vô hạn”. Tuy nhiên việc quản lý bộ nhớ bên ngoài tỏ ra khá phức tạp. Trong mục này, chúng ta sẽ nghiên cứu công việc quản lý bộ nhớ bên ngoài thông qua bộ nhớ ảo.

7.7.1. Overlay

Một phương pháp được thực hiện sớm nhất để sử dụng ổ đĩa ngoài để sử dụng như bộ nhớ chính đó là kỹ thuật overlays. Trong kỹ thuật này, lập trình viên có trách nhiệm quản lý bộ nhớ mà nó sử dụng. Hình 7.16. là một ví dụ trong đó một chương trình bao gồm đoạn mã chương trình chính và 3 chương trình con A, B, và C. Bộ nhớ chính có kích thước nhỏ hơn kích thước của toàn bộ chương trình, nhưng lớn hơn kích thước của đoạn chương trình chính và từng chương trình con. Một thủ thuật quản lý bộ nhớ được sử dụng là overlays sẽ thay đổi chương trình sao cho nó sẽ lần lượt cho từng phần của chương trình vào bộ nhớ

chính, đọc từng phần các dữ liệu đó và thực thi. Cụ thể, hệ thống sẽ hoạt động như sau. Chương trình chính sẽ phân chia thành các phân vùng. Phân vùng 0



Hình 7.16. Phân vùng hoạt động overlays

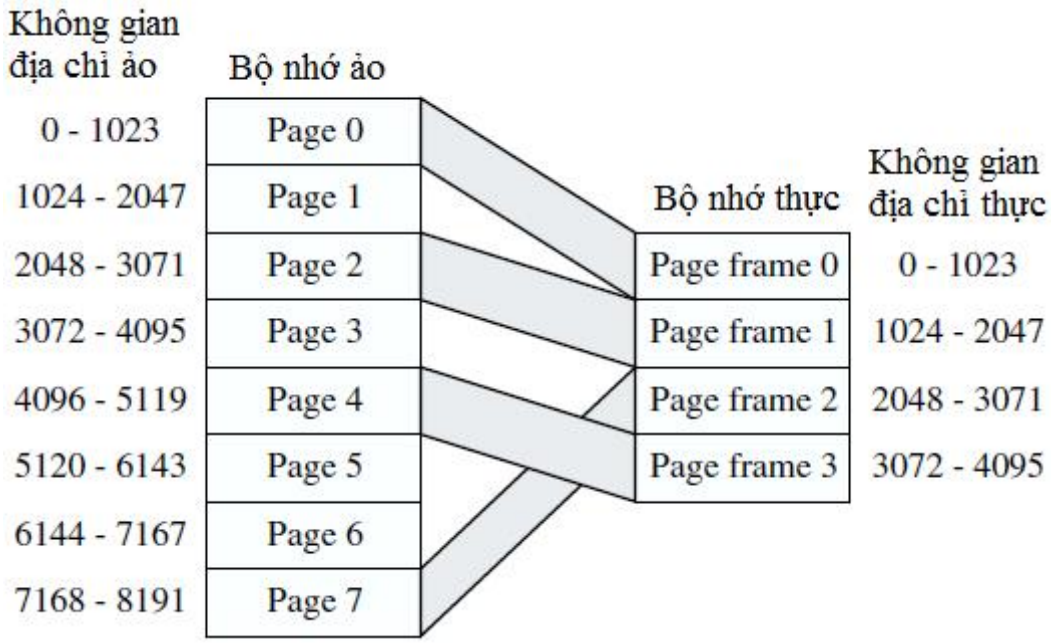
bao gồm mã code của chương trình con A bởi vì chỉ có chương trình chính mới gọi tới A. Phân vùng 1 bao gồm mã code của chương trình con B và C vì giả sử chương trình chính gọi đến B nhưng B lại gọi tới C, đồng thời chương trình chính cũng gọi tới C. Chương trình chính sẽ luôn được lưu trên bộ nhớ chính. Khi chương trình chính gọi đến chương trình con A, phân vùng 0 sẽ được nạp vào bộ nhớ, trong khi đó phân vùng 1 vẫn được lưu trong bộ nhớ gắn ngoài hoặc các ổ đĩa. Khi chương trình gọi tới B hoặc C, phân vùng 0 sẽ bị thay thế bởi phân vùng 1, và khi cần thiết, phân vùng 1 cũng có thể lại bị thay thế bởi phân vùng 0. Như vậy, tại mọi thời điểm, bộ nhớ chính của hệ thống không nhất thiết phải lưu toàn bộ chương trình. Nó sẽ chi lưu những phân vùng chương trình cần thiết cho sự hoạt động của chương trình.

Mặc dù phương thức này hoạt động tốt trong nhiều trường hợp khác nhau nhưng một giải pháp hoàn thiện hơn vẫn cần có để quản lý overlays. Hơn thế nữa, khi hệ thống chạy nhiều hơn 1 chương trình tại một thời điểm, việc quản lý

overlays là không thể. Đó chính là lý do dẫn đến một phương thức quản lý mới là phân trang sẽ được mô tả ở phần ngay sau đây

7.7.2. Phân trang

Phân trang là quá trình thực hiện overlays tự động được quản lý bởi hệ điều hành. Không gian địa chỉ được chia thành từng khối có kích thước bằng nhau được gọi là từng trang. Dung lượng mỗi trang thông thường là một giá trị số mũ cơ số 2, chẳng hạn là 2^{10} = 1024 byte. Phân trang làm cho bộ nhớ vật lý có vẻ lớn hơn thực tế bằng cách ánh xạ không gian địa chỉ thực tế lên không gian địa chỉ ảo. Các dữ liệu lưu trữ trên không gian địa chỉ ảo thực tế được lưu trữ trên ổ đĩa. Hình 7.17 mô tả quá trình ánh xạ này. 8 trang bộ nhớ ảo được ánh xạ lên 4 trang bộ nhớ thực

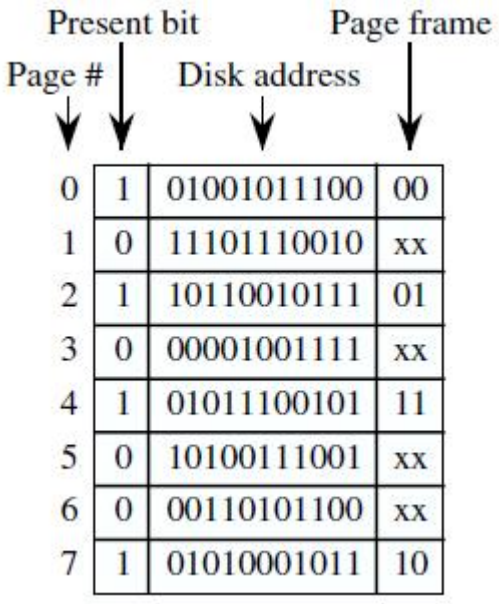


Hình 7.17. Phương pháp phân trang

Khi CPU cần truy xuất vào một ô nhớ, ô nhớ đó sẽ nằm trong không gian địa chỉ ảo. Trong mô tả hình 7.17, địa chỉ mà CPU có thể truy cập từ 0 đến 8191. Tuy nhiên dữ liệu ở dải địa chỉ trên có một nửa được lưu trữ trên bộ nhớ thật, một nửa nằm trên ổ đĩa cứng. Nếu dữ liệu nằm trên bộ nhớ thật, việc truy cập nó sẽ được thực hiện trực tiếp. Nếu dữ liệu không nằm trên bộ nhớ thật, hệ thống sẽ thực hiện quá trình phân trang theo quy trình sau

1. Xác định một trang sẽ bị xóa đi. Nội dung của trang sẽ bị xóa phải được ghi ngược trở lại ổ đĩa cứng bên ngoài nếu nội dung của trang đó có sự thay đổi.
2. Trang bộ nhớ ảo mà ta muốn truy cập sẽ được xác định trên ổ đĩa cứng và ghi vào bộ nhớ thực ở trang bộ nhớ vừa bị xóa đi (trang được xác định ở mục 1)
3. Cập nhật bảng phân trang page number để ghi lại trang bộ nhớ ảo được ghi lên trang bộ nhớ thật
4. Cập nhật bộ nhớ thực để lấy dữ liệu về hệ thống

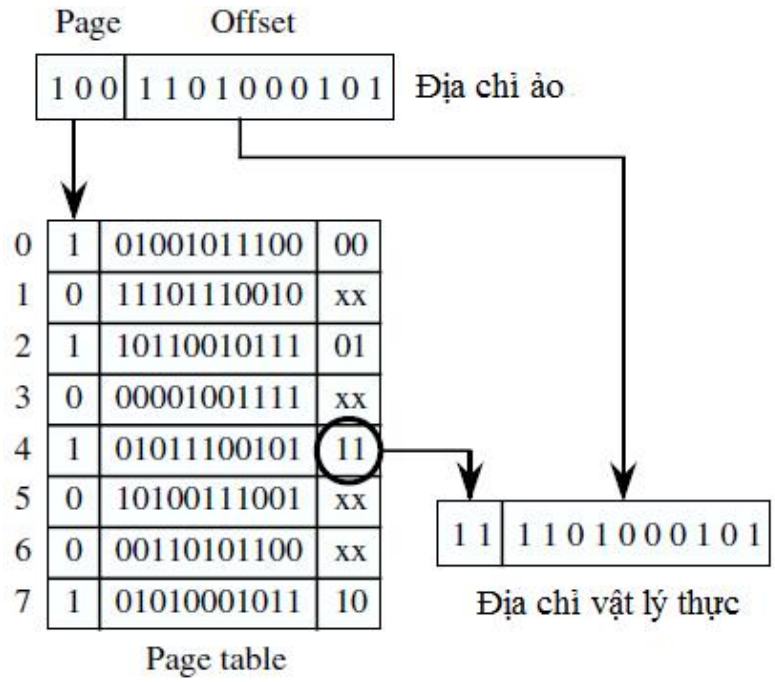
Để truy cập không gian địa chỉ bộ nhớ ảo 2^{13} như hình 7.17, CPU sẽ phải cung cấp một giá trị địa chỉ có độ lớn 13 bit. Địa chỉ này được chia làm 2 trường là trường page number có độ dài 3 bit để mã hóa số trang ảo từ page 0 đến page 7. 10 bit còn lại là địa chỉ offset trong trang. Khi hệ điều hành quản lý việc phân trang nhận được địa chỉ này, nó sẽ phân tích thành một địa chỉ có độ dài 13 bit nhưng sẽ có 3 trường. Trường thứ nhất là Present bit có nhiệm vụ xác định xem ô nhớ cần truy xuất có nằm trên bộ nhớ thực (nếu có giá trị 1) hay không (nếu có giá trị 0). Trường thứ 2 là địa chỉ offset trong trang vẫn có độ lớn 10 bit. Trường thứ 3 là trường page frame có nhiệm vụ xác định xem trang đó đang được lưu trữ trong trang bộ nhớ thực nào. Hệ thống sử dụng một bảng phân trang page table để quản lý việc phân trang này. Với những trang không nằm trên bộ nhớ thực, trường Page frame không có ý nghĩa, thông thường được ký hiệu là “xx”



Page #	Present bit	Disk address	Page frame
0	1	01001011100	00
1	0	11101110010	xx
2	1	10110010111	01
3	0	00001001111	xx
4	1	01011100101	11
5	0	10100111001	xx
6	0	00110101100	xx
7	1	01010001011	10

Hình 7.18. Bảng phân trang page number

Để thực hiện việc chuyển đổi từ một địa chỉ ảo (gồm 2 trường) sang địa chỉ truy xuất bộ nhớ thực (gồm 3 trường) như đã nói ở trên, ta sẽ thực hiện như trình bày ở hình 7.19



Hình 7.19. Quá trình chuyển đổi địa chỉ ảo sang địa chỉ thực

Như trên hình 7.19, ta cần truy cập vào địa chỉ 1001101000101. 3 bit có trọng số lớn nhất (100) sẽ chỉ ra trang ảo mà hệ thống cần truy cập. Giá trị này sẽ được so sánh với bảng phân trang page table để xem trang số 100 = 4 nằm trên bộ nhớ thực hay ảo. Trong trường hợp trên, địa chỉ đó nằm trên bộ nhớ thực ở trang thực số (11). Giá trị này sẽ tạo thành 2 bit có trọng số lớn nhất trong địa chỉ thực. Trường địa chỉ offset 10 bit ở địa chỉ ảo sẽ được đưa trực tiếp vào địa chỉ offset của địa chỉ vật lý thực. Kết quả ta sẽ truy cập vào địa chỉ 111101000101 để nhận được dữ liệu cần truy cập.

Phân trang theo nhu cầu - demand paging

Khi thực thi một chương trình, không phải chương trình đó nằm trên 1 trang mà thực tế nó có thể nằm trên nhiều trang khác nhau. Vấn đề đặt ra là làm thế nào có thể nạp được trang chứa chương trình cần thực thi vào bộ nhớ thực. Quá trình này thực hiện như sau. Giả sử ban đầu chưa có một nội dung nào của chương trình đã được nạp vào bộ nhớ thực. Khi cần nạp lệnh đầu tiên của chương trình vào hệ thống, CPU sẽ đưa ra địa chỉ ô nhớ (nằm trên đĩa cứng) của

lệnh đầu tiên đó. Vì nội dung này không có trong bộ nhớ thực nên sẽ xuất hiện một lỗi **fault**. Khi phát sinh lỗi này, hệ điều hành sẽ thực hiện ánh xạ trang tương ứng ở bộ nhớ ảo vào một trang trong bộ nhớ chính. Khi đó lỗi trang sẽ được khắc phục. Các lệnh của chương trình nằm trên trang này sẽ được truy cập tiếp tục trong trang bộ nhớ thật này. Trong khi tiếp tục thực thi các lệnh trong chương trình, nếu chương trình cần truy cập vào một trang khác, nó lại phát sinh một lỗi và lỗi này sẽ lại được khắc phục. Tiếp tục như thế, tất cả các trang chứa chương trình được thực thi sẽ được nạp **theo nhu cầu** vào bộ nhớ chính

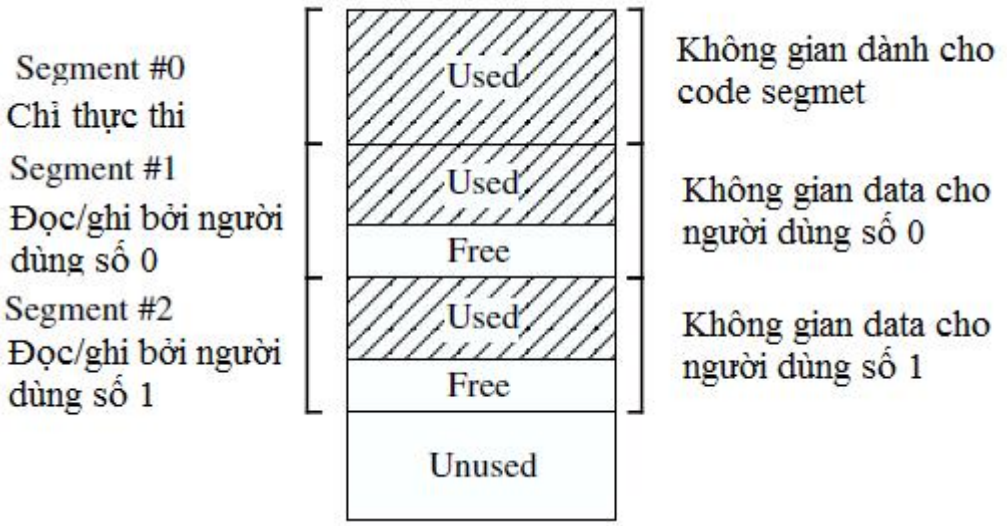
Chính sách thay thế trang

Một vấn đề nảy sinh đó là trong trường hợp kích thước chương trình lớn hơn kích thước bộ nhớ chính. Khi đó, bộ nhớ chính không thể chứa toàn bộ chương trình, nó sẽ phải xóa bớt trang không sử dụng để nhường chỗ cho một trang khác để xóa lỗi. Chính sách thay thế trang hay được sử dụng là FIFO hoặc LRU. Chính sách này tương tự như chính sách thay thế trong ánh xạ Cache mà ta đã đề cập trong chương này

7.7.3. Phân đoạn

Phân đoạn cũng là một phương thức quản lý bộ nhớ của hệ điều hành. Khác với phân trang quản lý bộ nhớ ảo (không gian bộ nhớ ảo), phân đoạn chỉ quản lý bộ nhớ thực. Nó cũng khác phân trang ở chỗ nó quản lý bộ nhớ theo chức năng chương trình. Một chương trình khi nạp vào bộ nhớ thực sẽ được quản lý theo từng đoạn **segment** theo chức năng. Đó là các phân đoạn: code segment chứa các lệnh của chương trình, data segment chứa các dữ liệu chương trình, stack segment chứa bộ nhớ stack.

Ngoài cách quản lý như trên, phân đoạn cũng có thể chia bộ nhớ thành từng đoạn, mỗi đoạn được sử dụng theo mục đích riêng, được truy cập vào bảo vệ theo từng mức mà hệ điều hành hỗ trợ. Trong ví dụ hình 7.20, hệ điều hành có thể thực hiện phân đoạn bộ nhớ thành 3 phân đoạn. Phân đoạn số 0 là code segment, chỉ chứa các lệnh của chương trình, do đó nó chỉ để chứa các lệnh thực thi, không cho phép xóa mà chỉ cho phép đọc. Phân đoạn số 1 được sử dụng để chứa các dữ liệu dành cho người dùng số 1. Chỉ người dùng số 1 mới được phép truy cập và thay đổi các dữ liệu này. Tương tự như vậy là phân đoạn số 2.



Hình 7.20. Ví dụ về phân đoạn

Phân đoạn không giống phân trang. Với phân trang, người sử dụng không cần quan tâm đến nó vì nó được thực hiện hoàn toàn tự động. Ngược lại với phân đoạn, người sử dụng (ở đây là các lập trình viên) dường như phải chú ý xem đang tương tác với phân đoạn nào, giới hạn của từng phân đoạn là gì, được phép truy cập hay không. Rất may cho chúng ta, hệ điều hành đã thay chúng ta thực hiện tất cả các nhiệm vụ đó, kể cả việc thay đổi các giá trị phân đoạn hay chuyển đổi các địa chỉ phân đoạn sang địa chỉ vật lý

TỔNG KẾT CHƯƠNG

Bộ nhớ được phân thành nhiều loại khác nhau, trong đó, loại có hiệu suất tốt nhất lại có giá thành đắt nhất, loại có hiệu suất nhỏ nhất, rẻ nhất lại có tính thương mại tốt nhất. Để dung hòa mọi lợi ích, các nhà thiết kế hệ thống đã sử dụng bộ nhớ Cache và bộ nhớ ảo

Bộ nhớ Cache nằm trong CPU chứa các khối dữ liệu nhỏ mà hệ thống hay sử dụng nhất, có khả năng truy xuất với tốc độ cao. Bộ nhớ ảo lại quản lý vùng nhớ đặt trong các ổ đĩa cứng để mở rộng dung lượng nhớ

Cache và bộ nhớ ảo đều được sử dụng trong cùng một hệ thống máy tính nhưng lại phục vụ các mục đích khác nhau. Cache tăng tốc độ truy cập bộ nhớ chính, còn bộ nhớ ảo mở rộng dung lượng bộ nhớ chính

CHƯƠNG 8: THIẾT BỊ NGOẠI VI VÀ GHÉP NỐI

Trong những chương trước, chúng ta đã tìm hiểu phương thức mà CPU tác động vào dữ liệu, hoặc phương thức để truy cập bộ nhớ chính, hoặc vùng nhớ trên ổ đĩa gắn ngoài thông qua bộ nhớ ảo. Tốc độ truy xuất bộ nhớ phụ thuộc rất nhiều vào từng loại bộ nhớ khác nhau. Tuy nhiên, việc CPU truy xuất vào các thiết bị vào ra lại khác rất nhiều so với truy xuất bộ nhớ.

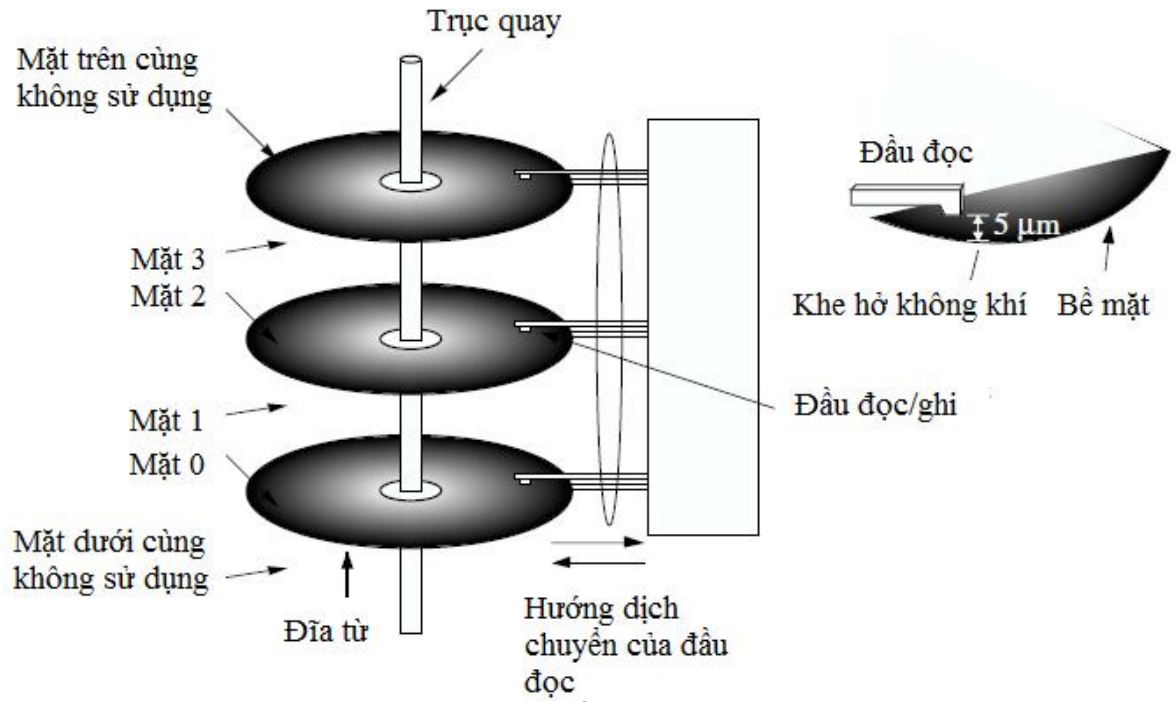
- Tốc độ truyền dữ liệu của các cổng I/O rất khác nhau, có thể là tốc độ rất chậm như đọc dữ liệu từ bàn phím, nhưng cũng có thể có tốc độ rất lớn như đọc dữ liệu từ ổ đĩa, hoặc tương tác dữ liệu thời gian thực với card đồ họa.
- Hoạt động của các I/O là không đồng bộ, tức là nó không có cùng tốc độ xung nhịp với xung nhịp CPU (việc truyền dữ liệu trên bộ nhớ có cùng tốc độ xung nhịp với CPU). Các tín hiệu bắt tay **handshaking** được bổ xung vào hoạt động của các I/O để phối hợp hoạt động của I/O trong quá trình các thiết bị đọc hoặc ghi dữ liệu.
- Chất lượng của dữ liệu truyền trên các cổng I/O không được bảo đảm tính toàn vẹn. Các tín hiệu nhiễu tác động vào đường truyền như tín hiệu điện thoại, hay các tín hiệu từ trường tác động vào các thiết bị ngoại vi có thể làm phát sinh lỗi dữ liệu. Do đó cần có những phương thức đặc biệt để đảm bảo tính toàn vẹn dữ liệu
- Một số các thiết bị ngoại vi có các thành phần cơ khí, nói chung là dễ hỏng so với CPU. Dữ liệu truyền trên hệ thống bus có thể bị gián đoạn do các thành phần cơ khí này bị hỏng, hoặc không hoạt động theo đúng chức năng của nó.
- Các modul phần mềm I/O được gọi là các driver phải được bổ xung để khắc phục các thiếu sót trên

Trong chương này, chúng ta sẽ tìm hiểu một số thiết bị ngoại vi cơ bản, bắt đầu là các ổ đĩa lưu trữ, tiếp đó là các thiết bị nhập liệu input, thiết bị xuất dữ liệu output. Tiếp theo, ta sẽ tìm hiểu một số kết nối giữa các thiết bị và một số phương pháp khắc phục lỗi trên đường truyền

8.1. Thiết bị lưu trữ

Trong chương 7, chúng ta đã tìm hiểu về các loại bộ nhớ máy tính. Bộ nhớ có tốc độ truy xuất nhanh nhất là các thanh ghi, tốc độ thấp nhất là các bộ nhớ ngoài. Đó là các ổ đĩa cứng, đĩa mềm, băng từ,...Các thanh ghi và bộ nhớ RAM cần phải được cung cấp năng lượng liên tục để lưu dữ liệu, trong khi đó các bộ nhớ ngoài như đĩa từ hay băng từ vẫn lưu được dữ liệu sau khi ngừng cung cấp nguồn. Loại ổ đĩa như vậy còn có tên gọi là non-volatile.

8.1.1. Ổ đĩa từ tính



Hình 8.1. Ổ đĩa từ tính

Đĩa từ là thiết bị lưu trữ dữ liệu với dung lượng lớn và tốc độ truy cập cao. Thành phần lưu dữ liệu của ổ đĩa gồm một hoặc nhiều đĩa được đặt cách nhau vài millimeters trên cùng một trục quay như hình 8.1. Các đĩa này thường được làm từ nhôm hoặc thủy tinh pha một lượng nhỏ nhôm. 2 bề mặt của đĩa đều được phủ một lớp từ tính như ocid sắt. Chính vì lớp ocid sắt này nên các mặt đĩa cứng, đĩa mềm hay băng từ đều có mầu nâu. Các giá trị nhị phân 0 và 1 được lưu trên các ô nhỏ trên các bề mặt này.

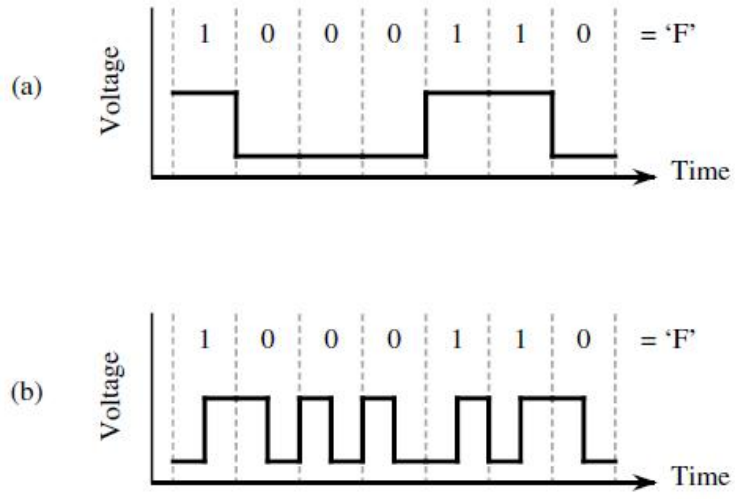
Mỗi một đầu đọc được sử dụng riêng cho từng bề mặt. Hình 8.1 sử dụng 3 đĩa từ gồm 6 mặt đĩa. Bề mặt trên cùng và dưới cùng của tổ hợp đĩa thông thường

không được sử dụng để lưu trữ bởi vì nó dễ bị ảnh hưởng của từ trường bên ngoài hơn so với các bề mặt bên trong. Các đầu đọc được gắn với nhau trên một cánh tay có thể dịch chuyển theo chiều ngang để có thể tiếp cận vào các vị trí khác nhau của bề mặt đĩa.

Trong một ổ đĩa cứng, các đĩa quay với tốc độ cố định, thông thường có thể từ 3600 đến 10000 vòng/phút (RPM). Các đầu đọc/ghi dữ liệu sẽ tạo ra từ trường ảnh hưởng đến vật liệu từ trong quá trình ghi hoặc cảm ứng điện từ trong quá trình đọc. Chỉ có một đầu đọc duy nhất hoạt động tại một thời điểm. Do đó dữ liệu đọc được truyền về theo phương thức nối tiếp serial mặc dù về mặt lý thuyết chúng có thể truyền song song. Một nguyên nhân mà chế độ đọc/ghi song song không được sử dụng nó có thể gây ra nhiễu dữ liệu gây hỏng dữ liệu. Mỗi một bề mặt đĩa sẽ chỉ tương tác với một đầu đọc bởi vì vị trí của đầu đọc luôn luôn được xác định một cách chính xác, tương ứng với vị trí của nó trên bề mặt đĩa.

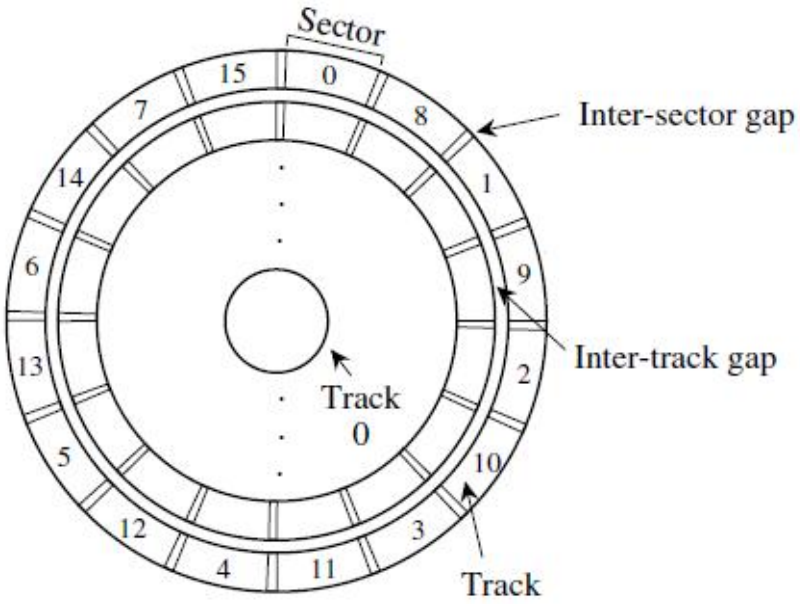
Mã hóa dữ liệu trên đĩa từ

Đầu đọc dữ liệu chỉ có thể nhận biết được sự thay đổi của từ trường, do đó, nó chỉ có thể phát hiện ra sự thay đổi logic từ 0 lên 1 hay từ 1 về 0. Phương pháp mã hóa thông dụng nhất là mã manchester, ngoài ra còn có thể sử dụng phương pháp MFM (modified frequency modulation). Để người đọc có thể so sánh, ta hãy xem xét hình 8.2. Tại hình a là mã NRZ (non-return-to-zero) của ký tự "F" được mã hóa bởi mã ASCII. Hình b cũng là ký tự đó nhưng được biểu diễn bởi mã Manchester. Trong cách mã hóa mã Manchester, trong bất cứ một chu kỳ



Hình 8.2. 2 phương pháp mã hóa ký tự

thời gian đều có một sự thay đổi giá trị điện áp. Tại thời điểm xung chuyển trạng thái từ mức thấp lên mức cao, giá trị logic thu được là 1. Ngược lại, tại thời điểm điện áp chuyển từ mức cao về mức thấp, giá trị logic thu được là 0. Ngoài ra, chu kỳ chuyển đổi trạng thái logic cũng cho ta thông tin về xung đồng bộ hệ thống.



Hình 8.3. Tổ chức dữ liệu trên một mặt đĩa từ

Để lưu dữ liệu trên đĩa cứng, đĩa cần được tổ chức theo những chuẩn được định nghĩa. Mỗi bề mặt được chia thành hàng trăm rãnh **track** là những vòng tròn đồng tâm Mỗi rãnh được chia thành từng khoảng nhỏ gọi là **sector**. Mỗi sector có thể lưu trữ được 512 byte. Giữa các sector có khoảng nhỏ gọi là inter-sector-gap. Tương tự như vậy giữa các track cũng có khoảng không gian phân cách inter-track-gap. Ngoài cách quản lý theo track và sector, hệ thống cũng có thể quản lý theo từ trụ **cylinder**. Cylinder là tập hợp các rãnh chứa đầu đọc của tất cả các bề mặt các đĩa. Ví dụ như track số 0 của của các bề mặt số 0, 1, 2, 3, 4, và 5 trên hình 8.1 sẽ tạo nên cylinder số 0. Dung lượng lưu trữ trên mỗi sector thông thường là không thay đổi đối với mỗi một loại đĩa từ.

Trong các loại đĩa cứng hiện đại, số lượng track trên cùng một sector cón thể thay đổi theo vùng, mỗi vùng chứa một số lượng sector cố định. Vùng gần tâm của đĩa từ sẽ có mật độ lưu trữ cao hơn những vùng bên ngoài. Do đó, ta có thể tăng mật độ lưu trữ tại các sector ở vùng xa tâm đĩa. Công nghệ này có tên gọi là zone-bit-recording.

Dung lượng và tốc độ đĩa

Nếu hệ thống đĩa từ có cùng một mật độ lưu trữ, dung lượng đĩa C lúc đó có thể được tính từ số lượng byte có trên 1 sector N, số lượng sector trên một track S, số lượng rãnh trên 1 bề mặt T và số lượng mặt đĩa P. Công thức tính là

$$C = N \times S \times T \times P$$

Một đĩa dung lượng cao có thể có N = 512 byte, S = 1000 sector, T = 5000 track và có P = 8 mặt đĩa. Tổng dung lượng của ổ đĩa đó là $C = 512 \times 1000 \times 5000 \times 8/2^{30}$ hay 19GB

Tốc độ truyền tối đa của ổ đĩa phụ thuộc vào 3 yếu tố: thời gian để đầu đọc di chuyển đến track dữ liệu được gọi là **seek time**, thời gian để sector cần đọc quay đến vị trí đầu đọc gọi là **rotational latency**, và thời gian để chuyển dữ liệu từ sector vào đầu đọc gọi là **transfer time**. Quá trình đọc và ghi dữ liệu đề phải trải qua 3 khoảng thời gian như vậy vì tại 1 thời điểm đầu đọc chỉ có thể đọc dữ liệu hoặc ghi dữ liệu.

Thời gian để đầu đọc dịch chuyển đến vị trí track cần đọc mất nhiều thời gian nhất trong quá trình truy cập đĩa. Nhà sản xuất thường tính theo thời gian trung bình là khoảng thời gian đầu đọc dịch chuyển $^1/_2$ khoảng cách ổ đĩa. Trong các ổ đĩa hiện đại, khoảng thời gian này xấp xỉ 10ms.

Khi đầu đọc dịch chuyển đến track cần truy cập, chúng ta một lần nữa không tính được chính xác thời gian mà đầu đọc tìm được đúng sector cần tìm. Khoảng thời gian này một lần nữa được tính bằng $^1/_2$ khoảng thời gian lớn nhất mà đầu đọc hoàn thành công việc này. Trung bình, đầu đọc sẽ mất khoảng 4 đến 8 ms.

Transfer time là khoảng thời gian mà đầu đọc hoàn thành việc đọc/ghi một lượng sector trên cùng một track. Nếu cần đọc/ghi một khối lượng dữ liệu lớn, khi kết thúc việc đọc/ghi dữ liệu trên track này, đầu đọc sẽ phải dịch chuyển sang track khác. Khoảng thời gian để đầu đọc dịch chuyển từ track này sang track khác xấp xỉ 2 ms.

Tốc độ truy cập có thể được tính toán từ các yếu tố kể trên, tuy nhiên tốc độ trong thực tế lại thấp hơn tính toán. Điều này là do có thể tốc độ truy cập bị ảnh hưởng bởi quá trình thâm nhập dữ liệu trên hệ thống bus kết nối giữa các ổ đĩa và các thành phần bên trong hệ thống máy tính, hoặc một nguyên nhân cũng tác

động đến tốc độ truy cập ổ đĩa là tốc độ kết nối dữ liệu giữa CPU và bộ nhớ chính. Các ổ đĩa hoạt động trong máy tính SCSI thường có tốc độ truy cập trung bình khoảng 5 đến 40 MB/s.

Ổ đĩa mềm

Ổ đĩa mềm, floppy disk hay diskette, bao gồm một đĩa nhựa được bao phủ lớp vật liệu từ ocid sắt từ ở trên cả 2 bề mặt. Cả 2 lớp vật liệu từ này đều có thể được sử dụng để lưu dữ liệu. Thời gian truy cập của đĩa mềm chậm hơn rất nhiều so với đĩa cứng bởi vì đĩa mềm không thể quay với tốc độ cao. Tốc độ trung bình của ổ đĩa mềm khoảng 300 RPM và tốc độ này có thể thay đổi phụ thuộc vào vị trí của đầu đọc để tối ưu hóa tốc độ truyền. Thời gian truy cập của đĩa mềm chỉ khoảng 250 đến 300 ms, thấp hơn khoảng 10 lần so với ổ đĩa cứng. Dung lượng đĩa mềm lên tới 1.44 MB.

Sử dụng ổ đĩa mềm rất kinh tế bởi vì có việc tháo lắp ổ đĩa với hệ thống rất dễ dàng và bởi vì kích thước của nó rất nhỏ. Vì đầu đọc đĩa mềm có thể gây ra những vết xước trên bề mặt đĩa nên ổ đĩa mềm chỉ quay khi nó cần đọc/ghi dữ liệu.

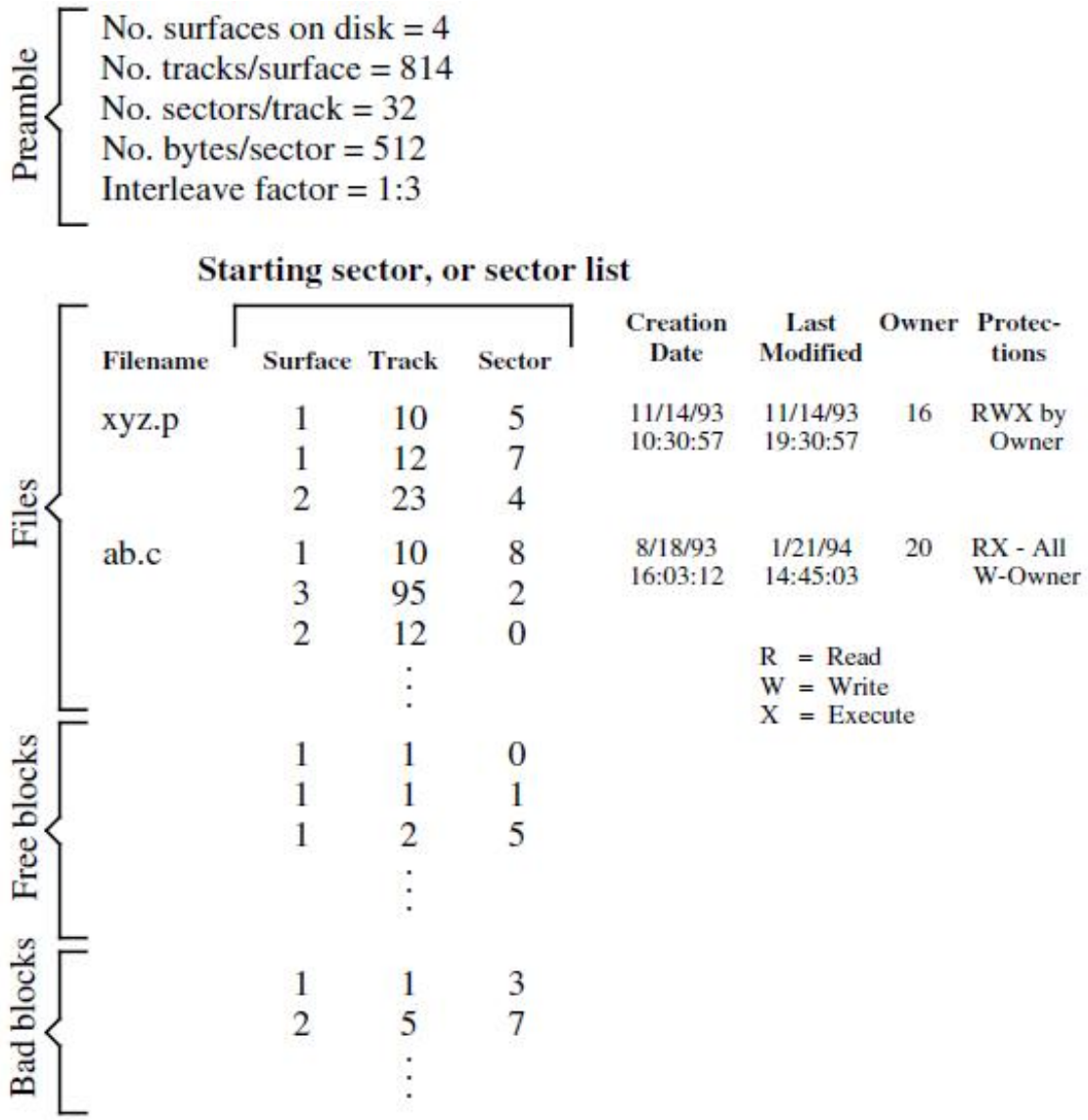
Một số đĩa mềm dung lượng cao có thể lưu trữ với dung lượng lớn. Ví dụ như đĩa mềm của Iomega Zip có thể lưu 100MB và thời gian truy cập gấp 2 lần so với ổ đĩa cứng. Đĩa có dung lượng lớn nhất hiện nay là Iomega Jaz có dung lượng lên tới 2GB với tốc độ truy cập tương đương Iomega Zip.

Hệ thống quản lý file trên ổ đĩa

Một file dữ liệu được lưu trữ trên nhiều sector ở các vị trí khác nhau trong ổ đĩa. Chúng sẽ được liên kết với nhau để tạo thành một thể thống nhất. Việc tổ chức các thành phần nhỏ của một file có thể được thực hiện theo nhiều cách khác nhau. Phương pháp có hiệu quả nhất là lưu giữ trong các sector liên tiếp nhau để giảm thiểu thời gian tìm kiếm các mảnh. Một ổ đĩa tất nhiên sẽ lưu trữ nhiều file, do đó tìm kiếm một vị trí đủ lớn để lưu trữ 1 file nói chung là khó thực hiện. Để làm được việc này, hệ điều hành phải có một bản sao chép dung lượng của các khoảng trống để sử dụng khi cần ghi 1 file vào ổ đĩa.

Một phương pháp khác được sử dụng là lưu trữ file trên các phân mảnh khác nhau, các phân mảnh này được liên kết với nhau thông qua một bảng lưu giá trị FAT (File allocation table). Với phương pháp này, dung lượng file lưu trữ có thể tăng lên. Tuy nhiên, sau một thời gian sử dụng, file này sẽ bị phân mảnh. Các nhà sản xuất khác nhau có các phương pháp khác nhau để có thể chống lại sự phân mảnh này.

Bàn kỹ hơn một chút về bảng FAT. Như đã nói ở trên, đây là một bảng lưu các giá trị, thông thường đó là tên file, vị trí của các sector chứa dữ liệu các file, ngày tạo ra file,.... Dung lượng file không thể hiện trên bảng dữ liệu này nhưng hệ điều hành có thể tính toán dung lượng file dựa vào số lượng sector chứa file.

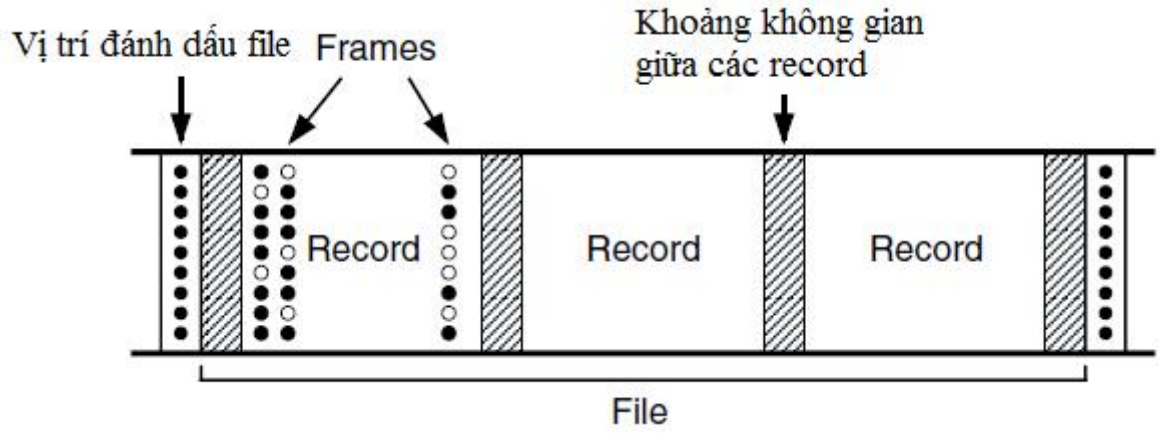


Hình 8.4. Ví dụ về nội dung của bảng FAT

8.1.2. Băng từ

Băng từ là một phương pháp lưu trữ dữ sử dụng một đầu đọc. Đầu đọc này có bộ phận để đọc và ghi dữ liệu riêng biệt. Khi dải băng từ chạy qua đầu đọc, nó sẽ gây ảnh hưởng từ lên băng từ trong quá trình ghi dữ liệu hoặc cảm ứng từ trường của băng từ trong quá trình ghi. Phương pháp lưu trữ sử dụng băng từ rất rẻ bởi vì nó có thể lưu trữ dung lượng rất lớn. Tuy nhiên việc truy cập dữ liệu lại rất chậm bởi vì tất cả các sector của băng từ ở phía trước vị trí cần đọc dữ liệu sẽ phải được đọc trước khi nội dung đọc được tìm thấy.

Thông tin lưu trữ trong băng từ theo 2 chiều được mô tả trong hình 8.5. Các bit được lưu dữ liệu theo chiều rộng của băng từ (ghi theo frame) và lưu trữ theo chiều dài của băng từ (ghi theo record). 1 file được lưu trữ theo một tập hợp các record. Và 1 record là một đơn vị vật lý nhỏ nhất mà dữ liệu có thể được đọc hoặc ghi lên băng từ. Bình thường các băng từ không tự dịch chuyển. Khi muốn



Hình 8.5. Lưu trữ dữ liệu trên băng từ

truy xuất dữ liệu, một động cơ sẽ được sử dụng để kéo băng từ chuyển động trong một khoảng thời gian. Khi băng từ đạt đến tốc độ cần thiết, các bản ghi record sẽ được ghi, sau đó băng từ sẽ ngừng chuyển động. Việc tăng tốc động cơ làm băng từ dịch chuyển và dừng động cơ làm cho trên băng từ có những khe hở giữa các bản record.

Băng từ thích hợp cho việc lưu trữ dữ liệu với dung lượng lớn,ví dụ như backup dữ liệu trên ổ đĩa hoặc quét ảnh, nhưng không thích hợp với các truy cập dữ liệu ngẫu nhiên. Có 2 nguyên nhân chính của ứng dụng trên. Thứ nhất, việc truy cập vào một bản ghi bất kỳ đòi hỏi rất nhiều thời gian vì vị trí băng từ cần truy cập không phải lúc nào cũng gần vị trí hiện tại. Thứ hai, khi tiến hành ghi dữ liệu lên một vị trí ở giữa băng từ, đè lên một nội dung khác có thể làm mất dữ liệu

của 1 bàn record ở gần đó mặc dù các record có cùng kích thước nhưng khoảng cách giữa các record lại không phải là một giá trị định trước.

Một bản ghi record vật lý thường được chia thành các bản ghi record logic. Ví dụ, một bản record vật lý có 4096 byte có thể chia thành 4 bản ghi logic với độ dài của mỗi bản ghi là 1024 byte. Việc truy cập các bản ghi logic được thực hiện bởi hệ điều hành, do đó người sử dụng không cần quan tâm đến nó.

Ngoài cách tổ chức trên, ta có thể có một cách tổ chức khác nữa là sử dụng các bản record có độ dài thay đổi. Một ký hiệu đặc biệt được đặt giữa các bản record để đánh dấu sự bắt đầu và kết thúc của mỗi bản ghi record.

8.1.3. Ổ đĩa quang

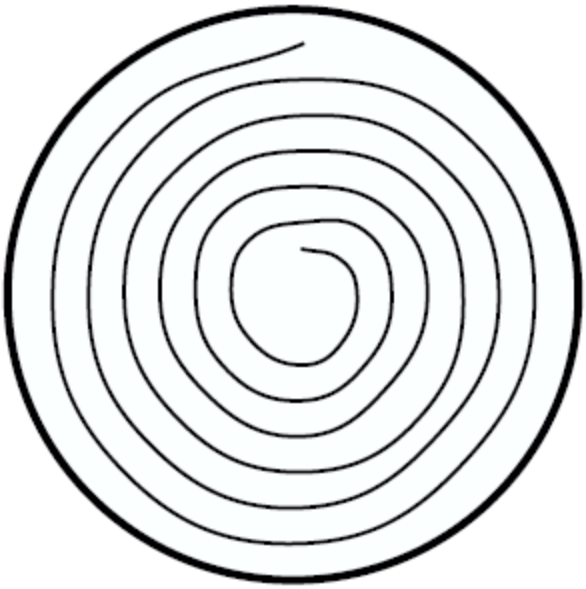
Các ổ đĩa quang áp dụng một số công nghệ mới trong quang học để lưu trữ và khôi phục dữ liệu. Đĩa CD (Compact Disc) và DVD (Digital Versatile Disc) sử dụng ánh sáng phản xạ để đọc dữ liệu mã hóa

Compact Disc – CD

Đĩa CD được giới thiệu vào năm 1983 được sử dụng để lưu dữ liệu âm thanh. Đĩa CD có dung lượng lưu trữ khoảng 74 phút âm thanh ở chế độ digital steoreo 2 kênh. Tín hiệu âm thanh được trích mẫu ở $2 \times 44{,}000$ 16-bit, hay là gần 700MB dung lượng. Từ khi được giới thiệu vào năm 1983, công nghệ CD đã phát triển để tăng mật độ lưu trữ, giảm giá thành để phát triển thành đĩa lưu trữ dữ liệu CD ROM (CD read only memories). Với giá thành rất rẻ, chỉ vài cent cho một đơn vị MB, CD ROM nhanh chóng phát triển và thay thế đĩa mềm.

CD ROM là đĩa chỉ đọc bởi vì nó được nhân bản từ 1 đĩa gốc master giống như là đối với đĩa nhạc CD. Một đĩa CD ROM được cấu tạo từ nhựa plastic được phủ bên ngoài lớp nhôm mỏng, có khả năng phản xạ ánh sáng theo cách hướng khác nhau để tạo thành các bề mặt Land hoặc các lỗ trống Pit. Đĩa master được chế tạo với độ chính xác cao dựa vào ánh sáng laser công suất cao. Các đĩa nhân bản có độ chính xác thấp hơn, do đó một bộ sửa lỗi đã được sử dụng để phát hiện và khôi phục dữ liệu. Dữ liệu trên CD ROM cũng được sử dụng mã Manchester.

Không giống như đĩa từ sử dụng lưu trữ thông tin trên các sector trên các track là các đường tròn đồng tâm, đĩa CD sử dụng các rãnh xoắn trôn ốc như hình 8.7



Hình 8.7. Bề mặt đĩa CD ROM

Các pit dữ liệu được đặt với khoảng cách bằng nhau từ đầu cho đến cuối đĩa. Tốc độ quay của đĩa khoảng 30 RPM giống như đối với đĩa mềm, nhưng tốc độ sẽ được điều chỉnh. Tốc độ sẽ chậm hơn khi đầu đọc ở rìa đĩa và nhanh hơn khi đầu đọc ở trung tâm đĩa. Với các đặc tính đó, tốc độ truy xuất trên CD ROM tương đương với trên đĩa mềm vì nó có độ trễ cao. Ổ CD ROM đọc dữ liệu với tốc độ 24× tức là gấp 24 lần tốc độ của một đĩa âm thanh audio CD. Tốc độ đọc này càng ngày càng được cải thiện.

Công nghệ CD ROM thích hợp với việc phân phối một lượng lớn đĩa dữ liệu, không tốn kém khi có càng nhiều bản sao. Tuy nhiên nếu chỉ một số lượng nhỏ đĩa được sao chép, giá thành cho từng đĩa sẽ tăng cao bởi vì các đĩa CD không thể được in với giá thành rẻ mà số lượng ít.

Một công nghệ mới được phát triên trên nền công nghệ CD ROM là các đĩa cho phép ghi 1 lần, đọc dữ liệu nhiều lần WORM (writen once read many). Với đĩa WORM, tốc độ ghi dữ liệu thấp hơn nhiều so với tốc độ đọc dữ liệu, đồng thời các ổ đĩa cũng đắt hơn so với ổ CD ROM thông thường.

Digital Versatile Disc – DVD

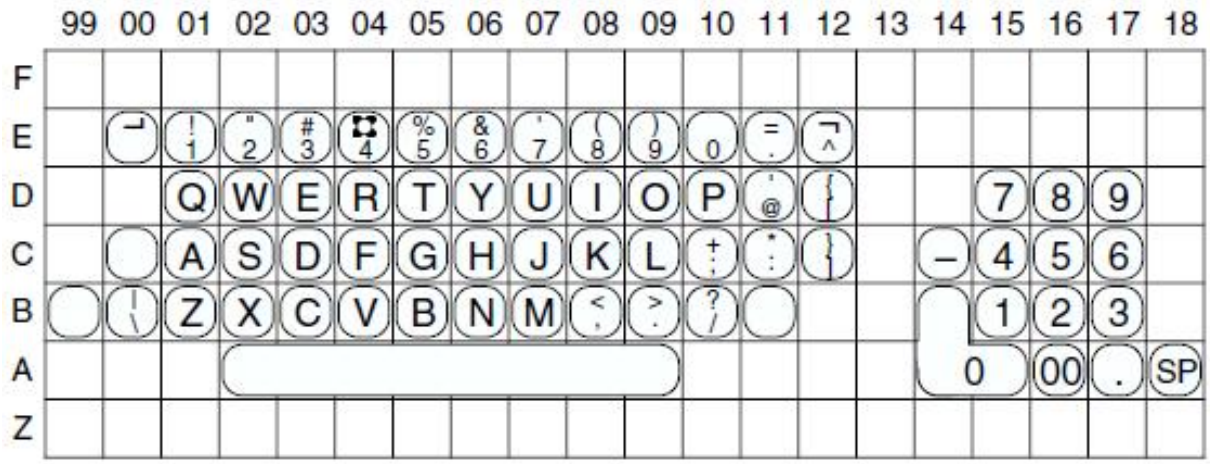
Một phiên bản mới của ổ đĩa quang là DVD. Với công nghệ này, có một số chuẩn công nghiệp dành cho các mục đích khác nhau như DVD-audio, DVD-Video, và DVD ROM, DVD RAM. Với một mặt đĩa được sử dụng, dung lượng lưu trữ sẽ có thể lên tới 4.7GB. Chuẩn DVD thông thường cũng có thể lưu trữ dữ liệu vào cả 2 mặt đĩa với tổng dung lượng lên tới 17GB. Công nghệ DVD là kế thừa từ CD nên trong thực tế, đầu đọc DVD có thể tương thích ngược với các đầu CD, và CD ROM thông thường.

8.2. Thiết bị nhập dữ liệu

Ổ đĩa lưu dữ liệu như trình bày trong mục 8.1 đều là các thiết bị vừa input vừa output và ứng dụng chủ yếu là để lưu trữ dữ liệu. Trong phần này, chúng ta sẽ tiếp tục tìm hiểu một số thiết bị được sử dụng để chuyên nhập dữ liệu input. Ta sẽ bắt đầu với bàn phím – keyboard.

8.2.1. Bàn phím

Bàn phím thường được sử dụng để nhập liệu bằng tay cho máy tính. Sơ đồ bàn phím theo chuẩn ECMA-23 được thể hiện trên hình 8.8. Sơ đồ "QWERTY" phù hợp với bàn phím nhập liệu chuẩn của máy đánh chữ. Các ký tự hay được sử

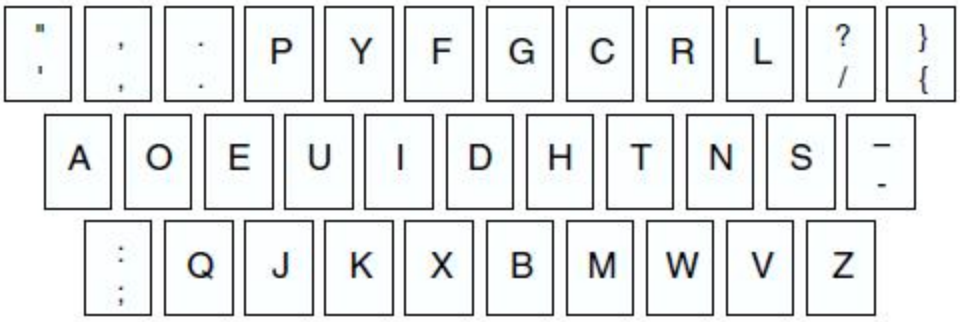


Hình 8.8. Sơ đồ bàn phím theo chuẩn ECMA-23

dụng được đặt ở xa để người đánh máy gõ với tốc độ chậm hơn, tránh gây nghẽn tín hiệu cơ khí. Mặc dù hiện tượng nghẽn cơ khí không xảy ra với các bàn phím điện tử nhưng sơ đồ bố trí bàn phím như trên mang tính chất lịch sử.

Khi một ký tự trên bàn phím được tác động, tín hiệu mẫu sẽ được tạo ra và chuyển về các cổng máy tính. Với các ký tự ASCII 7 bit, sẽ có 128 mẫu ký tự được sử dụng. Một số bàn phím khác được mở rộng từ chuẩn ECMA-23 có thêm các phím như shift, escape và control để mở rộng, do đó thường mã hóa 7 bit thường không đủ

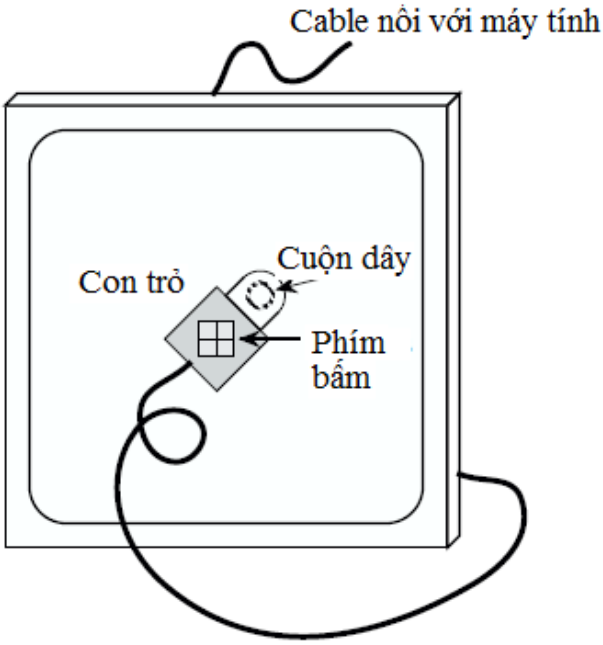
Có một số biến thể khác của bàn phím ECMA-23 là thêm các phím chức năng (ví dụ như các phím chức năng F), các phím đặc biệt như tab, delete,...Một trong số chúng là bàn phím Dvorak được mô tả ở hình 8.9. Mặc dù bàn phím Dvorak có nhiều ưu điểm nhưng thực tế nó lại không được chấp nhận



Hình 8.9. Sơ đồ bàn phím Dvorak

8.2.2. Bảng nhập liệu Bit pad

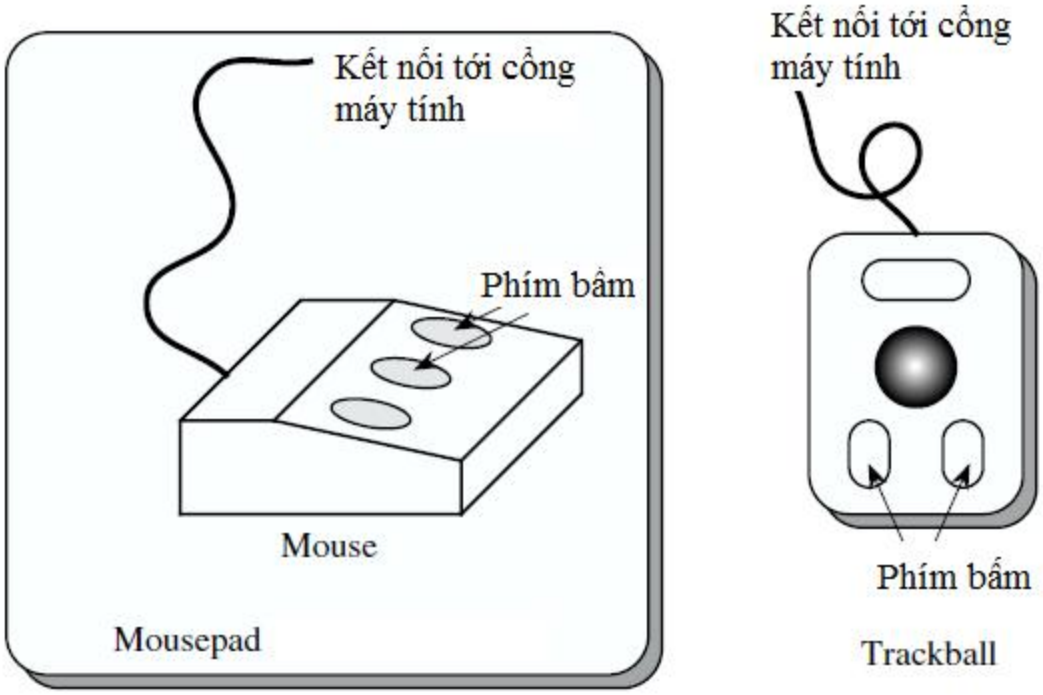
Bảng nhập liệu bit pad là một bảng kỹ thuật số có chức năng chuyên dụng là để nhập liệu. Bảng bit pad bao gồm một bút và một bảng phẳng như trên hình 8.10.



Hình 8.10. Bit pad

Bảng phẳng cảm ứng được cấu tạo từ một hệ thống ma trận 2 chiều các cuộn dây cảm ứng. Các cuộn dây này sẽ cảm ứng một dòng điện khi con trỏ bút di chuyển trên bảng. Các tín hiệu tọa độ X-Y của con trỏ cũng như trạng thái của các phím bấm đều là các tín hiệu tương tự liên tục sẽ được truyền về phần điều khiển, sau đó sẽ được truyền về máy tính. Ứng dụng chủ yếu của bit pad là trong trường hợp nhập các dữ liệu dạng đồ họa như bản đồ, ảnh, các biểu đồ cột hay biểu đồ graph.

8.2.3. Chuột và trackball



Hình 8.11. Chuột và trackball

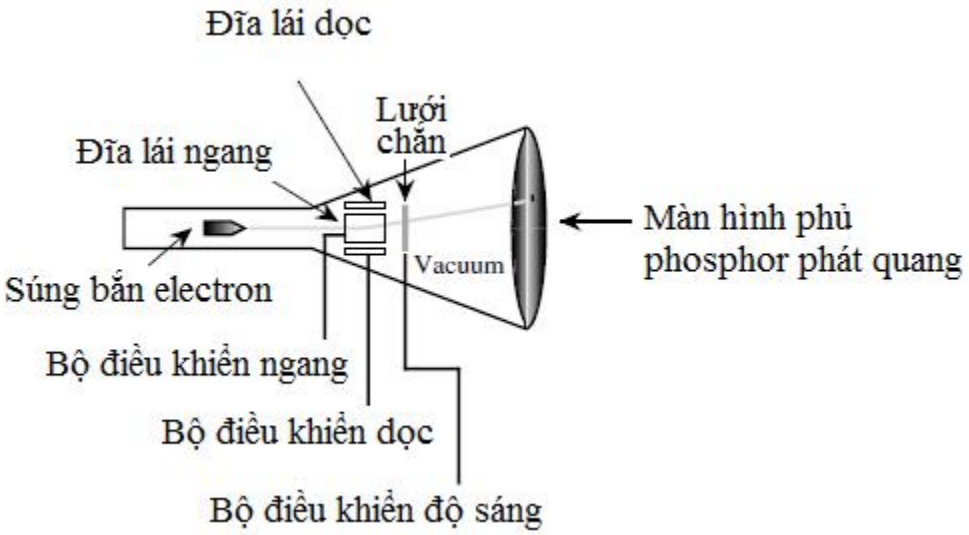
Chuột là một thiết bị nhập liệu cầm tay bao gồm một hay vài phím bấm ở mặt trên và có một quả bóng nhỏ bằng cao su ở mặt dưới. Hình 8.11 bên trái chỉ ra một dạng chuột cơ bản. Khi chuột di chuyển, quả bóng ở mặt dưới quay tương ứng với quãng đường dịch chuyển. Các chiết áp ở bên trong chuột sẽ cảm ứng chiều dịch chuyển và khoảng cách dịch chuyển, và sự dịch chuyển đó sẽ được ghi lại và chuyển về máy tính cùng với trạng thái của các phím bấm. Thông thường chuột có 2 phím trái và phải tương tác với máy tính với các chức năng khác nhau.

Trackball cũng có thể được coi là một dạng chuột bị quay ngược lên. Cấu trúc của trackball là cố định, còn quả bóng cao su sẽ được lăn bởi tay người sử dụng. Hình 8.11. bên phải nêu lên hình dạng cơ bản của một trackball. Nguyên lý hoạt động của nó giống hệt so với chuột.

Với sự phát triển của công nghệ, chuột quang dần thay thế chuột truyền thống. Chuột quang thay quả bóng cao su bởi các diode phát quang LED và sử dụng mousepad đặc biệt bao gồm các sọc có thể hấp thụ và phản xạ theo cả chiều dọc lẫn chiều ngang. Sự di chuyển sẽ được cảm nhận thông qua sự chuyển đổi giữa vùng hấp thụ và phản xạ. Chuột quang không bám bẩn như chuột truyền thống và có thể được sử dụng trong cả môi trường thẳng đứng hoặc trong môi trường không trọng lượng

8.3. Thiết bị hiển thị dữ liệu

Màn hình máy tính là thiết bị hiển thị dữ liệu thông dụng trong máy tính. Thông tin hiển thị có thể là chữ, số hoặc là đồ họa. Hiện nay có một số loại màn hình thông dụng là màn hình tia âm cực CRT (Cathode Ray Tube), màn hình tinh thể lỏng LCD (Liquid Crystal Display) và màn hình Plasma PD (Plasma Display).



Hình 8.12. Màn hình CRT

Cấu hình của màn hình CRT được thể hiện trên hình 8.12. Một súng tạo electron có chức năng tạo chùm electron bắn vào màn hình phủ chất phát quang phosphor. Vị trí phát sáng là vị trí chùm tia đập vào màn hình. Vị trí này được điều khiển bởi điện áp đặt vào các đĩa lái dọc và đĩa lái ngang. Các electron có

điện tích âm, một lưới mang điện tích dương được tạo ra để ngăn cản các electron đến được màn hình, làm cho độ sáng màn hình có thể thay đổi. Màu sắc trên màn hình được xác định bởi các đặc tính của phosphor. Đối với màn hình CRT mầu, thông thường sẽ có 3 loại phosphor (đỏ, lục, lam) được xen kẽ tại một khoảng vị trí trên bàn hình và sẽ tạo ra 3 mầu bởi 3 súng phóng tia electron khác nhau.

Để tạo ra một hình ảnh trọn vẹn trên màn hình, tín hiệu hình ảnh được phân tích và chuyển thành tín hiệu đưa vào súng phóng electron. Chùm tia này sẽ được quét theo chiều ngang từ trái qua phải tạo thành một vệt sáng ngang. Đến cuối dòng, nó được quét ngược trở lại bên trái để tiếp tục quét tiếp dòng thứ 2. Quá trình quét được thực hiện dịch chuyển dần từ trên xuống dưới cho đến cận dưới của màn hình tạo thành một hình ảnh, được gọi là một mành. Các mành được tạo ra nhiều lần trong 1 giây. Tốc độ lặp lại được gọi là tốc độ khung, hay tốc độ làm tươi. Tốc độ làm tươi thông thường khoảng 50-60Hz. Một số loại màn hình cao cấp còn có tốc độ quét lên tới 100Hz

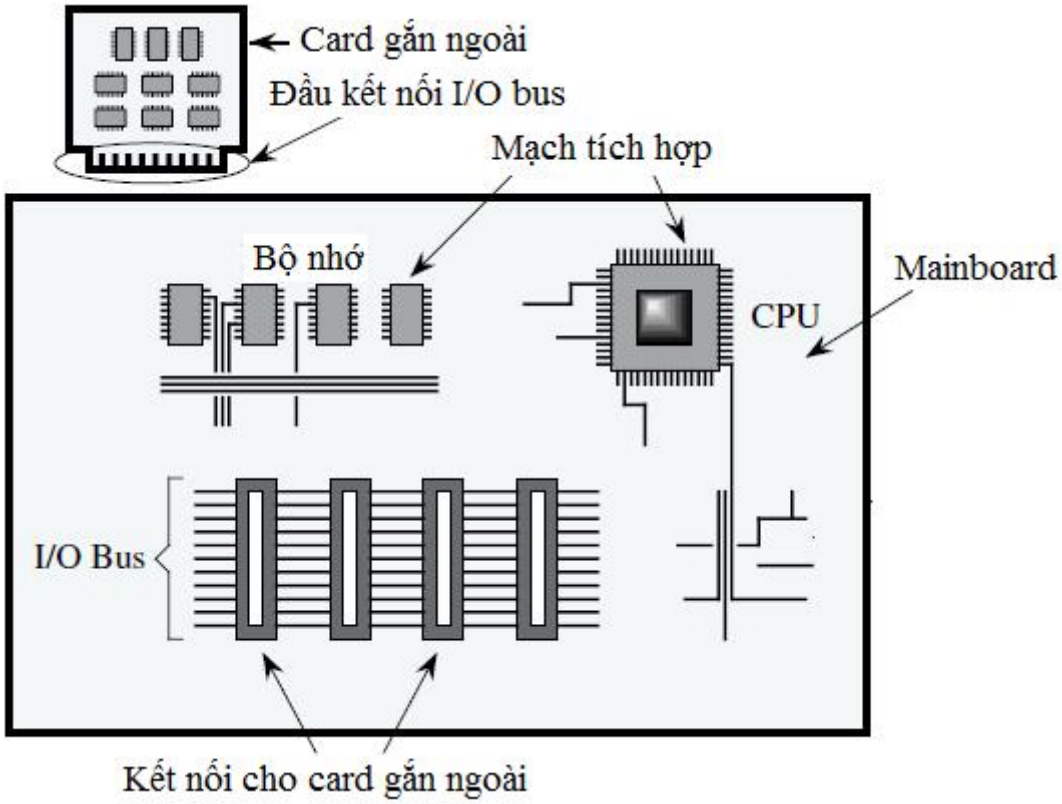
8.4. Kết nối truyền thông và ghép nối máy tính

Kết nối truyền thông là quá trình truyền thông tin từ một điểm nguồn đến một đích. Hệ thống kết nối truyền thông có phạm vi rất rộng, trải dài từ các kết nối trong hệ thống bus trong các mạch tích hợp cho đến các hệ thống điện thoại, radio và hệ thống truyền hình. Dữ liệu được truyền trong hệ thống truyền thông có thể là tín hiệu tương tự như tín hiệu tiếng nói, hoặc cũng có thể là tín hiệu số. Môi trương truyền tin cũng có thể là các loại cable đồng, cable quang, tín hiệu viba,... Trong phạm vi chương này, chúng ta sẽ tìm hiểu một số phương pháp truyền thông trong phạm vi nhỏ, ví dụ như quá trình tương tác dữ liệu giữa CPU và bộ nhớ chính hoặc tương tác giữa các thiết bị trong máy tính.

8.4.1. Kết nối bên trong máy tính

Không giống như các mạng viễn thông có nhiều bộ truyền tin và bộ nhận tin được đặt ở các vị trí địa lý rộng lớn khác nhau, máy tính chỉ có một số lượng cụ thể các thiết bị được nối với nhau trong một phạm vi có hạn, khoảng cách từ vài millimeter cho tới vài meter.Trong một hệ thống có N thiết bị cần kết nối với nhau. Với kịch bản xấu nhất, ta sẽ phải cần có $N^2/2$ đường kết nối. Tuy nhiên, rất may là không phải tất cả các thiết bị sẽ được kết nối như vậy.

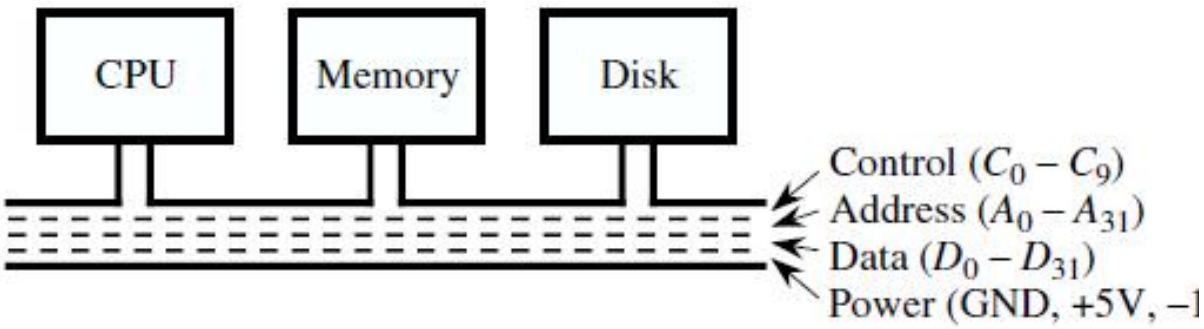
Bus là được định nghĩa là một hệ thống dây dẫn được sử dụng để kết nối một số lượng các thiết bị lại với nhau. Hình 8.13 là ví dụ hệ thống bus trên mainboard.



Hình 8.13. Mainboard nhìn từ trên xuống

Kết nối bus

Một hệ thống bus sẽ bao gồm nhiều thành phần như các bộ kết nối, các dây dẫn và giao thức truyền thông. Các dây dẫn có thể được phân loại thành từng nhóm: nhóm điều khiển, nhóm địa chỉ, nhóm dữ liệu và nhóm cung cấp nguồn như trên hình 8.14. Một hệ thống bus có thể có một vài đường dây cung cấp nguồn, thông thường đó là các tín hiệu nguồn đất GND, +5V và -15V

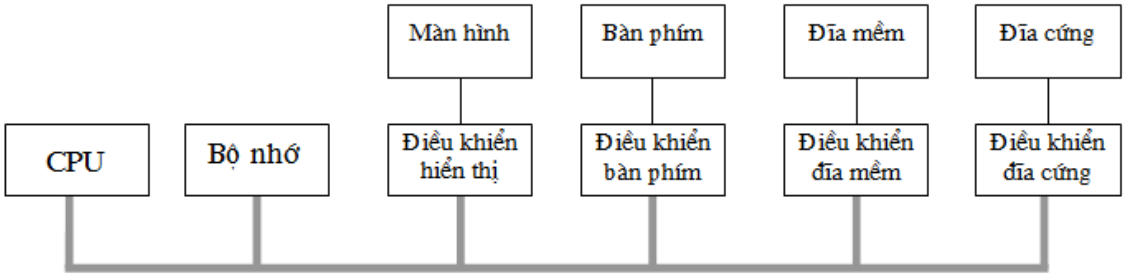


Hình 8.14. Hệ thống bus điển hình

Trong hệ thống truyền thông trong máy tính có thể có rất nhiều thiết bị. Tuy nhiên tại một thời điểm sẽ chỉ có 1 thiết bị có thể được truyền dữ liệu. Tất cả các thiết bị luôn luôn nghe ngóng đường truyền, nhưng sẽ chỉ có 1 thiết bị được phép nhận dữ liệu. Chỉ có 1 thiết bị được coi là thiết bị chủ bus master, còn tất cả các thiết bị còn lại được coi là thiết bị tớ slave. Thiết bị chủ sẽ điều khiển hệ thống bus và nó có thể là thiết bị truyền cũng như là thiết bị nhận.

