BỘ LAO ĐỘNG THƯƠNG BINH VÀ XÃ HỘI
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG

GIÁO TRÌNH

CẤU TRÚC MÁY TÍNH

(COMPUTER STRUCTURE)

BIÊN SOẠN
Nguyễn Duy Phúc

Vĩnh long – 10/2014

BỘ LAO ĐỘNG THƯƠNG BINH VÀ XÃ HỘI

TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT VĨNH LONG

GIÁO TRÌNH

CẤU TRÚC MÁY TÍNH

(COMPUTER STRUCTURE)

Mã số học phần: TH1205

Số tín chỉ: 3

BIÊN SOẠN

Nguyễn Duy Phúc

Vĩnh long – 10/2014

LỜI NÓI ĐẦU

Để đáp ứng nhu cầu học tập và nghiên cứu của các bạn sinh viên, đặc biệt là sinh viên chuyên ngành Công nghệ thông tin, Khoa Công nghệ thông tin - Trường Đại học Sư phạm Kỹ thuật Vĩnh Long đã tiến hành biên soạn các giáo trình, bài giảng chính trong chương trình học. Giáo trình môn Cấu trúc máy tính được biên soạn dựa trên:

1. "Giáo trình cấu trúc máy tính" của các tác giả Tống Văn On, Hoàng Đức Hải do nhà xuất bản Lao động xã hội phát hành.
2. "Giáo trình Cấu trúc máy tính" của các tác giả Võ Văn Chín, Nguyễn Hồng Vân, Phạm Hữu Tài – Trường đại học Cần Thơ
3. "Giáo trình lập trình hợp ngữ" của nhà xuất bản Thống kê

Giáo trình này cũng được biên soạn dựa trên kinh nghiệm giảng dạy nhiều năm môn Cấu trúc máy tính của các giáo viên trong khoa chúng tôi. Ngoài ra chúng tôi cũng đã tham khảo rất nhiều tài liệu của các trường đại học trong và ngoài nước.

Cấu trúc tài liệu này được biên soạn dựa theo đề cương chi tiết môn học Cấu trúc máy tính của Khoa Công nghệ thông tin Trường Đại học Sư phạm Kỹ thuật Vĩnh Long dùng cho sinh viên chuyên ngành Công nghệ thông tin. Nội dung gồm 8 chương:

- Chương 1: Tổng quan
- Chương 2: Kiến trúc phần mềm bộ xử lý
- Chương 3: Tổ chức bộ xử lý
- Chương 4: Các cấp bộ nhớ
- Chương 5: Thiết bị nhập xuất
- Chương 6: Tổng quan về lập trình hợp ngữ
- Chương 7: Tổ chức bộ xử lý Intel 8086/8088
- Chương 8: Lập trình Turbo Assembler

Mục tiêu của giáo trình nhằm giúp các bạn sinh viên chuyên ngành có một tài liệu cô đọng dùng để học tập và nghiên cứu. Ngoài ra, chúng tôi nghĩ rằng các bạn sinh viên thuộc các chuyên ngành khác và những người quan tâm, tìm hiểu vệ hoạt động của hệ thống, lập trình hệ thống máy tính cũng sẽ tìm được những điều bổ ích trong quyển giáo trình này.

Mặc dù chúng tôi đã rất cố gắng trong quá trình biên soạn nhưng chắc chắn giáo trình này sẽ còn nhiều thiếu sót và hạn chế. Rất mong nhận được sự đóng góp ý kiến quý báu của sinh viên và các bạn đọc để giáo trình ngày càng hoàn thiện hơn.

Chân thành cảm ơn!

Vĩnh Long, ngày 20 tháng 10 năm 2014

MỤC LỤC

Chương 1

TỔNG QUAN

1.1. NGÔN NGỮ, CẤP MÁY VÀ MÁY ẢO

1.1.1. 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 số với 0
- Di chuyển dữ liệu

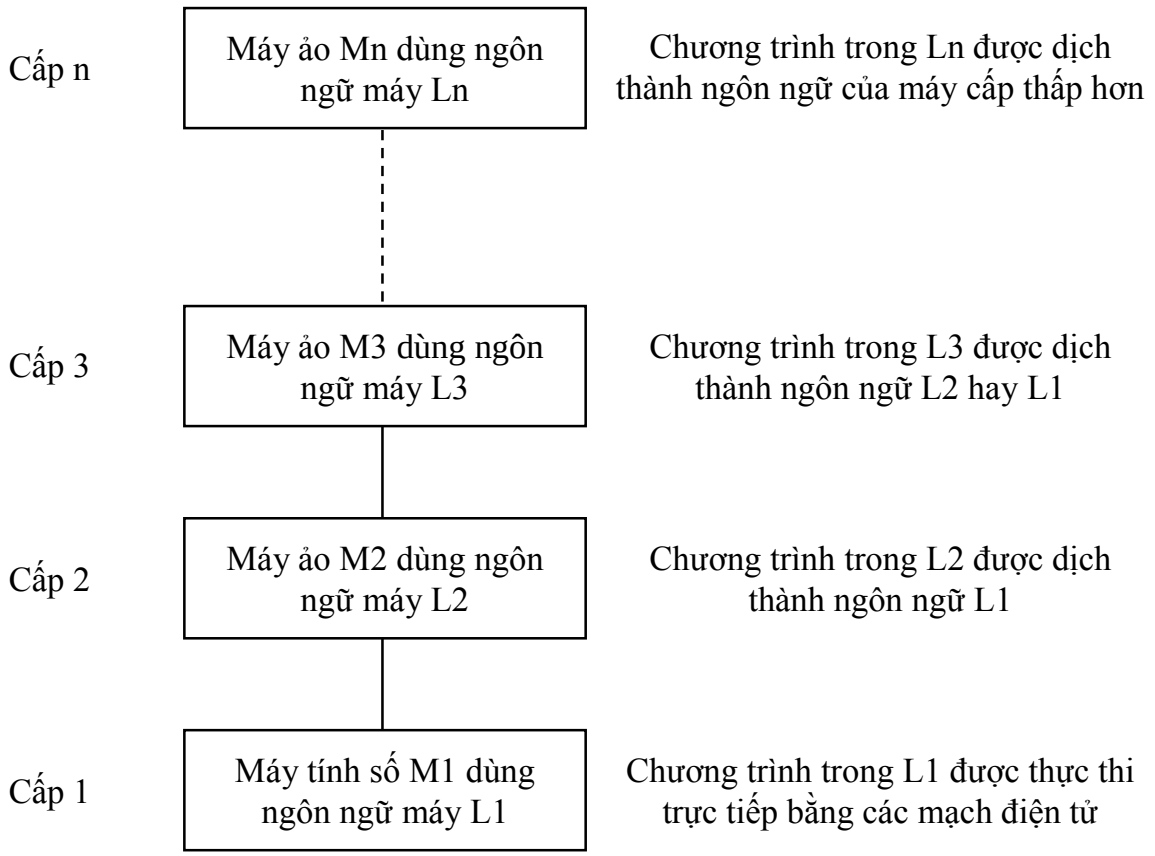
Tập lệnh của máy tính tạo thành một ngôn ngữ giúp con người có thể tác động lên máy tính, ngôn ngữ này gọi là ngôn ngữ máy (machine language). Tuy nhiên, hầu hết các ngôn ngữ máy đều đơn giản nên để thực hiện một yêu cầu nào đó, người thiết kế phải thực hiện công việc một cách phức tạp. Đó là chuyển 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 tính (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.

1.1.2. Máy nhiều cấp

Hầu hết các máy tính hiện nay đều có 6 cấp.

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.

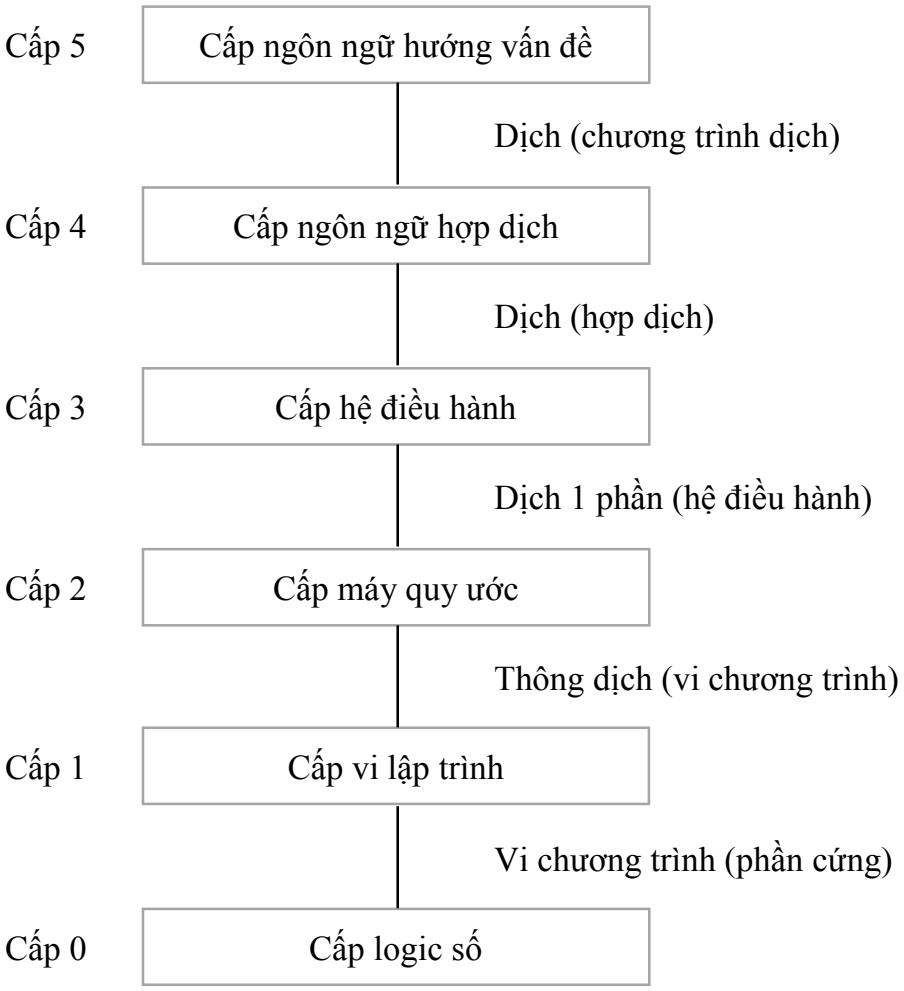
Cấp 1 là cấp ngôn ngữ máy thật sự. Cấp này có một chương trình gọi là vi chương trình (microprogram), vi chương trình có nhiệm vụ thông dịch các chỉ thị của cấp 2. Hầu hết các lệnh trong cấp này là di chuyển dữ liệu từ phần này đến phần khác của máy hay thực hiện việc một số kiểm tra đơn giản.

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).



Hình 1.2. Các cấp trên máy tính số

1.1.3. 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).

1.2. PHẦN CỨNG VÀ PHẦN MỀM

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.

1.3. LỊCH SỬ PHÁT TRIỂN CỦA MÁY TÍNH

1.3.1. 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.

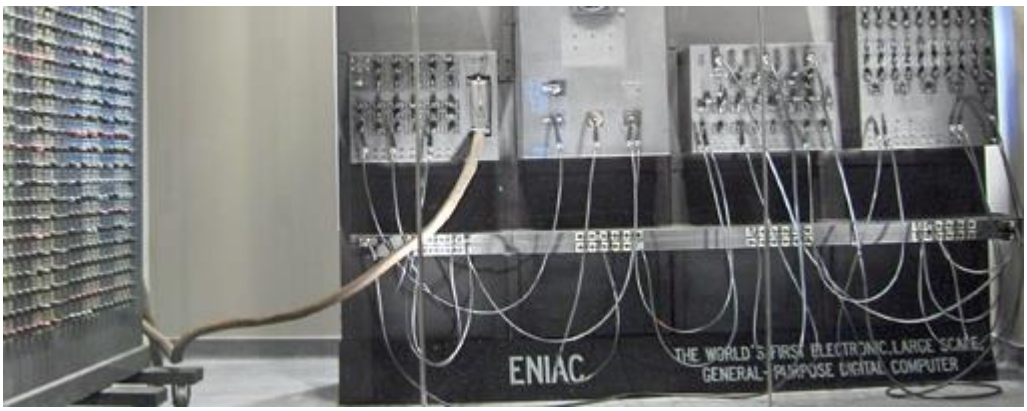
Phát triển tiếp theo của máy phân tích là máy đa năng. Máy đọc lệnh từ các thẻ đục lỗ và thực thi chúng. Bằng cách đục lỗ một chương trình khác trên thẻ nhập, máy phân tích có khả năng thực hiện các tính toán khác. Lập trình viên máy tính đầu tiên là Ada Lovelace đã tạo ra phần mềm cho máy phân tích.

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.

1.3.2. Máy tính đèn điện tử - thế hệ thứ nhất

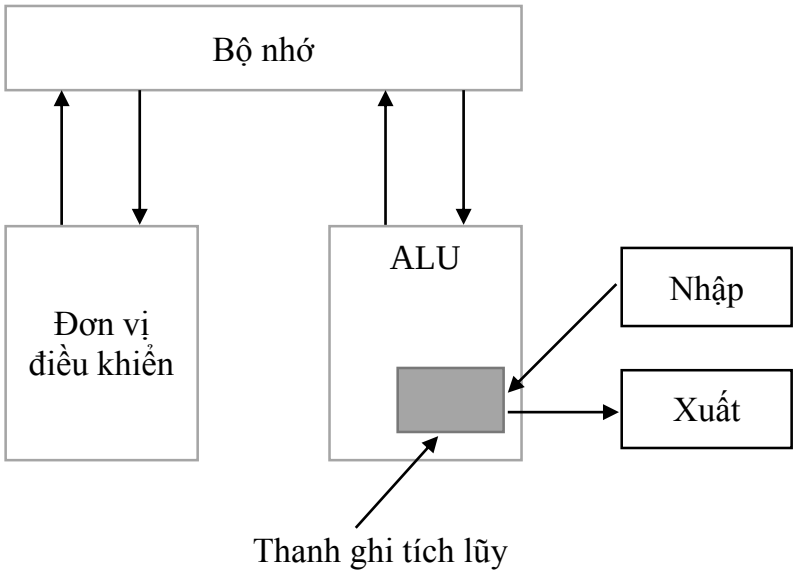
Năm 1943, máy tính số điện tử đầu tiên trên thế giới bắt đầu hoạt động, máy Colossus. Colossus do Alan Turing thiết kế nhằm thực hiện giải mã các thông điệp đã mã hóa trong chiến tranh thế giới thứ 2. Cũng trong năm 1943, Mauchley và Presper Eckert bắt đầu tiến hành xây dựng máy tính ENIAC (Electronic Numerical Integrator And Computer). ENIAC gồm 1800 đèn điện tử và 1500 rơ le (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.



Hình 1.3. Một bảng điều khiển của máy tính ENIAC

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.

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.



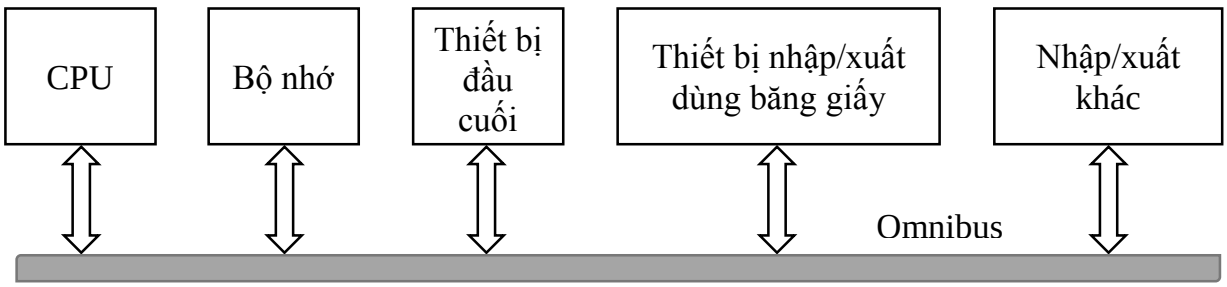
Hình 1.4. Máy von Neumann

1.3.3. 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 1.5. Omnibus của PDP-8

1.3.4. 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 224 byte nhớ (16 MB).

1.3.5. 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 trong thế hệ này có thể chia thành 5 loại: máy tính cá nhân, máy tính mini, siêu máy tính mini, mainframe, siêu máy tính.

Hình 1.6. Máy tính IBM PC 5050 (1981)

Hình 1.7. Siêu máy tính IBM Blue Gene/P (2007)

- Máy tính cá nhân: Các máy tính cá nhân thường dùng cho việc xử lý văn bản, các bảng tính và các ứng dụng tương hỗ khác.
- 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 cực 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.

1.3.6. Khuynh hướng hiện tại

Việc chuyển từ thế hệ thứ tư sang thế hệ thứ 5 còn chưa rõ ràng. Người Nhật đã và đang đi tiên phong trong các chương trình nghiên cứu để cho ra đời thế hệ thứ 5 của máy tính, thế hệ của những máy tính thông minh, dựa trên các ngôn ngữ trí tuệ nhân tạo như LISP và PROLOG,... và những giao diện người - máy thông minh. Đến thời điểm này, các nghiên cứu đã cho ra các sản phẩm bước đầu và gần đây nhất (2004) là sự ra mắt sản phẩm người máy thông minh gần giống với con người nhất: ASIMO (Advanced Step Innovative Mobility: Bước chân tiên tiến của đổi mới và chuyển động). Với hàng trăm nghìn máy móc điện tử tối tân đặt trong cơ thể, ASIMO có thể lên/xuống cầu thang một cách uyển chuyển, nhận diện người, các cử chỉ hành động, giọng nói và đáp ứng một số mệnh lệnh của con người. Thậm chí, nó có thể bắt chước cử động, gọi tên người và cung cấp thông tin ngay sau khi bạn hỏi, rất gần gũi và thân thiện. Hiện nay có nhiều công ty, viện nghiên cứu của Nhật thuê Asimo tiếp khách và hướng dẫn khách tham quan như: Viện Bảo tàng Khoa học năng lượng và Đổi mới quốc gia, hãng IBM Nhật Bản, Công ty điện lực Tokyo. Hãng Honda bắt đầu nghiên cứu ASIMO từ năm 1986 dựa vào nguyên lý chuyển động bằng hai chân. Cho tới nay, hãng đã chế tạo được 50 robot ASIMO.

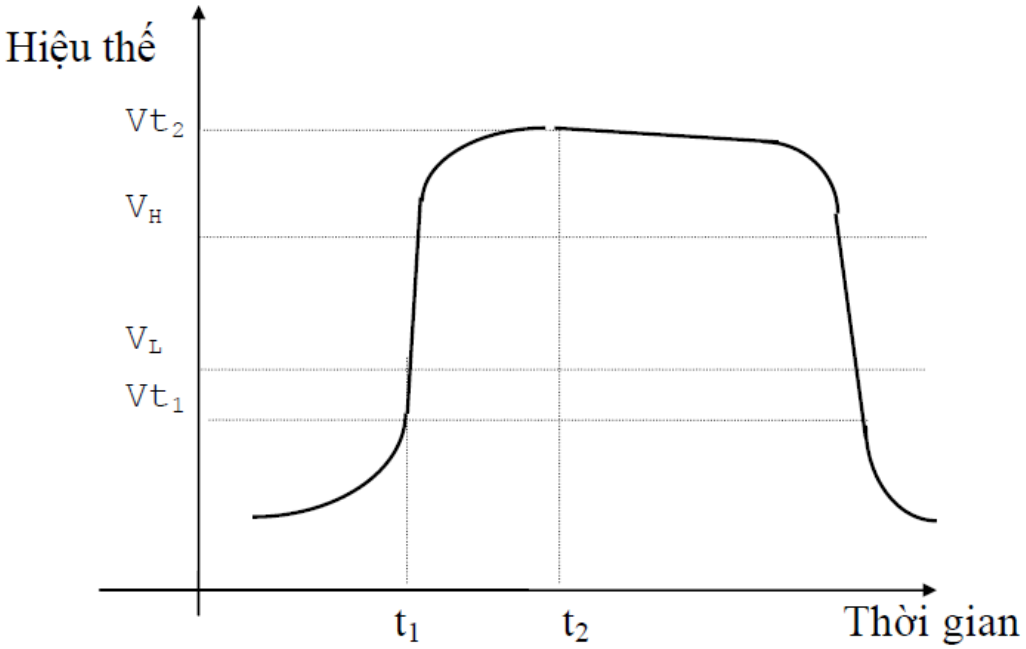
Các tiến bộ liên tục về mật độ tích hợp trong VLSI đã cho phép thực hiện các mạch vi xử lý ngày càng mạnh (8 bit, 16 bit, 32 bit và 64 bit với việc xuất hiện các bộ xử lý RISC năm 1986 và các bộ xử lý siêu vô hướng năm 1990). Chính các bộ xử lý này giúp thực hiện các máy tính song song với từ vài bộ xử lý đến vài ngàn bộ xử lý. Điều này làm các chuyên gia về kiến trúc máy tính tiên đoán thế hệ thứ 5 là thế hệ các máy tính xử lý song song.

1.4. BIỂU DIỄN THÔNG TIN TRONG MÁY TÍNH

1.4.1. Khái niệm thông tin

Khái niệm về thông tin gắn liền với sự hiểu biết một trạng thái cho sẵn trong nhiều trạng thái có thể có vào một thời điểm cho trước.

Trong 1.8, chúng ta quy ước có hai trạng thái có ý nghĩa: trạng thái thấp khi hiệu điện thế thấp hơn V_L và trạng thái cao khi hiệu điện thế lớn hơn V_H. Để có thông tin, ta phải xác định thời điểm ta nhìn trạng thái của tín hiệu. Thí dụ, tại thời điểm t_1 thì tín hiệu ở trạng thái thấp và tại thời điểm t_2 thì tín hiệu ở trạng thái cao.



Hình 1.8. Thông tin về 2 trạng thái có ý nghĩa của hiệu điện thế

1.4.2. Lượng thông tin và sự mã hoá thông tin

Thông tin được đo lường bằng đơn vị thông tin mà ta gọi là bit. Lượng thông tin được định nghĩa bởi công thức:

$$\mathbf{I = \log_2(N)}$$

Trong đó: I: là lượng thông tin tính bằng bit

N: là số trạng thái có thể có

Vậy một bit ứng với sự hiểu biết của một trạng thái trong hai trạng thái có thể có. Thí dụ, sự hiểu biết của một trạng thái trong 8 trạng thái có thể ứng với một lượng thông tin là:

$$\mathbf{I = \log_2(8) = 3\ bit}$$

Tám trạng thái được ghi nhận nhờ 3 số nhị phân (mỗi số nhị phân có thể có giá trị 0 hoặc 1).

Như vậy lượng thông tin là số con số nhị phân cần thiết để biểu diễn số trạng thái có thể có. Do vậy, một con số nhị phân được gọi là một bit. Một từ n bit có thể tượng trưng một trạng thái trong tổng số 2^n trạng thái mà từ đó có thể tượng trưng. Vậy một từ n bit tương ứng với một lượng thông tin n bit.

Trạng thái	x2	x1	x0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0

5	1	0	1
6	1	1	0
7	1	1	1

Bảng 1.1. Tám trạng thái khác nhau ứng với 3 bit nhị phân

1.4.3. Biểu diễn các số

Khái niệm hệ thống số: Cơ sở của một hệ thống số định nghĩa phạm vi các giá trị có thể có của một chữ số. Ví dụ: trong hệ thập phân, một chữ số có giá trị từ 0-9, trong hệ nhị phân, một chữ số (một bit) chỉ có hai giá trị là 0 hoặc 1.

Dạng tổng quát để biểu diễn giá trị của một số:

$$V_k = \sum_{i=-m}^{n-1} b_i . k^i$$

Trong đó:

V_k: Số cần biểu diễn giá trị

m: số thứ tự của chữ số phần lẻ

(phần lẻ của số có m chữ số được đánh số thứ tự từ -1 đến -m)

n-1: số thứ tự của chữ số phần nguyên

(phần nguyên của số có n chữ số được đánh số thứ tự từ 0 đến n-1)

b_i: giá trị của chữ số thứ i

k: hệ số (k=10: hệ thập phân; k=2: hệ nhị phân;...).

Ví dụ: biểu diễn số 541.25_{10}

$541.25_{10} = 5 * 10^2 + 4 * 10^1 + 1 * 10^0 + 2 * 10^{-1} + 5 * 10^{-2}$

$= (500)_{10} + (40)_{10} + (1)_{10} + (2/10)_{10} + (5/100)_{10}$

Một máy tính được chủ yếu cấu tạo bằng các mạch điện tử có hai trạng thái. Vì vậy, rất tiện lợi khi dùng các số nhị phân để biểu diễn số trạng thái của các mạch điện hoặc để mã hoá các ký tự, các số cần thiết cho vận hành của máy tính.

Để biến đổi một số hệ thập phân sang nhị phân, ta có hai phương thức biến đổi:

- Phương thức số dư để biến đổi phần nguyên của số thập phân sang nhị phân.
- Phương thức nhân để biến đổi phần lẻ của số thập phân sang nhị phân.

Ví dụ: Đổi 23.375_{10} sang nhị phân.

Bước 1: Chuyển đổi phần nguyên dùng phương thức số dư

Phép tính	Phần nguyên	Phần dư
23 / 2	11	1
11 / 2	5	1
5 / 2	2	1

2 / 2	1	0
1 / 2	0	1

⇨ $(23)_{10} = (10111)_2$

Bước 2: Chuyển đổi phần lẻ dùng phương thức nhân

Phép tính	Kết quả	Phần nguyên của kết quả
0.375 x 2	0.75	0
0.75 x 2	1.5	1
0.5 x 2	1.0	1

⇨ $(.375)_{10} = (.011)_2$

⇨ $(23.375)_{10} = (10111.011)_2$

Tuy nhiên, trong việc biến đổi phần lẻ của một số thập phân sang số nhị phân theo phương thức nhân, có một số trường hợp việc biến đổi số lặp lại vô hạn.

Ví dụ: 0.2 x 2 = 0.4

0.4 x 2 = 0.8

0.8 x 2 = 1.6

0.6 x 2 = 1.2

0.2 x 2 = 0.4

….

Trường hợp chuyển đổi qua lại giữa số hệ nhị phân và số hệ bát phân (Octal), hệ thập lục phân (Hexadecimal) ta có thể sử dụng quy tắc:

- Một ký số hệ bát phân (0..7) tương đương 3 ký số hệ nhị phân
- Một ký số hệ thập lục phân (0..9, A, B, C, D, E, F) tương đương 4 ký số hệ nhị phân

Ví dụ: $00111011_2 = (0011_2)(1011_2) = 3B_{16}$

$D6_{16} = 11010110_2$

$100110_2 = (100_2)(110_2) = 46_8$

$72_8 = 111010_2$

Một số nhị phân n bit có thể biểu diễn tất cả các số dương hệ thập phân có giá trị từ 0 tới $(2^n – 1)$.

Ví dụ: Một Byte (8 bit) có thể biểu diễn các số nguyên dương từ 0 đến 255

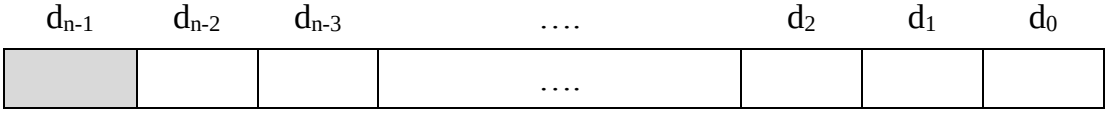
Một Word (từ nhớ 16 bit) có thể biểu diễn các số nguyên dương từ 0 đến 65535

1.4.4. Số nguyên có dấu

Có nhiều cách để biểu diễn một số n bit có dấu. Trong tất cả mọi cách thì bit cao nhất luôn tượng trưng cho dấu.

Khi đó, bit dấu có giá trị là 0 biểu diễn số nguyên dương, bit dấu có giá trị là 1 thì số nguyên âm. Tuy nhiên, cách biểu diễn dấu này không đúng trong trường hợp số được

biểu diễn bằng số thừa K mà ta sẽ xét ở phần sau trong chương này (bit dấu có giá trị là 1 thì số nguyên dương, bit dấu có giá trị là 0 thì số nguyên âm).



Số nguyên n bit có d_{n-1} là bit dấu và giá trị được biểu diễn bởi các bit từ d_0 đến d_{n-2}

a) Cách biểu diễn bằng trị tuyệt đối và dấu

Trong cách này, bit d_{n-1} là bit dấu và các bit từ d_0 tới d_{n-2} cho giá trị tuyệt đối. Một từ n bit tương ứng với số nguyên thập phân có dấu:

$$N = (-1)^{d_{n-1}} \sum_{i=0}^{n-2} d_i 2^i$$

Ví dụ: $+25_{10} = 00011001_2$ $-25_{10} = 10011001_2$

- Một Byte (8 bit) có thể biểu diễn các số có dấu từ -127 tới +127.
- Có hai cách biểu diễn số không là 0000 0000 (+0) và 1000 0000 (-0).

b) Cách biểu diễn bằng số bù 1

Trong cách biểu diễn này, số âm -N được có bằng cách thay các số nhị phân d_i của số dương N bằng số bù của nó (nghĩa là nếu $d_i = 0$ thì người ta đổi nó thành 1 và ngược lại).

Ví dụ: $+25_{10} = 00011001_2$ $-25_{10} = 11100110_2$

- Một Byte cho phép biểu diễn tất cả các số có dấu từ -127 ($1000\ 0000_2$) đến 127 ($0111\ 1111_2$)
- Có hai cách biểu diễn cho 0 là $0000\ 0000_2$ (+0) và $1111\ 1111_2$ (-0).

c) Cách biểu diễn bằng số bù 2

Để có số bù 2 của một số nào đó, người ta lấy số bù 1 rồi cộng thêm 1. Vậy một từ n bit (d_{n-1} ..... d_0) có trị thập phân:

$$N = -d_{n-1}2^{n-1} + \sum_{i=0}^{n-2} d_i 2^i$$

Một từ n bit có thể biểu diễn các số có dấu từ -2^{n-1} đến $2^{n-1} - 1$. Chỉ có một cách duy nhất để biểu diễn cho số không là tất cả các bit của số đó đều bằng không.

Ví dụ: $+25_{10} = 00011001_2$ $-25_{10} = 11100111_2$

- Dùng 1 Byte (8 bit) để biểu diễn một số có dấu lớn nhất là +127 và số nhỏ nhất là –128.
- Chỉ có một giá trị 0: $+0 = 00000000_2$, $-0 = 00000000_2$

d_3	d_2	d_1	d_0	N
0	0	0	0	**0**

d_3	d_2	d_1	d_0	N
1	0	0	0	**-8**

0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**

1	0	0	1	**-7**
1	0	1	0	**-6**
1	0	1	1	**-5**
1	1	0	0	**-4**
1	1	0	1	**-3**
1	1	1	0	**-2**
1	1	1	1	**-1**

Bảng 1.2. Số 4 bit có dấu theo cách biểu diễn số âm bằng số bù 2

d) Cách biểu diễn bằng số thừa K

Trong cách này, số dương của một số N có được bằng cách "cộng thêm vào" số thừa K được chọn sao cho tổng của K và một số âm bất kỳ luôn luôn dương. Số âm -N của số N có được bằng cách lấy K - N (hay lấy bù hai của số vừa xác định).

Ví dụ: (số thừa K=128, số "cộng thêm vào" 128 là một số nguyên dương. Số âm là số lấy bù hai số vừa tính, bỏ qua số giữ của bit cao nhất) :

$+25_{10} = 10011001_2$ $-25_{10} = 01100111_2$

- Dùng 1 Byte (8 bit) để biểu diễn một số có dấu lớn nhất là +127 và số nhỏ nhất là –128.
- Chỉ có một giá trị 0: $+0 = 10000000_2$, $-0 = 10000000_2$

Cách biểu diễn số nguyên có dấu bằng số bù 2 được dùng rộng rãi cho các phép tính số nguyên. Nó có lợi là không cần thuật toán đặc biệt nào cho các phép tính cộng và tính trừ, và giúp phát hiện dễ dàng các trường hợp bị tràn.

Các cách biểu diễn bằng "dấu , trị tuyệt đối" hoặc bằng "số bù 1" dẫn đến việc dùng các thuật toán phức tạp và bất lợi vì luôn có hai cách biểu diễn của số không. Cách biểu diễn bằng "dấu , trị tuyệt đối" được dùng cho phép nhân của số có dấu chấm động.

Cách biểu diễn bằng số thừa K được dùng cho số mũ của các số có dấu chấm động. Cách này làm cho việc so sánh các số mũ có dấu khác nhau trở thành việc so sánh các số nguyên dương.

1.4.5. Biểu diễn số với dấu chấm động

Trước khi đi vào cách biểu diễn số với dấu chấm động, chúng ta xét đến cách biểu diễn một số dưới dạng dấu chấm xác định.

Ví dụ:

- Trong hệ thập phân, số 254_{10} có thể biểu diễn dưới các dạng sau:

  $254 * 10^0$; $25.4 * 10^1$; $2.54 * 10^2$; $0.254 * 10^3$; $0.0254 * 10^4$; …
- Trong hệ nhị phân, số 0.00011_2 (tương đương với số 0.09375_{10}) có thể biểu diễn dưới các dạng:

  0.00011; $0.00011 * 2^0$; $0.0011 * 2^{-1}$; $0.011 * 2^{-2}$; $0.11 * 2^{-3}$; $1.1 * 2^{-4}$

Các cách biểu diễn này gây khó khăn trong một số phép so sánh các số. Để dễ dàng trong các phép tính, các số được chuẩn hóa về một dạng biểu diễn:

$$\pm 1.fff...f \times 2^{\pm E}$$

Trong đó: f là phần lẻ; E là phần mũ

Số chấm động được chuẩn hoá, cho phép biểu diễn gần đúng các số thập phân rất lớn hay rất nhỏ dưới dạng một số nhị phân theo một dạng qui ước. Thành phần của số chấm động bao gồm: *phần dấu*, *phần mũ* và *phần định trị*. Như vậy, cách này cho phép biểu diễn gần đúng các số thực, tất cả các số đều có cùng cách biểu diễn.

Có nhiều cách biểu diễn dấu chấm động, trong đó cách biểu diễn theo chuẩn IEEE 754 được dùng rộng rãi trong khoa học máy tính hiện nay. Trong cách biểu diễn này, phần định trị có dạng *1,f* với số *1* ẩn tăng và *f* là phần số lẻ.

Chuẩn IEEE 754 định nghĩa hai dạng biểu diễn số chấm động:

- Số chấm động chính xác đơn với định dạng được định nghĩa: chiều dài số là 32 bit được chia thành các trường:
  - Dấu S (Sign bit - 1 bit)
  - Mũ E (Exponent - 8 bit)
  - Phần lẻ F (Fraction - 23 bit).

Số này tương ứng với số thực $\mathbf{(-1)^S * (1,f_1f_2 ..... f_{23}) * 2^{(E - 127)}}$

bit	31	30 ... 23	22			1	0
	S	E	f_1	f_2	….	f_{22}	f_{23}

Hình 1.9. Biểu diễn số có dấu chấm động chính xác đơn với 32 bit

- Số chấm động chính xác kép với định dạng được định nghĩa: chiều dài số là 64 bit được chia thành các trường:
  - Dấu S (Sign bit - 1 bit)
  - Mũ E (Exponent - 11 bit)
  - Phần lẻ F (Fraction - 52 bit).

Số này tương ứng với số thực $\mathbf{(-1)^S * (1,f_1f_2 ..... f_{52}) * 2^{(E - 1023)}}$

bit	63	62 ... 52	51			1	0
	S	E	f_1	f_2	….	f_{51}	f_{52}

Hình 1.10. Biểu diễn số có dấu chấm động chính xác kép với 64 bit

Để thuận lợi trong một số phép tính toán, IEEE định nghĩa một số dạng mở rộng của chuẩn IEEE 754:

Tham số	Chính xác đơn	Mở rộng chính xác đơn	Chính xác kép	Mở rộng chính xác kép
Chiều dài (bit)	32	≥ 43	64	≥ 79
Chiều dài trường mũ (E)	8	≥ 11	11	≥ 15

Số thừa	127	_	1023	_
Giá trị mũ tối đa	127	≥ 1023	1023	≥ 16383
Giá trị mũ tối thiểu	-126	≤ -1022	-1022	≤ -16382
Chiều dài trường lẻ F (bit)	23	≥ 31	52	≥ 63

Chuẩn IEEE 754 cho phép biểu diễn các số chuẩn hoá (các bit của E không cùng lúc bằng 0 hoặc bằng 1), các số không chuẩn hoá (các bit của E không cùng lúc bằng 0 và phần số lẻ f1 f2 ... khác không), trị số 0 (các bit của E không cùng lúc bằng 0 và phần số lẻ bằng không), và các ký tự đặc biệt (các bit của E không cùng lúc bằng 1 và phần lẻ khác không).

Ví dụ: các bước biến đổi số thập phân -12.625_{10} sang số chấm động chuẩn IEEE 754 chính xác đơn (32 bit):

- Bước 1: Đổi số -12.625_{10} sang nhị phân: $-12.625_{10} = -1100.101_2$
- Bước 2: Chuẩn hoá: $-1100.101_2 = -1.100101_2 \times 2^3$ (Số 1.100101_2 dạng 1.f)
- Bước 3: Điền các bit vào các trường theo chuẩn:
  - Số âm: bit dấu S có giá trị 1
  - Phần mũ E với số thừa K=127, ta có: E – 127 = 3

    → E = 3 + 127 = 130 = 10000010_2

Kết quả nhận được:

32 bit		
1	**10000010**	**10010100000000000000000**
S	E	F

1.4.6. Số BCD (Binary Coded Decimal)

Một vài ứng dụng, đặc biệt ứng dụng quản lý, bắt buộc các phép tính thập phân phải chính xác, không làm tròn số. Với một số bit cố định, ta không thể đổi một cách chính xác số nhị phân thành số thập phân và ngược lại. Vì vậy, khi cần phải dùng số thập phân, ta dùng cách biểu diễn số thập phân mã bằng nhị phân (BCD: Binary Coded Decimal) theo đó mỗi số thập phân được mã với 4 số nhị phân.

d_3	d_2	d_1	d_0	Số thập phân
0	0	0	0	**0**
0	0	0	1	**1**
0	0	1	0	**2**
0	0	1	1	**3**
0	1	0	0	**4**

d_3	d_2	d_1	d_0	Số thập phân
0	1	0	1	**5**
0	1	1	0	**6**
0	1	1	1	**7**
1	0	0	0	**8**
1	0	0	1	**9**

Bảng 1.3. Số thập phân mã bằng nhị phân

Để biểu diễn số BCD có dấu, người ta thêm số 0 trước một số dương cần tính, ta có số âm của số BCD bằng cách lấy bù 10 số cần tính.