Ưu điểm của kết nối bus là giảm thiểu được các kết nối ở tất cả các thiết bị với tất cả các thiết bị khác, giảm thiểu được các kết nối dây phức tạp do đó giảm được giá thành hệ thống. Nhược điểm của kết nối bus là tốc độ kết nối bị hạn chế, vấn đề truy cập, chia sẻ chung một hệ thống bus và khả năng mở rộng hệ thống

Hầu hết các máy tính PC có cùng một cách tổ chức hệ thống bus. Đơn giản là tất cả các thiết bị đều sử dụng chung một hệ thống bus được gọi là system bus như hình 8.15. Phương pháp tổ chức này tỏ ra đơn giản, tiết kiệm dây dẫn do đó

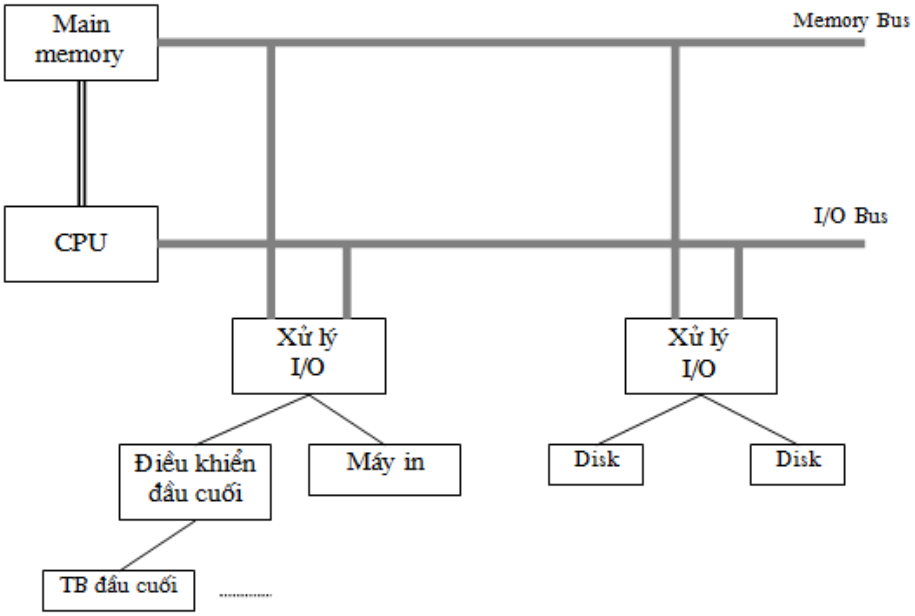


Hình 8.15. Tổ chức bus với máy tính PC thông thường

giảm thiểu chi phí hệ thống. Tuy nhiên, nhược điểm của nó là có thể xảy ra tình huống cả CPU và các thiết bị I/O cùng đòi chiếm dụng bus và sẽ xảy ra xung đột. Để giải quyết vấn đề này, ta sẽ phải sử dụng 1 chip chuyên dụng để phân xử bus. Vấn đề phân xử bus sẽ được đề cập ngay trong mục này.

Đối với hệ thống máy chủ, máy tính main frame đòi hỏi lượng dữ liệu trao đổi lớn, tổ chức bus được phân tách ra theo chức năng như trên hình 8.16. Hệ thống main frame có thể được trang bị một hay nhiều CPU, bộ nhớ chính và các bộ xử lý vào ra I/O. Trên máy tính loại này thường có 3 hệ thống bus: memory bus được sử dụng để các bộ xử lý I/O đọc và ghi dữ liệu với bộ nhớ chính mà không cần thông qua CPU; I/O bus cho phép CPU phát các lệnh đến tất cả các bộ xử lý I/O; và hệ thống bus tương tác trực tiếp giữa CPU với bộ nhớ chính. Với cách tổ chức như vậy, CPU có thể đồng thời trao đổi dữ liệu với các thiết bị I/O và trao

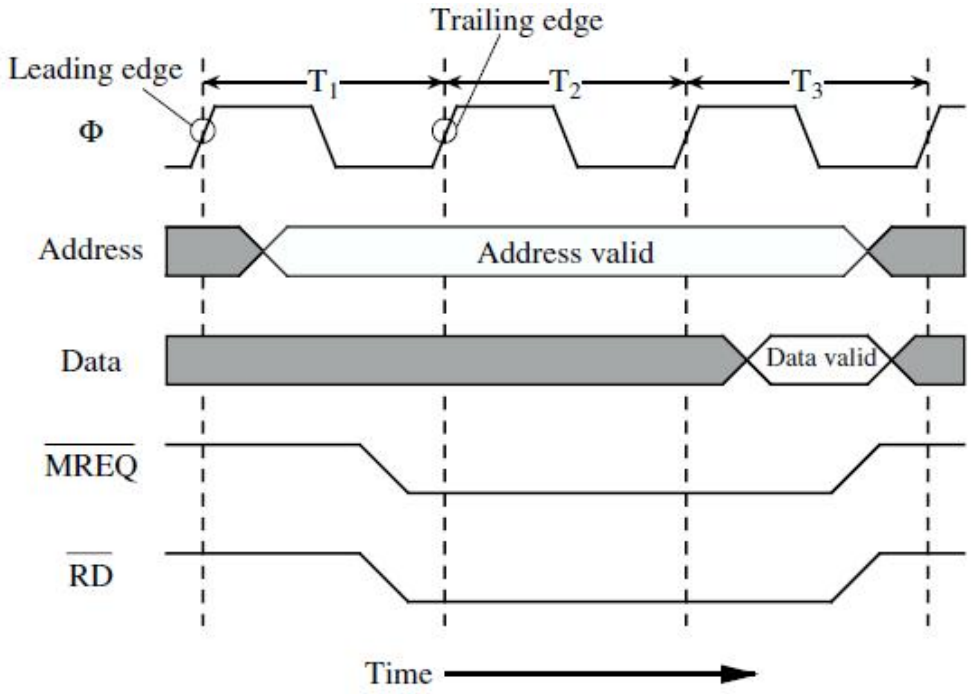
đổi dữ liệu với bộ nhớ chính (Tất nhiên là tại cùng một thời điểm không thể trao đổi với 2 thiết bị ngoại vi).



Hình 8.16. Tổ chức bus với máy tính main frame

Bus đồng bộ

Bus đồng bộ là hệ thống bus hoạt động dưới sự đồng bộ của một tín hiệu xung nhịp đồng hồ. Ví dụ như quá trình đọc dữ liệu từ bộ nhớ về CPU được thể hiện trên hình 8.17 dưới đây



Hình 8.17. Hoạt động của bus đồng bộ

Tại chu kỳ đầu tiên T_1, khi xung nhịp đồng hồ còn ở mức cao, CPU đặt địa chỉ của ô nhớ cần đọc dữ liệu lên bus địa chỉ. Ngay sau đó trong khoảng T_1, sau khi tín hiệu địa chỉ đã ổn định, tín hiệu $\overline{MREQ}$ và $\overline{RD}$ sẽ được kích hoạt. Tín hiệu $\overline{MREQ}$ là tín hiệu chỉ ra rằng CPU sẽ tương tác với bộ nhớ. Tín hiệu $\overline{RD}$ là tín hiệu mà CPU chỉ ra đây là quá trình đọc dữ liệu về CPU. Cả hai tín hiệu $\overline{MREQ}$ và $\overline{RD}$ đều tích cực ở mức điện áp 0V. Quá trình đọc dữ liệu thường diễn ra chậm hơn so với tốc độ bus, do đó toàn bộ thời gian T_2 được dành cho việc truy cập bộ nhớ. Tại chu kỳ T_3, CPU sẽ đọc dữ liệu truyền về trên đường data bus. Ngay sau khi hoàn thành quá trình đọc dữ liệu, CPU sẽ giải phóng tín hiệu $\overline{MREQ}$ và $\overline{RD}$.

Bus không đồng bộ

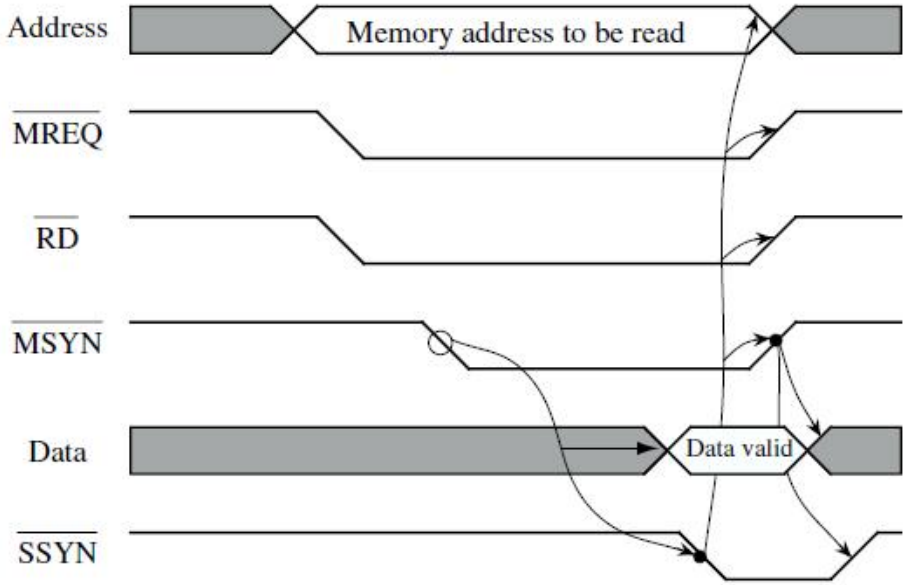
Nếu ta đặt bộ nhớ đồng bộ vào một hệ thống bus đồng bộ có tốc độ cao hơn tốc độ của bộ nhớ, khi đó thời gian truy cập vẫn không được cải thiện bởi vì tốc độc xung nhịp là không đổi. Nếu ta tăng tốc độ xung nhịp để phù hợp với tốc độ cao của bộ nhớ, các thiết bị có tốc độ thấp hơn sẽ không thể hoạt động bình thường.

Bus không đồng bộ được sử dụng để giải quyết vấn đề trên, tất nhiên là cách giải quyết của nó tương đối phức tạp vì nó không có xung nhịp đồng bộ. Một thiết bị chủ master sẽ đặt tất cả mọi thứ mà nó cần lên hệ thống bus (địa chỉ, dữ liệu, tín hiệu điều khiển), sau đó kích hoạt tín hiệu đồng bộ chủ $\overline{MSYN}$ (master synchronization). Thiết bị tớ sẽ đáp ứng với tốc độ nhanh nhất có thể rồi kích hoạt tín hiệu đồng bộ tớ $\overline{SSYN}$ (slave synchronization) khi kết thúc quá trình truyền nhận. Tiếp đó, thiết bị chủ sẽ giải phóng $\overline{MSYN}$ rồi thiết bị tớ giải phóng $\overline{SSYN}$. Bằng cách này, hệ thống chủ/tớ hoạt động với tốc độ cao sẽ hoạt động hiệu quả hơn so với hệ thống chủ/tớ hoạt động với tốc độ thấp.

Để tìm hiểu rõ hơn về quá trình trao đổi dữ liệu không đồng bộ, ta sẽ xem xét quá trình đọc dữ liệu không đồng bộ được miêu tả trên hình 8.16. Để đọc được dữ liệu trong bộ nhớ, CPU đặt tín hiệu địa chỉ lên bus địa chỉ, sau đó là kích hoạt tín hiệu $\overline{MREQ}$ và $\overline{RD}$. Khi các tín hiệu này ổn định, CPU sẽ kích hoạt $\overline{MSYN}$. Sự kiện này kích hoạt bộ nhớ thực hiện hoạt động đọc. Quá trình đọc bộ nhớ sẽ kết thúc khi tín hiệu $\overline{SSYN}$ được kích hoạt bởi bộ nhớ. Ta cũng có thể nhìn rõ quá trình này thông qua các mũi tên có trên hình 8.18. Phương pháp trao

đổi dữ liệu như trên còn được gọi là **full handshake**. Lưu ý rằng quá trình trao đổi dữ liệu không đồng bộ không sử dụng tín hiệu xung nhịp clock.

Khi phát sinh lỗi hệ thống, hệ thống bus không đồng bộ tỏ ra rất khó phân tích, kiểm tra lỗi so với hệ thống bus đồng bộ. Chính vì vậy mà trong máy tính PC, chủ yếu ta thường sử dụng bus đồng bộ.

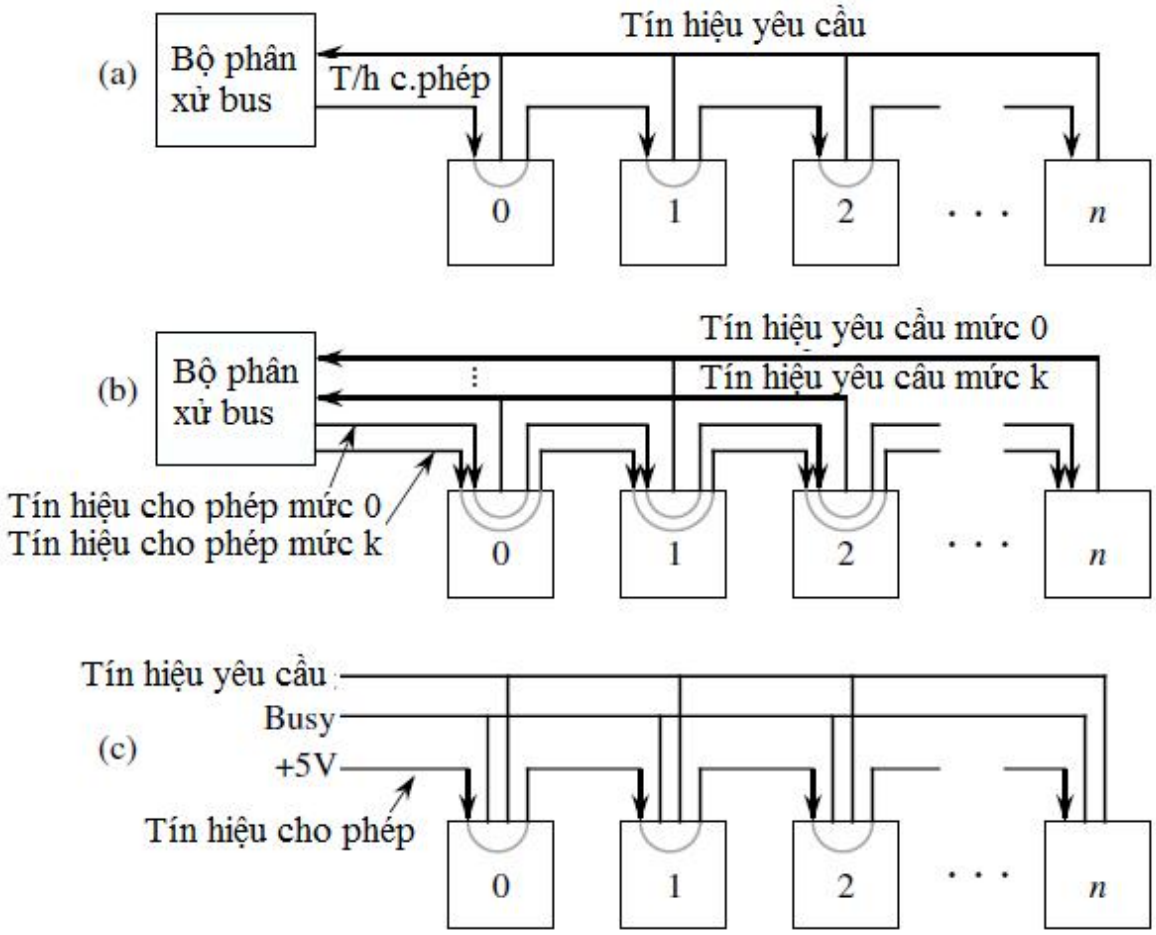


Hình 8.18. Hoạt động của bus không đồng bộ

Phân xử bus

Bây giờ hãy giả sử là tại một thời điểm có nhiều hơn một thiết bị cần truy cập bus. Câu hỏi đặt ra là thiết bị nào sẽ được quyết định làm master ? Vấn đề phân xử bus có thể được giải quyết bởi một trong hai phương pháp cơ bản: phân xử bus tập trung và phân xử bus không tập trung. Hình 8.19 mô tả hai phương pháp phân xử bus. Phân xử bus tập trung được mô tả trong hình 8.19a. Thiết bị 0 đến thiết bị n cùng được nối vào một hệ thống bus (hệ thống bus này không được mô tả trong hình) và có cùng một đường dây tín hiệu bus request nối với bộ phân xử. Khi một thiết bị muốn làm master để thực hiện trao đổi dữ liệu, nó kích hoạt tín hiệu yêu cầu. Khi bộ phân xử nhận thấy có tín hiệu yêu cầu, nó sẽ quyết định xem tín hiệu cho phép có được cung cấp hay không (trong một số trường hợp, hệ thống bus không được phép gián đoạn). Nếu tín hiệu phân xử được cung cấp, nó sẽ kích hoạt tín hiệu cho phép. Tín hiệu cho phép được nối nối tiếp từ thiết bị này đến thiết bị khác. Thiết bị đầu tiên nhận được tín hiệu cho phép và nó cũng đồng thời muốn trở thành bus master sẽ nắm quyền truy cập bus và ngăn cấm các thiết bị ở phía sau nó truy cập bus. Nếu thiết bị không

muốn truy cập bus, nó đơn giản là chuyển tín hiệu cho phép ra thiết bị phía sau. Thiết bị nào ở gần bộ phân xử sẽ có mức ưu tiên cao hơn so với các thiết bị phía sau.



Hình 8.19. Phân xử bus

Trong một số trường hợp, việc phân xử bus có 1 mức ưu tiên như vậy tỏ ra không thích hợp. Khi đó, ta có thể thực hiện phân xử bus có nhiều mức ưu tiên như hình 8.19b. Tín hiệu yêu cầu với số hiệu nhỏ hơn có mức ưu tiên cao hơn. Do vậy, để xác định mức ưu tiên cao hơn cho một thiết bị, nó sẽ được gắn với tín hiệu yêu cầu có số hiệu nhỏ. Các thiết bị có cùng mức ưu tiên sẽ được nhóm thành từng nhóm và đưa vào cùng một bộ phân xử bus.

Phân xử bus không tập trung được thể hiện trên hình 8.19c. Lưu ý rằng hệ thống sẽ không có bộ phân xử. Tín hiệu cho phép luôn được cung cấp vào hệ thống các thiết bị. Khi một thiết bị muốn truy cập bus, nó sẽ kích hoạt tín hiệu yêu cầu lên đường bus request rồi kiểm tra tín hiệu trên đường dây busy. Nếu tín hiệu trên đường busy là không tích cực, nó sẽ chuyển tín hiệu 0 đến các thiết bị được đánh số cao hơn trong dãy kết nối (tức là ngắt tín hiệu cho phép đến các thiết bị

phía sau), tích cực đường tín hiệu busy và giải phóng tín hiệu yêu cầu. Nếu tín hiệu busy là bận, hoặc nếu thiết bị không có nhu cầu truy cập bus, nó chỉ cần đơn giản là chuyển tín hiệu cho phép đến các thiết bị phía sau.

Phân xử bus cần hoạt động với tốc độ cao, do đó phân xử bus tập trung chỉ thích hợp với hệ thống có số lượng thiết bị nhỏ (thường tối đa là 8 thiết bị). Trong trường hợp số lượng thiết bị lớn, phân xử bus không tập trung tỏ ra thích hợp hơn

8.4.2. Một số chuẩn bus trong máy tính

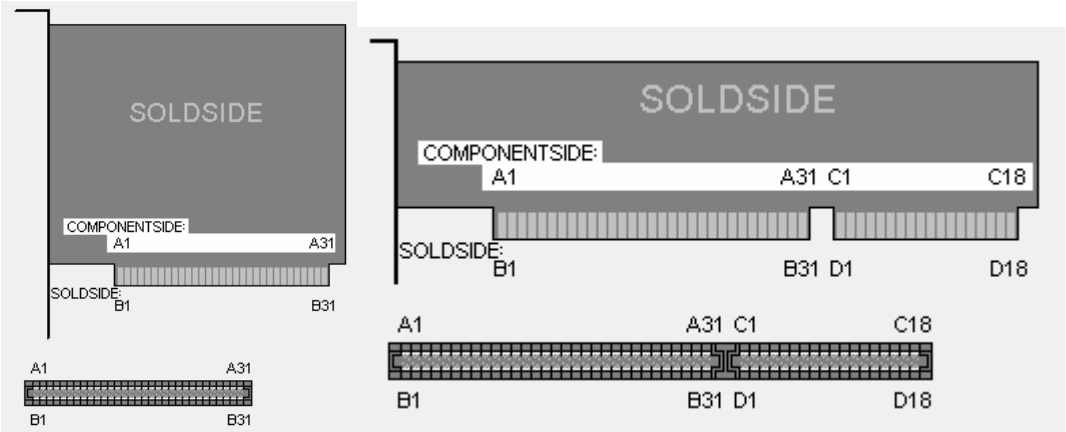
Trong mục 8.4.1. ở trên, ta đã tìm hiểu một số kiến thức cơ bản về hệ thống bus trong máy tính. Trong phần này, chúng ta sẽ tìm hiểu cụ thể hơn các chuẩn bus có trong một máy tính.

Chuẩn ISA

Ra đời vào năm 1981 trong một dự án của IBM dành cho các máy tính 16/8 bit. Ban đầu chuẩn ISA được xây dựng là hệ thống 8 bit. Theo nhu cầu phát triển, chuẩn ISA 16 bit được giới thiệu vào năm 1984. Tốc độ xung nhịp hoạt động trong ISA 8 bit thường là 4,77MHz còn đối với chuẩn ISA 16 bit là 6MHz rồi nhanh chóng tăng tốc độ lên 8MHz.

Chuẩn ISA thường được sử dụng để kết nối các card ngoại vi với mainboard như card đồ họa hoặc card âm thanh. Một số modem trong thời gian đầu cũng được kết nối với máy tính thông qua khe cắm ISA này. Số lượng thiết bị tối đa kết nối với hệ thống là 6 thiết bị

Hình 8.20 mô tả cấu trúc các chân tín hiệu của ISA 8 bit và cấu trúc ISA 16 bit



Hình 8.20. Cấu trúc ISA 8 bit và ISA 16 bit

Các tín hiệu của chuẩn ISA

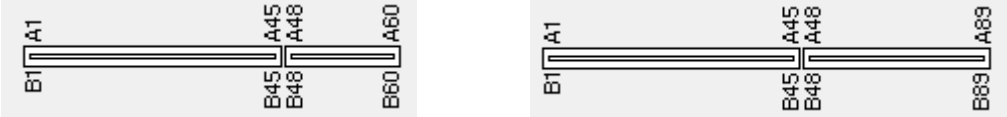
Pin	Name	Dir	Description
A1	/I/O CH CK	←	I/O channel check; active low=parity error
A2	D7	↔	Data bit 7
A3	D6	↔	Data bit 6
A4	D5	↔	Data bit 5
A5	D4	↔	Data bit 4
A6	D3	↔	Data bit 3
A7	D2	↔	Data bit 2
A8	D1	↔	Data bit 1
A9	D0	↔	Data bit 0
A10	I/O CH RDY	←	I/O Channel ready, pulled low to lengthen memory cycles
A11	AEN	→	Address enable; active high when DMA controls bus
A12	A19	→	Address bit 19
A13	A18	→	Address bit 18
A14	A17	→	Address bit 17
A15	A16	→	Address bit 16
A16	A15	→	Address bit 15
A17	A14	→	Address bit 14
A18	A13	→	Address bit 13
A19	A12	→	Address bit 12
A20	A11	→	Address bit 11
A21	A10	→	Address bit 10
A22	A9	→	Address bit 9
A23	A8	→	Address bit 8
A24	A7	→	Address bit 7
A25	A6	→	Address bit 6
A26	A5	→	Address bit 5
A27	A4	→	Address bit 4
A28	A3	→	Address bit 3
A29	A2	→	Address bit 2
A30	A1	→	Address bit 1
A31	A0	→	Address bit 0
B1	GND		Ground
B2	RESET	→	Active high to reset or initialize system logic
B3	+5V		+5 VDC
B4	IRQ2	←	Interrupt Request 2

B5	-5VDC		-5 VDC
B6	DRQ2	←	DMA Request 2
B7	-12VDC		-12 VDC
B8	/NOWS	←	No WaitState
B9	+12VDC		+12 VDC
B10	GND		Ground
B11	/SMEMW	→	System Memory Write
B12	/SMEMR	→	System Memory Read
B13	/IOW	→	I/O Write
B14	/IOR	→	I/O Read
B15	/DACK3	→	DMA Acknowledge 3
B16	DRQ3	←	DMA Request 3
B17	/DACK1	→	DMA Acknowledge 1
B18	DRQ1	←	DMA Request 1
B19	/REFRESH	↔	Refresh
B20	CLOCK	→	System Clock (67 ns, 8-8.33 MHz, 50% duty cycle)
B21	IRQ7	←	Interrupt Request 7
B22	IRQ6	←	Interrupt Request 6
B23	IRQ5	←	Interrupt Request 5
B24	IRQ4	←	Interrupt Request 4
B25	IRQ3	←	Interrupt Request 3
B26	/DACK2	→	DMA Acknowledge 2
B27	T/C	→	Terminal count; pulses high when DMA term. count reached
B28	ALE	→	Address Latch Enable
B29	+5V		+5 VDC
B30	OSC	→	High-speed Clock (70 ns, 14.31818 MHz, 50% duty cycle)
B31	GND		Ground
C1	SBHE	↔	System bus high enable (data available on SD8-15)
C2	LA23	↔	Address bit 23
C3	LA22	↔	Address bit 22
C4	LA21	↔	Address bit 21
C5	LA20	↔	Address bit 20
C6	LA18	↔	Address bit 19
C7	LA17	↔	Address bit 18
C8	LA16	↔	Address bit 17
C9	/MEMR	↔	Memory Read (Active on all memory read cycles)

C10	/MEMW	↔	Memory Write (Active on all memory write cycles)
C11	SD08	↔	Data bit 8
C12	SD09	↔	Data bit 9
C13	SD10	↔	Data bit 10
C14	SD11	↔	Data bit 11
C15	SD12	↔	Data bit 12
C16	SD13	↔	Data bit 13
C17	SD14	↔	Data bit 14
C18	SD15	↔	Data bit 15
D1	/MEMCS16	←	Memory 16-bit chip select (1 wait, 16-bit memory cycle)
D2	/IOCS16	←	I/O 16-bit chip select (1 wait, 16-bit I/O cycle)
D3	IRQ10	←	Interrupt Request 10
D4	IRQ11	←	Interrupt Request 11
D5	IRQ12	←	Interrupt Request 12
D6	IRQ15	←	Interrupt Request 15
D7	IRQ14	←	Interrupt Request 14
D8	/DACK0	→	DMA Acknowledge 0
D9	DRQ0	←	DMA Request 0
D10	/DACK5	→	DMA Acknowledge 5
D11	DRQ5	←	DMA Request 5
D12	/DACK6	→	DMA Acknowledge 6
D13	DRQ6	←	DMA Request 6
D14	/DACK7	→	DMA Acknowledge 7
D15	DRQ7	←	DMA Request 7
D16	+5 V		
D17	/MASTER	←	Used with DRQ to gain control of system
D18	GND		Ground

Chuẩn MCA

MCA – Micro Channel Architecture là chuẩn kết nối do IBM phát triển từ năm 1987 cho các máy tính PS/2. Tồn tại một thời gian khá dài, hiện nay chuẩn MCA có thể được tìm thấy trong một số máy mainframe như PS/2, RS/6000 hay Á/400 hoặc System/370. Chuẩn MCA cũng có dạng 16 bit và cả dạng 32 bit. Sơ đồ chân của chuẩn này được thể hiện trên hình 8.21. dưới đây



Hình 8.21. Sơ đồ chân chuẩn MCA 16 và 32 bit

Pin	Signal Name	Pin	Signal Name
B01	AUDIO GND	A01	-CD SETUP
B02	AUDIO	A02	MADE 24
B03	GND	A03	GND
B04	14.3 MHz OSZ	A04	A11
B05	GND	A05	A10
B06	A23	A06	A09
B07	A22	A07	+5 Vdc
B08	A21	A08	A08
B09	GND	A09	A07
B10	A20	A10	A06
B11	A19	A11	+5 Vdc
B12	A18	A12	A05
B13	GND	A13	A04
B14	A17	A14	A03
B15	A16	A15	+5 Vdc
B16	A15	A16	A02
B17	GND	A17	A01
B18	A14	A18	A00
B19	A13	A19	+12 Vdc
B20	A12	A20	-ADL
B21	GND	A21	-PREEMPT
B22	-IRQ 09	A22	-BURST
B23	-IRQ 03	A23	-12 Vdc
B24	-IRQ 04	A24	ARB 00
B25	GND	A25	-ARB 01
B26	-IRQ 05	A26	ARB 02
B27	-IRQ 06	A27	-12 Vdc
B28	-IRQ 07	A28	ARB 03
B29	GND	A29	ARB/-GNT
B30	-DPAREN	A30	-TC
B31	DPAR(0)	A31	+5 Vdc
B32	-CHCK	A32	-SO
B33	GND	A33	-S1
B34	-CMD	A34	M/-IO
B35	CHRDYRTN	A35	+12 Vdc
B36	-CD SFD	A36	CD CHRDY
B37	GND	A37	D 0
B38	D 01	A38	D 02
B39	D 03	A39	+5 Vdc
B40	D 04	A40	D 05

Pin	Signal Name	Pin	Signal Name
B47	- key -	A47	- key -
B48	D 08	A48	+5 Vdc
B49	D 09	A49	D 10
B50	GND	A50	D 11
B51	D 12	A51	D 13
B52	D 14	A52	+12 Vdc
B53	D 15	A53	DPAR(1)
B54	GND	A54	-SBHE
B55	-IRQ 10	A55	-CD DS 16
B56	-IRQ 11	A56	+5 Vdc
B57	-IRQ 12	A57	-IRQ 14
B58	GND	A58	-IRQ 15
B59	reserved	A59	reserved
B60	reserved	A60	reserved
Pin	32-bit bus Signal Name	Pin	32-bit bus Signal Name
B61	-SDR(1)	A61	GND
B62	-MSDR	A62	reserved
B63	GND	A63	reserved
B64	D 16	A64	-SFDBKRTN
B65	D 17	A65	+12 Vdc
B66	D 18	A66	D 19
B67	GND	A67	D 20
B68	D 22	A68	D 21
B69	D 23	A69	+5 Vdc
B70	DPAR(2)	A70	D 24
B71	GND	A71	D 25
B72	D 27	A72	D 26
B73	D 28	A73	+5 Vdc
B74	D 29	A74	D 30
B75	GND	A75	D 31
B76	-BE 0	A76	DPAR(3)
B77	-BE 1	A77	+12 Vdc
B78	-BE 2	A78	-BE 3
B79	GND	A79	-DS 32 RTN
B80	TR 32	A80	-CD DS 32
B81	A 24	A81	+5 Vdc
B82	A 25	A82	A 26
B83	GND	A83	A 27
B84	A 29	A84	A 28
B85	A 30	A85	+5 Vdc

B41	GND	A41	D 06
B42	CHRESET	A42	D 07
B43	-SD STROBE	A43	GND
B44	-SDR (0)	A44	-DS 16 RTN
B45	GND	A45	-REFRESH
B46	- key -	A46	- key -

B86	A 31	A86	-APAREN
B87	GND	A87	-APAR(0)
B88	APAR(2)	A88	-APAR(1)
B89	APAR(3)	A89	GND

Bus EISA (Extended ISA)

Chuẩn này ra đời nhằm mở rộng cấu trúc ISA để hỗ trợ ngoại vi 32 bit với tốc độ truyền dữ liệu cao hơn. Do cạnh tranh với MCA, khe cắm có 2 nấc : nấc trên có tiếp điểm hoàn toàn tương thích với ISA nên có thể sử dụng card ISA để cắm vào khe cắm đó. Nấc dưới cho phép cắm các card EISA 32 bit. Chuẩn EISA có tốc độ xung nhịp 8,33 MHz nhưng do làm việc với số liệu 32 bit nên tốc độ truyền tải tối đa ở đây lên tới 33,32 MB/s và bus có thể địa chỉ hóa được bộ nhớ có dung lượng 4 GB. Hình 8.22. mô tả sơ đồ chân của EISA

Bus cục bộ VL – VESA Local

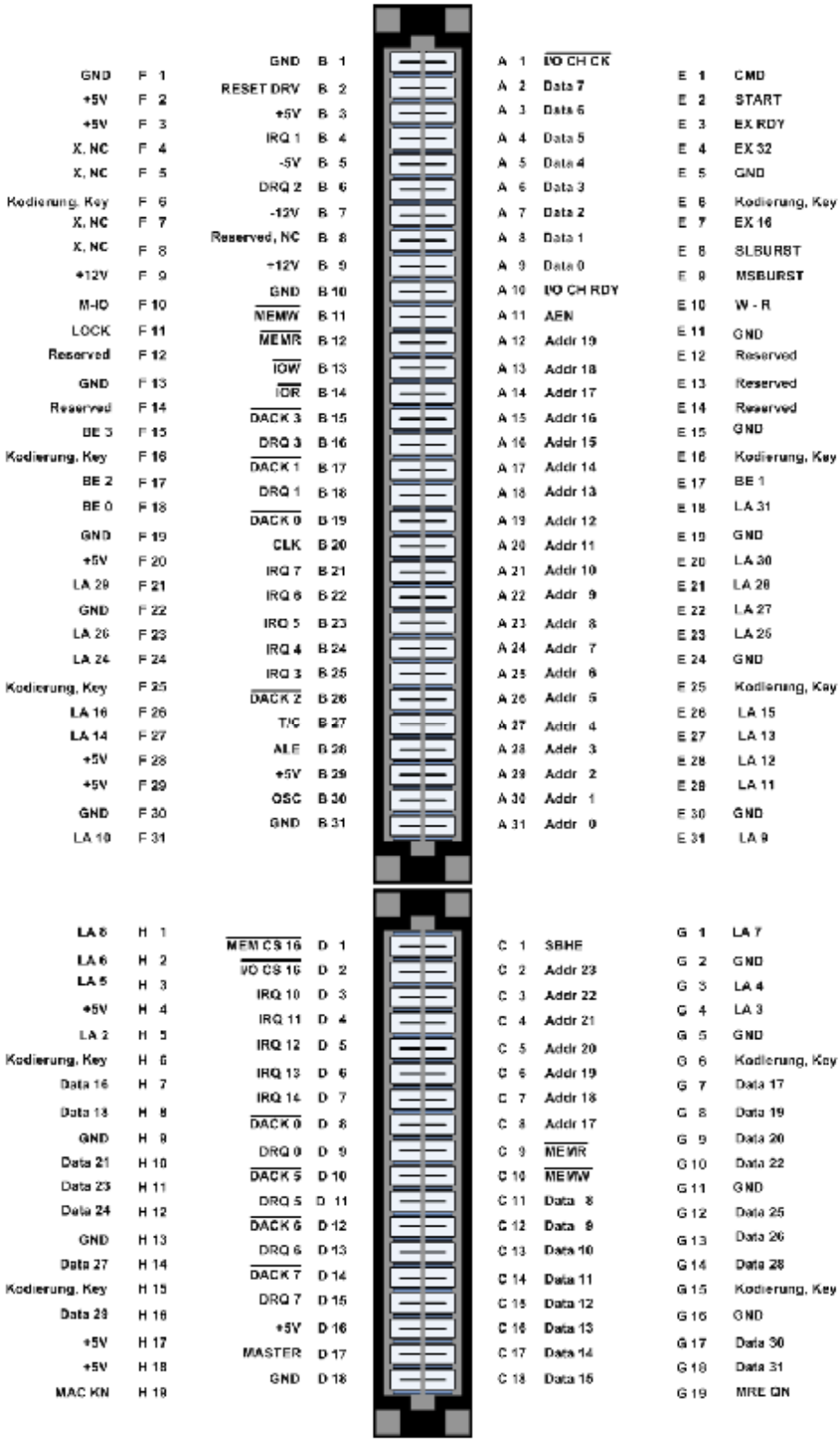
Nhược điểm chính của các bus ISA, EISA là gặp phải hiệu ứng “nghẹt cổ chai” làm lãng phí thời gian xử lý của các vi xử lý tốc độ cao hơn khi làm việc với các bus tốc độ thấp, như vậy làm giảm hiệu xuất của toàn bộ hệ thống. Do vậy, hiệp hội VESA (Video Electronic Standard Association) đã phát triển bus cục bộ VESA cho phép các card ghép nối từ vi xử lý tới bản mạch video tốc độ nhanh có thể hoạt động được. Bus VL hỗ trợ cả các thiết bị 32 lẫn 64 bit với tần số xung nhịp lên tới 50MHz nhưng chỉ tần số 33 MHz với thiết bị 32 bit là tối ưu với băng thông cực đại là 107MB/s. Vì lý do đó, chuẩn này trở nên hạn chế khi tốc độ các bus cục bộ quá 33MHz hoặc trên 100MHz. Hình 8.23. mô tả sơ đồ chân của VL bus

PCI bus (Peripheral Component Interconection)

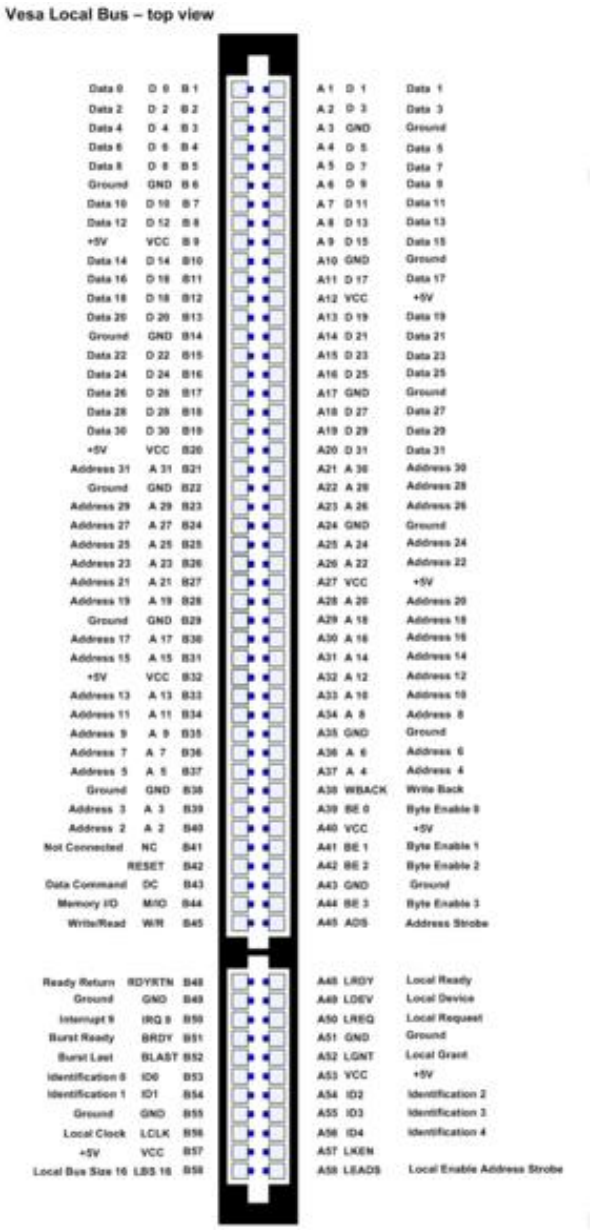
Chuẩn liên kết nối các thành phần ngoại vi PCI là chuẩn bus vào/ra được sử dụng phổ biến nhất hiện nay. Bus PCI cung cấp một đường dữ liệu chia sẻ ngay giữa vi xử lý và các bộ điều khiển ngoại vi trong tất cả các máy tính xách tay đến máy tính lớn

Bus PCI ra đời năm 1993, được thiết kế bởi công ty Intel, Compaq và Digital.

32 Bit EISA Bus – top view



Hình 8.22. Sơ đồ chân của EISA



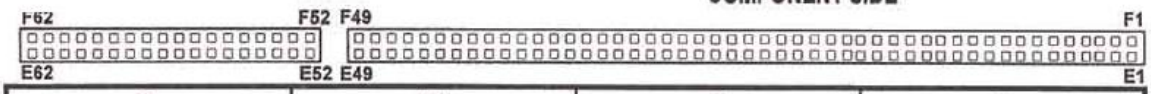
Hình 8.23. VESA Local bus

Bus PCI cùng tồn tại trên mainboard cùng với bus ISA trong nhiều năm. Do yêu cầu truyền tốc độ cao và giảm kích thước các bản mạch ghép nối PCI nên các khe cắm PCI không thể tương thích với các khe cắm ISA hay EISA. Tần số làm việc của PCI là 33MHz hay 66MHz, hỗ trợ các đường truyền dữ liệu 32 hay 64 bit. Trên bản mạch chính ngày nay tồn tại từ 3 đến 5 khe PCI kết nối với bộ vi xử lý qua một chip đặc biệt gọi là cầu PCI. Hình 8.24 mô tả sơ đồ chân của khe cắm bus PCI.

Ứng dụng chủ yếu của khe cắm PCI là để cắm các card xử lý âm thanh, đồ họa, mạng kết nối LAN hay các modem gắn trong.

PCI BUS PIN

COMPONENT SIDE



F		E		F		E	
PIN	ASSIGNMENT	PIN	ASSIGNMENT	PIN	ASSIGNMENT	PIN	ASSIGNMENT
F1	-12V	E1	TRST#	F31	+3.3V	E31	AD18
F2	TCK	E2	+12V	F32	AD17	E32	AD16
F3	GND	E3	TMS	F33	C/BE2#	E33	+3.3V
F4	TDO	E4	TDI	F34	GND	E34	FRAME#
F5	+5V	E5	+5V	F35	IRDY#	E35	GND
F6	+5V	E6	INTA#	F36	+3.3V	E36	TRDY#
F7	INTB#	E7	INTC#	F37	DEVSEL#	E37	GND
F8	INTD#	E8	+5V	F38	GND	E38	STOP#
F9	REQ3#	E9	CLKC	F39	LOCK#	E39	+3.3V
F10	REQ1#	E10	+5V(I/O)	F40	PERR#	E40	SDONE
F11	GNT3#	E11	CLKD	F41	+3.3V	E41	SB0#
F12	GND	E12	GND	F42	SERR#	E42	GND
F13	GND	E13	GND	F43	+3.3V	E43	PAR
F14	CLKA	E14	GNT1#	F44	C/BE1#	E44	AD15
F15	GND	E15	RST#	F45	AD14	E45	+3.3V
F16	CLKB	E16	+5V(I/O)	F46	GND	E46	AD13
F17	GND	E17	GNT0#	F47	AD12	E47	AD11
F18	REQ0#	E18	GND	F48	AD10	E48	GND
F19	+5V(I/O)	E19	REQ2#	F49	GND	E49	AD09
F20	AD31	E20	AD30	F52	AD08	E52	C/BE0#
F21	AD29	E21	+3.3V	F53	AD07	E53	+3.3V
F22	GND	E22	AD28	F54	+3.3V	E54	AD06
F23	AD27	E23	AD26	F55	AD05	E55	AD04
F24	AD25	E24	GND	F56	AD03	E56	GND
F25	+3.3V	E25	AD24	F57	GND	E57	AD02
F26	C/BE3#	E26	GNT2#	F58	AD01	E58	AD00
F27	AD23	E27	+3.3V	F59	+5V(I/O)	E59	+5V(I/O)
F28	GND	E28	AD22	F60	ACK64#	E60	REQ64#
F29	AD21	E29	AD20	F61	+5V	E61	+5V
F30	AD19	E30	GND	F62	+5V	E62	+5V

Hình 8.24. PCI bus

AGP bus

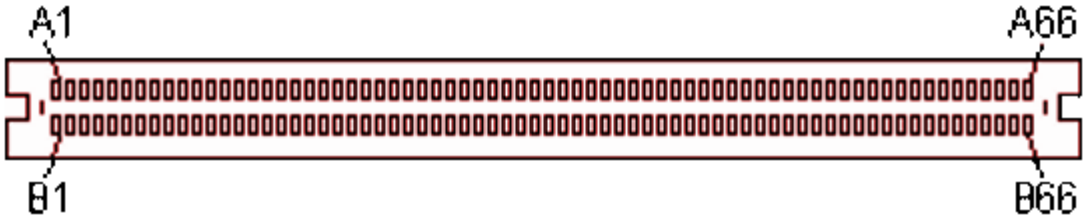
Khe cắm tăng tốc độ hiển thị đồ họa AGP có cấu trúc vật lý giống PCI. Nó làm việc tương tự PCI và được coi là một thiết bị PCI nhưng tốc độ nhanh gấp đôi PCI. Card AGP có khả năng truy cập trực tiếp bộ nhớ và bus AGP phù hợp cho việc hiển thị 3 chiều vì nó dùng bộ nhớ chính để lưu trữ dữ liệu vê bóng, trục z, nguồn sáng.

AGP có một số loại được phân loại theo băng thông

AGP	Độ rộng bus	Tần số làm việc	Số dữ liệu truyền 1 xung nhịp	Băng thông
1x	32 bit	66MHz	1	266 MBps
2x	32 bit	66MHz	2	533 MBps
4x	32 bit	66MHz	4	1066 MBps
8x	32 bit	66MHz	8	2133 MBps

Điện áp làm việc của AGP là 1.5V hoặc 3.3V tùy loại. Sơ đồ chân được thể hiện trên hình 8.25

Graphics Card Types	Connector Type*	Description
AGP 3.3V Card	3.3V slot	Supports only 3.3V signaling. Available speeds 1x, 2x.
AGP 1.5V Card	1.5V slot	Supports only 1.5V signaling. Available speeds 1x, 2x, 4x.
Universal AGP Card	Double slotted	Supports 3.3V and 1.5V signaling. Available speeds 1x, 2x at 3.3V and 1x, 2x, 4x at 1.5V.
AGP 3.0 Card	1.5V slot	Supports only 0.8V signaling. Available speeds 4x, 8x.
Universal 1.5V AGP 3.0 Card	1.5V slot	Supports 1.5V and 0.8V signaling. Available speeds 1x, 2x, 4x at 1.5V and 4x, 8x at 0.8V.
Universal AGP 3.0 Card	Double slotted	Supports AGP 3.3v, 1.5V, and 0.8V signaling. Available speeds 1x, 2x at 3.3V and 1x, 2x, 4x at 1.5V and 4x, 8x at 0.8V.



Hình 8.25. Sơ đồ chân của AGP

	3.3 Volt Boards		Universal Boards		1.5 Volt Boards	
Pin #	Side A	Side B	Side A	Side B	Side A	Side B
1	+12V	OVRCNT#	+12V	OVRCNT#	+12V	OVRCNT#
2	TYPEDET#	+5.0V	TYPEDET#	+5.0V	TYPEDET#	+5.0V
3	Reserved	5.0V	Reserved	5.0V	Reserved	5.0V
4	USB-	USB+	USB-	USB+	USB-	USB+
5	Ground	Ground	Ground	Ground	Ground	Ground

6	INTA#	INTB#	INTA#	INTB#	INTA#	INTB#
7	RST#	CLK	RST#	CLK	RST#	CLK
8	GNT#	REQ#	GNT#	REQ#	GNT#	REQ#
9	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3
10	ST1	ST0	ST1	ST0	ST1	ST0
11	Reserved	ST2	Reserved	ST2	Reserved	ST2
12	PIPE#	RBF#	PIPE#	RBF#	PIPE#	RBF#
13	Ground	Ground	Ground	Ground	Ground	Ground
14	Reserved	Reserved	WBF#	Reserved	WBF#	Reserved
15	SBA1	SBA0	SBA1	SBA0	SBA1	SBA0
16	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3
17	SBA3	SBA2	SBA3	SBA2	SBA3	SBA2
18	Reserved	SB_STB	SB_STB#	SB_STB	SB_STB#	SB_STB
19	Ground	Ground	Ground	Ground	Ground	Ground
20	SBA5	SBA4	SBA5	SBA4	SBA5	SBA4
21	SBA7	SBA6	SBA7	SBA6	SBA7	SBA6
22	Key	Key	Reserved	Reserved	Reserved	Reserved
23	Key	Key	GROUND	GROUND	GROUND	GROUND
24	Key	Key	Reserved	3.3Vaux	Reserved	3.3Vaux
25	Key	Key	Vcc 3.3	Vcc 3.3	Vcc 3.3	Vcc 3.3
26	AD30	AD31	AD30	AD31	AD30	AD31
27	AD28	AD29	AD28	AD29	AD28	AD29
28	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3
29	AD26	AD27	AD26	AD27	AD26	AD27
30	AD24	AD25	AD24	AD25	AD24	AD25
31	Ground	Ground	Ground	Ground	Ground	Ground
32	Reserved	AD STB1	AD STB1#	AD STB1	AD STB1#	AD STB1
33	C/BE3#	AD23	C/BE3#	AD23	C/BE3#	AD23
34	Vddq 3.3	Vddq 3.3	Vddq	Vddq	Vddq 1.5	Vddq 1.5
35	AD22	AD21	AD22	AD21	AD22	AD21
36	AD20	AD19	AD20	AD19	AD20	AD19
37	Ground	Ground	Ground	Ground	Ground	Ground
38	AD18	AD17	AD18	AD17	AD18	AD17
39	AD16	C/BE2#	AD16	C/BE2#	AD16	C/BE2#

40	Vddq 3.3	Vddq 3.3	Vddq	Vddq	Vddq 1.5	Vddq 1.5
41	FRAME#	IRDY#	FRAME#	IRDY#	FRAME#	IRDY#
42	Reserved	3.3Vaux	Reserved	3.3Vaux	KEY	KEY
43	Ground	Ground	Ground	Ground	KEY	KEY
44	Reserved	Reserved	Reserved	Reserved	KEY	KEY
45	VCC 3.3	VCC 3.3	VCC 3.3	VCC 3.3	KEY	KEY
46	TRDY#	DEVSEL#	TRDY#	DEVSEL#	TRDY#	DEVSEL#
47	STOP#	Vddq 3.3	STOP#	Vddq	STOP#	Vddq 1.5
48	PME#	PERR#	PME#	PERR#	PME#	PERR#
49	Ground	Ground	Ground	Ground	Ground	Ground
50	PAR	SERR#	PAR	SERR#	PAR	SERR#
51	AD15	C/BE1#	AD15	C/BE1#	AD15	C/BE1#
52	Vddq 3.3	Vddq 3.3	Vddq	Vddq	Vddq 1.5	Vddq 1.5
53	AD13	AD14	AD13	AD14	AD13	AD14
54	AD11	AD12	AD11	AD12	AD11	AD12
55	Ground	Ground	Ground	Ground	Ground	Ground
56	AD9	AD10	AD9	AD10	AD9	AD10
57	C/BE0#	AD8	C/BE0#	AD8	C/BE0#	AD8
58	Vddq 3.3	Vddq 3.3	Vddq	Vddq	Vddq 1.5	Vddq 1.5
59	Reserved	AD STB0	Reserved	AD STB0#	Reserved	AD STB0#
60	AD6	AD7	AD6	AD7	AD6	AD7
61	Ground	Ground	Ground	Ground	Ground	Ground
A62	AD4	AD5	AD4	AD5	AD4	AD5
63	AD2	AD3	AD2	AD3	AD2	AD3
64	Vddq 3.3	Vddq 3.3	Vddq	Vddq	Vddq 1.5	Vddq 1.5
65	AD0	AD1	AD0	AD1	AD0	AD1
66	Reserved	Reserved	Vrefcg	Vrefcg	Vrefcg	Vrefcg

PCI Express

PCI và PCI express (PCIe) khác biệt cơ bản trên giao thức ghép nối tiếp serial. PCIe là kiểu kết nối điểm-điểm theo 2 chiều với băng thông giống nhau và không chia sẻ cho các thiết bị khác. Do đó có thể tăng băng thông khi tăng số

đường truyền nối tiếp. Hiện tại có 5 kiểu khe cắm cho PCIe. PCIe x 16 có băng thông đến 8GB/s đáp ứng nhu cầu đồ họa ngày càng tăng

Ngày nay, một mainboard có thể cung cấp đến 20 đường PCI, thông thường sẽ có 1 PCI x 16 và 4 PCI x 1 để thay thế cho các khe cắm PCI thông thường

8.4.4. Kết nối máy tính với các thiết bị bên ngoài

Trong quá trình hoạt động, máy tính còn có nhu cầu tương tác với các thiết bị khác. Máy tính và các thiết bị đó sẽ được kết nối với nhau thông qua các cổng kết nối, thông thường là các cổng kết nối serial, nhưng đôi khi cũng là cổng kết nối song song (như kết nối máy in qua cổng LPT). Trong mục này, ta sẽ điểm qua một số cổng kết nối thông dụng

RS-232

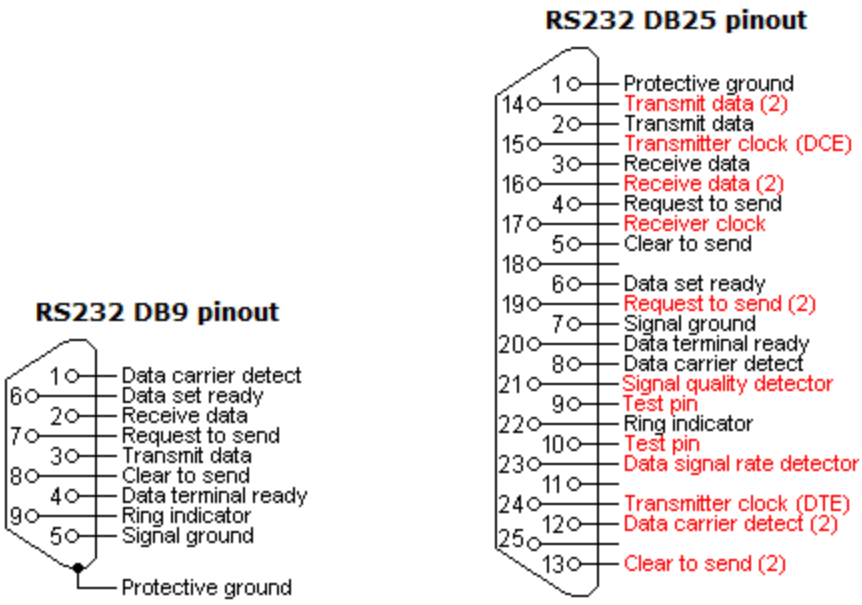
Kết nối theo chuẩn RS-232 là kết nối không đồng bộ nối tiếp. Ý nghĩa của nối tiếp là các bit sẽ được truyền đi lần lượt trên một đường truyền vật lý. Một byte dữ liệu được truyền đi trên một khung như sau

Start	0	1	2	3	4	5	6	7	P	Stop
0	LSB							MSB		1

Một khung truyền có 1 start bit và một stop bit để đánh dấu khung truyền. Dữ liệu truyền có thể có độ dài là 7 bit hoặc 8 bit. 1 bit kiểm tra chẵn lẻ được sử dụng để kiểm tra tính toàn vẹn của dữ liệu để phát hiện lỗi. Trên đường truyền, mức điện áp mang mức logic 1 nằm trong khoảng -3V đến -15V nhưng cũng có thể lên tới -25V. Ngược lại, mức điện áp mang mức logic 0 nằm trong khoảng 3V đến 15V nhưng cũng có thể lên tới 25V. Dải điện áp nằm trong khoảng -3V đến 3V là dải cấm.

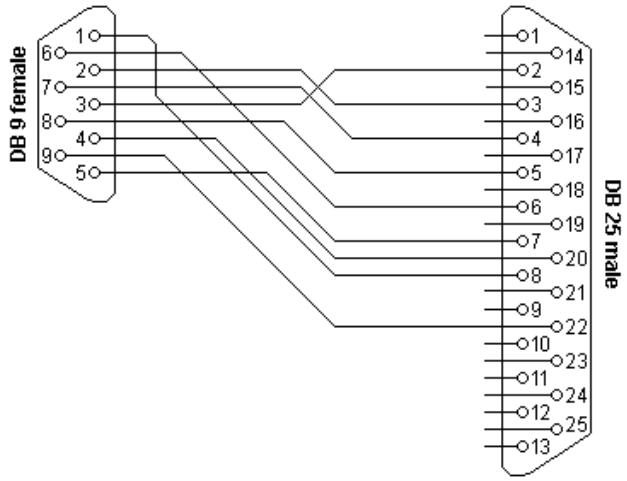
Tốc độ truyền phụ thuộc khoảng cách 19.2kBd với khoảng cách 30 đến 50m. Hiện nay có thể có những hệ thống truyền với tốc độ lên tới 460kBd và hơn nữa nhưng trong thực tế sử dụng thì những hệ thống như vậy rất ít và khó thực hiện.

Sơ đồ chân của cổng cắm RS-232 được thể hiện trên hình 8.26. RS-232 có 2 loại cổng 9 chân hoặc 25 chân. Hiện nay, cổng 25 chân gần như không còn được sử dụng.



Hình 8.26. Sơ đồ chân của RS-232

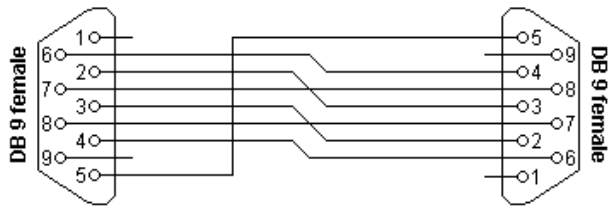
Phương pháp ghép nối chuyển đổi giữa 2 loại cổng 9 chân và 25 chân



DB9 - DB25 conversion

DB9	DB25	Function
1	8	Data carrier detect
2	3	Receive data
3	2	Transmit data
4	20	Data terminal ready
5	7	Signal ground
6	6	Data set ready
7	4	Request to send
8	5	Clear to send
9	22	Ring indicator

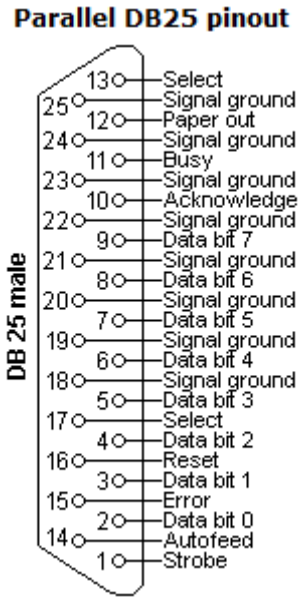
Phương pháp ghép nối 2 máy tính qua cổng 9 chân trong chế độ full handshaking



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
4	6	DTR → DSR
5	5	Signal ground
6	4	DSR ← DTR
7	8	RTS → CTS
8	7	CTS ← RTS

Cổng LPT

Cổng LPT là cổng truyền dữ liệu song song được thiết kế riêng cho truyền dữ liệu giữa máy tính và máy in. Cổng này có 25 chân được thể hiện trong hình 8.27



Hình 8.27. Sơ đồ chân của cổng LPT

Parallel printer cable

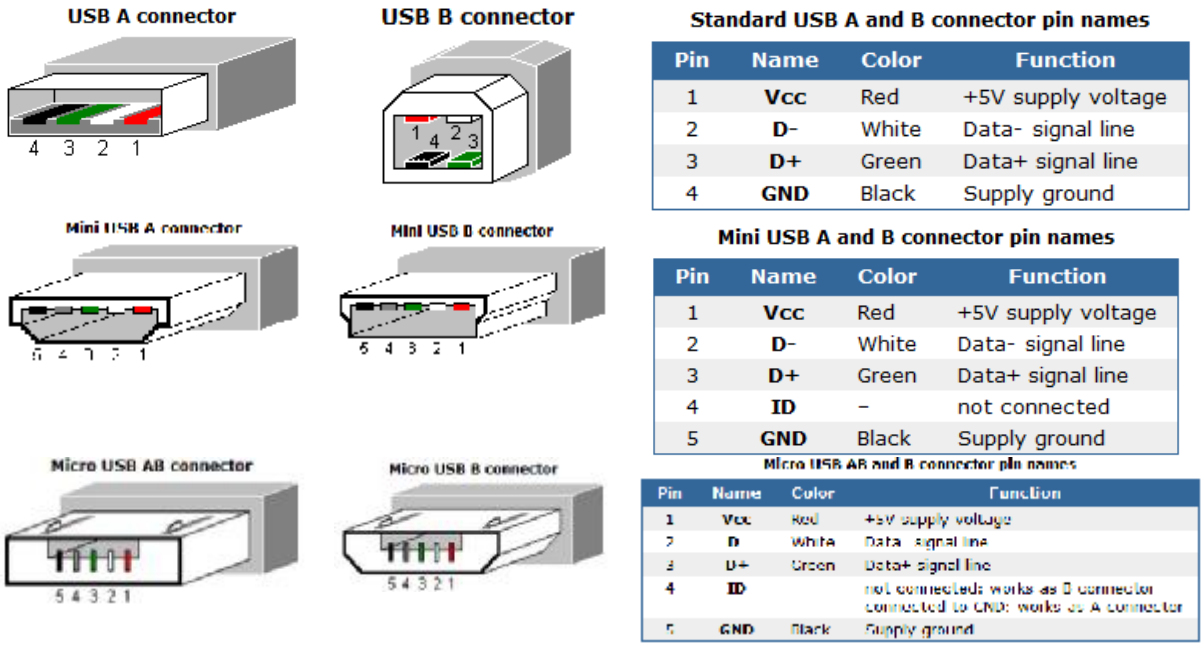
Line	DB 25 male (computer)		Centronics (printer)
Strobe	1	→	1
Data bit 0	2	→	2
Data bit 1	3	→	3
Data bit 2	4	→	4
Data bit 3	5	→	5
Data bit 4	6	→	6
Data bit 5	7	→	7
Data bit 6	8	→	8
Data bit 7	9	→	9
Acknowledge	10	←	10
Busy	11	←	11
Paper out	12	←	12
Select	13	←	13
Autofeed	14	→	14
Error	15	←	32
Reset	16	→	31
Select	17	→	36
Signal ground	18	↔	33
Signal ground	19	↔	19 + 20
Signal ground	20	↔	21 + 22
Signal ground	21	↔	23 + 24
Signal ground	22	↔	25 + 26
Signal ground	23	↔	27
Signal ground	24	↔	28 + 29
Signal ground	25	↔	16 + 30
Shield	Cover	↔	Cover + 17

USB (Univeral Serial Bus)

USB là chuẩn kết nối máy tính phổ biến nhất hiện nay. Đặc điểm quan trọng nhất của chuẩn này là nó có thể kết nối một số lượng lớn các thiết bị khác nhau như bàn phím, chuột, ổ đĩa, camera, máy in,... USB kết thúc tình trạng chắp vá, không đồng nhất của các giao diện ngoại vi trong máy tính cá nhân. Ngoài ra, bản thân tín hiệu trên cổng USB còn có khả năng cung cấp nguồn +5V cho các thiết bị bên ngoài. Đây là một đặc điểm mà không một chuẩn nào khác hỗ trợ.

Khi thực hiện truyền dữ liệu theo chuẩn USB, máy tính được coi là master và nó sẽ quản lý tất cả các thiết bị khác gắn vào nó. Thiết bị được gắn vào máy tính theo chuẩn USB đều có khả năng hoạt động ngay lập tức mà không cần khởi động lại hệ thống

Hình 8.28. mô tả một số phiên bản của chuẩn USB.



Standard USB A and B connector pin names

Pin	Name	Color	Function
1	Vcc	Red	+5V supply voltage
2	D-	White	Data- signal line
3	D+	Green	Data+ signal line
4	GND	Black	Supply ground

Mini USB A and B connector pin names

Pin	Name	Color	Function
1	Vcc	Red	+5V supply voltage
2	D-	White	Data- signal line
3	D+	Green	Data+ signal line
4	ID	-	not connected
5	GND	Black	Supply ground

Micro USB AB and B connector pin names

Pin	Name	Color	Function
1	Vcc	Red	+5V supply voltage
2	D	White	Data- signal line
3	D+	Green	Data+ signal line
4	ID		not connected; works as B connector connected to GND; works as A connector
5	GND	Black	Supply ground

Hình 8.28. Các phiên bản USB thông dụng

8.5. Lỗi truyền thông và các phương pháp phát hiện lỗi

Trong tất cả cấu trúc máy tính, và đặc biệt trong trường hợp liên quan tới truyền thông giữa các máy tính với nhau, luôn luôn có thể xảy ra trường hợp dữ liệu nhận được trên đương truyền bị lỗi. Nguyên nhân của lỗi là do nhiễu trên đường truyền. Cụ thể là các dữ liệu trên đường truyền được mã hóa dưới dạng số nhị phân, thực tế là các giá trị điện áp hoặc dòng điện. Nhiễu trên đường truyền xuất

phát từ bản thân cấu trúc đường truyền, xuất phát từ môi trường bên ngoài như áp suất, các tia gama,...Các nguyên nhân này làm phát sinh lỗi, tức là làm mức logic 0 chuyển thành 1 và mức logic 1 chuyển thành 0.

Giả sử hệ thống cần truyền một ký tự ASCII là ký tự 'b' từ bên truyền đến bên nhận và trong quá trình truyền phát sinh một lỗi ở bit có trong số nhỏ nhất. Chuỗi ký tự đúng của ký tự 'b' là 1100010, nhưng chuỗi nhận được là 1100011, thể hiện là ký tự 'c'. Bên nhận sẽ không có bất cứ một công cụ nào để có thể phát hiện được lỗi và chuỗi nhận được vẫn được chấp nhận

Để khắc phục hiện tượng này, hệ thống truyền và nhận dữ liệu sẽ được bổ xung thông tin "check bit" kèm với dữ liệu. Hệ thống nhận dữ liệu sẽ kiểm tra thông tin check bit kèm theo ở cùng một điều kiện đối với bên truyền để phát hiện ra lỗi, và trong một số trường hợp có thể sửa lại lỗi. Trong nội dung của chương này, ta sẽ xem xét 2 phương pháp cơ bản để kiểm tra lỗi và sửa lỗi. Nhưng trước hết ta hãy tìm hiểu một số định nghĩa cơ bản về truyền tin

8.5.1. Tỉ lệ bit lỗi

Có rất nhiều cách khác nhau để có thể giới thiệu về các lỗi trong hệ thống máy tính, và các lỗi này cũng có thể có rất nhiều dạng khác nhau. Tại thời điểm này, ta sẽ giả sử rằng xác suất một bit nhận được bị lỗi độc lập với xác suất các bit khác ở gần đó bị lỗi. Trong trường hợp đó, ta có thể định nghĩa tỉ lệ bit lỗi **BER** (bit error rate) là xác suất bit nhận được bị lỗi. Rõ ràng trong định nghĩa này, con số đó là rất nhỏ, thông thường tỉ lệ bit lỗi nhỏ hơn 10^{-12} cho mỗi bit được kiểm tra, tức là sẽ xuất hiện một lỗi trong 10^{12} bit được kiểm tra.

Trong một hệ thống máy tính, tỉ lệ bit lỗi BER có thể được chấp nhận ở giá trị 10^{-18} hoặc ít hơn. Theo một ước tính, nếu xung nhịp đồng hồ ở 100MHz, và hệ thống là 32 bit. Khi đó số lượng lỗi trong 1 giây sẽ được tính là $10^{-18} \times 100 \times 10^{6} \times 32$ hay 3.2×10^{-9} lỗi trong 1 giây, tức là xấp xỉ 1 lỗi sẽ xảy ra trong khoảng thời gian khoảng 10 năm.

8.5.2. Phát hiện lỗi và sửa lỗi single-bit-error

Một phương pháp đơn giản và lâu đời nhất để phát hiện lỗi trong quá trình truyền và nhận ký tự là kiểm tra chẵn lẻ **parity bit**. Một bit parity, mang giá trị

0 hoặc 1 được thêm vào mỗi một ký tự để kiểm tra tổng số bit mang giá trị 1 trong ký tự đó là chẵn hay lẻ. Trong ví dụ về truyền ký tự 'b' theo mã ASCII ở trên, chuỗi 1100010 nếu được kiểm tra chẵn, một bit mang giá trị 1 sẽ được thêm vào vì tổng số bit mang giá trị 1 trong chuỗi là lẻ. Cuối cùng, chuỗi truyền đi là 11000101. Hệ thống nhận sẽ nhận và đếm số bit mang giá trị 1 trong chuỗi nhận được là chẵn hay lẻ. Nếu kết quả là chẵn, quá trình truyền không có lỗi và hệ thống sẽ khôi phục nội dung đã được truyền. Ngược lại thì chuỗi nhận được đã bị lỗi. Phương pháp này không phát hiện ra lỗi nếu số bit bị lỗi trong chuỗi là 2 hoặc nhiều hơn (nhưng là số chẵn). Trong trường hợp đó, ta có thể sử dụng phương pháp khác sẽ được thảo luận trong phần dưới đây.

Hamming Codes

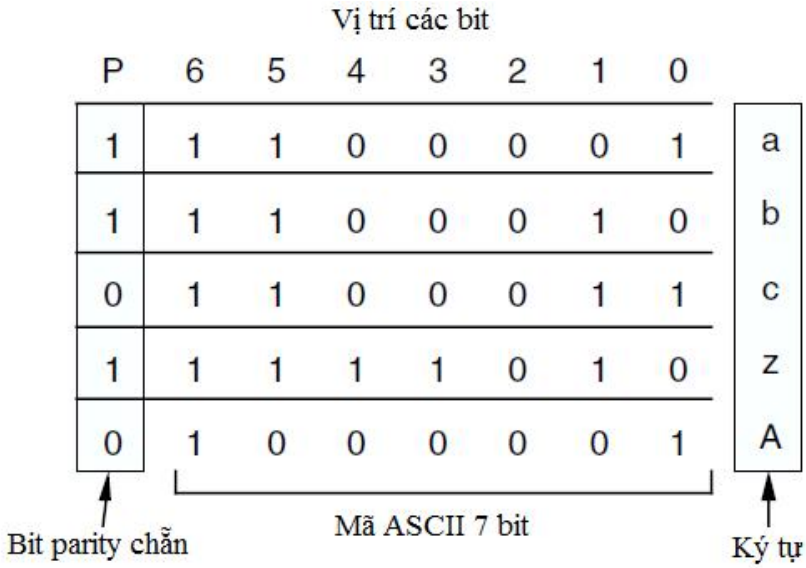
Những bit thêm vào dữ liệu ngoài chức năng phát hiện lỗi, nó còn cần có khả năng tự sửa lỗi. Một số phương pháp mã hóa phổ biến được sử dụng rộng rãi là mã Hamming. Phương pháp này được thực hiện dựa trên các nghiên cứu của Richard Hamming tại Bell Telephone Laboratories, hiện nay hoạt động dưới sự quản lý của Lucent Technologoies

Khoảng cách Hamming HD được định nghĩa là số lượng bit lỗi tối thiểu mà không đảm bảo chắc chắn phát hiện được trong một chuỗi bit. Khi truyền một chuỗi ký tự ASCII và nhận được 1 chuỗi ký tự ASCII khác, bên nhận không có khả năng phát hiện ra có lỗi trên đường truyền, do đó HD = 1. Nếu bây giờ bằng một phương pháp nào đó, ta tăng khoảng cách HD = 2 thì có nghĩa là ta có thể phát hiện tất cả các lỗi đơn (single-bit error)

Một phương pháp để tăng khoảng cách HD đơn giản là thêm 1 bit kiểm tra chẵn lẻ pariy bit như đã trình bày ở trên. Nếu thực hiện kiểm tra chẵn với bit kiểm tra được đặt ở phía bên trái. Ký tự 'a' khi đó sẽ được biểu diễn bởi chuỗi 11100001. Tương tự, ta có thể xác định được chuỗi thể hiện ký tự 'b', 'c', 'z' hoặc 'A' như thể hiện trên hình 8.29. Bảng mã ASCII lúc này sẽ có tổng số 2^8 = 256 trạng thái khác nhau. Một nửa trong số đó thể hiện mã ký tự sai. Khi bên nhận nhận được mã ký tự sai, bên nhận sẽ yêu cầu thực hiện truyền lại mã ký tự đó.

Quá trình truyền lại ký tự bị sai không phải lúc nào cũng có thể thực hiện được. Do đó cần có một phương pháp nào đó để đồng thời phát hiện lỗi và tự sửa lỗi.

Phương pháp kiểm tra chẵn lẻ có thể phát hiện lỗi nhưng nó không thể xác định được vị trí lỗi. Giả sử như ta nhận được 11100011 và phương pháp kiểm tra là



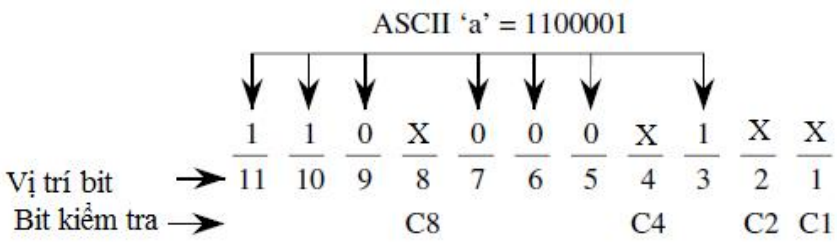
Hình 8.29. Kiểm tra chẵn một số ký tự

chẵn, ta biết chắc chắn là đã có lỗi xảy ra vì tổng số bit mang giá trị logic 1 là lẻ. Nhưng ta không có đủ thông tin để xác định bit truyền đi là ‘a’, ‘b’ hay bất kỳ một ký tự nào khác.

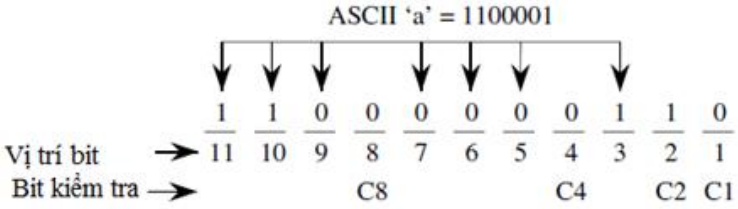
Để có thể xây dựng được một loại mã có khả năng phát hiện và tự sửa lỗi dạng single-bit-error, ta phải thêm nhiều hơn 1 bit dự phòng trong chuỗi dữ liệu 7 bit ban đầu. Thực tế ta sẽ phải thêm 4 bit dự phòng để có thể loại bỏ các trường hợp lỗi 1 bit trong chuỗi có thể xảy ra. Giả sử chuỗi ký tự ‘a’ cần truyền đi là 1100001 thì ta phải loại bỏ các trường hợp lỗi là 0100001, 1000001, 1110001, 1101001, 1100101, 1100011, 1100000. Như vậy, ta sẽ sử dụng 11bit để mã hóa lại bảng mã ASCII.

Quá trình mã hóa được thực hiện theo các bước như sau

1. Các bit dự phòng được thêm vào ở các vị trí C1, C2, C4, C8 như sau



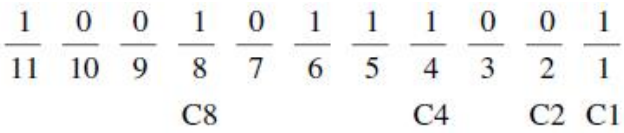
2. Bit C1 là kiểm tra chẵn hoặc lẻ của tổ hợp các bit số {1, 3, 5, 7, 9, 11}. Như ví dụ trên hình, C1 = 0 vì chỉ có các bit số 3 và 11 bằng 1 (kiểm tra chẵn)
3. Bit C2 là kiểm tra chẵn lẻ của tổ hợp các bit {2, 3, 6, 7, 10, 11}. Tương tự với phân tích trên ta có C2 = 1
4. Bit C4 là kiểm tra chẵn lẻ của tổ hợp các bit {4, 5, 6, 7}. C4 trong trường hợp này là 0
5. Bit C8 là kiểm tra chẵn lẻ của {8, 9, 10, 11}. C8 = 0
6. Chuỗi bit truyền đi sẽ là



Ở thiết bị nhận, ta sẽ tiến hành kiểm tra theo quy trình sau

1. Kiểm tra chẵn lẻ các bit dự phòng C1, C2, C4, C8 với các tổ hợp tương ứng.
2. Nếu kết quả kiểm tra đều phù hợp, thì dữ liệu truyền không lỗi
3. Nếu có lỗi tức là ít nhất một giá trị Ci có kết quả kiểm tra là lẻ thì vị trí lỗi được tính là $n = \sum_{i\ loi} Ci$

Tiếp tục ví dụ trên, giả sử ta nhận được chuỗi 10010111001.



Ta sẽ kiểm tra chẵn lẻ

Kiểm tra	C1	{1, 3, 5, 7, 9, 11}	Kết quả lẻ
Kiểm tra	C2	{2, 3, 6, 7, 10, 11}	Kết quả chẵn
Kiểm tra	C4	{4, 5, 6, 7}	Kết quả lẻ
Kiểm tra	C8	{8, 9, 10, 11}	Kết quả chẵn

Kết quả lẻ ở C1 và C4 do đó vị trí bit bị lỗi là n = 1 + 4 = 5. Chuỗi được sửa lại là 10010101001. Dữ liệu thu được 1000100. Đây chính là ký tự 'D' trong bảng mã ASCII

8.5.3. Kiểm tra dự phòng theo chiều dọc VRC

Phương pháp kiểm tra dự phòng theo chiều dọc VRC (Vertical Redundancy Checking) là phương pháp được sử dụng để kiểm tra theo khối dữ liệu. Khi truyền một nhóm dữ liệu word, các bit dự phòng được thêm vào mỗi word. Kết quả mỗi word sau khi được thêm vào bit kiểm tra chẵn lẻ lại được xếp theo từng hàng. Cuối cùng ta lại kiểm tra chẵn lẻ một lần nữa theo chiều dọc. Ta có thể hình dung rõ hơn quá trình kiểm tra VRC bằng ví dụ trên hình 8.30 khi ta truyền 8 ký tự từ A đến F

P	*Code*	*Character*
0	1000001	A
0	1000010	B
1	1000011	C
0	1000100	D
1	1000101	E
1	1000110	F
0	1000111	G
0	1001000	H
1	0001000	Checksum

Hình 8.30. Kiểm tra lỗi theo phương pháp VRC

8.5.4. Kiểm tra mã vòng CRC

Kiểm tra mã vòng CRC là một phương pháp rất mạnh để phát hiện lỗi và sửa lỗi. Nguyên lý cơ bản của CRC là từ một lỗi bit phát sinh, lỗi này sẽ bùng nổ và nếu có một cơ chế kiểm tra liên tục bit lỗi này thì xác suất phát hiện lỗi sẽ tăng lên

CRC sử dụng một mã đa thức như một khung truyền được chia thành 2 trường. Trường dữ liệu và trường kiểm tra CRC. Sau khi nhận được khung truyền (bao gồm dữ liệu và CRC bit), hệ thống sẽ kiểm tra theo cùng một cơ chế đối với bên gửi. Nếu kết quả phù hợp, tức là không có lỗi. Nếu kết quả có sự khác biệt, lỗi đã xảy ra.

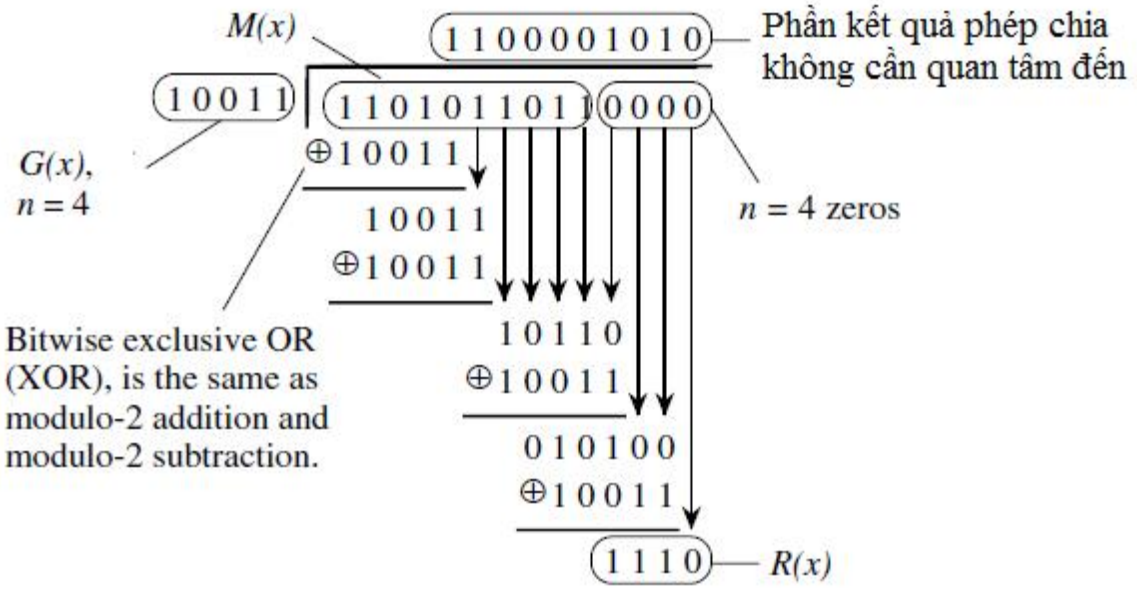
Để thực hiện kiểm tra CRC, bên gửi dữ liệu sẽ thực hiện theo các bước sau

1. Với chuỗi dữ liệu cần truyền đi, nó được coi là một đa thức M(x) có bậc là k sẽ được thêm n bit số 0 vào đằng sau. Với n là bậc của một đa thức sinh G(x) và k>n
2. Đa thức thu được gồm M(x) và chuỗi n bit 0 thêm vào được thực hiện phép chia cho đa thức G(x) theo phép chia modul 2 được phần dư R(x). R(x) có số bit không lớn hơn n bit.
3. Đa thức truyền đi là T(x) = M(x) + R(x)

Bên nhận dữ liệu giả sử là T'(x) sẽ tiến hành kiểm tra như sau

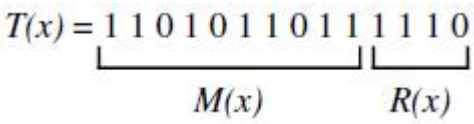
1. Thực hiện phép chia T'(x) cho đa thức sinh G(x)
2. Nếu kết quả R'(x) bằng 0 thì quá trình truyền không có lỗi, nếu kết quả khác 0 thì quá trình truyền đã có lỗi

Giả sử ta cần truyền chuỗi dữ liệu M(x) = 1101011011. Đa thức sinh G(x) = 10011 (tức là $G(x) = x^4 + x + 1$). Với đa thức sinh như vậy, ta thấy n = 4, tức là ta sẽ thêm 4 số 0 vào chuỗi cần truyền dữ liệu để được chuỗi 11010110110000. Thực hiện phép chia như hình 8.31 ta được R(x) = 1110.



Hình 8.31. Thực hiện phép chia trong kiểm tra CRC

Chuỗi truyền đi sẽ là



Một số đa thức sinh hay được sử dụng cho kết quả tốt trong việc phát hiện lỗi, đó là

$$CRC - 16 = x^{16} + x^{15} + x^2 + 1$$

$$CRC - CCITT = x^{16} + x^{12} + x^5 + 1$$

$$CRC - 32 = x^{32} + x^{26} + x^{23} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

TỔNG KẾT CHƯƠNG

Các ổ lưu trữ dữ liệu có rất nhiều dạng khác nhau. Vi dụ ổ lưu trữ có các loại như ổ cứng, băng từ. Ổ đĩa quangcos mật độ lưu trữ cao hơn ổ đĩa từ, nhưng có giá thành cao hơn và không hỗ trợ khả năng ghi dữ liệu. Các ổ đĩa quang thông dụng là CD ROM, DVD ROM.

Các thiết bị vào/ra cũng có rất nhiều loại khác nhau. Trong chương này chúng ta đã nghiên cứu một số thiết bị cơ bản. Đó là bàn phím, bit pad, chuột, trackball, màn hình hiển thị CRT.

Các cổng vào ra dữ liệu cũng liên quan tới hệ thống bus truyền thông bên trong máy tính. Chúng ta cũng tìm hiểu một số hệ thống bus cơ bản trong máy tính. Các hệ thống bus này khác nhau ở tốc độ truyền thông, độ rộng dữ liệu mà nó hỗ trợ, khả năng kết nối. Với mỗi hệ thống bus này, chúng ta cũng đã tìm hiểu sơ đồ chân và các tín hiệu trong bus.

Để kết nối với các thiết bị ngoại vi bên ngoài, máy tính phải sử dụng các kênh truyền dẫn riêng (khác với bus). Một số kênh truyền dẫn là RS-232, cổng LPT, và đặc biệt là chuẩn USB

Trong quá trình truyền nhận dữ liệu, lỗi phát sinh là không thể tránh khỏi. Ta đã tìm hiểu một số phương pháp phát hiện và tự sửa lỗi cơ bản, thường được sử dụng như kiểm tra chẵn lẻ, kiểm tra VRC, và một phương pháp rất mạnh là CRC.

Tài liệu Cấu trúc máy tính & Hợp ngữ

Chương 1
TỔNG QUAN VỀ CẤU TRÚC MÁY TÍNH

1. Ngôn ngữ, cấp máy và máy ảo (Language, level and virtual machine)

1.1. Giới thiệu

Máy tính số (Digital computer) là máy giải quyết các vấn đề bằng cách thực hiện các chỉ thị do con người cung cấp. Chuỗi các chỉ thị này gọi là chương trình (program). Các mạch điện tử trong một máy tính số sẽ thực hiện một số giới hạn các chỉ thị đơn giản cho trước. Tập hợp các chỉ thị này gọi là tập lệnh của máy tính. Tất cả các chương trình muốn thực thi đều phải được biến đổi sang tập lệnh trước khi được thi hành. Các lệnh cơ bản là:

- Cộng 2 số.
- So sánh với 0.
- Di chuyển dữ liệu.

Tập lệnh của máy tính tạo thành một ngôn ngữ giúp con người có thể tác động lên máy tính, ngôn ngữ này gọi là ngôn ngữ máy (machine language). Tuy nhiên, hầu hết các ngôn ngữ máy đều đơn giản nên để thực hiện một yêu cầu nào đó, người thiết kế phải thực hiện một công việc phức tạp. Đó là chuyển các yêu cầu này thành các chỉ thị có chứa trong tập lệnh của máy. Vấn đề này có thể giải quyết bằng cách thiết kế một tập lệnh mới thích hợp cho con người hơn tập lệnh đã cài đặt sẵn trong máy (built-in). Ngôn ngữ máy sẽ được gọi là ngôn ngữ cấp 1 (L1) và ngôn ngữ vừa được hình thành gọi là ngôn ngữ cấp 2 (L2).

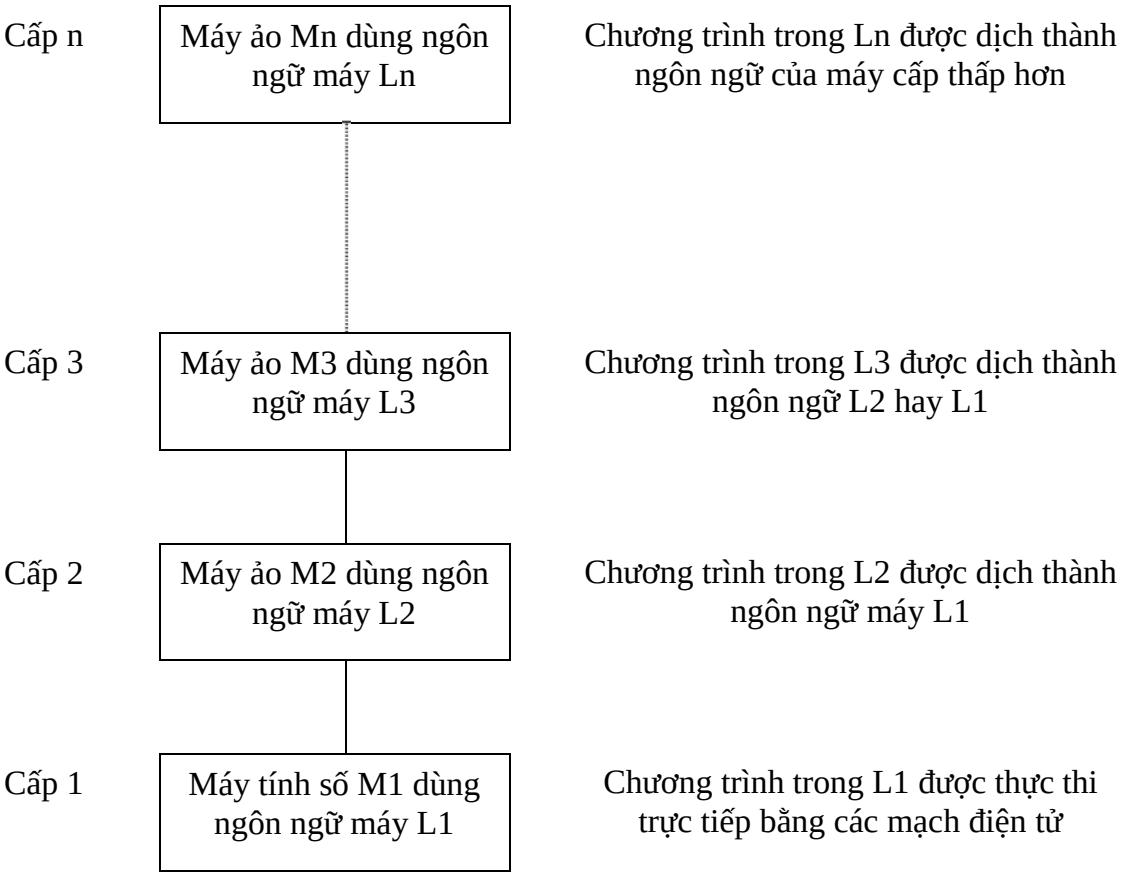
Một phương pháp thực thi chương trình L2 là chuyển một lệnh trong L2 bằng một chuỗi các lệnh tương đương trong L1. Kết quả là sẽ tạo thành một chương trình L1 và máy tính sẽ thực hiện chương trình tương đương L1 thay vì thực hiện chương trình L2. Kỹ thuật này gọi là biên dịch (compile). Cách khác là một lệnh trong chương trình L2 sẽ được xem như dữ liệu ngõ vào của chương trình L1 và toàn bộ chương trình L2 sẽ được thực thi tuần tự. Kỹ thuật này gọi là thông dịch (interprete), nó không yêu cầu tạo ra một chương trình mới trong L1.

Biên dịch và thông dịch đều thực hiện chương trình L2 thông qua tập lệnh trong chương trình L1. Chúng khác nhau ở chỗ là khi biên dịch thì toàn bộ chương trình L2 sẽ được chuyển thành chuỗi lệnh L1 rồi sau đó mới được thực thi còn đối với phương pháp thông dịch thì sẽ thực thi từng lệnh trong L2. Để thuận tiện hơn, ta giả sử tồn tại một máy tính sử dụng ngôn ngữ máy là L2, ta gọi máy tính này là máy ảo (virtual machine).

Tuy nhiên, trong thực tế, để có thể thực hiện biên dịch và thông dịch , các ngôn ngữ L1 và L2 không được khác nhau nhiều. Như vậy, ngôn ngữ L2 cũng không thật sự giúp ích nhiều cho người thiết kế. Do đó, một tập lệnh kế tiếp được hình thành sẽ hướng về con người nhiều hơn là máy tính, tập lệnh này sẽ tạo thành một ngôn ngữ và ta gọi là ngôn ngữ L3. Ta có thể viết các chương trình trong L3 như là đã tồn tại máy tính sử dụng

ngôn ngữ L3 (máy ảo L3). Các chương trình này sẽ được dịch sang ngôn ngữ L2 và được thực thi bằng một chương trình dịch L2.

Việc xây dựng toàn bộ chuỗi các ngôn ngữ, mỗi ngôn ngữ được tạo ra sẽ thích hợp hơn ngôn ngữ trước đó sẽ có thể tiếp tục cho đến khi nhận được ngôn ngữ thích hợp nhất. Sơ đồ một máy ảo n cấp có thể biểu diễn như sau:

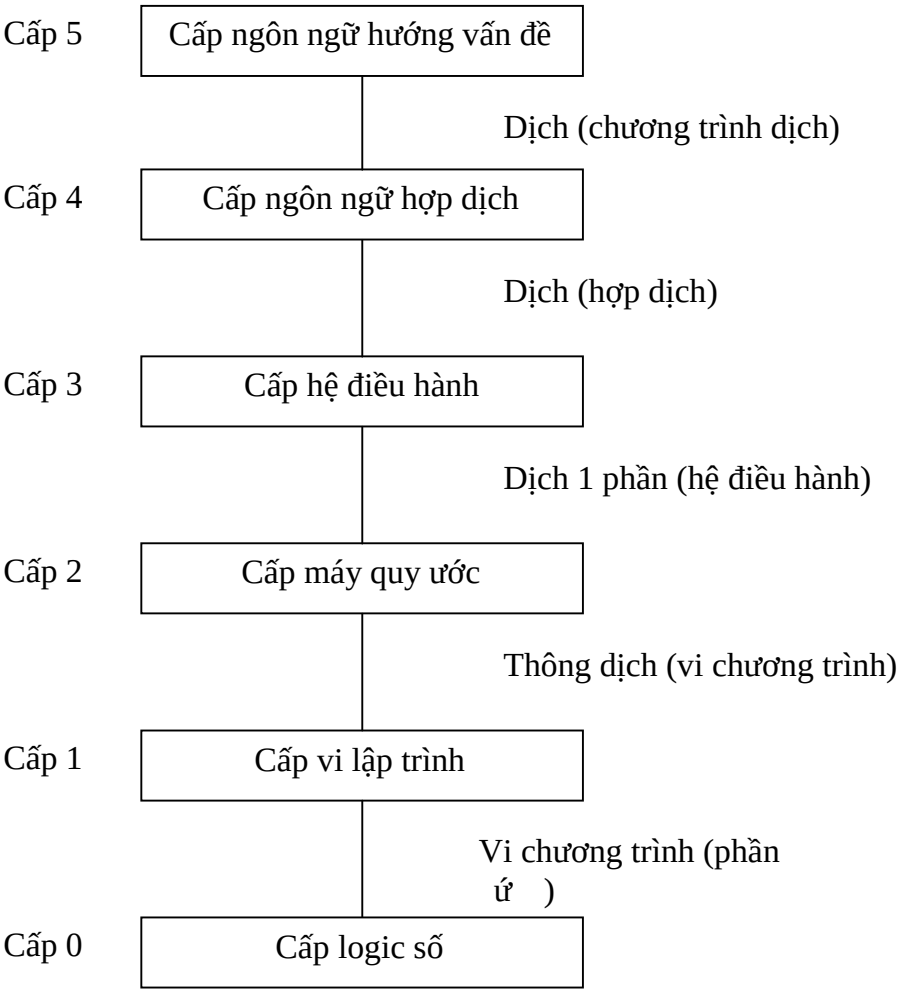


Hình 1.1. Máy ảo n cấp

Một máy tính số có n cấp có thể xem như có n-1 máy ảo khác nhau, mỗi máy ảo có một ngôn ngữ máy riêng. Các chương trình viết trên các máy ảo này không thể thực thi trực tiếp mà phải dịch thành các ngôn ngữ máy cấp thấp hơn. Chỉ có máy thật dùng ngôn ngữ máy L1 mới có thể thực thi trực tiếp bằng các mạch điện tử. Một lập trình viên sử dụng máy ảo cấp n không cần biết tất cả các trình dịch này. Chương trình trong máy ảo cấp n sẽ được thực thi bằng cách dịch thành ngôn ngữ máy cấp thấp hơn và ngôn ngữ máy này sẽ được dịch thành ngôn ngữ máy thấp hơn nữa hay dịch trực tiếp thành ngôn ngữ máy L1 và thực thi trực tiếp trên các mạch điện tử.

1.2. Máy nhiều cấp

Hầu hết các máy tính hiện nay gồm có 6 cấp:



Hình 1.2 – Các cấp trên máy tính số

Cấp 0 chính là phần cứng của máy tính. Các mạch điện tử của cấp này sẽ thực thi các chương trình ngôn ngữ máy của cấp 1. Trong cấp logic số, đối tượng quan tâm là các cổng logic. Các cổng này được xây dựng từ một nhóm các transistor.

Cấp 1 là cấp ngôn ngữ máy thật sự. Cấp này có một chương trình gọi là vi chương trình (microprogram), vi chương trình có nhiệm vụ thông dịch các chỉ thị của cấp 2. Hầu hết các lệnh trong cấp này là di chuyển dữ liệu từ phần này đến phần khác của máy hay thực hiện việc một số kiểm tra đơn giản.

Mỗi máy cấp 1 có một hay nhiều vi chương trình chạy trên chúng. Mỗi vi chương trình xác định một ngôn ngữ cấp 2. Các máy cấp 2 đều có nhiều điểm chung ngay cả các máy cấp 2 của các hãng sản xuất khác nhau. Các lệnh trên máy cấp 2 được thực thi bằng cách thông dịch bởi vi chương trình mà không phải thực thi trực tiếp bằng phần cứng.

Cấp thứ 3 thường là cấp hỗn hợp. Hầu hết các lệnh trong ngôn ngữ của cấp máy này cũng có trong ngôn ngữ cấp 2 và đồng thời có thêm một tập lệnh mới, một tổ chức bộ nhớ khác và khả năng chạy 2 hay nhiều chương trình song song. Các lệnh mới thêm vào sẽ được thực thi bằng một trình thông dịch chạy trên cấp 2, gọi là hệ điều hành. Nhiều lệnh cấp 3 được thực thi trực tiếp do vi chương trình và một số lệnh khác được thông dịch bằng hệ điều hành (do đó, cấp này là cấp hỗn hợp).

Cấp 4 thật sự là dạng tượng trưng cho một trong các ngôn ngữ. Cấp này cung cấp một phương pháp viết chương trình cho các cấp 1, 2, 3 dễ dàng hơn. Các chương trình viết bằng hợp ngữ được dịch sang các ngôn ngữ của cấp 1, 2, 3 và sau đó được thông dịch bằng các máy ảo hay thực tương ứng.

Cấp 5 bao gồm các ngôn ngữ được thiết kế cho người lập trình nhằm giải quyết một vấn đề cụ thể. Các ngôn ngữ này được gọi là cấp cao. Một số ngôn ngữ cấp cao như Basic, C, Cobol, Fortran, Lisp, Prolog, Pascal và các ngôn ngữ lập trình hướng đối tượng như C++, J++, … Các chương trình viết bằng các ngôn ngữ này thường được dịch sang cấp 3 hay 4 bằng các trình biên dịch (compiler).

1.3. Quá trình phát triển của máy nhiều cấp

Các máy tính đầu tiên trong thập niên 40 chỉ có 2 cấp: cấp máy quy ước và cấp logic số. Các lập trình viên phải làm việc trên cấp máy quy ước và chương trình được thực thi trên cấp logic số. Trong thập niên 50, Wikes đề xuất ý tưởng thiết kế máy tính 3 cấp. Máy tính này có một trình thông dịch cài đặt sẵn, không thay đổi, có nhiệm vụ thực thi các chương trình trong cấp máy quy ước. Như vậy, phần cứng chỉ thực thi các vi chương trình với số lệnh giới hạn nên các mạch điện tử cũng đơn giản hơn.

Trình dịch hợp ngữ (assembler) và các trình biên dịch cho ngôn ngữ cấp cao (compiler) phát triển vào những năm 50 tạo điều kiện dễ dàng hơn cho lập trình viên. Tuy nhiên, vào lúc này, lập trình viên phải tự điều hành máy. Vào những năm 60, việc tự động hóa công việc điều hành bắt đầu được thực hiện. Một chương trình gọi là hệ điều hành (operating system) luôn được lưu trữ bên trong máy tính. Lập trình viên cung cấp các thẻ điều khiển và chương trình, chúng sẽ được đọc và thực thi bằng hệ điều hành.

Trong nhiều năm tiếp theo, hệ điều hành càng trở nên phức tạp. Các lệnh, tiện ích và đặc trưng mới được thêm vào cấp máy quy ước cho đến khi xuất hiện một cấp mới. Một số lệnh của cấp mới này giống như cấp máy quy ước nhưng một số lệnh lại hoàn toàn khác, nhất là các lệnh xuất nhập. Vào những năm đầu thập niên 60, các nghiên cứu ở đại học Dartmouth, MIT đã phát triển các hệ điều hành cho phép lập trình viên có thể tác động trực tiếp lên máy tính. Trong các hệ thống này, thiết bị đầu cuối từ xa được nối với máy tính trung tâm qua các đường điện thoại. Một lập trình viên có thể gõ chương trình và nhận kết quả trả về tức thời ở bất cứ nơi nào có thiết bị đầu cuối. Các hệ thống này gọi là hệ thống chia sẻ thời gian (time-sharing system).

2. Phần cứng và phần mềm (Hardware and software)

Các chương trình viết bằng ngôn ngữ máy (cấp 1) được thực thi trực tiếp bằng các mạch điện tử của máy tính, không có trình thông dịch và biên dịch nào can thiệp vào. Các mạch điện tử cùng với bộ nhớ và các thành phần xuất / nhập tạo nên phần cứng máy tính.

Phần cứng bao gồm các mạch tích hợp, các board mạch in, cable, nguồn cung cấp, bộ nhớ, thiết bị đầu cuối, …

Phần mềm bao gồm các giải thuật và các biểu diễn của các giải thuật này gọi là chương trình. Nó chính là tập hợp các lệnh tạo thành một chương trình, chứ không phải là các phương tiện vật lý lưu trữ chúng.

Một dạng trung gian giữa phần mềm và phần cứng gọi là phần dẻo (firmware). Nó chính là thành phần bao gồm phần mềm được đặt vào bên trong các mạch điện tử trong quá trình sản xuất. Phần dẻo được dùng khi chương trình không thay đổi hay hiếm khi phải thay đổi như chương trình điều khiển đặt trong ROM BIOS.

Một thao tác bất kỳ thực thi bằng phần mềm có thể được gắn trực tiếp vào phần cứng và một lệnh bất kỳ thực thi bằng phần cứng cũng có thể được mô phỏng bằng phần mềm. Quyết định đặt một số chức năng vào phần mềm và các chức năng khác vào phần cứng dựa trên các yếu tố giá thành, tốc độ, độ tin cậy. Trên nhiều máy tính đầu tiên, phần cứng và phần mềm được phân biệt rõ ràng. Phần cứng thực hiện vài lệnh đơn giản như cộng và nhảy, các thủ tục khác phải do lập trình viên tự thiết kế. Sau đó, một số thao tác thường xuyên thực thi đòi hỏi các nhà thiết kế hướng đến yêu cầu xây dựng các mạch điện từ thực thi các thao tác này. Kết quả là hình thành xu hướng di chuyển các thao tác theo hướng từ cấp cao xuống cấp thấp hơn. Một số thao tác trước đây được lập trình ở cấp máy quy ước, sau đó được chuyển xuống thực thi ở phần cứng.

Tuy nhiên, khi xuất hiện thế hệ máy tính dùng vi lập trình và thế hệ máy tính nhiều cấp, lại xuất hiện xu hướng ngược lại, nghĩa là di chuyển các thao tác từ cấp thấp lên cấp cao hơn. Ví dụ như lệnh cộng sẽ được thực hiện trực tiếp bằng phần cứng ở các máy trước kia. Đối với máy tính được vi lập trình hóa, lệnh cộng của cấp máy quy ước được thông dịch bằng một vi chương trình chạy trên cấp thấp nhất và được thực thi bằng một chuỗi các bước nhỏ: tìm lệnh, nạp lệnh, xác định lệnh, định vị dữ liệu, tìm và nạp dữ liệu từ bộ nhớ, thực thi phép cộng và lưu trữ kết quả.

Một số đặc trưng trước đây được lập trình ở cấp máy quy ước, sau đó được thực hiện bằng phần cứng hay vi chương trình:

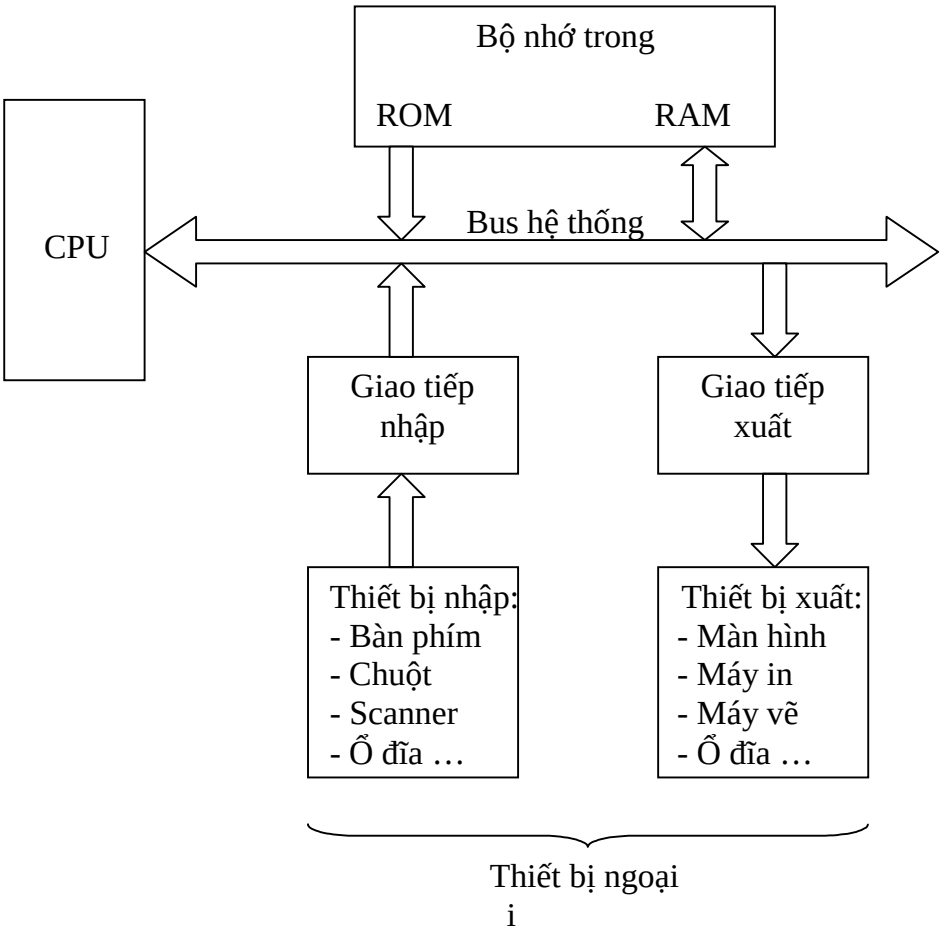
- Các lệnh nhân, chia số nguyên.
- Các lệnh xử lý dấu chấm động.
- Các lệnh gọi thủ tục và quay về từ lệnh gọi thủ tục.
- Các lệnh đếm.
- Các lệnh quản lý chuỗi ký tự.
- Các đặc trưng làm tăng tốc độ tính toán chuỗi: định địa chỉ chỉ số và định địa chỉ gián tiếp.
- Các đặc trưng cho phép chương trình di chuyển trong bộ nhớ sau khi đã thực thi (cấp phát lại bộ nhớ).
- Các xung clock cho thủ tục định thời.
- Các ngắt báo hiệu cho máy tính.

- Khả năng chuyển đổi quá trình.

Như vậy, ta thấy ranh giới giữa phần cứng và phần mềm là không nhất định và thường xuyên thay đổi. Theo quan điểm của lập trình viên, cách thức thực thi một lệnh là không quan trọng, ngoại trừ tốc độ thực thi. Như vậy, phần cứng của người này có thể là phần mềm của người kia.Từ đó dẫn đến ý tưởng thiết kế máy tính có cấu trúc (structured computer). Đó là cấu trúc một máy tính thành một chuỗi các cấp, lập trình viên làm việc trên cấp n không quan tâm đến các cấp khác.

3. Tổ chức hệ thống máy tính

3.1. Cấu trúc một hệ thống máy tính

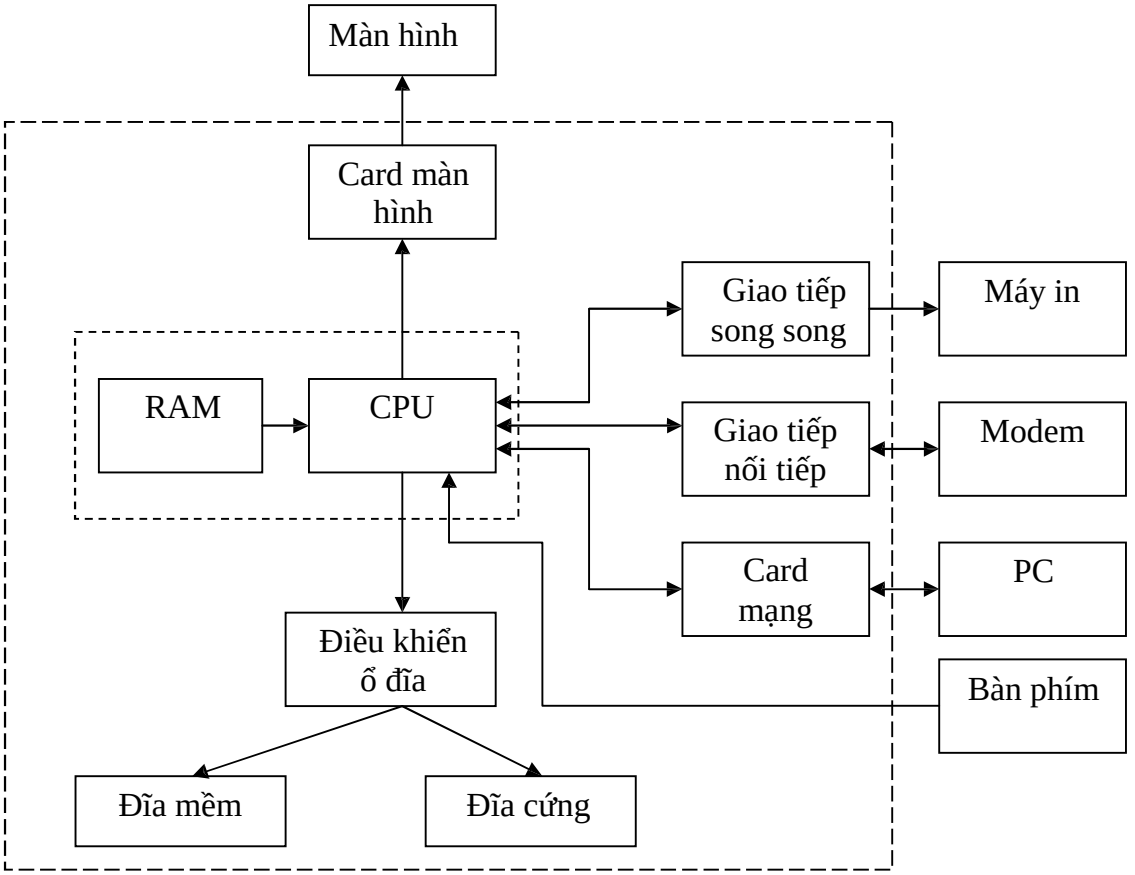


Hình 1.3 – Sơ đồ khối một hệ thống máy tính

Sơ đồ khối của một hệ thống máy vi tính có thể mô tả như hình vẽ. Nó bao gồm các khối:

- ***Khối xử lý trung tâm (CPU – Central Processing Unit)***: nhận và thực thi các lệnh. Bên trong CPU gồm các mạch điều khiển logic, mạch tính toán số học, …
- ***Bộ nhớ (Memory)***: lưu trữ các lệnh và dữ liệu. Nó bao gồm 2 loại: bộ nhớ trong và bộ nhớ ngoài. Bộ nhớ thường được chia thành các ô nhớ nhỏ. Mỗi ô nhớ được gán một địa chỉ để CPU có thể định vị khi cần đọc hay ghi dữ liệu.
- ***Thiết bị ngoại vi (Input / Output)***: dùng để nhập hay xuất dữ liệu. Bàn phím, chuột, scanner, … thuộc thiết bị nhập; màn hình, máy in, … thuộc thiết bị xuất. Các ổ đĩa thuộc bộ nhớ ngoài cũng có thể coi vừa là thiết bị xuất vừa là thiết bị nhập. Các thiết bị ngoại vi liên hệ với CPU qua các mạch giao tiếp I/O (I/O interface)/
- ***Bus hệ thống***: tập hợp các đường dây để CPU có thể liên kết với các bộ phận khác.

3.2. Hoạt động của máy tính



Hình 1.4 – Sơ đồ khối một PC với các thiết bị ngoại vi

CPU được nối với các thành phần khác bằng bus hệ thống nghĩa là sẽ có nhiều thiết bị cùng dùng chung một hệ thống dây dẫn để trao đổi dữ liệu. Do đó, để hệ thống không bị xung đột, CPU phải xử lý sao cho trong một thời điểm, chỉ có một thiết bị hay ô nhớ đã chỉ định mới có thể chiếm dụng bus hệ thống. Do mục đích này, bus hệ thống bao gồm 3 loại:

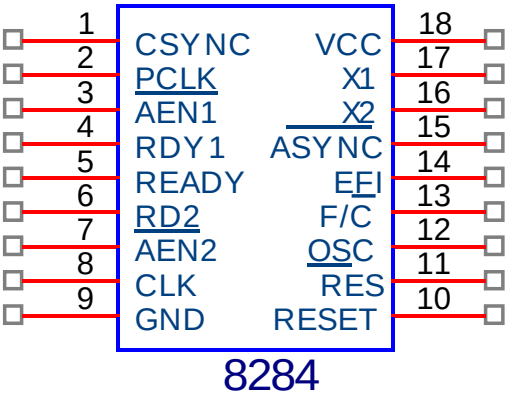
- Bus dữ liệu (data bus): truyền tải dữ liệu
- Bus địa chỉ (address bus): chọn ô nhớ hay thiết bị ngoại vi
- Bus điều khiển (control bus): hỗ trợ trao đổi thông tin trạng thái như phân biệt CPU phải truy xuất bộ nhớ hay ngoại vị, thao tác xử lý là đọc/ghi, …

CPU phát tín hiệu địa chỉ của thiết bị lên bus địa chỉ. Tín hiệu này được dưa vào mạch giải mã địa chỉ chọn thiết bị. Bộ giải mã sẽ phát ra chỉ một tín hiệu chọn chip đúng sẽ cho phép mở bộ đệm của thiết bị cần thiết, dữ liệu lúc này sẽ được trao đổi giữa CPU và thiết bị. Trong quá trình này, các tín hiệu điều khiển cũng được phát trên control bus để xác định mục đích của quá trình truy xuất.

3.3. Các chip hỗ trợ

3.3.1. Mạch tạo xung clock 8284

Mạch tạo xung clock dùng để cung cấp xung clock cho CPU.



Hình 1.5 – Mạch tạo xung clock 8284

CSYNC (Clock Synchronisation): ngõ vào xung đồng bộ chung khi hệ thống có các 8284 dùng dao động ngoài tại chân EFI. Khi dùng mạch dao động trong thì phải nối GND.

PCLK (Peripheral Clock): xung clock f = $f_X/6$ (f_X là tần số thạch anh) với chu kỳ bổn phận 50%.

$\overline{AEN1}$, $\overline{AEN2}$ (Address Enable): cho phép chọn các chân tương ứng RDY1, RDY2 báo hiệu trạng thái sẵn sàng của bộ nhớ hay thiết bị ngoại vi.

RDY1, RDY2 (Bus ready): kết hợp với $\overline{AEN1}$, $\overline{AEN2}$ tạo các chu kỳ đợi ở CPU

READY: nối đến chân READY của µP.

CLK (Clock): xung clock f = $f_X/3$, nối với chân CLK của CPU.

RESET: nối với chân RESET của CPU, là tín hiệu khởi động lại toàn hệ thống.

$\overline{\textbf{\textit{RES}}}$ (Reset Input): chân khởi động cho 8284, được nối với mạch RC để tự khởi động khi bật nguồn.

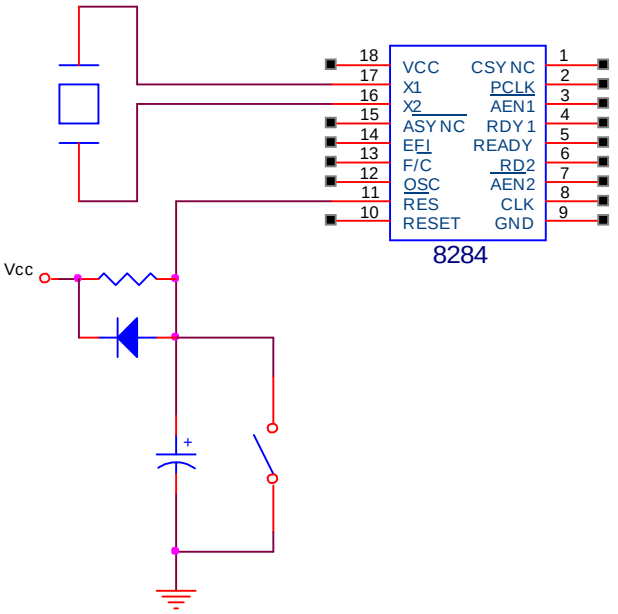
OSC: ngõ ra xung clock có tần số f_X.

F/ $\overline{\textbf{\textit{C}}}$ (Frequency / Crystal): chọn nguồn tín hiệu chuẩn cho 8284, nếu ở mức cao thì chọn tần số xung clock bên ngoài, ngược lại thì dùng xung clock từ thạch anh.

EFI (External Frequency Input): xung clock từ bộ dao động ngoài.

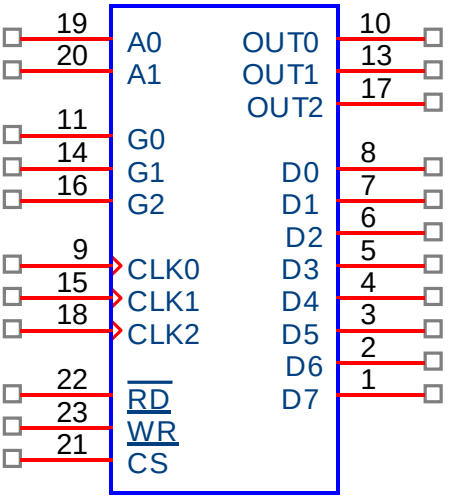
$\overline{\textbf{\textit{ASYNC}}}$: chọn chế độ làm việc cho tín hiệu RDY. Nếu $\overline{\text{ASYNC}}$ = 1, tín hiệu RDY có ảnh hưởng đến tín hiệu READY cho đến khi có xung âm của xung clock. Ngược lại thì RDY chỉ ảnh hưởng khi xuất hiện xung âm.

X1,X2: ngõ vào của thạch anh, dùng để tạo xung chuẩn cho hệ thống.

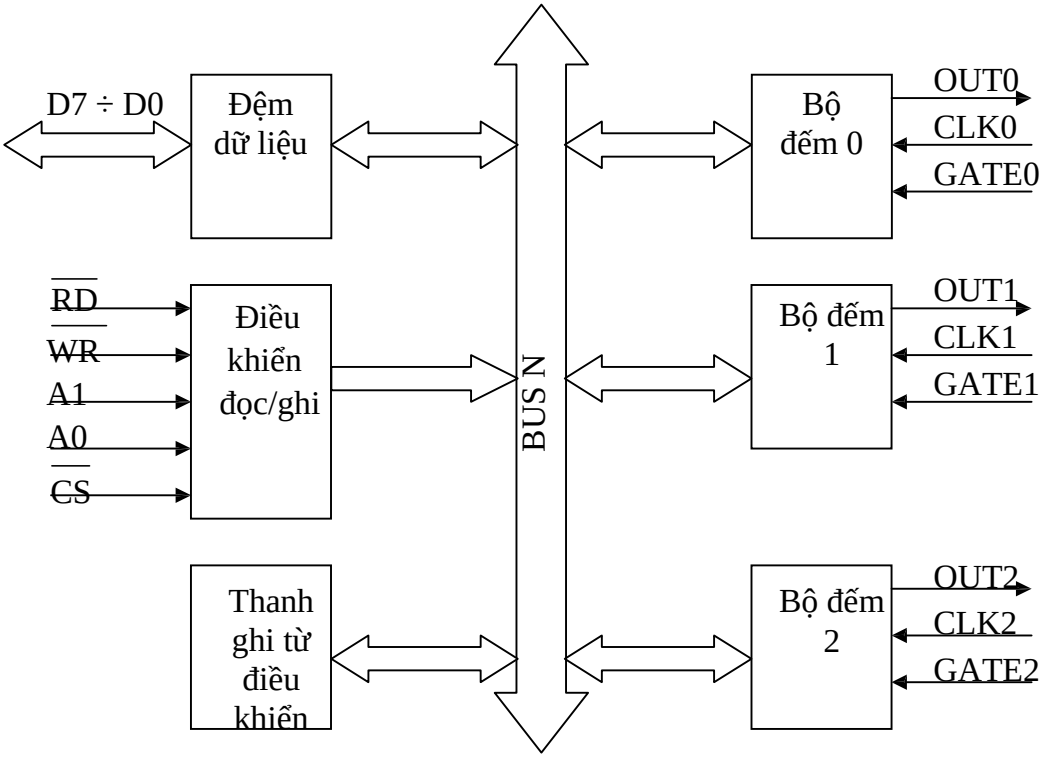


Hình 1.6 – Mạch khởi động cho 8284

3.3.2. Mạch định thời PIT – 8253 / 8254 (Programmable Interval Timer)



Hình 1.7 – Sơ đồ chân của PIT 8253



Hình 1.8 – Sơ đồ khối của PIT 8253

D7 ÷ D0: bus dữ liệu

CLK0 ÷ CLK2: ngõ vào xung clock cho các bộ đếm

OUT0 ÷ OUT2: ngõ ra bộ đếm

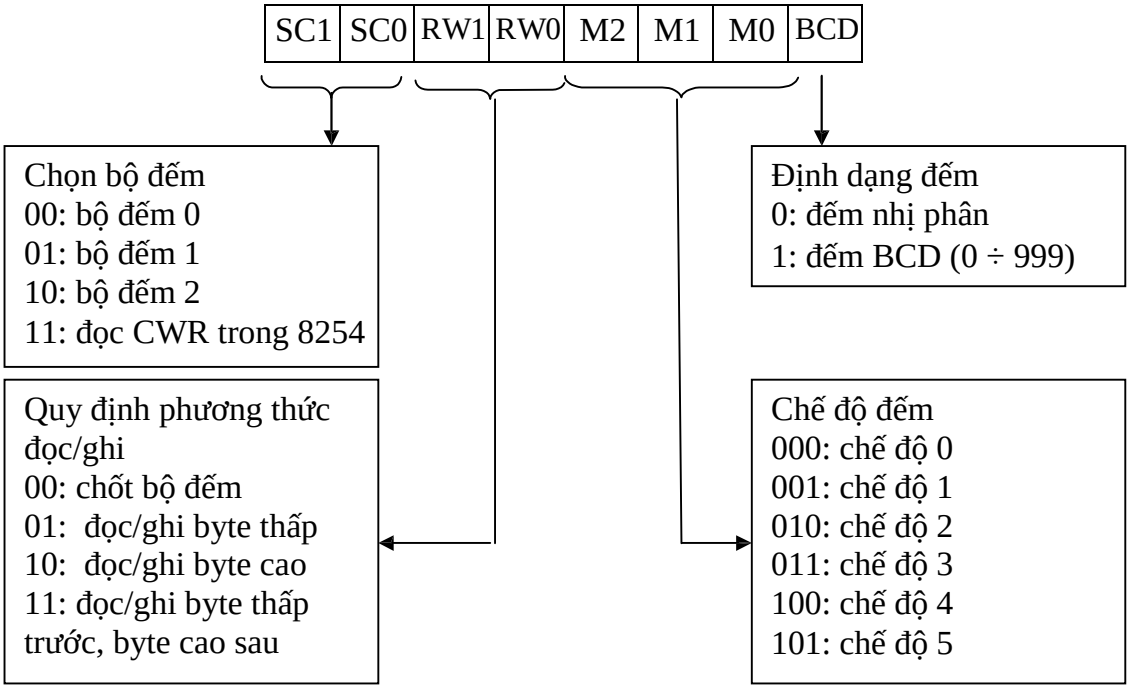
$\overline{RD}$, $\overline{WR}$: cho phép CPU đọc / ghi dữ liệu từ / đến các thanh ghi của 8253

A1, A0: giải mã chọn bộ đếm hay thanh ghi điều khiển, thường được nối với bus địa chỉ của CPU

A1	A0	Chọn
0	0	Bộ đếm 0
0	1	Bộ đếm 1
1	0	Bộ đếm 2
1	1	Thanh ghi từ điều khiển

G0 ÷ ***G2*** (Gate): cho phép hay cấm các bộ đếm hoạt động (=1: cho phép, =0: cấm).

PIT 8253 có tất cả 5 chế độ đếm tùy thuộc vào giá trị trong thanh ghi điều khiển.



Hình 1.9 – Dạng từ điều khiển của 8253

PIT 8253 có 3 bộ đếm lùi 16 bit có thể lập trình và độc lập với nhau. Mỗi bộ đếm có tín hiệu xung clock riêng (8254 tương tự như 8253 nhưng có thêm lệnh đọc thanh ghi từ điều khiển CWR). Địa chỉ các thanh ghi của PIT đối với PC là:

Port (1)	Port (2)	Thanh ghi
40h	48h	Bộ đếm 0
41h	49h	Bộ đếm 1
42h	4Ah	Bộ đếm 2
43h	4Bh	CWR

Các chế độ đếm:

Chế độ 0 (Interrupt on Terminal Count): tín hiệu ngõ ra ở mức thấp cho tới khi bộ đếm tràn thì sẽ chuyển lên mức cao.

Chế độ 1 (Programmable Monoflop): tín hiệu ngõ ra chuyển xuống mức thấp tại cạnh âm của xung clock đầu tiên và sẽ chuyển lên mức cao khi bộ đếm kết thúc.

Chế độ 2 (Rate Generator): tín hiệu ngõ ra xuống mức thấp trong chu kỳ đầu tiên và sau đó chuyển lên mức cao trong các chu kỳ còn lại.

Chế độ 3 (Square-Wave Generator): tương tự như chế độ 2 nhưng xung ngõ ra là sóng vuông khi giá trị đếm chẵn và sẽ thêm một chu kỳ ở mức cao khi giá trị đếm lẻ.

Chế độ 4 (Software-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà sẽ đếm ngay khi số đếm ban đầu được nạp. Ngõ ra ở mức cao để đếm và xuống mức thấp trong chu kỳ xung đếm. Sau đó, ngõ ra sẽ trở lại mức cao.

Chế độ 5 (Hardware-triggered Pulse): giống như chế độ 2 nhưng xung Gate không khởi động quá trình đếm mà được khởi động bằng cạnh dương của xung clock ngõ vào. Ngõ ra ở mức cao và xuống mức thấp sau một chu kỳ clock khi quá trình đếm kết thúc.

Ba chức năng của 8253 trong PC:

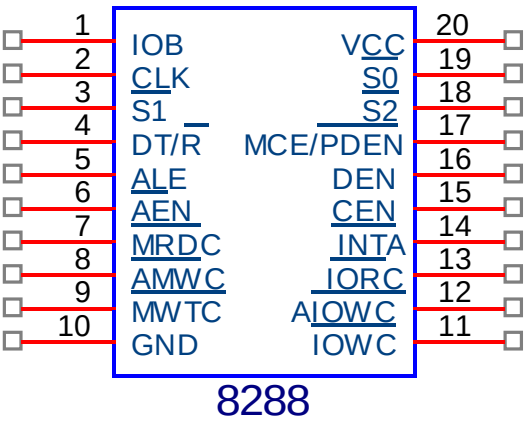
Cập nhật đồng hồ hệ thống: bộ đếm 0 của PIT phát tuần hoàn một ngắt cứng qua IRQ0 của 8259 để CPU có thể thay đổi đồng hồ hệ thống. Bộ đếm hoạt động trong chế độ 2. Ngõ vào được cấp xung clock tần số 1.19318 MHz. G0 = 1 để bộ đếm luôn được phép đếm. Giá trị ban đầu được nạp là 0 cho phép PIT phát ra xung chính xác với tần số:1.19318/65536 = 18.206Hz. Cạnh dương của mỗi xung này sẽ tạo ra một ngắt cứng trong 8259. Yêu cầu này sẽ dẫn tới ngắt 08h để cập nhật đồng hổ hệ thống 18.206 lần trong 1 giây.

Làm tươi bộ nhớ: PIT nối với chip DMAC dùng làm tươi bộ nhớ DRAM. Bộ đếm 1 sẽ định kỳ kích hoạt kênh 0 của DMAC-8237A để tiến hành 1 chu trình đọc giả làm tươi bộ nhớ. Bộ nhớ 1 hoạt động trong chế độ 3 phát sóng vuông với giá trị nạp ban đầu là 18. Do đó sóng vuông được phát ra có tần số 1,19318 MHz/18 = 66288 Hz (chu kỳ bằng 0.015s). Như vậy cứ sau 15 ms cạnh dương của sóng vuông này sẽ tạo 1 chu kỳ đọc giả để làm tươi bộ nhớ.

Phát sóng âm với tần số biến đổi ra loa của PC: Bộ đếm 2 của PIT được dùng để phát sóng âm ra loa của PC.

3.3.3. Mạch điều khiển bus 8288

Mạch điều khiển bus 8288 lấy một số tín hiệu điều khiển của CPU và cung cấp các tín hiệu điều khiển cần thiết cho hệ vi xử lý.



Hình 1.10 – Mạch điều khiển bus 8288

IOB (Input / Output Bus Mode): điều khiển để 8288 làm việc ở các chế độ bus khác nhau.

CLK (Clock): ngõ vào lấy từ xung clock hệ thống (từ 8284) và dùng để đồng bộ toàn bộ các xung điều khiển đi ra từ mạch 8288.

$\overline{S2}, \overline{S1}, \overline{S0}$: các tín hiệu trạng thái lấy trực tiếp từ CPU. Tuỳ theo các giá trị nhận được mà 8288 sẽ đưa các tín hiệu theo bảng:

$\overline{S2}$	$\overline{S1}$	$\overline{S0}$	**Tạo tín hiệu**
0	**0**	**0**	$\overline{INTA}$
0	**0**	**1**	$\overline{IORC}$
0	**1**	**0**	$\overline{IOWC}$, $\overline{AIOWC}$
0	**1**	**1**	**Không**
1	**0**	**0**	$\overline{MRDC}$
1	**0**	**1**	$\overline{MRDC}$
1	**1**	**0**	$\overline{MWTC}$, $\overline{AMWC}$
1	**1**	**1**	**Không**

DT/$\overline{R}$ (Data Transmit/Receive): CPU truyền (1) hay nhận (0) dữ liệu.

ALE (Address Latch Enable): tín hiệu cho phép chốt địa chỉ, tín hiệu này thường được nối với chân G của 74573 để điều khiển chốt địa chỉ.

$\overline{AEN}$ (Address Enable): chờ thời gian trễ khoảng 150 ns sẽ tạo các tín hiệu điều khiển ở đầu ra của 8288 để đảm bảo rằng địa chỉ sử dụng đã hợp lệ.

$\overline{MRDC}$ (Memory Read Command): điều khiển đọc bộ nhớ

$\overline{MWTC}$ (Memory Write Command): điều khiển ghi bộ nhớ

$\overline{AMWC}$ (Advanced MWTC),: giống như $\overline{MWTC}$ nhưng hoạt động sớm hơn một chút dùng cho các bộ nhớ chậm đáp ứng kịp tốc độ CPU.

$\overline{IOWC}$ (I/O Write Command): điều khiển ghi ngoại vi

$\overline{AIOWC}$ (Advanced IOWC),: giống như $\overline{IOWC}$ nhưng hoạt động sớm hơn một chút dùng cho các ngoại vi chậm đáp ứng kịp tốc độ CPU.

$\overline{IORC}$ (I/O Read Command): điều khiển đọc ngoại vi

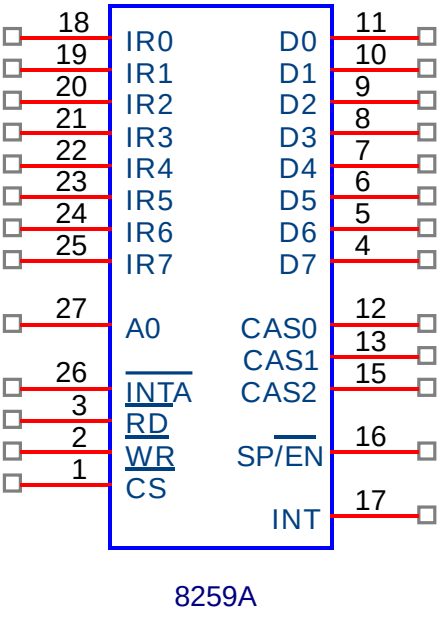
$\overline{INTA}$ (Interrupt Acknowledge): ngõ ra thông báo CPU chấp nhận yêu cầu ngắt của thiết bị ngoại vi

CEN (Command Enable): cho phép đưa ra tín hiệu DEN và các tín hiệu điều khiển khác của 8288.

DEN (Data Enable): điều khiển bus dữ liệu thành bus cục bộ hay bus hệ thống.

MCE / ***$\overline{PDEN}$*** (Master Cascade Enable / Peripheral Data Enable): định chế độ làm việc cho mạch điều khiển ngắt PIC 8259 để nó làm việc ở chế độ master.

3.3.4. Chip điều khiển ngắt ưu tiên PIC 8259A (Priority Interrupt Controller)



Hình 1.11 – Sơ đồ chân của 8259A

Trong trường hợp nhiều yêu cầu ngắt cần phải phục vụ, ta thường dùng vi mạch 8259A để giải quyết vấn đề ưu tiên. 8259A có thể giải quyết được 8 yêu cầu ngắt với 8 mức ưu tiên khác nhau.

Các khối chức năng:

IRR (thanh ghi yêu cầu ngắt): lưu trữ các yêu cầu ngắt tại ngõ vào

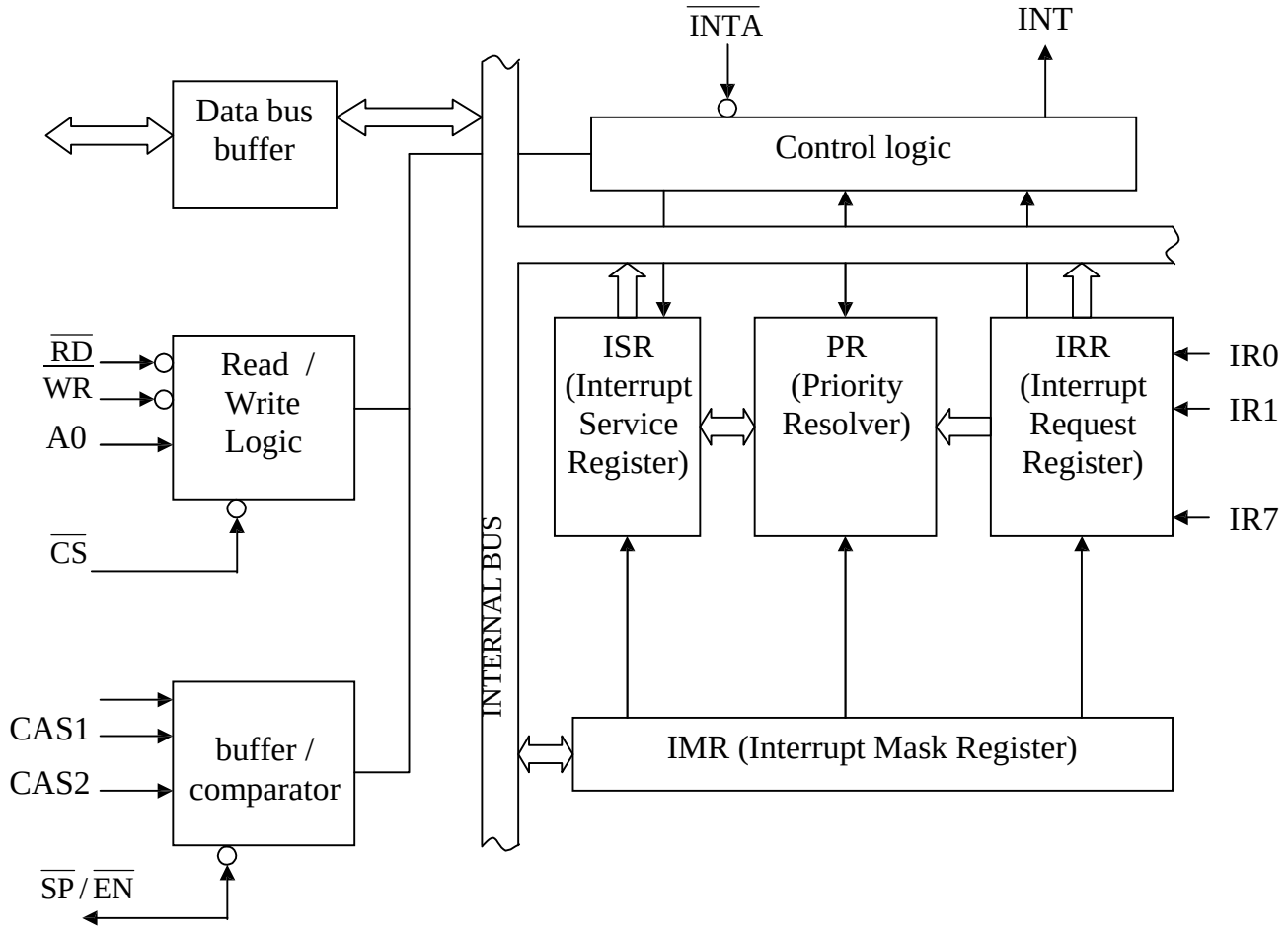
ISR (thanh ghi phục vụ ngắt): lưu trữ các yêu cầu ngắt đang phục vụ

IMR (thanh ghi mặt nạ ngắt): lưu trữ mặt nạ của các yêu cầu ngắt tại ngõ vào

Control logic (logic điều khiển): gởi yêu cầu ngắt tới chân INTR của CPU khi có tín hiệu ngắt tại ngõ vào của 8259A và nhận trả lời chấp nhận yêu cầu ngắt hay không $\overline{\text{INTA}}$ từ CPU để đưa kiểu ngắt vào CPU.

Data bus buffer (đệm bus dữ liệu): giao tiếp giữa 8259A với bus dữ liệu của CPU.

Cascade buffer / comparator (đệm nối tầng và so sánh): lưu trữ và so sánh số hiệu của các kiểu ngắt trong trường hợp dùng nhiều mạch 8259A.



Hình 1.12 – Sơ đồ khối của PIC 8259A

Các tín hiệu điều khiển:

CAS0 ÷ 2 (In, Out): các ngõ vào chọn mạch 8259A tớ (slave) từ mạch 8259A chủ (master) trong trường hợp dùng nhiều mạch 8259A để tăng yêu cầu ngắt.

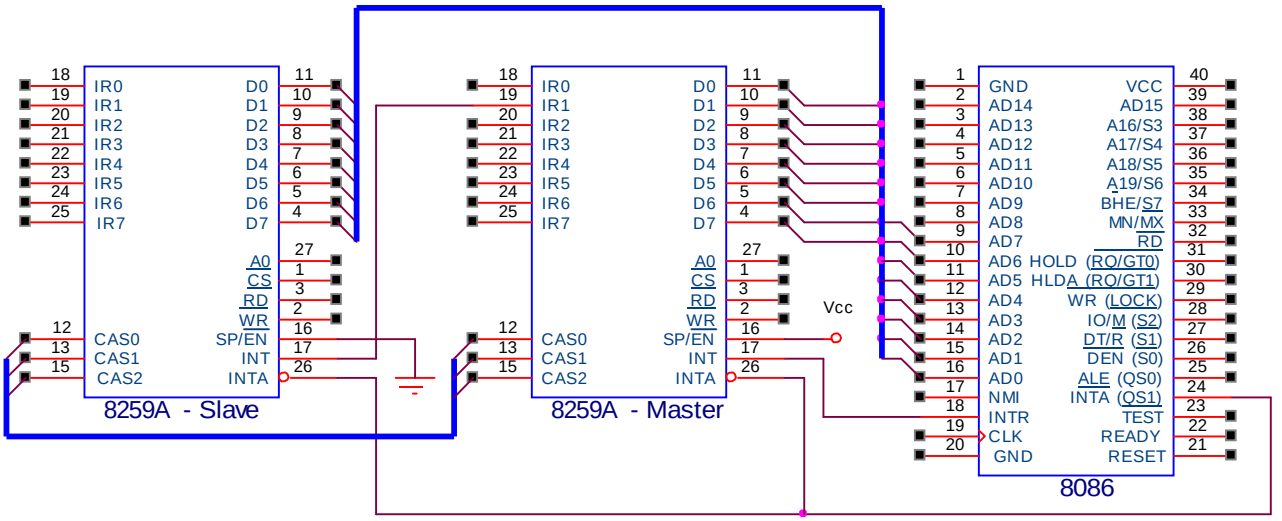
$\overline{SP}$ / $\overline{EN}$ (In, Out) (Slave Program / Enable Buffer): nếu 8259A hoạt động ở chế độ không dùng đệm dữ liệu thì tín hiệu này dùng để xác định mạch 8259A là mạch chủ ($\overline{SP}$ = 1) hay tớ ($\overline{SP}$ = 0). Nếu 8259A hoạt động ở chế độ có đệm dữ liệu thì tín hiệu này dùng để cho phép giao tiếp giữa 8259A và CPU, khi đó mạch 8259A là master hay slave phải dựa vào từ lệnh khởi động ICW4.

INT (Out): tín hiệu yêu cầu ngắt đưa đến CPU (chân INTR).

$\overline{INTA}$ (In): nhận trả lời chấp nhận ngắt hay không từ CPU (chân $\overline{INTA}$)

A0: cho phép chọn các từ điều khiển của 8259A.

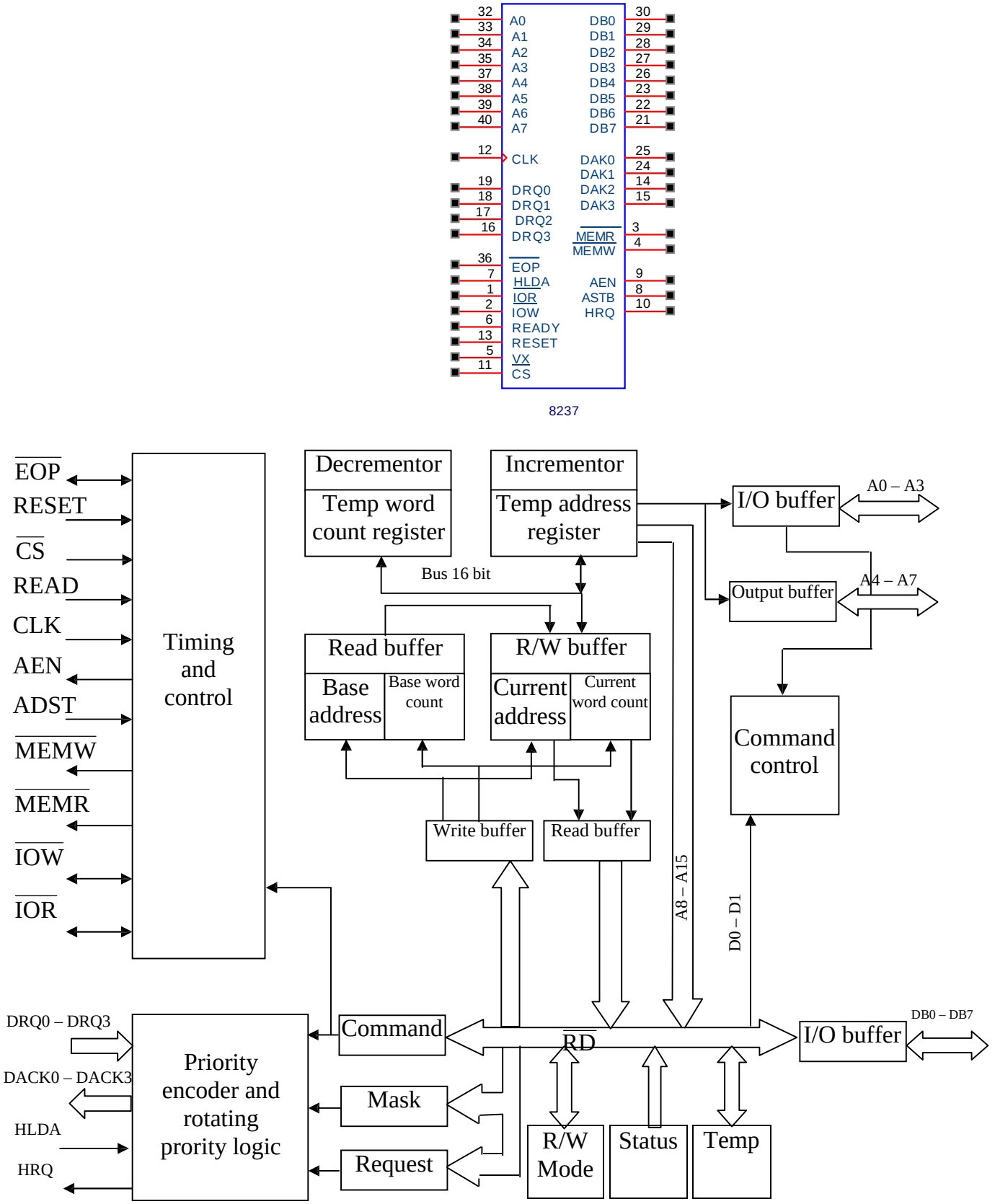
8259A cho phép xử lý 8 ngắt với 8 mức ưu tiên khác nhau. Trong trường hợp hệ thống có số lượng ngắt lớn hơn thì có thể mắc nhiều 8259A liên tầng.



Hình 1.13 – 8259A mắc liên tầng

3.3.5. Chip điều khiển truy nhập bộ nhớ trực tiếp DMAC 8237 (Direct Memory Access Controller)

DMAC 8237 có thể thực hiện truyền dữ liệu theo 3 kiểu: kiểu đọc (từ bộ nhớ ra thiết bị ngoại vi), kiểu ghi (từ thiết bị ngoại vi đến bộ nhớ) và kiểu kiểm tra.



Hình 1.14 – Sơ đồ chân và sơ đồ khối của DMAC 8237A

Khối Timing and Control (định thời và điều khiển):

Tạo các tín hiệu định thời và điều khiển cho bus ngoài (external bus). Các tín hiệu này được đồng bộ với xung clock đưa vào DMAC (tần số xung clock tối đa là 5 MHz).

Khối Priority encoder and rotating priority logic (mã hóa ưu tiên và quay mức ưu tiên):

DMAC 8237A có 2 mô hình ưu tiên: mô hình ***ưu tiên cố định*** (***fixed priority***) và mô hình ***ưu tiên quay*** (***rotating priority***). Trong mô hình ưu tiên cố định, kênh 0 sẽ có mức ưu tiên cao nhất còn kênh 3 có mức ưu tiên thấp nhất. Còn đối với mô hình ưu tiên quay thì mức ưu tiên khi khởi động giống như mô hình ưu tiên cố định nhưng khi yêu cầu DMA tại một kênh nào đó được phục vụ thì sẽ được đặt xuống mức ưu tiên thấp nhất.

Khối Command Control (điều khiển lệnh):

Giải mã các thanh ghi lệnh (xác định thanh ghi sẽ được truy xuất và loại hoạt động cần thực hiện).

Các thanh ghi:

DMAC 8237A có tất cả 12 loại thanh ghi nội khác nhau:

Tên	Kích thước (bit)	Số lượng
Thanh ghi địa chỉ cơ sở (Base Address Register)	16	4
Thanh ghi đếm từ cơ sở (Base Word Count Register) Thanh	16	4
ghi địa chỉ hiện hành (Current Address Register) Thanh ghi	16	4
đếm từ hiện hành (Current Word Count Register) Thanh ghi	16	4
địa chỉ tạm (Temporary Address Register)	16	1
Thanh ghi đếm từ tạm (Temporary Word Count Register)	16	1
Thanh ghi trạng thái (Status Register)	8	1
Thanh ghi lệnh (Command Register)	8	1
Thanh ghi tạm (Temporary Register)	8	1
Thanh ghi chế độ (Mode Register)	6	4
Thanh ghi mặt nạ (Mask Register)	4	1
Thanh ghi yêu cầu (Request Register)	4	1

Chức năng các chân của 8237A:

$\overline{\textbf{\textit{CLK}}}$ (Input): tín hiệu xung clock của mạch. Tín hiệu này thường được lấy từ 8284 sau khi qua cổng đảo.

$\overline{\textbf{\textit{CS}}}$ (Input): thường được nối với bộ giải mã địa chỉ.

RESET (Input): khởi động 8237A, được nối với ngõ RESET của 8284. Khi Reset thì thanh ghi mặt nạ được lập còn các phần sau bị xóa:

+ Thanh ghi lệnh

+ Thanh ghi trạng thái

+ Thanh ghi yêu cầu

+ Thanh ghi tạm

+ Flip-flop đầu/cuối (First/Last flip-flop)

READY (Input): nối với READY của CPU để tạo chu kỳ đợi khi truy xuất các thiết bị ngoại vi hay bộ nhớ chậm.

HLDA (Hold Acknowledge)(Input): tín hiệu chấp nhận yêu cầu treo từ CPU

$\boldsymbol{DRQ_0 - DRQ_3}$ (DMA Request)(Input): các tín hiệu yêu cầu treo từ thiết bị ngoại vi.

DB0 – DB7 (Input, Output): nối đến bus địa chỉ và dữ liệu của CPU

$\overline{\boldsymbol{IOR}}$, $\overline{\boldsymbol{IOW}}$ (Input, Output): sử dụng trong các chu kỳ đọc và ghi

$\overline{\boldsymbol{EOP}}$ (End Of Process)(Input,Output): bắt buộc DMAC kết thúc quá trình DMA nếu là ngõ vào hay dùng để báo cho một kênh biết là dữ liệu đã chuyển xong (Terminal count – TC), thường dùng như yêu cầu ngắt để CPU kết thúc quá trình DMA.

A0 – A3 (Input, Output): chọn các thanh ghi trong 8237A khi lập trình hay dùng để chứa 4 bit địa chỉ thấp.

A4 – A7 (Output): chứa 4 bit địa chỉ

HRQ (Hold Request)(Output): tín hiệu yêu cầu treo đến CPU

$\boldsymbol{DACK_0 - DACK_3}$ (DMA Acknowledge)(Output): tín hiệu trả lời yêu cầu DMA cho các kênh.

AEN (Output): cho phép lấy địa chỉ vùng nhớ cần trao đổi

ADSTB (Address Strobe)(Output): chốt các bit địa chỉ cao A8 – A15 chứa trong các chân DB0 – DB7

$\overline{\boldsymbol{MEMR}}$, $\overline{\boldsymbol{MEMW}}$ (Output): dùng để đọc / ghi bộ nhớ.

Các thanh ghi nội:

Các thanh ghi nội trong DMAC 8237A được truy xuất nhờ các bit địa chỉ thấp A0 – A3.

Bit địa chỉ				Địa chỉ	Chọn chức năng	R/W?
A3	A2	A1	A0			
0	0	0	0	X0	Thanh ghi địa chỉ bộ nhớ kênh 0	R/W
0	0	0	1	X1	Thanh ghi đếm từ kênh 0	R/W
0	0	1	0	X2	Thanh ghi địa chỉ bộ nhớ kênh 1	R/W
0	0	1	1	X3	Thanh ghi đếm từ kênh 1	R/W

0	1	0	0	X4	Thanh ghi địa chỉ bộ nhớ kênh 2	R/W
0	1	0	1	X5	Thanh ghi đếm từ kênh 2	R/W
0	1	1	0	X6	Thanh ghi địa chỉ bộ nhớ kênh 3	R/W
0	1	1	1	X7	Thanh ghi đếm từ kênh 3	R/W
1	0	0	0	X8	Thanh ghi trạng thái / lệnh	R/W
1	0	0	1	X9	Thanh ghi yêu cầu	W
1	0	1	0	XA	Thanh ghi mặt nạ cho một kênh	W
1	0	1	1	XB	Thanh ghi chế độ	W
1	1	0	0	XC	Xóa flip-flop đầu/cuối	W
1	1	0	1	XD	Xóa toàn bộ các thanh ghi / đọc thanh ghi tạm	W/R
1	1	1	0	XE	Xóa thanh ghi mặt nạ	W W
1	1	1	1	XF	Thanh ghi mặt nạ	

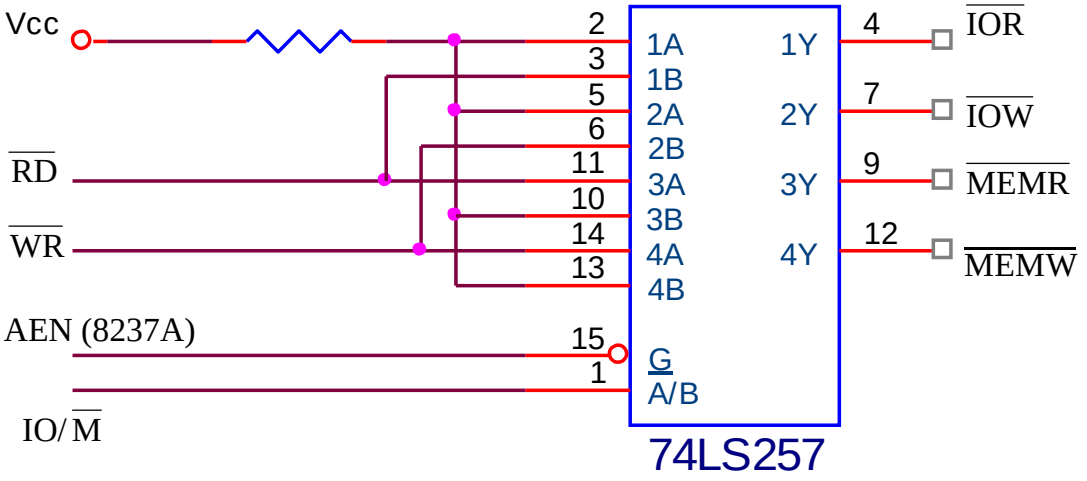
Địa chỉ các thanh ghi nội dùng ghi / đọc địa chỉ:

Kênh	$\overline{\text{IOR}}$	$\overline{\text{OW}}$	**A3**	**A2**	**A1**	**A0**	**Thanh ghi**	**R/W?**
0	1	0	0	0	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	0	0	Địa chỉ hiện hành	R
	1	0	0	0	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	0	1	Bộ đếm hiện hành	R
1	1	0	0	0	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	0	1	0	Địa chỉ hiện hành	R
	1	0	0	0	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	0	1	1	Bộ đếm hiện hành	R
2	1	0	0	1	0	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	0	0	Địa chỉ hiện hành	R
	1	0	0	1	0	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	0	1	Bộ đếm hiện hành	R
3	1	0	0	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành	W
	0	1	0	1	1	0	Địa chỉ hiện hành	R
	1	0	0	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành	W
	0	1	0	1	1	1	Bộ đếm hiện hành	R

Địa chỉ các thanh ghi trạng thái và điều khiển:

$\overline{IOR}$ ‘	$\overline{IOW}$	A3	A2	A1	A0	Thanh ghi
1	0	1	0	0	0	Ghi thanh ghi lệnh
0	1	1	0	0	0	Đọc thanh ghi trạng thái
1	0	1	0	0	1	Ghi thanh ghi yêu cầu
1	0	1	0	1	0	Ghi thanh ghi mặt nạ
1	0	1	0	1	1	Ghi thanh ghi chế độ
1	0	1	1	0	0	Xóa flip-flop đầu/cuối
1	0	1	1	0	1	Xóa tất cả các thanh ghi nội
0	1	1	1	0	1	
1	0	1	1	1	0	Địa chỉ cơ sở và địa chỉ hiện hành
0	1	1	1	1	0	Địa chỉ hiện hành
1	0	1	1	1	1	Bộ đếm cơ sở và bộ đếm hiện hành
0	1	1	1	1	1	Bộ đếm hiện hành

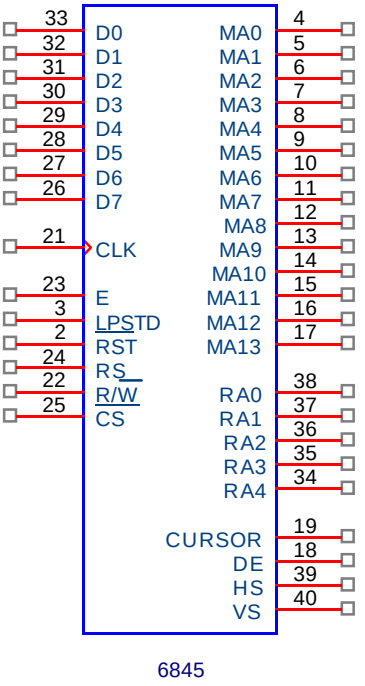
Mạch 8273A-5 chứa 4 kênh trao đổi dữ liệu DMA với mức ưu tiên lập trình được. 8237A-5 có tốc độ truyền 1 MBps cho mỗi kênh và 1 kênh có thể truyền 1 mảng có độ dài 64 KB. Để có thể sử dụng mạch DMAC 8237A, ta cần tạo tín hiệu điều khiển như sau:



Hình 1.15 – Tín hiệu điều khiển cho hệ thống làm việc với DMAC 8237A

Tín hiệu AEN từ 8237A dùng để cấm các tín hiệu điều khiển từ CPU khi DMAC đã nắm quyền điều khiển bus.