Ví dụ: Biểu diễn số $+079_{10}$ bằng số BCD: 0000 0111 1001

Bù 9: 1001 0010 0000

+1

Bù 10: 1001 0010 0001

Vậy, ta có: Số -079_{10} trong cách biểu diễn số BCD: $1001\ 0010\ 0001_{BCD}$

Cách tính toán trên tương đương với cách sau:

- Trước hết ta lấy số bù 9 của số 079 bằng cách: 999 - 079 = 920
- Cộng 1 vào số bù 9 ta được số bù 10: 920 + 1 = 921
- Biểu diễn số 921 dưới dạng số BCD, ta có: $1001\ 0010\ 0001_{BCD}$

1.4.7. Biểu diễn các ký tự

Tùy theo các hệ thống khác nhau, có thể sử dụng các bảng mã khác nhau: ASCII, EBCDIC, UNICODE, ... Các hệ thống trước đây thường dùng bảng mã ASCII (American Standard Codes for Information Interchange) để biểu diễn các chữ, số và một số dấu thường dùng mà ta gọi chung là ký tự. Mỗi ký tự được biểu diễn bởi 7 bit trong một Byte. Hiện nay, một trong các bảng mã thông dụng được dùng là Unicode, trong bảng mã này, mỗi ký tự được mã hoá bởi 2 Byte.

Binary	Dec	Hex	Glyph
010 0000	32	20	(space)
010 0001	33	21	!
010 0010	34	22	"
010 0011	35	23	#
010 0100	36	24	$
010 0101	37	25	%
010 0110	38	26	&
010 0111	39	27	'
010 1000	40	28	(
010 1001	41	29	)
010 1010	42	2A	*
010 1011	43	2B	+
010 1100	44	2C	,
010 1101	45	2D	-
010 1110	46	2E	.
010 1111	47	2F	/

Binary	Dec	Hex	Glyph
100 0000	64	40	@
100 0001	65	41	A
100 0010	66	42	B
100 0011	67	43	C
100 0100	68	44	D
100 0101	69	45	E
100 0110	70	46	F
100 0111	71	47	G
100 1000	72	48	H
100 1001	73	49	I
100 1010	74	4A	J
100 1011	75	4B	K
100 1100	76	4C	L
100 1101	77	4D	M
100 1110	78	4E	N
100 1111	79	4F	O

Binary	Dec	Hex	Glyph
110 0000	96	60	`
110 0001	97	61	a
110 0010	98	62	b
110 0011	99	63	c
110 0100	100	64	d
110 0101	101	65	e
110 0110	102	66	f
110 0111	103	67	g
110 1000	104	68	h
110 1001	105	69	i
110 1010	106	6A	j
110 1011	107	6B	k
110 1100	108	6C	l
110 1101	109	6D	m
110 1110	110	6E	n
110 1111	111	6F	o

011 0000	48	30	0	101 0000	80	50	P	111 0000	112	70	p
011 0001	49	31	1	101 0001	81	51	Q	111 0001	113	71	q
011 0010	50	32	2	101 0010	82	52	R	111 0010	114	72	r
011 0011	51	33	3	101 0011	83	53	S	111 0011	115	73	s
011 0100	52	34	4	101 0100	84	54	T	111 0100	116	74	t
011 0101	53	35	5	101 0101	85	55	U	111 0101	117	75	u
011 0110	54	36	6	101 0110	86	56	V	111 0110	118	76	v
011 0111	55	37	7	101 0111	87	57	W	111 0111	119	77	w
011 1000	56	38	8	101 1000	88	58	X	111 1000	120	78	x
011 1001	57	39	9	101 1001	89	59	Y	111 1001	121	79	y
011 1010	58	3A	:	101 1010	90	5A	Z	111 1010	122	7A	z
011 1011	59	3B	;	101 1011	91	5B	[	111 1011	123	7B	{
011 1100	60	3C	<	101 1100	92	5C	\	111 1100	124	7C	|
011 1101	61	3D	=	101 1101	93	5D	]	111 1101	125	7D	}
011 1110	62	3E	>	101 1110	94	5E	^	111 1110	126	7E	~
011 1111	63	3F	?	101 1111	95	5F	_				

Bảng 1.4. Bảng mã ASCII (ASCII printable code chart)

1.4.8. Đơn vị đo thông tin

Máy tính sử dụng hệ nhị phân để biểu diễn thông tin. Do đó một chữ số nhị phân được xem như là đơn vị chứa thông tin nhỏ nhất gọi là bit (b). Một bit tương ứng với một sự kiện có 1 trong 2 trạng thái.

Tên gọi	Ký hiệu	Giá trị
Byte	B	8 bit
KiloByte	KB	2^{10} B = 1024 B
MegaByte	MB	2^{10} KB = 2^{20} B
GigaByte	GB	2^{10} MB = 2^{30} B
TeraByte	TB	2^{10} GB = 2^{40} B
PetaByte	PB	2^{10} TB = 2^{50} B
ExaByte	EB	2^{10} PB = 2^{60} B
ZettaByte	ZB	2^{10} EB = 2^{70} B
YottaByte	YB	2^{10} ZB = 2^{80} B

Bảng 1.5. Các đơn vị đo thông tin trong máy tính

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

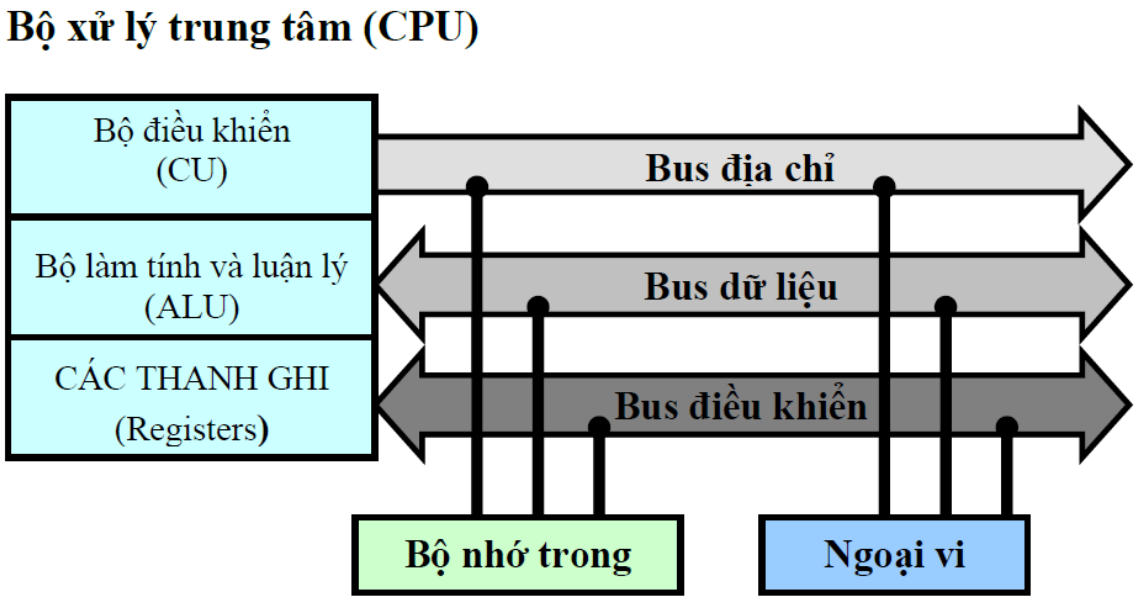
1. Dựa vào tiêu chuẩn nào người ta phân chia máy tính thành các thế hệ?
2. Đặc trưng cơ bản của các máy tính thế hệ thứ nhất?
3. Đặc trưng cơ bản của các máy tính thế hệ thứ hai?
4. Đặc trưng cơ bản của các máy tính thế hệ thứ ba?
5. Đặc trưng cơ bản của các máy tính thế hệ thứ tư?
6. Khuynh hướng phát triển của máy tính điện tử ngày nay là gì?
7. Việc phân loại máy tính dựa vào tiêu chuẩn nào?
8. Khái niệm thông tin trong máy tính được hiểu như thế nào?
9. Lượng thông tin là gì ?
10. Sự hiểu biết về một trạng thái trong 4096 trạng thái có thể có ứng với lượng thông tin là bao nhiêu?
11. Điểm chung nhất trong các cách biểu diễn một số nguyên n bit có dấu là gì?
12. Số nhị phân 8 bit $(11001100)_2$, số này tương ứng với số nguyên thập phân có dấu là bao nhiêu nếu số đang được biểu diễn trong cách biểu diễn:
    a. Dấu và trị tuyệt đối.
    b. Số bù 1.
    c. Số bù 2.
13. Đổi các số sau đây:
    a. $(011011)_2$ ra số thập phân.
    b. $(-2005)_{10}$ ra số nhị phân 16 bits.
    c. $(55.875)_{10}$ ra số nhị phân.
14. Biểu diễn số thực (31.75)10 dưới dạng số có dấu chấm động chính xác đơn 32 bit.
15. Cho biết giá trị thập phân của các số được biểu diễn theo dạng chính xác đơn sau:
    a. 11000110010101100000000000000000
    b. 01100110111001101110000000000000

Chương 2

KIẾN TRÚC PHẦN MỀM BỘ XỬ LÝ

2.1. THÀNH PHẦN CƠ BẢN CỦA MỘT MÁY TÍNH

Thành phần cơ bản của một bộ máy tính gồm: bộ xử lý trung tâm (CPU: Central Processing Unit), bộ nhớ trong, các bộ phận nhập - xuất thông tin. Các bộ phận trên được kết nối với nhau thông qua các hệ thống bus. Hệ thống bus bao gồm: bus địa chỉ, bus dữ liệu và bus điều khiển. Bus địa chỉ và bus dữ liệu dùng trong việc chuyển dữ liệu giữa các bộ phận trong máy tính. Bus điều khiển làm cho sự trao đổi thông tin giữa các bộ phận được đồng bộ. Thông thường người ta phân biệt một bus hệ thống dùng trao đổi thông tin giữa CPU và bộ nhớ trong (thông qua cache), và một bus vào-ra dùng trao đổi thông tin giữa các bộ phận vào-ra và bộ nhớ trong.



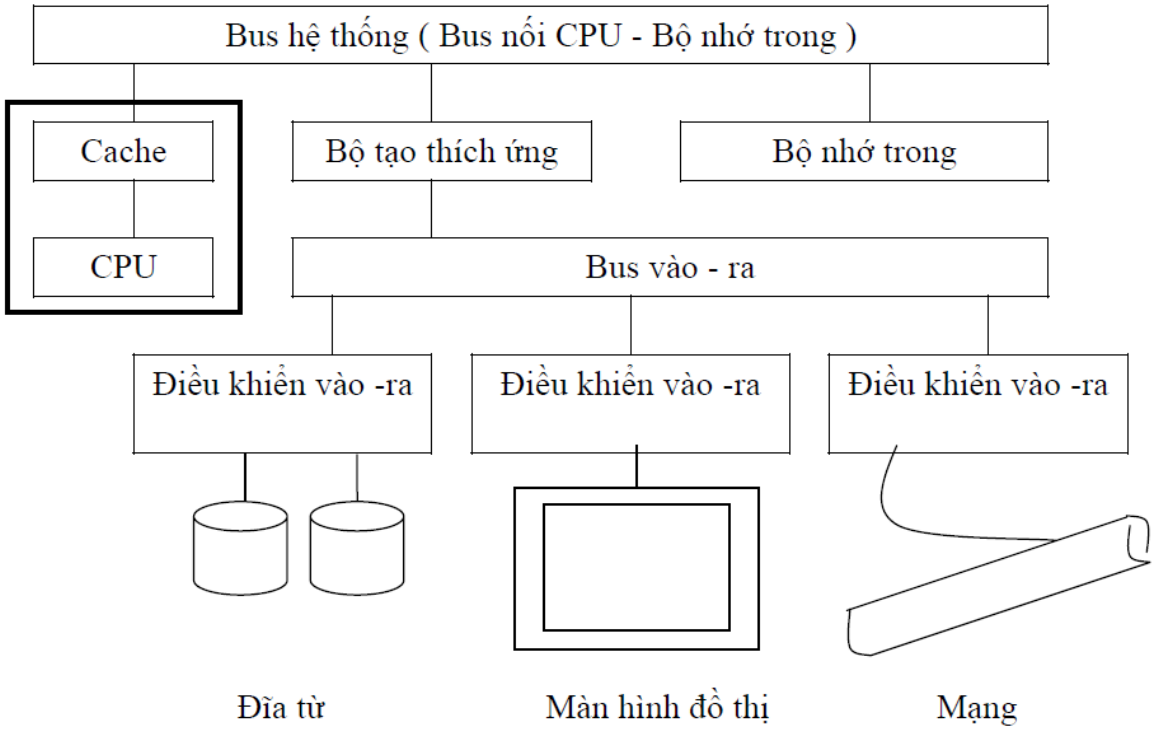
Hình 2.1. Cấu trúc của một hệ máy tính đơn giản

Một chương trình sẽ được sao chép từ đĩa cứng vào bộ nhớ trong cùng với các thông tin cần thiết cho chương trình hoạt động, các thông tin này được nạp vào bộ nhớ trong từ các bộ phận cung cấp thông tin (ví dụ như một bàn phím hay một đĩa từ). Bộ xử lý trung tâm sẽ đọc các lệnh và dữ liệu từ bộ nhớ, thực hiện các lệnh và lưu các kết quả trở lại bộ nhớ trong hay cho xuất kết quả ra bộ phận xuất thông tin (màn hình hay máy in).

Thành phần cơ bản của một máy tính bao gồm :

- **Bộ nhớ trong:** Đây là một tập hợp các ô nhớ, mỗi ô nhớ có một số bit nhất định và chức một thông tin được mã hoá thành số nhị phân mà không quan tâm đến kiểu của dữ liệu mà nó đang chứa. Các thông tin này là các lệnh hay số liệu. Mỗi ô nhớ của bộ nhớ trong đều có một địa chỉ. Thời gian thâm nhập vào một ô nhớ bất kỳ trong bộ nhớ là như nhau. Vì vậy, bộ nhớ trong còn được gọi là bộ nhớ truy cập ngẫu nhiên (RAM: Random Access Memory). Độ dài của một từ máy tính (Computer Word) là 32 bit (hay 4 byte), tuy nhiên dung lượng một ô nhớ thông thường là 8 bit (1 Byte).

- **Bộ xử lý trung tâm (CPU):** đây là bộ phận thi hành lệnh. CPU lấy lệnh từ bộ nhớ trong và lấy các số liệu mà lệnh đó xử lý. Bộ xử lý trung tâm gồm có hai phần: phần thi hành lệnh và phần điều khiển. Phần thi hành lệnh bao gồm bộ làm toán và luận lý (ALU: Arithmetic And Logic Unit) và các thanh ghi (Registers). Nó có nhiệm vụ làm các phép toán trên số liệu. Phần điều khiển có nhiệm vụ đảm bảo thi hành các lệnh một cách tuần tự và tác động các mạch chức năng để thi hành các lệnh.
- **Bộ phận vào - ra:** đây là bộ phận xuất nhập thông tin, bộ phận này thực hiện sự giao tiếp giữa máy tính và người dùng hay giữa các máy tính trong hệ thống mạng (đối với các máy tính được kết nối thành một hệ thống mạng). Các bộ phận xuất nhập thường gặp là: bộ lưu trữ ngoài, màn hình, máy in, bàn phím, chuột, máy quét ảnh, các giao diện mạng cục bộ hay mạng diện rộng... Bộ tạo thích ứng là một vi mạch tổng hợp (chipset) kết nối giữa các hệ thống bus có các tốc độ dữ liệu khác nhau.



Hình 2.2. Sơ đồ mô tả hoạt động điển hình của một máy tính

2.2. ĐỊNH NGHĨA KIẾN TRÚC MÁY TÍNH

Kiến trúc máy tính bao gồm ba phần: Kiến trúc phần mềm, tổ chức của máy tính và lắp đặt phần cứng.

Kiến trúc phần mềm của máy tính chủ yếu là kiến trúc phần mềm của bộ xử lý, bao gồm: tập lệnh, dạng các lệnh và các kiểu định vị.

- Trong đó, tập lệnh là tập hợp các lệnh mã máy (mã nhị phân) hoàn chỉnh có thể hiểu và được xử lý bởi bộ xử lý trung tâm, thông thường các lệnh trong tập lệnh được trình bày dưới dạng hợp ngữ. Mỗi lệnh chứa thông tin yêu cầu bộ xử lý thực hiện, bao gồm: mã tác vụ, địa chỉ toán hạng nguồn, địa chỉ toán hạng kết quả, lệnh kế tiếp (thông thường thì thông tin này ẩn).
- Kiểu định vị chỉ ra cách thức thâm nhập toán hạng.

Kiến trúc phần mềm là phần mà các lập trình viên hệ thống phải nắm vững để việc lập trình hiểu quả, ít sai sót.

Tổ chức của máy tính liên quan đến cấu trúc bên trong của bộ xử lý, cấu trúc các bus, các cấp bộ nhớ và các mặt kỹ thuật khác của máy tính. Phần này sẽ được nói đến ở các chương sau.

Lắp đặt phần cứng của máy tính ám chỉ việc lắp ráp một máy tính dùng các linh kiện điện tử và các bộ phận phần cứng cần thiết. Chúng ta không nói đến phần này trong giáo trình.

Ta nên lưu ý rằng một vài máy tính có cùng kiến trúc phần mềm nhưng phần tổ chức là khác nhau (VAX- 11/780 và VAX 8600). Các máy VAX- 11/780 và VAX-11/785 có cùng kiến trúc phần mềm và phần tổ chức gần giống nhau. Tuy nhiên việc lắp đặt phần cứng các máy này là khác nhau. Máy VAX- 11/785 đã dùng các mạch tích hợp hiện đại để cải tiến tần số xung nhịp và đã thay đổi một ít tổ chức của bộ nhớ trong.

2.3. TẬP LỆNH

Mục tiêu của phần này là dùng các ví dụ trích từ các kiến trúc phần mềm được dùng nhiều nhất, để cho thấy các kỹ thuật ở mức ngôn ngữ máy dùng để thi hành các cấu trúc trong các ngôn ngữ cấp cao.

Để minh hoạ bằng thí dụ, ta dùng cú pháp lệnh trong hợp ngữ sau đây :

Từ gợi nhớ mã lệnh, thanh ghi đích, thanh ghi nguồn 1, thanh ghi nguồn 2

Từ gợi nhớ mã lệnh mô tả ngắn gọn tác vụ phải thi hành trên các thanh ghi nguồn, kết quả được lưu giữ trong thanh ghi đích.

Mỗi lệnh của ngôn ngữ cấp cao được xây dựng bằng một lệnh mã máy hoặc một chuỗi nhiều lệnh mã máy. Lệnh nhảy (GOTO) được thực hiện bằng các lệnh hợp ngữ về nhảy (JUMP) hoặc lệnh hợp ngữ về vòng. Chúng ta phân biệt lệnh nhảy làm cho bộ đếm chương trình được nạp vào địa chỉ tuyệt đối nơi phải nhảy đến (PC ← địa chỉ tuyệt đối nơi phải nhảy tới), với lệnh vòng theo đó ta chỉ cần cộng thêm một độ dời vào bộ đếm chương trình (PC ← PC + độ dời). Ta lưu ý là trong trường hợp sau, PC chứa địa chỉ tương đối so với địa chỉ của lệnh sau lệnh vòng.

2.3.1. Gán trị

Việc gán trị, gồm cả gán trị cho biểu thức số học và logic, được thực hiện nhờ một số lệnh mã máy. Cho các kiến trúc RISC, ta có thể nêu lên các lệnh sau :

- Lệnh bộ nhớ

LOAD Ri, M (địa chỉ) M[địa chỉ] ← Ri

STORE Ri, M(địa chỉ) ; Ri ← M[địa chỉ]

Địa chỉ được tính tùy theo kiểu định vị được dùng.

- Lệnh tính toán số học: tính toán số nguyên trên nội dung của hai thanh ghi Ri, Rj và xếp kết quả vào trong Rk:

ADD (cộng)

ADDD (cộng số có dấu chấm động, chính xác kép)

SUB (trừ)

SUBD (trừ số có dấu chấm động, chính xác kép)

MUL (nhân)

DIV (chia)

- Lệnh logic: thực hiện phép tính logic cho từng bit một.