3.3.6. Chip điều khiển màn hình CRTC 6845 (Cathode Ray Tube Controller)



Hình 1.16 – Sơ đồ chân của 6845

RST (Reset): khởi động lại 6845.

LPSTD (Light Pen Strobe): lưu trữ địa chỉ hiện hành của RAM màn hình trong thanh ghi bút sáng. CPU đọc thanh ghi và xác định vị trí bút sáng trên màn hình.

MA0 ÷ ***MA13*** (Memory Address): 14 địa chỉ nhớ cho RAM màn hình.

DE (Display Enable): cho phép (=1) hay không (=0) các tín hiệu điều khiển và địa chỉ vùng hiện lên màn hình.

CURSOR: vị trí con trỏ đã quét (=1) hay chưa (=0).

VS (Vertical Synchronization): ngõ ra tín hiệu đồng bộ quét dọc

HS (Horizontal Synchronization): ngõ ra tín hiệu đồng bộ quét ngang

RA0 ÷***RA4*** (Row Address): phân định hàng quét của ký tự trong chế độ văn bản (32 hàng quét). Trong chế độ đồ họa, chúng kết hợp với MA0 ÷ MA13 tạo các địa chỉ cho các bank RAM màn hình.

D0 ÷ ***D7***: đường dữ liệu.

CS : chọn chip.

RS (Regigter Select): chọn thanh ghi địa chỉ (=0) hay thanh ghi dữ liệu (=1).

E: xung âm kích hoạt bus dữ liệu và dùng như xung clock cho 6845 đọc / ghi dữ liệu vào các thanh ghi bên trong.

R/$\overline{W}$: đọc / ghi dữ liệu vào các thanh ghi.

CLK: dùng đồng bộ với tín hiệu của màn hình và thường bằng tốc độ hiện ký tự trên màn hình.

3.3.7. Chip đồng xử lý toán học 8087/80287/80387 (Mathematical co-processor)

Các bộ đồng xử lý toán 80x87 hỗ trợ CPU trong việc tính toán các biểu thức dùng dấu chấm động như cộng, trừ, nhân, chia các số dấu chấm động, căn thức, logarit, … Chúng cho phép xử lý các phép toán này nhanh hơn nhiều so với CPU. Thời gian xử lý giữa 8087 và 8086 như sau (dùng xung clock 8 MHz):

Phép toán	**8087 [μs]**	**8086 [μs]**
Cộng / trừ	10.6	1000
Nhân	11.9	1000
Chia	24.4	2000
Căn bậc hai	22.5	12250
Tang	56.3	8125
Lũy thừa	62.5	10680
Lưu trữ	13.1	750

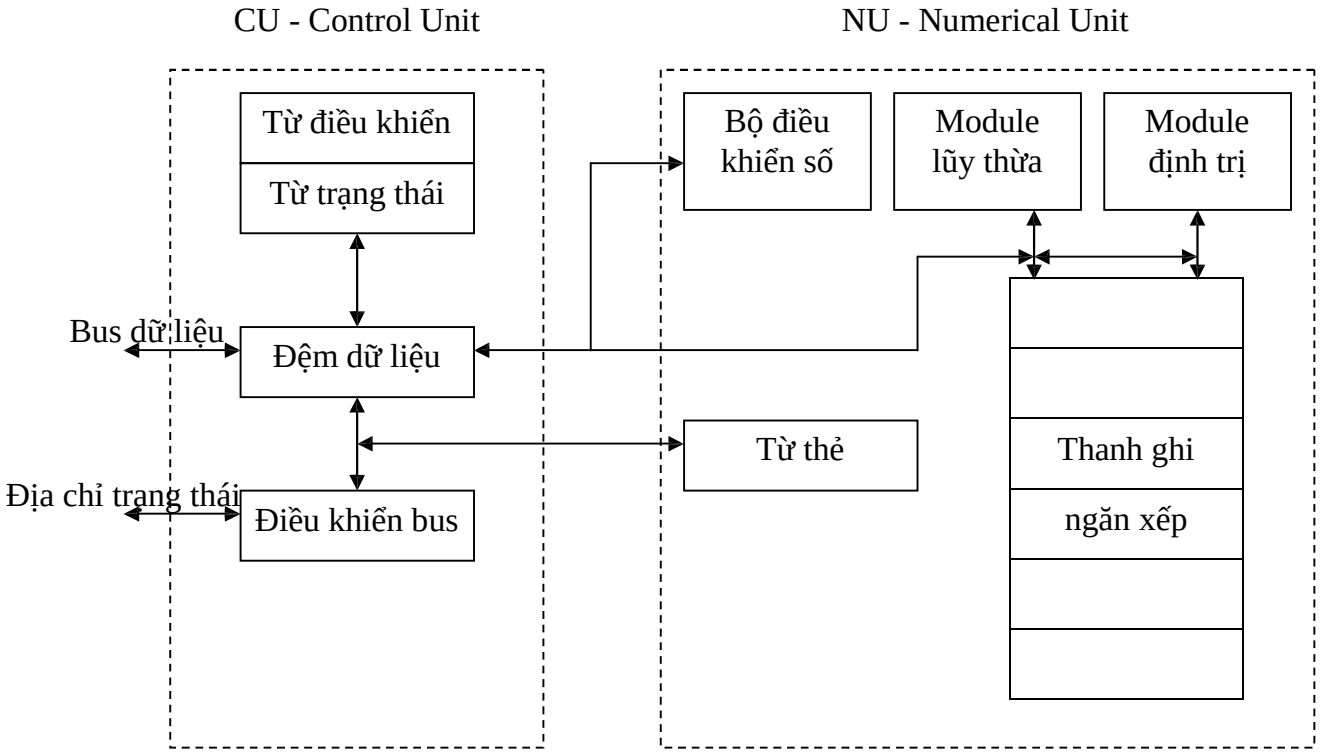
8087:

8087 gồm một đơn vị điều khiển (CU – Control Unit) dùng để điều khiển bus và một đơn vị số học (NU – Numerical Unit) để thực hiện các phép toán dấu chấm động trong các mạch tính lũy thừa (exponent module) và mạch tính phần định trị (mantissa module). Khác với 8086, thay vì dùng các thanh ghi rời rạc là một ngăn xếp thanh ghi.

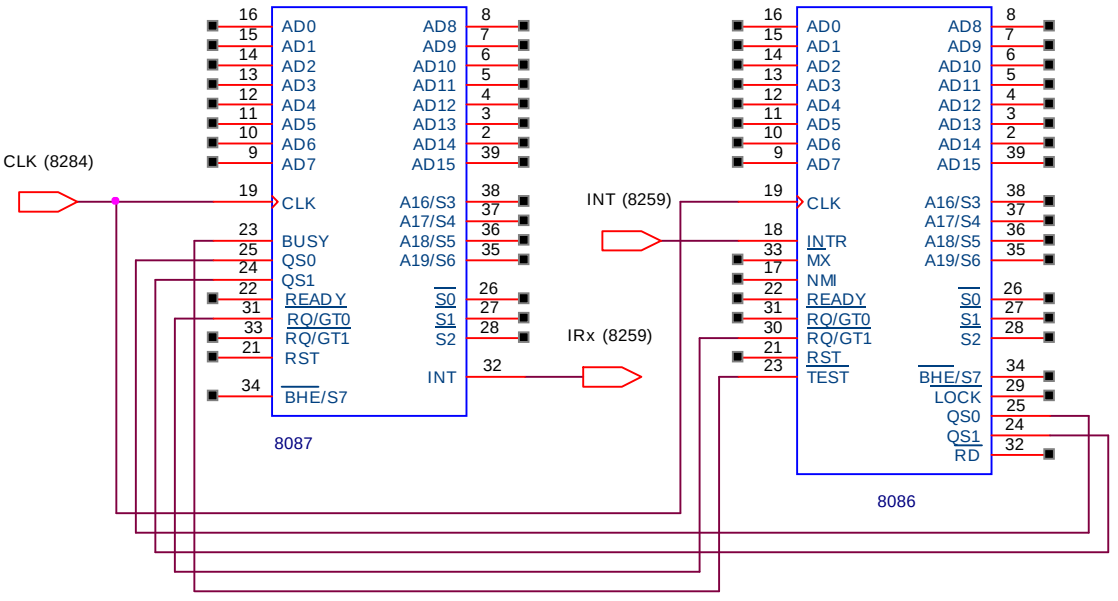
Đơn vị điều khiển nhận và giải mã lệnh, dọc và ghi các toán hạng, chạy các lệnh điều khiển riêng của 8087. Do đó, CU có thể đồng bộ với CPU trong khi NU đang thực hiện các công việc tính toán. CU bao gồm bộ điều khiển bus, bộ đệm dữ liệu và hàng lệnh.

Ngăn xếp thanh ghi có tất cả 8 thanh ghi từ R0 ÷ R7, mỗi thanh ghi dài 80 bit trong đó bit 79 là bit dấu, bit 64 ÷ 78 dùng cho số mũ và phần còn lại là phần định trị. Dữ liệu truyền giữa các thanh ghi này được thực hiện rất nhanh do 8087 có độ rộng bus dữ liệu là 84 bit và không cần phải biến đổi định dạng.

Ngay sau khi reset PC, bộ đồng xử lý kiểm tra xem nó có được nối với PC hay không bằng các đường $\overline{\text{BHE}}$/S7. 8087 sẽ điều chỉnh độ dài của hàng lệnh cho phù hợp với CPU (nếu dùng 8086 thì độ dài là 6 byte).



Hình 1.17 – Sơ đồ khối của 8087



Hình 1.18 – Sơ đồ kết nối 8087 và CPU 8086

8087 có một thanh ghi trạng thái là thanh ghi từ thẻ (tag word) gồm các cặp bit Tag0 ÷ Tag7 để lưu trữ các thông tin liên quan đến nội dung của các thanh ghi R0 ÷ R7 để cho phép thực hiện một số tác vụ nhanh hơn. Mỗi thanh ghi từ thẻ có 2 bit xác định 4 giá trị khác nhau của các thanh ghi Ri.

Tag = 00: xác định

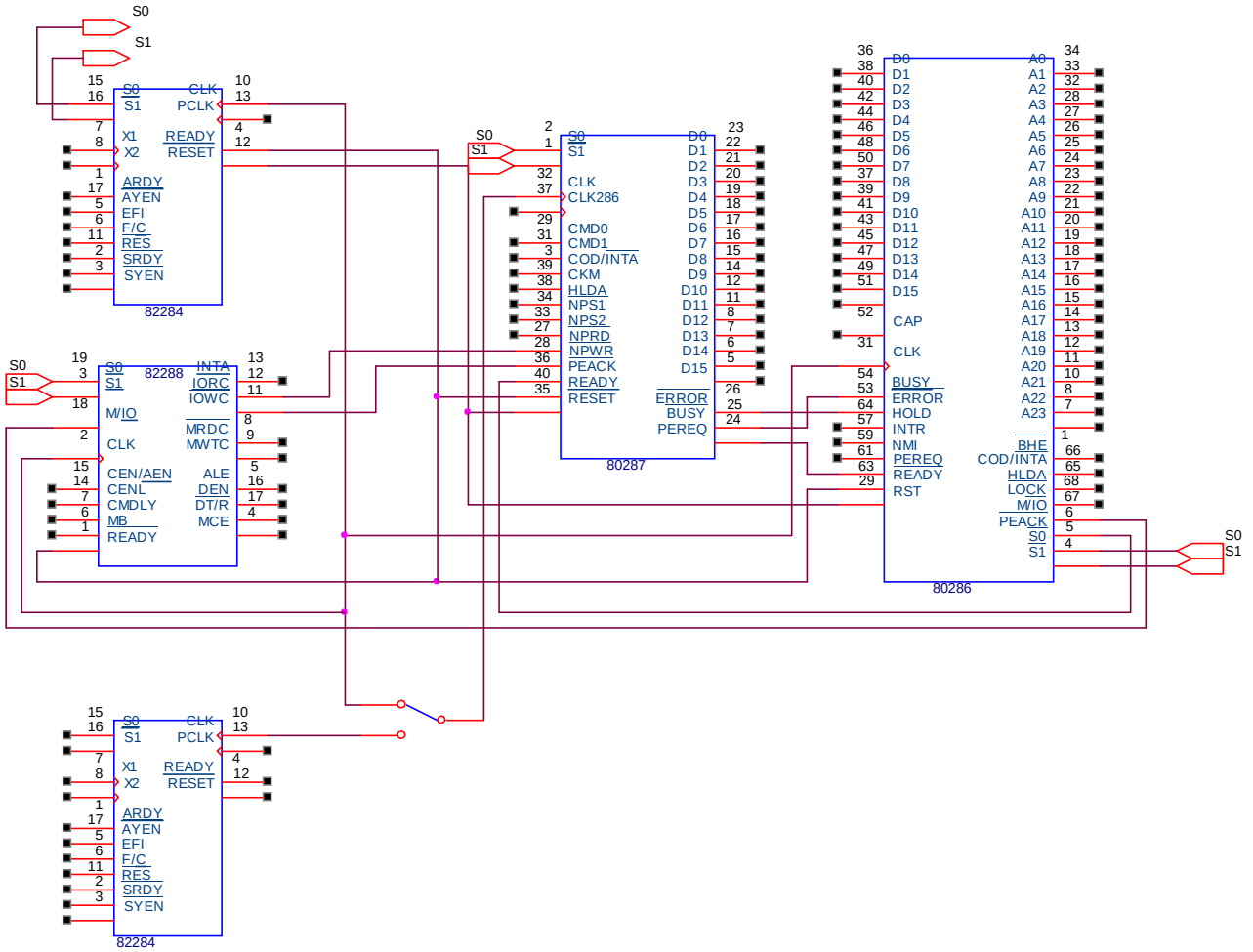
Tag = 01: zero

Tag = 10: NAN, giá trị bất thường

Tag = 11: rỗng

80287:

Do 80286 có chế độ mạch bảo vệ nên mạch ghép nối giữa 80286 và 80287 được thiết kế khác 8087 ở đơn vị điều khiển CU. Bộ đồng xử lý ở đây không thực hiện truy xuất bộ nhớ trực tiếp. Để truy xuất được bộ nhớ, 80287 không những cần một đơc vị định địa chỉ đơn giản của nó mà còn phải được tăng cường thêm chức năng quản lý bộ nhớ của 80286. Cấu trúc bên trong của 80287 cũng tương tự như 8087, chỉ có đơn vị bus thay đổi cho phù hợp với 80286.



Hình 1.19 – Sơ đồ kết nối giữa 80286 và 80287

Khác vơi 8087, 80287 hoạt động không đồng bộ với CPU nên có thể dùng xung clock riêng.

80387:

Ưu điểm của 80387 so với 80287 là có thể thực hiện các phép toán số học nhanh hơn. No có bus dữ liệu 32 bit như CPU và sử dụng công nghệ CMOS nên công suất tiêu thụ thấp hơn.

4. Các thế hệ máy tính

4.1. Máy tính cơ khí

Năm 1942, nhà khoa học Pháp Blaise Pascal xây dựng một máy đầu tiên thực hiện công việc tính toán. Đây là thiết bị hoàn toàn bằng cơ khí sử dụng các bánh răng và cung cấp lực bằng một cánh tay quay. Nó chỉ thực hiện được các phép toán cộng và trừ. 30 năm sau, nhà toán học Đức Baron Gottfried Wilherm von Leibniz xây dựng một máy cơ khí làm được phép nhân và chia.

Sau đó, giáo sư Charles Babbage đã thiết kế và xây dựng máy sai phân (difference engine). Nó được thiết kế để chạy một giải thuật đơn: phương pháp sai phân hữu hạn sử dụng các đa thức và cũng chỉ thực hiện các phép toán cộng và trừ. Năm 1834, Babbage thiết kế và xây dựng máy phân tích (analytical engine). Máy phân tích có 4 thành phần: bộ lưu trữ (bộ nhớ), bộ tính toán, thành phần nhập (đầu đọc thẻ đục lỗ) và thành phần xuất (in và đục lỗ). Bộ tính toán có thể nhận các toán hạng từ bộ lưu trữ, thực hiện phép toán cộng, trừ, nhân hay chia chúng và trả kết quả về bộ lưu trữ.

Phát triển tiếp theo của máy phân tích là máy đa năng. Máy đọc lệnh từ các thẻ đục lỗ và thực thi chúng. Bằng cách đục lỗ một chương trình khác trên thẻ nhập, máy phân tích có khả năng thực hiện các tính toán khác. Lập trình viên máy tính đầu tiên là Ada Lovelace đã tạo ra phần mềm cho máy phân tích.

Vào những năm 1930, Konrad Zuse xây dựng một chuỗi các máy tính toán tự động bằng cách sử dụng các relay từ. Sau đó, John Atanasoff và George Stibbitz đã thiết kế các máy tính (calculator). Máy của Atanasoff sử dụng số nhị phân và có các tụ điện làm cho bộ nhớ được làm tươi theo chu kỳ. Tuy nhiên, máy này bị thất bại do công nghệ phần cứng không tương xứng với ý tưởng thiết kế.

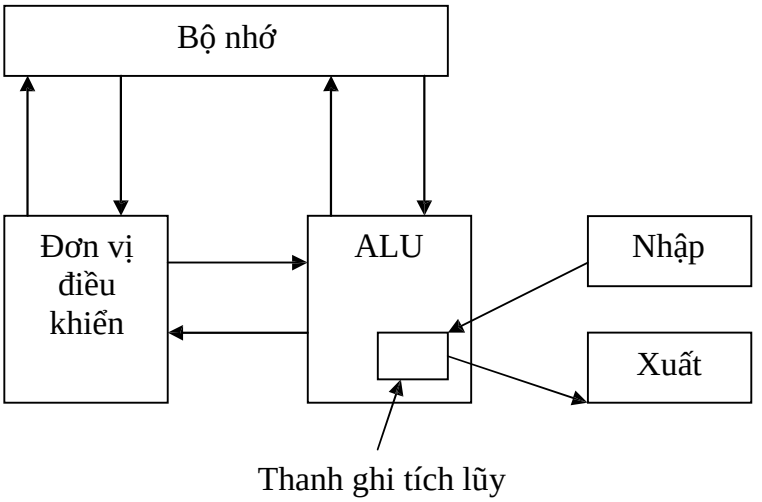
Năm 1944, Aiken hoàn tất máy tính Mark 1, có tất cả 72 từ, mỗi từ 23 số thập phân và có thời gian một chu kỳ là 6 giây. Việc nhập và xuất thực hiện bằng các băng giấy đục lỗ.

4.2. Máy tính đèn điện tử - thế hệ thứ nhất

Năm 1943, máy tính số điện tử đầu tiên trên thế giới bắt đầu hoạt động, máy Colossus. Colossus do Alan Turing thiết kế nhằm thực hiện giải mã các thông điệp đã mã hóa trong chiến tranh thế giới thứ 2. Cũng trong năm 1943, Mauchley và Presper Eckert bắt đầu tiến hành xây dựng máy tính ENIAC (Electronic Numerical Integrator And Computer). ENIAC gồm 1800 đèn điện tử và 1500 relay, cân nặng 30 tấn, công suất tiêu

thụ 140 kWh. Nó có tất cả 20 thanh ghi, mỗi thanh ghi có thể lưu trữ một số thập phân 10 chữ số.

Sau đó, John von Neumann thiết kế máy IAS dựa cơ sở trên máy EDVAC, là một phiên bản nâng cao của ENIAC. Máy von Neumman có 5 phần cơ bản: bộ nhớ, đơn vị luận lý số học (ALU – Arithmetich Logic Unit), đơn vị điều khiển chương trình, thiết bị nhập và thiết bị xuất. Bộ nhớ có tất cả 4096 từ, mỗi từ lưu trữ 40 bit. Mỗi từ chứa 2 lệnh 20 bit hay một số nguyên có dấu 39 bit. Mỗi lệnh 20 bit gồm có 8 bit xác định loại lệnh và 12 bit xác định 1 trong 4096 từ nhớ.



Hình 1.20 – Máy von Neumann

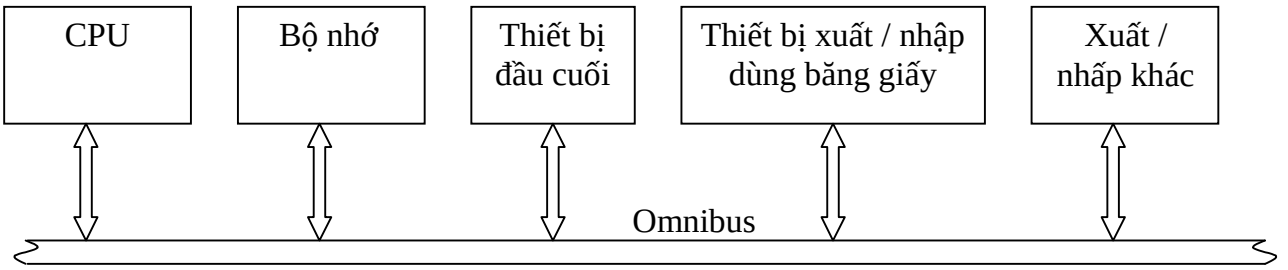
Vào cùng thời gian của máy IAS, các nhà nghiên cứu ở MIT cũng đang xây dựng một máy tính, máy Whirlwind 1. Nó có từ dài 16 bit và thiết kế để điều khiển thời gian thực.

4.3. Máy tính transistor – thế hệ thứ hai

Năm 1948, John Bardeen, Walter Brattain và William Shockley phát minh ra transistor đã làm cuộc cách mạng trong lĩnh vực máy tính. Máy tính transistor đầu tiên được xây dựng tại MIT, máy TX-0 (Transistorized experimental computer 0), có 16 bit tương tự như Whirlwind 1.

Năm 1961, máy tính PDP-1 xuất hiện có 4K từ 18 bit và khoảng thời gian một chu kỳ là 5 μs. Vài năm sau, PDP-8 ra đời có 12 bit nhưng giá thành rẻ hơn PDP-1 rất nhiều (16.000 USD so với 120.000 USD). PDP-8 có một đổi mới đó là hình thành một bus đơn gọi là omnibus trong đó bus là tập hợp các dây nối song song dùng để kết nối các thành phần của máy tính.

Trong khi đó, IBM xây dựng một phiên bản của 709 bằng transistor, đó là máy tính 7094 có thời gian một chu kỳ là 2 µs và bộ nhớ 32K từ 36 bit. Năm 1964, công ty CDC giới thiệu máy 6600 có tốc độ nhanh hơn 7094 do bên trong CPU có một cơ chế song song. CPU có vài đơn vị thực hiện phép cộng, các đơn vị khác thực hiện phép nhân, chia và tất cả chúng đều hoạt động song song. Với một công việc, máy có khả năng thực thi 10 lệnh đồng thời.



Hình 1.21 – Omnibus của PDP-8

4.4. Máy tính IC – thế hệ thứ ba

Vi mạch được phát minh cho phép đặt vài chục transistor trong một chip đơn. Việc này giúp cho các máy tính xây dựng trên IC nhỏ hơn, nhanh hơn và rẻ hơn so với các máy tính transistor. Lúc này, IBM giới thiệu một sản phẩm đơn, máy System 360, được thiết kế dựa trên các vi mạch. Đổi mới quan trọng trong 360 là khả năng đa lập trình (multiprogramming), có vài chương trình trong bộ nhớ đồng thời để khi một chương trình đang chờ xuất / nhập dữ liệu thì chương trình khác có thể tính toán. Một đặc trưng khác của 360 là không gian địa chỉ lớn (thời điểm lúc đó), với 2^{24} byte nhớ (16 MB).

4.5. Máy tính cá nhân và VLSI – thế hệ thứ tư

Vào thập niên 80, vi mạch VLSI (Very Large Scale Integrate) có khả năng chứa vài chục ngàn, vài trăm ngàn và vài triệu transistor trên một chip đơn đã được chế tạo. Sự phát triển này dẫn đến việc sản xuất các máy tính nhỏ hơn và nhanh hơn. Do đó, giá cả đã giảm xuống đến mức một cá nhân có thể sở hữu một máy tính. Các máy tính cá nhân thường dùng cho việc xử lý từ, các bảng tính và các ứng dụng tương hỗ khác. Các máy tính trong thế hệ này có thể chia thành 5 loại: máy tính cá nhân, máy tính mini, siuê máy tính mini, mainframe, siêu máy tính.

Máy tính mini sử dụng trong các ứng dụng thời gian thực như điều khiển không lưu hay tự động hóa. Siêu máy tính mini dùng trong các hệ thống chia sẻ thời gian, các máy chủ. Mainframe dùng trong các nhóm công việc lớn hay đòi hỏi cơ sở dữ liệu lớn, … Siêu máy tính được thiết kế đặc biệt để cựa đại hóa số các thao tác dấu chấm động trong 1s (FLOP – floating point operations per second). Máy tính nào có tốc độ dưới 1 GF/s thì không được xem là siêu máy tính.

Chương 2
TỔ CHỨC CPU (8086/8088/80286)

1. Định thời chu kỳ bus

Mỗi chu kỳ bus bắt đầu bằng việc xuất địa chỉ bộ nhớ hoặc I/O port (chu kỳ xung nhịp T1). Với 8086 thì địa chỉ này có thể là địa chỉ bộ nhớ 20 bit, địa chỉ I/O gián tiếp 16 bit (thanh ghi DX) hay địa chỉ I/O trực tiếp 8 bit. Bus điều khiển có 4 tín hiệu tác động mức thấp là $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$ và $\overline{\text{IOW}}$.

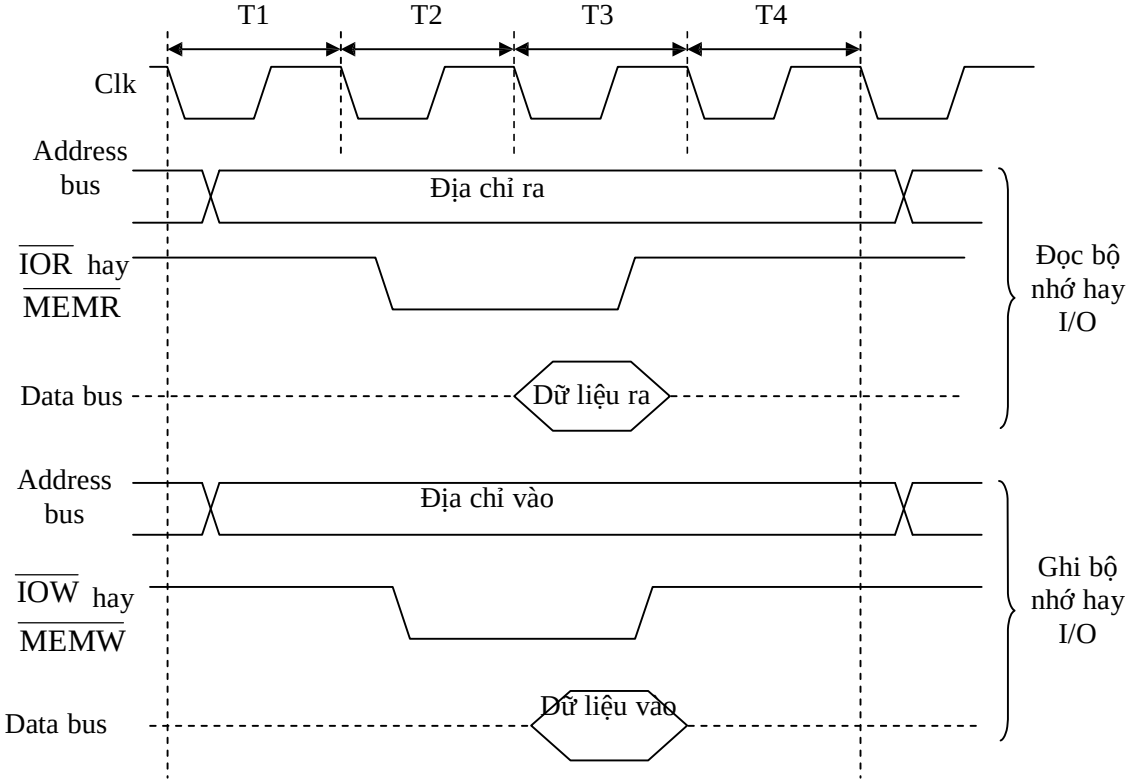
Các chuỗi sự kiện xảy ra trong một chu kỳ bus đọc bộ nhớ:

T1: CPU xuất địa chỉ bộ nhớ. Các đường dữ liệu không hoạt động và các đường điều khiển bị cấm

T2: Đường điều khiển $\overline{\text{MEMR}}$ xuống mức thấp. Đơn vị bộ nhớ ghi nhận chu kỳ bus này là quá trình đọc bộ nhớ và đặt byte hay word có địa chỉ đó lên bus dữ liệu.

T3: CPU đặt cấu hình để các đường bus dữ liệu là nhập. Trạng thái này chủ yếu để bộ nhớ có thời gian tìm kiếm byte hay word dữ liệu

T4: CPU đợi dữ liệu trên bus dữ liệu. Do đó, nó thực hiện chốt bus dữ liệu và giải phóng các đường điều khiển đọc bộ nhớ. Quá trình này sẽ kết thúc chu kỳ bus.



Hình 2.1 – Định thì chu kỳ bus

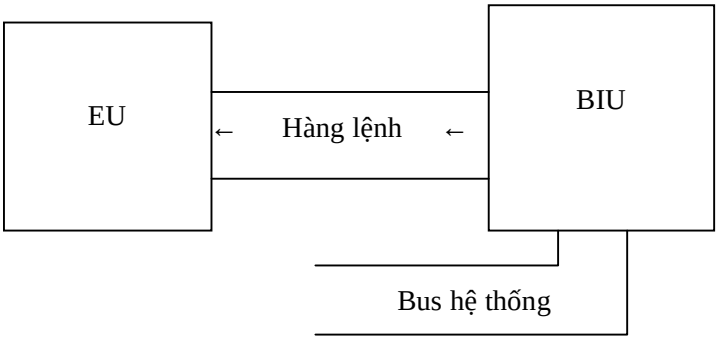
Trong một chu kỳ bus, CPU có thể thực hiện đọc I/O, ghi I/O, đọc bộ nhớ hay ghi bộ nhớ. Các đường bus địa chỉ và bus điều khiển dùng để xác định địa chỉ bộ nhớ hay I/O và hướng truyền dữ liệu trên bus dữ liệu.

Chú ý rằng CPU điều khiển tất cả các quá trình trên nên bộ nhớ bắt buộc phải cung cấp được dữ liệu vào lúc $\overline{\text{MEMR}}$ lên mức cao trong trạng thái T4. Nếu không, CPU sẽ đọc dữ liệu ngẫu nhiên không mong muốn trên bus dữ liệu. Để giải quyết vấn đề này, ta có thể dùng thêm các trạng thái chờ (wait state).

2. Kiến trúc nội

2.1. Kiến trúc nội

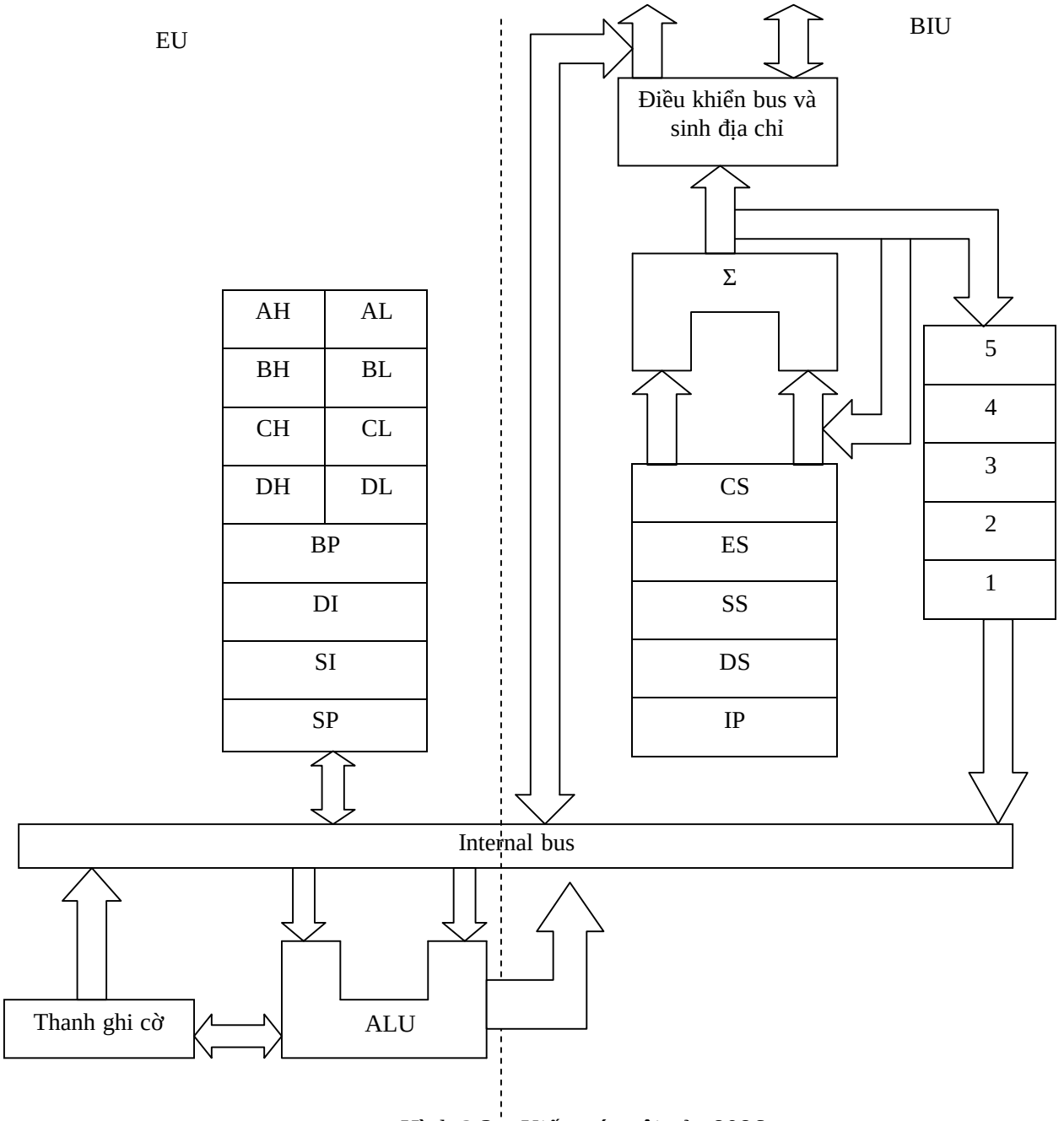
CPU có khả năng thực hiện các tác vụ dữ liệu theo tập lệnh bên trong. Một lệnh được ghi nhận bằng mã đã được định nghĩa trước, gọi là mã lệnh (opcode). Trước khi thực thi một lệnh, CPU phải nhận được mã lệnh từ bộ nhớ chương trình của nó. Quá trình xử lý này gọi là chu kỳ nhận lệnh (fetch cycle). Một khi các mã được nhận và được giải mã thì mạch bên trong CPU có thể tiến hành thực thi (execute) mã lệnh.



Hình 2.2 – Kiến trúc tổng quát của CPU 8086

BIU (Bus Interface Unit – đơn vị giao tiếp bus) nhận các mã lệnh từ bộ nhớ và đặt chúng vào hàng chờ lệnh. EU (Execute Unit – đơn vị thực thi) sẽ giải mã và thực hiện các lệnh trong hàng. Chú ý rằng các đơn vị EU và BIU làm việc độc lập với nhau nên BIU có khả năng đang nhận một lệnh mới trong khi EU dang thực thi lệnh trước đó. Khi EU đã thực hiện xong lệnh, nó sẽ lấy mã lệnh kế tiếp trong hàng lệnh (instruction queue).

Kiến trúc nội của CPU 8086 ở hình 2.3. Nó có 2 bộ xử lý riêng: BIU và EU. BIU cung cấp các chức năng phần cứng, bao gồm tạo các địa chỉ bộ nhớ và I/O để chuyển dữ liệu giữa EU và bên ngoài CPU. EU nhận các mã lệnh chương trình và dữ liệu từ BIU, thực thi các lệnh này và chứa các kết quả trong các thanh ghi. Ngoài ra, dữ liệu cũng có thể chứa trong một vị trí bộ nhớ hay được ghi vào thiết bị xuất. Chú ý rằng EU không có bus hệ thống nên phải thực hiện nhận và xuất tất cả các dữ liệu của nó thông qua BIU. Sự khác biệt giữa CPU 8086 và 8088 là BIU. Trong 8088, đường bus dữ liệu là 8 bit trong khi của 8086 là 16 bit. Ngoài ra hàng lệnh của 8088 dài 4 byte trong khi của 8086 là 6 byte. Tuy nhiên do EU giữa hai loại µP này giống nhau nên ***các chương trình viết cho 8086 có thể chạy được trên 8088 mà không cần thay đổi gì cả***.



Hình 2.3 – Kiến trúc nội của 8086

2.2. Cơ chế đường ống (pipeline)

Quá trình nhận lệnh và thực thi lệnh:

1/ BIU xuất nội dung của thanh ghi con trỏ lệnh IP (Instruction Pointer) ra bus địa chỉ để chọn byte hay word đọc vào BIU.

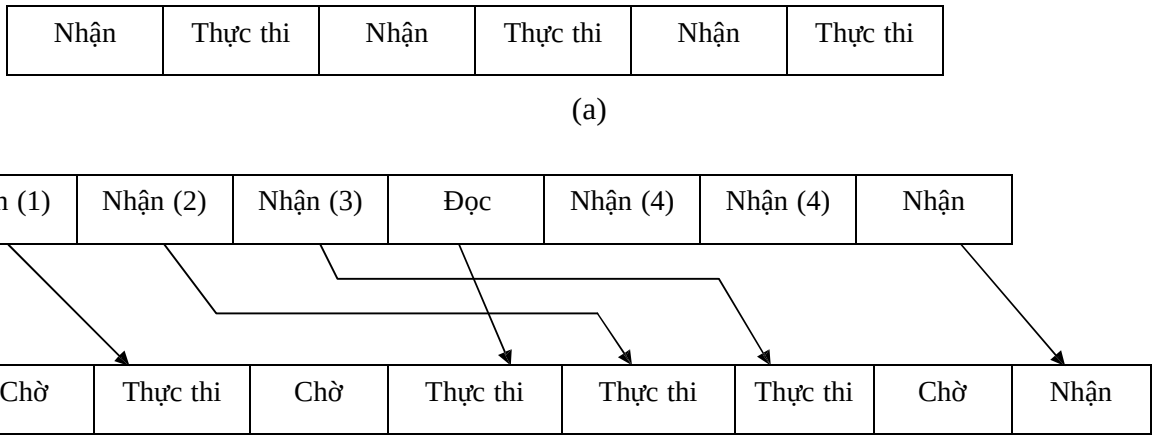
2/ Thanh ghi IP được tăng lên để chuẩn bị nhận lệnh kế (số byte tăng lên của IP tùy thuộc vào kích thước lệnh trước đó).

3/ Khi lệnh ở trong BIU, nó được đưa sang hàng lệnh (queue). Đây là một thanh ghi lưu trữ dạng FIFO (First In First Out – Vào trước ra trước), dùng cơ chế xử lý xen kẽ liên tục các dòng mã lệnh (kỹ thuật đường ống – pipelining).

4/ Giả sử ban đầu hàng lệnh trống, EU sẽ không làm gì cả cho đến khi bắt đầu xuất hiện một lệnh trong hàng, EU sẽ lấy lệnh ra khỏi hàng và bắt đầu thực thi lệnh đó.

5/ Trong khi EU đang thực thi lệnh, BIU tiến hành nhận lệnh mới. Tuỳ theo thời gian thực thi lệnh mà BIU có thể đưa vào hàng lệnh nhiều lệnh mới trước khi EU thực hiện lệnh xong và tiếp tục lấy lệnh mới.

BIU được lập trình để có thể nhận một lệnh mới bất kỳ lúc nào hàng lệnh có chỗ cho 1 byte (8088) hay 2 byte (8086). Lợi ích của phương pháp xử lý theo cơ chế pipeline là EU có thể thực thi các lệnh gần như liên tục thay vì phải đợi BIU nhận thêm lệnh mới.



(1): lệnh thực thi không cần dữ liệu trong hàng
(2): lệnh thực thi cần dữ liệu trong hàng
(3): lệnh nhảy
(4): các lệnh bị bỏ qua do lệnh nhảy

Hình 2.4

(a) CPU thông thường dùng chu kỳ nhận và thực thi lệnh tuần tự
(b) Kiến trúc dạng pipeline của 8086/8088 cho phép thực thi các lệnh mà không bị trễ do quá trình nhận lệnh

Có 3 điều kiện làm cho EU ở chế độ chờ:

- Điều kiện thứ nhất xảy ra khi lệnh cần truy xuất đến một vị trí bộ nhớ không ở trong hàng. BIU phải treo quá trình nhận lệnh và xuất ra địa chỉ của ô nhớ này. Sau khi truy xuất bộ nhớ, EU có thể tiếp tục quá trình thực thi lệnh từ hàng lệnh và BIU có thể tiếp tục đưa các lệnh vào hàng.
- Điều kiện thứ hai xảy ra khi lệnh được thực thi là lệnh nhảy (jump). Trong trường hợp này, thay vì dùng địa chỉ lệnh kế tiếp, ta phải chuyển đến địa chỉ mới (không tuần tự). Tuy nhiên, BIU vẫn luôn đặt các lệnh theo tuần tự và do đó sẽ lưu các lệnh không sử dụng. Trong khi nhận lệnh kế tiếp tại địa chỉ do lệnh jump chỉ đến, EU phải đợi và tất cả các byte trong hàng phải bỏ.

- Điều kiện thứ ba có thể làm BIU treo quá trình nhận lệnh đó là khi thực thi các lệnh có thời gian thực thi lớn. Giả sử như lệnh AAM (ASCII Adjust for Multiplication) cần 83 chu kỳ xung nhịp để hoàn tất trong khi đó với 4 chu kỳ xung nhịp cho quá trình nhận lệnh thì hàng sẽ bị đầy. Như vậy BIU phải đợi cho đến khi lệnh được thực hiện xong và EU nhận mã lệnh từ hàng thì mới có thể tiếp tục quá trình nhận lệnh.

2.3. Cơ chế siêu phân luồng (hyper-threading)

Internet, thương mại điện tử và phần mềm ứng dụng doanh nghiệp đang ngày càng đòi hỏi nhiều năng lực tính toán của các máy chủ hơn. Để nâng cao tốc độ, phần mềm cần phải được phân luồng - các chỉ thị sẽ được chia thành nhiều dòng lệnh để có thể xử lý đồng thời trên nhiều bộ xử lý. Intel đã đưa ra kỹ thuật phân luồng cho phép nâng cao tốc độ và khả năng tính toán song song cho những ứng dụng đa luồng. Công nghệ mới của Intel mô phỏng mỗi bộ vi xử lý vật lý như là hai bộ vi xử lý luận lý (logic), tài nguyên vật lý được chia sẻ và có cấu trúc chung giống hệt nhau cho cả hai bộ xử lý logic. Hệ điều hành và phần mềm ứng dụng sẽ xem như như đang chạy trên hai hay nhiều bộ xử lý, kết quả là tốc độ xử lý trung bình có thể tăng lên xấp xỉ 40% đối với một bộ xử lý vật lý, Intel gọi kỹ thuật này là siêu phân luồng.

Kỹ thuật siêu phân luồng cho phép các phần mềm ứng dụng được viết cho những máy chủ đa luồng có thể thực hiện các chỉ thị song song đồng thời trên mỗi bộ xử lý riêng, bằng cách này sẽ cải thiện tức thì tốc độ giao dịch cũng như thời gian đáp ứng và các yêu cầu đặc thù khác của phần mềm nghiệp vụ và thương mại điện tử. Kỹ thuật này tương thích với các phần mềm ứng dụng và hệ điều hành sẵn có trên các máy chủ (server), nó cho phép hỗ trợ nhiều người dùng hơn và tăng khối lượng công việc được xử lý trên một máy chủ. Với các máy trạm (workstation) cao cấp, kỹ thuật siêu phân luồng cũng sẽ tăng đáng kể tốc độ các phần mềm ứng dụng đòi hỏi năng lực tính toán cao, ví dụ như phần mềm thiết kế 3 chiều, xử lý ảnh hay video… Trong thời gian tới sẽ xuất hiện ngày càng nhiều phần mềm được thiết kế đặc biệt và tối ưu hoá cho Kỹ thuật này.

Từ tháng 01/2002, kỹ thuật siêu phân luồng đã được Intel đưa vào các bộ vi xử lý Xeon đời mới, khởi đầu với các bộ xử lý có tốc độ 1.8GHz và 2.0GHz với 512KB cache thứ cấp, sản xuất bằng công nghệ 0.13 micron (Xeon 1.7GHz, 1.8GHz, 2.0GHz với 256KB cache thứ cấp được sản xuất bằng công nghệ 0.18 không hỗ trợ siêu phân luồng). Tại thời điểm đầu tiên khi Intel giới thiệu bộ xử lý Xeon cùng với chipset 860, chỉ có một số rất ít các nhà sản xuất hàng đầu như IBM, Compaq, Dell, SuperMicro, Tyan… hỗ trợ bộ vi xử lý này, số lượng sản phẩm cũng rất ít. Tuy nhiên, khi có thêm các chipset hỗ trợ bộ xử lý Xeon như E7500 và Serverworks GC, nhiều nhà sản xuất khác đã có sản phẩm hỗ trợ bộ xử lý Xeon. Tuy nhiên đối với đa số người dùng, nhất là người dùng máy tính để bàn (desktop) thì kỹ thuật siêu phân luồng còn khá xa lạ. Intel chỉ chuẩn bị đưa ra bộ xử lý Pentium IV dành cho desktop áp dụng kỹ thuật siêu luồng (tốc độ khởi điểm là 3.06GHz).

Kỹ thuật siêu phân luồng (hyper-threading) cho phép các ứng dụng đa luồng thực hiện các luồng song song. Trong các kỹ thuật trước, sự phân luồng thực hiện bằng cách cắt các lệnh thành nhiều dòng (stream) khác nhau, mỗi dòng sẽ do một vi xử lý thực hiện (trong hệ thống đa xử lý). Với kỹ thuật siêu phân luồng, sự phân luồng sử dụng các tài nguyên của vi xử lý hiệu quả hơn do quá trình song song là tốt hơn.

Kỹ thuật siêu phân luồng cung cấp trạng thái song song ở cấp độ luồng (TLP – thread level parallelism) cho mỗi vi xử lý, kết quả là gia tăng khả năng tận dụng tài nguyên của vi xử lý. Siêu phân luồng là một dạng của kỹ thuật đa luồng song song (SMT – Simultaneous Multi Threading) trong đó nhiều luồng có thể được thực thi tại cùng một thời điểm trên một vi xử lý. Vấn đề này thực hiện bằng cách kết hợp 2 AS (Architectural State) trong mỗi vi xử lý, các AS sẽ dùng chung tài nguyên của vi xử lý. Kỹ thuật này làm đáp ứng thời gian của vi xử lý sẽ nhanh hơn trong môi trường đa nhiệm và cho phép thực hiện nhanh các hoạt động đa luồng và đa nhiệm bằng cách sử dụng các tài nguyên nhàn rỗi.

Kỹ thuật siêu phân luồng và đa luồng song song (SMT - Simultaneous Multi-Threading)

Intel phát triển SMT từ một công nghệ gốc có tên mã là Jackson với cái tên khác là Hyper-Threading – kỹ thuật siêu phân luồng. Trước khi có thể hiểu về cách thức hoạt động của kỹ thuật này, chúng ta cần phải tìm hiểu cơ bản về nó, đặc biệt là về chuỗi lệnh và cách chúng hoạt động.

Cái gì làm cho một ứng dụng có thể chạy? Làm thế nào CPU biết các chỉ dẫn để thực hiện và thực hiện với dữ liệu nào? Tất cả những thông tin này có chứa trong mã biên dịch của ứng dụng đang chạy mỗi khi nạp ứng dụng đó vào. Ứng dụng lần lượt gửi các chuỗi lệnh báo cho CPU biết phải làm gì để đáp ứng, và đối với CPU chuỗi lệnh sẽ là một tập các chỉ thị cần phải thực thi. CPU biết chính xác các chỉ thị này nằm ở đâu nhờ thanh ghi bộ đếm chương trình (PC – Program Counter). PC luôn chỉ đến vị trí trong bộ nhớ nơi mà các chỉ thị cần thực hiện tiếp theo đã được lưu giữ, như vậy một khi chuỗi lệnh được gửi đến CPU thì địa chỉ trong bộ nhớ của chuỗi lệnh này đã được nạp sẵn vào PC, vì vậy CPU biết bắt đầu thực hiện từ đâu. Sau mỗi chỉ thị, PC sẽ tăng lên và quá trình tiếp tục đến hết chuỗi lệnh. Khi chuỗi lệnh được thực hiện xong, PC sẽ bị ghi đè bởi chỉ thị tiếp theo. Chuỗi lệnh có thể bị ngắt bởi một yêu cầu khác, khi đó CPU sẽ lưu giá trị hiện tại của PC trong ngăn xếp (stack) và nạp giá trị mới vào PC, tuy nhiên hạn chế là tại mỗi thời điểm chỉ có thể có duy nhất một chuỗi lệnh được thực thi. Một hướng giải quyết chung cho vấn đề này là sử dụng hai hay nhiều CPU, nếu tại mỗi thời điểm một CPU chỉ có thể thực thi một chuỗi lệnh thì hai hay nhiều CPU sẽ thực thi được hai hay nhiều chuỗi lệnh. Tuy vậy, lại có nhiều vấn đề nảy sinh với cách giải quyết này, trước hết là nhiều CPU sẽ tốn nhiều tiền, quan trọng hơn nữa là việc quản lý hai hay nhiều CPU để chúng chia sẻ tốt tài nguyên chung. Ví dụ, cho tới trước khi chipset AMD 760MP được đưa ra, tất cả các nền tảng x86 đa xử lý chỉ hỗ trợ việc chia băng thông sẵn có giữa các CPU, điều quan trọng nhất là các ứng dụng và hệ điều hành cần phải có khả năng hỗ trợ tính năng này. Hiện nay, để giải quyết nhanh các chuỗi lệnh phức tạp, phần cứng nói chung phải nhờ vào phương án xử lý đa luồng, hệ điều hành phải hỗ trợ xử lý đa luồng, và phải tăng tốc độ một cách thật sự, giống như có nhiều bộ xử lý (trong hầu hết các trường hợp). Kỹ thuật siêu phân luồng của Intel giải quyết vấn đề bằng cách thực hiện nhiều hơn một chuỗi lệnh tại cùng một thời điểm.

Hiệu quả của các bộ vi xử lý

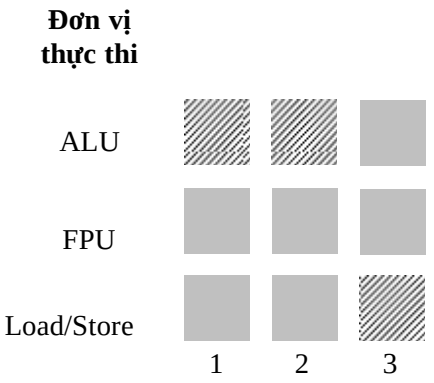
Lấy P4 làm ví dụ, CPU này có tổng cộng 7 đơn vị thực thi, hai trong số đó có thể thực hiện hai lệnh mỗi xung clock (gọi là double pumped ALUs). Nhưng ngay cả như vậy thì cũng không thể tìm được phần mềm nào tận dụng hết các đơn vị thực thi đó. Hầu hết các phần mềm cho máy tính cá nhân đang sử dụng chỉ làm việc với một ít

phép tính số nguyên như nạp và lưu trữ mà không hề động đến đơn vị thực thi dấu chấm động. Còn một số phần mềm chỉ tập trung vào mỗi đơn vị xử lý dấu chấm động mà không sử dụng đến đơn vị xử lý số nguyên. Ngay cả ứng dụng chủ yếu sử dụng phép tính số nguyên cũng không tận dụng tất cả các đơn vị xử lý số nguyên, đặc biệt là một thành phần trong CPU chuyên dùng cho phép dịch hay quay.

Giả sử một CPU với 3 đơn vị thực thi: một đơn vị số nguyên (ALU – Arithmetic Logic Unit), một đơn vị dấu chấm động (FPU – Floating Point Unit) và một đơn vị nạp/lưu trữ (đơn vị dùng để đọc/ghi bộ nhớ). Giả sử CPU có thể thực hiện mọi lệnh trong vòng một chu kỳ xung clock và đồng thời giải quyết nhiều lệnh tới cả ba đơn vị thực thi. Ta cần CPU thực thi chuỗi lệnh sau:

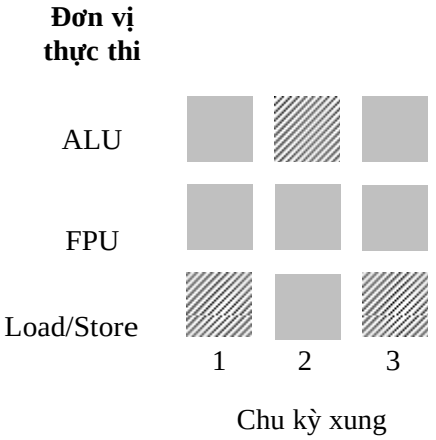
1+1
10+1
Lưu trữ kết quả

Biểu đồ dưới đây sẽ giúp minh họa mức độ của các đơn vị thực thi, màu xám biểu thị đơn vị thực thi không sử dụng, gạch chéo cho biết đơn vị thực thi hoạt động.



Có thể thấy rằng trong mỗi xung clock sẽ chỉ có 33% trong số các đơn vị được sử dụng, và trong các phép toán này hoàn toàn không sử dụng FPU.

Giả sử gửi một chuỗi lệnh khác đến các đơn vị thực thi của CPU, lần này là các lệnh tải, cộng và lưu trữ:

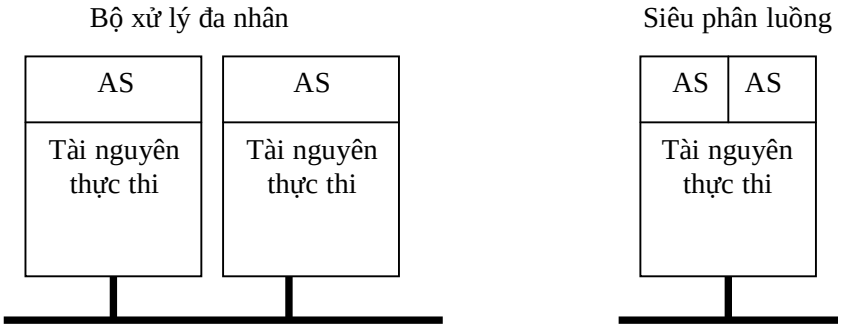


Ta thấy rằng cũng chỉ sử dụng có 33% số các đơn vị thực thi. Thuật toán xử lý song song được gọi là ILP (instruction level parallelism), ở đó các chỉ dẫn phức tạp được thực hiện đồng thời bởi vì CPU có khả năng tận dụng các đơn vị xử lý song song, tức là có nhiều hơn 33% số đơn vị xử lý được sử dụng. Tuy nhiên trên thực tế hầu hết các mã lệnh x86 không phải là ILP, vì vậy ta phải tìm những cách khác để tăng hiệu quả. Ví dụ, hệ thống có 2 CPU và chúng có thể thực hiện các chuỗi lệnh đồng thời, cách này được biết đến như là xử lý song song theo luồng để tăng cường hiệu năng, tuy nhiên lại rất tốn kém.

Kỹ thuật siêu phân luồng

Các đơn vị thực thi không được sử dụng thường xuyên là do CPU không thể lấy dữ liệu nhanh như nó mong muốn do tắc nghẽn đường truyền (memory bus và front-side-bus), dẫn đến sự giảm sút hoạt động của các đơn vị thực thi. Ngoài ra, một nguyên nhân khác đã được đề cập là có quá ít ILP trong hầu hết các chuỗi lệnh thực thi.

Kỹ thuật siêu phân luồng



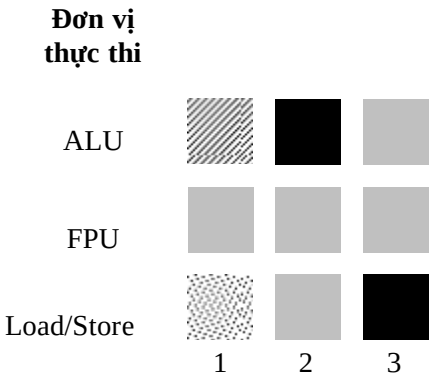
Hình 2.5 – So sánh bộ xử lý đa nhân và siêu phân luồng

Hiện thời đa số các phương pháp dùng để cải thiện hiệu năng trong các thế hệ CPU là tăng tốc độ xung clock và tăng độ lớn của bộ nhớ đệm (cache). Nhưng cho dù cả hai cách này cùng được sử dụng thì vẫn không thực sự sử dụng hết được tài nguyên sẵn có của CPU. Nếu có cách nào đó cho phép thực thi được nhiều chuỗi lệnh đồng thời mới có thể tăng hiệu quả sử dụng tài nguyên của CPU. Đó chính là cách mà kỹ thuật siêu phân luồng của Intel đã làm được, bản chất của nó là chia sẻ tài nguyên để sử dụng hiệu quả hơn các đơn vị thực thi lệnh đã có sẵn trên CPU.

Siêu phân luồng là một kỹ thuật nằm ngoài x86, là một phần nhỏ của SMT. Ý tưởng của SMT rất đơn giản: một CPU vật lý sẽ xuất hiện trên hệ điều hành như là hai CPU logic và hệ điều hành không thể phân biệt được. Nhiệm vụ của hệ điều hành là gửi 2 chuỗi lệnh tới 2 CPU và phần cứng sẽ đảm nhiệm những công việc còn lại.

Trong các CPU sử dụng kỹ thuật siêu phân luồng, mỗi CPU logic sở hữu một tập các thanh ghi, kể cả thanh ghi bộ đếm chương trình riêng (separate program counter), CPU vật lý sẽ luân phiên các giai đoạn tìm/giải mã lệnh giữa hai CPU logic và thực thi những thao tác từ hai chuỗi lệnh đồng thời theo cách hướng tới những đơn vị thực thi ít được sử dụng.

Hạn chế của siêu phân luồng



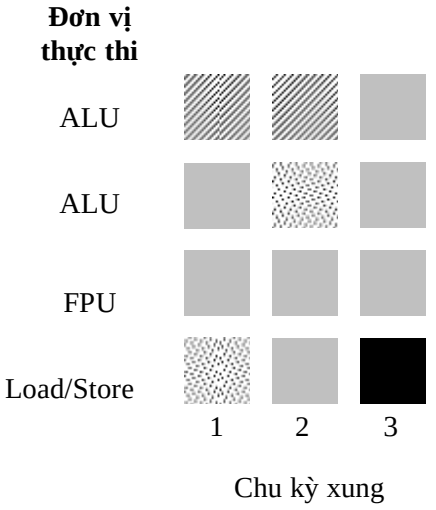
Giả sử rằng CPU đơn giản trước đây cũng có các đặc tính của siêu phân luồng:

Các ô gạch chéo hiển thị một chỉ dẫn từ chuỗi lệnh thứ nhất đang được thực hiện, trong khi những ô chấm chấm hiển thị một chỉ dẫn từ chuỗi lệnh thứ hai đang được thực hiện. Các ô màu xám hiển thị những đơn vị thực hiện không được sử dụng, trong khi các ô màu đen hiển thị xung đột khi mà cả hai chỉ dẫn đều sử dụng cùng một đơn vị thực thi. Rõ ràng là việc thực thi song song hai chuỗi lệnh với kỹ thuật siêu phân luồng lại thực hiện chậm hơn so với một CPU thông thường. Nguyên nhân thật ra rất đơn giản: CPU đồng thời thực hiện hai chuỗi lệnh quá đơn giản, tất cả đều là trùng lặp với lệnh add, load, store. Nếu thực thi các ứng dụng đòi hỏi nhiều phép toán động cùng với các ứng dụng số nguyên thì kết quả sẽ khác đi. Hiện tại các ứng dụng văn phòng trên máy tính để bàn hầu như chỉ sử dụng số nguyên (và trong tương lai chắc cũng vẫn chỉ sử dụng số nguyên). Vì vậy lợi ích mà công nghệ siêu phân luồng đem lại thấp (và đôi khi còn kém hơn không dùng công nghệ siêu phân luồng). Trên thực tế, nếu kích hoạt tính năng siêu phân luồng trên desktop, có thể giảm tốc độ tới 10%. Tuy nhiên người dùng các ứng dụng tính toán phức tạp thì sẽ được hưởng lợi rất nhiều từ kỹ thuật này. Ngoài ra kỹ thuật này cũng tăng tốc đáng kể cho các máy chủ, nhất là các máy chủ web server.

Lợi ích của siêu phân luồng

Intel đã tạo ra siêu phân luồng không chỉ để cho các CPU máy chủ. Thực ra kiến trúc NetBurst của P4 và Xeon hiện nay hoàn chỉnh với lõi SMT. Xét ví dụ ở trên, ta cho thêm một ALU thứ 2 và thực hiện hai chuỗi lệnh trên.

Với một ALU thứ 2, xung đột duy nhất gặp phải là lần lưu trữ cuối cùng. Ta biết rằng CPU P4 được thiết kế với ba đơn vị số nguyên (hai ALU và một đơn vị xử lý số nguyên khác chậm hơn cho phép dịch/quay). Quan trọng hơn nữa là mỗi ALU của P4 có thể thực hiện hai vi lệnh trong cùng một xung clock, nghĩa là trong hai chỉ dẫn add (phép cộng) mỗi chỉ dẫn có thể từ hai chuỗi lệnh khác nhau, được thực hiện đồng thời trong một xung clock duy nhất trên P4/Xeon.



Nhưng điều đó vẫn chưa giải quyết được vấn đề, do việc tăng thêm các đơn vị xử lý để tăng hiệu quả với kỹ thuật siêu phân luồng lại tốn kém đứng từ quan điểm vật lý (làm cho CPU có nhiều transistor hơn, tiêu tốn nhiều điện năng hơn; hoặc phải giảm kích thước CPU với các công nghệ chế tạo mới). Thay vào đó, Intel đang khuyến khích các nhà phát triển tối ưu hoá kỹ thuật siêu phân luồng. Chẳng hạn sử dụng lệnh dừng (HALT) một trong các bộ xử lý logic sẽ tối đa được tốc độ cho các ứng dụng không sử dụng được kỹ thuật siêu phân luồng, CPU còn lại chỉ hoạt động như là hệ thống một CPU. Khi một ứng dụng có thể sử dụng lợi ích từ siêu phân luồng, bộ xử lý logic thứ hai lại tiếp tục được hoạt động.

3. Các thanh ghi

CPU 8086/8088 có tất cả 14 thanh ghi nội. Các thanh ghi này có thể phân loại như sau:

- Thanh ghi dữ liệu (data register)
- Thanh ghi chỉ số và con trỏ (index & pointer register)
- Thanh ghi đoạn (segment register)
- Thanh ghi trạng thái và điều khiển (status & control register)

3.1. Các thanh ghi dữ liệu

Các thanh ghi dữ liệu gồm có các thanh ghi 16 bit AX, BX, CX và DX trong đó nửa cao và nửa thấp của mỗi thanh ghi có thể định địa chỉ một cách độc lập. Các nửa thanh ghi này (8 bit) có tên là AH và AL, BH và BL, CH và CL, DH và DL.

Các thanh ghi này được sử dụng trong các phép toán số học và logic hay trong quá trình chuyển dữ liệu.

Thanh ghi	Sử dụng trong
AX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word) IN (nhập word) OUT (xuất word)

	CWD Các phép toán xử lý chuỗi (string)
AL	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) IN (nhập byte) OUT (xuất byte) XLAT AAA, AAD, AAM, AAS (các phép toán ASCII) CBW (đổi sang word) DAA, DAS (số thập phân) Các phép toán xử lý chuỗi (string)
AH	MUL, IMUL (toán hạng nguồn kích thước byte) DIV, IDIV (toán hạng nguồn kích thước byte) CBW (đổi sang word)
BX	XLAT
CX	LOOP, LOOPE, LOOPNE Các phép toán string với tiếp dầu ngữ REP
CL	RCR, RCL, ROR, ROL (quay với số đếm byte) SHR, SAR, SAL (dịch với số đếm byte)
DX	MUL, IMUL (toán hạng nguồn kích thước word) DIV, IDIV (toán hạng nguồn kích thước word)

AX (ACC – Accumulator): thanh ghi tích luỹ

BX (Base): thanh ghi cơ sở

CX (Count): đếm

DX (Data): thanh ghi dữ liệu

3.2. Các thanh ghi chỉ số và con trỏ

Bao gồm các thanh ghi 16 bit SP, BP, SI và DI, thường chứa các giá trị offset (độ lệch) cho các phần tử định địa chỉ trong một phân đoạn (segment). Chúng có thể được sử dụng trong các phép toán số học và logic. Hai thanh ghi con trỏ (SP – Stack Pointer và BP – Base Pointer) cho phép truy xuất dễ dàng đến các phần tử đang ở trong ngăn xếp (stack) hiện hành. Các thanh ghi chỉ số (SI – Source Index và DI – Destination Index) được dùng để truy xuất các phần tử trong các đoạn dữ liệu và doạn thêm (extra segment). Thông thường, các thanh ghi con trỏ liên hệ đến đoạn stack hiện hành và các thanh ghi chỉ số liên hệ đến doạn dữ liệu hiện hành. SI và DI dùng trong các phép toán chuỗi.

3.3. Các thanh ghi đoạn

Bao gồm các thanh ghi 16 bit CS (Code segment), DS (Data segment), SS (stack segment) và ES (extra segment), dùng để định địa chỉ vùng nhớ 1 MB bằng cách chia thành 16 đoạn 64 KB.

Tất cả các lệnh phải ở trong đoạn mã hiện hành, được định địa chỉ thông qua thanh ghi CS. Offset (độ lệch) của mã được xác định bằng thanh ghi IP. Dữ liệu chương trình thường được đặt ở đoạn dữ liệu, định vị thông qua thanh ghi DS. Stack

định vị thông qua thanh ghi SS. Thanh ghi đoạn thêm có thể sử dụng để định địa chỉ các toán hạng, dữ liệu, bộ nhớ và các phần tử khác ngoài đoạn dữ liệu và stack hiện hành.

3.4. Các thanh ghi điều khiển và trạng thái

Thanh ghi con trỏ lệnh IP (Instruction Pointer) giống như bộ đếm chương trình (Program Counter). Thanh ghi điều khiển này do BIU quản lý nhằm lưu trữ offset từ bắt đầu đoạn mã đến lệnh thực thi kế tiếp và không thể xử lý trực tiếp thanh ghi IP.

Thanh ghi cờ (Flag register) dài 16 bit chứa 3 bit điều khiển (TF, IF và DF) và 6 bit trạng thái (OF, SF, ZF, AF, PF và CF) còn các bit còn lại không sử dụng.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	OF	DF	IF	TF	SF	ZF	X	AF	X	PF	X	CF

- OF (Overflow - tràn): OF = 1 xác định tràn số học, xảy ra khi kết quả vượt ra ngoài phạm vi biểu diễn
- DF (Direction- hướng): xác định hướng chuyển chuỗi, DF = 1 khi CPU làm việc với chuỗi theo thứ tự từ phải sang trái và ngược lại.
- IF (Interrupt - ngắt): cho phép hay cấm các ngắt có mặt nạ.
- TF (Trap - bẫy): đặt CPU vào chế độ từng bước, dùng cho các chương trình gỡ rối (debugger).
- SF (Sign - dấu): dùng để chỉ các kết quả số học là số dương (SF = 0) hay âm (SF = 1).
- ZF (Zero): = 1 nếu kết quả của phép toán trước là 0.
- AF (Auxiliary – nhớ phụ): dùng trong các số thập phân để chỉ nhớ từ nửa byte thấp hay mượn từ nửa byte cao.
- PF (Parity): PF = 1 nếu kết quả của phép toán là có tổng số bit 1 là chẵn (dùng để kiểm tra lỗi truyền dữ liệu)
- CF (Carry): CF = 1 nếu có nhớ hay mượn từ bit cao nhất của kết quả. Cờ này cũng dùng cho các lệnh quay.

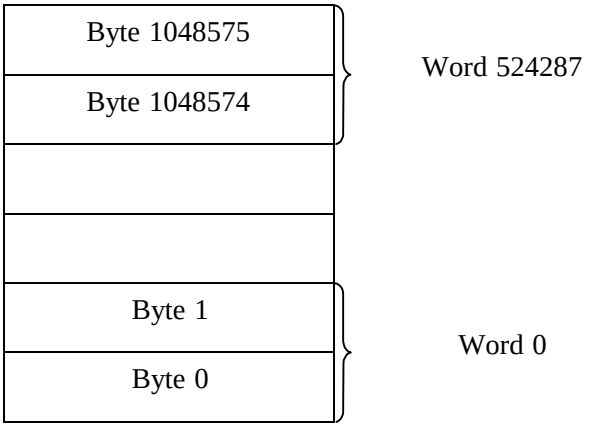
4. Phân đoạn bộ nhớ

Ta biết rằng dù 8086 là CPU 16 bit (có bus dữ liệu 16 bit) nhưng vẫn dùng bộ nhớ theo các byte. Điều này cho phép CPU làm việc với byte cũng như word, nó rất quan trọng trong giao tiếp với các thiết bị I/O như máy in, thiết bị đầu cuối và modem (chúng được thiết kế để chuyển dữ liệu mã hoá ASCII 7 hay 8 bit). Ngoài ra, nhiều mã lệnh của 8086/8088 có chiều dài 1 byte nên cần phải truy xuất được các byte riêng biệt để có thể xử lý các lệnh này.

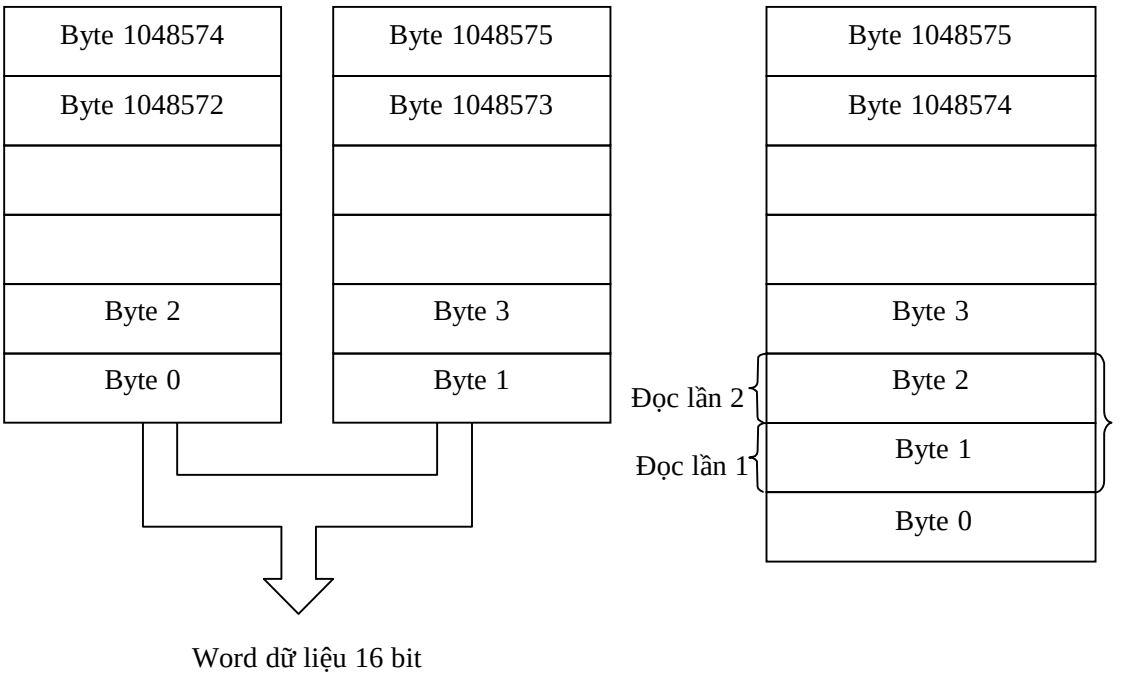
8086/8088 có bus địa chỉ 20 bit nên có thể cho phép truy xuất 2^{20} = 1048576 địa chỉ bộ nhớ khác nhau.

Để thực hiện đọc 16 bit từ bộ nhớ, 8086 sẽ thực hiện đọc đồng thời byte có địa chỉ lẻ và byte có địa chỉ chẵn. Do đó, 8086 tổ chức bộ nhớ thành các bank chẵn và lẻ.

Theo hình 2.6, ta có thể thấy rằng các word luôn bắt đầu tại địa chỉ chẵn nhưng ta vẫn có thể đọc word có địa chỉ lẻ bằng cách thực hiện 2 chu kỳ đọc bộ nhớ: một chu kỳ đọc byte thấp và một chu kỳ đọc byte cao nhưng điều này làm chậm tốc độ xử lý. Đối với 8088 thì do bus dữ liệu 8 bit nên dù word có địa chỉ chẵn hay lẻ, nó cũng cần phải thực hiện 2 chu kỳ đọc hay ghi bộ nhớ và giao tiếp với bộ nhớ như một bank.

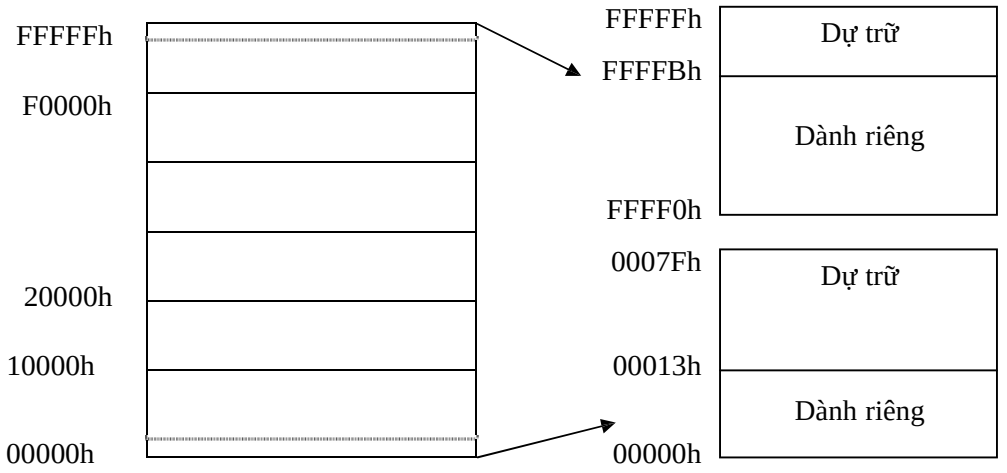


Hình 2.6 – Vùng nhớ của 8086/8088 có 1048576 byte hay 524288 word



Hình 2.7 – Đọc word địa chỉ chẵn và địa chỉ lẻ

Ngoài ra bộ nhớ cũng chia thành 16 khối, mỗi khối có kích thước 64 KB, bắt đầu ở địa chỉ 00000h và kết thúc ở FFFFFh. Địa chỉ bắt đầu mỗi khối sẽ tăng lên 1 ở số hex có ý nghĩa nhiều nhất khi thay đổi từ khối này sang khối kia. Ví dụ như khối 00000h → 10000h → 20000h …



Hình 2.8 – Bảng bộ nhớ cho 8086/8088

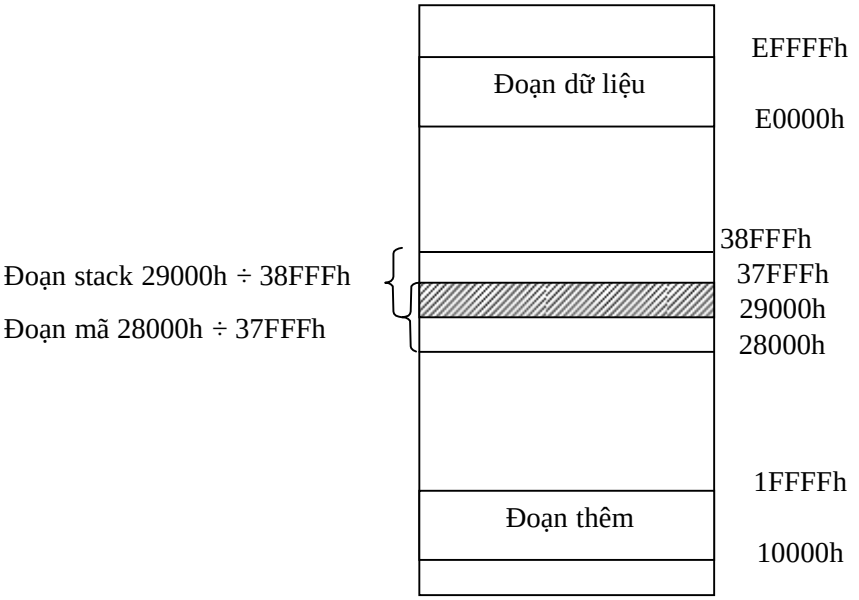
8086/8088 định nghĩa 4 khối bộ nhớ 64KB: đoạn mã (code segment) giữ các mã lệnh chương trình, đoạn ngăn xếp (stack segment) lưu các địa chỉ sẽ trả về từ các chương trình con (subroutine) hay trình phục vụ ngắt (interrupt subroutine), đoạn dữ liệu (data segment) lưu trữ dữ liệu cho chương trình và đoạn thêm (extra segment) thường dùng cho các dữ liệu dùng chung.

Các thanh ghi đoạn (CS, DS, SS và ES) dùng để chỉ vị trí nền của mỗi đoạn. Các thanh ghi này có 16 bit trong khi địa chỉ bộ nhớ là 20 bit nên để xác dịnh vị trí bộ nhớ, ta sẽ thêm 4 bit 0 vào các bit thấp của thanh ghi đoạn. Giả sử như thanh ghi CS chứa giá trị 1111h thì nó sẽ chỉ tới địa chỉ nền là 11110h. Chú ý rằng địa chỉ bắt đầu một đoạn không thể tuỳ ý mà phải bắt đầu tại một địa chỉ chia hết cho 16. Nghĩa là 4 bit thấp phải là 0. Ta cũng chú ý rằng 4 đoạn có thể không tách rời nhau mà chồng lấp lên nhau và ta cũng có thể cho 4 giá trị của các thanh ghi đoạn bằng nhau nghĩa là 4 đoạn này trùng nhau.

VD: Thanh ghi DS có giá trị là 1000h thì địa chỉ nền là 10000h. Địa chỉ kết thúc tìm được bằng cách cộng địa chỉ nền với giá trị FFFFh (64K) → địa chỉ kết thúc là 10000h + FFFFh = 1FFFFh. Như vậy đoạn dữ liệu có địa chỉ từ 10000h ÷ 1FFFFh.

Các vị trí bộ nhớ không được định nghĩa trong các đoạn hiện hành không thể truy xuất được. Muốn truy xuất đến các vị trí đó, ta phải định nghĩa lại một trong các thanh ghi đoạn sau cho đoạn phải chứa vị trí đó. Như vậy, tại một thời điểm bất kỳ ta chỉ có thể truy xuất tối đa 4 × 64 KB = 256 KB bộ nhớ. Nội dung của các thanh ghi đoạn chỉ có thể xác định thông qua phần mềm.

VD: Giả sử các thanh ghi đoạn có các giá trị CS = 2800h, DS = E000h, SS = 2900h và ES = 1000h. Ta có vị trí các đoạn trong bảng bộ nhớ như sau:



Hình 2.9 – Vị trí các phân đoạn theo giá trị các thanh ghi đoạn

Địa chỉ logic và địa chỉ vật lý:

Các địa chỉ trong một đoạn thay đổi từ 0000h ÷ FFFFh, tương ứng với chiều dài đoạn là 64 KB. Một địa chỉ trong một đoạn được gọi là ***địa chỉ logic hay offset***. Ví dụ như địa chỉ logic 0010h của đoạn mã trong hình 2.9 sẽ có địa chỉ thật sự là 28000h + 0010h = 28010h. Địa chỉ này gọi là ***địa chỉ vật lý***. Địa chỉ vật lý chính là địa chỉ thật sự xuất hiện ở bus địa chỉ, nó có chiều dài 20 bit còn địa chỉ logic là độ lệch (offset) từ vị trí 0 của một đoạn cho trước.

VD: Giả sử xét các đoạn như hình 2.9. Địa chỉ vật lý tương ứng với địa chỉ logic 1000h trong đoạn stack là:

29000h + 1000h = 2A000h

Địa chỉ vật lý tương ứng với địa chỉ logic 2000h trong đoạn mã là:

28000h + 2000h = 2A000h

Ta thấy rằng có thể địa chỉ vật lý trùng nhau khi địa chỉ logic khác nhau nghĩa là một địa chỉ vật lý có thể có nhiều địa chỉ logic khác nhau.

Để chỉ địa chỉ logic 1000h trong đoạn mã, ta dùng ký hiệu CS:1000h. Tương tự như vậy cho các đoạn khác, nghĩa là địa chỉ logic 1111h trong đoạn dữ liệu sẽ là DS:1111h.

Mọi lệnh tham chiếu bộ nhớ sẽ có một thanh ghi đoạn mặc nhiên. Thanh ghi IP cung cấp địa chỉ offset khi truy xuất đến đoạn mã và BP cho đoạn stack. Ví dụ như IP = 1000h và CS = 2000h thì BIU sẽ truy xuất đến địa chỉ 20000h + 1000h = 21000h và nhận byte tại vị trí này.

Tham chiếu bộ nhớ	Đoạn mặc nhiên	Đoạn khác	Offset
Nhận lệnh	CS	Không	IP
Tác vụ stack	SS	Không	SP
Dữ liệu tổng quát	DS	CS,ES,SS	Địa chỉ hiệu dụng
Nguồn của string	DS	CS,ES,SS	SI
Đích của string	ES	Không	DI
BX dùng làm con trỏ	DS	CS,ES,SS	Địa chỉ hiệu dụng
BP dùng làm con trỏ	SS	CS,ES,SS	Địa chỉ hiệu dụng

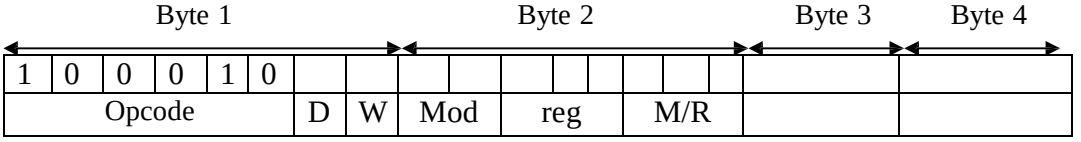
VD: Ta sử dụng lệnh MOV [BP],AL với BP = 2C00h. Ở đây BP dùng làm con trỏ nên dùng đoạn stack. Giả sử các phân đoạn như hình 2.9 thì địa chỉ vật lý sẽ là 29000h + 2C00h = 2BC00h

5. Cách mã hoá lệnh

Lệnh của CPU sẽ biểu diễn bằng các ký tự dưới dạng gợi nhớ (mnemonic) để có thể dễ dàng sử dụng. Đối với CPU thì các lệnh được biểu diễn bằng các mã lệnh (opcode) nên sau khi nhận lệnh CPU phải thực hiện giải mã lệnh rồi mới thực thi nó. Một lệnh CPU có thể dài 1 byte hay nhiều byte. Nếu ta dùng 1 byte để mã hoá thì sẽ mã hoá được 256 lệnh khác nhau. Tuy nhiên do một lệnh không phải chỉ có một cách thực hiện nên ta không thể thực hiện đơn giản như trên.

Để tìm hiểu cách mã hoá lệnh, ta xét lệnh **MOV des,src** dùng để chuyển dữ liệu giữa hai thanh ghi hay một ô nhớ và một thanh ghi.

Lệnh MOV mã hoá như sau:



Để mã hóa lệnh MOV, ta cần dùng ít nhất là 2 byte trong đó 6 bit dùng cho mã lệnh.

Bit D xác định hướng truyền của dữ liệu, D = 0 xác định dữ liệu sẽ đi từ thanh ghi cho bởi 3 bit Reg, D = 1 xác định dữ liệu sẽ đi đến thanh ghi cho bởi 3 bit Reg.

Bit W xác định sẽ truyền 1 byte (W = 0) hay 1 word (W = 1).

3 bit Reg dùng để chọn thanh ghi sử dụng:

Mã	Thanh ghi	
	W = 1	W = 0
000	AX	AL
001	CX	CL
010	DX	DL
011	BX	BL
100	SP	AH
101	BP	CH
110	SI	DH
111	DI	BH

2 bit mod và 3 bit R/M (Register / Memory) dùng để xác định chế độ địa chỉ cho các toán hạng của lệnh.

MOD / R/M	00	01	10	11	
				W = 1	W = 0
000	[BX]+[SI]	[BX]+[SI]+addr8	[BX]+[SI]+addr16	AX	AL
001	[BX]+[DI]	[BX]+[DI]+addr8	[BX]+[DI]+addr16	CX	CL
010	[BP]+[SI]	[BP]+[SI] +addr8	[BP]+[SI] +addr16	DX	DL
011	[BP]+[DI]	[BP]+[DI] +addr8	[BP]+[DI] +addr16	BX	BL
100	[SI]	[SI] +addr8	[SI] +addr16	SP	AH
101	[DI]	[DI] +addr8	[DI] +addr16	BP	CH
110	addr16	[BP] +addr8	[BP] +addr16	SI	DH
111	[BX]	[BX] +addr8	[BX] +addr16	DI	BH

Tổng quát, 8086/8088 có khoảng 300 tác vụ có thể có trong tập lệnh của nó. Mỗi lệnh kéo dài từ 1 đến 6 byte. Từ ví dụ trên, ta thấy mã lệnh có các vùng:

- Vùng mã lệnh (opcode): chứa mã lệnh của lệnh sẽ thực thi
- Vùng thanh ghi (reg): chứa các thanh ghi sẽ thực hiện
- Vùng chế độ (mod)
- Vùng thanh ghi / bộ nhớ R/M (Reg/Mem)

6. Các cách định địa chỉ

Cách định địa chỉ	Mã đối tượng	Ví dụ			
		Từ gợi nhớ	Đoạn truy xuất	Hoạt động	Mô tả
Tức thời	B80010	MOV AX,1000h	Mã	AH ← 10h AL ← 00h	(1)
Thanh ghi	8BD1	MOV DX,CX	Trong µP	DX ← CX	(2)
Trực tiếp	8A260010	MOV AH,[1000h]	Döõ lieäu	AH ← [1000h]	(3)
Gián tiếp thanh ghi	8B04 FF25 FE4600 FF0F	MOV AX,[SI] JMP [DI] INC BYTE PTR [BP] DEC WORD PTR [BX]	Dữ liệu Dữ liệu Stack Dữ liệu	AL ← [SI]; AH ←[SI+1] IP←[DI+1:DI] [BP]←[BP]+1 [BX+1:BX]← [BX+1:BX]-1	(4)
Có chỉ số	8B4406 FF6506	MOV AX,[SI+6] JMP [DI+6]	Dữ liệu Dữ liệu	AL ← [SI+6]; AH ←[SI+7] IP←[DI+7:DI+6]	(5)
Có nền	8B4602 FF6702	MOV AX,[BP+2] JMP [BP+2]	Stack Dữ liệu	AL←[BP+2]; AH ←[BP+3] IP←[BX+3:BX+6]	(6)
Có nền và có chỉ số	8B00 FF21 FE02 FF0B	MOV AX,[BX+SI] JMP [BX+DI] INC BYTE PTR [BP+SI] DEC WORD PTR [BP+DI]	Dữ liệu Dữ liệu Stack Stack	AL←[BX+SI];AH←[BX+SI+1] IP←[BX+DI+1:BX+DI] [BP+SI]←[BP+SI]+1 [BP+DI+1:BP+DI]← [BP+DI+1:BP+DI]-1	(7)

Có nền và có chỉ số với độ dời	8B4005 FF6105 FE4205 FF4B05	MOV AX,[BX+SI+5] JMP [BX+DI+5] INC BYTE PTR [BP+SI+5] DEC WORD PTR [BP+DI+5]	Dữ liệu Dữ liệu Stack Stack	AL ← [BX+SI+5] AH ← [BX+SI+1] IP ← [BX+DI+6:BX+DI+5] [BP+SI+5] ← [BP+SI+5]+1 [BP+DI+6:BP+DI+5] ← [BP+DI+6:BP+DI+5]-1	(8)
String	A4	MOVSB	Thêm, dữ liệu	[ES:DI] ← [DS:DI] Nếu DF = 0 thì SI ← SI + 1; DI ← DI + 1 Nếu DF = 1 thì SI ← SI - 1; DI ← DI - 1	(9)

- BYTE PTR và WORD PTR tránh lầm giữa truy xuất byte và word.
- Độ dời được cộng vào thanh ghi con trỏ hay nền là số nhị phân dạng bù 2.
- (1): nguồn dữ liệu trong lệnh
- (2): đích và nguồn là các thanh ghi của µP
- (3): địa chỉ bộ nhớ cung cấp trong lệnh
- (4): địa chỉ bộ nhớ cung cấp trong thanh ghi con trỏ hay chỉ số
- (5): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số cộng với độ dời trong lệnh
- (6): địa chỉ bộ nhớ là tổng của thanh ghi BX hay BP cộng với độ dời trong lệnh
- (7): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số và thanh ghi nền
- (8): địa chỉ bộ nhớ là tổng của thanh ghi chỉ số, thanh ghi nền và độ dời trong lệnh
- (9): địa chỉ nguồn bộ nhớ là thanh ghi SI trong đoạn dữ liệu và địa chỉ đích bộ nhớ là thanh ghi DI trong đoạn thêm

6.1. Định địa chỉ tức thời

Các lệnh dùng cách định địa chỉ tức thời lấy dữ liệu trong lệnh làm một phần của lệnh. Trong cách này, dữ liệu sẽ được chứa trong đoạn mã thay vì trong đoạn dữ liệu. Dữ liệu cho lệnh ***MOV AX,1000h*** được cung cấp tức thời sau mã lệnh B8. Chú ý rằng trong mã đối tượng byte dữ liệu cao đi sau byte dữ liệu thấp.

Cách định địa chỉ tức thời thường dùng để nạp một thanh ghi hay vị trí bộ nhớ với các dữ liệu ban đầu. Sau đó, các lệnh kế tiếp sẽ làm việc với các dữ liệu này. Tuy nhiên, cách định địa chỉ này không sử dụng được cho các thanh ghi đoạn.

6.2. Định địa chỉ thanh ghi

Một số lệnh chỉ làm công việc chuyển dữ liệu giữa các thanh ghi của CPU. Ví dụ như ***MOV DX,CX*** sẽ chuyển dữ liệu từ thanh ghi CX vào thanh ghi DX. Ở đây ta không cần thực hiện tham chiếu bộ nhớ.

Ta có thể kết hợp cách định địa chỉ tức thời và định địa chỉ thanh ghi để nạp dữ liệu cho các thanh ghi đoạn.

6.3. Định địa chỉ trực tiếp

Ngoài 2 cách định địa chỉ trên, tất cả các cách định địa chỉ còn lại đều cần phải truy xuất đến bộ nhớ với ít nhất một toán hạng. Trong cách định địa chỉ trực tiếp, địa chỉ bộ nhớ được cung cấp trực tiếp như là một phần của lệnh. Ví dụ như lệnh ***MOV***

AH,[1000h] sẽ đưa nội dung chứa trong ô nhớ DS:1000h vào thanh ghi AH hay lệnh ***MOV [2000h],AX*** sẽ đưa nội dung chứa trong AX vào 2 ô nhớ liên tiếp DS:2000h và DS:2001h

6.4. Định địa chỉ truy xuất bộ nhớ gián tiếp

Các cách định địa chỉ trực tiếp sẽ thuận lợi cho các truy xuất bộ nhớ không thường xuyên. Tuy nhiên, nếu một ô nhớ cần phải truy xuất nhiều lần trong một chương trình thì quá trình nhận địa chỉ (2 byte) sẽ phải thực hiện nhiều lần. Điều này sẽ không hiệu quả. Để giải quyết vấn đề này, ta thực hiện lưu trữ địa chỉ của ô nhớ cần truy xuất trong một thanh ghi con trỏ, chỉ số hay thanh ghi cơ sở (BX, BP, SI hay DI). Ngoài ra, ta có thể sử dụng độ dời bù 2 bằng cách cộng vào các thanh ghi để dời đi so với vị trí được các thanh ghi chỉ đến.

Cách định địa chỉ	**Địa chỉ hiệu dụng (EA – Effective Address)**		
	Độ dời	**Thanh ghi nền**	**Thanh ghi chỉ số**
Gián tiếp thanh ghi	Không	BX hay BP	Không
	Không	Không	SI hay DI
Có chỉ số	-128 ÷ 127	Không	SI hay DI
Có nền	-128 ÷ 127	BX hay BP	Không
Có nền và chỉ số	Không	BX hay BP	SI hay DI
Có nền và chỉ số với độ dời	-128 ÷ 127	BX hay BP	SI hay DI

Như vậy, một độ dời có thể được cộng vào thanh ghi nền và kết quả này được cộng tiếp vào thanh ghi chỉ số. Địa chỉ thu được gọi là địa chỉ hiệu dụng EA.

Ngoài ra ta cũng có thể viết cách định địa chỉ gián tiếp như sau:

MOV AX,table[SI]

Trong đó ***table*** là nhãn gán cho một vị trí ô nhớ nào đó. Lệnh này sẽ truy xuất phần tử thứ SI trong dãy ***table*** (giả sử SI = 2 thì sẽ truy xuất phần tử thứ 2).

Chú ý rằng các đoạn mặc định cho các cách định địa chỉ gián tiếp là đoạn stack khi dùng BP, là đoạn dữ liệu khi dùng BX, SI hay DI.

VD: Lệnh:

MOV AH,10h	thực hiện định địa chỉ tức thời
MOV AX,[BP + 10]	thực hiện định địa chỉ có nền
MOV AH,[BP + SI]	thực hiện định địa chỉ có nền và có chỉ số

6.5. Định địa chỉ chuỗi

Chuỗi là một dãy liên tục các byte hay word lưu trữ trong bộ nhớ dưới dạng các ký tự ASCII. 8086/8088 có các lệnh dùng để xử lý chuỗi, các lệnh này sử dụng cặp thanh ghi DS:SI để chỉ nguồn chuỗi ký tự và ES:DI để chỉ đích chuỗi. Lệnh MOVSB sẽ chuyển byte dữ liệu nguồn đến vị trí đích trong đó SI và DI sẽ tăng hay giảm tuỳ theo giá trị của DF.

Chương 3
BỘ NHỚ

1. Một số khái niệm

1.1. Bộ nhớ (memory)

Là thiết bị nhớ có thể ghi và chứa thông tin. ROM, RAM, cache, đĩa cứng, đĩa mềm, CD.... đều có thể gọi là bộ nhớ (vì chúng đều lưu trữ thông tin). Các tính chất:

- *Dung lượng*: khả năng lưu trữ dữ liệu của thiết bị. Ví dụ: CD chứa được 700MB, đĩa mềm chứa được 1.44MB, đĩa cứng chứa được 40 GB, 60GB, cache L1 chứa được 16KB, cache L2 chứa được 256 KB ...
- *Tốc độ truy nhập*: liên quan đến tốc độ truyền dữ liệu của thiết bị. Tính về tốc độ thì CPU là lớn nhất, kế tiếp là Cache, sau nữa là các loại RAM.
- *Giao tiếp*: cấu trúc bên ngoài của bộ nhớ. Ví dụ, các RAM có số chân cắm và đặc tính khác nhau.

1.2. Phân loại bộ nhớ

1.2.1. ROM (Read Only Memory)

Đây là loại bộ nhớ dùng trong các hãng sãn xuất là chủ yếu. Nó có đặc tính là thông tin lưu trữ trong ROM không thể xoá được và không sửa được, thông tin sẽ được lưu trữ mãi mãi. Nhưng ngược lại ROM có bất lợi là một khi đã cài đặt thông tin vào rồi thì ROM sẽ không còn tính đa dụng. Ví dụ điển hình là các con "chip" trên motherboard hay là BIOS ROM để vận hành khi máy tính vừa khởi động.

1.2.2. PROM (Programmable ROM)

Mặc dù ROM nguyên thủy là không ghi hay xóa được, nhưng các thế hệ sau của ROM đã đa dụng hơn như PROM. Các hãng sản xuất có thể cài đặt lại ROM bằng cách dùng các loại dụng cụ đặc biệt và đắt tiền. Thông tin có thể cài đặt vào chip và nó sẽ lưu lại mãi trong chip. Một đặc điểm lớn nhất của loại PROM là thông tin chỉ cài đặt một lần mà thôi. CD cũng có thể được gọi là PROM vì chúng ta có thể lưu trữ thông tin vào nó chỉ một lần duy nhất và không thể xoá được.

1.2.3. EPROM (Erasable Programmable ROM)

Một dạng cao hơn PROM là EPROM, tức là ROM có thể xoá và ghi lại được. EPROM khác PROM ở chỗ là thông tin có thể được viết và xoá nhiều lần theo ý người sử dụng, và phương pháp xoá là phần cứng (dùng tia hồng ngoại).

1.2.4. EEPROM (Electrically Erasable Programmable ROM)

Đây là một dạng cao hơn EPROM, đặt điểm khác biệt duy nhất so với EPROM là có thể ghi và xoá thông tin lại nhiều lần bằng phần mềm.

1.2.5. RAM (Random Access Memory)

RAM là thế hệ kế tiếp của ROM, cả RAM và ROM đều là bộ nhớ truy xuất ngẫu nhiên, tức là dữ liệu được truy xuất không cần theo thứ tự. Tuy nhiên ROM chạy chậm hơn RAM rất nhiều. Thông thường ROM cần trên 50ns để xử lý dữ liệu trong khi đó RAM cần dưới 10ns.

1.2.6. SRAM (Static RAM) và DRAM (Dynamic RAM)

SRAM (RAM tĩnh) là loại RAM lưu trữ dữ liệu không cần cập nhật thường xuyên trong khi DRAM là loại RAM cần cập nhật dữ liệu thường xuyên. Thông thường dữ liệu trong DRAM sẽ được làm tươi (refresh) nhiều lần trong một giây để giữ lại những thông tin đang lưu trữ, nếu không thì dữ liệu trong DRAM cũng sẽ bị mất do hiện tượng rò rỉ điện tích của các tụ điện. Các khác biệt của SRAM so với DRAM:

- Tốc độ của SRAM lớn hơn DRAM do không phải tốn thời gian refresh..
- Chế tạo SRAM tốn kém hơn DRAM nên thông thường sử dụng DRAM để hạ giá thành sản phẩm.

1.2.7. FPM - DRAM (Fast Page Mode DRAM)

Là một dạng cải tiến của DRAM, về nguyên lý thì FPM - DRAM sẽ chạy nhanh hơn DRAM do cải tiến cách dò địa chỉ trước khi truy xuất dữ liệu. FPM - DRAM hầu như không còn sản xuất trên thị trường hiện nay nữa.

1.2.8. EDO - DRAM (Extended Data Out DRAM)

Là một dạng cải tiến của FPM - DRAM, nó truy xuất nhanh hơn FPM - DRAM nhờ một số cải tiến cách dò địa chỉ trước khi truy cập dữ liệu. Tuy nhiên, EDO - DRAM là cần hỗ của chipset hệ thống. Loại bộ nhớ nầy chạy với máy 486 trở lên (tốc độ dưới 75MHz). EDO DRAM cũng đã quá cũ so với kỹ thuật hiện nay, tốc độ của EDO-DRAM nhanh hơn FPM-DRAM từ 10 - 15%.

1.2.9. BDEO-DRAM (Burst Extended Data Out DRAM)

Là thế hệ sau của EDO DRAM, dùng kỹ thuật đường ống (pipeline) để rút ngắn thời gian dò địa chỉ.

1.2.10. SDRAM (Synchronous DRAM)

Đây là một loại RAM có nguyên lý chế tạo khác hẳn với các loại RAM trước. Đồng bộ (synchronous) là một khái niệm rất quan trọng trong lĩnh vực số. RAM hoạt động do một bộ điều khiển xung nhịp (clock memory), dữ liệu sẽ được truy xuất hay cập nhật mỗi khi clock chuyển từ logic 0 sang 1, đồng bộ có nghĩa là ngay lúc clock nhảy từ logic 0 sang 1 chứ không hẳn là chuyển sang logic 1 hoàn toàn (tác động bằng cạnh xung). Do kỹ thuật này, SDRAM và các thế hệ sau có tốc độ cao hơn hẳn các loại DRAM trước, đạt tốc độ 66, 100, 133 MHz.

1.2.11. DDR SDRAM (Double Data Rate SDRAM)

Đây là loại bộ nhớ cải tiến từ SDRAM. Nó nhân đôi tốc độ truy cập của SDRAM bằng cách dùng cả hai quá trình đồng bộ khi clock chuyển từ logic 0 sang 1 và từ logic 1

sang 0 (dùng cả cạnh âm và cạnh dương). Loại RAM này được CPU Intel và AMD hỗ trợ, tốc độ vào khoảng 266 MHz. (DDR-SDRAM đã ra đời trong năm 2000)

1.2.12. DRDRAM (Direct Rambus DRAM)

Hệ thống Rambus (tên hãng chế tạo) có nguyên lý và cấu trúc chế tạo hoàn toàn khác loại SDRAM truyền thống. Bộ nhớ sẽ được vận hành bởi một hệ thống phụ gọi là kênh truyền Rambus trực tiếp (direct Rambus channel) có độ rộng bus 16 bit và một xung clock 400MHz (có thể lên tới 800MHz). Theo lý thuyết thì cấu trúc mới nầy sẽ có thể trao đổi dữ liệu với tốc độ 400MHz x 16 bit = 400MHz x 2 bytes = 800 MBps. Hệ thống Rambus DRAM cần một chip serial presence detect (SPD) để trao đổi với motherboard. Ta thấy kỹ thuật mới nầy dùng giao tiếp 16 bit, khác hẳn với cách chế tạo truyền thống là dùng 64 bit cho bộ nhớ nên kỹ thuật Rambus cho ra đời loại chân RIMM (Rambus Inline Memory Module), khác so với bộ nhớ truyền thống. Loại RAM này chỉ được hỗ trợ bởi CPU Intel Pentum IV, tốc độ vào khoảng 400 – 800 MHz

1.2.13. SLDRAM (Synchronous - Link DRAM)

Là thế hệ sau của DRDRAM, thay vì dùng kênh Rambus trực tiếp 16 bit và tốc độ 400MHz, SLDRAM dùng bus 64 bit chạy với tốc độ 200MHz. Theo lý thuyết thì hệ thống mới có thể đạt được tốc độ 200MHz x 64 bit = 200MHz x 8 bytes = 1600 MBps, tức là gấp đôi DRDRAM. Điều thuận tiện là là SLDRAM được phát triển bởi một nhóm 20 công ty hàng đầu về vi tính cho nên nó rất da dụng và phù hợp nhiều hệ thống khác nhau.

1.2.14. VRAM (Video RAM)

Khác với bộ nhớ trong hệ thống, do nhu cầu về đồ hoạ ngày càng cao, các hãng chế tạo card đồ họa đã chế tạo VRAM riêng cho video card của họ mà không cần dùng bộ nhớ của hệ thống chính. VRAM chạy nhanh hơn vì ứng dụng kỹ thuật Dual Port nhưng đồng thời cũng đắt hơn rất nhiều.

1.2.15. SGRAM (Synchronous Graphic RAM)

Là sản phẩm cải tiến của VRAM, nó sẽ đọc và viết từng block thay vì từng mảng nhỏ.

1.2.16. Flash Memory

Là sản phẩm kết hợp giữa RAM và đĩa cứng, bộ nhớ flash có thể chạy nhanh như SDRAM mà và vẫn lưu trữ được dữ liệu khi không có nguồn cung cấp.

1.2.17. Một số thuật ngữ liên quan

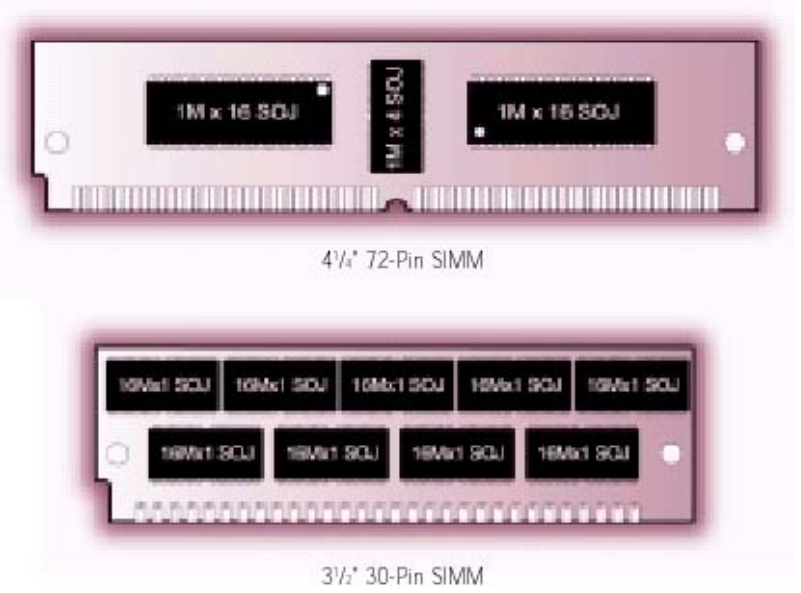
- PC66, PC100, PC133, PC1600, PC2100, PC2400:

PC66, 100, 133MHz là tốc độ của hệ thống chipset của motherboard. PC1600, PC2100, PC2400: ra đời khi kỹ thuật Rambus phát triển. Đặc điểm của loại motherboard này là dùng loại DDR SDRAM (Double Data Rate Synchronous Dynamic RAM). DDR SDRAM sẽ chạy gấp đôi (trên lý thuyết) loại RAM bình thường vì nó dùng cả cạnh dương và âm của xung clock. Do đó PC100 sẽ thành PC200 và nhân lên 8 bytes độ rộng

bus của DDR SDRAM: PC200 * 8 = PC1600. Tương tự PC133 sẽ là PC133 * 2 * 8bytes = PC2100 và PC150 sẽ là PC150 * 2 * 8 = PC2400.

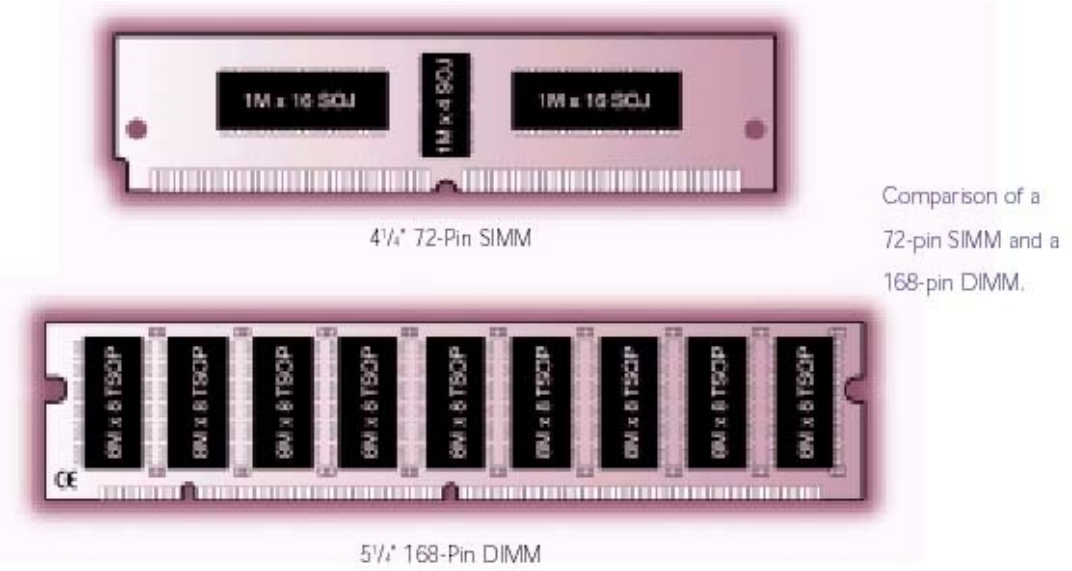
- BUS: gồm nhiều dây dẫn điện nhỏ gộp lại, là hệ thống truyền dữ liệu giữa các bộ phận trong máy tính.
- FSB (Front Side Bus): bus từ CPU tới bộ nhớ chính.
- BSB (Back Side Bus): bus từ bộ điều khiển bộ nhớ tới Cache level 2.
- Cache memory: Là loại bộ nhớ có dung lượng rất nhỏ (thường nhỏ hơn 1MB) và chạy rất nhanh (gần bằng tốc độ của CPU). Thông thường thì Cache nằm gần CPU và có nhiệm vụ cung cấp những dữ liệu thường hay đang sử dụng cho CPU. Sự hình thành của Cache là một cách nâng cao hiệu quả truy xuất của máy tính mà thôi. Những dữ liệu thường sử dụng (hoặc đang) được chứa trong Cache, mỗi khi xử lý hay thay đổi dữ liệu, CPU sẽ dò trong Cache trước xem có tồn tại hay không, nếu có nó sẽ lấy ra dùng lại còn không thì sẽ tìm tiếp vào RAM hoặc các bộ phận khác. Lấy một ví dụ đơn giản là nếu mở Microsoft Word lên lần đầu tiên sẽ thấy hơi lâu nhưng mở lên lần thứ hai thì nhanh hơn rất nhiều vì trong lần mở thứ nhất các lệnh để mở Microsoft Word đã được lưu giữ trong Cache, CPU chỉ việc tìm nó và dùng lại. Cache rất đắt tiền và chế tạo rất khó khăn bởi nó gần như là CPU (về cấu thành và tốc độ). Thông thường Cache nằm gần CPU, trong nhiều trường hợp Cache nằm bên trong CPU. Người ta gọi Cache Level 1 (L1), Cache level 2 (L2)...là do vị trí của nó gần hay xa CPU. Cache L1 gần CPU nhất, sau đó là Cache L2...
- Xen kẽ (interleave): là một kỹ thuật làm tăng tốc độ truy xuất bằng cách giảm bớt thời gian nhàn rỗi của CPU. Ví dụ, CPU cần đọc thông tin thông từ hai nơi A và B khác nhau, vì CPU chạy quá nhanh nên A chưa kịp lấy dữ liệu ra, CPU phải chờ. Khi đó CPU có thể lấy dữ liệu từ B rồi sau đó trở lại A. Do đó, CPU có thể rút bớt thời gian mà lấy được dữ liệu ở cả A và B.
- Bursting: là một kỹ thuật khác để giảm thời gian truyền tải dữ liệu trong máy tính. Thay vì CPU lấy từng byte một, bursting sẽ giúp CPU lấy thông tin mỗi lần là một block.
- ECC (Error Correction Code): là một kỹ thuật để kiểm tra và sửa lỗi trong trường hợp 1 bit nào đó của bộ nhớ bị sai giá trị trong khi lưu chuyển dữ liệu. Những loại RAM có ECC thường dùng cho server. Tuy nhiên không có ECC cũng không phải là mối lo lớn vì theo thống kê 1 bit trong bộ nhớ có thể bị sai giá trị khi chạy trong gần 750 giờ (tức là khoảng 1 tháng).
- CAS (Column Address Strobe) latency: là diễn tả thời gian trễ trong việc truy xuất dữ liệu của bộ nhớ và được tính bằng chu kỳ xung clock. Ví dụ, CAS3 là trễ 3 chu kỳ xung clock. Các nhà sản xuất cố gắng hạ thấp chỉ số trễ xuống nhưng nó sẽ tỷ lệ nghịch với giá thành sản phẩm.
- Số chân của RAM: thông thường là 30, 72, 144, 160, 168, 184.
- Cách tính dung lượng: Thông thường RAM có hai chỉ số, ví dụ, 16Mx8. Thông số đầu biểu thị số hàng của RAM trên đơn vị bit, thông số thứ nhì biểu thị số cột của RAM. 16Mx8 = 16 MegaBit x 8 cột = 128 Mega Bit = 16MB.

- SIMM (Single In-Line Memory Module): là loại ra đời sớm và có hai loại 30 hay 72 chân. Loại RAM thường tải dữ liệu mỗi lần 8 bit, sau đó phát triển lên 32 bit. Loại 72-pin SIMM có chiều rộng 4½" trong khi loại 30-pin SIMM có chiều rộng 3½".



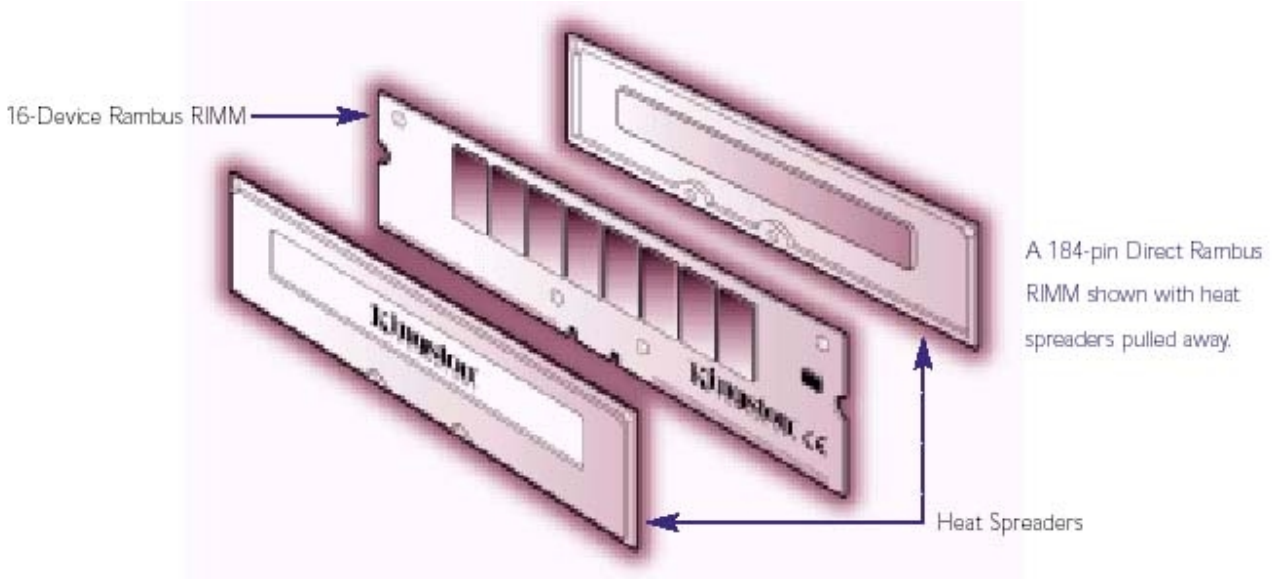
Hình 3.1 – Dạng chân của SIMM

- DIMM (Dual In-line Memory Modules): cũng gần giống như loại SIMM nhưng có số chân là 72 hoặc 168. Một đặc điểm khác để phân biệt DIMM với SIMM là các chân của SIMM dính lại với nhau tạo thành một mảng để tiếp xúc với memory slot trong khi DIMM có các chân hoàn toàn cách rời độc lập với nhau. Một đặc điểm phụ nữa là DIMM được cài đặt thẳng đứng trong khi SIMM thì ấn vào nghiêng khoảng 45^0. Thông thường loại 30 chân tải dữ liệu 16 bit, loại 72 chân tải dữ liệu 32 bit, loại 144 (dùng cho notebook) hay 168 chân tải dữ liệu 64 bit.



Hình 3.2 – Dạng chân của DIMM

- SO DIMM (Small Outline DIMM): là loại bộ nhớ dùng cho notebook, có hai loại chân là 72 hoặc 144. Loại 72 chân dùng bus 32 bit, loại 144 chân dùng bus 64 bit.
- RIMM (Rambus In-line Memory Modules) và SO RIMM (RIMM dùng cho notebook): là kỹ thuật của hãng Rambus, có 184 chân (RIMM) và 160 chân (SO RIMM) và truyền dữ liệu mỗi lần 16 bit (thế hệ cũ chỉ có 8 bit) nên chạy nhanh hơn các loại cũ. Tuy nhiên do chạy với tốc độ cao, RIMM tụ nhiệt rất cao nên cách chế tạo cũng phải khác so với các loại RAM truyền thống. Như hình vẽ bên dưới bạn sẽ thấy RAM có hai thanh giải nhiệt kẹp hai bên gọi là heat speader.



Hình 3.3 – Dạng chân của RIMM

2. Bộ nhớ trong

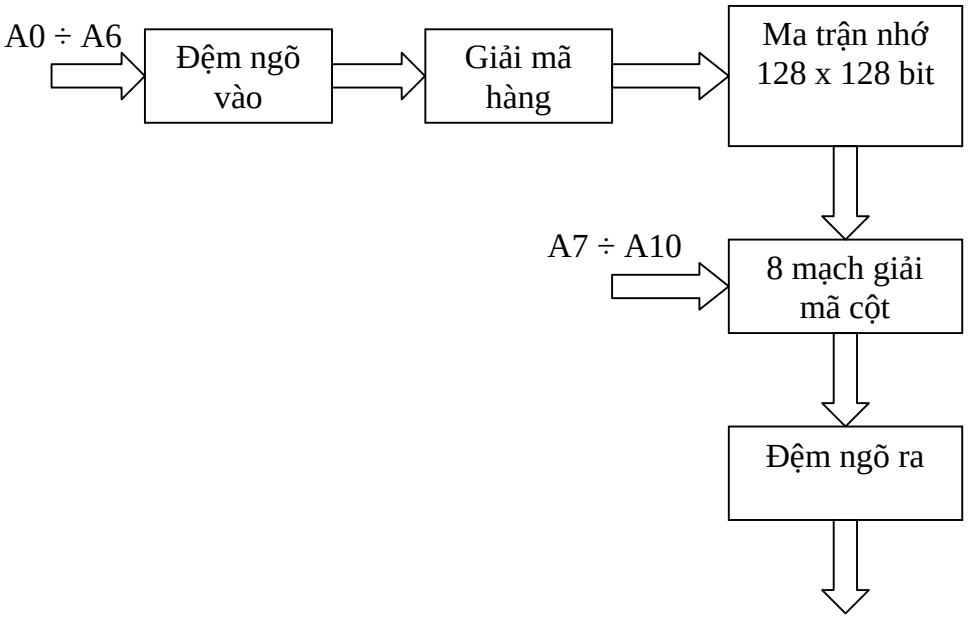
2.1. Tổ chức bộ nhớ

Bộ nhớ thường được tổ chức từ nhiều vi mạch nhớ ghép lại để có độ rộng bus địa chỉ và dữ liệu cần thiết. Các chip nhớ có đầy đủ chức năng của một bộ nhớ bao gồm:

- Ma trận nhớ: gồm các ô nhớ, mỗi ô nhớ tương ứng với một bit nhớ.
- Mạch giải mã địa chỉ cho bộ nhớ.
- Mạch logic cho phép đọc.
- Mạch logic cho phép ghi.
- Các mạch đệm vào, ra.

Cách tổ chức đơn giản nhất là tổ chức theo word. Một ma trận nhớ có độ dài của cột bằng số lượng word W và độ dài hàng bằng số lượng bit B của một word. Phương pháp này có thời gian truy xuất ngắn nhưng đòi hỏi bộ giải mã lớn khi tổng số word lớn.

Phương pháp giải mã hai bước cho phép giảm kích thước của phần giải mã địa chỉ bằng cách sử dụng khái niệm word logic và word vật lý. Word vật lý bao gồm tất cả các bit trong một hàng của ma trận và word logic là số bit tương ứng được gởi ra đồng thời. Lúc này, bộ nhớ cần hai mạch giải mã: giải mã hàng để chọn word vật lý và mạch giải mã cột có các mạch dồn kênh (multiplexer) chọn một word logic từ một word vật lý. Ví dụ như: 1 RAM dung lượng 2048 x 8 được tổ chức giải mã 2 bước như hình vẽ:



Hình 3.4 – Giải mã hai bước cho bộ nhớ

Ma trận nhớ là 128 x 128 bit, có $128 = 2^7$ word vật lý. Một word vật lý được chọn bởi 7 đường địa chỉ từ A0 ÷A6. Độ giải mã hàng chọn 1 hàng từ 128 hàng.

Một word vật lý được chia làm 16 nhóm 8 bit. Nhóm thứ nhất chứa bit cao nhất của 16 word logic. Nhóm thứ 2 chứa các bit tiếp theo và nhóm cuối cùng chứa các bit thấp nhất. Như vậy mạch giải mã cột gồm 8 bộ dồn kênh 1 → 16 để cung cấp 1 word logic 8 bit. Các bit địa chỉ từ A7÷A10 đều khiển mạch giải mã cột. Trong trường hợp đặc biệt khi số phần tử trong 1 word vật lý bằng số bit trong 1word vật lý thì đó là bộ nhớ tổ chức theo bit nghĩa là mỗi word logic có độ dài 1 bit.

Các mạch điệm ngõ ra đảm bảo không những mức logic mong muốn và cung cấp đủ dòng mà còn có ngõ ra cực thu hở hay ba trạng thái cho phép kết nối chung với môt vài bộ nhớ khác. Mạch đệm ngõ ra được điều khiển bằng các tín hiệu chọn mạch $\overline{CS}$, cho phép bộ nhớ $\overline{CE}$, cho phép ngõ ra $\overline{OE}$.

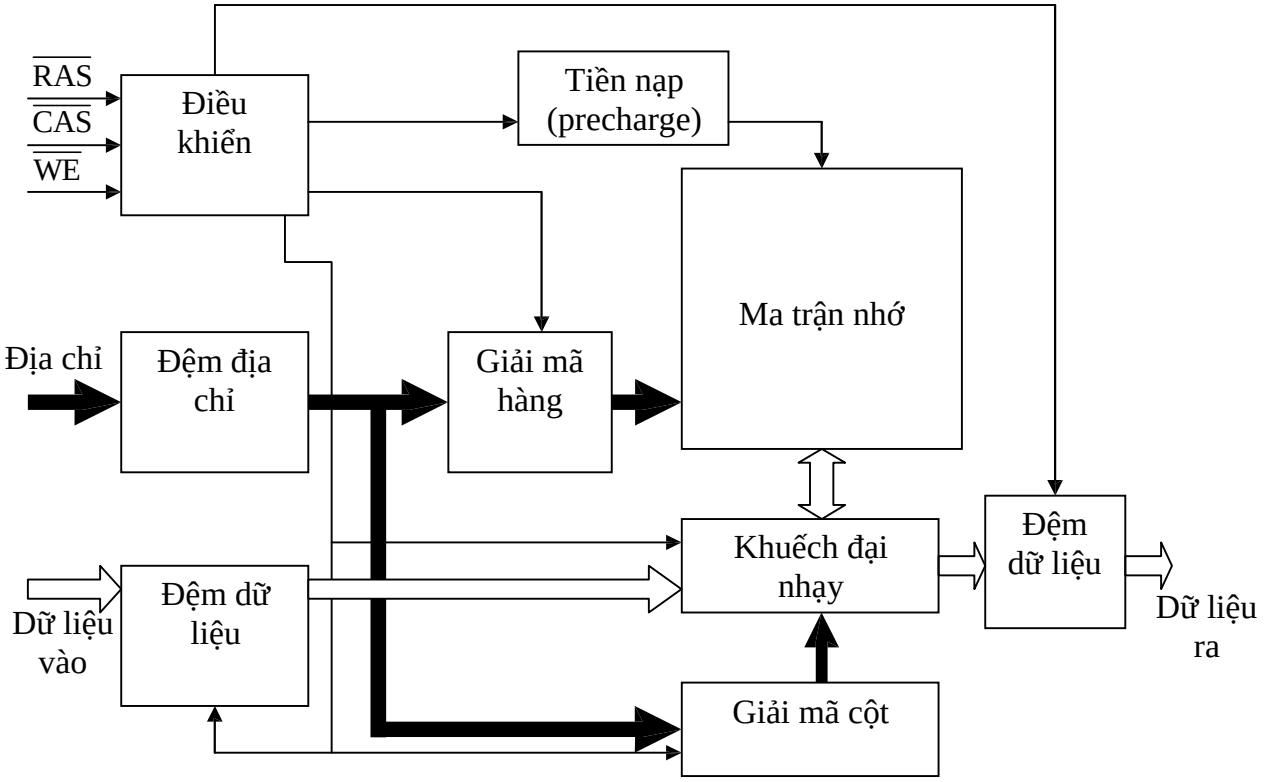
2.2. DRAM

2.2.1. Cấu tạo của DRAM

Địa chỉ xác định ô nhớ chia thành 2 phần: địa chỉ hàng và cột. Hai địa chỉ này được đưa lần lượt vào bộ đệm. Quá trình dồn kênh địa chỉ điều khiển bằng các tín hiệu $\overline{RAS}$ (Row Access Strobe) và $\overline{CAS}$ (Column Access Strobe). Bộ điều khiển nhớ của CPU phải

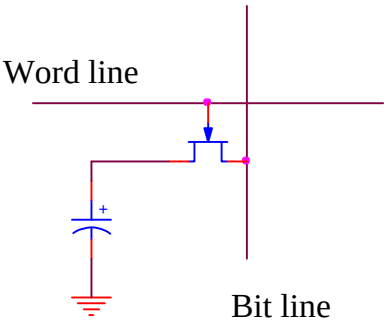
thực hiện 3 công việc sau: chia địa chỉ từ CPU thành các địa chỉ hàng và cột; tích cực các chân $\overline{RAS}$, $\overline{CAS}$ và $\overline{WE}$ một cách chính xác; truyền và nhận các dữ liệu đọc, ghi.

Sơ đồ tổ chức của một DRAM:



Hình 3.5 – Sơ đồ cấu tạo DRAM

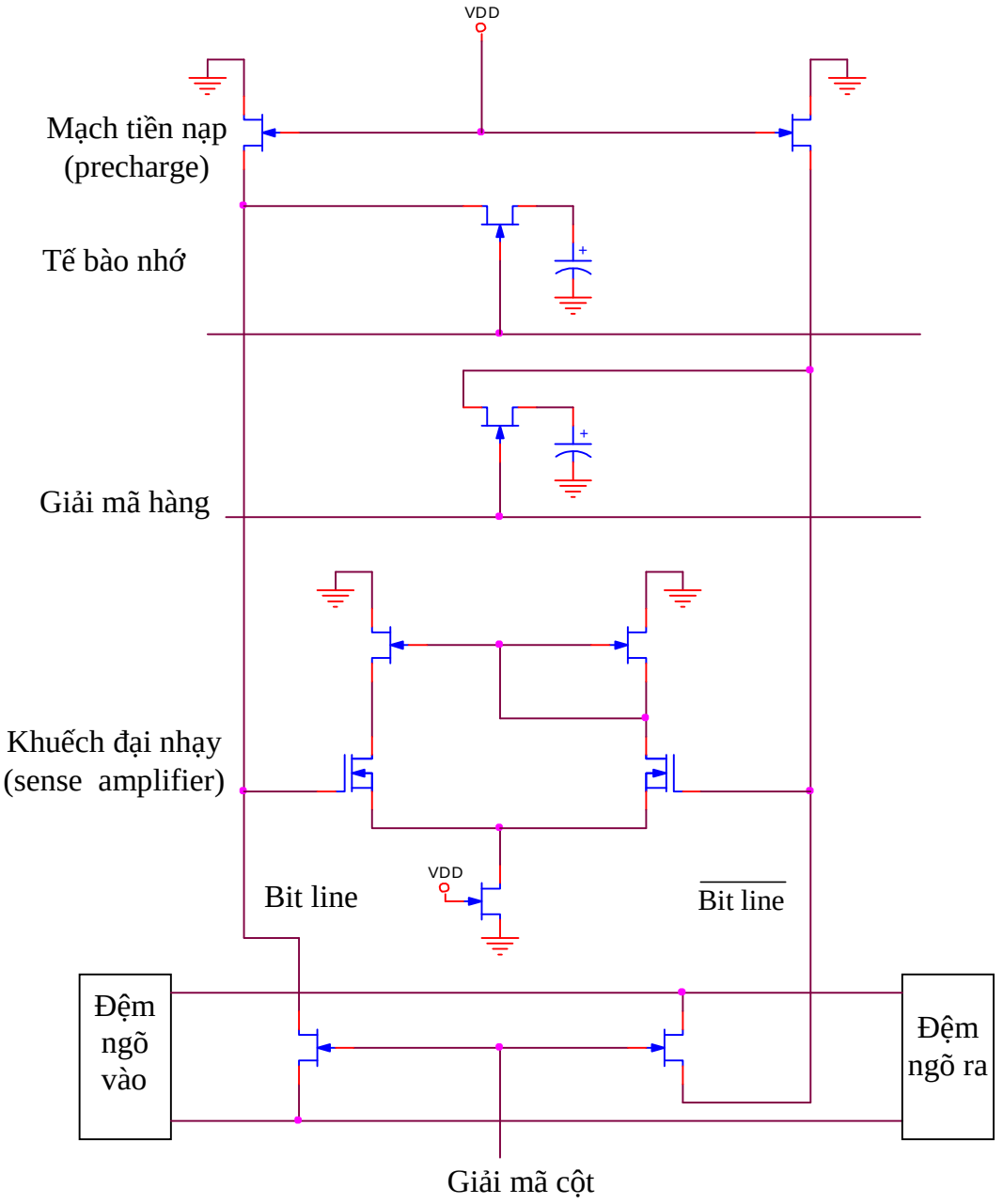
Một ô nhớ của DRAM có thể biểu diễn như hình vẽ:



Hình 3.6 – Cấu tạo một tế bào nhớ DRAM

FET hoạt động như một công tắc, khi đường word tích cực thì cho phép mở FET. Do hiện tượng rò rỉ trên FET, nên sau một thời gian, điện áp trên tụ sẽ giảm dần.

2.2.2. Quá trình đọc / ghi bộ nhớ



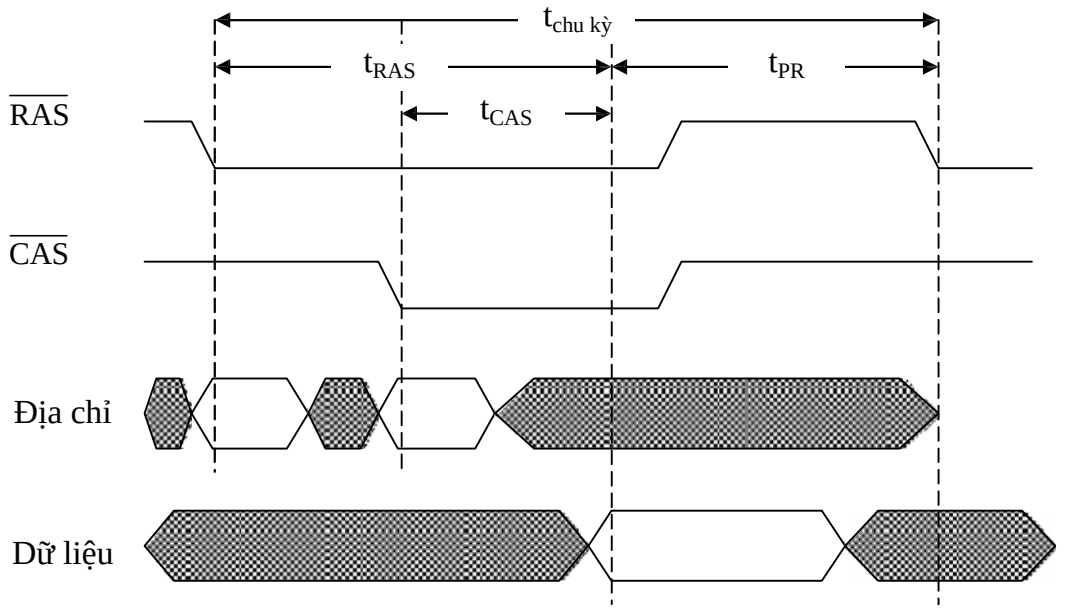
Hình 3.7 – Sơ đồ nguyên lý của DRAM

Mạch tiền nạp cho phép xâm nhập và kích hoạt word line khi nạp tất cả các bit line bằng $V_{cc}/2$. Quá trình tiền nạp sẽ dừng khi các cặp dây này có giá trị bằng nhau về điện áp. Thời gian đòi hỏi cho quá trình này gọi là thời gian tiền nạp RAS. Khi quá trình này kết thúc thì mới có thể thực hiện truy xuất ô nhớ. Mạch tiền nạp làm tăng tính ổn định của điện áp ra.

Do điện dung của tụ điện rất nhỏ so với điện dung ghép giữa các bit line nên điện áp trên các bit line thay đổi nhỏ, khoảng 100 mV. Nếu điện tích của tụ điện ban đầu bằng

0 thì điện thế trên bit line sẽ giảm xuống và kéo điện áp này xuống mức 0. Ngược lại, nếu điện tích khác 0 thì điện thế trên bit line sẽ tăng lên và nâng điện áp này lên V_{cc}. Tín hiệu giải mã cột được cấp sau tín hiệu giải mã hàng để cho phép chọn chính xác cột.

Giản đồ thời gian đọc dữ liệu của DRAM có thể biểu diễn như sau:



Hình 3.7 – Giản đồ thời gian đọc DRAM

t_{RAS}: thời gian thâm nhập RAS – là thời gian từ lúc đặt địa chỉ hàng cho tới khi dữ liệu ra khỏi bộ đệm.

t_{CAS}: thời gian thâm nhập CAS – là thời gian từ lúc đặt địa chỉ cột cho tới khi dữ liệu ra khỏi bộ đệm ($t_{CAS} < t_{RAS}$).

t_{PR}: thời gian hồi phục (thời gian tiền nạp RAS) – thời gian cần thiết từ lúc ngõ ra ổn định cho đến khi có thể cung cấp một địa chỉ mới.

2.2.3. Làm tươi DRAM

Điện tích trên tụ điện bị giảm theo thời gian do chúng phóng qua FET và lớp điện môi oxide làm cho dữ liệu có thể bị mất. Do đó, tụ điện phải được nạp lại một cách tuần hoàn (refresh), thông thường khoảng từ 1 ÷ 16 ms tùy theo loại RAM. Có 3 phương pháp refresh:

- ***Chỉ tác động RAS***: tạo chu kỳ đọc giả (dummy read) để làm tươi ô nhớ. Trong chu kỳ giả này, tín hiệu $\overline{RAS}$ tích cực và địa chỉ hàng được đưa vào DRAM nhưng $\overline{CAS}$ bị cấm nên không thể truyền dữ liệu ra ngoài được. Để làm tươi toàn bộ bộ nhớ thì tất cả các địa chỉ phải được cấp lần lượt. Nhược điểm của phương pháp này là cần phải dùng mạch logic hay một chương trình để làm tươi. Trong máy tính, điều này thực hiện bằng kênh 0 của DMAC 8237, tác động định kỳ bằng bộ đếm 1 của PIT 8253.

- ***Tác động CAS trước RAS***: DRAM có một mạch logic làm tươi của riêng nó với một bộ đếm địa chỉ. Tín hiệu $\overline{CAS}$ sẽ tích cực trong một khoảng thời gian trước khi $\overline{RAS}$ tích cực. Địa chỉ làm tươi được phát ra bên trong bằng bộ đếm địa chỉ mà không cần mạch logic bên ngoài. Sau mỗi chu kỳ làm tươi, bộ đếm địa chỉ sẽ tự động tăng lên 1 để chỉ địa chỉ kế tiếp.
- ***Ẩn***: Chu kỳ làm tươi được chứa sau chu kỳ đọc bộ nhớ. Tín hiệu $\overline{CAS}$ giữ nguyên mức thấp trong khi chỉ có $\overline{RAS}$ lên mức cao. Ở đây, bộ đếm địa chỉ cũng tự phát ra địa chỉ làm tươi.Việc đọc dữ liệu trong chu kỳ đọc cũng có thể thực hiện ngay cả khi đang thực hiện chu kỳ làm tươi. Phương pháp này sẽ tiết kiệm được thời gian do thời gian làm tươi thường ngắn hơn so với thời gian đọc.

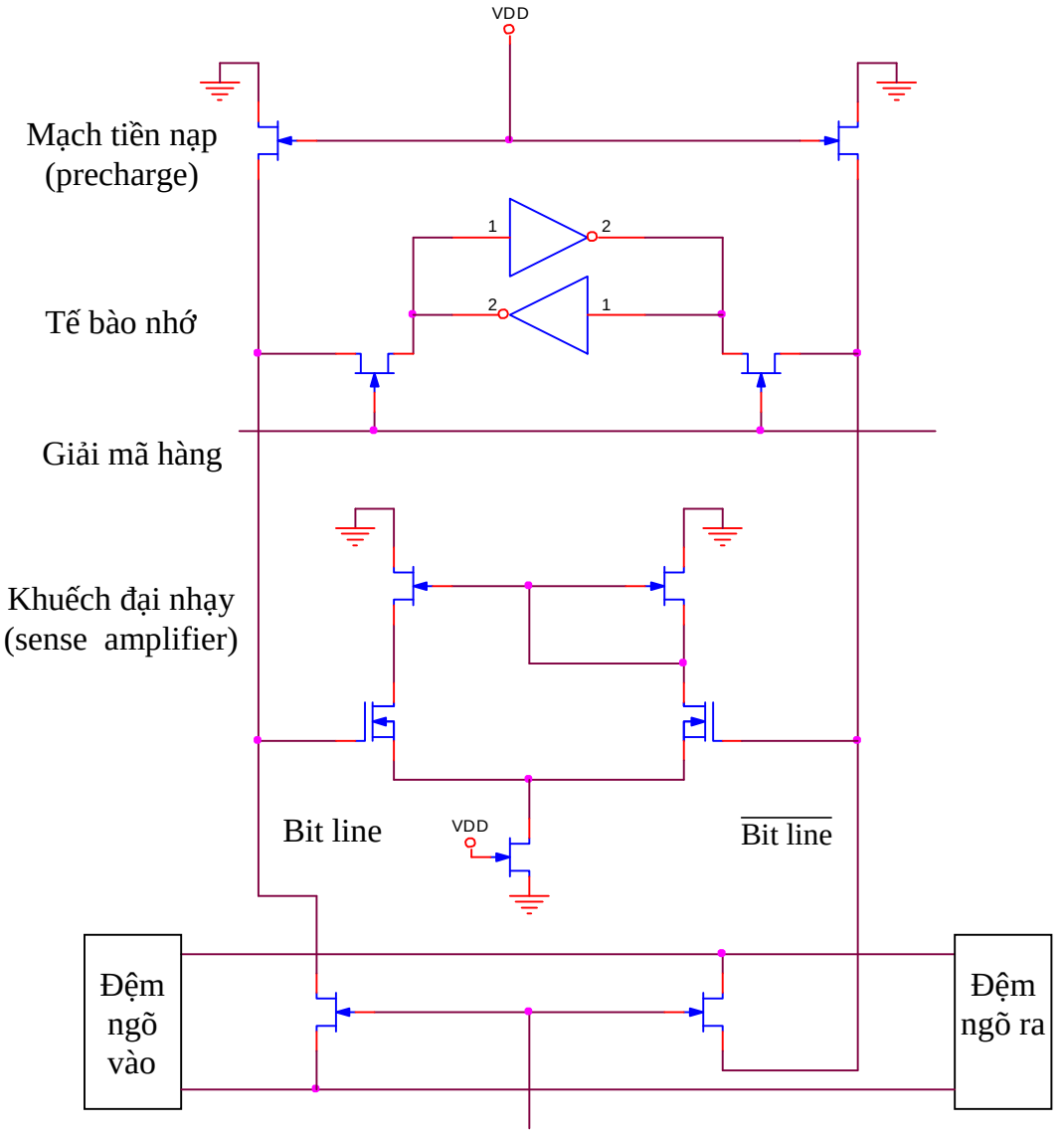
2.2.4. Các chế độ hoạt động nhanh của DRAM

DRAM có khả năng thực hiện một hay nhiều chế độ cột để giảm thời gian t_{CAS}.

- ***Chế độ trang***: quá trình truy xuất ô nhớ chỉ thay đổi địa chỉ cột. Như vậy, một trang sẽ tương ứng với một hàng trong ma trận nhớ. Để khởi động quá trình đọc, mạch điều khiển nhớ tác động tín hiệu $\overline{RAS}$ như thông thường nhưng lại bỏ qua địa chỉ hàng. Trong chế độ này, nếu ô nhớ tiếp theo trong cùng một trang thì tín hiệu $\overline{RAS}$ vẫn giữ liên tục ở mức tích cực. Do đó, thời gian truy xuất có thể giảm xuống 50%.
- ***Chế độ cột tĩnh***: tương tự như chế độ trang nhưng tín hiệu $\overline{CAS}$ giữ nguyên không đổi. DRAM sẽ tự phát hiện sự thay đổi địa chỉ sau một khoảng thời gian $\overline{CAS}$ không đổi.
- ***Chế độ nibble***: thay đổi tín hiệu $\overline{CAS}$ 4 lần để chuyển 1 nibble dữ liệu.
- ***Chế độ nối tiếp***: tương tự như chế độ nibble nhưng không phải chỉ chuyển 4 lần trạng thái của tín hiệu $\overline{CAS}$. Về nguyên tắc, toàn bộ hàng có thể đưa ra tuần tự.
- ***Chế độ xen kẽ***: chế độ này là phương pháp hạn chế trễ do thời gian tiền nạp RAS. Bộ nhớ chia thành vài bank xen kẽ nhau theo một tỷ số xác định. Dữ liệu có địa chỉ chẵn đặt ở bank 0 và địa chỉ lẻ đặt ở bank 1. Khi đó, thời gian tiền nạp của bank 0 là thời gian truy xuất của bank 1 và ngược lại.

2.3. SRAM

Đối với SRAM, nội dung ô nhớ vẫn giữ nguyên khi chưa mất nguồn cung cấp mà không cần phải tốn thời gian làm tươi ô nhớ. Do điện áp chênh lệch lớn nên thời gian xử lý khuếch đại sẽ nhỏ hơn trong DRAM (thời gian truy xuất của DRAM là khoảng 1ns trong khi của DRAM khoảng 40ns). Từ đó, SRAM không thực hiện phân kênh địa chỉ hàng và cột (điều này sẽ làm giảm thời gian truy xuất). Sau khi dữ liệu ổn định, bộ giải mã cột chọn cột phù hợp và đưa dữ liệu đến bộ đệm ngõ ra.



Hình 3.8 – Sơ đồ nguyên lý của SRAM

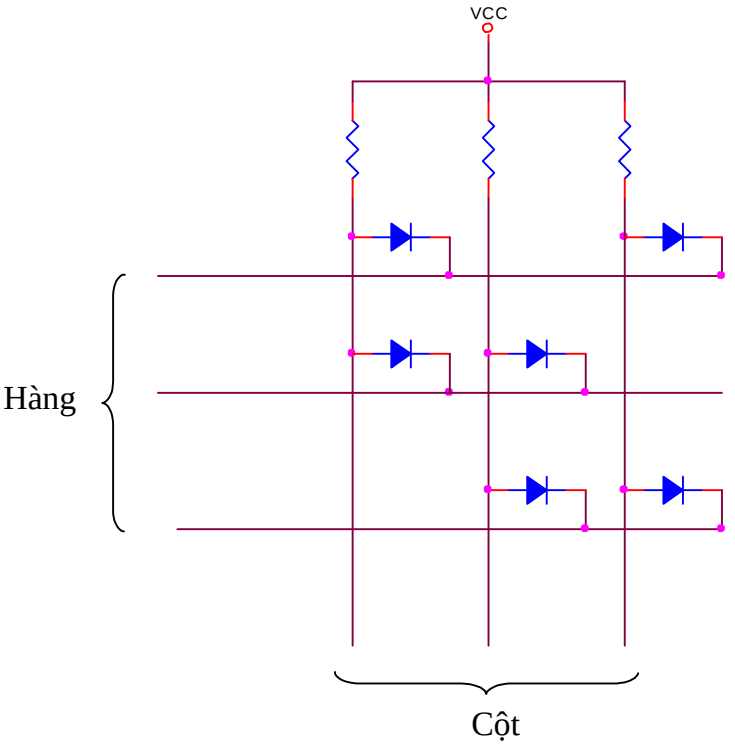
2.4. ROM

RAM không thích hợp cho các chương trình khởi động do dữ liệu trên RAM sẽ mất khi không cung cấp điện. Do đó, phải dùng đến ROM, là chip nhớ trong đó dữ liệu sẽ được lưu trữ mãi không cần duy trì nguồn cung cấp.

2.4.1. ROM

ROM được chế tạo trên một phiến Silic theo một số bước xử lý như quang khắc và khuếch tán để tạo các tiếp xúc bán dẫn dẫn điện một chiều (diode, FET, …). Người thiết

kế xác định chương trình muốn ghi vào ROM và thông tin này dùng để điều khiển trong quá trình tạo ROM. Sơ đồ đơn giản về ROM có thể biểu diễn như sau:



Hình 3.9 – ROM đơn giản dùng diode

Giao giữa một hàng và một cột là một vị trí ô nhớ. Nếu vị trí này tồn tại một diode thì sẽ lưu trữ dữ liệu 0, ngược lại thì lưu trữ dữ liệu 1. Khi đọc một ô nhớ nào đó trên một hàng thì bộ giải mã sẽ đặt hàng đó xuống mức 0, các hàng còn lại ở mức logic 1. Nếu giao giữa hàng và cột không có diode (nghĩa là lưu trữ dữ liệu 1) thì giá trị tương ứng trên cột là 1 (do cột nối với V_{cc} thông qua điện trở R). Nếu có diode (lưu trữ dữ liệu 0) thì diode sẽ phân cực thuận nên điện áp rơi trên diode khoảng 0.7V Æ điện áp trên cột cũng là 0.7V tương ứng với mức logic 0.

Công nghệ chế tạo ROM được sử dụng là lưỡng cực và MOS. Thời gian truy xuất của bộ nhớ lưỡng cực khoảng từ 50 ÷ 90 ns còn của MOS lớn hơn khoảng 10 lần nhưng bộ nhớ MOS có kích thước nhỏ hơn và tiêu thụ năng lượng ít hơn.

2.4.2. PROM

Cũng tượng tự như ROM nhưng tất cả các điểm giao giữa hàng và cột đều có một diode mắc nối tiếp với một cầu chì. Khi chưa lập trình, các cầu chì vẫn còn nguyên thì nội dung trên ROM là 0. Khi lập trình, nếu cần bit nào bằng 1 thì ta đặt một xung điện ứng với bit đó, cầu chì sẽ bị đứt và xem như tại vị trí này không có diode (ứng với mức logic 1).

2.4.3. EPROM

Đối với EPROM, dữ liệu có thể ghi vào bằng điện và có thể xóa được. EPROM sử dụng một transistor có cấu trúc FAMOST (Floating gate avalanche injection MOS transistor). Đối với transistor loại này, cấu tạo cũng giống như dạng FET thông thường nhưng trong cực gate tồn tại thêm một cực, gọi là cực nổi (floating gate). Nếu cực nổi không có điện tích thì transistor này hoạt động như một FET thông thường, nghĩa là nếu cực Gate có điện áp dương thì FET dẫn, cực drain nối với cực source Æ cực drain có mức logic 1. Nếu cực nổi có chứa điện tích thì nó sẽ tạo ra trường điện từ đủ sức ngăn chặn không cho FET dẫn Æ cực drain có mức logic 0.

Quá trình nạp điện tử vào cửa nổi được thực hiện bằng các xung điện có độ rộng khoảng 50 ms và biên độ 20V đặt vào cực gate và cực drain. Điện tử được lưu trữ tại vùng cửa nổi khi xung điện tắt (thời gian lưu trữ ít nhất là 10 năm). Để xóa dữ liệu trên EPROM, ta phải chiếu ánh sáng tử ngoại vào chip nhớ. Các điện tử sẽ hấp thu năng lượng đủ để có thể vượt ra khỏi cực nổi làm cho cực nổi không chứa điện tử Æ toàn bộ EPROM chứa giá trị 1. Do đó, trên các chip EPROM sẽ có một cửa số bằng thạch anh cho phép ánh sáng tử ngoại đi qua.

2.4.4. EEPROM

Do việc sử dụng cửa sổ thạch anh không tiện lợi nên những năm gần đây đã xuất hiện them chip ROM có thể xóa bằng điện. Cấu tạo của EEPROM cũng giống như EPROM nhưng lúc này có thêm một lớp màng mỏng oxide giữa vùng cực nổi và cực drain cho phép các điện tử di chuyển từ vùng cực nổi sang cực drain khi đặt một điện áp âm. Như vậy, quá trình lập trình cho EEPROM tương tự như EPROM trong khi đó để xóa dữ liệu thì chỉ cần đặt một điện áp -20V vào cực gate và cực drain vói thời gian thích hợp (vì nếu thời gian quá dài thì các điện tử sẽ di chuyển thêm sang cực drain làm cho cực nổi sẽ có điện tích dương Æ FET sẽ không hoạt động như bình thường).

2.5. Bộ nhớ Flash và bộ nhớ Cache

Bộ nhớ flash có cấu trúc giống như EEPROM nhưng lớp oxide mỏng hơn nên chỉ cần điện áp 12V là có thể xóa được dữ liệu. Bộ nhớ flash có thể hoạt động như RAM nhưng có thể lưu trữ được dữ liệu khi mất nguồn cung cấp. Thành phần chính của flash là ma trận nhớ gồm các ô nhớ FAMOST và không thực hiện phân kênh địa chỉ. Quá trình đọc dữ liệu thực hiện bằng điện áp 5V và lập trình dùng điện thế 12V. Thời gian lưu trữ của flash ít nhất là 10 năm.

Do sự ra đời của các CPU tốc độ nhanh, khi sử dụng các DRAM thì tốc độ đáp ứng của DRAM không theo kịp tốc độ của CPU nên hoạt động của hệ thống sẽ bị chậm lại làm giảm hiệu suất của máy tính (do phải thêm vào các khoảng thời gian chờ). Một giải pháp có thể thực hiện là thay bằng các SRAM có tốc độ cao hơn nhưng lại đòi hỏi giá thành cao. Bộ nhớ cache ra đời bằng cách kết hợp tốc độ cao của SRAM và giá thành rẻ của DRAM. Cache sẽ được đặt giữa CPU và bộ nhớ chính DRAM, đó là bộ nhớ SRAM có dung lượng nhỏ, bộ nhớ này sẽ lưu trữ các dữ liệu tạm thời mà CPU thường sử dụng nhằm làm giảm thời gian chờ đợi của CPU. Khi CPU cần dữ liệu, nó sẽ đọc dữ liệu trước từ cache.

3. Bộ nhớ ngoài

Bộ nhớ chính bằng vật liệu bán dẫn không thể lưu trữ một khối lượng rất lớn dữ liệu nên cần phải có thêm các thiết bị nhớ bên ngoài như băng giấy đục lỗ, băng cassette, trống từ, đĩa từ, đĩa quang, … Các thiết bị lưu trữ này còn được gọi là bộ nhớ khối (mass storage). Thiết bị nhớ khối thông dụng nhất là đĩa từ. Đĩa từ là một tấm đĩa tròn, mỏng làm bằng chất dẻo, thủy tinh cứng hay kim loại cứng, trên đó có phủ một lớp bột từ tính oxide sắt từ. Đĩa từ sử dụng kỹ thuật ghi từ để lưu trữ dữ liệu. Khi đã ghi dữ liệu trên đĩa, dữ liệu có thể tồn tại khi không còn nguồn cung cấp và cũng có khả năng xóa đi, thay thế bằng dữ liệu mới.