AND (lệnh VÀ)

OR (lệnh HOẶC)

XOR (lệnh HOẶC LOẠI)

NEG (lệnh lấy số bù 1)

- Các lệnh dịch chuyển số học hoặc logic (SHIFT), quay vòng (ROTATE) có hoặc không có số giữ ở ngã vào, sang phải hoặc sang trái. Các lệnh này được thực hiện trên một thanh ghi và kết quả lưu giữ trong thanh ghi khác. Số lần dịch chuyển (mỗi lần dịch sang phải hoặc sang trái một bit) thường được xác định trong thanh ghi thứ ba.

Cho các kiến trúc kiểu RISC, ta có :

SLL (shift left logical: dịch trái logic)

SRL (shift right logical: dịch phải logic)

SRA (shift right arithemtic: dịch phải số học)

2.3.2. Lệnh có điều kiện

Lệnh có điều kiện có dạng :

Nếu <điều kiện> thì <chuỗi lệnh 1> nếu không <chuỗi lệnh 2>

(IF <condition> THEN <instructions1> ELSE <instructions2>)

Lệnh này buộc phải ghi nhớ điều kiện và nhảy vòng nếu điều kiện được thỏa

a) Ghi nhớ điều kiện

Bộ làm tính ALU cung cấp kết quả ở ngã ra tùy theo các ngã vào và phép tính cần làm. Nó cũng cho một số thông tin khác về kết quả dưới dạng các bit trạng thái. Các bit này là những đại lượng logic ĐÚNG hoặc SAI.

Trong các bit trạng thái ta có bit dấu S (Sign - Đúng nếu kết quả âm), bit trắc nghiệm zero Z (Zero - Đúng nếu kết quả bằng không), bit tràn OVF (Overflow) ĐÚNG nếu phép tính số học làm thanh ghi không đủ khả năng lưu trữ kết quả, bit số giữ C (carry) ĐÚNG nếu số giữ ở ngã ra là 1 .... Các bit trên thường được gọi là bit mã điều kiện.

Có hai kỹ thuật cơ bản để ghi nhớ các bit trạng thái:

- Cách thứ nhất, ghi các trạng thái trong một thanh ghi đa dụng.

Ví dụ: lệnh CMP Rk, Ri, Rj

Lệnh trên sẽ làm phép tính trừ Ri - Rj mà không ghi kết quả phép trừ, mà lại ghi các bit trạng thái vào thanh ghi Rk. Thanh ghi này được dùng cho một lệnh nhảy có điều kiện. Điểm lợi của kỹ thuật này là giúp lưu trữ nhiều trạng thái sau nhiều phép tính để dùng về sau. Điểm bất lợi là phải dùng một thanh ghi đa dụng để ghi lại trạng thái sau mỗi phép tính mà số thanh ghi này lại bị giới hạn ở 32 trong các bộ xử lý hiện đại.

- Cách thứ hai, là để các bit trạng thái vào một thanh ghi đặc biệt gọi là thanh ghi trạng thái. Vấn đề lưu giữ nội dung thanh ghi này được giải quyết bằng nhiều cách. Trong kiến trúc SPARC, chỉ có một số giới hạn lệnh được phép thay đổi thanh ghi trạng thái ví dụ như lệnh ADDCC, SUBCC (các lệnh này thực hiện các phép tính cộng ADD và phép tính trừ SUB và còn làm thay đổi thanh ghi trạng thái). Trong kiến trúc PowerPC, thanh ghi trạng thái được phân thành 8 trường, mỗi trường 4 bit, vậy là thanh ghi đã phân thành 8 thanh ghi trạng thái con.

b) Nhảy vòng

Các lệnh nhảy hoặc nhảy vòng có điều kiện, chỉ thực hiện lệnh nhảy khi điều kiện được thỏa. Trong trường hợp ngược lại, việc thực hiện chương trình được tiếp tục với lệnh sau đó. Lệnh nhảy xem xét thanh ghi trạng thái và chỉ nhảy nếu điều kiện nêu lên trong lệnh là đúng.

Chúng ta xem một ví dụ thực hiện lệnh nhảy có điều kiện.

Giả sử trạng thái sau khi bộ xử lý thi hành một tác vụ, được lưu trữ trong thanh ghi, và bộ xử lý thi hành các lệnh sau :

1. CMP R4, R1, R2 : So sánh R1 và R2 bằng cách trừ R1 cho R2 và lưu giữ trạng thái trong R4
2. BGT R4, +2 : Nhảy bỏ 2 lệnh nếu R1 > R2
3. ADD R3, R0, R2 : R0 có giá trị 0. Chuyển nội dung của R2 vào R3
4. BRA +1 : nhảy bỏ 1 lệnh
5. ADD R3, R0, R1 : chuyển nội dung R1 vào R3
6. Lệnh kế

Nếu R1 > R2 thì chuỗi lệnh được thi hành là 1, 2, 5, 6 được thi hành, nếu không thì chuỗi lệnh 1, 2, 3, 4, 6 được thi hành.

Chuỗi các lệnh trên , trong đó có 2 lệnh nhảy, thực hiện công việc sau đây:

Nếu R1 > R2 thì R3 = R1 nếu không R3 = R2

Các lệnh nhảy làm tốc độ thi hành lệnh chậm lại, trong các CPU hiện đại dùng kỹ thuật ống dẫn. Trong một vài bộ xử lý người ta dùng lệnh di chuyển có điều kiện để tránh dùng lệnh nhảy trong một vài trường hợp. Thí dụ trên đây có thể được viết lại :

1. CMP R4, R1, R2 : So sánh R1 và R2 và để các bit trạng thái trong R4.
2. ADD R3, R0, R2 : Di chuyển R2 vào R3
3. MGT R4, R3, R1 : (MGT : Move if greater than). Nếu R1 > R2 thì di chuyển R1 vào R3

2.3.3. Vòng lặp

Các lệnh vòng lặp có thể được thực hiện nhờ lệnh nhảy có điều kiện mà ta đã nói ở trên. Trong trường hợp này, ta quản lý số lần lặp lại bằng một bộ đếm vòng lặp, và người ta kiểm tra bộ đếm này sau mỗi vòng lặp để xem đã đủ số vòng cần thực hiện hay chưa.

Bộ xử lý PowerPC có một lệnh quản lý vòng lặp

BNCT Ri, độ dời

Với thanh ghi Ri chứa số lần lặp lại.

Lệnh này làm các công việc sau:

Ri := Ri -1

Nếu Ri <> 0, PC := PC + độ dời. Nếu không thì tiếp tục thi hành lệnh kế.

2.3.4. Thâm nhập bộ nhớ ngăn xếp

Ngăn xếp là một tổ chức bộ nhớ sao cho ta chỉ có thể đọc một từ ở đỉnh ngăn xếp hoặc viết một từ vào đỉnh ngăn xếp. Địa chỉ của đỉnh ngăn xếp được chứa trong một thanh ghi đặc biệt gọi là con trỏ ngăn xếp SP (Stack Pointer).

Ứng với cấu trúc ngăn xếp, người ta có lệnh viết vào ngăn xếp PUSH và lệnh lấy ra khỏi ngăn xếp POP. Các lệnh này vận hành như sau:

- Cho lệnh PUSH

SP := SP +1

M (SP) := Ri (Ri là thanh ghi cần viết vào ngăn xếp)

- Cho lệnh POP

Ri := M(SP) (Ri là thanh ghi, nhận từ lấy ra khỏi ngăn xếp)

SP := SP -1

Trong các bộ xử lý RISC, việc viết vào hoặc lấp ra khỏi ngăn xếp dùng các lệnh bình thường. Ví dụ thanh ghi R30 là con trỏ ngăn xếp thì việc viết vào ngăn xếp được thực hiện bằng các lệnh:

```
ADDI R30, R30, 4 ; tăng con trỏ ngăn xếp lên 4 vì từ dài 32 bit
STORE Ri, (R30) ; Viết Ri vào đỉnh ngăn xếp
```

Việc lấy ra khỏi ngăn xếp được thực hiện bằng các lệnh :

```
LOAD Ri, (R30) ; lấy số liệu ở đỉnh ngăn xếp và nạp vào Ri
SUBI R30, R30,4 ; giảm con trỏ ngăn xếp bớt 4
```

2.3.5. Các thủ tục

Các thủ tục được gọi từ bất cứ nơi nào của chương trình nhờ lệnh gọi thủ tục CALL. Để khi chấm dứt việc thi hành thủ tục thì chương trình gọi được tiếp tục bình thường, ta cần lưu giữ địa chỉ trở về tức địa chỉ của lệnh sau lệnh gọi thủ tục CALL. Khi chấm dứt thi hành thủ tục, lệnh trở về RETURN nạp địa chỉ trở về vào PC.

Trong các kiến trúc CISC (VAX 11, 80x86, 680x0), địa chỉ trở về được giữ ở ngăn xếp. Trong các kiến trúc RISC, một thanh ghi đặc biệt (thường là thanh ghi R31) được dùng để lưu giữ địa chỉ trở về.

Lệnh gọi thủ tục là một lệnh loại JMPL Ri, lệnh này làm các tác vụ :

```
R31 := PC ; để địa chỉ trở về trong R31
PC := Ri ; nhảy tới địa chỉ của thủ tục nằm trong thanh ghi Ri
```

Lệnh trở về khi chấm dứt thủ tục là JMP R31, vì thanh ghi R31 chứa địa chỉ trở về.

Việc dùng một thanh ghi đặc biệt để lưu trữ địa chỉ trở về là một giải pháp chỉ áp dụng cho các thủ tục cuối cùng, nghĩa là cho thủ tục không gọi thủ tục nào cả. Để có thể cho các thủ tục có thể gọi một thủ tục khác, ta có hai giải pháp:

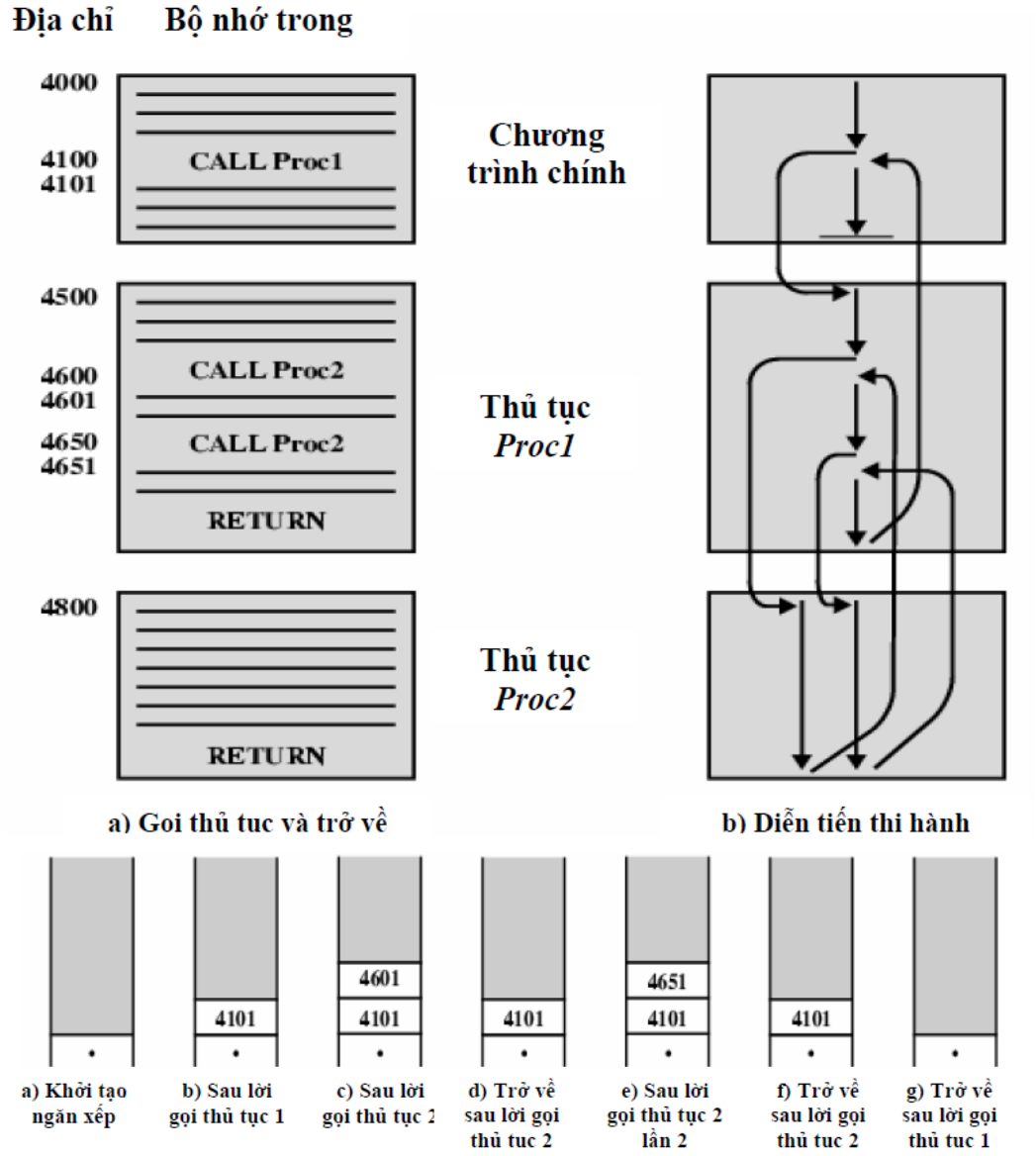
- Giải pháp 1: có nhiều thanh ghi để lưu trữ địa chỉ trở về
- Giải pháp 2: lưu giữ địa chỉ trở về ở ngăn xếp.

Việc gọi thủ tục có thể được thực hiện bằng chuỗi lệnh sau đây :

```
ADDI R30, R30,4 ; R30 là con trỏ ngăn xếp
STORE R31, (R30) ; lưu giữ địa chỉ trở về
JMPL Ri ; gọi thủ tục
```

Người ta dùng chuỗi lệnh sau đây để trở về chương trình gọi :

```
LOAD R31, (R30) ; phục hồi địa chỉ trở về
SUBI R30, R30,4 ; cập nhật con trỏ ngăn xếp
JMP R31 ; trở về chương trình gọi
```



Hình 2.3. Gọi thủ tục và trở về khi thực hiện xong thủ tục

Việc truyền tham số từ thủ tục gọi đến thủ tục bị gọi có thể thực hiện bằng cách dùng các thanh ghi của bộ xử lý hoặc dùng ngăn xếp. Nếu số tham số cần truyền ít, ta dùng các thanh ghi.

2.4. KIẾN TRÚC RISC VÀ CISC

Các kiến trúc với tập lệnh phức tạp CISC (Complex Instruction Set Computer) được nghĩ ra từ những năm 1960. Vào thời kỳ này, người ta nhận thấy các chương trình dịch khó dùng các thanh ghi, rằng các vi lệnh được thực hiện nhanh hơn các lệnh và cần thiết phải làm giảm độ dài các chương trình. Các đặc tính nầy khiến người ta ưu tiên chọn các kiểu ô nhớ - ô nhớ và ô nhớ - thanh ghi, với những lệnh phức tạp và dùng nhiều kiểu định vị. Điều này dẫn tới việc các lệnh có chiều dài thay đổi và như thế thì dùng bộ điều khiển vi chương trình là hiệu quả nhất.

Bộ xử lý	IBM 370/168	DEC 11/780	iAPX 432
Năm sản xuất	1973	1978	1982
Số lệnh	208	303	222
Bộ nhớ vi chương trình	420KB	480KB	64KB
Chiều dài lệnh (bits)	16 – 48	16 – 456	6 – 321
Kỹ thuật chế tạo	ECL – MSI	TTL – MSI	NMOS VLSI
Cách thực hiện lệnh	Thanh ghi – thanh ghi Thanh ghi – bộ nhớ Bộ nhớ - bộ nhớ	Thanh ghi – thanh ghi Thanh ghi – bộ nhớ Bộ nhớ - bộ nhớ	Ngăn xếp Bộ nhớ - bộ nhớ
Dung lượng cache	64KB	64KB	0

Bảng 2.1. Đặc tính của một vài máy CISC

Bảng trên cho biết các đặc tính của vài máy CISC tiêu biểu. Ta nhận thấy cả ba máy đều có điểm chung là có nhiều lệnh, các lệnh có chiều dài thay đổi. Nhiều cách thực hiện lệnh và nhiều vi chương trình được dùng.

Tiến bộ trong lãnh vực mạch kết (IC) và kỹ thuật dịch chương trình làm cho các nhận định trước đây phải được xem xét lại, nhất là khi đã có một khảo sát định lượng về việc dùng tập lệnh các máy CISC.

Ví dụ, chương trình dịch đã biết sử dụng các thanh ghi và không có sự khác biệt đáng kể nào khi sử dụng ô nhớ cho các vi chương trình hay ô nhớ cho các chương trình. Điều này dẫn tới việc đưa vào khái niệm về một máy tính với tập lệnh rút gọn RISC vào đầu những năm 1980. Các máy RISC dựa chủ yếu trên một tập lệnh cho phép thực hiện kỹ thuật ống dẫn một cách thích hợp nhất bằng cách thiết kế các lệnh có chiều dài cố định, có dạng đơn giản, dễ giải mã. Máy RISC dùng kiểu thực hiện lệnh thanh ghi - thanh ghi. Chỉ có các lệnh ghi hoặc đọc ô nhớ mới cho phép thâm nhập vào ô nhớ. Bảng II.7 diễn tả ba mẫu máy RISC đầu tiên: mẫu máy của IBM (IBM 801) của Berkeley (RISC1 của Patterson) và của Stanford (MIPS của Hennessy). Ta nhận thấy cả ba máy đó đều có bộ điều khiển bằng mạch điện (không có ô nhớ vi chương trình), có chiều dài các lệnh cố định (32 bits), có một kiểu thi hành lệnh (kiểu thanh ghi - thanh ghi) và chỉ có một số ít lệnh.

Bộ xử lý	IBM 801	RISC1	MIPS
Năm sản xuất	1980	1982	1983
Số lệnh	120	39	55
Bộ nhớ vi chương trình	0	0	0

Chiều dài lệnh (bits)	32	32	32
Kỹ thuật chế tạo	ECL – MSI	NMOS VLSI	NMOS VLSI
Cách thực hiện lệnh	Thanh ghi – thanh ghi	Thanh ghi – thanh ghi	Thanh ghi – thanh ghi

Bảng 2.2. Đặc tính của của ba mẫu đầu tiên máy RISC

Tóm lại, ta có thể định nghĩa mạch xử lý RISC bởi các tính chất sau:

- Có một số ít lệnh (thông thường dưới 100 lệnh).
- Có một số ít các kiểu định vị (thông thường hai kiểu: định vị tức thì và định vị gián tiếp thông qua một thanh ghi).
- Có một số ít dạng lệnh (một hoặc hai)
- Các lệnh đều có cùng chiều dài.
- Chỉ có các lệnh ghi hoặc đọc ô nhớ mới thâm nhập vào bộ nhớ.
- Dùng bộ tạo tín hiệu điều khiển bằng mạch điện để tránh chu kỳ giải mã các vi lệnh làm cho thời gian thực hiện lệnh kéo dài.
- Bộ xử lý RISC có nhiều thanh ghi để giảm bớt việc thâm nhập vào bộ nhớ trong.
- Ngoài ra các bộ xử lý RISC đầu tiên thực hiện tất cả các lệnh trong một chu kỳ máy.

Bộ xử lý RISC có các ưu điểm sau:

- Diện tích của bộ xử lý dùng cho bộ điều khiển giảm từ 60% (cho các bộ xử lý CISC) xuống còn 10% (cho các bộ xử lý RISC). Như vậy có thể tích hợp thêm vào bên trong bộ xử lý các thanh ghi, các cổng vào ra và bộ nhớ cache, …
- Tốc độ tính toán cao nhờ vào việc giải mã lệnh đơn giản, nhờ có nhiều thanh ghi (ít thâm nhập bộ nhớ), và nhờ thực hiện kỹ thuật ống dẫn liên tục và có hiệu quả (các lệnh đều có thời gian thực hiện giống nhau và có cùng dạng).
- Thời gian cần thiết để thiết kế bộ điều khiển là ít. Điều này góp phần làm giảm chi phí thiết kế.
- Bộ điều khiển trở nên đơn giản và gọn làm cho ít rủi ro mắc phải sai sót mà ta gặp thường trong bộ điều khiển.

Kiến trúc RISC có một số bất lợi:

- Các chương trình dài ra so với chương trình viết cho bộ xử lý CISC. Điều này do các nguyên nhân sau:
    - Cấm thâm nhập bộ nhớ đối với tất cả các lệnh ngoại trừ các lệnh đọc và ghi vào bộ nhớ. Do đó ta buộc phải dùng nhiều lệnh để làm một công việc nhất định.
    - Cần thiết phải tính các địa chỉ hiệu dụng vì không có nhiều cách định vị.
    - Tập lệnh có ít lệnh nên các lệnh không có sẵn phải được thay thế bằng một chuỗi lệnh của bộ xử lý RISC.

- Các chương trình dịch gặp nhiều khó khăn vì có ít lệnh làm cho có ít lựa chọn để diễn dịch các cấu trúc của chương trình gốc. Sự cứng nhắc của kỹ thuật ống dẫn cũng gây khó khăn.
- Có ít lệnh trợ giúp cho ngôn ngữ cấp cao.

Các bộ xử lý CISC trợ giúp mạnh hơn các ngôn ngữ cao cấp nhờ có tập lệnh phức tạp. Hãng Honeywell đã chế tạo một máy có một lệnh cho mỗi động từ của ngôn ngữ COBOL.

Các tiến bộ gần đây cho phép xếp đặt trong một vi mạch, một bộ xử lý RISC nền và nhiều toán tử chuyên dùng.

Thí dụ, bộ xử lý 860 của Intel bao gồm một bộ xử lý RISC, bộ làm tính với các số lẻ và một bộ tạo tín hiệu đồ hoạ.

2.5. TOÁN HẠNG

Kiểu của toán hạng thường được đưa vào trong mã tác vụ của lệnh. Có bốn kiểu toán hạng được dùng trong các hệ thống:

- Kiểu địa chỉ.
- Kiểu dạng số: số nguyên, dấu chấm động,...
- Kiểu dạng chuỗi ký tự: ASCII, EBIDEC,...
- Kiểu dữ liệu logic: các bit, cờ,...

Tuy nhiên một số ít máy tính dùng các nhãn để xác định kiểu toán hạng.

Thông thường loại của toán hạng xác định luôn chiều dài của nó. Toán hạng thường có chiều dài là byte (8 bit), nữa từ máy tính (16 bit), từ máy tính (32 bit), từ đôi máy tính (64 bit). Đặc biệt, kiến trúc PA của hãng HP (Hewlet Packard) có khả năng tính toán với các số thập phân BCD. Một vài bộ xử lý có thể xử lý các chuỗi ký tự.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Các thành phần của một hệ máy tính đơn giản
2. Nhiệm vụ của mỗi bus trong hệ thống bus của một hệ máy tính đơn giản? Tại sao trong thực tế cần có một hệ thống bus vào ra?
3. Sự khác biệt giữa CPU RISC và CPU CISC?

Chương 3

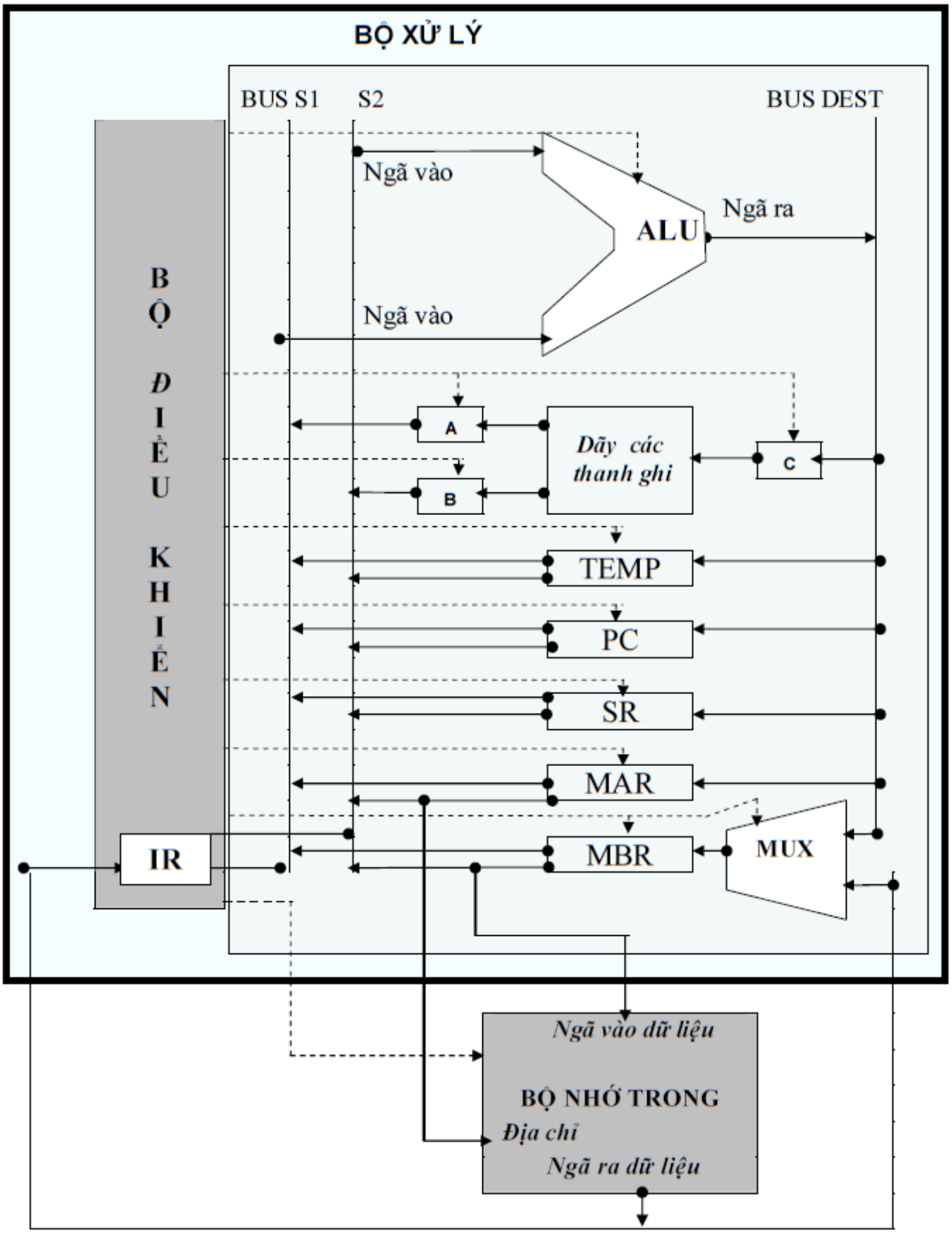
TỔ CHỨC BỘ XỬ LÝ

3.1. ĐƯỜNG ĐI DỮ LIỆU

Phần đường đi dữ liệu gồm có bộ phận làm tính và luận lý (ALU: Arithmetic and Logic Unit), các mạch dịch, các thanh ghi và các đường nối kết các bộ phận trên. Phần này chứa hầu hết các trạng thái của bộ xử lý. Ngoài các thanh ghi tổng quát, phần đường đi dữ liệu còn chứa thanh ghi đếm chương trình (PC: Program Counter), thanh ghi trạng thái (SR: Status Register), thanh ghi đệm TEMP (Temporary), các thanh ghi địa chỉ bộ nhớ (MAR: Memory Address Register), thanh ghi số liệu bộ nhớ (MBR: Memory Buffer Register), bộ đa hợp (MUX: Multiplexor), đây là điểm cuối của các kênh dữ liệu - CPU và bộ nhớ, với nhiệm vụ lập thời biểu truy cập bộ nhớ từ CPU và các kênh dữ liệu, hệ thống bus nguồn (S1, S2) và bus kết quả (Dest).

Nhiệm vụ chính của phần đường đi dữ liệu là đọc các toán hạng từ các thanh ghi tổng quát, thực hiện các phép tính trên toán hạng này trong bộ làm tính và luận lý ALU và lưu trữ kết quả trong các thanh ghi tổng quát. Ở ngã vào và ngã ra các thanh ghi tổng quát có các mạch chốt A, B, C. Thông thường, số lượng các thanh ghi tổng quát là 32.

Phần đường đi của dữ liệu chiếm phân nửa diện tích của bộ xử lý nhưng là phần dễ thiết kế và cài đặt trong bộ xử lý.



Hình 3.1. Tổ chức của một bộ xử lý điển hình
(Các đường không liên tục là các đường tín hiệu điều khiển)

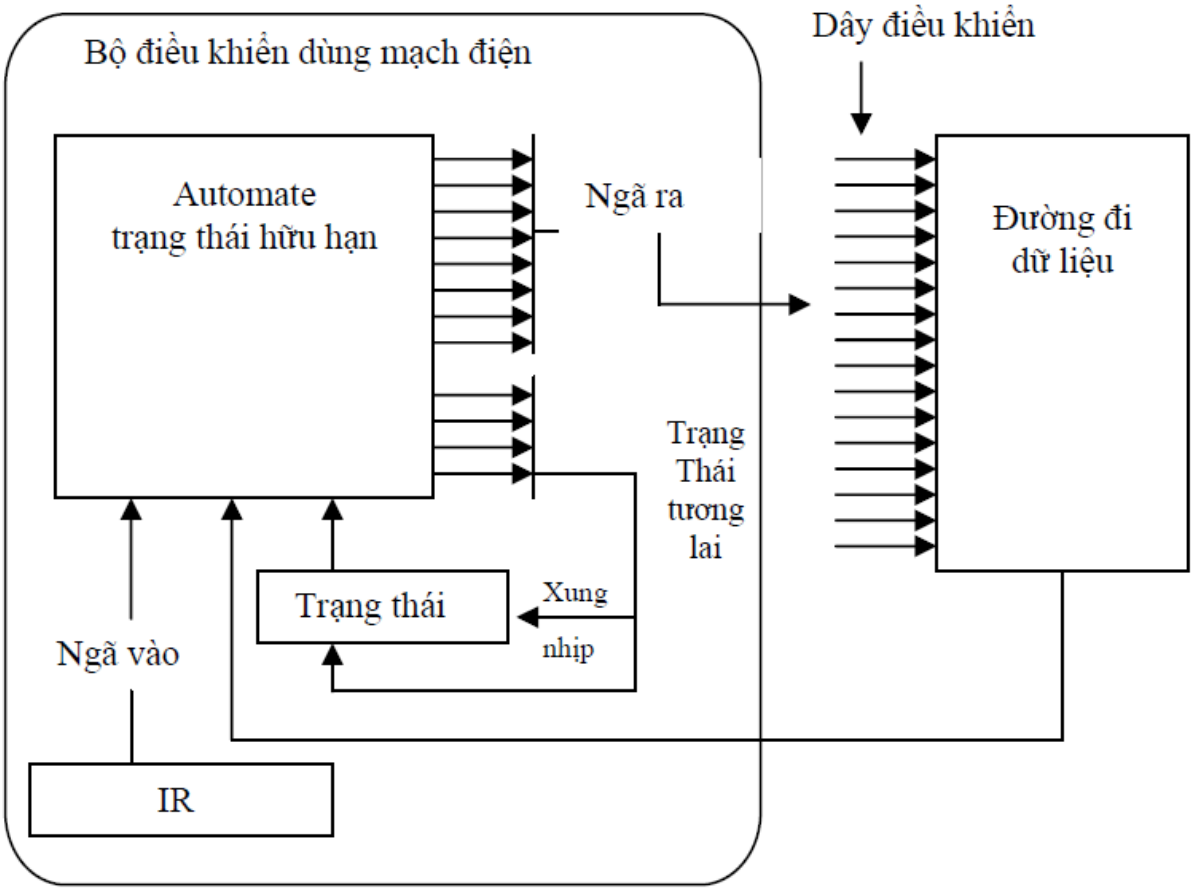
3.2. BỘ ĐIỀU KHIỂN

Bộ điều khiển tạo các tín hiệu điều khiển di chuyển số liệu (tín hiệu di chuyển số liệu từ các thanh ghi đến bus hoặc tín hiệu viết vào các thanh ghi), điều khiển các tác vụ mà các bộ phận chức năng phải làm (điều khiển ALU, điều khiển đọc và viết vào bộ nhớ trong...). Bộ điều khiển cũng tạo các tín hiệu giúp các lệnh được thực hiện một cách tuần tự.

Việc cài đặt bộ điều khiển có thể dùng một trong hai cách sau: dùng mạch điện tử hoặc dùng vi chương trình (microprogram).

3.2.1. Bộ điều khiển mạch điện tử

Để hiểu được vận hành của bộ điều khiển mạch điện tử, chúng ta xét đến mô tả về Automate trạng thái hữu hạn: có nhiều hệ thống hay nhiều thành phần mà ở mỗi thời điểm xem xét đều có một trạng thái (state). Mục đích của trạng thái là ghi nhớ những gì có liên quan trong quá trình hoạt động của hệ thống. Vì chỉ có một số trạng thái nhất định nên nói chung không thể ghi nhớ hết toàn bộ lịch sử của hệ thống, do vậy nó phải được thiết kế cẩn thận để ghi nhớ những gì quan trọng. Ưu điểm của hệ thống (chỉ có một số hữu hạn các trạng thái) đó là có thể cài đặt hệ thống với một lượng tài nguyên cố định. Chẳng hạn, chúng ta có thể cài đặt Automate trạng thái hữu hạn trong phần cứng máy tính ở dạng mạch điện hay một dạng chương trình đơn giản, trong đó, nó có khả năng quyết định khi chỉ biết một lượng giới hạn dữ liệu hoặc bằng cách dùng vị trí trong đoạn mã lệnh để đưa ra quyết định.



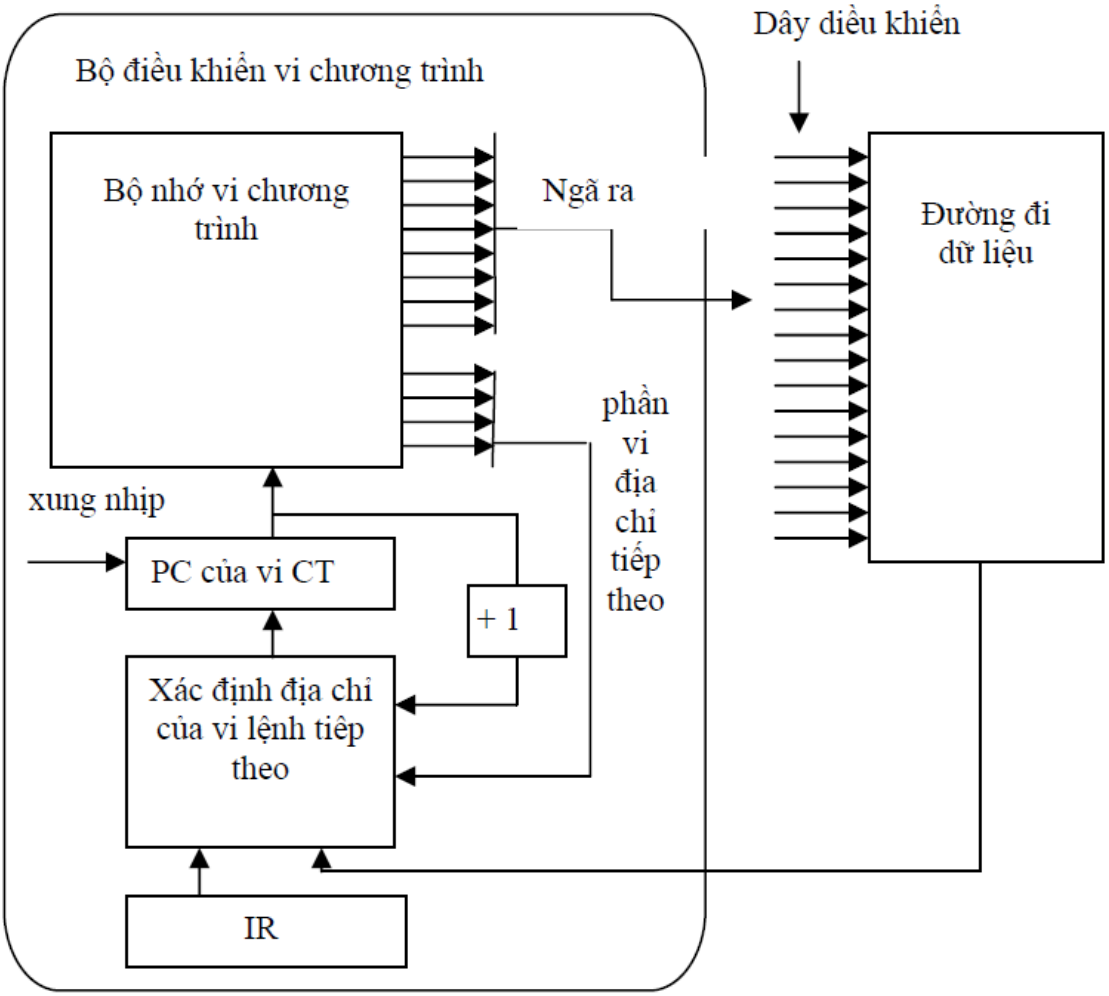
Hình 3.2. Nguyên tắc vận hành của bộ điều khiển dùng mạch điện

Hình trên cho thấy nguyên tắc của một bộ điều khiển bằng mạch điện. Các đường điều khiển của phần đường đi số liệu là các ngã ra của một hoặc nhiều Automate trạng

thái hữu hạn. Các ngã vào của Automate gồm có thanh ghi lệnh, thanh ghi này chứa lệnh phải thi hành và những thông tin từ bộ đường đi số liệu. Ứng với cấu hình các đường vào và trạng thái hiện tại, Automate sẽ cho trạng thái tương lai và các đường ra tương ứng với trạng thái hiện tại. Automate được cài đặt dưới dạng là một hay nhiều mạch mảng logic lập trình được (PLA: Programmable Logic Array) hoặc các mạch logic ngẫu nhiên.

Kỹ thuật điều khiển này đơn giản và hữu hiệu khi các lệnh có chiều dài cố định, có dạng thức đơn giản. Nó được dùng nhiều trong các bộ xử lý RISC.

3.2.2. Bộ điều khiển vi chương trình



Hình 3.3. Nguyên tắc vận hành của bộ điều khiển vi chương trình

Sơ đồ nguyên tắc của bộ điều khiển dùng vi chương trình được trình bày ở hình 3.3. Trong kỹ thuật này, các đường dây điều khiển của bộ đường đi dữ liệu ứng với các ngã ra của một vi lệnh nằm trong bộ nhớ vi chương trình. Việc điều khiển các tác vụ của một lệnh mã máy được thực hiện bằng một chuỗi các vi lệnh. Một vi máy tính nằm bên trong bộ điều khiển thực hiện từng lệnh của vi chương trình này. Chính vi máy tính này điều khiển việc thực hiện một cách tuần tự các vi lệnh để hoàn thành tác vụ mà lệnh mã máy phải thực hiện. Các tác vụ của lệnh mã máy cũng tùy thuộc vào trạng thái của phần đường đi dữ liệu.

Bộ điều khiển bằng vi chương trình được dùng rộng rãi trong các bộ xử lý CISC. Bộ xử lý này có tập lệnh phức tạp với các lệnh có chiều dài khác nhau và có dạng thức phức tạp. Trong các bộ xử lý CISC, người ta cài đặt một lệnh mã máy bằng cách viết

một vi chương trình. Như vậy công việc khá đơn giản và rất hữu hiệu. Các sai sót trong thiết kế automat điều khiển cũng dễ sửa đổi.