3.1. Đĩa mềm và ổ đĩa mềm (Floppy disk and floppy disk drive)

3.1.1. Đĩa mềm

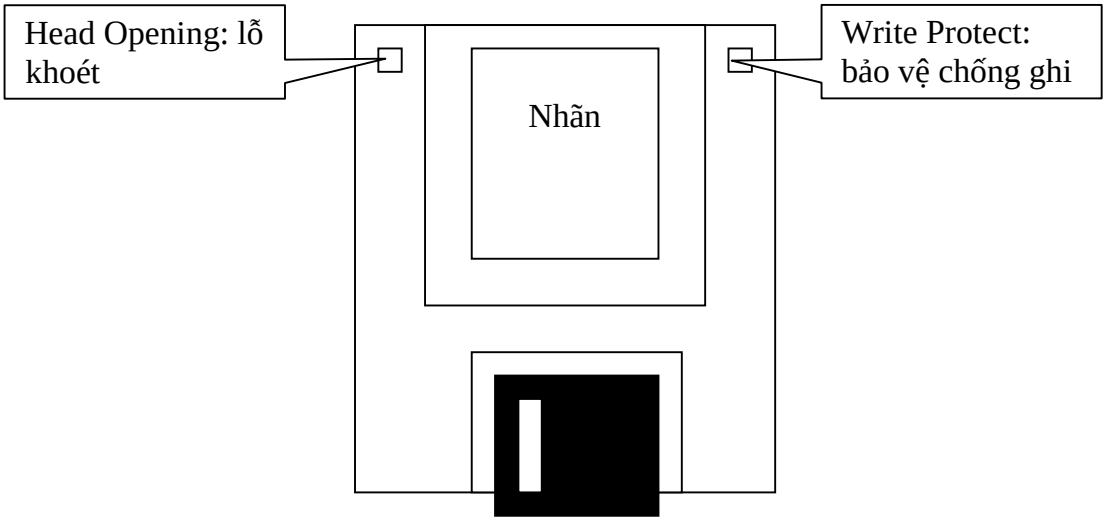
Đĩa mềm gồm một đĩa từ bằng nhựa dẻo được bảo vệ bằng một bao giấy hay nhựa cứng. Trên bao có khoét một lỗ dài cho phép đầu đọc của ổ đĩa có thể tiếp xúc với mặt đĩa để đọc / ghi dữ liệu. Có 2 loại đĩa mềm: đường kính 5.25 inch (hầu như không còn sử dụng) và đường kính 3.5 inch (chỉ dùng dung lượng 1.44 MB).

Mỗi đĩa mềm được tổ chức thành các đơn vị sau:

Track (rãnh từ): là vùng đường tròn đồng tâm lưu trữ dữ liệu. Mật độ ghi dữ liệu tính bằng đơn vị track/inch. Track được đánh số bắt đầu từ 0 kể từ vòng ngoài vào.

Sector (cung từ): mỗt track sẽ được chia thành nhiều sector, mỗi sector chứa 512 bytedữ liệu. Số sector/track tùy thuộc vào từng loại đĩa (từ 8 ÷ 36). Sector được đánh số từ 1.

Cluster (liên cung): là một nhóm gồm 2, 4 hay 8 sector.



Hình 3.10 – Đĩa mềm 3.5 inch

Dung lượng đĩa mềm = số track× số sector/track × số mặt× 512 byte

Loại đĩa	Dung lượng	Số track	Số sector/track	Tổng số sector	Track/inch
5.25 SS/SD	160KB	40	8	320	48
5.25 SS/DD	180KB	40	9	360	48
5.25 DS/DD	320KB	40	8	640	48
5.25 DS/DD	360KB	40	9	720	48
5.25 DS/HD	1.2MB	80	15	2400	48
3.5 DS/DD	720KB	80	9	1440	135
3.5 DS/HD	1.44MB	80	18	2880	270
3.5 DS/ED	2.88MB	80	36	5760	540

SS: Single Side DS: Double Side SD: Single Density DD: Double Density

HD: High Density ED: Extra High Density

Chương trình định dạng đĩa mềm (format) cho phép tạo ra các track và sector trên đĩa. Ngoài 512 byte dữ liệu, các track và sector còn chứa các byte lưu trữ thông tin dùng cho mục đích định vị và đồng bộ.

3.1.2. Ổ đĩa mềm

Ổ đĩa mềm cho phép CPU đọc / ghi dữ liệu lên đĩa mềm. Khi đó, đĩa được quay bằng một động cơ điều khiển với tốc độ 300 vòng/phút đối với đĩa 300 KB hay 360 vòng/phút đối với các loại đĩa khác. Đĩa mềm có hai mặt thì phải cần hai đầu từ đọc / ghi dữ liệu. Đầu từ được gắn ở đầu cần truy xuất (arm access). Chuyển động quay của một động cơ bước sẽ biến thành chuyển động tịnh tiến theo phương bán kính của cần truy xuất qua cơ cấu bánh răng. Đầu từ có một cuộn dây cảm ứng. Khi đọc, sự biến thiên từ thông của phần tử lưu trữ tạo thành điện thế cảm ứng ở hai đầu ra cuộn dây tạo nên tín hiệu dữ liệu. Khi ghi, cuộn dây sẽ phát ra từ trường qua khe từ để từ hóa bột oxide sắt trên mặt đĩa thành các trạng thái tương ứng với mức dữ liệu 0 và 1.

Mạch điều khiển ổ đĩa mềm:

Mạch điều khiển ổ đĩa mềm thường được cắm trên một khe cắm mở rộng. Bộ điều khiển có một vi xử lý riêng với chương trình trong ROM của nó, thông thường là NEC µPD765 hay Intel 8207A. Việc ghi dữ liệu được thực hiện qua các bước sau:

- Dữ liệu truyền từ bus vào bộ giao tiếp bus.
- Bộ điều khiển xác định byte CRC (Cycle Redundancy Check), đổi dữ liệu song song thành nối tiếp và định dạng thích hợp
- Bộ tách dữ liệu đổi chuỗi dữ liệu thành chuỗi mã FM hay MFM (Modified FM) và phát ra các xung đánh dấu.
- Mạch giao tiếp SA-450 truyền chuỗi dữ liệu đã mã hóa tới ổ đĩa.
- Đầu từ ghi số liệu đã mã hóa lên đĩa.

Địa chỉ các thanh ghi của mạch điều khiển ổ đĩa mềm:

	Sơ cấp	**Thứ cấp**	**R/W**
Địa chỉ cơ sở	3F0h	370h	
Thanh ghi trạng thái A	3F1h	371h	R R
Thanh ghi trạng thái B	3F1h	371h	R/W
Thanh ghi ngõ ra số DOR	3F2h	372h	R
Thanh ghi trạng thái chính	3F4h	374h	W
Thanh ghi chọn tốc độ truyền dữ liệu	3F4h	374h	R/W
Thanh ghi dữ liệu	3F5h	375h	R
Thanh ghi ngõ vào số	3F7h	377h	W
Thanh ghi điều khiển cấu hình	3F7h	377h	
Kênh DMA	2	2	
Yêu cầu ngắt IRQ	6	6	
Ngắt INTR	0Eh	0Eh	

Thanh ghi ngõ ra số (DOR - Digital Output Register): điều khiển động cơ, chọn ổ đĩa và khởi tạo bộ điều khiển

D7	D6	D5	D4	D3	D2	D1	D0
MOTD	MOTC	MOTB	MOTA	$\overline{\text{DMA}}$	$\overline{\text{RESET}}$	DR1	DR0

MOTD, MOTC, MOTB, MOTA: điều khiển động cơ (motor) cho các ổ đĩa

$\overline{\text{DMA}}$: cho phép (=1) hay cấm (=0) kênh DMA và IRQ

$\overline{\text{RESET}}$: cho phép (=1) hay cấm (=0) reset bộ điều khiển

DR1, DR0: chọn ổ đĩa 00 (A), 01 (B), 10 (C), 11 (D)

Thanh ghi trạng thái chính (main status register): là thanh ghi chỉ đọc, chứa thông tin về bộ điều khiển đĩa mềm

D7	D6	D5	D4	D3	D2	D1	D0
MRQ	DIO	NDMA	BUSY	ACTD	ACTC	ACTB	ACTA

MRQ: sẵn sàng (=1) hay không sẵn sàng (=0) ghi dữ liệu.

DIO: chiều truyền từ bộ điều khiển tới CPU (=1) hay ngược lại (=0).

NMDA: có chế độ DMA (=0) hay không (=1).

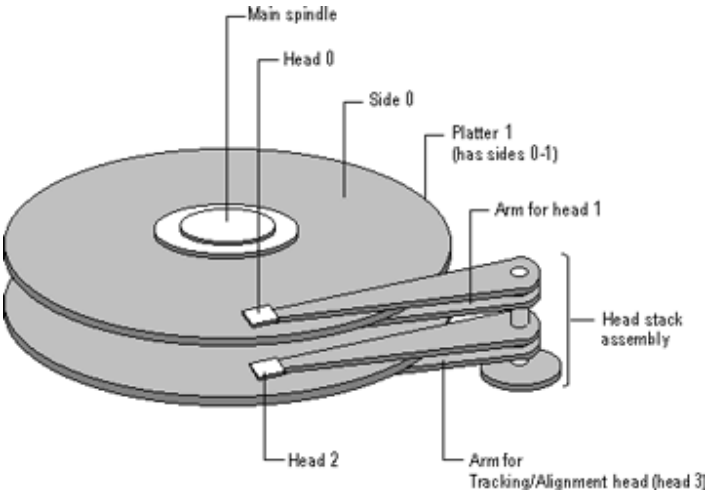
BUSY: kích hoạt về lệnh (=1) hay không (=0).

ACTA, ACTB, ACTC, ACTD: chọn ổ đĩa tương ứng (=1).

3.2. Đĩa cứng và ổ đĩa cứng

3.2.1. Cấu tạo

Đĩa cứng gồm một hay nhiều đĩa từ bằng kim loại hay nhựa cứng được xếp thành một chồng theo một trục đứng và được đặt trong một hộp kín. Dung lượng đĩa cứng lớn hơn nhiều so với đĩa mềm. Ổ đĩa cứng có nhiều đầu từ, các đầu từ này gắn trên một cần truy xuất và di chuyển thành một khối. Khi đĩa quay, đầu từ không chạm vào mặt đĩa mà cách một lớp đệm không khí. Khoảng cách giữa mặt đĩa và đầu từ tùy theo tốc độ quay và mật độ ghi dữ liệu của đĩa và rất nhỏ so với kích thước đĩa (khoảng 0.3 µm).



Hình 3.11 – Cấu tạo đĩa cứng

Đĩa cứng cũng được phân thành các đơn vị vật lý như đĩa mềm. Ngoài ra, nó còn một khái niệm nữa là *cylinder*. Cylinder là vị trí của đầu từ khi di chuyển trên các mặt tạo thành một hình trụ, đó là một chồng các track xếp nằm lên nhau đối với một vị trí đầu từ.

Dung lượng đĩa cứng = số head× số cylinder× số sector/track × số mặt× 512 byte

Tốc độ quay của đĩa cứng là 3600 vòng/phút nên thời gian truy xuất của đĩa cứng nhanh hơn đĩa mềm nhiều. *Thời gian truy xuất dữ liệu* (data access time) là một thông số quan trọng của đĩa cứng, bao gồm *thời gian tìm kiếm* (seek time), *thời gian chuyển đầu từ* (head switch time) và *thời gian quay trễ* (rotational latency). Thời gian tìm kiếm là thời gian chuyển đầu từ từ một track này sang track khác. Thời gian chuyển đầu từ là thời gian chuyển giữa hai trong số các đầu từ khi đọc hay ghi dữ liệu. Thời gian quay trễ là thời gian tính từ khi đầu từ được đặt trên một track cho đến khi tới được sector mong muốn.

3.2.2. Định dạng cấp thấp (low – level format)

Đĩa cứng phải được định dạng cấp thấp trước khi sử dụng. Đó là tạo ra các track và sector trên đĩa bằng cách ghi lên đĩa các thông tin liên quan đến chúng. Với các sector, thông tin này được ghi vào vùng *tiêu đề nhận dạng sector*, được đặt ở đầu mỗi sector. Vùng tiêu đề này chứa các thông tin như số thứ tự đầu từ, số sector, số cylinder, khai báo nhận dạng ID và mã CRC để phát hiện lỗi dữ liệu.

Trong đĩa cứng có thêm một khái niệm là ***hệ số xen kẽ*** (interleave factor) của các sector nhằm làm khớp tốc độ quay của đĩa từ với tốc độ mà đầu từ có thể xử lý dữ liệu khi chúng qua hết một sector. Ví dụ như với đĩa có 17 sector/track, đầu từ sẽ đọc được 512 x 16 x 60 = 522,240 byte/s. Do đó, vùng đệm sẽ đầy lên rất nhanh. Đồng thời, CPU cần phải xử lý dữ liệu nên sẽ mất thêm một khoảng thời gian nữa. Như vậy, nếu dữ liệu được ghi lên các sector liên tiếp thì CPU không thể xử lý kịp Æ sector kế tiếp sẽ đặt cách đó n sector. Khi đó, đĩa cứng sẽ có hệ số xen kẽ n Æ các sector dữ liệu không liên tiếp nhau về mặt vật lý. Chương trình định dạng cấp thấp sẽ đánh số thứ tự các sector liên tiếp theo một trật tự định trước phụ thuộc vào hệ số xen kẽ. Nếu đĩa cứng không được định dạng với hệ số xen kẽ tối ưu thì hiệu suất sử dụng sẽ thấp.

Sector xấu (bad sector): trong quá trình định dạng cấp thấp, các sector xấu không thể lưu trữ dữ liệu sẽ được đánh dấu để không dùng nữa. Quá trình này gọi là *định bản đồ các sector xấu*.

3.2.3. Bộ điều khiển và giao tiếp đĩa cứng

Khác với bộ điều khiển đĩa mềm chỉ dùng các byte CRC, bộ điều khiển đĩa cứng còn dùng thêm các byte ECC (Error Correcting Code) cho phép sửa lỗi và dùng mạch giao tiếp ST412/506.

3.2.3.1. Chuẩn giao tiếp IDE (Integrated Drive Electronics)

Với cách giao tiếp thông thường, ổ đĩa chỉ chứa các linh kiện điện tử tối thiểu đòi hỏi cho việc điều khiển động cơ và các cổng logic còn quá trình điều khiển lệnh (đọc sector, đọc tín hiệu mã hóa, …) được thực hiện trên mạch điều khiển đĩa cứng. Như vậy, các dữ liệu được mã hóa phải đi từ ổ đĩa qua cable truyền, tới bộ điều khiển để giải mã Æ có thể làm sai lạc dữ liệu. Giao tiếp IDE giải quyết vấn đề này bằng cách tích hợp cả ổ đĩa và bộ điều khiển vào cùng một khối. Việc kết nối giữa bus và mạch giao tiếp IDE được thực hiện bằng mạch host-adapter, mạch này cung cấp một số bộ đệm và giải mã dùng cho kết nối ổ đĩa. Giao tiếp IDE cho phép phục vụ nhiều nhất là hai ổ đĩa (master gán địa chỉ 0 và slave gán địa chỉ 1). Một số ổ đĩa IDE trang bị thêm bộ nhớ cache cho ít nhất 2 track nhằm giảm thời gian truy xuất trung bình của đĩa.

CPU truy xuất bộ điều khiển IDE qua một số thanh ghi dữ liệu và điều khiển. Chúng được phân thành hai nhóm với địa chỉ cơ sở 1F0h và 3F0h.

Thanh ghi	Địa chỉ	Độ rộng thanh ghi	R/W
Thanh ghi dữ liệu	1F0h	16	R/W

Thanh ghi lỗi	1F1h	8	R
Bù trước	1F1h	8	W
Số các sector	1F2h	8	R/W
Số sector	1F3h	8	R/W
Cylinder LSB	1F4h	8	R/W
Cylinder MSB	1F5h	8	R/W
Ổ đĩa / đầu từ	1F6h	8	R/W
Thanh ghi trạng thái	1F7h	8	R
Thanh ghi lệnh	1F7h	8	W
Thanh ghi trạng thái biến đối	3F6h	8	R
Thanh ghi ngõ ra số	3F6h	8	W
Địa chỉ ổ đĩa	3F7h	8	R

Chương trình giao tiếp CPU với IDE gồm 3 bước:

- Giai đoạn lệnh: CPU chuẩn bị các thanh ghi và chuyển mã lệnh để khởi động việc thực thi lệnh.
- Giai đoạn dữ liệu: ổ đĩa định vị đầu từ và truyền dữ liệu với bộ nhớ chính.
- Giai đoạn kết quả: bộ điều khiển cung cấp thông tin trạng thái cho lệnh đã chạy trong các thanh ghi tương ứng và phát một ngắt tại IRQ14 (INT 76h).

3.2.3.2. Chuẩn giao tiếp SCSI (Small Computer System Interface)

SCSI là một chuẩn giao tiếp giữa ổ đĩa và PC rất mềm dẻo và mạnh. Chuẩn này xây dựng một bus giao tiếp gồm 8 đơn vị SCSI trong đó một host-adapter nối 7 đơn vị còn lại với bus hệ thống. Khác với IDE, host-adapter này phức tạp hơn vì nó phải thực hiện các chức năng của bus SCSI và không bị ràng buộc bởi các hạn chế của bus hệ thống. Bus SCSI chỉ phục vụ cho quá trình trao đổi dữ liệu giữa các đơn vị được nối với nó. Trong một thời điểm chỉ có hai đơn vị có thể hoạt động. Việc trao đổi dữ liệu có thể thực hiện giữa host-adapter và ổ đĩa hay giữa hai đơn vị SCSI mà không có sự tham gia của CPU.

3.3. Tổ chức logic của đĩa mềm và đĩa cứng

3.3.1. Sector logic

BIOS dùng các sector vật lý để quản lý dữ liệu trong khi hệ điều hành dùng một sơ đồ khác gọi là sector logic. Đó là đánh số các sector liên tục từ 0. Ví dụ như một đĩa mềm 3 ¼ 2 mặt với 80 track và 18 sector/track, dung lượng 1.44 MB như sau:

Sector vật lý Sector logic

Head 0, track 0, sector 1	0
……	
Head 0, track 0, sector 18	17
Head 1, track 0, sector 1	18
……	
Head 1, track 0, sector 18	35
Head 0, track 1, sector 1	36
……	
Head 0, track 1, sector 18	53
Head 1, track 79, sector 1	2862
……	
Head 1, track 79, sector 18	2879

3.3.2. Phân vùng (Partition)

Một đĩa cứng có thể chia thành nhiều ổ đĩa logic và được xem như những ổ đĩa vật lý riêng biệt. Về mặt logic, một phân vùng được xem như một đĩa cứng và có thể cài đặt một hệ điều hành tùy ý lên đó. Có 3 loại phân vùng trên đĩa cứng: DOS chính (Primary DOS), DOS mở rộng (Extended DOS) và phi DOS (non-DOS).

Để lưu trữ thông tin về các phân vùng, DOS lưu trữ trong một vùng cố định: head 0, track 0, sector 1 (sector vật lý đầu tiên của đĩa cứng), sector này được gọi là sector phân vùng (partition sector). Thông tin về từng phân vùng được lưu trữ bởi các điểm vào phân vùng (partition entries) trong bảng phân vùng (partition table).

Cấu trúc của sector phân vùng:

MBR (chương trình kiểm tra bảng phân vùng và gọi boot sector): 446 byte
Bảng phân vùng: 64 byte
Nhận dạng (thường là AA55h): 2 byte

Một ví dụ của sector phân vùng như sau:

- MBR chiếm 664 byte đầu tiên của sector và kết thúc bằng ký hiệu nhận dạng đĩa (in đậm: FD 4E F2 14).
- Phần còn lại là bảng phân vùng và nhận dạng (in nghiêng)

```
Physical Sector:Cyl 0,Side 0,Sector 1

  00000000:00 33 C0 8E D0 BC 00 7C -8B F4 50 07 50 1F FB FC .3.....|..P.P..
  00000010:BF 00 06 B9 00 01 F2 A5 -EA 1D 06 00 00 BE BE 07 ................
  00000020:B3 04 80 3C 80 74 0E 80 -3C 00 75 1C 83 C6 10 FE ...<.t..<.u.....
```

```
00000030:CB 75 EF CD 18 8B 14 8B -4C 02 8B EE 83 C6 10 FE .u......L.......
00000040:CB 74 1A 80 3C 00 74 F4 -BE 8B 06 AC 3C 00 74 0B .t..<.t.....<.t.
00000050:56 BB 07 00 B4 0E CD 10 -5E EB F0 EB FE BF 05 00 V.......^.......
00000060:BB 00 7C B8 01 02 57 CD -13 5F 73 0C 33 C0 CD 13 ..|...W.._s.3...
00000070:4F 75 ED BE A3 06 EB D3 -BE C2 06 BF FE 7D 81 3D Ou..........}.=
00000080:55 AA 75 C7 8B F5 EA 00 -7C 00 00 49 6E 76 61 6C U.u.....|..Inval
00000090:69 64 20 70 61 72 74 69 -74 69 6F 6E 20 74 61 62 id partition tab
000000A0:6C 65 00 45 72 72 6F 72 -20 6C 6F 61 64 69 6E 67 le.Error loading
000000B0:20 6F 70 65 72 61 74 69 -6E 67 20 73 79 73 74 65 operating syste
000000C0:6D 00 4D 69 73 73 69 6E -67 20 6F 70 65 72 61 74 m.Missing operat
000000D0:69 6E 67 20 73 79 73 74 -65 6D 00 00 80 45 14 15 ing system...E..
000000E0:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
000000F0:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000100:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000110:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000120:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000130:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000140:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000150:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000160:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000170:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000180:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
00000190:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
000001A0:00 00 00 00 00 00 00 00 -00 00 00 00 00 00 00 00 ................
000001B0:00 00 00 00 00 00 00 00 -FD 4E F2 14 00 00       .........N......
                                                  80 01                 ..
000001C0:01 00 06 0F 7F 96 3F 00 -00 00 51 42 06 00 00 00 .....?...QB....
000001D0:41 97 07 0F FF 2C 90 42 -06 00 A0 3E 06 00 00 00 A....,.B...>....
000001E0:C1 2D 05 0F FF 92 30 81 -0C 00 A0 91 01 00 00 00 .-....0.........
000001F0:C1 93 01 0F FF A6 D0 12 -0E 00 C0 4E 00 00 55 AA ...........N..U.
```

Cấu trúc của bảng phân vùng:

Bảng phân vùng có 64 byte chia thành 4 điểm vào, mỗi điểm vào có vị trì bắt đầu tại byte thứ 446 (phân vùng 1), 462 (phân vùng 2), 478 (phân vùng 3) và 494 (phân vùng 4).

Byte offset	Kích thước	Ý nghĩa
00	1 byte	Phân vùng có khởi động được (=80h) hay không (=00h)
01	1 byte	Head bắt đầu
02	6 bit	Sector bắt đầu (bit 0-5, bit 6-7 dùng cho cylinder)
03	10 bit	Cylinder bắt đầu
04	1 byte	Loại phân vùng
05	1 byte	Head kết thúc
06	6 bit	Sector kết thúc
07	10 bit	Cylinder kết thúc
08	4 byte	Sector tương đối
12	4 byte	Tổng số sector

Loại phân vùng:

Giá trị	Ý nghĩa
01h	FAT-12 hay ổ đĩa logic với số sector < 32680
04h	FAT-16hay ổ đĩa logic với số sector ≥ 2680 và ≤ 65535
05h	Phân vùng mở rộng
06h	BIGDOS FAT (> 32MB)
07h	NTFS chính
0Bh	FAT32 chính, dùng INT 13 mở rộng
0Ch	FAT32 mở rộng, dùng INT 13 mở rộng
0Eh	FAT16 mở rộng, dùng INT 13 mở rộng
0Fh	FAT16 chính, dùng INT 13 mở rộng

4 loại phân vùng sau cho phép dùng hệ thống file FAT mà Windows NT không thể nhận dạng được.

Sector tương đối: là độ dời của sector bắt đầu của phân vùng với sector bắt đầu của đĩa.

```
                                                  80 01              ..
000001C0:01 00 06 0F 7F 96 3F 00 -00 00 51 42 06 00 00 00 .....?...QB....
000001D0:41 97 07 0F FF 2C 90 42 -06 00 A0 3E 06 00 00 00 A....,.B...>....
000001E0:C1 2D 05 0F FF 92 30 81 -0C 00 A0 91 01 00 00 00 .-....0.........
000001F0:C1 93 01 0F FF A6 D0 12 -0E 00 C0 4E 00 00 55 AA ...........N..U.
```

Xét phân vùng đầu tiên: 80 01 01 00 06 0F 7F 96 3F 00 00 00 51 42 06 00

Byte 0 = 80h: phân vùng khởi động được

Byte 1 = 01h: head 1

Byte 2 = 01h = 0000 0001b: sector 1 (chỉ dùng 6 bit 000001b)

Byte 3 = 00h = 0000 0000b: cylinder 0 (kết hợp với 2 bit cao của byte 2)
Æ bắt đầu tại Head 1, Sector 1, Cylinder 0

Byte 4 = 06: phân vùng BIGDOS

Byte 5 = 0Fh:head 15

Byte 6 = 7Fh = 0111 1111b: sector 63 (dùng 6 bit thấp 111111b)

Byte 7 = 96h = 1001 0110b: cylinder 406 (= 01 1001 0110b)

Æ kết thúc tại Head 15, Sector 63, Cylinder 406

Byte 8-11= 0000003Fh: độ dời so với sector đầu tiên là 63 sector.

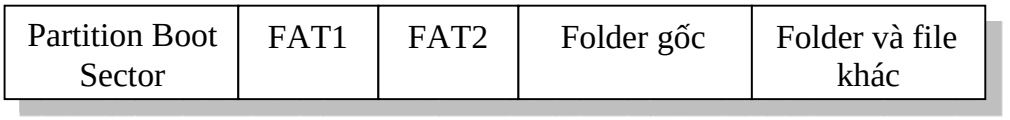
Byte 9-12 = 00064251: tổng cộng có 410.193 sector trong phân vùng

3.3.3. Hệ thống file FAT (File Allocation Table)

Hệ thống file FAT là hệ thống file đơn giản thiết kế cho đĩa dung lượng nhỏ và cấu trúc folder đơn giản. Đối với hệ thống file này, hệ điều hành lưu trữ file lên đĩa cứng theo đơn vị cluster. Mỗi cluster gồm một hay nhiều sector. Để theo dõi những cluster này, hệ điều hành sử dụng một cấu trúc là bảng định vị file (FAT). Để bảo vệ đĩa, một bản sao của FAT sẽ được lưu trữ thêm để tránh trường hợp FAT hỏng, đồng thời FAT và folder gốc sẽ được lưu trữ tại một vị trí cố định để có thể lấy được các file cần thiết khi khởi động hệ thống.

Sự khác nhau giữa các hệ thống FAT:

Hệ thống file	Số byte/cluster	Số cluster tối đa
FAT12	1.5	< 4087
FAT16	2	≤ 65,526 và ≥ 4087
FAT32	4	≤ 268,435,456 và ≥ 65,526



Hình 3.12 – Cấu trúc của FAT

3.3.3.1. Sector khởi động phân vùng PBS

Sector khởi động dài 512 byte đặt tại sector logic 0 của ổ đĩa logic. Nó chứa một chương trình cho phép nạp nhân (kernel) của hệ điều hành để khởi động hệ thống.

Cấu trúc của PBS:

Byte Offset	Kích thước [byte]	Ý nghĩa
00h	3	Lệnh nhảy
03h	8	Tên nhà sản xuất
0Bh	25	Khối thông số BIOS (BPB)
24h	26	Khối thông số BIOS mở rộng
3Eh	448	Mã khởi động (bootstrap code)
1FEh	2	Đánh dấu kết thúc sector

Cấu trúc của BPB (BIOS Parameter Block) và BPB mở rộng (Extended BPB):

Byte Offset	Kích thước [byte]	Ý nghĩa
0Bh	2	Số byte/sector (thường là 512)
0Dh	1	Số sector/cluster
0Eh	2	Số sector dành cho boot sector
10h	1	Số lượng FAT (thường là 2)
11h	2	Số điểm vào gốc (root entry)
13h	2	Số sector/đĩa (nếu < 16 bit). Ngược lại thì = 0
15h	1	Môi trường lưu trữ, F8h để chỉ đĩa cứng
16h	2	Số sector/FAT
18h	2	Số sector/track
1Ah	2	Số head
1Ch	4	Số sector ẩn (giống như sector tương đối)
20h	4	Số sector/đĩa (nếu > 16 bit). Ngược lài thì = 0
24h	1	Số thứ tự của đĩa vật lý (thường là 80h)
25h	1	Head hiện hành, không dùng cho FAT
26h	1	Nhận dạng (=28h hay 29h để WinNT có thể nhận ra)
27h	4	Số serial của đĩa
2Bh	11	Nhãn đĩa
36h	8	Nhận dạng hệ thống file

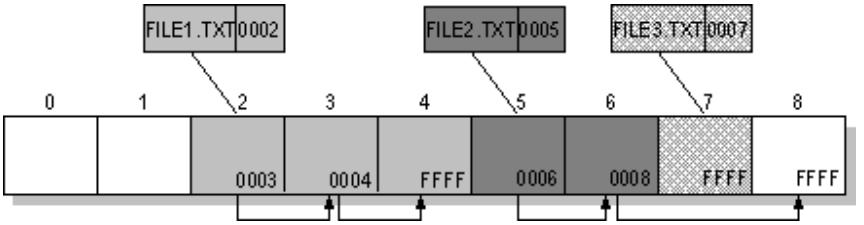
3.3.3.2. Bảng định vị file FAT

FAT là một bảng gồm một số điểm vào ứng với số cluster trên đĩa. Số cluster bằng với tổng số sector trên đĩa chia cho số sector trên một cluster. Kích thước của mỗi điểm vào tùy thuộc vào hệ điều hành, có thể là 12, 16 hay 32 bit (ứng với FAT12, FAT16 và FAT32). Hai điểm vào đầu tiên dùng để nhận dạng loại đĩa (đĩa cứng, đĩa mềm, bao nhiêu mặt, bao nhiêu sector/track, …). Nội dung của một điểm vào:

- 0000h: cluster trống
- FFF7h: cluster xấu
- FFF8h – FFFFh: cluster cuối cùng của file
- Cluster đang sử dụng bởi file

Mỗi cluster chứa một con trỏ chỉ tới cluster kế tiếp trong file hay giá trị FFFFh để xác định cluster kết thúc file.

File1.txt chứa trong 3 cluster liên tục, file2.txt cũng chứa trong 3 cluster nhưng bị phân mảnh còn file3.txt chỉ chứa trong 1 cluster.



Hình 3.13 – Ví dụ của FAT

Một ví dụ của bảng FAT16 như sau (16 byte):

```
F8 FF FF FF 03 00 00 04 00 05 00 FF FF B1 05 01 A9
```

Byte 1,2 (F8 FF): nhận dạng đĩa

Byte 3,4 (FF FF): cluster kết thúc của một file nào đó.
Btye 5,6 (03 00): chứa giá trị 0003h Æ cluster kế tiếp là cluster 3.
Byte 7,8 (04 00): chứa giá trị 0004h Æ cluster kế tiếp là cluster 4.
Byte 9,10 (05 00): chứa giá trị 0005h Æ cluster kế tiếp là cluster 5.
Byte 11,12 (FF FF): cluster kết thúc fileÆ file chứa trong 4cluster.

Các byte tiếp theo: dùng cho các file khác.

3.3.3.3. Folder gốc

Folder gốc chứa các điểm vào cho các file và folder nằm trên gốc của ổ đĩa. Folder gốc khác với các folder khác là nó có vị trí và kích thước cố định (512 điểm vào đối với đĩa cứng). Một điểm vào folder gốc có nội dung như sau:

Byte offset	Kích thước [byte]	Ý nghĩa
00h	8	Tên file
08h	3	Phần mở rộng
0Bh	1	Thuộc tính file
0Ch	10	Dự trữ
16h	2	Giờ thay đổi thông tin cuối cùng
18h	2	Ngày thay đổi thông tin cuối cùng
1Ah	2	Cluster đầu tiên của file
1Ch	4	Kích thước file

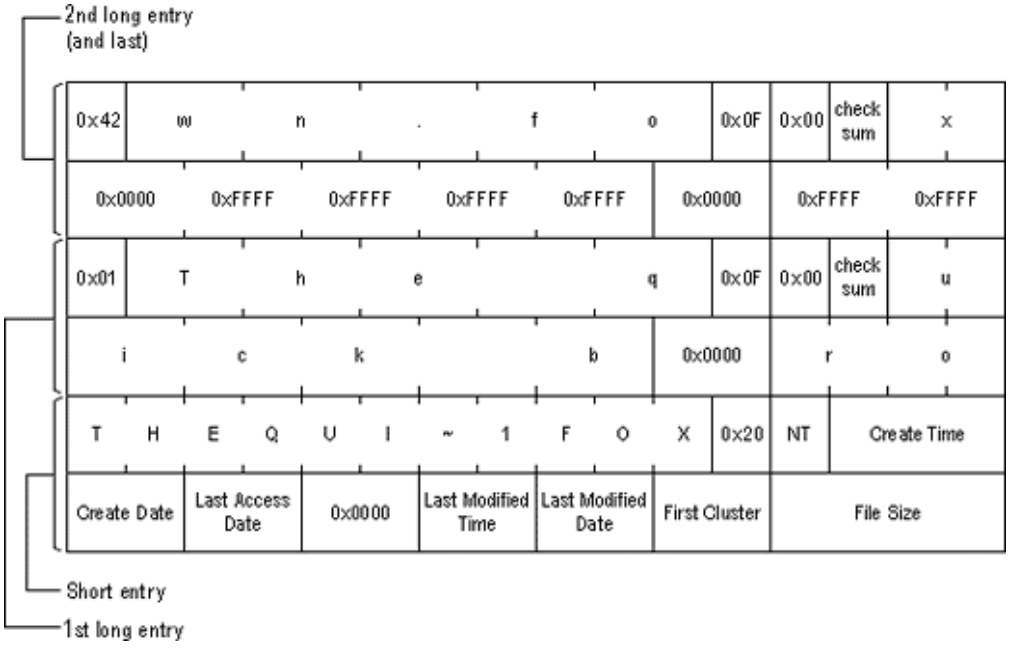
Folder có một tập các điểm vào folder 32 byte (folder entry) cho mỗi file và folder con chứa trong nó. Điểm vào folder bao gồm:

- Tên: 8.3 ký tự

- Byte thuộc tính: 1 byte bao gồm thuộc tính lưu trữ (archive), ẩn (hidden), hệ thống (system) và chỉ đọc (read-only). User có thể bật hay tắt các thuộc tính này.
- Thời gian tạo: 3 byte
- Ngày tạo: 2 byte
- Ngày truy xuất cuối cùng: 2 byte
- Thời gian thay đổi thông tin cuối cùng: 2 byte
- Ngày thay đổi thông tin cuối cùng: 2 byte
- Số thứ tự của cluster bắt đầu file: 2 byte
- Kích thước: 2 byte

Tên file:

Bắt đầu từ WinNT 3.5, file được tạo trên đĩa FAT dùng các bit thuộc tính để hỗ trợ tên file dài mà không ảnh hưởng đến các hệ điều hành trước (DOS). Khi tạo một file, nếu tên file dài thì Windows sẽ tạo một tên dạng 8.3 cho file và sẽ thêm các điểm vào thứ cấp của file, mỗi điểm vào chứa 13 ký tự. DOS sẽ bỏ qua các điểm vào này, xem như chúng không tồn tại và xử lý như dạng file 8.3 chuẩn.



Hình 3.14 – Ví dụ về tên file dài

3.3.4. FAT32

FAT32 là một dạng mở rộng của hệ thống file FAT để hỗ trợ lưu trữ lớn hơn 2 GB. Ổ đĩa FAT32 có thể chứa nhiều hơn 65256 cluster và mỗi cluster nhỏ hơn so với FAT16 nên hiệu suất sử dụng sẽ cao hơn. File lớn nhất có thể lưu trữ trên ổ đĩa FAT32 là 4GB-2.

Thay đổi trong boot sector và bootstrap:

Thay đổi trong boot sector:

Thay đổi	Mô tả
Các sector dành riêng	FAT32 chứa nhiều sector dành riêng hơn FAT16 và FAT12, thường là 32
Thay đổi của boot sector	Do FAT32 BPB lớn hơn BPB chuẩn, MBR trên ổ đĩa FAT32 lớn hơn 1 sector đồng thời 1 sector trong khu vực dành riêng chứa số lượng cluster chưa sử dụng và số lượng cluster mới sử dụng gần nhất. Các giá trị này là thành phần của cấu trúc BIGFATBOOTFSINFO cho phép hệ thống khởi tạo các giá trị mà không cần đọc toàn bộ FAT.
Folder gốc	Folder gốc trên ổ đĩa FAT32 không chứa tại một vị trí cố định như FAT16 hay FAT12. Folder gốc là một chuỗi cluster thông thường. Thành phần *A_BF_BPB_RootDirStrtClus* trong cấu trúc BPB chứa số thứ tự của cluster đầu tiên của folder gốc và *BPB_RootEntries* sẽ bị bỏ qua.
Số sector/FAT	Thành phần *A_BF_BPB_SectorsPerFAT* của BPB luôn bằng 0 trên ổ đĩa FAT32 mà thay vào đó là 2 thành phần *A_BF_BPB_BigSectorsPerFat* và *A_BF_BPB_BigSectorsPerFatHi*.

BPB (BIOS Parameter Block):

BPB trong FAT32 là phiên bản mở rộng của BPB FAT16/FAT12. Nó cũng cung cấp thông tin như trước nhưng đồng thời thêm vào một số trường:

Tên	Kích thước [byte]	Mô tả
A_BF_BPB_BytesPerSector	2	Số byte/sector
A_BF_BPB_SectorsPerCluster	1	Số sector/cluster
A_BF_BPB_ReservedSectors	2	Số thứ tự của sector dành riêng, bắt đầu bằng sector 0
A_BF_BPB_NumberOfFATs	1	Số lượng FAT
A_BF_BPB_RootEntries	2	Bỏ qua
A_BF_BPB_TotalSectors	2	Tổng số sector
A_BF_BPB_MediaDescriptor	1	Tương tự như FAT16
A_BF_BPB_SectorsPerFAT	2	= 0
A_BF_BPB_SectorsPerTrack	2	Số sector/track
A_BF_BPB_Heads	2	Số head trên đĩa
A_BF_BPB_HiddenSectors	2	Số sector ẩn trên đĩa
A_BF_BPB_HiddenSectorsHigh	2	Số sector ẩn trên đĩa (2 byte cao)

A_BF_BPB_BigTotalSectors	2	Tổng số sector trên FAT32
A_BF_BPB_BigTotalSectorsHigh	2	Tổng số sector trên FAT32 (2 byte cao)
A_BF_BPB_BigSectorsPerFat	2	Số sector/FAT
A_BF_BPB_BigSectorsPerFatHi	2	Số sector/FAT (2 byte cao)
A_BF_BPBExtFlags	2	Bit 7 xác định thông tin trên FAT hiện hành có cập nhật có tất cả các FAT khác hay không.
A_BF_BPB_FS_Version	2	Phiên bản của hệ thống file
A_BF_BPB_RootDirStrtClus	2	Số thứ tự cùa cluster đầu tiên của folder gốc
A_BF_BPB_RootDirStrtClusHi	2	Số thứ tự cùa cluster đầu tiên của folder gốc (2 byte cao)
A_BF_BPB_FSInfoSec	2	Số thứ tự của sector chứa thông tin hệ thống file.
A_BF_BPB_BkUpBootSec	2	Số thứ tự của bản sao boot sector.
A_BF_BPB_Reserved	12	Dành riêng

Cấu trúc BIGFATBOOTFSINFO:

Cấu trúc này chứa các thông tin về hệ thống file trên ổ đĩa FAT32.

Tên	Kích thước [byte]	Mô tả
bfFSInf_Sig	4	Nhận dạng của sector thông tin hệ thống file. Giá trị này là FSINFOSIG (0x61417272L).
bfFSInf_free_clus_cnt	4	Số cluster không sử dụng (=-1 nếu không xác định)
bfFSInf_next_free_clus	4	Số thứ tự cluster định vị gần nhất
bfFSInf_resvd	12	Dành riêng

Phản chiếu FAT (mirroring):

Trên FAT16/FAT12, FAT đầu tiên luôn là bản sơ cấp và bất kỳ thay đổi nào đầu được tự động cập nhật trên các bản sao. Đối với FAT32, phản chiếu FAT có thể bỏ qua và một bản sao khác với bản đầu tiên có thể là bản sơ cấp. Phản chiếu được cho phép bằng cách xóa bit 0080h trong thành phần *extddpb_flags* của cấu trúc DPB.

Phản chiếu	Mô tả
Cho phép (xóa bit 0080h)	Bất kỳ khi nào một sector FAT thay đổi thì nó sẽ cập nhật cho các FAT khác.
Không	Chỉ một FAT tích cực, thường dùng khi FAT có bad sector.

DPB (Drive Parameter Block):

Cấu trúc của DPB:

```
DPB STRUC
    dpb_drive            DB    ?
    dpb_unit               DB  ?
    dpb_sector_size       DW   ?
    dpb_cluster_mask      DB   ?
    dpb_cluster_shift     DB   ?
    dpb_first_fat           DW  ?
    dpb_fat_count           DB  ?
    dpb_root_entries      DW   ?
    dpb_first_sector      DW   ?
    dpb_max_cluster       DW   ?
    dpb_fat_size            DW  ?
    dpb_dir_sector          DW  ?
    dpb_reserved2         DD   ?
    dpb_media            DB    ?
ifdef NOTFAT32
    dpb_first_access     DB    ?
else
    dpb_reserved         DB    ?
endif
    dpb_reserved3        DD    ?
    dpb_next_free        DW    ?
    dpb_free_cnt         DW    ?
ifndef NOTFAT32
    extdpb_free_cnt_hi   DW    ?
    extdpb_flags         DW    ?
    extdpb_FSInfoSec     DW    ?
    extdpb_BkUpBootSec   DW    ?
    extdpb_first_sector  DD    ?
    extdpb_max_cluster   DD    ?
    extdpb_fat_size      DD    ?
    extdpb_root_clus     DD    ?
    extdpb_next_free     DD    ?
endif
DPB ENDS
```

Tên	Mô tả
dpb_drive	Số thứ tự đĩa (0 = A, 1 = B, …)
dpb_unit	Số thứ tự của đơn vị đĩa (unit number), cho phép trình điều khiển thiết bị phân biệt đĩa.
dpb_sector_size	Số byte/sector
dpb_cluster_mask	Số sector/cluster – 1
dpb_cluster_shift	Số sector/cluster biểu diễn theo số mũ của 2
dpb_first_fat	Số thứ tự của sector đầu tiên chứa FAT
dpb_fat_count	Số FAT/đĩa
dpb_root_entries	Số điểm vào/folder gốc
dpb_first_sector	Số thứ tự sector đầu tiên của cluster đầu tiên
dpb_max_cluster	Số cluster/đĩa -1 (không dùng trong FAT32)

dpb_fat_size	Số sector đang sử dụng /FAT (= 0 đối với FAT32, thay thế bằng *extdpb_fat_size*)
dpb_dir_sector	Số thứ tự của sector đầu tiên chứa folder gốc (không dùng trong FAT32)
dpb_reserved2	Dành riêng
dpb_media	Môi trường trữ tin
reserved	Dành riêng
dpb_first_access	Xác định đĩa có dùng được hay không
dpb_reserved3	Dành riêng
dpb_next_free	Số thứ tự cluster định vị gần nhất
dpb_free_cnt	Số lượng cluster trống (=FFFFh nếu không xác định)
extdpb_free_cnt_hi	2 byte cao xác định số cluster trống
extdpb_flags	Mô tả đĩa
extdpb_FSInfoSec	Số thứ tự của sector chứa thông tin hệ thống file
extdpb_BkUpBootSec	Số thứ tự của bản sao boot sector
extdpb_first_sector	Sector đầu tiên của cluster đầu tiên
extdpb_max_cluster	Số sector/đĩa + 1
extdpb_fat_size	Số sector sử dụng bởi FAT
extdpb_root_clus	Số thứ tự của cluster đầu tiên trong folder gốc
extdpb_next_free	Số thứ tự cluster định vị gần nhất

Loại phân vùng:

Loại phân vùng hợp lệ cho trong bảng sau (giá trị tương ứng chứa trong thành phần *Part_FileSystem* của cấu trúc S_PARTITION.

Giá trị	Mô tả
PART_UNKNOWN (00h)	Unknown
PART_DOS2_FAT (01h)	FAT12
PART_DOS3_FAT (04h)	FAT16 (< 32 MB)
PART_EXTENDED (05h)	Phân vùng DOS mơ rộng
PART_DOS4_FAT (06h)	FAT16 (> 32 MB)
PART_DOS32 (0Bh)	FAT32 (có thể tới2047GB)
PART_DOS32X (0Ch)	FAT32 dùng Int 13h mở rộng
PART_DOSX13 (0Eh)	Giống PART_DOS4_FAT (06h) dùng Int 13h mở rộng
PART_DOSX13X (0Fh)	Giống PART_EXTENDED (05h) dùng Int 13h mở rộng

Cấu trúc S_PARTITION:

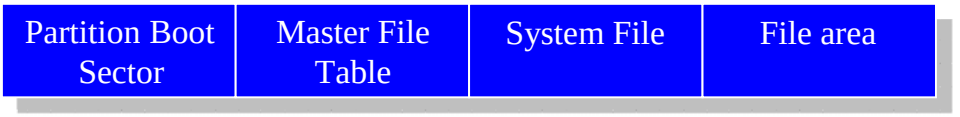
```
s_partition    STRUC
    Part_BootInd        DB    ?
    Part_FirstHead      DB    ?
    Part_FirstSector    DB    ?
    Part_FirstTrack     DB    ?
    Part_FileSystem     DB    ?
    Part_LastHead       DB    ?
    Part_LastSector     DB    ?
    Part_LastTrack      DB    ?
    Part_StartSector    DD    ?
    Part_NumSectors     DD    ?
s_partition    ENDS
```

Tên	Mô tả
Part_BootInd	Phân vùng có khởi động được (=80h) hay không (=00h)
Part_FirstHead	Head đầu tiên của phân vùng
Part_FirstSector	Sector đầu tiên của phân vùng (bit 0-5; 6-7 dùng cho Part_FirstTrack)
Part_FirstTrack	Track đầu tiên của phân vùng
PartFileSystem	Hệ thống file của phân vùng
Part_LastHead	The last head of the partition. This is a 0-based number that represents the offset from the beginning of the disk. The partition includes the head specified by this member.
Part_LastSector	Sector đầu tiên của phân vùng (bit 0-5; 6-7 dùng cho Part_LastTrack)
Part_LastTrack	Track đầu tiên của phân vùng
Part_StartSector	Số thứ tự của sector đầu tiên trên đĩa
Part_NumSectors	Số sector/phân vùng

3.3.5. Hệ thống file NTFS (New Technology File System)

NTFS là hệ thống file có hiệu suất cao và có thể tự sửa lỗi sử dụng cho Windows XP, 2000, NT. Hệ thống này hỗ trợ vấn đề bảo mật ở cấp độ file, nén file và kiểm định. Nó cũng hỗ trợ đĩa dung lượng lớn và giải pháp lưu trữ cao như RAID. NTFS cung cấp khả năng kết hợp giữa sự thực thi, độ tin cậy và tính tương thích không có trong hệ thống FAT đồng thời làm giảm thời gian thực thi các hoạt động trên file như đọc, ghi, tìm kiếm và cả các hoạt động nâng cao như khôi phục hệ thống file ở các đĩa cứng dung lượng rất lớn.

NTFS bao gồm các đặc trưng bảo mật cần thiết cho các máy dịch vụ file (file server) và các máy tính cá nhân đầu trên (high-end) trong môi trường làm việc theo nhóm. NFTS cũng hỗ trợ điều khiển xử lý dữ liệu và cấp quyền quản lý dữ liệu nhằm bảo đảm tính chính xác của dữ liệu. NTFS có thể cho phép cấp quyền xử lý cho file hay folder theo từng user riêng lẻ.



Hình 3.15 – Cấu trúc của NTFS

PBS (Partition Boot Sector) bắt đầu tại sector 0, dài 16 sector. File đầu tiên trên đĩa NTFS là MFT (Master File Table). MFT chứa các thông tin về tất cả các file và folder trên đĩa.

Các đặc trưng mới trong NTFS5 (Windows 2000):

- Mã hóa: hệ thống file mã hóa (EFS – Encrypting File System) cung cấp kỹ thuật mã hóa lõi file để lưu trữ tên đĩa NTFS.
- Chỉ tiêu đĩa (disk quota): quản lý và giới hạn dung lượng đĩa có thể sử dụng.
- Điểm phân tích lại (Reparse point): dùng cho nhiều đặc trưng lưu trữ của Windows 2000.
- Điểm cài đặt đĩa (Volume mount point): dựa trên các điểm phân tích lại để cho phép người quản trị xử lý kết hợp thư mục gốc (root) của một đĩa như là cấu trúc folder của một đĩa khác.
- File phân mảnh (Sparse file): cho phép tạo file rất lớn (dùng tất cả dung lượng đĩa).
- Hiệu chỉnh liên kết đã sắp xếp (Distributed link tracking): cung cấp dịch vụ hiệu chỉnh liến kết để bảo đảm tính toàn vẹn của shortcut.

3.3.5.1. PBS – Partition Boot Sector

Boot sector của đĩa dùng định dạng NTFS mô tả như sau:

Byte Offset	Kích thước [byte]	Mô tả
00h	3	Lệnh nhảy
03h	8	Tên nhà sản xuất
0Bh	25	BPB
24h	48	Extended BPB
54h	426	Bootstrap code
01FEh	2	Kết thúc sector

Trong đĩa NTFS, dữ liệu trong trường BPB và trường BPB mở rộng cho phép chương trình nạp (Ntldr – NT loader program) tìm kiếm MFT trong suốt quá trình khởi động. Trên đĩa NTFS, không giống như trong FAT, MFT không được chỉ rõ trên một

sector xác định trước. Do đó, MFT có thể di chuyển sang vị trí khác nếu như tồn tại sector lỗi tại vị trí mặc định. Tuy nhiên, khi dữ liệu bị sai, Windows NT/2000 sẽ xem như là đĩa chưa định dạng.

Trường BPB và BPB mở rộng trên đĩa NTFS mô tả như sau:

Byte Offset	Kích thước [byte]	Mô tả
0Bh	2	Số byte/sector
0Dh	1	Số sector/cluster
0Eh	2	Sector dự trữ
10h	3	Luôn bằng 0
13h	2	Không dùng
15h	1	Media Descriptor
16h	2	Luôn bằng 0
18h	2	Số sector/track
1Ah	2	Số lượng head
1Ch	4	Các sector ẩn
20h	4	Không dùng
24h	4	Không dùng
28h	8	Tổng số sector
30h	8	Số cluster vật lý cho file $MFT
38h	8	Số cluster vật lý cho file $MFTMirr
40h	4	Số Cluster/File Record Segment
44h	4	Số Cluster/Index Block
48h	8	Số serial của đĩa
50h	4	Checksum

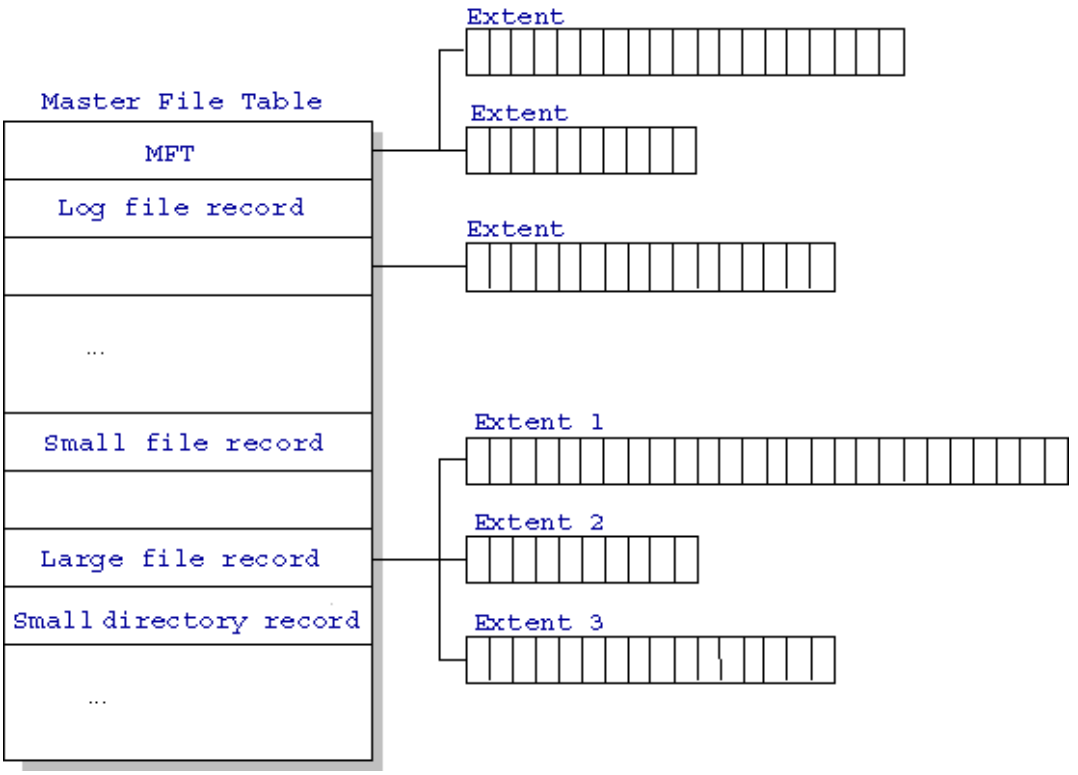
Một ví dụ của đĩa NTFS chạy Windows 2000 như sau:

- Byte 00h ÷ 0Ah: chứa lệnh nhảy và OEM ID (Original Equipment Manufacturer Identification) (in đậm)
- Byte 0Bh ÷ 53h: BPB và BPB mở rộng.
- Phần còn lại: mã tự khởi động và đánh dấu kết thúc sector (in đậm).

```
Physical Sector:Cyl 0, Side 1, Sector 1

00000000:EB 52 90 4E 54 46 53 20 -20 20 20 00 02 08 00 00 .R.NTFS ........
00000010:00 00 00 00 00 F8 00 00 -3F 00 FF 00 3F 00 00 00 ........?...?...
00000020:00 00 00 00 80 00 80 00 -4A F5 7F 00 00 00 00 00 ........J.......
00000030:04 00 00 00 00 00 00 00 -54 FF 07 00 00 00 00 00 ........T.......
00000040:F6 00 00 00 01 00 00 00 -14 A5 1B 74 C9 1B 74 1C ...........t..t.
00000050:00 00 00 00 FA 33 C0 8E -D0 BC 00 7C FB B8 C0 07 .....3.....|....
00000060:8E D8 E8 16 00 B8 00 0D -8E C0 33 DB C6 06 0E 00 ..........3.....
00000070:10 E8 53 00 68 00 0D 68 -6A 02 CB 8A 16 24 00 B4 ..S.h..hj....$..
00000080:08 CD 13 73 05 B9 FF FF -8A F1 66 0F B6 C6 40 66 ...s......f...@f
00000090:0F B6 D1 80 E2 3F F7 E2 -86 CD C0 ED 06 41 66 0F .....?.......Af.
```

```
000000A0:B7 C9 66 F7 E1 66 A3 20 -00 C3 B4 41 BB AA 55 8A ..f..f....A..U.
000000B0:16 24 00 CD 13 72 0F 81 -FB 55 AA 75 09 F6 C1 01 .$...r...U.u....
000000C0:74 04 FE 06 14 00 C3 66 -60 1E 06 66 A1 10 00 66 t......f`..f...f
000000D0:03 06 1C 00 66 3B 06 20 -00 0F 82 3A 00 1E 66 6A ....f;....:..fj
000000E0:00 66 50 06 53 66 68 10 -00 01 00 80 3E 14 00 00 .fP.Sfh.....>...
000000F0:0F 85 0C 00 E8 B3 FF 80 -3E 14 00 00 0F 84 61 00 ........>.....a.
00000100:B4 42 8A 16 24 00 16 1F -8B F4 CD 13 66 58 5B 07 .B..$.......fX [.
00000110:66 58 66 58 1F EB 2D 66 -33 D2 66 0F B7 0E 18 00 fXfX.-f3.f......
00000120:66 F7 F1 FE C2 8A CA 66 -8B D0 66 C1 EA 10 F7 36 f......f..f....6
00000130:1A 00 86 D6 8A 16 24 00 -8A E8 C0 E4 06 0A CC B8 ......$.........
00000140:01 02 CD 13 0F 82 19 00 -8C C0 05 20 00 8E C0 66 ..............f
00000150:FF 06 10 00 FF 0E 0E 00 -0F 85 6F FF 07 1F 66 61 ..........o...fa
00000160:C3 A0 F8 01 E8 09 00 A0 -FB 01 E8 03 00 FB EB FE ................
00000170:B4 01 8B F0 AC 3C 00 74 -09 B4 0E BB 07 00 CD 10 .....<.t........
00000180:EB F2 C3 0D 0A 41 20 64 -69 73 6B 20 72 65 61 64 .....A disk read
00000190:20 65 72 72 6F 72 20 6F -63 63 75 72 72 65 64 00 error occurred.
000001A0:0D 0A 4E 54 4C 44 52 20 -69 73 20 6D 69 73 73 69 ..NTLDR is missi
000001B0:6E 67 00 0D 0A 4E 54 4C -44 52 20 69 73 20 63 6F ng...NTLDR is co
000001C0:6D 70 72 65 73 73 65 64 -00 0D 0A 50 72 65 73 73 mpressed...Press
000001D0:20 43 74 72 6C 2B 41 6C -74 2B 44 65 6C 20 74 6F Ctrl+Alt+Del to
000001E0:20 72 65 73 74 61 72 74 -0D 0A 00 00 00 00 00 00 restart........
000001F0:00 00 00 00 00 00 00 00 -83 A0 B3 C9 00 00 55 AA ..............U.
```

3.3.5.2. Bảng file chính (MFT – Master File Table)



Hình 3.16 - Cấu trúc của MFT

Mỗi file trên đĩa NTFS đại diện bằng một bản ghi (record) trong một file đặc biệt gọi là MFT. NTFS dành riêng 16 sector đầu tiên cho các thông tin đặc biệt. Bản ghi đầu tiên mô tả chính MFT, sau đó là bản ghi MFT ảnh (MFT mirror record). Nếu bản ghi đầu

tiên bị sai thì NTFS sẽ đọc bản ghi thứ hai để tìm file MFT ảnh trong đó bản ghi đầu tiên của MFT ảnh giống hệt như trong MFT. Vị trí của đoạn dữ liệu của cả hai file MFT và MFT ảnh được ghi lại trong boot sector. Một bản sao của boot sector được lưu trữ tại vị trí giữa (vật lý) của đĩa.

Bản ghi thứ ba là log file dùng cho mục đích khôi phục file. Bản ghi thứ 17 và các bản ghi ở phía sau dùng cho mỗi file và thư mục trong đĩa. MFT cấp một không gian nào đó cho mỗi bản ghi. File và thư mục nhỏ (1500 byte hay nhỏ hơn) có thể chứa hoàn toàn bên trong MFT.

Thiết kế này làm cho tốc độ xử lý file nhanh hơn. Đối với hệ thống file FAT (dùng FAT để liệt kê tên và địa chỉ của mỗi file), mỗi điểm vào (entry) FAT chứa một chỉ số trong bảng. Như vậy, để tìm một file trong hệ thống FAT, đầu tiên phải đọc bảng định vị file để đảm bảo file tồn tại. Sau đó, FAT lấy file bằng cách tìm kiếm chuỗi các vị trí được gán cho file. Đối với NTFS, file sẽ được lấy ra ngay lập tức.

Hình 3.17 - Bản ghi MFT cho file nhỏ và thư mục

Bản ghi thư mục được cấp chỗ trong MFT giống như bản ghi file (thư mục không chứa dữ liệu mà chứa các thông tin chỉ số). Bản ghi thư mục nhỏ có thể chứa hoàn toàn bên trong cấu trúc MFT trong đó các thư mục lớn hơn sẽ được chứa dưới dạng cây nhị phân (B-tree) và dùng con trỏ để xác định các cluster chứa các mục không chứa trong cấu trúc MFT.

3.3.5.3. Phân loại file NTFS

Thuộc tính file NTFS:

NTFS quan niệm mỗi file (hay folder) như tập hợp các thuộc tính file. Các phần tử của tập hợp này là tên file, các thông tin bảo mật file và cả dữ liệu trong file. Mỗi thuộc tính được xác định bằng mã và tên thuộc tính. Khi các thuộc tính chứa trong bản ghi file MFT, chúng được gọi là các thuộc tính nội trú (resident attribute). Ví dụ như tên file và đặc tính thời gian luôn chứa trong bản ghi file MFT. Nếu các thông tin của file quá lớn so với bản ghi MFT, một số thuộc tính của chúng phải là ngoại trú (nonresident). Các thuộc tính ngoại trú được định vị trên một hay nhiều cluster bất kỳ nào đó trên đĩa. NTFS tạo thuộc tính Attribute List (danh sách thuộc tính) để mô tả vị trí của tất cả các bản ghi thuộc tính.

Các thuộc tính định nghĩa bằng hệ thống file NTFS:

Loại thuộc tính	Mô tả
Thông tin chuẩn (Standard Information)	Đặc trưng thời gian và số lượng liên kết
Danh sách thuộc tính (Attribute List)	Danh sách vị trí các bản ghi thuộc tính không chứa trong MFT
Tên file (File Name)	Tên file dài có thể đến 255 ký tự Unicode, tên file ngắn có dạng 8.3, không phân biệt chữa hoa và thường
Mô tả bảo mật (Security Descriptor)	Xác định tác giả và các user được phép xử lý file
Dữ liệu (Data)	Chứa dữ liệu file. NTFS cho phép nhiều thuộc tính dữ liệu trên mỗi file. Về cơ bản, mỗi file có một thuộc tính dữ liệu không tên và có thể có thêm các thuộc tính được đặt tên, mỗi thuộc tính có cú pháp riêng
Nhận dạng đối tượng (Object ID)	Duy nhất cho mỗi file, dùng cho dịch vụ hiệu chỉnh liên kết và không phải file nào cũng có thuộc tính này
Logged Tool Stream	Tương tự luồng dữ liệu nhưng dùng cho hoạt động lưu trữ log file
Điểm phân tích lại (Reparse Point)	Các điểm cài đặt đĩa, dùng cho IFS (Installable File System)
Chỉ số gốc (Index root)	Xử lý folder và các chỉ số khác
Chỉ số định vị (Index allocation)	Xử lý folder và các chỉ số khác
Bitmap	Xử lý folder và các chỉ số khác
Thông tin đĩa (Volume information)	Chứa phiên bản đĩa
Tên đĩa (Volume name)	Nhãn đĩa

File hệ thống NTFS:

File hệ thống là file lưu trữ dữ liệu (metadata) của hệ thống file và bổ sung thêm cho hệ thống file. Các file hệ thống được đặt trên đĩa bằng ứng dụng Format.

File hệ thống	Tên file	Bản ghi MFT	Mô tả
MFT	$Mft	0	Chứa một bản ghi file cơ sở cho mỗi file và folder trên đĩa NTFS. Nếu thông tin lớn hơn một bản ghi đơn thì sẽ dùng thêm các bản ghi khác.
MFT2	$MftMirr	1	Ảnh của 4 bản ghi đầu tiên của MFT.
Log file	$LogFile	2	Chứa danh sách các bước dùng cho khả năng khôi phục NTFS. Kích thước log file phụ thuộc vào kích thước đĩa

			(có thể tới 4 MB).
Đĩa	$Volume	3	Chứa thông tin đĩa như nhãn đĩa và phiên bản.
Định nghĩa thuộc tính	$AttrDef	4	Tên thuộc tính, số thứ tự và mô tả.
Chỉ số tên file gốc	$	5	Folder gốc
Cluster bitmap	$Bitmap	6	Mô tả cluster nào sử dụng
Boot sector	$Boot	7	Bao gồm BPB và mã khởi động (bootstrap loader).
File cluster xấu	$BadClus	8	Bad cluster của đĩa
Bảo mật file	$Secure	9	Chứa các mô tả bảo mật của file
Bảng chữ hoa	$Upcase	10	Chuyển chữ thường thành chữ hoa để thích hợp với ký tự Unicode.
File mở rộng NTFS	$Extend	11	Dùng cho các mục đích mở rộng như chỉ tiêu đĩa, điểm phân tích lại và nhận dạng đối tượng
		12–15	Dành riêng

Đa luồng dữ liệu NTFS:

NTFS hỗ trợ đa luồng dữ liệu trong đó tên luồng đồng nhất với thuộc tính dữ liệu mới của file. Mỗi luồng dữ liệu, tập hợp của thuộc tính file, đại diện bằng một handle. Đặc trưng này cho phép quản lý dữ liệu như các đơn vị riêng lẻ. Khi sao chép một file trên đĩa NTFS sang đĩa FAT, các luồng dữ liệu và các thuộc tính khác không hỗ trợ bởi FAT sẽ bị mất.

File nén NTFS:

WinNT/2000 hỗ trợ nén file, folder riêng lẻ và toàn bộ đĩa NTFS. File nén trên đĩa NTFS có thể đọc / ghi bằng các ứng dụng trên nền Windows mà không cần giải nén bằng các chương trình khác. Khi đọc file, file sẽ được tự động giải nén và sau đó nén lại một lần nữa khi đóng hay lưu file. Thuật toán nén trên NTFS hỗ trợ kích thước cluster lên tới 4 KB. Nếu kích thước cluster lớn hơn 4 KB thì không sử dụng chức năng nén. Mỗi luồng dữ liệu NTFS chứa các thông tin xác định có phần nào của luồng được nén hay không.

Hệ thống file mã hóa (EFS - Encrypting File System):

- ***Files and Folders (chỉ trong NTFS5):***

EFS cung cấp kỹ thuật mã hóa file để lưu trữ file trên đĩa NTFS. EFS cung cấp khả năng lưu trữ an toàn, chống lại sự xâm nhập từ bên ngoài. User làm việc với file và folder đã mã hóa giống như với các file thông thường. Hệ thống sẽ tự động giải mã khi xử lý file

hay folder và sau đó sẽ mã hóa lại. Nếu user không được cấp quyền thì sẽ khnôg thể xử lý được file. EFS có các lợi ích sau cho các ứng dụng mã hóa của hãng thứ 3:

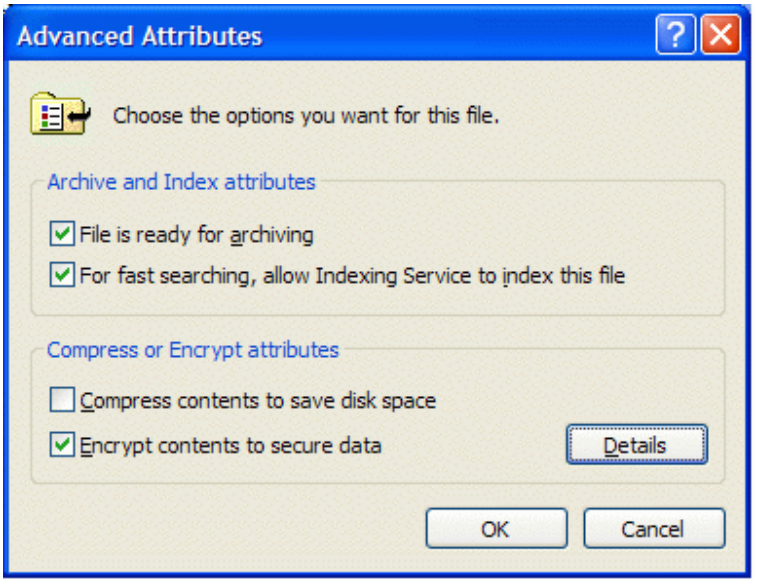
+ Trong suốt đối với user và ứng dụng: user không cần nhớ mật mã để giải mã file.

+ Độ bảo mật từ khóa cao.

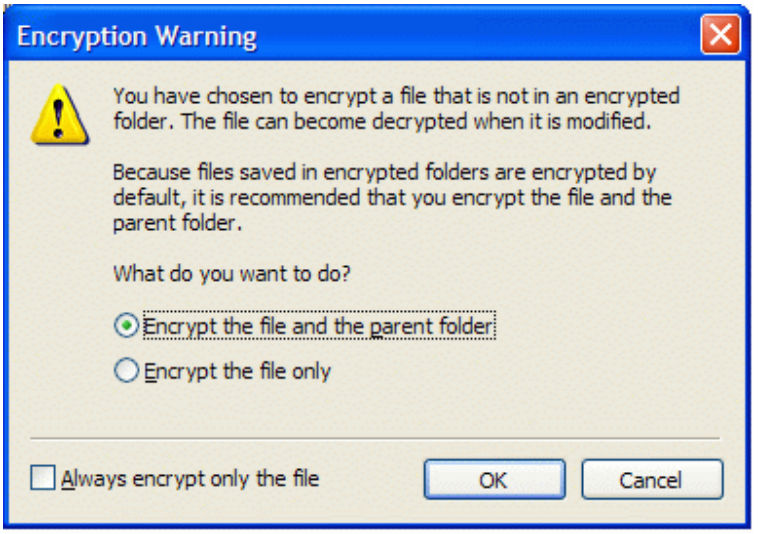
+ Các quá trình mã hóa / giải mã thực hiện ở chế độ nhân (kernel), tránh rủi ro mất từ khóa.

+ Cung cấp cơ chế khôi phục dữ liệu rất có giá trị trong môi trường thương mại, cho phép khôi phục dữ liệu ngay cả khi nhân viên mã hóa đã rời công ty.

User có thể hiển thị các đặc trưng EFS bằng cách dùng lệnh **cipher.exe** hay dùng Windows Explorer (right-click trên file, chọn thẻ *General*, click nút *Advanced*).



Hình 3.18 – Cửa sổ thuộc tính Advanced

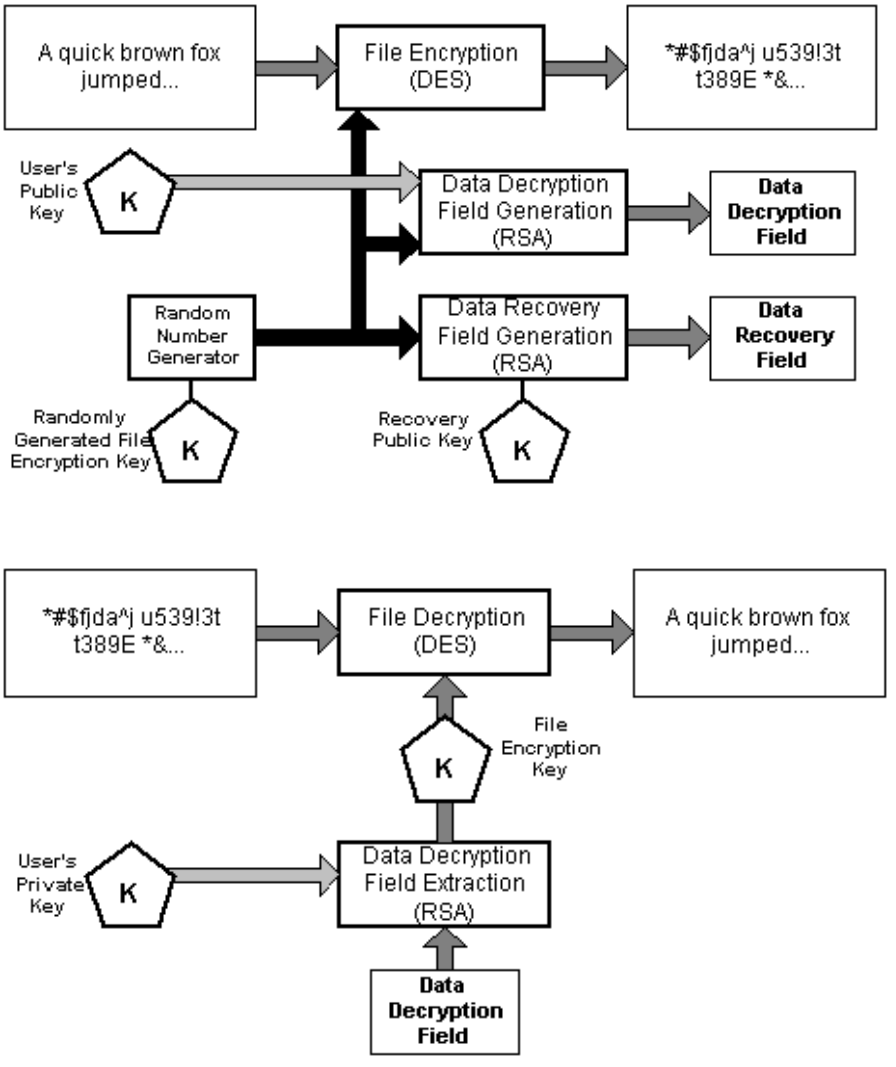


Hình 3.19 – Cửa sổ Encryption Warning

Sau đó, Windows yêu cầu user xác định chỉ mã hóa một file hay toàn folder. Khi xử lý file, Windows sẽ tạo ra một bản sao ẩn không mã hóa và làm việc trên bản sao này. Sau khi xử lý xong, Windows sẽ cập nhật file và xóa bản sao. Bản sao này chính là một lỗ hổng bảo mật do nó được lưu trữ ở dạng không mã hóa. Quá trình mã hóa toàn folder sẽ giải quyết được vấn đề này.

- ***EFS:***

EFS kết hợp kỹ thuật khóa công cộng và mã hóa khóa đối xứng để bảo vệ file. Dữ liệu trên file được mã hóa dùng thuật toán đối xứng. Khóa mã gọi là khóa mã hóa file (FEK – File Encryption Key). FEK được mã hóa dùng thuật toán công cộng RSA (1024 bit) và lưu trữ trên file. Sau khi file đã mã hóa, chỉ có các user có DDF và DRF phù hợp mới có thể xử lý file.



Hình 3.20 – Sơ đồ mã hóa và giải mã

3.3.6. So sánh NTFS và FAT

Tiêu chuẩn	NTFS5	NTFS	FAT32	FAT16
Hệ điều hành	Win2000, XP	WinNT, 2000, XP	Win98, ME, 2000, XP	DOS, Windows
Giới hạn				
Kích thước đĩa tối đa	2TB	2TB	2TB	2GB
Số file tối đa	Không giới hạn	Không giới hạn	Không giới hạn	~65000
Kích thước file tối đa	Theo kích thước đĩa	Theo kích thước đĩa	4GB	2GB
Số cluster tối đa	Không giới hạn	Không giới hạn	268,435,456	65,535
Chiều dài tên file tối đa	255	255	255	Chuẩn: 8.3 Mở rộng: 255
Đặc trưng của hệ thống file				
Tên file Unicode	Ký tự Unicode	Ký tự Unicode	Ký tự hệ thống	Ký tự hệ thống
Ảnh bản ghi hệ thống	File MFT Mirror	File MFT Mirror	Bản sao của FAT	Bản sao của FAT
Vị trí boot sector	Sector đầu tiên và cuốicùng	Sector đầu tiên và cuốicùng	Sector đầu tiên	Sector đầu tiên
Thuộc tính file	Chuẩn và tùy ý	Chuẩn và tùy ý	Chuẩn	Chuẩn
Luồng liên tục	Có	Có	Không	Không
Nén	Có	Có	Không	Không
Mã hóa	Có	Không	Không	Không
Cấp quyền	Có	Có	Không	Không
Chỉ tiêu đĩa	Có	Không	Không	Không
File rời rạc	Có	Không	Không	Không
Điểm phân tích lại	Có	Không	Không	Không
Hiệu suất toàn đĩa				
Tự bảo mật	Có	Có	Không	Không
Khả năng khôi phục	Có	Có	Không	Không
Hiệu suất	Thấp trên đĩa nhỏ Cao trên đĩa lớn	Thấp trên đĩa nhỏ Cao trên đĩa lớn	Thấp trên đĩa lớn Cao trên đĩa nhỏ	Thấp trên đĩa lớn Cao trên đĩa nhỏ
Tiết kiệm không gian đĩa	Lớn	Lớn	Trung bình	Nhỏ trên đĩa lớn

3.4. Truy xuất ổ đĩa qua DOS và BIOS

3.4.1. Đĩa mềm

3.4.1.1. DOS

DOS cung cấp 3 ngắt cho quá trình truy xuất đĩa mềm và cứng là 25h, 26h, 21h.