3.3. DIỄN TIẾN THI HÀNH LỆNH MÃ MÁY

Việc thi hành một lệnh mã máy có thể chia thành 5 giai đoạn:

1. Đọc lệnh (IF: Instruction Fetch)
2. Giải mã lệnh (ID: Instruction Decode)
3. Thi hành lệnh (EX: Execute)
4. Thâm nhập bộ nhớ trong hoặc nhảy (MEM: Memory access)
5. Lưu trữ kết quả (RS: Result Storing)

Mỗi giai đoạn được thi hành trong một hoặc nhiều chu kỳ xung nhịp.

3.3.1. Đọc lệnh

MAR ← PC

IR ← M[MAR]

Bộ đếm chương trình PC được đưa vào MAR. Lệnh được đọc từ bộ nhớ trong, tại các ô nhớ có địa chỉ nằm trong MAR và được đưa vào thanh ghi lệnh IR.

3.3.2. Giải mã lệnh và đọc các thanh ghi nguồn

A ← Rs1

B ← Rs2

PC ← PC + 4

Lệnh được giải mã. Kế đó các thanh ghi Rs1 và Rs2 được đưa vào A và B. Thanh ghi PC được tăng lên để chỉ tới lệnh kế đó.

Để hiểu rõ giai đoạn này, ta lấy dạng thức của một lệnh làm tính tiêu biểu sau đây:

	Mã lệnh	Thanh ghi Rs1	Thanh ghi Rs2	Thanh ghi Rd	Tác vụ
bit	6	5	5	5	11

Các thanh ghi nguồn Rs1 và Rs2 được sử dụng tùy theo tác vụ, kết quả được đặt trong thanh ghi đích Rd.

Ta thấy việc giải mã được thực hiện cùng lúc với việc đọc các thanh ghi Rs1 và Rs2 vì các thanh ghi này luôn nằm tại cùng vị trí ở trong lệnh.

3.3.3. Thi hành lệnh

Tùy theo loại lệnh mà một trong ba nhiệm vụ sau đây được thực hiện:

Liên hệ tới bộ nhớ

MAR ← Địa chỉ do ALU tính tùy theo kiểu định vị (Rs2).

MBR ← Rs1

Địa chỉ hiệu dụng do ALU tính được đưa vào MAR và thanh ghi nguồn Rs1 được đưa vào MBR để được lưu vào bộ nhớ trong.

Một lệnh của ALU

Ngã ra ALU ← Kết quả của phép tính

ALU thực hiện phép tính xác định trong mã lệnh, đưa kết quả ra ngã ra.

Một phép nhảy

Ngã ra ALU ← Địa chỉ lệnh tiếp theo do ALU tính.

ALU cộng địa chỉ của PC với độ dời để làm thành địa chỉ đích và đưa địa chỉ này ra ngã ra. Nếu là một phép nhảy có điều kiện thì thanh ghi trạng thái được đọc quyết định có cộng độ dời vào PC hay không.

3.3.4. Thâm nhập bộ nhớ trong hoặc nhảy lần cuối

Giai đoạn này thường chỉ được dùng cho các lệnh nạp dữ liệu, lưu giữ dữ liệu và lệnh nhảy.

Tham khảo đến bộ nhớ

MBR ← M[MAR] hoặc M[MAR] ← MBR

Số liệu được nạp vào MBR hoặc lưu vào địa chỉ mà MAR trỏ đến.

Nhảy

If (điều kiện), PC ← ngả ra ALU

Nếu điều kiện đúng, ngã ra ALU được nạp vào PC. Đối với lệnh nhảy không điều kiện, ngã ra ALU luôn được nạp vào thanh ghi PC.

3.3.5. Lưu trữ kết quả

Rd ← Ngã ra ALU hoặc Rd ← MBR

Lưu trữ kết quả trong thanh ghi đích.

3.4. NGẮT QUÃNG (INTERRUPT)

Ngắt quãng là một sự kiện xảy ra một cách ngẫu nhiên trong máy tính và làm ngưng tính tuần tự của chương trình (nghĩa là tạo ra một lệnh nhảy). Phần lớn các nhà sản xuất máy tính (ví dụ như IBM, INTEL) dùng từ ngắt quãng để ám chỉ sự kiện này, tuy nhiên một số nhà sản xuất khác dùng từ "ngoại lệ", "lỗi", "bẩy" để chỉ định hiện tượng này.

Bộ điều khiển của CPU là bộ phận khó thực hiện nhất và ngắt quãng là phần khó thực hiện nhất trong bộ điều khiển. Để nhận biết được một ngắt quãng lúc đang thi hành một lệnh, ta phải biết điều chỉnh chu kỳ xung nhịp và điều này có thể ảnh hưởng đến hiệu quả của máy tính.

Người ta đã nghỉ ra "ngắt quãng" là để nhận biết các sai sót trong tính toán số học, và để ứng dụng cho những hiện tượng thời gian thực. Bây giờ, ngắt quãng được dùng cho các công việc sau đây:

- Ngoại vi đòi hỏi nhập hoặc xuất số liệu.
- Người lập trình muốn dùng dịch vụ của hệ điều hành.
- Cho một chương trình chạy từng lệnh.
- Làm điểm dừng của một chương trình.
- Báo tràn số liệu trong tính toán số học.
- Trang bộ nhớ thực sự không có trong bộ nhớ.

- Báo vi phạm vùng cấm của bộ nhớ.
- Báo dùng một lệnh không có trong tập lệnh.
- Báo phần cứng máy tính bị hư.
- Báo điện bị cắt.

Dù rằng ngắt quãng không xảy ra thường xuyên nhưng bộ xử lý phải được thiết kế sao cho có thể lưu giữ trạng thái của nó trước khi nhảy đi phục vụ ngắt quãng. Sau khi thực hiện xong chương trình phục vụ ngắt, bộ xử lý phải khôi phục trạng thái của nó để có thể tiếp tục công việc.

Để đơn giản việc thiết kế, một vài bộ xử lý chỉ chấp nhận ngắt sau khi thực hiện xong lệnh đang chạy. Khi một ngắt xảy ra, bộ xử lý thi hành các bước sau đây:

1. Thực hiện xong lệnh đang làm
2. Lưu trữ trạng thái hiện tại
3. Nhảy đến chương trình phục vụ ngắt
4. Khi chương trình phục vụ chấm dứt, bộ xử lý khôi phục lại trạng thái cũ của nó và tiếp tục thực hiện chương trình mà nó đang thực hiện khi bị ngắt.

3.5. KỸ THUẬT ỐNG DẪN

Đây là một kỹ thuật làm cho các giai đoạn khác nhau của nhiều lệnh được thi hành cùng một lúc.

Ví dụ: Chúng ta có những lệnh đều đặn, mỗi lệnh được thực hiện trong cùng một khoản thời gian. Giả sử, mỗi lệnh được thực hiện trong 5 giai đoạn và mỗi giai đoạn được thực hiện trong 1 chu kỳ xung nhịp. Các giai đoạn thực hiện một lệnh là: lấy lệnh (IF: Instruction Fetch), giải mã (ID: Instruction Decode), thi hành (EX: Execute), thâm nhập bộ nhớ (MEM: Memory Access), lưu trữ kết quả (RS: Result Storing).

Hình 3.4 cho thấy chỉ trong một chu kỳ xung nhịp, bộ xử lý có thể thực hiện một lệnh (bình thường lệnh này được thực hiện trong 5 chu kỳ).

Chuỗi lệnh	**Chu kỳ xung nhịp**								
	1	2	3	4	5	6	7	8	9
Lệnh thứ i	IF	ID	EX	MEM	RS				
Lệnh thứ i+1		IF	ID	EX	MEM	RS			
Lệnh thứ i+2			IF	ID	EX	MEM	RS		
Lệnh thứ i+3				IF	ID	EX	MEM	RS	
Lệnh thứ i+4					IF	ID	EX	MEM	RS

Hình 3.4. Các giai đoạn khác nhau của nhiều lệnh được thi hành cùng một lúc

So sánh với kiểu xử lý tuần tự thông thường, 5 lệnh được thực hiện trong 25 chu kỳ xung nhịp, thì xử lý lệnh theo kỹ thuật ống dẫn thực hiện 5 lệnh chỉ trong 9 chu kỳ xung nhịp.

Như vậy kỹ thuật ống dẫn làm tăng tốc độ thực hiện các lệnh. Tuy nhiên kỹ thuật ống dẫn có một số ràng buộc:

- Cần phải có một mạch điện để thi hành mỗi giai đoạn của lệnh vì tất cả các giai đoạn của lệnh được thi hành cùng lúc. Trong một bộ xử lý không dùng kỹ thuật ống dẫn, ta có thể dùng bộ làm toán ALU để cập nhật thanh ghi PC, cập nhật địa chỉ của toán hạng bộ nhớ, địa chỉ ô nhớ mà chương trình cần nhảy tới, làm các phép tính trên các toán hạng vì các phép tính này có thể xảy ra ở nhiều giai đoạn khác nhau.
- Phải có nhiều thanh ghi khác nhau dùng cho các tác vụ đọc và viết. Trên hình 3.4, tại một chu kỳ xung nhịp, ta thấy cùng một lúc có 2 tác vụ đọc (ID, MEM) và 1 tác vụ viết (RS).
- Trong một máy có kỹ thuật ống dẫn, có khi kết quả của một tác vụ trước đó, là toán hạng nguồn của một tác vụ khác. Như vậy sẽ có thêm những khó khăn mà ta sẽ đề cập ở mục tới.
- Cần phải giải mã các lệnh một cách đơn giản để có thể giải mã và đọc các toán hạng trong một chu kỳ duy nhất của xung nhịp.
- Cần phải có các bộ làm tính ALU hữu hiệu để có thể thi hành lệnh số học dài nhất, có số giữ, trong một khoảng thời gian ít hơn một chu kỳ của xung nhịp.
- Cần phải có nhiều thanh ghi lệnh để lưu giữ lệnh mà chúng ta phải xem xét cho mỗi giai đoạn thi hành lệnh.
- Cuối cùng phải có nhiều thanh ghi bộ đếm chương trình PC để có thể tiếp tục các lệnh trong trường hợp có ngắt quãng.

3.6. KHÓ KHĂN TRONG KỸ THUẬT ỐNG DẪN

Khi thi hành lệnh trong một máy tính dùng kỹ thuật ống dẫn, có nhiều trường hợp làm cho việc thực hiện kỹ thuật ống dẫn không thực hiện được như là: thiếu các mạch chức năng, một lệnh dùng kết quả của lệnh trước, một lệnh nhảy.

Ta có thể phân biệt 3 loại khó khăn: khó khăn do cấu trúc, khó khăn do số liệu và khó khăn do điều khiển.

3.6.1. Khó khăn do cấu trúc

Đây là khó khăn do thiếu bộ phận chức năng, ví dụ trong một máy tính dùng kỹ thuật ống dẫn phải có nhiều ALU, nhiều PC, nhiều thanh ghi lệnh IR ... Các khó khăn này được giải quyết bằng cách thêm các bộ phận chức năng cần thiết và hữu hiệu.

3.6.2. Khó khăn do số liệu

Lấy ví dụ trường hợp các lệnh liên tiếp sau:

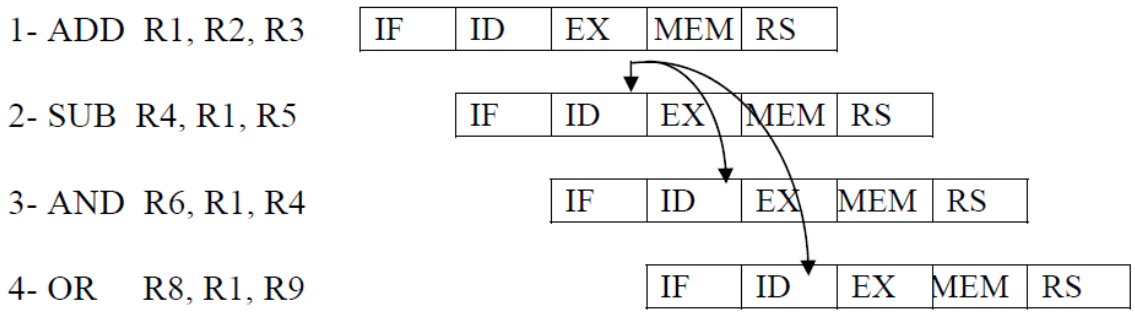
Lệnh 1: **ADD R1, R2, R3**

Lệnh 2: **SUB R4, R1, R5**

Lệnh 3: **AND R6, R1, R7**

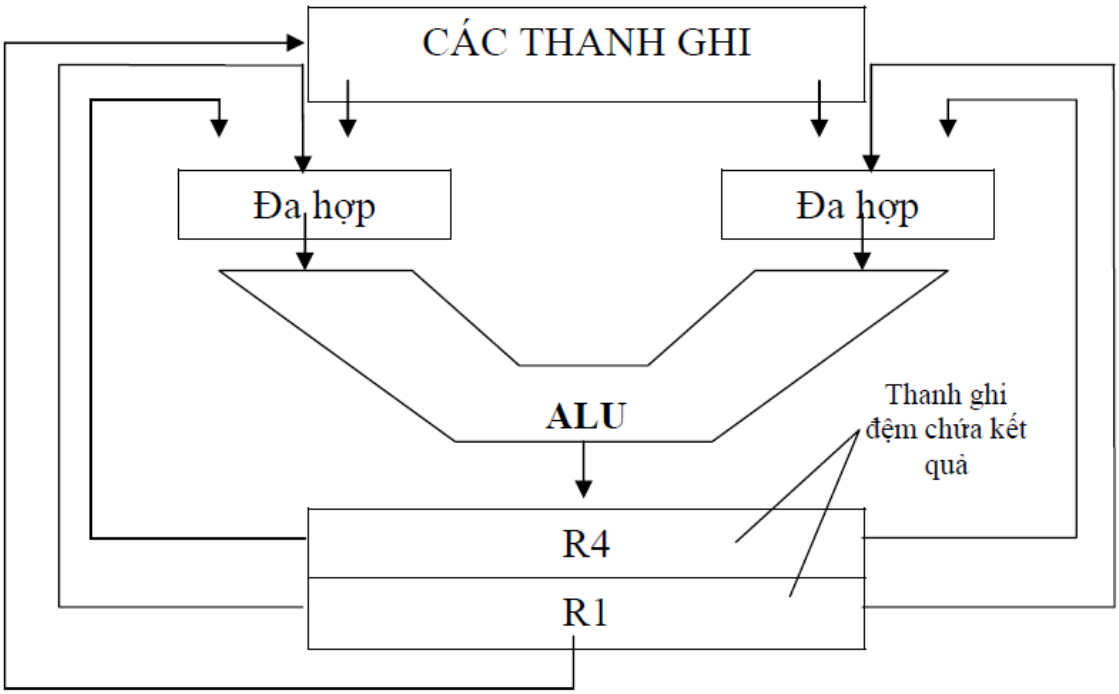
Lệnh 4: **OR R8, R1, R9**

Hình 3.5 cho thấy R1, kết quả của lệnh 1 chỉ có thể được dùng cho lệnh 2 sau giai đoạn MEM của lệnh 1, nhưng R1 được dùng cho lệnh 2 vào giai đoạn EX của lệnh 1. Chúng ta cũng thấy R1 được dùng cho các lệnh 3 và 4.



Hình 3.5. Chuỗi lệnh minh họa khó khăn do số liệu

Để khắc phục khó khăn này, một bộ phận phần cứng được dùng để đưa kết quả từ ngã ra ALU trực tiếp vô một trong các thanh ghi ngã vào như trong hình 3.6.



Hình 3.6. ALU với bộ phận phần cứng đưa kết quả tính toán trở lại ngã vào

Khi bộ phận phần cứng nêu trên phát hiện có dùng kết quả của ALU làm toán hạng cho liệt kê, nó tác động vào mạch đa hợp để đưa ngã ra của ALU vào ngã vào của ALU hoặc vào ngã vào của một đơn vị chức năng khác nếu cần.

3.6.3. Khó khăn do điều khiển

Các lệnh làm thay đổi tính thi hành các lệnh một cách tuần tự (nghĩa là PC tăng đều đặn sau mỗi lệnh), gây khó khăn về điều khiển. Các lệnh này là lệnh nhảy đến một địa chỉ tuyệt đối chứa trong một thanh ghi, hay lệnh nhảy đến một địa chỉ xác định một cách tương đối so với địa chỉ hiện tại của bộ đếm chương trình PC. Các lệnh nhảy trên có thể có hoặc không điều kiện.

Trong trường hợp đơn giản nhất, tác vụ nhảy không thể biết trước giai đoạn giải mã (xem hình 3.4). Như vậy, nếu lệnh nhảy bắt đầu ở chu kỳ C thì lệnh mà chương trình nhảy tới chỉ được bắt đầu ở chu kỳ C+2. Ngoài ra, phải biết địa chỉ cần nhảy đến mà ta có ở cuối giai đoạn giải mã ID. Trong lệnh nhảy tương đối, ta phải cộng độ dời chứa trong thanh ghi lệnh IR vào thanh ghi PC. Việc tính địa chỉ này chỉ được thực hiện vào giai đoạn ID với điều kiện phải có một mạch công việc riêng biệt.

Vậy trong trường hợp lệnh nhảy không điều kiện, lệnh mà chương trình nhảy đến bắt đầu thực hiện ở chu kỳ C+2 nếu lệnh nhảy bắt đầu ở chu kỳ C.

Cho các lệnh nhảy có điều kiện thì phải tính toán điều kiện. Thông thường các kiến trúc RISC đặt kết quả việc so sánh vào trong thanh ghi trạng thái, hoặc vào trong thanh ghi tổng quát. Trong cả 2 trường hợp, đọc điều kiện tương đương với đọc thanh ghi. Đọc thanh ghi có thể được thực hiện trong phân nửa chu kỳ cuối giai đoạn ID.

Một trường hợp khó hơn có thể xảy ra trong những lệnh nhảy có điều kiện. Đó là điều kiện được có khi so sánh 2 thanh ghi và chỉ thực hiện lệnh nhảy khi kết quả so sánh là đúng. Việc tính toán trên các đại lượng logic không thể thực hiện được trong phân nửa chu kỳ và như thế phải kéo dài thời gian thực hiện lệnh nhảy có điều kiện. Người ta thường tránh các trường hợp này để không làm giảm mức hữu hiệu của máy tính.

Vậy trường hợp đơn giản, người ta có thể được địa chỉ cần nhảy đến và điều kiện nhảy cuối giai đoạn ID. Vậy có chậm đi một chu kỳ mà người ta có thể giải quyết bằng nhiều cách.

Cách thứ nhất là đóng băng kỹ thuật ống dẫn trong một chu kỳ, nghĩa là ngưng thi hành lệnh thứ i+1 đang làm nếu lệnh thư i là lệnh nhảy. Ta mất trắng một chu kỳ cho mỗi lệnh nhảy.

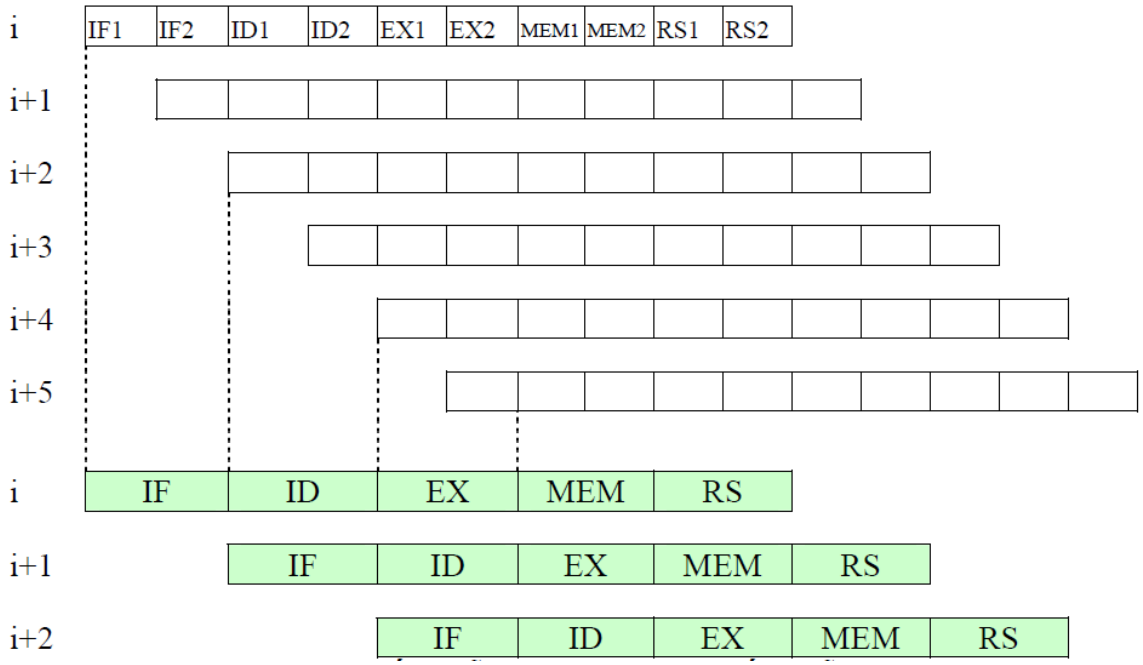
Cách thứ hai là thi hành lệnh sau lệnh nhảy nhưng lưu ý rằng hiệu quả của một lệnh nhảy bị chậm mất một lệnh. Vậy lệnh theo sau lệnh nhảy được thực hiện trước khi lệnh mà chương trình phải nhảy tới được thực hiện. Chương trình dịch hay người lập trình có nhiệm vụ xen vào một lệnh hữu ích sau lệnh nhảy.

Trong trường hợp nhảy có điều kiện, việc nhảy có thể được thực hiện hay không thực hiện. Lệnh hữu ích đặt sau lệnh nhảy không làm sai lệch chương trình dù điều kiện nhảy đúng hay sai.

Bộ xử lý RISC SPARC có những lệnh nhảy với hủy bỏ. Các lệnh này cho phép thi hành lệnh sau lệnh nhảy nếu điều kiện nhảy đúng và hủy bỏ thực hiện lệnh đó nếu điều kiện nhảy sai.

3.7. SIÊU ỐNG DẪN

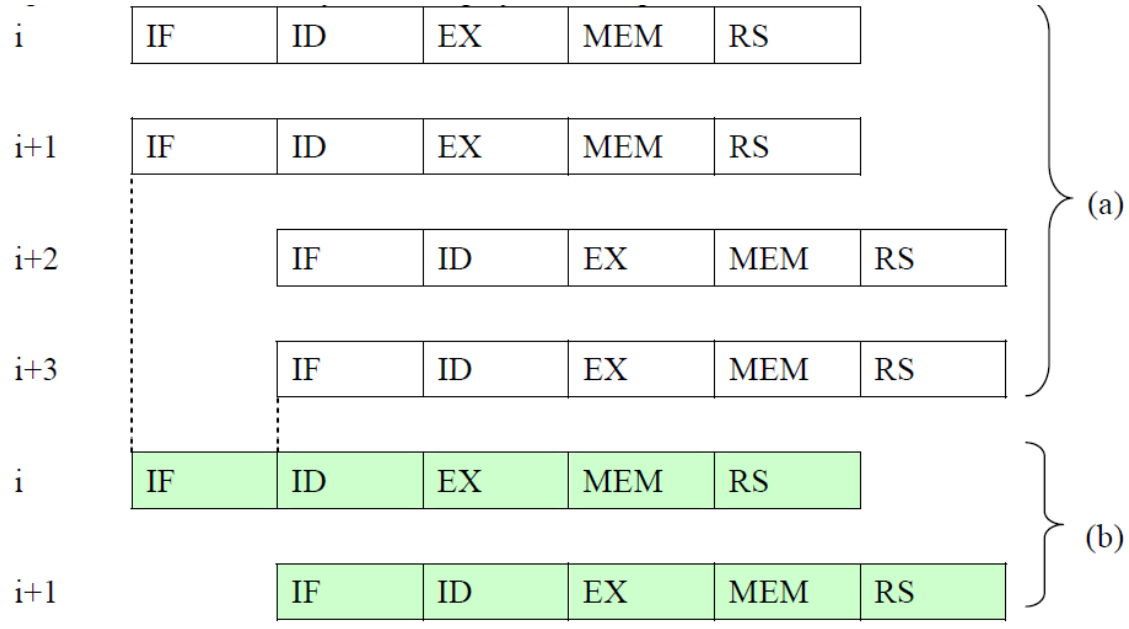
Máy tính có kỹ thuật siêu ống dẫn bậc n bằng cách chia các giai đoạn của kỹ thuật ống dẫn đơn giản, mỗi giai đoạn được thực hiện trong khoản thời gian Tc, thành n giai đoạn con thực hiện trong khoản thời gian Tc/n. Độ hữu hiệu của kỹ thuật này tương đương với việc thi hành n lệnh trong mỗi chu kỳ Tc. Hình 3.7 trình bày thí dụ về siêu ống dẫn bậc 2, có so sánh với siêu ống dẫn đơn giản. Ta thấy trong một chu kỳ Tc, máy dùng kỹ thuật siêu ống dẫn làm 2 lệnh thay vì làm1 lệnh trong máy dùng kỹ thuật ống dẫn bình thường. Trong máy tính siêu ống dẫn, tốc độ thực hiện lệnh tương đương với việc thực hiện một lệnh trong khoảng thời gian Tc/n. Các bất lợi của siêu ống dẫn là thời gian thực hiện một giai đoạn con ngắn Tc/n và việc trì hoãn trong thi hành lệnh nhảy lớn. Trong ví dụ ở hình 3.7, nếu lệnh thứ i là một lệnh nhảy tương đối thì lệnh này được giải mã trong giai đoạn ID, địa chỉ nhảy đến được tính vào giai đoạn EX, lệnh phải được nhảy tới là lệnh thứ i+4, vậy có trì trệ 3 lệnh thay vì 1 lệnh trong kỹ thuật ống dẫn bình thường.



Hình 3.7. Siêu ống dẫn bậc 2 so với siêu ống dẫn đơn giản. Trong khoảng thời gian Tc, máy có siêu ống dẫn làm 2 lệnh thay vì 1 lệnh như trong máy có kỹ thuật ống dẫn đơn giản.

3.8. SIÊU VÔ HƯỚNG

Máy tính siêu vô hướng bậc n có thể thực hiện đồng thời n lệnh trong một chu kỳ xung nhịp Tc. Hình 3.8 trình bày một ví dụ về sự vận hành của một máy tính siêu vô hướng bậc 2 so với một máy tính dùng kỹ thuật ống dẫn.



Hình 3.8. Siêu vô hướng (a) so với kỹ thuật ống dẫn (b)

Trong một máy tính siêu vô hướng phần cứng phải quản lý việc đọc và thi hành đồng thời nhiều lệnh. Vậy nó phải có khả năng quản lý các quan hệ giữa số liệu với nhau. Cũng cần phải chọn các lệnh có khả năng được thi hành cùng một lúc. Những bộ xử lý đầu tiên đưa ra thị trường dùng kỹ thuật này là các bộ xử lý Intel i860 và IBM

RS/6000. Các bộ xử lý này có khả năng thực hiện song song nhiều tác vụ trên số nguyên và trên số lẻ.

Năm 1992, người ta thấy xuất hiện các bộ xử lý có nhiều bộ thực hiện tác vụ độc lập với nhau (nhiều ALU, bộ tính toán số lẻ, nạp dữ liệu, lưu dữ liệu, nhảy), có thể thực hiện song song nhiều lệnh (lệnh tính số nguyên, số lẻ, lệnh bộ nhớ, lệnh nhảy...). Số lệnh có thể được thi hành song song càng nhiều thì phần cứng thực hiện việc này càng phức tạp.

3.9. MÁY TÍNH SONG SONG

Trong các máy tính siêu ống dẫn, siêu vô hướng, máy tính vectơ, máy tính VLIW, người ta đã dùng tính thực hiện song song các lệnh ở các mức độ khác nhau để làm tăng hiệu quả của chúng. Giới hạn về khả năng tính toán của loại máy trên cùng với sự phát triển của công nghệ máy tính khiến người ta nghĩ tới giải pháp song song theo đó người ta tăng cường hiệu quả của máy tính bằng cách tăng số lượng bộ xử lý.

Các máy tính có thể sắp xếp vào 4 loại sau:

1. SISD (Single Instructions Stream, Single Data Stream): Máy tính một dòng lệnh, một dòng số liệu.
2. SIMD (Single Instructions Stream, Multiple Data Stream): Máy tính một dòng lệnh, nhiều dòng số liệu.
3. MISD (Multiple Instructions Stream, Single Data Stream): Máy tính nhiều dòng lệnh, một dòng số liệu.
4. MIMD (Multiple Instruction Stream, Multiple Data Stream): Máy tính nhiều dòng lệnh, nhiều dòng số liệu.

Kiểu phân loại này đơn giản, dễ hiểu, vẫn còn hiệu lực đến hôm nay, mặc dù có những máy tính dùng kiến trúc hỗn tạp.

Các máy tính SISD tương ứng với các máy một bộ xử lý mà chúng ta đã nghiên cứu.

Các máy MISD kiểu máy tính này không sản xuất thương mại.

Các máy SIMD có một số lớn các bộ xử lý giống nhau, cùng thực hiện một lệnh giống nhau để xử lý nhiều dòng dữ liệu khác nhau. Mỗi bộ xử lý có bộ nhớ dữ liệu riêng, nhưng chỉ có một bộ nhớ lệnh và một bộ xử lý điều khiển, bộ này đọc và thi hành các lệnh. Máy CONNECTION MACHINE 2 (65536 bộ xử lý 1 bít) của công ty Thinking Machine Inc, là một ví dụ điển hình của SIMD. Tính song song dùng trong các máy SIMD là tính song song của các dữ liệu. Nó chỉ có hiệu quả nếu cấu trúc các dữ liệu dễ dàng thích ứng với cấu trúc vật lý của các bộ xử lý thành viên. Các bộ xử lý véc-tơ và mảng thuộc loại máy tính này

Các máy MIMD có kiến trúc song song, những năm gần đây, các máy MIMD nổi lên và được xem như một kiến trúc đương nhiên phải chọn cho các máy nhiều bộ xử lý dùng trong các ứng dụng thông thường, một tập hợp các bộ xử lý thực hiện một chuỗi các lệnh khác nhau trên các tập hợp dữ liệu khác nhau. Các máy MIMD hiện tại có thể được xếp vào ba loại hệ thống sẽ được giới thiệu trong phần tiếp theo của chương trình là: SMP (Symmetric Multiprocesors), Cluster và NUMA (Nonunifrom Memory Access)

Hệ thống SMP bao gồm nhiều bộ xử lý giống nhau được lắp đặt bên trong một máy tính, các bộ xử lý này kết nối với nhau bởi một hệ thống bus bên trong hay một vài sự sắp xếp chuyển mạch thích hợp. Vấn đề lớn nhất trong hệ thống SMP là sự kết hợp các hệ thống cache riêng lẻ. Vì mỗi bộ xử lý trong SMP có một cache riêng của nó, do đó, một khối dữ liệu trong bộ nhớ trong có thể tồn tại trong một hay nhiều cache khác nhau. Nếu một khối dữ liệu trong một cache của một bộ xử lý nào đó bị thay đổi sẽ dẫn đến dữ liệu trong cache của các bộ xử lý còn lại và trong bộ nhớ trong không đồng nhất. Các giao thức cache kết hợp được thiết kế để giải quyết vấn đề này.

Hệ thống cluster gồm các máy tính độc lập được kết nối với nhau thông qua một hệ thống kết nối tốc độ cao (mạng tốc độ cao Fast Ethernet hay Gigabit) và hoạt động như một máy tính thống nhất. Mỗi máy trong hệ thống được xem như là một phần của cluster, được gọi là một nút (node). Hệ thống cluster có các ưu điểm:

- Tốc độ cao: Có thể tạo ra một hệ thống cluster có khả năng xử lý mạnh hơn bất cứ một máy tính đơn lẻ nào. Mỗi cluster có thể bao gồm hàng tá máy tính, mỗi máy có nhiều bộ xử lý.
- Khả năng mở rộng cao: có thể nâng cấp, mở rộng một cluster đã được cấu hình và hoạt động ổn định.
- Độ tin cậy cao: Hệ thống vẫn hoạt động ổn định khi có một nút (node) trong hệ thống bị hư hỏng. Trong nhiều hệ thống, khả năng chịu lỗi (fault tolerance) được xử lý tự động bằng phần mềm.
- Chi phí đầu tư thấp: hệ thống cluster có khả năng mạnh hơn một máy tính đơn lẻ mạnh nhất với chi phí thấp hơn.

Hệ thống NUMA (Nonunifrom Memory Access) là hệ thống đa xử lý được giới thiệu trong thời gian gần đây, đây là hệ thống với bộ nhớ chia sẻ, thời gian truy cập các vùng nhớ dành riêng cho các bộ xử lý thì khác nhau. Điều này khác với kiểu quản lý bộ nhớ trong hệ thống SMP (bộ nhớ dùng chung, thời gian truy cập các vùng nhớ khác nhau trong hệ thống cho các bộ xử lý là như nhau). Hệ thống này có những thuận lợi và bất lợi như sau:

Thuận lợi

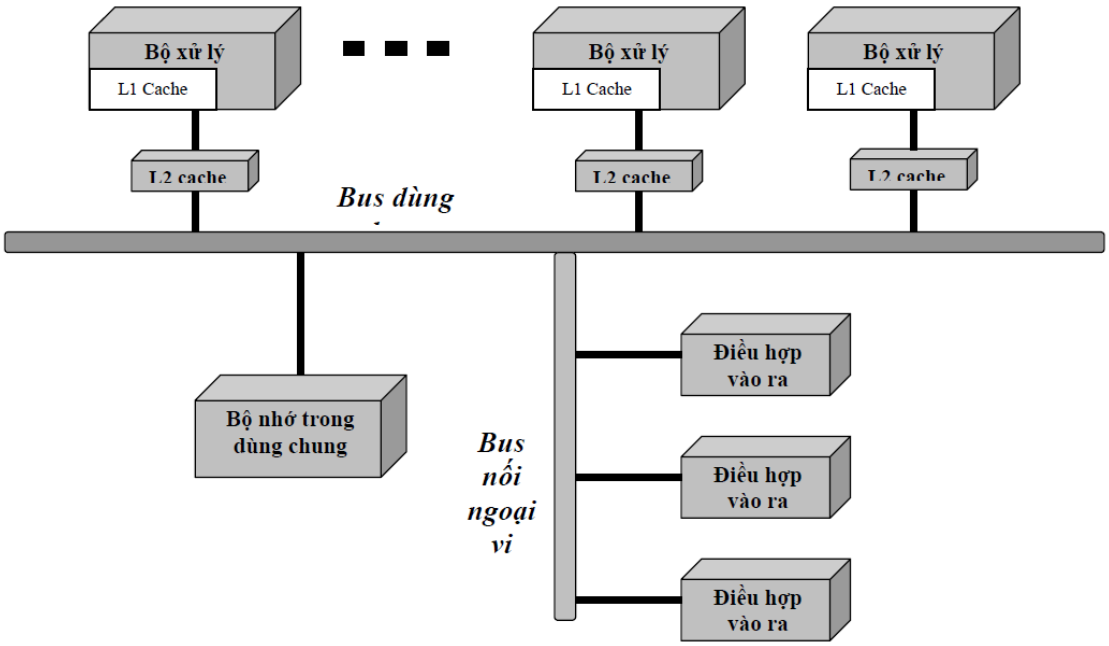
- Thực hiện hiệu quả hơn so với hệ thống SMP trong các xử lý song song.
- Không thay đổi phần mềm chính.
- Bộ nhớ có khả năng bị nghẽn nếu có nhiều truy cập đồng thời, nhưng điều này có thể được khắc phục bằng cách:
    - Cache L1&L2 được thiết kế để giảm tối thiểu tất cả các thâm nhập bộ nhớ.
    - Cần các phần mềm cục bộ được quản lý tốt để việc các ứng dụng hoạt động hiệu quả.
    - Quản trị bộ nhớ ảo sẽ chuyển các trang tới các nút cần dùng.

Bất lợi

- Hệ thống hoạt động không trong suốt như SMP: việc cấp phát các trang, các quá trình có thể được thay đổi bởi các phần mềm hệ thống nếu cần.
- Hệ thống phức tạp.

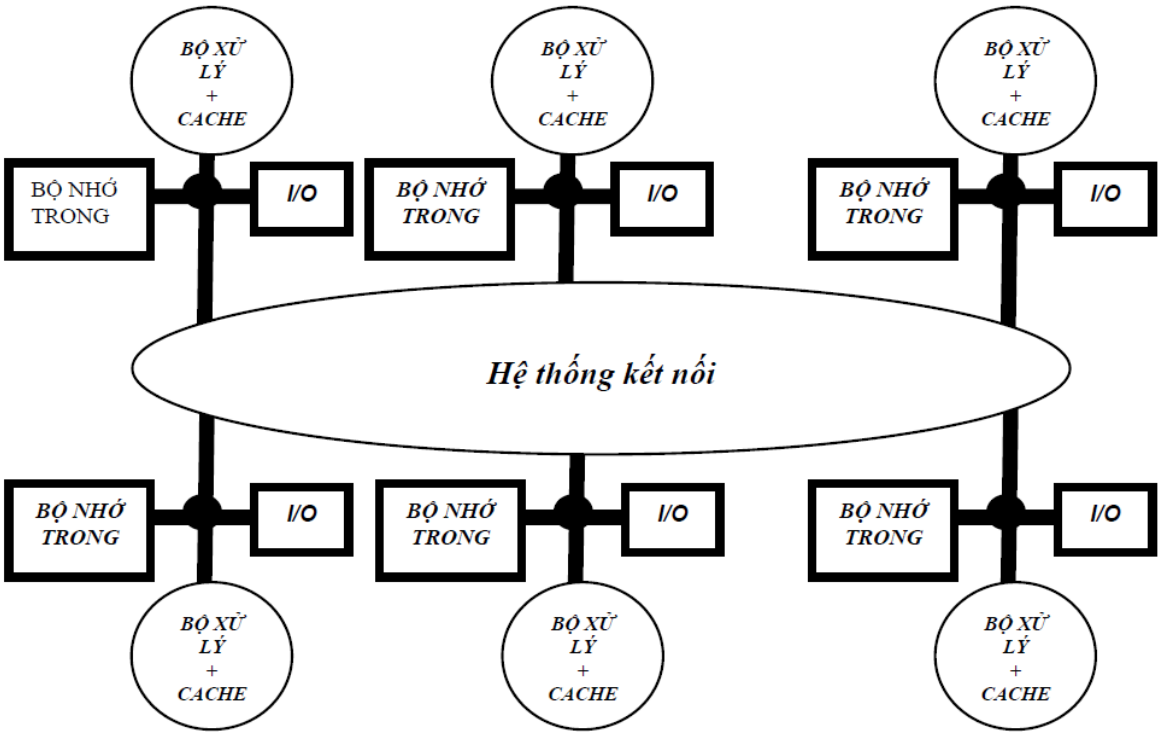
Liên quan đến bộ nhớ trong các máy tính song song, chúng ta có thể chia thành hai nhóm máy:

- Nhóm máy thứ nhất, mà ta gọi là máy có kiến trúc bộ nhớ chia sẻ, có một bộ nhớ trung tâm duy nhất được phân chia cho các bộ xử lý và một hệ thống bus chia sẻ để nối các bộ xử lý và bộ nhớ. Vì chỉ có một bộ nhớ trong nên hệ thống bộ nhớ không đủ khả năng đáp ứng nhu cầu thâm nhập bộ nhớ của một số lớn các bộ xử lý. Kiểu kiến trúc bộ nhớ chia sẻ được dùng trong hệ thống SMP.
- Nhóm máy thứ hai bao gồm các máy có bộ nhớ phân tán vật lý. Mỗi máy của nhóm này gồm có các nút, mỗi nút chứa một bộ xử lý, bộ nhớ, một vài ngã vào ra và một giao diện với hệ thống kết nối giữa các nút.