Ngắt 25h: đọc sector

Thanh ghi	Giá trị gọi	Trả về
AL	Số thứ tự ổ đĩa	
CX	Số sector	
DX	Sector đầu	Mã lỗi
BX	Offset của vùng đệm đọc	
DS	Đoạn của vùng đệm đọc	
CF		

Ngắt 26h: ghi sector

Thanh ghi	Giá trị gọi	Trả về
AL	Số thứ tự ổ đĩa	
CX	Số sector	
DX	Sector đầu	Mã lỗi
BX	Offset của vùng đệm ghi	
DS	Đoạn của vùng đệm ghi	
CF		Lỗi nếu > 0

Mã lỗi trả về như sau:

Mã	Lỗi
01h	Lệnh không hợp lệ
02h	Che địa chỉ không chính xác
04h	Không tìm thấy sector
08h	Tràn DMA
10h	Lỗi CRC hay ECC
20h	Lỗi bộ điều khiển
40h	Lỗi tìm kiếm
80h	Ổ đĩa không sẵn sàng

3.4.1.2. BIOS

Việc truy xuất ổ đĩa dùng ngắt 13h:

Hàm	Chức năng
00h	Khởi động ổ đĩa
01h	Đọc trạng thái và tác vụ cuối cùng
02h	Đọc sector Ghi
03h	sector Kiểm tra
04h	sector
05h	Định dạng track

3.4.2. Đĩa cứng

Giống như đĩa mềm, các sector logic có thể truy xuất bằng ngắt 25h, 26h của DOS. Các sector vật lý truy xuất bằng ngắt 13h của BIOS. Các hàm dùng cho đĩa cứng như sau:

Hàm	Chức năng
05h	Định dạng track và cylinder
06h	Định dạng và đánh dấu track xấu
07h	Định dạng và đánh dấu ổ đĩa
08h	Xác định các thông số ổ đĩa
09h	Cài đặt thông số ổ đĩa
0Ah	Đọc sector mở rộng
0Bh	Ghi sector mở rộng
0Ch	Tìm kiếm
0Dh	Khởi tạo đĩa cứng
0Eh	Đọc bộ đệm sector
0Fh	Ghi bộ đệm sector
10h	Kiểm tra ổ đĩa xem đã sẵn sàng chưa
11h	Chuẩn lại ổ đĩa
19h	Nâng đầu từ đọc / ghi

Mã lỗi điều khiển:

Mã	Lỗi
00h	Không lỗi
02h	Không có tín hiệu tìm kiếm
03h	Lỗi ghi
04h	Ổ đĩa không sẵn sàng
06h	Không tìm tấy track 0
10h	Lỗi ECC trong trường ID
11h	Lỗi ECC trong trường dữ liệu
12h	Không có che địa chỉ ID
13h	Không có che địa chỉ dữ liệu
14h	Không có trường ID
15h	Lỗi tìm kiếm
16h	Lỗi bộ điều khiển bên trong
17h	Lỗi DMA
18h	Lỗi dữ liệu có thể sửa được

3.5. Đĩa quang

Ngày này, đĩa quang đã được sử dụng phổ biến, chúng có mật độ ghi thông tin cao hơn đĩa từ thông thường. Các đĩa quang dựa trên cùng một cộng nghệ được sử dụng trong Compact Disc để ghi âm nên được gọi tên là CD ROM.

- ***Nguyên lý chế tạo:***

Các đĩa CD ROM được tạo ra bằng cách dùng một tia laser mạnh đốt chảy các hốc đường kính 1 µm trên một đĩa chủ, từ đĩa chủ này sẽ tạo ra các khuôn để tạo các bản copy trên đĩa chất dẻo. Sau đó, phủ một lớp nhôm mỏng lên trên mặt đĩa và lại phủ một lớp chất dẻo trong suốt lên lớp nhôm để bảo vệ. Lớp nhôm có tác dụng phản xạ tia laser. Các hốc nhỏ được gọi là pit, diện tích không bị dốt nóng gọi là land. Chúng có độ phản xạ khác nhau nên có thể phân biệt được pit và land.

- ***Tổ chức dữ liệu:***

Thông tin trên CD ROM được ghi theo một đường xoắn ốc duy nhất và ghi thành từng nhóm 24 byte, mỗi byte được mở rộng thành 14 bit bằng cách dùng mã sửa sai Reed – Solomon. Ba bit được thêm vào giữa các nhóm và một byte đồng bộ được bổ sung để tạo thành 1 frame. 98 frame tạo thành một block chứa 2 KB dữ liệu. CD ROM có thể chứa 270,000 block tương ứng với dung lượng 553 MB.

Các đĩa CD ROM được đọc bằng một đầu dò đo năng lượng phản xạ từ bề mặt đĩa khi chiếu lên đó một tia laser công suất nhỏ. Dữ liệu được đọc với tốc độ 75 inches/s, cho tốc độ đọc dữ liệu là 153.6 KBps.

- ***WORM (Write Once Read Many):***

Các CD ROM mô tả như trên không thể ghi được, do đó thế hệ đĩa quang thứ hai ra đời, đó là WORM. Thiết bị này cho phép người sử dụng tự mình ghi thông tin nhưng sau khi đã tạo ra các pit thì không thể thay đổi được nữa. Cơ chế ghi của WORM có 2 loại tùy vào cấu trúc bề mặt:

+ Lớp phủ đĩa ở vùng bị đốt nóng sẽ bay hơi và làm lộ ra bề mặt của vùng đĩa dưới không còn lớp phủ. Hai vùng này có hệ số phản xạ khác nhau nên lưu trữ bit khác nhau.

+ Ở cuối quá trình ghi bằng xung laser, lớp phủ đĩa bị chảy ra sẽ được làm lạnh nhanh và kết rắn lại ở dạng vô định hình. Lớp này sẽ có hệ số phản xạ khác với lớp phủ cũ.

- ***Ổ đĩa quang từ (Magneto Optical):***

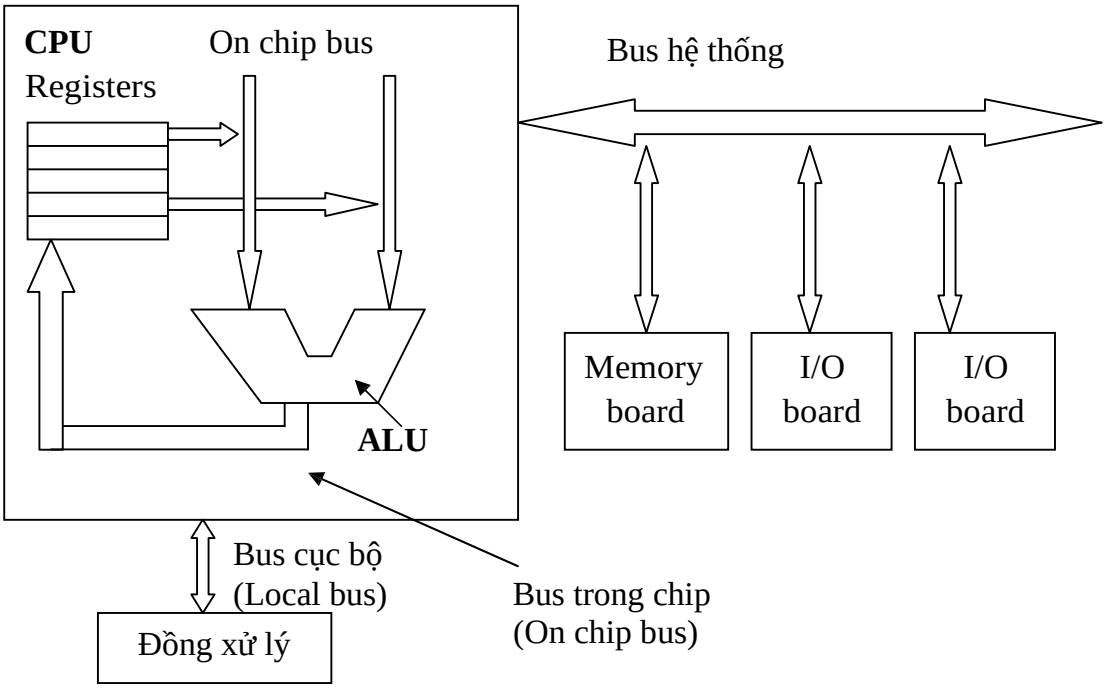
Các đĩa quang thế hệ thứ 3 là những môi trường quang học có thể xóa được. Đĩa có một lớp phủ đồng nhất làm bằng vật liệu hợp kim latan sắt từ. Khi tia laser phân cực đốt bề mặt đĩa, hướng phân cực của tia phản xạ được quay phụ thuộc vào mức nhiễm từ của bề mặt.

Khi cần ghi một bit lên đĩa, một xung laser ngắn và mạnh sẽ đốt bề mặt đĩa ở vùng cần thiết làm mức nhiễm từ của vùng này bằng 0. Cùng lúc đó, một nam châm phát ra từ trường có hướng phụ thuộc vào bit cần ghi là 0 hay 1.Hướng của từ trường sẽ xác định hướng của các domain từ trong vùng bị đốt nóng khi được làm lạnh xuống.

Khi đọc một bit, tia laser sẽ quét bề mặt đĩa và hệ phân cực sẽ nhạy với hướng phân cực của tia phản xạ. Nếu chiếu tia laser trên vùng đã ghi dữ liệu, hướng của mặt phẳng phân cực tia phản xạ phụ thuộc vào hướng của các domain từ. Từ đó, ánh sáng sẽ có thể tới bộ lọc phân cực (ứng với mức 1) hay không (ứng với mức 0). Để xóa dữ liệu, ta phải làm nóng các điểm xác định với hướng phù hợp của từ trường.

Chương 4
BUS

Bus là đường truyền tín hiệu điện nối các thiết bị khác nhau trong một hệ thống máy tính. Bus thường có từ 50 đến 100 dây dẫn được gắn trên mainboard, trên các dây này có các đầu nối đưa ra, các đầu này được sắp xếp và cách nhau những khoảng quy định để có thể cắm vào đó những I/O board hay board bộ nhớ (bus hệ thống – system bus).



Hình 4.1 - Các bus trong một hệ thống máy tính

Cũng có những bus dùng cho mục đích chuyên biệt, thí dụ nối 1 vi xử lý với 1 hay nhiều vi xử lý khác hoặc nối với bộ nhớ cục bộ (local bus).

Trong vi xử lý cũng có một số bus để nối các thành phần bên trong của bộ vi xử lý với nhau. Người thiết kế chip vi xử lý có thể tuỳ ý lựa chọn loại bus bên trong nó, còn với các bus liên hệ bên ngoài cần phải xác định rõ các quy tắc làm việc cũng như các đặc điểm kỹ thuật về điện và cơ khí của bus để người thiết kế mainboard có thể ghép nối chip vi xử lý với các thiết bị khác. Nói cách khác, các bus này phải tuân theo 1 chuẩn nào đó. Tập các quy tắc của chuẩn còn được gọi là giao thức bus (bus protocol)

Ngoài ra, có rất nhiều loại bus khác nhau được sử dụng, các bus này nói chung là không tương thích với nhau. Một số bus được sử dụng phổ biến:

Tên bus	Lĩnh vực áp dụng
Camac	Vật lý hạt nhân.
EISA	Một số hệ thống có chip 80386
IBM PC, PC/AT	Máy IBM PC, IBM/PC/AT
Massbus	Máy PDP-11 và VAX
Microchannel	Máy PS/2
Multibus I	Một số hệ thống có 8086
Multibus II	Một số hệ thống có chip 80386
Versabus	Một số hệ thống có chip vi xử lý của Motorola
VME	Một số hệ thống có chip vi xử lý họ 68x0 của Motorola

Bus thường phân loại theo 3 cách sau:

- Theo tổ chức phần cứng (như trên).
- Theo giao thức truyền thông (bus đồng bộ và không đồng bộ).
- Theo loại tín hiệu truyền trên bus (bus địa chỉ, bus dữ liệu,…).

1. Bus hệ thống

Thường có nhiều thiết bị nối với bus, một số thiết bị là tích cực (active) có thể đòi hỏi truyền thông trên bus, trong khi đó có các thiết bị thụ động chờ yêu cầu từ các thiết bị khác. Các thiết bị tích cực được gọi là chủ (master) còn thiết bị thụ động là tớ (slave).

Ví dụ: Khi CPU ra lệnh cho bộ điều khiển đĩa đọc/ghi một khối dữ liệu thì CPU là master còn bộ điều khiển đĩa là slave. Tuy nhiên, bộ điều khiển đĩa ra lệnh cho bộ nhớ nhận dữ liệu thì nó lại giữ vai trò master.

1.1. Bus Driver và Bus Receiver

Tín hiệu điện trong máy tính phát ra thường không đủ để điều khiển bus, nhất là khi bus khá dài và có nhiều thiết bị nối với nó. Chính vì thế mà hầu hết các bus master được nối với bus thông qua 1 chip gọi là ***bus driver***, về cơ bản nó là một bộ khuếch đại tín hiệu số. Tương tự như vậy, hầu hết các slave được nối với bus thông qua ***bus receiver***. Đối với các thiết bị khi thì đóng vai trò master, khi thì đóng vai trò slave, người ta sử dụng 1 chip kết hợp gọi là ***transceiver***. Các chip này đóng vai trò ghép nối và là các thiết bị 3 trạng thái, cho phép nó có thể ở trạng thái thứ 3 – hở mạch (thả nổi).

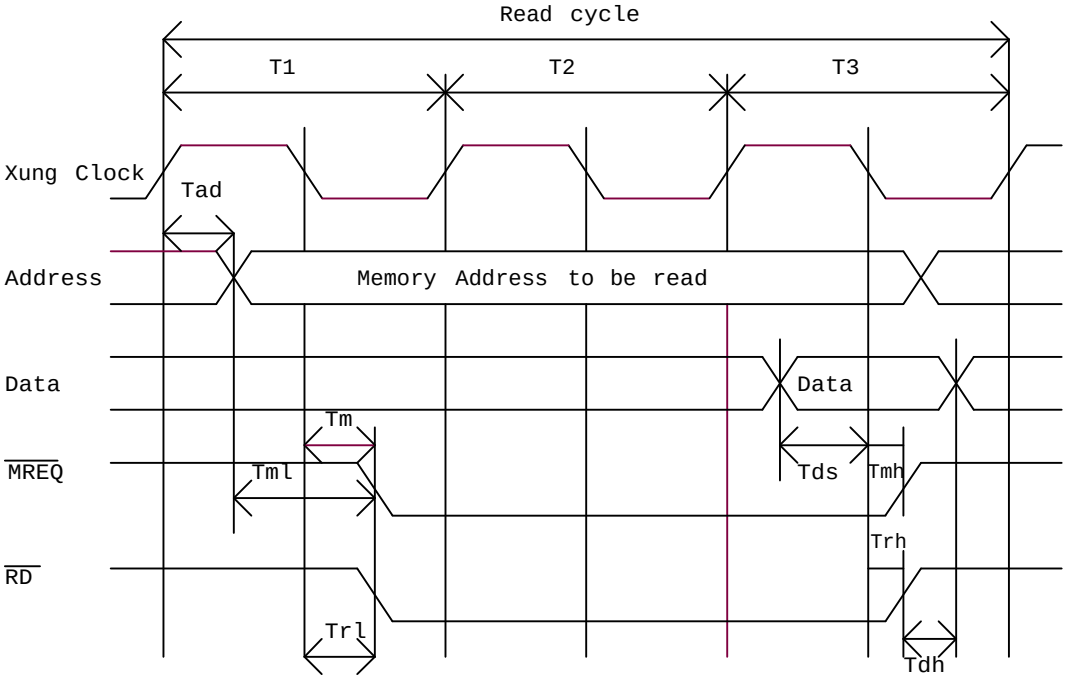
Giống như vi xử lý, bus có các đường địa chỉ, đường số liệu và đường điều khiển. Tuy nhiên, không nhất thiết có ánh xạ 1 – 1 giữa các tín hiệu ở các chân ra của vi xử lý và các đường dây của bus. Thí dụ: một số chíp vi xử lý có 3 chân ra, truyền ra các tín hiệu báo chíp vi xử lý đang thực hiện các thao tác $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, $\overline{\text{IOR}}$, $\overline{\text{IOW}}$ hay thao tác khác. Một bus điển hình thường có 4 đường trên.

Các vấn đề quan trọng nhất liên quan đến thiết kế bus là: xung clock bus (sự phân chia thời gian, hay còn gọi là **bus blocking**), cơ chế phân xử bus (bus arbitration), xử lý ngắt và xử lý lỗi.

Các bus có thể được chia theo giao thức truyền thông thành hai loại riêng biệt là bus đồng bộ và bus không đồng bộ phụ thuộc vào việc sử dụng clock bus.

1.2. Bus đồng bộ (Synchronous bus)

Bus đồng bộ có một đường điều khiển bởi một bộ dao động thạch anh, tín hiệu trên đường dây này có dạng sóng vuông, với tần số thường nằm trong khoảng 5MHz ÷ 50MHz. Mọi hoạt động bus xảy ra trong một số nguyên lần chu kỳ này và được gọi là chu kỳ bus.



Hình 4.2 - Chu kỳ đọc trong bus đồng bộ

Hình trên là giản đồ thời gian của một bus đồng bộ với tần số xung clock là 4MHz, như vậy chu kỳ bus là 250ns.

Giả sử đọc 1 byte từ bộ nhớ chiếm 3 chu kỳ bus (750ns), tương ứng với T1, T2, T3 như hình vẽ. Vì tất cả các tín hiệu điện thay đổi mức không phải là tức thời, nên trên hình vẽ có các sườn xung, ta giả sử các sườn xung kéo dài 10ns.

- T1 bắt đầu bằng cạnh dương của xung clock, trong một phần thời gian của T1, vi xử lý đặt địa chỉ byte cần đọc lên bus địa chỉ. Sau khi tín hiệu địa chỉ được xác lập, vi xử lý đặt các tín hiệu $\overline{\text{MREQ}}$ và $\overline{\text{RD}}$ tích cực (mức thấp). Tín hiệu $\overline{\text{MREQ}}$ (Memory Request) - xác định truy xuất bộ nhớ chứ không phải thiết bị I/O, còn tín hiệu $\overline{\text{RD}}$ - chọn đọc chứ không phải ghi dữ liệu.
- T2: thời gian cần thiết để bộ nhớ giải mã địa chỉ và đưa dữ liệu lên bus dữ liệu.

- T3: tại cạnh âm của T3, vi xử lý nhận dữ liệu trên bus dữ liệu, chứa vào thanh ghi bên trong vi xử lý và chốt dữ liệu. Sau đó vi xử lý đảo các tín hiệu $\overline{\text{MREQ}}$ và $\overline{\text{RD}}$.

Như vậy thao tác đọc đã hoàn thành, tại chu kỳ máy tiếp theo vi xử lý có thể thực hiện thao tác khác. Các giá trị cụ thể về thời gian của hình vẽ trên có thể được giải thích chi tiết như sau:

- T_{AD}: $T_{AD} \leq$ 110ns, nghĩa là nhà sản xuất vi xử lý đảm bảo rằng trong mọi chu kỳ đọc toán hạng từ bộ nhớ, vi xử lý sẽ đưa ra tín hiệu địa chỉ không nhiều hơn 110 ns tính từ thời điểm cạnh dương của T1.
- T_{DS}: giá trị nhỏ nhất là 50ns, có nghĩa là nhà sản xuất bộ nhớ phải đảm bảo rằng dữ liệu đã ổn định trên bus dữ liệu ít nhất là 50ns trước điểm giữa cạnh âm của T3. Yêu cầu này đảm bảo cho vi xử lý đọc dữ liệu tin cậy.

Khoảng thời gian bắt buộc đối với T_{AD} và T_{DS} xác định rằng trong trường hợp xấu nhất, bộ nhớ chỉ có 250 + 250 + 125 – 110 – 50 = 465 ns tính từ thời điểm có tín hiệu địa chỉ cho tới khi tạo ra dữ liệu trên bus dữ liệu. Nếu bộ nhớ không có khả năng đáp ứng đủ nhanh, nó phát tín hiệu $\overline{\text{WAIT}}$ trước cạnh âm của T2. Thao tác này đưa thêm các trạng thái chờ – wait state (tức là đưa thêm vào 1 chu kỳ bus), khi bộ nhớ đã đưa ra tín hiệu ổn định, nó sẽ đảo $\overline{\text{WAIT}}$ thành WAIT.

- T_{ML}: đảm bảo tín hiệu địa chỉ sẽ được xác lập trước tín hiệu $\overline{\text{MREQ}}$ ít nhất 60ns. Khoảng thời gian này sẽ quan trọng nếu tín hiệu $\overline{\text{MREQ}}$ điều khiển quá trình tạo tín hiệu chọn chip ($\overline{\text{CS}}$ hay $\overline{\text{CE}}$) do một số chip nhớ đòi hỏi phải nhận được tín hiệu địa chỉ trước tín hiệu chọn chip. Như vậy, không thể chọn chip nhớ với thời gian thiết lập 75ns.
- T_M, T_{RL}: cho phép 2 tín hiệu $\overline{\text{MREQ}}$ và $\overline{\text{RD}}$ tích cực trong khoảng thời gian 85ns tính từ thời điểm xuống của xung clock T1. Trong trường hợp xấu nhất, chíp nhớ chỉ có 250 + 250 – 85 – 50 = 365ns sau khi 2 tín hiệu trên tích cực để đưa dữ liệu ra bus dữ liệu. Sự bắt buộc về thời gian này bổ sung thêm sự bắt buộc thời gian với tín hiệu clock.
- T_{MH}, T_{RH} : thời gian để các tín hiệu $\overline{\text{MREQ}}$ và $\overline{\text{RD}}$ được đảo sau khi dữ liệu đã được vi xử lý nhận vào.
- T_{DH}: Thời gian bộ nhớ cần giữ data trên bus sau khi tín hiệu $\overline{\text{RD}}$ đã đảo

Giản đồ thời gian một chu kỳ đọc trên bus đồng bộ đã được đơn giản hoá so với thực tế, trong đó các tín hiệu cần sử dụng lớn hơn nhiều. Giá trị tới hạn của các thông số cho trong bảng sau:

Ký hiệu	Tham số	Min (ns)	Max(ns)
T_{AD}	Thời gian trễ của địa chỉ		110
T_{ML}	Thời gian địa chỉ ổn định trước $\overline{MREQ}$	60	
T_M	Thời gian trễ của $\overline{MREQ}$ so với cạnh âm của T1		85
T_{RL}	Thời gian trễ của $\overline{RD}$ so với sườn xuống của tín hiệu đồng hồ T1		85
T_{DS}	Thời gian thiết lập dữ liệu trước sườn xuống của tín hiệu xung clock(tín hiệu đồng hồ)	50	
T_{MH}	Thời gian trễ của $\overline{MREQ}$ so với sườn xuống của tín hiệu đồng hồ T3		85
T_{RH}	Thời gian trễ của $\overline{RD}$ so với sườn xuống của tín hiệu đồng hồ T3		85
T_{DH}	Thời gian lưu trữ dữ liệu từ lúc đảo tín hiệu $\overline{RD}$	0	

Truyền theo khối:

Ngoài các chu kỳ đọc/ghi, một số bus truyền dữ liệu đồng bộ còn hỗ trợ truyền dữ liệu theo khối. Khi bắt đầu thao tác đọc khối, bus master báo cho slave biết số byte cần được truyền đi, thí dụ truyền con số này đi trong chu kỳ T1, sau đó đáng lẽ truyền đi 1 byte, slave đưa ra trong mỗi chu kỳ 1 byte cho tới khi đủ số byte được thông báo. Như vậy, khi đọc dữ liệu theo khối, n byte dữ liệu cần n+2 chu kỳ clock chứ không phải 3n chu kỳ.

Một cách khác để cho truyền dữ liệu nhanh hơn là giảm chu kỳ. Ở ví dụ trên: 1 byte được truyền đi trong 750ns, vậy bus có tốc độ truyền 1.33MBps. Nếu xung clock có tần số 8MHz, thời gian 1 chu kỳ chỉ còn một nửa, tốc độ sẽ là 2.67MBps. Tuy nhiên, giảm chu kỳ bus dẫn đến khó khăn về mặt kỹ thuật, các tín hiệu truyền trên các đường khác nhau không phải luôn có cùng tốc độ, dẫn đến hiệu ứng *bus skew*. Điều quan trọng là thời gian chu kỳ phải dài hơn so với skew để tránh việc những khoảng thời gian được số hoá lại trở thành các đại lượng biến thiên liên tục.

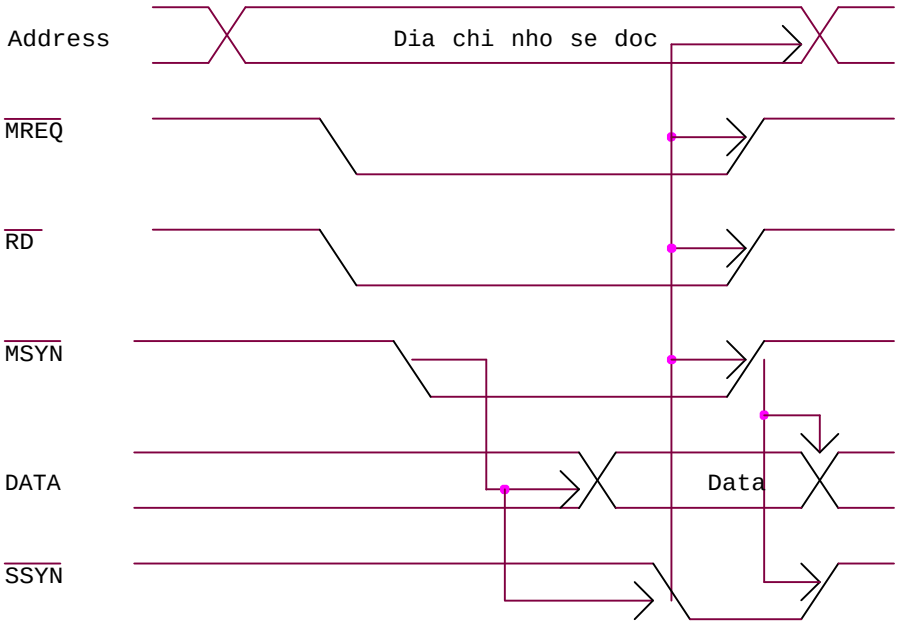
1.3. Bus bất đồng bộ(Asynchronous bus)

Bus bất đồng bộ không sử dụng xung clock đồng bộ, chu kỳ của nó có thể kéo dài tuỳ ý và có thể khác nhau đối với các cặp thiết bị khác nhau. Làm việc với các bus đồng bộ dễ dàng hơn do nó được định thời một cách gián đoạn , tuy vậy chính đặc điểm này cũng dẫn đên nhược điểm. Mọi công việc được tiến hành trong khoảng thời gian là bội số của xung clock, nếu 1 thao tác nào đó của vi xử lý hay bộ nhớ hoàn thành trong 3.1 chu kỳ thì nó cũng sẽ phải kéo dài trong 4 chu kỳ. Khi đã chọn chu kỳ bus và đã xây dựng bộ nhớ, I/O card cho bus này thì khó có thể tận dụng những tiến bộ của công nghệ. Chẳng hạn sau khi đã xây bus với sự định thời như trên, công nghệ mới đưa ra các vi xử lý và bộ nhớ có thời gian chu kỳ là 100ns chứ không còn là 750ns như cũ, thì chúng vẫn chạy với tốc độ thấp như các vi xử lý, bộ nhớ loại cũ, bởi vì giao thức bus đòi hỏi bộ nhớ phải đưa được dữ liệu ra và ổn định trước thời điểm cạnh âm của T3. Nếu có nhiều thiết bị khác

nhau cùng nối với 1 bus, trong đó có thể có một số thiết bị hoạt động nhanh hơn hơn các thiết bị khác thì cần phải đặt bus hoạt động phù hợp với thiết bị có tốc độ thấp nhất.

Bus bất đồng bộ ra đời nhằm khắc phục những nhược điểm của bus đồng bộ. Trước hết master phát ra địa chỉ nhớ mà nó muốn truy cập, sau đó phát tín hiệu $\overline{\text{MREQ}}$ tích cực để xác định cần truy xuất bộ nhớ. Tín hiệu này cần thiết khi bộ nhớ và các cổng I/O sử dụng chung miền địa chỉ. Sau khi phát địa chỉ, bên master cũng phải phát tín hiệu $\overline{\text{RD}}$ tích cực để bên slave biết rằng master sẽ thực hiện thao tác đọc chứ không phải ghi.

Các tín hiệu $\overline{\text{MREQ}}$ và $\overline{\text{RD}}$ được đưa ra sau tín hiệu địa chỉ một khoảng thời gian phụ thuộc tốc độ hoạt động của master. Sau khi 2 tín hiệu này đã ổn định, master sẽ phát ra tín hiệu $\overline{\text{MSYN}}$ (master synchrization) ở mức tích cực để báo cho slave biết rằng các tín hiệu cần thiết đã sẵn sàng trên bus, slave có thể nhận lấy. Khi slave nhận được tín hiệu này, nó sẽ thực hiện công việc với tốc độ nhanh nhất có thể được, đưa dữ liệu của ô nhớ được yêu cầu lên bus dữ liệu. Khi hoàn thành slave sẽ phát tín hiệu $\overline{\text{SSYN}}$ (slave synchronization) tích cực.



Hình 4.3 - Chu kỳ đọc của bus bất đồng bộ

Master nhận được tín hiệu $\overline{\text{SSYN}}$ tích cực thì xác định được dữ liệu của slave đã sẵn sàng nên thực hiện việc chốt dữ liệu, sau đó đảo các đường địa chỉ cũng như các tín hiệu $\overline{\text{MREQ}}$, $\overline{\text{RD}}$ và $\overline{\text{MSYN}}$. Khi slave nhận được tín hiệu $\overline{\text{MSYN}}$ không tích cực, nó xác định kết thúc chu kỳ và đảo tín hiệu $\overline{\text{SSYN}}$ làm bus trở lại trạng thái ban đầu, mọi tín hiệu đều không tích cực, chờ bus master mới.

Trên giản đồ thời gian của bus bất đồng bộ, ta sử dụng mũi tên để thể hiện nguyên nhân và kết quả. $\overline{\text{MSYN}}$ tích cực dẫn đến việc truyền dữ liệu ra bus dữ liệu và đồng thời

cũng dẫn đến việc slave phát ra tín hiệu $\overline{SSYN}$ tích cực, đến lượt mình tín hiệu $\overline{SSYN}$ lại gây ra sự đảo mức của các đường địa chỉ, $\overline{MREQ}$, $\overline{RD}$ và $\overline{MSYN}$. Cuối cùng sự đảo mức của $\overline{MSYN}$ lại gây ra sự đảo mức tín hiệu $\overline{SSYN}$ và kết thúc chu kỳ.

Tập các tín hiệu phối hợp với nhau như vậy được gọi là bắt tay toàn phần (full handshake), chủ yếu gồm 4 tín hiệu sau:

- $\overline{MSYN}$ tích cực.
- $\overline{SSYN}$ tích cực để đáp lại tín hiệu $\overline{MSYN}$.
- $\overline{MSYN}$ được đảo để đáp lại tín hiệu $\overline{SSYN}$ (tích cực).
- $\overline{SSYN}$ được đảo để đáp lại tín hiệu $\overline{MSYN}$ không tích cực.

Ta có thể nhận thấy bắt tay toàn phần là độc lập thời gian, mỗi sự kiện được gây ra bởi 1 sự kiện trước đó chứ không phải bởi xung clock. Nếu 1 cặp master-slave nào đó hoạt động chậm thì cặp master-slave kế tiếp không hề bị ảnh hưởng.

Tuy ưu điểm của bus bất đồng bộ rất rõ ràng, nhưng trong thực tế phần lớn các bus đang sử dụng là loại đồng bộ. Nguyên nhân là các hệ thống sử dụng bus đồng bộ dễ thiết kế hơn. Vi xử lý chỉ cần chuyển các mức tín hiệu cần thiết sang trạng thái tích cực là bộ nhớ đáp ứng ngay, không cần tín hiệu phản hồi. Chỉ cần các chọn phù hợp thì mọi hoạt động đều trôi chảy, không cần phải bắt tay.

1.4. Phân xử bus (bus arbitration)

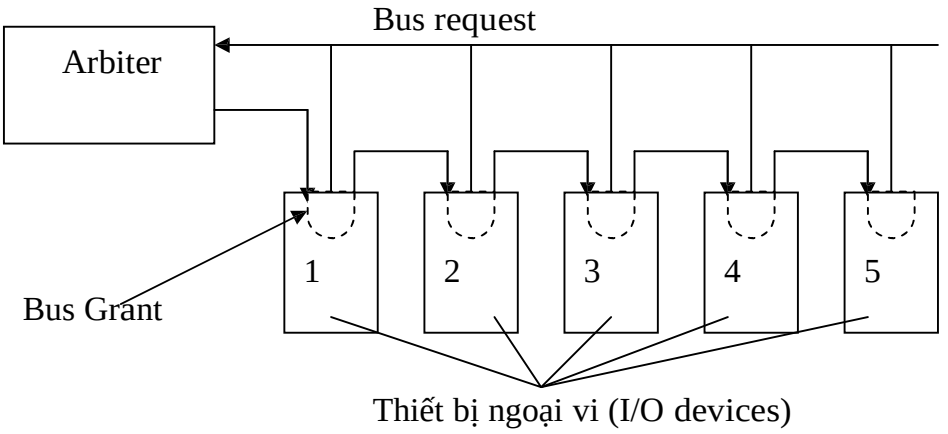
Trong hệ thống máy tính không phải chỉ có CPU làm bus master, các chip I/O cũng có lúc làm bus master để có thể đọc hay ghi bộ nhớ và gọi ngắt. Các bộ đồng xử lý cũng có thể làm bus master. Như vậy nảy sinh ra vấn đề: điều gì sẽ xảy ra khi 2 thiết bị trở lên đồng thời cần làm bus master? Từ đó cần có một cơ chế phân xử để tránh sự hỗn loạn của hệ thống. Cơ chế phân xử có thể là tập trung hay không tập trung.

1.4.1. Phân xử bus tập trung

Nhiều vi xử lý có đơn vị phân xử được chế tạo nằm ngay trong chip CPU, trong một số máy tính mini, đơn vị này nằm ngoài chíp CPU. Theo cơ chế này thì bộ phân xử (arbiter) chỉ có thể biết có yêu cầu chiếm dụng bus hay không mà không biết có bao nhiêu đơn vị muốn chiếm dụng bus. Khi arbiter nhận được yêu cầu, nó sẽ phát ra 1 tín hiệu cho phép trên đường dây (bus grant: cho phép sử dụng bus). Đường dây này nối qua tất cả các thiết bị I/O theo kiểu nối tiếp.

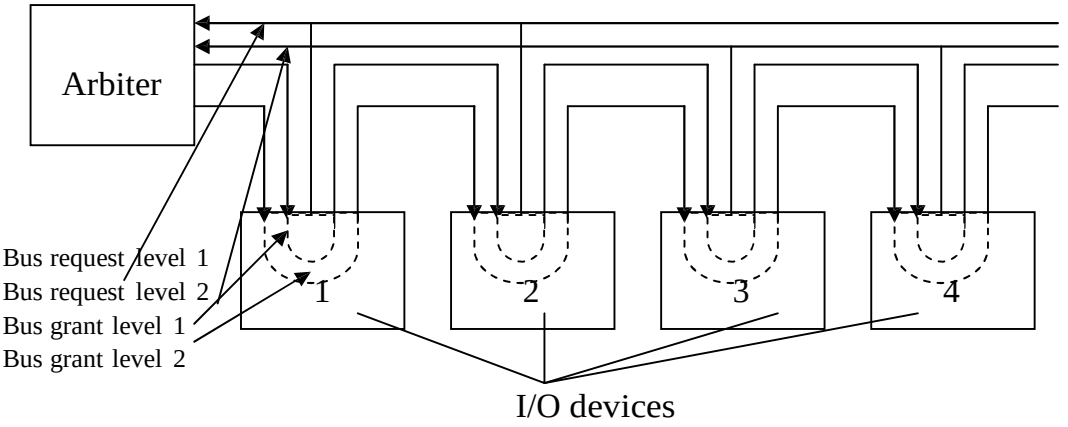
Khi thiết bị nằm gần arbiter nhất nhận được tín hiệu cho phép, nó kiểm tra xem có phải chính nó đã phát ra yêu cầu hay không. Nếu có thì nó sẽ chiếm lấy bus và không truyền tiếp tín hiệu cho phép trên đường dây. Nếu không thì nó sẽ truyền tín hiệu cho phép tới thiết bị kế tiếp trên đường dây, với thiết bị này sự việc xảy ra giống thiết bị trước nó, quá trình cứ tiếp diễn cho đến khi có một thiết bị chiếm lấy bus.

Sơ đồ xử lý như vậy có tên gọi là daisy chaining (chuỗi cánh hoa). Điểm nổi bật của sơ đồ này là các thiết bị được gán thứ tự ưu tiên tuỳ thuộc vào vị trí của nó so với arbiter, thiết bị gần hơn thì mức ưu tiên cao hơn.



Hình 4.4 – Phân xử bus tập trung 1 mức nối tiếp

Một số loại bus có nhiều mức độ ưu tiên, với mỗi mức độ ưu tiên có đường yêu cầu bus (bus request) và đường dây cho phép bus (bus grant). Ví dụ: giả sử 1 bus có 2 mức ưu tiên 1 và 2 (các bus thực tế có 4, 8 hay 16 mức). Mỗi thiết bị trong hệ thống máy tính nối với 1 trong các mức yêu cầu bus, các đường thường được sử dụng nhiều hơn được gắn với đường dây có mức ưu tiên cao hơn. Ở ví du, các thiết bị 1, 2 sử dụng mức ưu tiên 1, còn các thiết bị 3, 4 sử dụng mức ưu tiên 2.



Hình 4.5 – Phân xử bus tập trung 2 mức

Nếu có một số thiết bị ở các mức ưu tiên khác nhau cùng yêu cầu, arbiter chỉ phát ra tín hiệu grant đối với yêu cầu có mức ưu tiên cao nhất. Trong số các thiết bị có cùng mức ưu tiên, thiết bị nào gần arbiter hơn sẽ ưu tiên hơn.

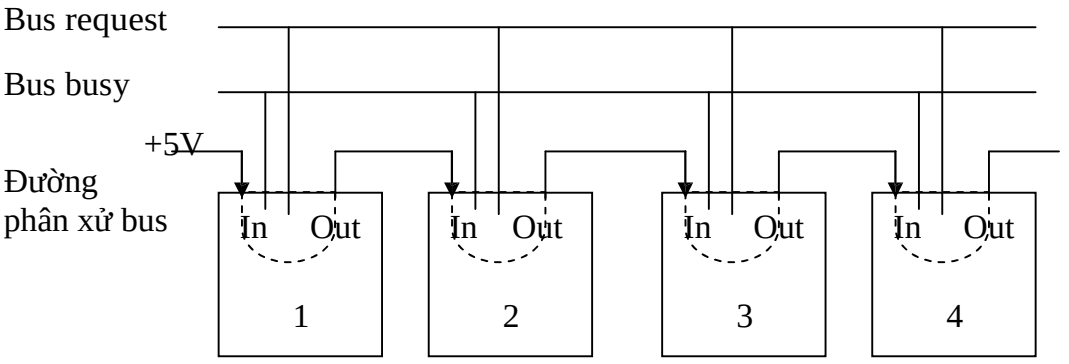
Về mặt kỹ thuật, không cần nối đường grant level 2 giữa các thiết bị vì chúng không bao giờ đòi hỏi bus ở mức 2. Tuy nhiên, trong thực tế để thuận tiện cho việc lắp đặt người ta hay làm như sau: nối tất cả các đường grant thông qua tất cả các thiết bị, như vậy

sẽ dễ dàng hơn là nối các đường grant một cách riêng biệt, và từ đó căn cứ vào thiết bị nào có quyền ưu tiên cao hơn.

Một arbiter có đường dây thứ 3 nối tới các thiết bị để các thiết bị xác nhận đã nhận được tín hiệu grant và chiếm dụng bus – đường ACK (acknowledgement). Ngay sau 1 thiết bị phát tín hiệu tích cực trên đường dây ACK, có thể đảo tín hiệu trên các đường dây request và grant xuống mức không tích cực. Các thiết bị khác có thể yêu cầu bus khi thiết bị đầu tiên đang dùng bus. Khi sự truyền thông kết thúc, bus master kế tiếp sẽ được lựa chọn. Cách làm việc như vậy làm tăng hiệu quả sử dụng bus, nhưng cần thêm 1 đường truyền tín hiệu và cấu trúc thiết bị cũng phức tạp hơn. Các chip trong máy tính PDP-11 và các chip Motorola làm việc với các bus như vậy.

1.4.2. Phân xử bus không tập trung:

Trong Multibus, người ta cho phép có thể lựa chọn sử dụng phân xử bus tập trung hay không tập trung, cơ chế phân xử bus không tập trung được thực hiện như sau:



Hình 4.6 – Phân xử bus không tập trung trong multibus

Cơ chế sử dụng 3 đường dây, không phụ thuộc vào số lượng thiết bị nối với bus:

- Bus request: yêu cầu chiếm dụng bus.
- Bus busy: đường báo bận, được bus master đặt ở mức tích cực khi có thiết bị đang chiếm dụng bus
- Bus arbitration: được mắc nối tiếp thành 1 chuỗi xích qua tất cả các thiết bị ngoại vi. Đầu của chuỗi này được gắn với mức điện áp 5V của nguồn nuôi.

Khi không có đơn vị nào yêu cầu chiếm dụng bus, đường dây phân xử bus truyền mức tích cực tới tất cả các thiết bị trong chuỗi xích. Khi 1 đơn vị nào đó muốn chiếm dụng bus, đầu tiên nó kiểm tra xem bus có rảnh không và đầu vào In của đường trọng tài bus có mức tích cực hay không. Nếu không (not active) thì nó không trở thành bus master. Ngược lại, nó sẽ đảo đầu Out thành không tích cực, làm cho các thiết bị đứng sau nó trong chuỗi xích có đầu In không tích cực.

Khi trạng thái có thể hiểu lầm (khoảng thời gian tín hiệu trên đầu In và Out đang thay đổi) qua đi, chỉ còn lại duy nhất 1 thiết bị có đầu In tích cực và Out không tích cực.

Thiết bị này trở thành bus master, nó sẽ đặt bus busy tích cực và bắt đầu truyền thông tin trên bus.

1.5. Xử lý ngắt

Ở trên, ta chỉ khảo sát các chu kỳ bus thông thường, trong đó master nhận hay gởi thông tin từ / đến slave. Một ứng dụng quan trọng nữa của bus là dùng để xử lý ngắt. Khi CPU ra lệnh cho thiết bị I/O làm một việc gì đó, nó thường chờ đợi tín hiệu ngắt do thiết bị I/O phát ra khi hoàn thành công việc được CPU yêu cầu. Khi nhận được tín hiệu ngắt, CPU sẽ đáp ứng ngay, có thể nhận dữ liệu do thiết bị I/O truyền về, hay gởi tiếp dữ liệu ra thiết bị I/O, hay CPU sẽ sử dụng bus cho một thao tác khác…. Như vậy chính ngắt phát ra tín hiệu yêu cầu sử dụng bus.

Vì có thể nhiều thiết bị ngoại vi cùng phát ra ngắt, cho nên cần có 1 cơ chế phân xử giống như đối với các bus thông thường. Giải pháp thường dùng là gán các mức độ ưu tiên cho các thiết bị và sử dụng 1 arbiter tập trung để trao quyền ưu tiên cho các thiết bị quan trọng thường xuyên được sử dụng. Hiện trên thị trường có những chip điều khiển ngắt được tiêu chuẩn hóa và được sử dụng rộng rãi. Ví dụ: Các máy IBM PC, PC/AT, PS/2 và cả dòng máy tương thích với IBM PC đều sử dụng chip 8259A.

Có thể nối 8 chip điều khiển I/O tới các đầu IRx (Interrupt request) của 8259A. Khi có 1 thiết bị nào đó muốn ngắt, nó đặt mức tích cực lên chân Irx, 8259A nhận được tín hiệu tích cực ở 1 hay một số đầu vào Irx thì sẽ đặt mức tích cực lên đầu dây INT. Tín hiệu INT sẽ truyền trực tiếp đến chân Interrupt của CPU. Khi CPU có thể xử lý được ngắt, nó gởi lại 1 tín hiệu chấp nhận ngắt cho 8259A. Lúc này, CPU chờ 8259A chỉ ra I/O nào yêu cầu ngắt, bằng cách gởi số hiệu của I/O đó lên bus dữ liệu (D0-D7) để đi đến CPU. Sau đó, phần cứng CPU sẽ sử dụng con số đó để tính chỉ số trong 1 bảng con trỏ - bảng vector ngắt (interrupt vector) để tìm địa chỉ chương trình con, cho chạy chương trình này để phục vụ ngắt. Các chương trình con này gọi là chương trình con xử lý ngắt.

2. Bus mở rộng (Expansion bus)

Bus mở rộng cho phép PC liên lạc được với các thiết bị ngoại vi, các thiết bị này được cài đặt qua các khe cắm mở rộng (expansion slot). Các thông số chính của bus mở rộng: tốc độ truyền tối đa giữa các thiết bị với nhau và giữa các thiết bị với bộ nhớ chính, số đường địa chỉ (số lượng ô nhớ có thể được truy xuất bởi 1 thiết bị), số đường ngắt cứng, ….

2.1. Bus ISA (Industry Standard Architecture)

Bus ISA dùng cho hệ thống chỉ được điều khiển bởi 1 CPU trên bảng mạch chính, tức là tất cả các chương trình và thiết bị đều chỉ được điều khiển bởi CPU đó. Tần số làm việc cực đại là 8.33 MHz (tốc độ chuyển tải cực đại là 16.66 MBps với số liệu 2 bytes). Bề rộng dữ liệu là 8 hay 16 bits. ISA có 24 đường địa chỉ nên quản lý được 16 MB bộ nhớ. Bus ISA tương thích 90% với bus AT.

2.2. Bus EISA và MCA

Sử dụng cho các CPU 32 bits (số liệu và đường địa chỉ) từ 80386 trở đi.

2.2.1. Bus EISA (Extended ISA)

Đây là chuẩn mở rộng của ISA để bố trí các dữ liệu 32 bits nhưng vẫn giữ được sự tương thích với mạch nối ghép cũ. Bus EISA có 2 nấc, các tín hiệu ISA được gửi qua nấc trên và các tín hiệu phụ trợ EISA qua nấc dưới. Các đặc trưng của EISA như sau:

- Về mặt cơ khí: có nhiều chân cắm hơn nhưng vẫn tương thích với ISA.
- Độ rộng dữ liệu: có thể truy xuất 2 đường 8 bits (tương thích với ISA), hay 2 đường 16 bits. Do đó, đơn vị quản lý bus 32 bits có thể chuyển tải 4 byte với bộ nhớ hoặc thiết bị ngoại vi. Điều này góp phần tăng tốc độ truyền tải lên khoảng 33 MBps so với 16.66 MBps của ISA.
- Độ rộng địa chỉ: ngoài 24 đường như ISA còn thêm 8 đường bổ sung nữa, do đó có thể định địa chỉ trong 4 GB bộ nhớ.
- Phần cứng được thiết kế theo hệ thống EISA phức tạp hơn ISA vì nó cũng phải thực hiện các chu kỳ bus tương thích với ISA. EISA có thể thực hiện phân xử bus, nó cho phép vi xử lý nằm ngoài bảng mạch chính có thể điều khiển toàn bộ bus. Điều này rất hiệu quả trong các hệ thống đa xử lý (multiprocessor). Hãng Intel đã phát triển 4 chip điện tử phục vụ cho bus EISA như sau:
    - ISP (Intergrated system peripheral)
    - BMIC (Bus master interface controller)
    - EBC (EISA bus controller)
    - EBB (EISA bus buffer)

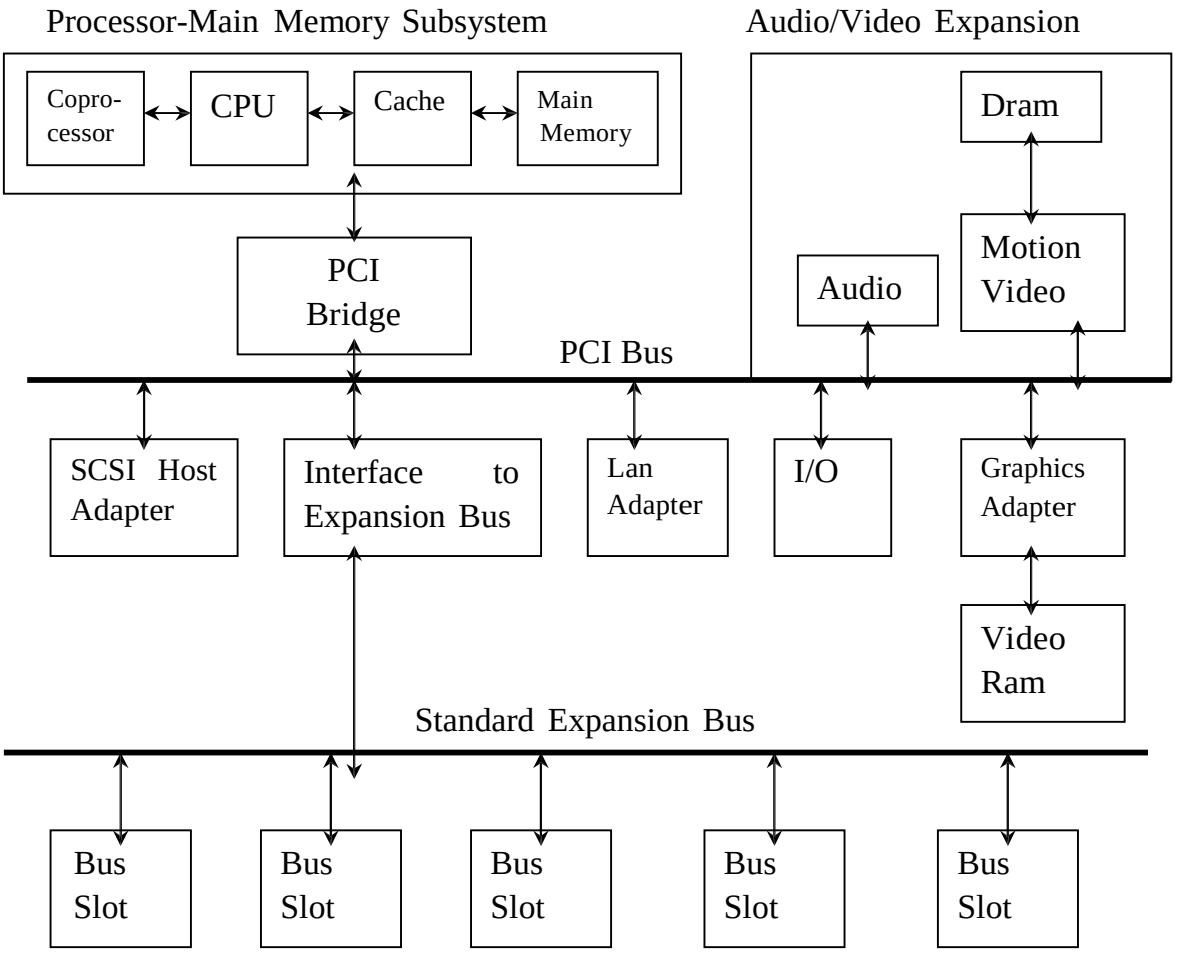
2.2.2. Bus MCA (Micro Channel Architecture)

Phục vụ cho hệ thống IBM PS/2 không tương thích với bus ISA, có thể hoạt động với 16 hay 32 bits dữ liệu. Nó có nhiều đường dẫn hơn ISA, thiết kế phức tạp cho phép giảm bớt các nhiễu cao tần của PC tới các thiết bị xung quanh. Tốc độ truyền dữ liệu có thể lên tới 160 MBps.

2.3. Bus cục bộ (Local Bus)

Nhược điểm của các bus chuẩn trên là mặc dù xung clock của CPU rất cao nhưng cũng chỉ làm việc với các ngoại vi với tốc độ truyền tải không quá 33MBps. Điều này không thể đáp ứng được tốc độ của các card đồ hoạ cắm vào khe cắm của bus mở rộng trong chế độ đồ họa. Chuẩn các *bus cục bộ* tạo thêm *các khe cắm mở rộng* nối trực tiếp vào bus cục bộ (bus nối giữa CPU và các bộ đệm). Do vậy, bus mở rộng loại này cho phép truy xuất lên trên 32 bit cũng như tận dụng được tốc độ xung clock của chính CPU, tránh được rào cản 8.33MHz của bus hệ thống. Theo hướng giải quyết này, Intel đã phát triển bus PCI và Uỷ ban VESA (Video Electronics Standards Association) đã phát triển bus VL.

2.3.1. Bus PCI (Peripheral Component Interconnect)



Hình 4.7 - Sơ đồ bus PCI

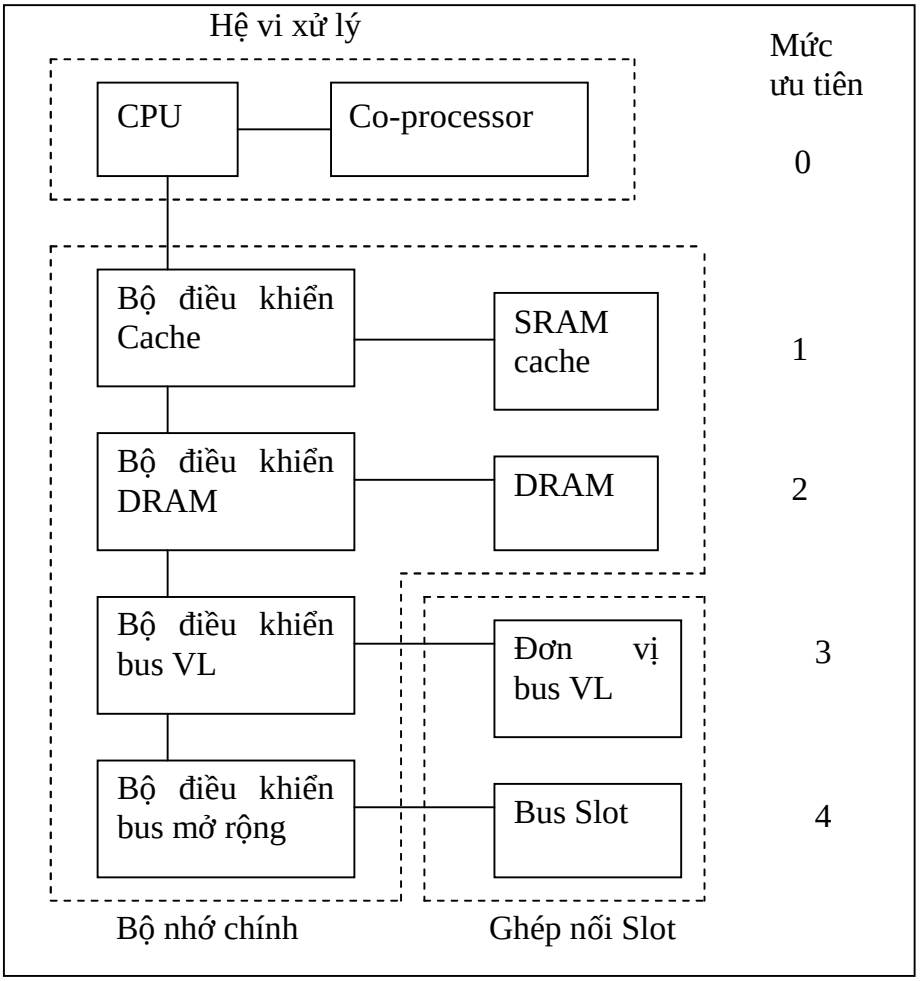
Bus PCI là bus của i486 trong đó dữ liệu và địa chỉ được gởi đi theo cách thức dồn kênh (multiplexing), các đường địa chỉ và dữ liệu được dồn chung trên các đường của PCI. Cách này tiết kiệm được số chân PCI nhưng lại hạn chế tốc độ vì phải cần 2 xung clock cho một quá trình truyền dữ liệu (1 cho địa chỉ và 1 cho dữ liệu). Việc nối giữa CPU, bộ nhớ chính, và bus PCI được thực hiện bằng *cầu PCI* (PCI bridge), qua đó bus PC sẽ phục vụ cho tất cả các đơn vị của bus PCI. Tối đa là 10 thiết bị có thể được nối tới bus PCI, trong đó cầu PCI được coi là một. Chu kỳ bus của PCI đạt gần bằng tốc độ chu kỳ bus của i486. Nó có thể hoạt động với độ rộng 32 bits dữ liệu và tốc độ 33MHz (có thể đạt 64 bits với tốc độ 66 MHz). Một điểm mạnh của PCI là dữ liệu được truyền tải theo kiểu cụm (burst)**,** trong đó địa chỉ chỉ truyền đi 1 lần, sau đó nó sẽ được hiểu ngầm bằng cách cho các đơn vị phát hoặc thu đếm lên trong mỗi xung clock. Do đó, bus PCI hầu như được lấp đầy bởi dữ liệu. Tốc độ truyền tối đa trong kiểu burst có thể lên đến 120MBps.

2.3.2. Bus VL (VESA local bus)

Giống như PCI, bus VL cũng phân cách giữa hệ CPU, bộ nhớ chính, và bus mở rộng chuẩn. Thông qua bus cục bộ trên board mạch chính, nó có thể để điều khiển tối đa 3

thiết bị ngoại vi. Khe cắm VL có 116 tiếp điểm. Bus VL chạy với xung clock bên ngoài CPU, như vậy trong các máy DX2 thì tần số này chỉ bằng một nửa clock CPU. Về mặt logic, mỗi một thiết bị có thể ở một trong hai vị trí: LMB (Local bus master) hoặc LBT (Local bus target). Bộ phận điều khiển bus cục bộ LBC (local bus controller) trên main board sẽ quyết định thiết bị nào sẽ trở thành LMB , tức là được nắm quyền điều khiển bus và cho phép nhường quyền đó cho thiết bị có quyền ưu tiên cao hơn. Thường có 3 cấp ưu tiên được sắp xếp theo thứ tự giảm dần như sau: DMA/làm tươi, CPU/đơn vị làm chủ bus (bus master) và các đơn vị làm chủ bus khác. Thiết bị nào ở vị trí LBT thì không có khả năng làm các việc liên quan đến chuyển tải dữ liệu.

Bus VL chỉ làm việc với 32 bits, trong tương lai sẽ được mở rộng đến 64 bits.



Hình 4.8 – Bus VL

Chương 5
TỔ CHỨC VÀO/RA

1. Giao tiếp nối tiếp

1.1. Cấu trúc cổng nối tiếp

Cổng nối tiếp được sử dụng để truyền dữ liệu hai chiều giữa máy tính và ngoại vi, có các ưu điểm sau:

- Khoảng cách truyền xa hơn truyền song song.
- Số dây kết nối ít.
- Có thể truyền không dây dùng hồng ngoại.
- Có thể ghép nối với vi điều khiển hay PLC (Programmable Logic Device).
- Cho phép nối mạng.
- Có thể tháo lắp thiết bị trong lúc máy tính đang làm việc.
- Có thể cung cấp nguồn cho các mạch điện đơn giản

Các thiết bị ghép nối chia thành 2 loại: DTE (Data Terminal Equipment) và DCE (Data Communication Equipment). DCE là các thiết bị trung gian như MODEM còn DTE là các thiết bị tiếp nhận hay truyền dữ liệu như máy tính, PLC, vi điều khiển, … Việc trao đổi tín hiệu thông thường qua 2 chân RxD (nhận) và TxD (truyền). Các tín hiệu còn lại có chức năng hỗ trợ để thiết lập và điều khiển quá trình truyền, được gọi là các tín hiệu bắt tay (handshake). Ưu điểm của quá trình truyền dùng tín hiệu bắt tay là có thể kiểm soát đường truyền.

Tín hiệu truyền theo chuẩn RS-232 của EIA (Electronics Industry Associations). Chuẩn RS-232 quy định mức logic 1 ứng với điện áp từ -3V đến -25V (mark), mức logic 0 ứng với điện áp từ 3V đến 25V (space) và có khả năng cung cấp dòng từ 10 mA đến 20 mA. Ngoài ra, tất cả các ngõ ra đều có đặc tính chống chập mạch.

Chuẩn RS-232 cho phép truyền tín hiệu với tốc độ đến 20,000 bps nhưng nếu cáp truyền đủ ngắn có thể lên đến 115,200 bps.

Các phương thức nối giữa DTE và DCE:

- Đơn công (simplex connection): dữ liệu chỉ được truyền theo 1 hướng.

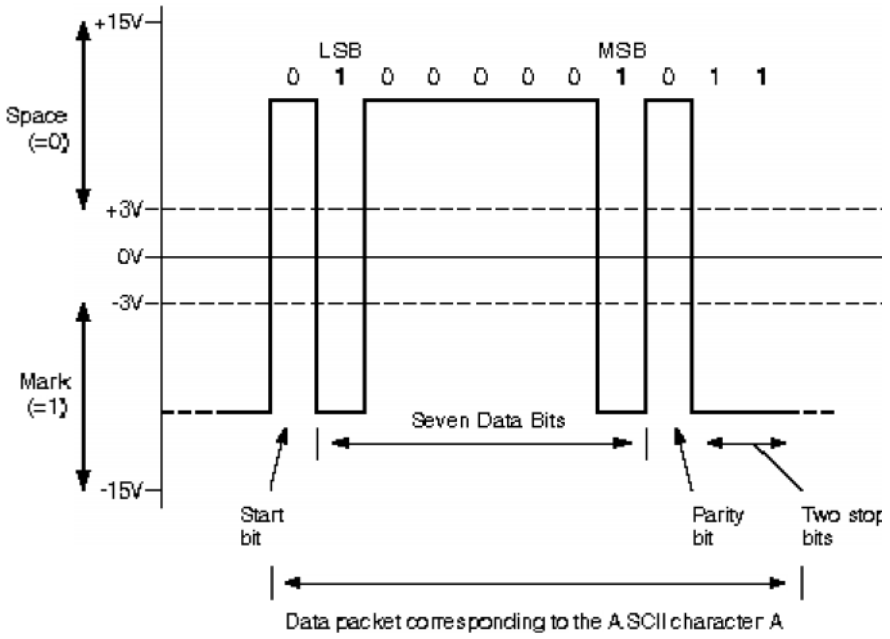
- Bán song công (half-duplex): dữ liệu truyền theo 2 hướng, nhưng mỗi thời điểm chỉ được truyền theo 1 hướng.

- Song công (full-duplex): số liệu được truyền đồng thời theo 2 hướng.

Định dạng của khung truyền dữ liệu theo chuẩn RS-232 như sau:

	Start	D0	D1	D2	D3	D4	D5	D6	D7	P	Stop
	0										1

Khi không truyền dữ liệu, đường truyền sẽ ở trạng thái mark (điện áp -10V). Khi bắt đầu truyền, DTE sẽ đưa ra xung Start (space: 10V) và sau đó lần lượt truyền từ D0 đến D7 và Parity, cuối cùng là xung Stop (mark: -10V) để khôi phục trạng thái đường truyền. Dạng tín hiệu truyền mô tả như sau (truyền ký tự A):



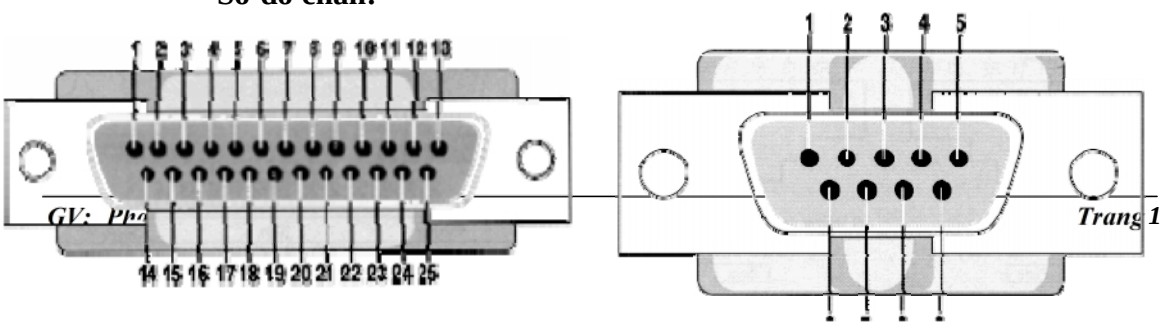
Hình 5.1 – Tín hiệu truyền của ký tự ‘A’

Các đặc tính kỹ thuật của chuẩn RS-232 như sau:

Chiều dài cable cực đại	15m
Tốc độ dữ liệu cực đại	20 Kbps
Điện áp ngõ ra cực đại	± 25V
Điện áp ngõ ra có tải	± 5V đến ± 15V
Trở kháng tải	3K đến 7K
Điện áp ngõ vào	± 15V
Độ nhạy ngõ vào	± 3V
Trở kháng ngõ vào	3K đến 7K

Các tốc độ truyền dữ liệu thông dụng trong cổng nối tiếp là: 1,200 bps, 4,800 bps, 9,600 bps và 19,200 bps.

Sơ đồ chân:



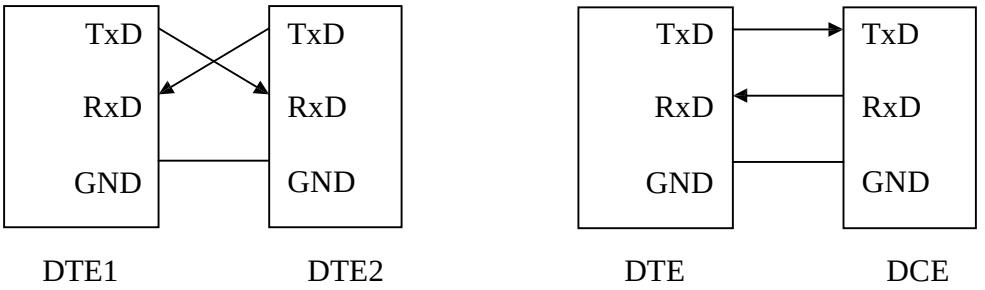
Hình 5.2 – Sơ đồ chân cổng nối tiếp

Cổng COM có hai dạng: đầu nối DB25 (25 chân) và đầu nối DB9 (9 chân) mô tả như hình 5.2. Ý nghĩa của các chân mô tả như sau:

D25	**D9**	**Tín hiệu**	**Hướng truyền**	**Mô tả**
1	-	-	-	Protected ground: nối đất bảo vệ
2	3	TxD	DTEÆDCE	Transmitted data: dữ liệu truyền
3	2	RxD	DCEÆDTE	Received data: dữ liệu nhận
4	7	RTS	DTEÆDCE	Request to send: DTE yêu cầu truyền dữ liệu
5	8	CTS	DCEÆDTE	Clear to send: DCE sẵn sàng nhận dữ liệu
6	6	DSR	DCEÆDTE	Data set ready: DCE sẵn sàng làm việc
7	5	GND	-	Ground: nối đất (0V)
8	1	DCD	DCEÆDTE	Data carier detect: DCE phát hiện sóng mang
20	4	DTR	DTEÆDCE	Data terminal ready: DTE sẵn sàng làm việc
22	9	RI	DCEÆDTE	Ring indicator: báo chuông
23	-	DSRD	DCEÆDTE	Data signal rate detector: dò tốc độ truyền
24	-	TSET	DTEÆDCE	Transmit Signal Element Timing: tín hiệu định thời truyền đi từ DTE
15	-	TSET	DCEÆDTE	Transmitter Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
17	-	RSET	DCEÆDTE	Receiver Signal Element Timing: tín hiệu định thời truyền từ DCE để truyền dữ liệu
18	-	LL		Local Loopback: kiểm tra cổng
21	-	RL	DCEÆDTE	Remote Loopback: Tạo ra bởi DCE khi tín hiệu nhận từ DCE lỗi
14	-	STxD	DTEÆDCE	Secondary Transmitted Data
16	-	SRxD	DCEÆDTE	Secondary Received Data
19	-	SRTS	DTEÆDCE	Secondary Request To Send
13	-	SCTS	DCEÆDTE	Secondary Clear To Send
12	-	SDSRD	DCEÆDTE	Secondary Received Line Signal Detector
25	-	TM		Test Mode
9	-			Dành riêng cho chế độ test
10	-			Dành riêng cho chế độ test
11				Không dùng

1.2. Truyền thông giữa hai nút

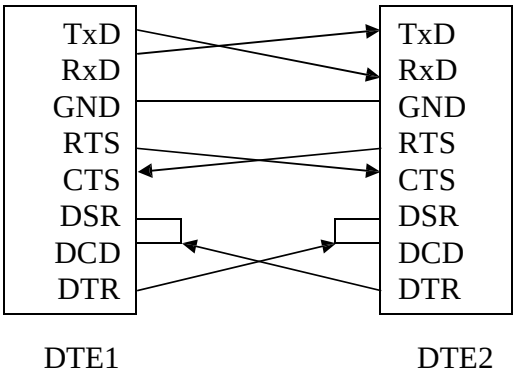
Các sơ đồ khi kết nối dùng cổng nối tiếp:



Hình 5.3 – Kết nối đơn giản trong truyền thông nối tiếp

Khi thực hiện kết nối như trên, quá trình truyền phải bảo đảm tốc độ ở đầu phát và thu giống nhau. Khi có dữ liệu đến DTE, dữ liệu này sẽ được đưa vào bộ đệm và tạo ngắt.

Ngoài ra, khi thực hiện kết nối giữa hai DTE, ta còn dùng sơ đồ sau:



Hình 5.4 – Kết nối trong truyền thông nối tiếp dùng tín hiệu bắt tay

Khi DTE1 cần truyền dữ liệu thì cho DTR tích cực Æ tác động lên DSR của DTE2 cho biết sẵn sàng nhận dữ liệu và cho biết đã nhận được sóng mang của MODEM (ảo). Sau đó, DTE1 tích cực chân RTS để tác động đến chân CTS của DTE2 cho biết DTE1 có thể nhận dữ liệu. Khi thực hiện kết nối giữa DTE và DCE, do tốc độ truyền khác nhau nên phải thực hiện điều khiển lưu lượng. Quá trinh điều khiển này có thể thực hiện bằng phần mềm hay phần cứng. Quá trình điều khiển bằng phần mềm thực hiện bằng hai ký tự Xon và Xoff. Ký tự Xon được DCE gởi đi khi rảnh (có thể nhận dữ liệu). Nếu DCE bận thì sẽ gởi ký tự Xoff. Quá trình điều khiển bằng phần cứng dùng hai chân RTS và CTS. Nếu DTE muốn truyền dữ liệu thì sẽ gởi RTS để yêu cầu truyền, DCE nếu có khả năng nhận dữ liệu (đang rảnh) thì gởi lại CTS.

1.3. Truy xuất trực tiếp thông qua cổng

Các cổng nối tiếp trong máy tính được đánh số là COM1, COM2, COM3, COM4 với các địa chỉ như sau:

Tên	Địa chỉ	Ngắt	Vị trí chứa địa chỉ
COM1	3F8h	4	0000h:0400h
COM2	2F8h	3	0000h:0402h
COM3	3E8h	4	0000h:0404h
COM4	2E8h	3	0000h:0406h

Giao tiếp nối tiếp trong máy tính sử dụng vi mạch UART với các thanh ghi cho trong bảng sau:

Offset	DLAB	R/W	Tên	Chức năng
0	0	W	THR	Transmitter Holding Register (đệm truyền)
	0	R	RBR	Receiver Buffer Register (đệm thu)
	1	R/W	BRDL	Baud Rate Divisor Latch (số chia byte thấp)
1	0	R/W	IER	Interrupt Enable Register (cho phép ngắt)
	1	R/W	BRDH	Số chia byte cao
2		R	IIR	Interrupt Identification Register (nhận dạng ngắt)
		W	FCR	FIFO Control Register
3		R/W	LCR	Line Control Register (điều khiển đường dây)
4		R/W	MCR	Modem Control Register (điều khiển MODEM)
5		R	LSR	Line Status Register (trạng thái đường dây)
6		R	MSR	Modem Status Register (trạng thái MODEM)
7		R/W		Scratch Register (thanh ghi tạm)

Các thanh ghi này có thể truy xuất trực tiếp kết hợp với địa chỉ cổng (ví dụ như thanh ghi cho phép ngắt của COM1 có địa chỉ là BA_{COM1} + 1 = 3F9h.

IIR (Interrupt Identification):

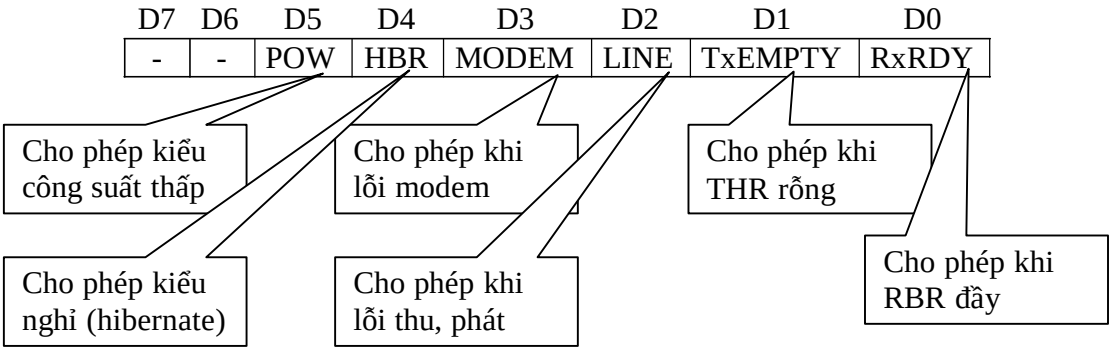
IIR xác định mức ưu tiên và nguồn gốc của yêu cầu ngắt mà UART đang chờ phục vụ. Khi cần xử lý ngắt, CPU thực hiện đọc các bit tương ứng để xác định nguồn gốc của ngắt. Định dạng của IIR như sau:

D7 D6	D5	D4	D3	D2 D1	D0
00: không có FIFO 11: cho phép FIFO	Cho phép FIFO 64 byte (trong 16750)	-	1: ngắt time-out (trong 16550)	Xác định nguồn gốc ngắt	0: có ngắt 1: không ngắt

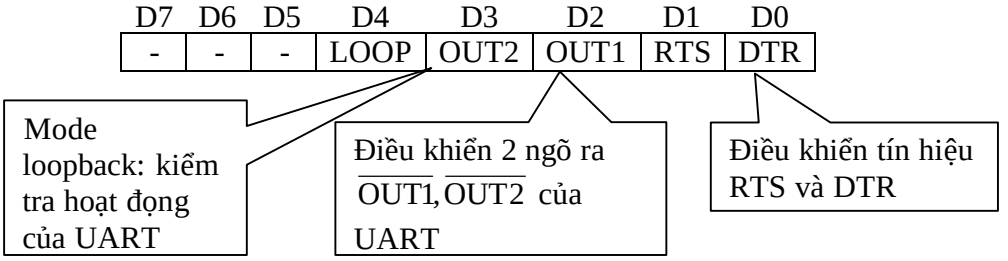
D2	D1	Ưu tiên	Tên	Nguồn	D2 – D0 bị xoá khi
0	0	4	Đường truyền	Lỗi khung, thu đè, lỗi parity, gián đoạn khi thu	Đọc LSR
0	1	3	Đệm thu	Đệm thu đầy	Đọc RBR
1	0	2	Đệm phát	Đệm phát rỗng	Đọc IIR, ghi THR
1	1	1	Modem	CTS, DSR, RI, RLSD	Đọc MSR

IER (Interrupt Enable Register):

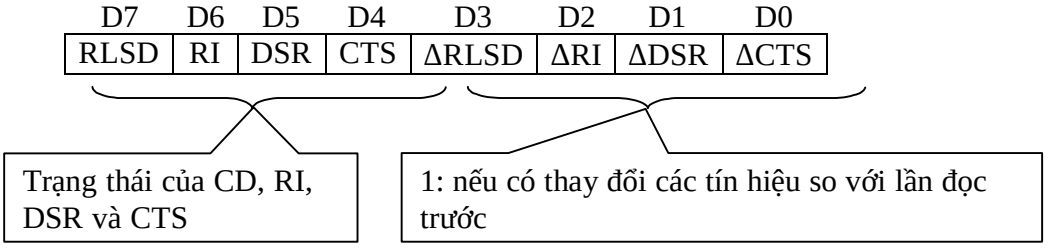
IER cho phép hay cấm các nguyên nhân ngắt khác nhau (1: cho phép, 0: cấm ngắt)



MCR (Modem Control Register):



MSR (Modem Status Register):



LSR (Line Status Register):

D7	D6	D5	D4	D3	D2	D1	D0
FIE	TSRE	THRE	BI	FE	PE	OE	RxDR

FIE: FIFO Error – sai trong FIFO

TSRE: Transmitter Shift Register Empty – thanh ghi dịch rỗng (=1 khi đã phát 1 ký tự và bị xoá khi có 1 ký tự chuyển đến từ THR.

THRE: Transmitter Holding Register Empty (=1 khi có 1 ký tự đã chuyển từ THR – TSR và bị xoá khi CPU đưa ký tự tới THR).

BI: Break Interrupt (=1 khicó sự gián đoạn khi truyền, nghĩa là tồn tại mức logic 0 trong khoảng thời gian dài hơn khoảng thời gian truyền 1 byte và bị xoá khi CPU đọc LSR)

FE: Frame Error (=1 khi có lỗi khung truyền và bị xoá khi CPU đọc LSR)

PE: Parity Error (=1 khi có lỗi parity và bị xoá khi CPU đọc LSR)

OE: Overrun Error (=1 khi có lỗi thu đè, nghĩa là CPU không đọc kịp dữ liệu làm cho quá trình ghi chồng lên RBR xảy ra và bị xoá khi CPU đọc LSR)

RxDR: Receiver Data Ready (=1 khi đã nhận 1 ký tự và đưa vào RBR và bị xoá khi CPU đọc RBR).

LCR (Line Control Register):

D7	D6	D5	D4	D3	D2	D1	D0
DLAB	SBCB	PS2	PS1	PS0	STB	WLS1	WLS0

DLAB (Divisor Latch Access Bit) = 0: truy xuất RBR, THR, IER, = 1 cho phép đặt bộ chia tần trong UART để cho phép đạt tốc độ truyền mong muốn.

UART dùng dao động thạch anh với tần số 1.8432 MHz đưa qua bộ chia 16 thành tần số 115,200 Hz. Khi đó, tuỳ theo giá trị trong BRDL và BRDH, ta sẽ có tốc độ mong muốn. Ví dụ như đường truyền có tốc độ truyền 2,400 bps có giá trị chia 115,200 / 2,400 = 48d = 0030h Æ BRDL = 30h, BRDH = 00h.

Một số giá trị thông dụng xác định tốc độ truyền cho như sau:

Tốc độ (bps)	**BRDH**	**BRDL**
1,200	00h	60h
2,400	00h	30h
4,800	00h	18h
9,600	00h	0Ch
19,200	00h	06h
38,400	00h	03h
57,600	00h	02h
115,200	00h	01h

SBCB (Set Break Control Bit) =1: cho phép truyền tín hiệu Break (=0) trong khoảng thời gian lớn hơn một khung

PS (Parity Select):

PS2	**PS1**	**PS0**	**Mô tả**
X	X	0	Không kiểm tra
0	0	1	Kiểm tra lẻ
0	1	1	Kiểm tra chẵn
1	0	1	Parity là mark
1	1	1	Parity là space

STB (Stop Bit) = 0: 1 bit stop, =1: 1.5 bit stop (khi dùng 5 bit dữ liệu) hay 2 bit stop (khi dùng 6, 7, 8 bit dữ liệu).

WLS (Word Length Select):

WLS1	WLS0	Độ dài dữ liệu
0	0	5 bit
0	1	6 bit
1	0	7 bit
1	1	8 bit

Một ví dụ khi lập trình trực tiếp trên cổng như sau:

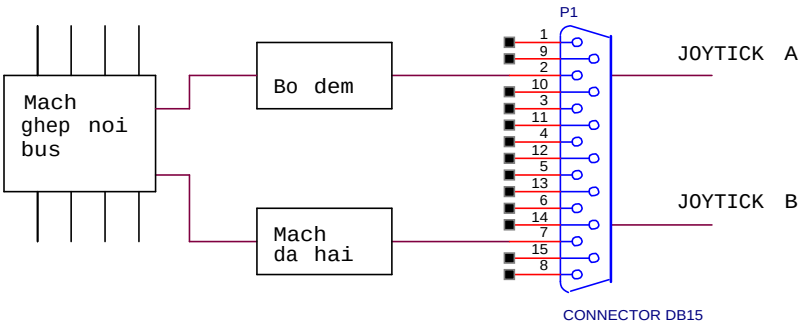
```
.MODEL SMALL
.STACK 100h
.DATA
     Com1      EQU 3F8h
     Com_int   EQU 08h
     Buffer    DB   251 DUP(?)
     Bufferin  DB   0
     Bufferout      DB   0
     Char      DB   ?
     Seg_com   DW   ?    ; Vector ngắt cũ
     Off_com   DW   ?
     Mask_int  DB   ?
     Msg       DB   'Press any key to exit$’
.CODE
Main PROC
     MOV  AX,@DATA
     MOV  DS,AX

     MOV  AH,35h
     MOV  AL,Com_int
     INT  21h
     MOV  Seg_com,ES    ; Lưu vector ngắt cũ
     MOV  Off_com,BX

     PUSH DS
     MOV  BX,CS
     MOV  DS,BX
     LEA  DX,Com_ISR
     MOV  AH,35h        ;Gán vector ngắt mới
     MOV  AL,Com_int
     INT  21h
     POP  DS

     MOV  DX,Com1+3     ; Địa chỉ LCR
     MOV  AL,80h        ; Set DLAB=1 cho phép định tốc
     OUT  DX,AL         ; độ truyền dữ liệu
```

```
        MOV  DX,Com1        ; Gởi byte thấp
        MOV  AL,0Ch
        OUT  DX,AL

        MOV  DX,Com1+1
        MOV  AL,00h         ; Gởi byte cao=000Ch: xác định
        OUT  DX,AL          ; tốc độ truyền 9600bps

        MOV  DX,Com1+3      ; LCR = 0000 0011B
        MOV  AL,03h         ; DLAB=0, SBCB=0 Æ cấm Break
        OUT  DX,AL          ; PS = 000 Æ no parity
                            ; STB = 0 Æ 1 stop bit
                            ; WLS = 11 Æ 8 bit dữ liệu

        MOV  DX,Com1+4      ; Tác động đến DTR và RTS
        MOV  AL,03h         ; MCR=00000011b Æ DTR=RTS = 1
        OUT  DX,AL          ; Ængõ DTR và RTS của cổng
                            ; nối tiếp = 0

        MOV  DX,21h         ; Kiểm tra trạng thái ngắt
        IN   AL,DX          ; D7 – D0 xác định các IRQi
        MOV  Mask_int,AL    ; =0: cho phép, =1: cấm

        AND  AL,0EFh        ; =1110 1111b Æ cho phép IRQ4
        OUT  DX,AL          ; Æ cho phép COM1

        MOV  AL,01h         ; IER = 0000 0001b Æ cho phép
        MOV  DX,Com1+1      ; ngắt khi RBR đầy
        OUT  DX,AL

        MOV  AH,09h
        LEA  Dx,Msg
        INT  21h

Lap:
        MOV  AH,0Bh
        INT  21h
        CMP  AL,0FFh
        JE   Exit

        MOV  AL,bufferin
        CMP  AL,bufferout
        JE   Lap
        MOV  AL,buffer[bufferout]
        MOV  char,AL
        INC  bufferout
        MOV  AL,bufferout
```

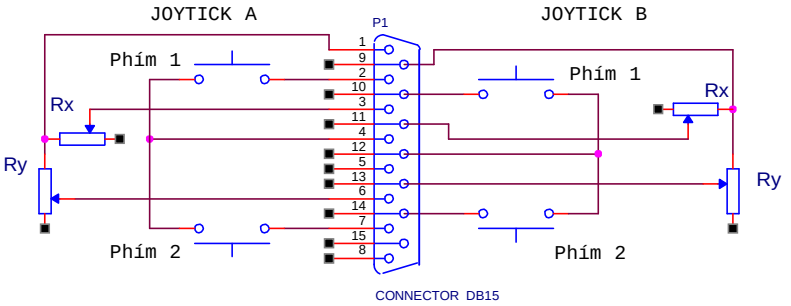
```
        CMP  AL,251
        JNE  Next
        MOV  bufferout,0
Next:
        MOV DL,char         ; Xuất giá trị ra màn hình
        MOV  AH,02h
        INT  21h

        MOV  AL,char        ; Xuất ra cổng nối tiếp
        MOV  DX,Com1
        OUT  DX,AL
        JMP  Lap

Exit:
        MOV  AL,Mask_int
        OUT  21h,AL         ; Khôi phục trạng thái ngắt

        MOV  DX,Off_com
        MOV  BX,Seg_com
        MOV  DS,BX
        MOV  AH,35h         ;Khôi phục vector ngắt
        MOV  AL,Com_int
        INT  21h


        MOV  AH,4Ch
        INT  21h
Main ENDP

Com_ISR  PROC
        MOV  DX,Com1+5      ; Đọc nội dung LSR
        IN   AL,DX
        AND  AL,1           ; Nếu D0 = 1 thì có dữ liệu
        JZ   exit_ISR

        MOV  DX,Com1
        IN   AL,DX
        MOV  buffer[bufferin],AL
        INC  bufferin
        MOV  AL,bufferin
        CMP  AL,251
        JNE  Exit_ISR
        MOV  bufferin,0
Exit_ISR:
        MOV  AL,20h         ; Báo cho PIC kết thúc ngắt
        OUT  20h,AL
        IRET
Com_ISR  ENDP
END Main
```

2. Giao tiếp PC Game

Cấu trúc và chức năng của board ghép nối trò chơi (PC game) như hình bên dưới. Bằng lệnh IN và OUT có thể truy xuất qua địa chỉ 201h.



Hình 5.5 - Cấu trúc và chức năng của board ghép nối trò chơi



Hình 5.6 - Cấu tạo của đầu nối 15 chân và 2 joystick A và B

Chân của đầu nối 15 chân	Sử dụng cho
2	Phím 1 của Joystick A (BA1)
3	Biến trở X của Joystick A
6	Biến trở Y của Joystick A
7	Phím 2 của Joystick A (BA2)
10	Phím 1 của Joystick A (BB1)
11	Biến trở X của Joystick B
13	Biến trở Y của Joystick B
14	Phím 2 của Joystick A (BB2)
1, 8, 9, 15	Vcc (+5V)
4, 5, 12	GND (0V)

Board mạch được nối với bus hệ thống của PC chỉ qua 8 bits thấp của bus dữ liệu, 10 bits thấp của bus địa chỉ và các đường điều khiển $\overline{IOR}$ và $\overline{IOW}$. Một đầu nối 15 chân được nối với board mạch cho phép nối cực đại hai thiết bị cho PC game gọi là joystick. Mỗi joystick có 2 biến trở có giá trị biến đổi từ 0 đến 100kΩ được đặt vuông góc với nhau đại diện cho vị trí x và y của joystick. Thêm nữa chúng có 2 phím bấm, thường là các công tắc thường hở phù hợp với các mức logic cao của các dây trên mạch.

Có thể xác định được trạng thái nhấn hoặc nhả phím một cách dễ dàng bằng lệnh IN tới địa chỉ 201h. Nibble cao chỉ thị trạng thái của phím. Vì board không dùng đường IRQ do đó không có khả năng phát ra 1 ngắt, do vậy board chỉ hoạt động trong chế độ hỏi vòng (polling). Byte trạng thái của board game như sau:

D7	D6	D5	D4	D3	D2	D1	D0
BB_2	BB_1	BA_2	BA_1	BY	BX	AY	AX

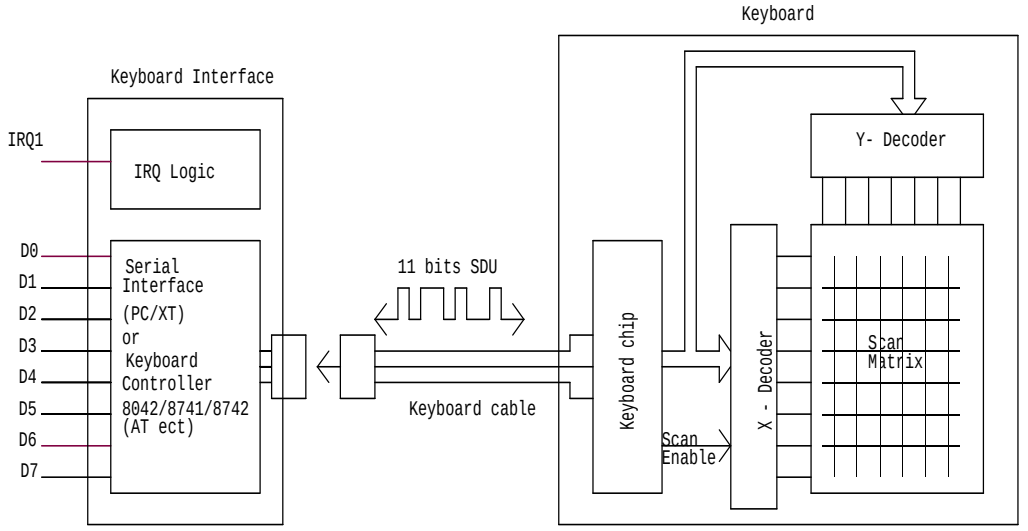
BB_2, BB_1, BA_2, BA_1: Trạng thái của các phím B_2, B_1, A_2, A_1; 1 = nhả; 0 = nhấn

BY, BX, AY, AX: Trạng thái của mạch đa hài tuỳ thuộc vào biến trở tương ứng.

3. Giao tiếp với bàn phím và mouse

3.1. Bàn phím

3.1.1. Nguyên lý hoạt động



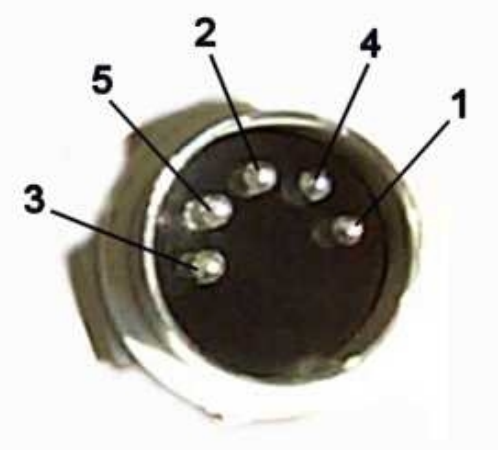
Hình 5.7 - Sơ đồ nguyên lý và các ghép nối của bàn phím

Chip xử lý bàn phím liên tục kiểm tra trạng thái của ma trận quét (scan matrix) để xác định công tắc tại các tọa độ X, Y đang được đóng hay mở và ghi một mã tương ứng vào bộ đệm bên trong bàn phím. Sau đó mã này sẽ được truyền nối tiếp tới mạch ghép nối bàn phím trong PC. Cấu trúc của SDU (Serial Data Unit) cho việc truyền số liệu:

0										10
STRT	DB_0	DB_1	DB_2	DB_3	DB_4	DB_5	DB_6	DB_7	PAR	STOP

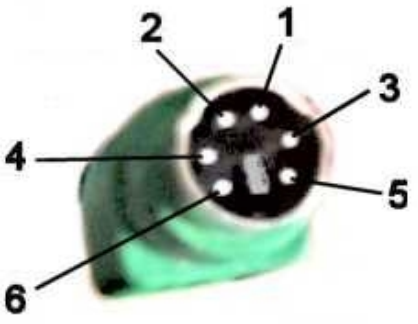
STRT: bit start (luôn bằng 0)
DB_0 - DB_7: bit số liệu từ 0 đến 7.
PAR: bit parity (luôn lẻ)
STOP: bit stop (luôn bằng 1).

Chân 1: clock
Chân 2: dữ liệu
Chân 3: Reset
Chân 4: GND
Chân 5: Vcc



Hình 5.8 – Đầu cắm bàn phím AT

Chân 1: dữ liệu
Chân 2: không dùng
Chân 3: GND
Chân 4: Vcc
Chân 5: clock
Chân 6: không dùng



Hình 5.9 – Đầu cắm bàn phím PS/2

Mỗi phím nhấn sẽ được gán cho 1 mã quét (scan code) gồm 1 byte. Nếu 1 phím được nhấn thì bàn phím phát ra 1 mã make code tương ứng với mã quét truyền tới mạch ghép nối bàn phím của PC. Ngắt cứng INT 09h được phát ra qua IRQ1.

Chương trình xử lý ngắt sẽ xử lý mã này tuỳ theo phím SHIFT có được nhấn hay không. Ví dụ: nhấn phím SHIFT trước, không rời tay và sau đó nhấn ‘C’:

make code được truyền - 42(SHIFT) - 46 (‘C’).

Nếu rời tay nhấn phím SHIFT thì bàn phím sẽ phát ra break code và mã này được truyền như make code. Mã này giống như mã quét nhưng bit 7 được đặt lên 1, do vậy nó tương đương với make code cộng với 128. Tuỳ theo break code, chương trình con xử lý ngắt sẽ xác định trạng thái nhấn hay rời của các phím. Thí dụ, phím SHIFT và ‘C’ được rời theo thứ tự ngược lại với thí dụ trên:

break code được truyền 174 (bằng 46 cộng 128 tương ứng với ‘C’) và 170 (bằng 42 cộng 128 tương ứng với SHIFT).

Phần cứng và phần mềm xử lý bàn phím còn giải quyết các vấn đề vật lý sau:

- Nhấn và nhả phím nhưng không được phát hiện.

- Khử nhiễu rung cơ khí và phân biệt 1 phím được nhấn nhiều lần hay được nhấn chỉ 1 lần nhưng được giữ trong một khoảng thời gian dài.

3.1.2. Lập trình giao tiếp qua các cổng

Bàn phím cũng là một thiết bị ngoại vi nên về nguyên tắc có thể truy xuất nó qua các cổng vào ra.

Các thanh ghi và các port:

Sử dụng 2 địa chỉ port 60h và 64h có thể truy xuất bộ đệm vào, bộ đệm ra và thanh ghi điều khiển của bàn phím.

Port	Thanh ghi	R/W
60h	Đệm ngõ ra	R
60h	Đệm ngõ vào	W
64h	Thanh ghi điều khiển	W
64h	Thanh ghi trạng thái	R

Thanh ghi trạng thái xác định trạng thái hiện tại của bộ điều khiển bàn phím. Thanh ghi này chỉ đọc (read only) và đọc bằng lệnh IN tại port 64h.

7							0
PARE	TIM	AUXB	KEYL	C/D	SYSF	INPB	OUTB

PARE: Lỗi chẵn lẻ của byte cuối cùng được vào từ bàn phím; 1 = có lỗi chẵn lẻ, 0 = không có.

TIM: Lỗi quá thời gian (time-out); 1 = có lỗi, 0 = không có.

AUXB: Đệm ra cho thiết bị phụ (chỉ có ở máy PS/2); 1 = giữ số liệu cho thiết bị, 0 = giữ số liệu cho bàn phím.

KEYL: Trạng thái khóa bàn phím; 1 = không khóa, 0 = khóa.

C/D: Lệnh/dữ liệu; 1 = Ghi qua port 64h, 0 = Ghi qua port 60h.

SYSF: cờ hệ thống; 1 = tự kiểm tra thành công, 0 = reset khi cấp điện

INPB: Trạng thái đệm vào; 1 = dữ liệu CPU trong bộ đệm vào, 0 = đệm vào rỗng.

OUTB: Trạng thái đệm ra; 1 = dữ liệu bộ điều khiển bàn phím trong bộ đệm ra, 0 = đệm ra rỗng.

Thanh ghi điều khiển

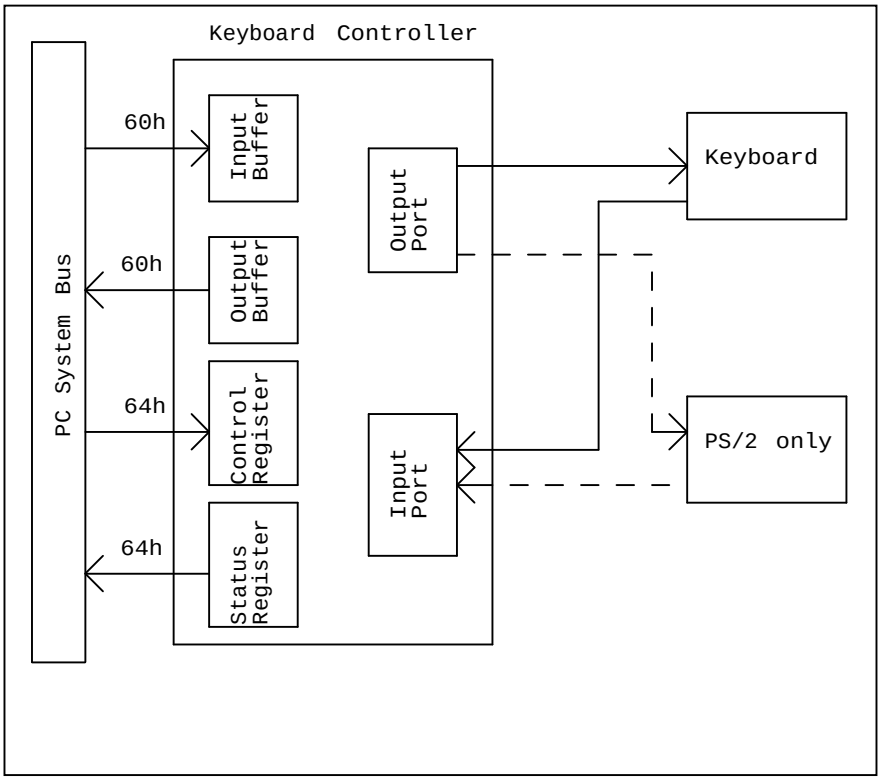
Các lệnh cho bộ điều khiển bàn phím:

Mã	Mô tả
A7h	Cấm thiết bị phụ
A8h	Cho phép thiết bị phụ
A9h	Kiểm tra giao tiếp thiết bị phụ và lưu mã kiểm tra vào bộ đệm ra 00h: không lỗi 01h: CLK ở mức thấp 02h: CLK ở mức cao 03h: DATA ở mức thấp 04h: DATA ở mức cao

	FFh: lỗi khác
AAh	Tự kiểm tra (ghi 55h vào bộ đệm ra nếu không lỗi
ABh	Kiểm tra giao tiếp bàn phím và lưu mã kiểm tra vào bộ đệm ra
ADh	Cấm bàn phím
AEh	Cho phép bàn phím
C0h	Đọc cổng vào và truyền dữ liệu đến bộ đệm ra
C1h	Đọc các bit 3 – 0 của cổng vào và truyền đến các bit 3- 0 của thanh ghi trạng thái cho đến khi INPB = 1
C2h	Đọc các bit 7 – 4 của cổng vào và truyền đến các bit 7- 4 của thanh ghi trạng thái cho đến khi INPB = 1
D0h	Đọc cổng ra
D1h	Ghi cổng ra
D2h	Ghi vào bộ đệm ra và xoá AUXB
D3h	Ghi vào bộ đệm ra và set AUXB
D4h	Ghi byte dữ liệu tiếp theo vào thiết bị phụ

Khóa bàn phím:

```
Start:
IN AL, 64h          ; đọc byte trạng thái
TEST AL, 02h        ; kiểm tra bộ đệm có đầy hay không
JNZ start
OUT 64h, 0ADh       ; khóa bàn phím
```



Hình 5.10 - Bộ điều khiển bàn phím

Các lệnh cho bàn phím:

Mã	Lệnh	Mô tả
EDh	Bật/tắt LED	Bật/tắt các đèn led của bàn phím
EEh	Echo	Trả về byte EEh
F0h	Đặt/nhận dạng mã quét	Đặt 1 trong 3 tập mã quét và nhận diện các mã quét tập mã quét hiện tại.
F2h	Nhận diện bàn phím	Nhận diện ACK = AT, ACK+abh+41h=MF II.
F3h	Đặt tốc độ lặp lại/trễ	Đặt tốc độ lặp lại và thời gian trễ của bàn phím
F4h	Enable	Cho phép bàn phím hoạt động
F5h	Chuẩn/không cho phép	Đặt giá trị chuẩn và cấm bàn phím.
F6h	Chuẩn/cho phép	Đặt giá trị chuẩn và cho phép bàn phím.
FEh	Resend	Bàn phím truyền ký tự cuối cùng một lần nữa tới bộ điều khiển bàn phím
FFh	Reset	Chạy reset bên trong bàn phím

Thí dụ: lệnh bật đèn led cho phím NUMCLOCK, tắt tất cả các đèn khác.

```
MOV AL,0EDh
OUT  60H, AL
WAIT:
IN  AL, 64H          ; đọc thanh ghi trạng thái
JNZ   WAIT
MOV AL,02h
OUT  60H, AL         ; bật đèn cho numclock
```

Cấu trúc của byte chỉ thị như sau:

7					2	1	0
0	0	0	0	0	CPL	NUM	SCR

CPL: 1 = bật đèn Caps Lock; 0 = tắt
NUM: 1 = bật đèn Num Lock; 0 = tắt
SCR: 1 = bật đèn Scroll Lock; 0 = tắt

3.1.3. Lập trình giao tiếp qua các hàm của DOS, BIOS

BIOS ghi các ký tự do việc nhấn các phím vào bộ đệm tạm thời được gọi là bộ đệm bàn phím (keyboard buffer), có địa chỉ 40h:1Eh, gồm 32 byte và kết thúc ở địa chỉ 40h:3Dh. Mỗi ký tự được lưu trữ bằng 2 byte, byte cao là mã quét, và byte thấp là mã ASCII. Chương trình xử lý ngắt sẽ xác định mã ASCII từ mã quét bằng bảng biến đổi và ghi cả 2 mã vào bộ đệm bàn phím. Bộ đệm bàn phím được tổ chức như bộ đệm vòng (ring buffer) và được quản lý bởi 2 con trỏ. Các giá trị con trỏ được lưu trữ trong vùng dữ liệu của BIOS ở địa chỉ 40h:1Ah và 40h:1Ch. Con trỏ ghi (40h:1Ch) cho biết vị trí còn trống kế tiếp để ghi ký tự nhập, con trỏ đọc (40h:1Ah) cho biết vị trí ký tự đầu tiên sẽ đọc. Từ đó, bộ đệm bàn phím rỗng khi con trỏ ghi và con trỏ đọc trùng nhau Æ bộ đệm chỉ chứa được 15 ký tự.

Các hàm của ngắt 16h:

Hàm 0h - đọc ký tự từ bàn phím, nếu không nhấn thì sẽ chờ

Ra: AH = scancode, AL = mã ASCII. Nếu phím nhấn là các phím đặc biệt thì AL = 0

Hàm 1h - ZF = 1 nếu không có ký tự trong bộ đệm. Giá trị trả về giống như hàm 00h nhưng không xoá ký tự ra khỏi bộ đệm

Hàm 2h - Trả về trạng thái của các phím, kết quả chứa trong AL

7	6	5	4	3	2	1	0
INS	CAPS LOCK	NUM LOCK	SCROLL LOCK	ALT	CTRL	LEFT SHIFT	RIGHT SHIFT

Hàm 10h - Giống hàm 00h nhưng trả về mã mở rộng

Hàm 11h - Giống hàm 01h nhưng trả về mã mở rộng

Hàm 12h - Giống hàm 02h nhưng AH chứa thêm các thông tin

7	6	5	4	3	2	1	0
SYS REQ	CAPS LOCK	NUM LOCK	SCROLL LOCK	RIGHT ALT	RIGHT CTRL	LEFT ALT	LEFT CTRL

Các thí dụ:

- Giả sử phím ‘c’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 2Eh (mã quét cho phím ‘a’); AL = 63h (ASCII cho ‘c’)

- Giả sử phím ‘HOME’ đã được nhấn.

```
MOV AH,00h
INT 16h
```

Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = 0 (các phím chức năng và điều khiển không có mã ASCII)

- Giả sử phím ‘HOME’ đã được nhấn.

```
MOV AH,10h
INT 16h
```

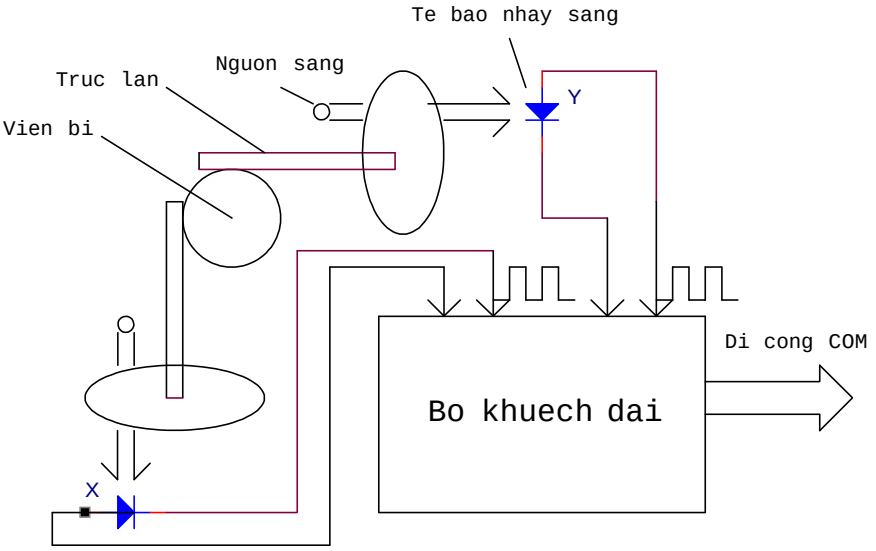
Kết quả: AH = 47h (mã quét cho phím ‘HOME’)
AL = E0h

3.2. Chuột

3.2.1. Cấu tạo

Cấu tạo của chuột rất đơn giản, phần trung tâm là 1 viên bi thép được phủ keo hoặc nhựa được quay khi dịch chuyển chuột. Chuyển động này được truyền tới 2 thanh nhỏ được đặt vuông góc với nhau. Các thanh này sẽ biến chuyển động của chuột theo 2 hướng X,Y thành sự quay tương ứng của 2 đĩa gắn với chúng. Trên 2 đĩa có những lỗ nhỏ liên tục đóng và ngắt 2 chùm sáng tới các sensor nhạy sáng để tạo ra các xung điện. Số các xung điện tỷ lệ với lượng chuyển động của chuột theo các hướng X,Y và

số xung trên 1 sec biểu hiện tốc độ của chuyển động chuột. Kèm theo đó có 2 hay 3 phím bấm.



Hình 5.11 - Sơ đồ cấu tạo của chuột

3.2.2. Mạch ghép nối và chương trình điều khiển chuột

Hầu hết chuột được nối với PC qua cổng nối tiếp, qua đó chuột cũng được cấp nguồn nuôi từ PC. Khi dịch chuyển hoặc nhấn, nhả các phím chuột, nó sẽ phát ra một gói các số liệu tới mạch giao tiếp và mạch sẽ phát ra 1 ngắt. Phần mềm điều khiển chuột làm các nhiệm vụ: chuyển ngắt tới mạch giao tiếp nối tiếp xác định, đọc gói số liệu và cập nhật các giá trị bên trong liên quan tới trạng thái của bàn phím cũng như vị trí của chuột. Hơn nữa, nó còn cung cấp 1 giao tiếp mềm qua ngắt của chuột là 33h để định các giá trị bên trong này cũng như làm dịch chuyển con trỏ chuột trên màn hình tương ứng với vị trí của chuột.

Có thể chọn kiểu con trỏ chuột cứng hoặc mềm trong chế độ văn bản hay con trỏ chuột đồ hoạ trong chế độ đồ họa. Các hàm 09h và 0Ah trong ngắt 33h cho phép định nghĩa loại và dạng con trỏ chuột.

3.2.3. Chương trình với con trỏ

Ngắt 33h cho phép xác định vị trí, số lần click chuột và hình dạng con trỏ (số thứ tự hàm chứa trong AX).

Hàm	Ý nghĩa	Tham số
0	Reset chuột	Ra: AX = 0: nếu có, = 1: không BX = số nút nhấn
1	Hiển thị con trỏ	
2	Ẩn con trỏ	
3	Nhận vị trí con trỏ và trạng thái nút	Ra: BX: trạng thái nút (D0: nút trái, D1: nút phải, D2: nút

		giữa) (= 0: nhả, = 1: nhấn) CX: vị trí ngang DX: vị trí dọc
4	Đặt vị trí con trỏ	Vào: CX: vị trí ngang DX: vị trí dọc
5	Trạng thái nút và số lần nhấn từ khi gọi	Vào: BX = nút kiểm tra (=0: trái, =1: phải) Ra: AX = trạng thái nút BX = số lần nhấn CX: vị trí ngang DX: vị trí dọc lần nhấn cuối
6	Giống hàm 05h nhưng kiểm tra số lấn nhả	
7	Giới hạn dịch chuyển ngang của con trỏ	Vào: CX = cột trái DX = cột phải
8	Giới hạn dịch chuyển dọc của con trỏ	Vào: CX = dòng dưới DX = dòng trên
9	Xác định hình dạng con trỏ đồ hoạ	Vào: BX = vị trí ngang CX = vị trí dọc ES:DX: địa chỉ mặt nạ màn hình và con trỏ
A	Xác định hình dạng con trỏ văn bản	Vào: BX = 0: con trỏ phần mềm CX = mặt nạ màn hình DX = mặt nạ con trỏ BX = 1: con trỏ phần cứng CX = dòng bắt đầu DX = dòng kết thúc

Chú ý rằng toạ độ con trỏ xác định theo pixel với độ phân giải 640x200 trong khi chế độ văn bản sử dụng toạ độ ký tự 80x25 nên để chuyển sang toạ độ ký tự thì phải chia cho 8. Con trỏ chuột hiển thị trên màn hình đồ hoạ bằng cách thực hiện:

Từ mới = (từ cũ AND mặt nạ màn hình) XOR mặt nạ con trỏ

Nếu ta đặt mặt nạ màn hình là 0 thì ký tự màn hình tại đó sẽ bị xoá.

VD: Con trỏ chuột mềm nhấp nháy và chứa ký tự ‘A’

```
MOV AH,0Ah
MOV BX,0
MOV CX,0 ; mặt nạ màn hình = 0
MOV DH,8Bh;=10001011b Æ màu nền Gray, màu ký tự Cyan
MOV DL,’A’
INT 33h
```

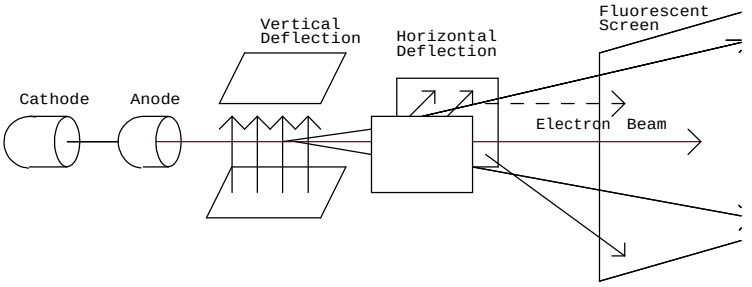
VD: Con trỏ chuột cứng có các đường quét 3 và 8

```
MOV AH,0Ah
MOV BX,1
MOV CX,03h
MOV DX,08h
INT 33h
```

4. Monitor và card giao tiếp đồ hoạ

4.1. Nguyên lý hiện ảnh trên monitor

Phương pháp hiện ảnh trên màn hình của monitor máy tính cũng giống như trong máy thu hình thông thường. Hình bên dưới minh họa việc hiện ảnh trên màn hình kiểu ống phóng tia âm cực CRT (cathode ray tube).



Hình 5.12 – Cấu tạo ống hình CRT

Các điện tử phát xạ từ cathode trong ống được hội tụ thành 1 chùm tia, sau đó được tăng tốc và được làm lệch hướng chuyển động bởi các bộ phận lái tia. Tia này sẽ đập vào màn hình có phủ chất huỳnh quang để tạo thành 1 điểm sáng gọi là 1 điểm ảnh.

Do hiện tượng lưu ảnh trong võng mạc của mắt người nên khi tia điện tử được quét rất nhanh theo chiều ngang từ trái sang phải sẽ tạo nên 1 vệt sáng ngang được gọi là dòng quét. Đến cuối 1 dòng, nó được quét ngược trở về bên trái để quét tiếp dòng thứ 2 bên dưới ..v..v.. Quá trình quét các dòng được dịch dần từ trên xuống dưới cho suốt chiều dọc của màn hình được gọi là quét dọc.

Độ chói (sáng tối) được quyết định bởi cường độ chùm tia đập vào màn hình huỳnh quang và 1 điểm màu tự nhiên được hiện nhờ sự trộn lẫn của 3 màu: đỏ, xanh dương, xanh lá cây theo 1 tỉ lệ nào đó. Ba màu này được hiện nhờ 3 tia điện tử cùng bắn vào 3 điểm trên màn hình kề cận nhau, mỗi điểm được phủ chất huỳnh quang phát ra các màu tương ứng. 3 chùm tia điện tử đó được phát ra bởi 3 súng điện tử là 3 cathode được xếp đặt bên trong CRT một cách cẩn thận. Có 2 kiểu quét tia điện tử:

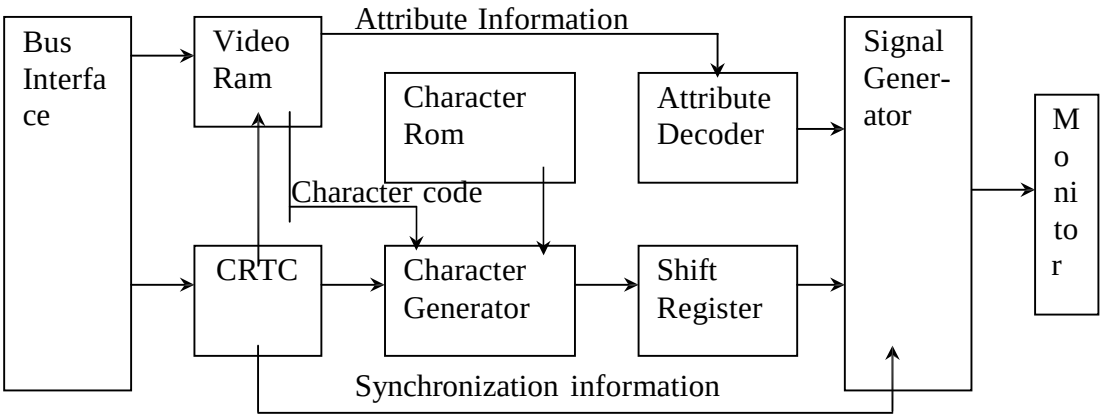
- Quét xen kẽ (interlaced): các dòng lẻ được quét trước cho đến hết màn hình theo chiều dọc, gọi là mành lẻ; sau đó các dòng chẵn tạo nên mành chẵn được quét sau. Phương pháp này có ưu điểm là thu hẹp được dải tần số làm việc của thiết bị nhưng có nhược điểm là hình ảnh bị nhấp nháy.

- Quét không xen kẽ (non-interlaced): các dòng quét được thực hiện tuần tự. Ưu điểm là hình ảnh được điều chỉnh chính xác và ổn định nhưng thiết kế mạch điện sẽ khó hơn vì phải giải quyết vấn đề tăng dải tần làm việc.

Hiện nay còn có các monitor dùng màn hình tinh thể lỏng LCD hoặc ống chứa khí được hoạt động theo nguyên lý tương tự như trên nhưng không có tia điện tử quét nên thay vì các điểm ảnh riêng biệt là các phần tử phát sáng được định địa chỉ một cách tuần tự. Do vậy, trên các monitor này hình ảnh cũng được phát ra từng dòng một. Quá trình quét ngược cũng không còn nữa vì ở đây đơn giản chỉ việc thay đổi địa chỉ về phần tử đầu dòng tiếp theo.

4.2. Card giao tiếp đồ họa

Để hiện các hình ảnh, ký tự, hay hình vẽ trên màn hình, PC phải thông qua mạch ghép nối màn hình (graphics adapter). Board mạch này thường được cắm trên khe cắm mở rộng của PC. Sơ đồ khối như hình sau:



Hình 5.13 - Sơ đồ khối của bản mạch ghép nối màn hình

Bus Interface: ghép nối bus;
generator: máy phát tín hiệu;
Attribute information: thông tin thuộc tính;
Attribute decoder: bộ giải mã thuộc tính;
Character generator: máy phát ký tự;
Video Ram: Ram Video Signal
Character code: mã ký tự
Character rom: rom ký tự
Shift register: thanh ghi dịch
Synchronization information: thông tin đồng bộ.

Phần trung tâm là chip điều khiển ống hình CRTC (cathode ray tube controller). CPU thâm nhập RAM Video qua mạch ghép nối bus để ghi thông tin xác định ký tự hay hình vẽ cần hiển thị. CRTC liên tục phát ra các địa chỉ để Ram video đọc các ký tự trong đó và truyền chúng tới máy phát ký tự (character generator).

Trong chế độ văn bản (text mode), các ký tự được xác định bởi mã ASCII, trong đó có cả các thông tin về thuộc tính của ký tự, thí dụ ký tự được hiện theo cách nhấp nháy hay đảo màu đen trắng ….ROM ký tự (character rom) lưu trữ các hình mẫu điểm ảnh của các ký tự tương ứng để máy phát ký tự biến đổi các mã ký tự đó thành 1 chuỗi các bit điểm ảnh (pixel bit) và chuyển chúng tới thanh ghi dịch (shift register). Máy phát tín hiệu sẽ sử dụng các bít điểm ảnh này cùng với các thông tin thuộc tính từ

Ram video và các tín hiệu đồng bộ từ CRTC để phát ra các tín hiệu cần thiết cho monitor.

Trong chế độ đồ họa (graphics mode), thông tin trong RAM video được sử dụng trực tiếp cho việc phát ra các ký tự. Lúc này các thông tin về thuộc tính cũng không cần nữa. Chỉ từ các giá trị bit trong thanh ghi dịch, máy phát tín hiệu sẽ phát các tín hiệu về độ sáng và màu cho monitor.

4.2.1. Máy phát ký tự trong các chế độ văn bản và đồ họa:

Mỗi ký tự được biểu diễn bởi 1 từ 2 byte trong RAM video. Byte thấp chứa mã ký tự, byte cao chứa thuộc tính. Cấu trúc của một từ nhớ video như sau:

15	14	13	12	11	10	9	8
BLNK	BAK_2	BAK_1	BAK_0	INT	FOR_2	FOR_1	FOR_0

7	6	5	4	3	2	1	0
CHR_7	CHR_6	CHR_5	CHR_4	CHR_3	CHR_2	CHR_1	CHR_0

BLNK:	Nhấp nháy; 1 = bật, 0 = tắt
BAK_2 … BAK_0:	Màu nền; (từ bảng màu hiện tại)
INT:	Cường độ sáng ; 1 = cao, 0 = bình thường
FOR_2 … FOR_0:	Màu nền trước (từ bảng màu hiện tại)
CHR_7…$CHR_{0:}$	Mã ký tự.

Trong chế độ văn bản, 6845 liên tục xuất các địa chỉ cho RAM video qua MA0-MA13. Ký tự ở góc tận cùng phía trên bên trái màn hình có địa chỉ thấp nhất mà 6845 sẽ cung cấp ngay sau khi quét dọc ngược. Logic ghép nối định địa chỉ cho RAM video bằng việc lấy ra mã ký tự cùng với thuộc tính. Mã ký tự dùng cho máy phát ký tự như là chỉ số thứ nhất trong ROM ký tự. Lúc này, 6845 định địa chỉ hàng quét đầu tiên của ma trận ký tự, địa chỉ hàng bằng 0. Các bit của ma trận điểm ảnh bây giờ sẽ được truyền đồng bộ với tần số video từ thanh ghi dịch tới máy phát tín hiệu. Nếu máy phát tín hiệu nhận được giá trị 1 từ thanh ghi dịch, nó sẽ phát tín hiệu video tương ứng với màu của ký tự. Nếu nhận được 0 nó sẽ cấp tín hiệu tương ứng với màu nền. Vậy dòng quét thứ nhất được hiện phù hợp với các ma trận điểm ảnh của các ký tự trong hàng ký tự thứ nhất. Khi tia điện tử đạt tới cuối dòng quét, 6845 kích hoạt lối ra HS để tạo ra quá trình quét ngược và đồng bộ ngang. Tia điện tử quay trở về bắt đầu quét dòng tiếp. Sau mỗi dòng quét, 6845 tăng giá trị RA0-RA4 lên 1. Địa chỉ dòng này hình thành một giá trị offset bên trong ma trận điểm ảnh cho ký tự được hiện. Dựa trên mỗi dòng quét như vậy, một dòng các điểm ảnh của ký tự trong hàng ký tự được hiện ra. Điều này có nghĩa là với ma trận 9x14 điểm ảnh cho 1 ký tự, hàng ký tự thứ nhất đã được hiện sau 14 dòng quét. Khi địa chỉ RA0-RA4 trở về giá trị 0, 6845 sẽ cấp 1 địa chỉ MA0-MA13 mới và hàng ký tự thứ hai sẽ được hiện ra cũng như vậy. Ở cuối dòng quét cuối cùng, 6845 sẽ reset địa chỉ MA0-MA13 và RA0-RA4 và cho phép lối ra VS phát ra tín hiệu quét ngược cùng tín hiệu đồng bộ dọc.

Mỗi ký tự có chiều cao cực đại ứng với 32 dòng vì có 5 đường địa chỉ RA0-RA4, còn bộ nhớ video trong trường hợp này được tới 16K từ vì có địa chỉ MA0-MA13 là 14 bit. Trong chế độ đồ họa, chúng kết hợp với nhau để tạo thành địa chỉ 19 bit, lúc đó 6845 có thể định địa chỉ cho bộ nhớ video lên tới 512k từ. Trong trường hợp

này, các byte trong RAM video không được dịch thành mã ký tự và thuộc tính nữa mà trực tiếp xác định cường độ sáng và màu của điểm ảnh. Đa số các RAM video được chia thành vài băng được định địa chỉ bởi RA0-RA4. Các đường MA0-MA13 sẽ định địa chỉ offset bên trong mỗi băng. Số liệu trong RAM video lúc này được trực tiếp truyền tới thanh ghi dịch và máy phát tín hiệu. ROM ký tự và máy phát ký tự không làm việc.

4.2.2. Tổ chức của RAM video

RAM video được tổ chức khác nhau tuỳ theo chế độ hoạt động và bản mạch ghép nối. Thí dụ, với RAM video 128 KB, có thể địa chỉ hóa toàn bộ bộ nhớ màn hình qua CPU như bộ nhớ chính. Nhưng nếu kích thước RAM video lớn hơn thì làm như vậy sẽ đè lên vùng ROM mở rộng ở địa chỉ C0000h. Do đó, card EGA và VGA với trên 128 KB nhớ được tăng cường thêm 1 chuyển mạch mềm (soft-switch) cho phép thâm nhập các cửa sổ 128 KB khác nhau vào RAM video lớn hơn nhiều. Các chuyển mạch này được quy định bởi riêng các nhà sản xuất board mạch.

4.2.2.1. Tổ chức trong chế độ văn bản

RAM video được coi như một dãy từ tuyến tính, từ đầu tiên được gán cho ký tự góc trên tận cùng bên trái màn hình gọi là hàng 1 cột 1. Từ thứ 2 là hàng 1, cột 2, …. Số từ tuỳ thuộc vào độ phân giải của kiểu hiện ký tự.

Thí dụ: độ phân giải chuẩn 25 hàng, 80 ký tự đòi hỏi 2000 từ nhớ 2 byte. Như vậy, tổng cộng cần 4 KB bộ nhớ RAM video. Trong khi đó với card có độ phân giải cao SVGA 60 hàng, 132 ký tự cần đến 15840 byte. Do đó RAM video thường được chia thành vài trang. Kích thước của mỗi trang tuỳ thuộc vào chế độ hiện của màn hình và số trang cực đại, phụ thuộc cả vào kích thước của RAM video. 6845 có thể được chương trình hóa sao cho địa chỉ khởi phát của MA0-MA13 sau quét ngược dọc là khác 00h. Nếu địa chỉ khởi phát là bắt đầu của 1 trang thì có thể quản lý RAM video theo vài trang tách biệt nhau, nếu CPU thay đổi nội dung của 1 trang mà trang đó hiện đang không hiện thì màn hình cũng không thay đổi. Do đó, cần phân biệt trang nhớ đang được kích hoạt (đang hiện) và trang đang được xử lý.

Đoạn chương trình ghi ký tự 'A' có cường độ sáng cao vào góc trên bên trái với màu số 7 và màu nền số 0. Trang thứ nhất và là duy nhất bắt đầu ở địa chỉ B0000h.

```
MOV AX, 0B000h;     nạp thanh ghi ax với địa chỉ đoạn của Ram video
MOV ES, AX;         truyền địa chỉ đoạn vào ES
MOV AH, 0F8h;       nạp byte thuộc tính 1111 1000 vào AH
MOV AL, 41h;        nạp mã ký tự của 'A' vào AL
MOV ES:[00H],AX;    ghi byte thuộc tính và mã ký tự vào RAM video.
```

4.2.2.2. Tổ chức trong chế độ đồ họa:

Tổ chức trong chế độ này phức tạp hơn. Ví dụ: với bản mạch Hercules, RAM video được chia thành 4 băng trên 1 trang . Băng thứ nhất: đảm bảo các điểm ảnh cho các dòng 0, 4, 8, …, 344; băng thứ hai cho các dòng 1, 5, 9, …, 345; băng thứ 3 cho các dòng 2, 6, 10, …., 346; và băng thứ 4 cho các dòng 3, 7, 11, …, 347. 64 KB được chia thành 2 trang 32 KB. Độ phân giải trong chế độ đồ họa là 720 x 348 điểm ảnh, mỗi điểm ảnh được biểu diễn bởi 1 bit. Do vậy, một dòng cần 90 byte (720 điểm ảnh /

8 điểm ảnh trên 1 byte). Địa chỉ của byte chứa điểm ảnh thuộc đường i và cột j trong trang k là:

B0000h+8000h*k+2000h*(i mod 4)+ 90*int(i/4)+int(j/8)

B0000h là đoạn video, 8000h là kích thước của trang, 2000h*(i mod 4) là offset của băng chứa byte đó, 90*int(i/4) là offset của dòng i trong băng và int(j/8) là offset của cột j trong băng.

Trong bản mạch CGA bộ nhớ video được chia thành 2 băng còn với EGA và VGA thì phức tạp hơn.

4.2.3. Truy xuất màn hình qua DOS và BIOS

4.2.3.1. Truy xuất qua DOS

Các hàm của int 21h có thể hiện các ký tự trên màn hình nhưng không can thiệp được vào màu:

- Hàm 02h: ra màn hình.
- Hàm 06h: ra một ký tự.
- Hàm 09h: ra một chuỗi.
- Hàm 40h: ghi file/ thiết bị

Từ DOS 4.0 trở đi có thể dùng lệnh **mode** để điều chỉnh số cột văn bản từ 40 đến 80 hay số dòng từ 25 đến 50.

Các lệnh **copy, type** và **print** trong command.com cho phép hiện text trên màn hình. DOS gộp chung bàn phím và monitor thành 1 thiết bị mang tên CON (console). Ghi CON là truyền số liệu tới monitor, còn đọc CON là nhận ký tự từ bàn phím. Ví dụ: để hiện nội dung của file output.txt lên màn hình của monitor sẽ có các cách sau:

- copy output.txt con
- type output.txt > con
- print output.txt /D:con

4.2.3.2. Truy xuất qua BIOS

Bios thâm nhập monitor bằng int 10h với nhiều chức năng hơn DOS, như đặt chế độ hiện hình, quản lý tự động các trang, phân biệt các điểm trên màn hình nhờ các tọa độ,…

Những thường trình đồ họa:

BIOS trên main board có sẵn những hàm dùng cho thâm nhập MDA và CGA. BIOS của riêng EGA và VGA có những hàm mở rộng tương ứng trong khi vẫn giữ nguyên định dạng gọi.

Một trong những hàm quan trọng nhất của int 10h là hàm 00h dùng để đặt chế độ hiện hình. Để thay đổi chế độ hiện hình cần phải làm rất nhiều bước chương trình phức tạp để nạp các thanh ghi của chip 6845. Trong khi đó, hàm 00h làm cho ta tất cả các công việc này.

Thí dụ: tạo kiểu 6 với độ phân giải 640*200 trên CGA.

```
Mov ah, 00h        ; hàm 00h
Mov al, 06h        ; chế độ 6
Int 10h            ; gọi ngắt
```

Các board EGA/VGA có riêng BIOS của chúng. Trong quá trình khởi động PC, nó sẽ chặn int 10h lại và chạy chương trình BIOS của riêng board mạch. Thường trình cũ (của BIOS trên board mach chính) được thay địa chỉ tới int 42h. Tất cả các lệnh gọi int 10h sẽ được BIOS của EGA/VGA thay địa chỉ tới int 42h nếu board mạch EGA/VGA đang chạy các kiểu hiện tương thích với MDA hay CGA. Có các kiểu hoạt động từ 0 đến 7.

BIOS của EGA/VGA dùng vùng 40:84h tới 40:88h để lưu số liệu BIOS và các thông số của EGA/VGA. Nó có các hàm mới với các hàm phụ sau:

- Hàm 10h: truy xuất các thanh ghi màu và bảng màu
- Hàm 11h: cài đặt các bảng định nghĩa ký tự mới
- Hàm 12h: đặt cấu hình hệ con video
- Hàm 1Bh: thông tin về trạng thái và chức năng của BIOS video (chỉ có ở VGA)
- Hàm 1Ch: trạng thái save/restore của video (chỉ có ở VGA)

Sau đây là chức năng của các hàm và thí dụ sử dụng chúng:

- Hàm 10h, hàm phụ 03h – xoá/đặt thuộc tính

Ví dụ: Xoá thuộc tính nhấp nháy:

```
Mov ah, 10h     ; dùng hàm 10h
Mov al, 03h     ; dùng hàm phụ 03h
Mov bl, 00h     ; xoá thuộc tính nhấp nháy
Int 10h         ; gọi ngắt
```

- Hàm 11h – ghép nối với máy phát ký tự

Ví dụ: Nạp bảng định nghĩa ký tự 8*14 không cần chương trình CRTC:

```
Mov ah, 11h     ; dùng hàm 11h
Mov al, 01h     ; nạp bảng ký tự từ Rom Bios vào Ram máy phát ký tự.
Mov bl, 03h     ; gán số 3 cho bảng
Int 10h         ; gọi ngắt
```

- Hàm 12h, hàm phụ 20h – chọn thường trình in màn hình. Dùng hàm phụ này có thể thay thế thường trình chuẩn cho INT 05h bằng thường trình có thể dùng cho các độ phân giải mới của EGA/VGA.

Ví dụ: Cho phép thường trình mới in màn hình:

```
Mov ah, 12h     ; dùng hàm 12h
Mov bl, 20h     ; dùng hàm phụ 20h
```

Ấn PRINT hoặc SHIFT+PRINT để gọi thường trình in đã được lắp đặt.

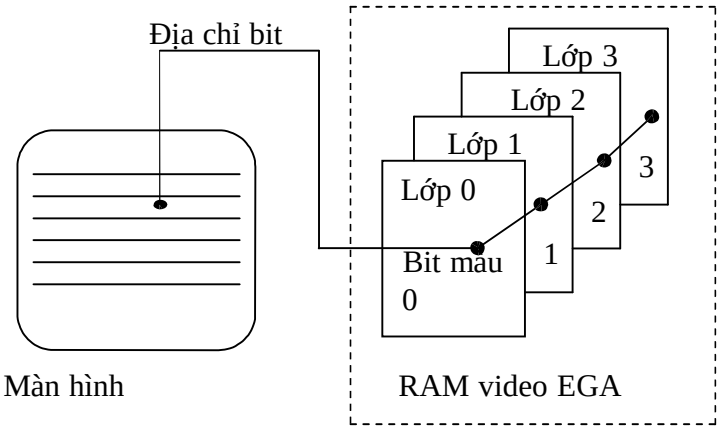
Truy xuất trực tiếp bộ nhớ video:

Để vẽ 1 điểm trên màn hình, BIOS phải làm nhiều thủ tục nhưng nếu muốn vẽ toàn bộ 1 cửa sổ hình hay lư trữ thì phải truy xuất trực tiếp RAM video.

- Với board đơn sắc MDA trong kiểu hiện văn bản số 7, 4 KB RAM được tổ chức như 1 dãy (array) gồm 2000 từ nhớ kề nhau (mỗi từ là mã thuộc tính: ký tự) tạo nên 25 dòng, 80 cột. RAM video bắt đầu ở đọan B0000h, trong đó ký tự góc trên cùng bên trái là từ thứ nhất trong RAM video. Như vậy mỗi dòng có 160 byte (A0h). Địa chỉ của từ nhớ ứng với ký tự ở dòng i, cột j (i = 0-24, j = 0-79) được tính theo công thức sau:

  Address (i,j) = B0000h +A0h*i +02h*j.

- Với board EGA, ở kiểu hiện văn bản từ 0 đến 3 mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video. Mạch logic chuyển địa chỉ trên board thực hiện sự kết hợp nhất định nào đó sao cho tổ chức và cấu trúc của RAM video cũng như cách tính địa chỉ vẫn tương đồng với cách của CPU. Trong chế độ đồ họa từ 13 đến 16, RAM video bắt đầu từ địa chỉ đoạn A000h. Các điểm ảnh được xếp kề cận nhau trong bộ nhớ và mỗi điểm ảnh đòi hỏi 4 bit, các bit này được phân ra ở 4 lớp nhớ. Như vậy địa chỉ của 1 trong 4 bit này trên 1 điểm ảnh không chỉ gồm đoạn video và offset mà còn thêm vào số lớp nhớ nữa.



Hình 5.14 - Các lớp nhỏ của RAM Video

Để hiện 1 điểm ảnh với 1 trong 16 màu, không phải chỉ tính địa chỉ bit mà còn phải thâm nhập 4 lớp nhớ. Muốn vậy, phải dùng thanh ghi mặt nạ bản đồ (map mask register). Thanh ghi này được định địa chỉ qua cổng chỉ số 3C4h với địa chỉ 02h và có thể được ghi qua cổng số liệu 3C5h. Cấu trúc của thanh ghi mặt nạ bản đồ như sau:

7	6	5	4	3	2	1	0
Res	Res	Res	Res	LY_3	LY_2	LY_1	LY_0

LY_3-LY_0: Thâm nhập ghi tới các lớp từ 0Æ3;
1 = cho phép; 0 = không cho phép
Res : Dự trữ

Ví dụ: Đặt bit 0 của byte ở địa chỉ A000:0000h cho lớ 0, 1, 3.

```
Mov AX, 0A000h      ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset 0000h vào BX
Out 3C4h, 02h       ; chỉ số 2 Æ thanh ghi mặt nạ bản đồ
Out  3C5h, 0Bh      ; ghi 0000 1011b vào thanh ghi mặt nạ bản đồ
                          (cho phép lớp 0, 1, 3)
Mov 3C5h, 0Bh       ; đặt bit 0 trong các lớp 0, 1 và 3
```

Để lưu trữ nội dung màn hình cần phải đọc các giá trị bit của 4 lớp khi dùng thanh ghi chọn bản đồ đọc (read map select register). Nó được định địa chỉ với chỉ số 04h qua cổng chỉ số 3CEh, và có thể được ghi qua cổng số liệu 3CFh. Cấu trúc của thanh ghi này:

7	6	5	4	3	2	1	0
res	res	res	res	res	res	LY_1	LY_0

LY_1-LY_0: cho phép thâm nhập đọc với:
00 = lớp 0
01 = lớp 1
10 = lớp 2
11 = lớp 3
res : dự trữ

Ví dụ: đọc byte ở địa chỉ A000:0000h cho lớp 2:

```
Mov AX, A000h       ; nạp đọan video vào AX
Mov ES, AX          ; truyền đọan video vào ES
Mov BX, 0000h       ; nạp offset vào BX
Out 3Ceh, 04h       ; chỉ số 4 Æ thanh ghi chọn bản đồ đọc
Out 3CFh, 02h       ; ghi 0000 0010b vào thanh ghi chọn bản đồ đọc
                          (cho phép lớp 2)
Mov  AL, [ES:BX]    ; nạp byte trong lớp 2 vào AL.
```

Chú ý rằng 4 bit tại 4 lớp đại diện cho 1 điểm ảnh nên trong kiểu hiện 16 EGA có độ phân giải cao nhất mỗi dòng cần 80byte (640 điểm ảnh / 8 điểm ảnh trên 1 byte); mỗi trang màn hình gồm 32 KB. Địa chỉ byte của điểm ảnh ở dòng i, cột j trang k (i=0-349, j=0-639, k=0-1) là:

Address (i,j,k) = A0000h + 8000h*k + 50h*j + int (i/8).

Với board VGA, các chế độ hiện văn bản từ 0 đến 3 và 7 cũng như các chế độ đồ họa từ 4 đến 6 và 13 đến 16 của CGA. EGA và MDA đều chạy được trên nó.

Trong chế độ văn bản, mã ký tự được lưu trữ trong lớp nhớ 0 cùng với thuộc tính trong lớp 1 của RAM video VGA. Quá trình chuyển hóa địa chỉ cũng giống như EGA nhưng khác ở chổ nó vẫn đảm bảo chế độ văn bản 7 với độ phân giải 720x400, ma trận điểm ảnh 9x16. Trong chế độ đồ họa $4 \div 6$ và $13 \div 19$, mọi tổ chức, cấu trúc cũng như cách tính địa chỉ tương tự như CGA và EGA. VGA được tăng cường 3 kiểu hiện hình mới từ 17 đến 19.

Kiểu 17 tương thích với board đồ họa của máy PS/2 kiểu 30 là MCGA (multi colour graphics array). Các điểm ảnh chỉ gồm 1 bit (2 màu) được định vị chỉ trên lớp 0. Thí dụ, trong VGA kiểu 17 với 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte). Mỗi trang màn hình gồm 40 KB. Địa chỉ của byte ở dòng i, cột j (i= 0-479), j=0-639) như sau:

Address (i,j) = A0000h+50h*j+int (i/8)

Kiểu 18, 4 bit của điểm ảnh được phân trong 4 lớp nhớ như ở EGA. Trong kiểu VGA phân giải cao với 16 màu khác nhau, 80 byte trên 1 dòng (640 điểm ảnh / 8 điểm ảnh trên 1 byte), mỗi trang màn hình gồm 40 KB (A0000h byte); địa chỉ của mỗi byte ở dòng i, cột j (i=0-479; j = 0-639) bằng:

Address (i,j) = A0000h + 50h*j + int (i/8).

Kiểu 19 với 256 màu cho 1 điểm ảnh thì RAM video lại được tổ chức rất đơn giản như 1 dãy tuyến tính, trong đó 1 byte tương ứng với 1 điểm ảnh. Giá trị của byte phân định màu của điểm ảnh. Kiểu này đòi hỏi 320 byte (140h) trên 1 dòng (320 điểm ảnh / 1 điểm ảnh trên 1 byte). Một trang màn hình gồm 64 KB (10000h) nhưng chỉ có 64000 byte được sử dụng. Địa chỉ của điểm ảnh trong dòng i, cột j (i = 0-199, j=0-319) là:

Address (i,j) = A0000h + 140h*j + i

4.2.4. Bus cục bộ và chip xử lý đồ họa

Để tăng tốc độ hiện đồ họa có 2 giải pháp:

- Dùng bus cục bộ 32 bit để tránh hiện tượng nghẽn cổ chai (bottleneck) do bus ISA chỉ có 16 bit và tốc độ hạn chế (8.33Mhz); điều này cho phép 1 lượng thông tin nhiều hơn được trao đổi giữa CPU và board mạch trong 1 đơn vị thời gian.
- Dùng chip xử lý đồ họa với BIOS riêng trên board mạch điều khiển monitor. Chip này sẽ làm hầu hết các công việc trừ một ít lệnh và thông số mô tả nội dung phần màn hình cần hiện là được cấp từ CPU. Thí dụ cần vẽ 1 hình chữ nhật với màu nào đó, board chỉ cần vài thông số ban đầu từ CPU như tọa độ của 2 góc và giá tri màu là đủ. Cách giải quyết như vậy rất có lợi khi PC chạy trong chế độ đa nhiệm.

2. Giao tiếp song song

2.1. Cấu trúc cổng song song

Cổng song song gồm có 4 đường điều khiển, 5 đường trạng thái và 8 đường dữ liệu bao gồm 5 chế độ hoạt động:

- Chế độ tương thích (compatibility).
- Chế độ nibble.
- Chế độ byte.
- Chế độ EPP (Enhanced Parallel Port).
- Chế độ ECP (Extended Capabilities Port).

3 chế độ đầu tiên sử dụng port song song chuẩn (SPP – Standard Parallel Port) trong khi đó chế độ 4, 5 cần thêm phần cứng để cho phép hoạt động ở tốc độ cao hơn. Sơ đồ chân của máy in như sau:

Chân	Tín hiệu	Mô tả
1	$\overline{\text{STR}}$ (Out)	Mức tín hiệu thấp, truyền dữ liệu tới máy in
2	D0	Bit dữ liệu 0
3	D1	Bit dữ liệu 1
4	D2	Bit dữ liệu 2
5	D3	Bit dữ liệu 3
6	D4	Bit dữ liệu 4
7	D5	Bit dữ liệu 5
8	D6	Bit dữ liệu 6
9	D7	Bit dữ liệu 7
10	$\overline{\text{ACK}}$ (In)	Mức thấp: máy in đã nhận 1 ký tự và có khả năng nhận nữa
11	BUSY (In)	Mức cao: ký tự đã được nhận; bộ đệm máy in đầy; khởi động máy in; máy in ở trạng thái off-line.
12	PAPER EMPTY (In)	Mức cao: hết giấy
13	SELECT (In)	Mức cao: máy in ở trạng thái online
14	$\overline{\text{AUTOFEED}}$ (Out)	Tự động xuống dòng; mức thấp: máy in xuống dòng tự động
15	$\overline{\text{ERROR}}$ (In)	Mức thấp: hết giấy; máy in ở offline; lỗi máy in
16	$\overline{\text{INIT}}$ (Out)	Mức thấp: khởi động máy in
17	$\overline{\text{SELECTIN}}$ (Out)	Mức thấp: chọn máy in
18-25	GROUND	0V

Cổng song song có ba thanh ghi có thể truyền dữ liệu và điều khiển máy in. Địa chỉ cơ sở của các thanh ghi cho tất cả cổng LPT (line printer) từ LPT1 đến LPT4 được lưu trữ trong vùng dữ liệu của BIOS. Thanh ghi dữ liệu được định vị ở offset 00h, thanh ghi trang thái ở 01h, và thanh ghi điều khiển ở 02h. Thông thường, địa chỉ

cơ sở của LPT1 là 378h, LPT2 là 278h, do đó địa chỉ của thanh ghi trạng thái là 379h hoặc 279h và địa chỉ thanh ghi điều khiển là 37Ah hoặc 27Ah. Tuy nhiên trong một số trường hợp, địa chỉ của cổng song song có thể khác do quá trình khởi động của BIOS. BIOS sẽ lưu trữ các địa chỉ này như sau:

Địa chỉ	**Chức năng**
0000h:0408h	Địa chỉ cơ sở của LPT1
0000h:040Ah	Địa chỉ cơ sở của LPT2
0000h:040Ch	Địa chỉ cơ sở của LPT3

Định dạng các thanh ghi như sau:

Thanh ghi dữ liệu (hai chiều):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	D7	D6	D5	D4	D3	D2	D1	D0
Chân số	9	8	7	6	5	4	3	2

Thanh ghi trạng thái máy in (chỉ đọc):

	7	6	5	4	3	2	1	0
Tín hiệu máy in	BUSY	$\overline{\text{ACK}}$	PAPER EMPTY	SELECT	$\overline{\text{ERROR}}$	$\overline{\text{IRQ}}$	x	x
Số chân cắm	11	10	12	13	15	-	-	-

Thanh ghi điều khiển máy in:

	7	6	5	4	3	2	1	0
Tín hiệu máy in	x	x	DIR	IRQ Enable	$\overline{\text{SELECTIN}}$	$\overline{\text{INIT}}$	$\overline{\text{AUTOFEED}}$	$\overline{\text{STROBE}}$
Số chân cắm	-	-	-	-	17	16	14	1

x: không sử dụng

IRQ Enable: yêu cầu ngắt cứng; 1 = cho phép; 0 = không cho phép

Chú ý rằng chân BUSY được nối với cổng đảo trước khi đưa vào thanh ghi trạng thái, các bit $\overline{\text{SELECTIN}}$, $\overline{\text{AUTOFEED}}$ và $\overline{\text{STROBE}}$ được đưa qua cổng đảo trước khi đưa ra các chân của cổng máy in.

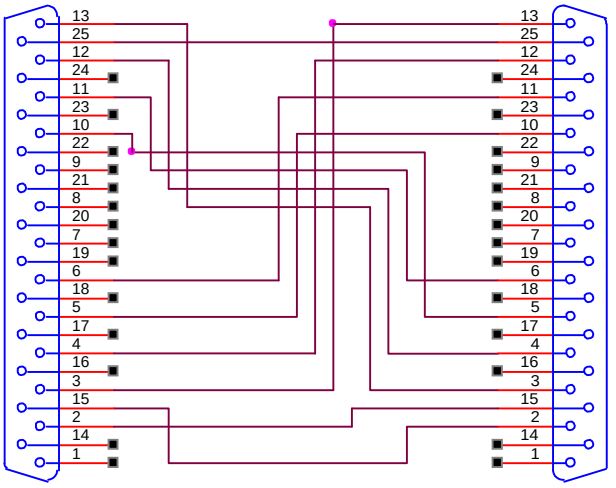
Thông thường tốc độ xử lý dữ liệu của các thiết bị ngoại vi như máy in chậm hơn PC nhiều nên các đường $\overline{\text{ACK}}$, BUSY và $\overline{\text{STR}}$ được sử dụng cho kỹ thuật bắt tay. Khởi đầu, PC đặt dữ liệu lên bus sau đó kích hoạt đường $\overline{\text{STR}}$ xuống mức thấp để thông tin cho máy in biết rằng dữ liệu đã ổn định trên bus. Khi máy in xử lý xong dữ

liệu, nó sẽ trả lại tín hiệu $\overline{ACK}$ xuống mức thấp để ghi nhận. PC đợi cho đến khi đường BUSY từ máy in xuống thấp (máy in không bận) thì sẽ đưa tiếp dữ liệu lên bus.

2.2. Giao tiếp với thiết bị ngoại vi

2.2.1. Giao tiếp với máy tính

Quá trình giao tiếp với cổng song song dùng 2 chế độ: chế độ chuẩn SPP và chế độ mở rộng. Việc giao tiếp ở chế độ chuẩn mô tả như sau:

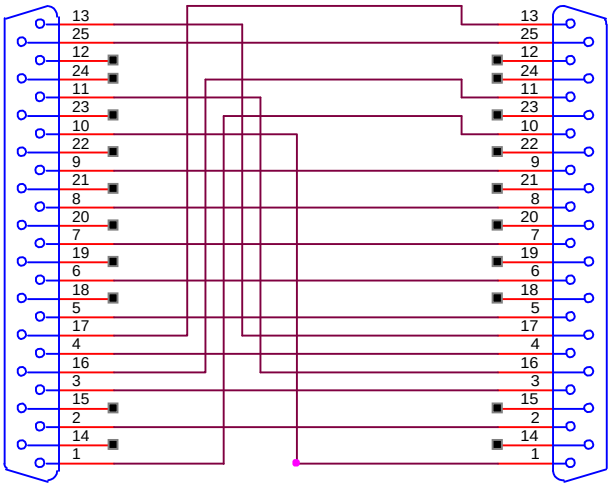


Hình 5.15 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ chuẩn

Sơ đồ chân kết nối mô tả như sau:

PC1		**PC2**	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	15	$\overline{\text{ERROR}}$
D1	3	13	SELECT
D2	4	12	PAPER EMPTY
D3	5	10	$\overline{\text{ACK}}$
D4	6	11	BUSY
BUSY	11	6	D4
$\overline{\text{ACK}}$	10	5	D3
PAPER EMPTY	12	4	D2
SELECT	13	3	D1
$\overline{\text{ERROR}}$	15	2	D0
GND	25	25	GND

Ngoài ra, việc kết nối giữa 2 máy tính sử dụng cổng song song có thể dùng chế độ mở rộng, chế độ này cho phép giao tiếp với tốc độ cao hơn.



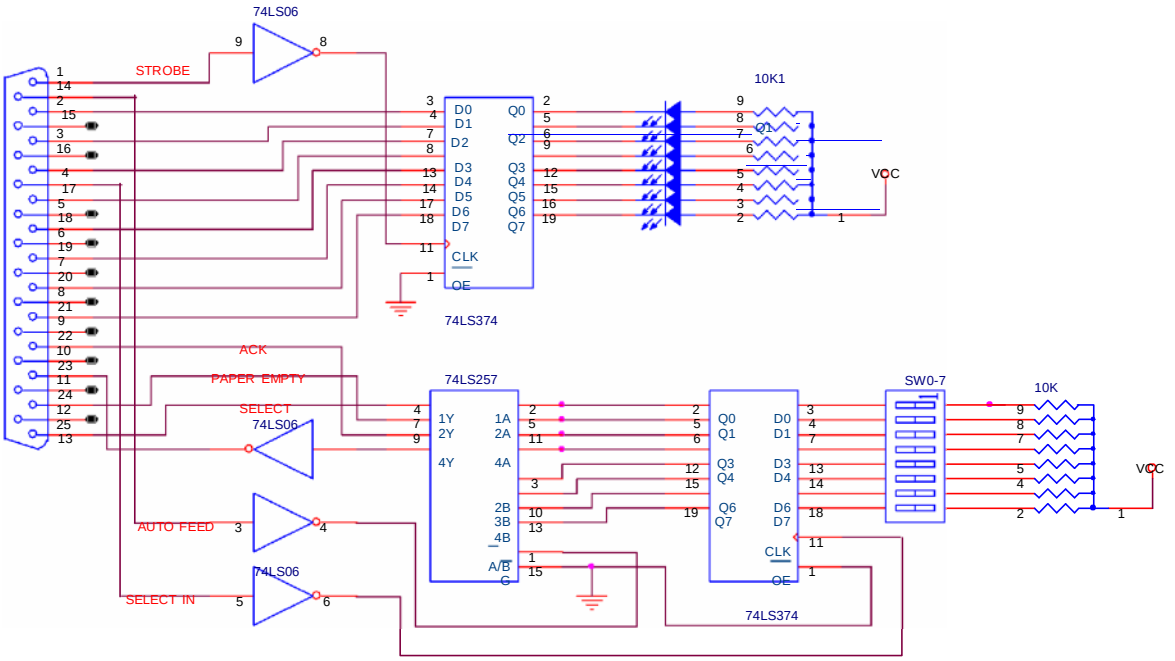
Hình 5.16 - Trao đổi dữ liệu qua cổng song song giữa 2 PC dùng chế độ mở rộng

Sơ đồ chân kết nối mô tả như sau:

PC1		**PC2**	
Chức năng	**Chân**	**Chân**	**Chức năng**
D0	2	2	D0
D1	3	3	D1
D2	4	4	D2
D3	5	5	D3
D4	6	6	D4
D5	7	7	D5
D6	8	8	D6
D7	9	9	D7
SELECT	13	17	$\overline{\text{SELECTIN}}$
BUSY	11	16	$\overline{\text{INIT}}$
$\overline{\text{ACK}}$	10	1	$\overline{\text{STROBE}}$
$\overline{\text{SELECTIN}}$	17	13	SELECT
$\overline{\text{INIT}}$	16	11	BUSY
$\overline{\text{STROBE}}$	1	10	$\overline{\text{ACK}}$

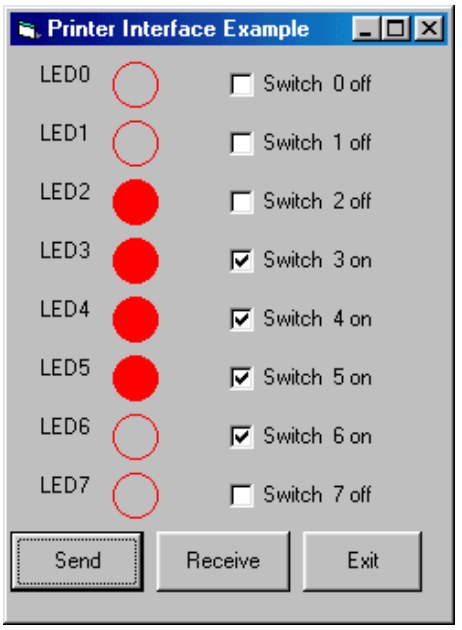
2.2.2. Giao tiếp thiết bị khác

Quá trình giao tiếp với các thiết bị ngoại vi có thể thực hiện thông qua chế độ chuẩn. Để đọc dữ liệu, có thể dùng một IC ghép kênh 2Æ1 74LS257 và dùng 4 bit trạng thái của cổng song song còn xuất dữ liệu thì sử dụng 8 đường dữ liệu D0 – D7.



Hình 5.17 – Mạch giao tiếp đơn giản thông qua cổng máy in

Giao diện:



Hình 5.18 – Giao diện của chương trình giao tiếp với cổng máy in

Chương trình giao tiếp trên VB sử dụng thư viện liên kết động để trao đổi dữ liệu với cổng máy in. Thư viện IO.DLL bao gồm các hàm sau:

- Hàm PortOut: xuất 1 byte ra cổng

```
Private Declare Sub PortOut Lib "IO.DLL" (ByVal
Port As Integer, ByVal Data As Byte)
```

Port: địa chỉ cổng, Data: dữ liệu xuất

- Hàm PortWordOut: xuất 1 word ra cổng

```
    Private Declare Sub PortWordOut Lib "IO.DLL"
(ByVal Port As Integer, ByVal Data As Integer)
```

- Hàm PortDWordOut: xuất 1 double word ra cổng

```
    Private Declare Sub PortDWordOut Lib "IO.DLL"
(ByVal Port As Integer, ByVal Data As Long)
```

- Hàm PortIn: nhập 1 byte từ cổng, trả về giá trị nhập

```
    Private Declare Function PortIn Lib "IO.DLL"
(ByVal Port As Integer) As Byte
```

- Hàm PortWordIn: nhập 1 word từ cổng

```
    Private Declare Function PortWordIn Lib "IO.DLL"
(ByVal Port As Integer) As Integer
```

- Hàm PortDWordIn: nhập 1 double word từ cổng

```
    Private Declare Function PortDWordIn Lib
"IO.DLL" (ByVal Port As Integer) As Long
```