Hình 3.9. Máy tính song song với bộ nhớ, hệ thống bus dùng chung

Việc phân tán bộ nhớ cho các nút có hai điểm lợi. Trước hết, đây là một cách phân tán việc thâm nhập bộ nhớ. Thứ hai, cách này làm giảm thời gian chờ đợi lúc thâm nhập bộ nhớ cục bộ. Các lợi điểm trên làm cho kiến trúc có bộ nhớ phân tán được dùng cho các máy đa xử lý có một số ít bộ xử lý. Điểm bất lợi chính của kiến trúc máy tính này là việc trao đổi dữ liệu giữa các bộ xử lý trở nên phức tạp hơn và mất nhiều thời gian hơn vì các bộ xử lý không cùng chia sẻ một bộ nhớ trong chung. Cách thực hiện việc trao đổi thông tin giữa bộ xử lý và bộ nhớ trong, và kiến trúc logic của bộ nhớ phân tán là một tính chất đặc thù của các máy tính với bộ nhớ phân tán.



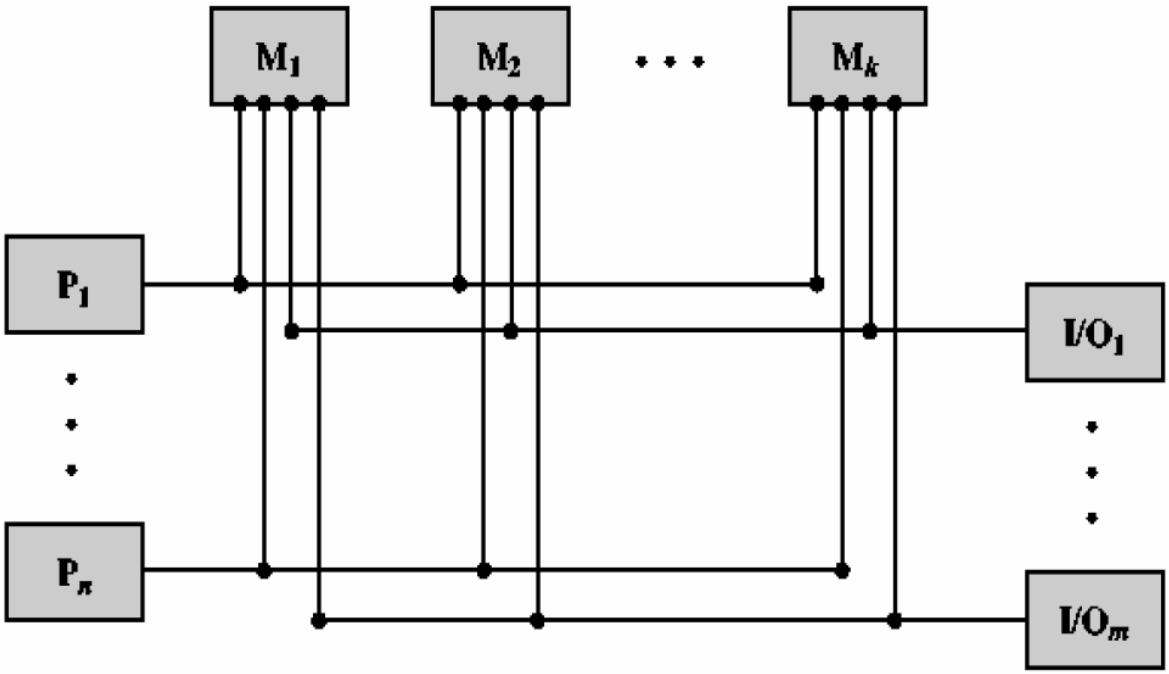
Hình 3.10. Cấu trúc nền của bộ nhớ phân tán

Có 2 phương pháp được dùng để truyền dữ liệu giữa các bộ xử lý.

1. Phương pháp thứ nhất là các bộ nhớ được phân chia một cách vật lý có thể được thâm nhập với một định vị chia sẻ một cách logic, nghĩa là nếu một bộ xử lý bất kỳ có quyền truy xuất, thì nó có thể truy xuất bất kỳ ô nhớ nào. Trong phương pháp này các máy được gọi có kiến trúc bộ nhớ chia sẻ phân tán (DSM: Distributed Sharing Memory). Từ bộ nhớ chia sẻ cho biết không gian định vị bị chia sẻ. Nghĩa là cùng một địa chỉ vật lý cho 2 bộ xử lý tường ứng với cùng một ô nhớ.
2. Phương pháp thứ hai, không gian định vị bao gồm nhiều không gian định vị nhỏ không giao nhau và có thể được một bộ xử lý thâm nhập. Trong phương pháp này, một địa chỉ vật lý gắn với 2 máy khác nhau thì tương ứng với 2 ô nhớ khác nhau trong 2 bộ nhớ khác nhau. Mỗi mô-đun bộ xử lý-bộ nhớ thì cơ bản là một máy tính riêng biệt và các máy này được gọi là đa máy tính. Các máy này có thể gồm nhiều máy tính hoàn toàn riêng biệt và được nối vào nhau thành một mạng cục bộ.

Kiến trúc song song phát triển mạnh trong thời gian gần đây do các lý do:

- Việc dùng xử lý song song đặc biệt trong lãnh vực tính toán khoa học và công nghệ. Trong các lãnh vực này người ta luôn cần đến máy tính có tính năng cao hơn.
- Người ta đã chấp nhận rằng một trong những cách hiệu quả nhất để chế tạo máy tính có tính năng cao hơn các máy đơn xử lý là chế tạo các máy tính đa xử lý.
- Máy tính đa xử lý rất hiệu quả khi dùng cho đa chương trình. Đa chương trình được dùng chủ yếu cho các máy tính lớn và cho các máy phục vụ lớn.



Hình 3.11. Tổ chức kết nối của máy tính song song có bộ nhớ phân tán

Các ví dụ về các siêu máy tính dùng kỹ thuật xử lý song song:

- Máy điện toán Blue Gene/L của IBM đang được đặt tại Phòng thí nghiệm Lawrence Livermore, và đứng đầu trong số 500 siêu máy tính mạnh nhất thế giới. Siêu máy tính Blue Gene/L sẽ được sử dụng cho các công việc "phi truyền thống", chủ yếu là giả lập và mô phỏng các quá trình sinh học và nguyên tử. Máy điện toán Blue Gene/L đã đạt tốc độ hơn 70 teraflop (nghìn tỷ phép tính/giây). Kết quả này có thể sẽ đưa cỗ máy lên vị trí dẫn đầu trong danh sách các siêu máy tính nhanh nhất thế giới, được công bố ngày 8/11/2004. Theo đó, siêu máy tính do IBM lắp ráp đã đạt tốc độ 70,72 teraflop trong các cuộc thử nghiệm hồi tháng 10/2004. IBM nghiên cứu và phát triển Blue Gene với mục đích thử nghiệm nhằm tạo ra các hệ thống cực mạnh nhưng chiếm ít không gian và tiêu thụ ít năng lượng. IBM dự kiến, sẽ lắp đặt cho phòng thí nghiệm quốc gia Lawrence Livermore một siêu máy tính có tốc độ nhanh gấp 4 lần so với kỷ lục vừa đạt được. Khi đó, thiết bị sẽ được ứng dụng vào nhiều nghiên cứu khoa học. Hệ thống mới bao gồm 16,384 giao điểm điện toán kết nối 32.768 bộ xử lý.
- Thông tin mới nhất (02/2005) cho biết: siêu máy tính IBM Blue Gene/L vừa thiết lập kỷ lục mới đó là có khả năng xử lý 135,5 nghìn tỷ phép tính/giây (135,3 teraflop), vượt xa kỷ lục 70,72 teraflop do chính siêu máy tính này lập nên. Số bộ xử lý (BXL) của Blue Gene/L vừa được các nhà khoa học tăng lên gấp đôi (64.000 BXL) nhằm tăng cường khả năng tính toán cho siêu máy tính này. Cũng cần phải nhắc lại rằng thiết kế hoàn thiện của siêu máy tính Blue Gene/L, dự kiến sẽ hoàn tất vào khoảng tháng 6 tới, sẽ bao gồm 130.000 BXL với tốc độ tính toán được kỳ vọng vào khoảng 360 teraflop.
- Hãng điện tử khổng lồ NEC phát hành một supercomputer dạng vector, máy SX-8 mới ra đời có tốc độ xử lý cực đại lên tới 65 teraflop (65 nghìn tỷ phép tính dấu phẩy động/giây) và khả năng hoạt động ổn định ở mức xấp xỉ 90% của tốc độ 58,5% teraflop. Máy SX-8 có kiến trúc khác hẳn Blue Gene/L của

IBM. Nó dùng kiến trúc vector nên đem đến độ ổn định khi hoạt động cao hơn nhiều so với dạng máy tính vô hướng (scalar) như của IBM

- Một hệ thống tại trung tâm nghiên cứu của Cơ quan hàng không vũ trụ Mỹ (NASA) tại California cũng đạt được tốc độ 42,7 teraflop. Với tên gọi Columbia, siêu máy tính này sẽ được sử dụng để nghiên cứu khí tượng và thiết kế máy bay. Hệ thống trị giá 50 triệu USD (thời điểm tháng 10/2004) này sử dụng phần mềm Linux và đã được SGI ký hợp đồng bán cho Cơ quan hàng không vũ trụ Mỹ NASA. Nó có thể thực hiện 42,7 nghìn tỷ phép tính/giây (42,7 teraflop). Tuy nhiên, tốc độ đó chưa phải là tất cả những gì nổi bật của siêu máy tính này: hệ thống mới chỉ khai thác có 4/5 công suất của 10.240 bộ xử lý Intel Itanium 2 trong toàn bộ cỗ máy đặt ở trung tâm nghiên cứu của NASA ở California (Mỹ). Siêu máy tính này không giống với hầu hết các siêu máy tính hiện nay thường được tạo nên theo kiểu cluster, với sự tham gia của nhiều cỗ máy giá rẻ. Columbia được thiết lập từ 20 máy tính mà mỗi chiếc có 512 bộ xử lý, kết nối bằng công nghệ mạng cao tốc và đều chạy một hệ điều hành độc lập. Cách xây dựng này rất hữu ích cho những công việc như giả lập các yếu tố khí động lực cho tàu không gian. Một ứng dụng khác của siêu máy tính Columbia là việc dự báo bão. Phần mềm cho tác vụ này đang được thiết kế và hứa hẹn khả năng dự báo chính xác đường đi của bão sớm 5 ngày. Toàn bộ máy Columbia chiếm dụng một diện tích bằng khoảng 3 sân bóng rổ.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Các thành phần và nhiệm vụ của đường đi dữ liệu?
2. Mô tả đường đi dữ liệu ứng với các lệnh sau:
    a. ADD R1,R2,R3
    b. SUB R1, R2, (R3)
    c. ADD R1, R5, #100
    d. JMP R1 (Nhảy đến ô nhớ mà R1 trỏ tới)
    e. BRA +5 (Nhảy bỏ 5 lệnh)
3. Thế nào là ngắt quãng? Các giai đoạn thực hiện ngắt quãng của CPU.
4. Vẽ hình để mô tả kỹ thuật ống dẫn. Kỹ thuật ống dẫn làm tăng tốc độ CPU lên bao nhiêu lần (theo lý thuyết)? Tại sao trên thực tế sự gia tăng này lại ít hơn?
5. Các điều kiện mà một CPU cần phải có để tối ưu hoá kỹ thuật ống dẫn. Giải thích từng điều kiện.
6. Các khó khăn trong kỹ thuật ống dẫn và cách giải quyết khó khăn này.
7. Thế nào là máy tính vectơ? Các kiểu của kiến trúc vectơ?
8. Cho ví dụ về máy tính một dòng lệnh, nhiều dòng số liệu (SIMD)
9. Các máy tính song song nhiều dòng lệnh, nhiều dòng số liệu (MIMD) dùng nhiều bộ xử lý, được phân thành 2 loại tùy theo tổ chức bộ nhớ của chúng là: máy tính đa xử lý có bộ nhớ tập trung chia sẻ và máy tính đa xử lý có bộ nhớ phân tán. Phân tích ưu - khuyết điểm của hai loại máy tính này.

10. Các loại hệ thống MIMD.
11. Tìm hiểu các kỹ thuật được ứng dụng trong các dòng vi xử lý cho PC của Intel: MMX, HyperThreading, SSE, Turbo Boost, VMX/VT-x, …

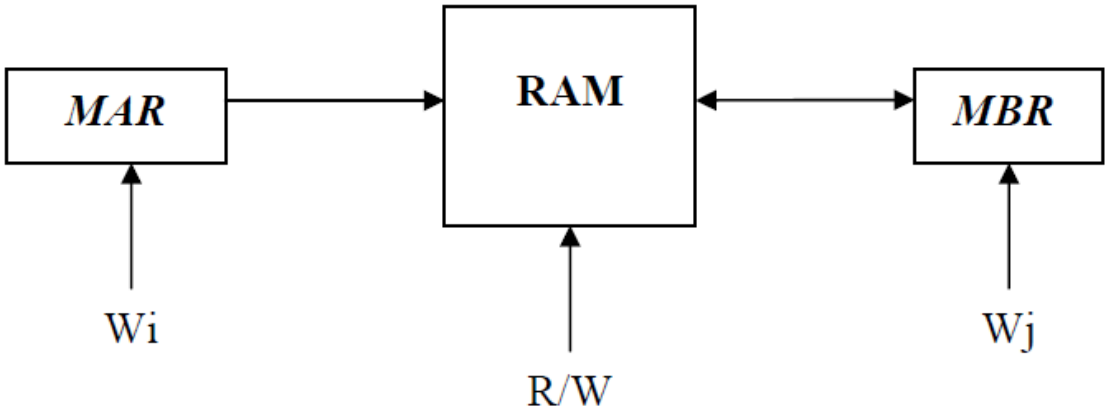
Chương 4

CÁC CẤP BỘ NHỚ

4.1. CÁC LOẠI BỘ NHỚ

Bộ nhớ chứa chương trình, nghĩa là chứa lệnh và số liệu. Người ta phân biệt các loại bộ nhớ: Bộ nhớ trong (RAM-Bộ nhớ vào ra ngẫu nhiên), được chế tạo bằng chất bán dẫn; bộ nhớ chỉ đọc (ROM) cũng là loại bộ nhớ chỉ đọc và bộ nhớ ngoài bao gồm: đĩa cứng, đĩa mềm, băng từ, trống từ, các loại đĩa quang, các loại thẻ nhớ,...

Bộ nhớ RAM có đặc tính là các ô nhớ có thể được đọc hoặc viết vào trong khoảng thời gian bằng nhau cho dù chúng ở bất kỳ vị trí nào trong bộ nhớ. Mỗi ô nhớ có một địa chỉ, thông thường, mỗi ô nhớ là một byte (8 bit), nhưng hệ thống có thể đọc ra hay viết vào nhiều byte (2,4, hay 8 byte). Bộ nhớ trong (RAM) được đặc trưng bằng dung lượng và tổ chức của nó (số ô nhớ và số bit cho mỗi ô nhớ), thời gian thâm nhập (thời gian từ lúc đua ra địa chỉ ô nhớ đến lúc đọc được nội dung ô nhớ đó) và chu kỳ bộ nhớ (thời gian giữa hai lần liên tiếp thâm nhập bộ nhớ).



Hình 4.1. Vận hành của bộ nhớ RAM
(Wi, R/W, Wj là các tín hiệu điều khiển)

Tùy theo công nghệ chế tạo, người ta phân biệt RAM tĩnh (SRAM: Static RAM) và RAM động (Dynamic RAM).

RAM tĩnh được chế tạo theo công nghệ ECL (CMOS và BiCMOS). Mỗi bit nhớ gồm có các cổng logic với độ 6 transistor MOS, việc nhớ một dữ liệu là tồn tại nếu bộ nhớ được cung cấp điện. SRAM là bộ nhớ nhanh, việc đọc không làm hủy nội dung của ô nhớ và thời gian thâm nhập bằng chu kỳ bộ nhớ.

RAM động dùng kỹ thuật MOS. Mỗi bit nhớ gồm có một transistor và một tụ điện. Cũng như SRAM, việc nhớ một dữ liệu là tồn tại nếu bộ nhớ được cung cấp điện. Việc ghi nhớ dựa vào việc duy trì điện tích nạp vào tụ điện và như vậy việc đọc một bit nhớ làm nội dung bit này bị hủy. Vậy sau mỗi lần đọc một ô nhớ, bộ phận điều khiển bộ nhớ phải viết lại ô nhớ đó nội dung vừa đọc và do đó chu kỳ bộ nhớ động ít nhất là gấp đôi thời gian thâm nhập ô nhớ. Việc lưu giữ thông tin trong bit nhớ chỉ là tạm thời vì tụ điện sẽ phóng hết điện tích đã nạp vào và như vậy phải làm tươi bộ nhớ sau mỗi 2μs. Làm tươi bộ nhớ là đọc ô nhớ và viết lại nội dung đó vào lại ô nhớ. Việc làm tươi được thực

hiện với tất cả các ô nhớ trong bộ nhớ. Việc làm tươi bộ nhớ được thực hiện tự động bởi một vi mạch bộ nhớ. Bộ nhớ DRAM chậm nhưng rẻ tiền hơn SRAM.

SDRAM (Synchronous DRAM – DRAM đồng bộ), một dạng DRAM đồng bộ bus bộ nhớ. Tốc độ SDRAM đạt từ 66-133MHz (thời gian thâm nhập bộ nhớ từ 75ns-150ns).

DDR SDRAM (Double Data Rate SDRAM) là cải tiến của bộ nhớ SDRAM với tốc độ truyền tải gấp đôi SDRAM nhờ vào việc truyền tải hai lần trong một chu kỳ bộ nhớ. Tốc độ DDR SDRAM đạt từ 200-400MHz

RDRAM (Rambus RAM) là một loại DRAM được thiết kế với kỹ thuật hoàn toàn mới so với kỹ thuật SDRAM. RDRAM hoạt động đồng bộ theo một hệ thống lặp và truyền dữ liệu theo một hướng. Một kênh bộ nhớ RDRAM có thể hỗ trợ đến 32 chip DRAM. Mỗi chip được ghép nối tuần tự trên một module gọi là RIMM (Rambus Inline Memory Module) nhưng việc truyền dữ liệu giữa các mạch điều khiển và từng chíp riêng biệt chứ không truyền giữa các chip với nhau. Bus bộ nhớ RDRAM là đường dẫn liên tục đi qua các chip và module trên bus, mỗi module có các chân vào và ra trên các đầu đối diện. Do đó, nếu các khe cắm không chứa RIMM sẽ phải gắn một module liên tục để đảm bảo đường truyền được nối liền.Tốc độ RDRAM đạt từ 400-800MHz

Bộ nhớ chỉ đọc ROM cũng được chế tạo bằng công nghệ bán dẫn. Chương trình trong ROM được viết vào lúc chế tạo nó. Thông thường, ROM chứa chương trình khởi động máy tính, chương trình điều khiển trong các thiết bị điều khiển tự động, ...

PROM (Programable ROM): Chế tạo bằng các mối nối (cầu chì - có thể làm đứt bằng điện). Chương trình nằm trong PROM có thể được viết vào bởi người sử dụng bằng thiết bị đặc biệt và không thể xóa được.

EPROM (Erasable Programable ROM): Chế tạo bằng nguyên tắt phân cực tĩnh điện. Chương trình nằm trong ROM có thể được viết vào (bằng điện) và có thể xóa (bằng tia cực tím - trung hòa tĩnh điện) để viết lại bởi người sử dụng.

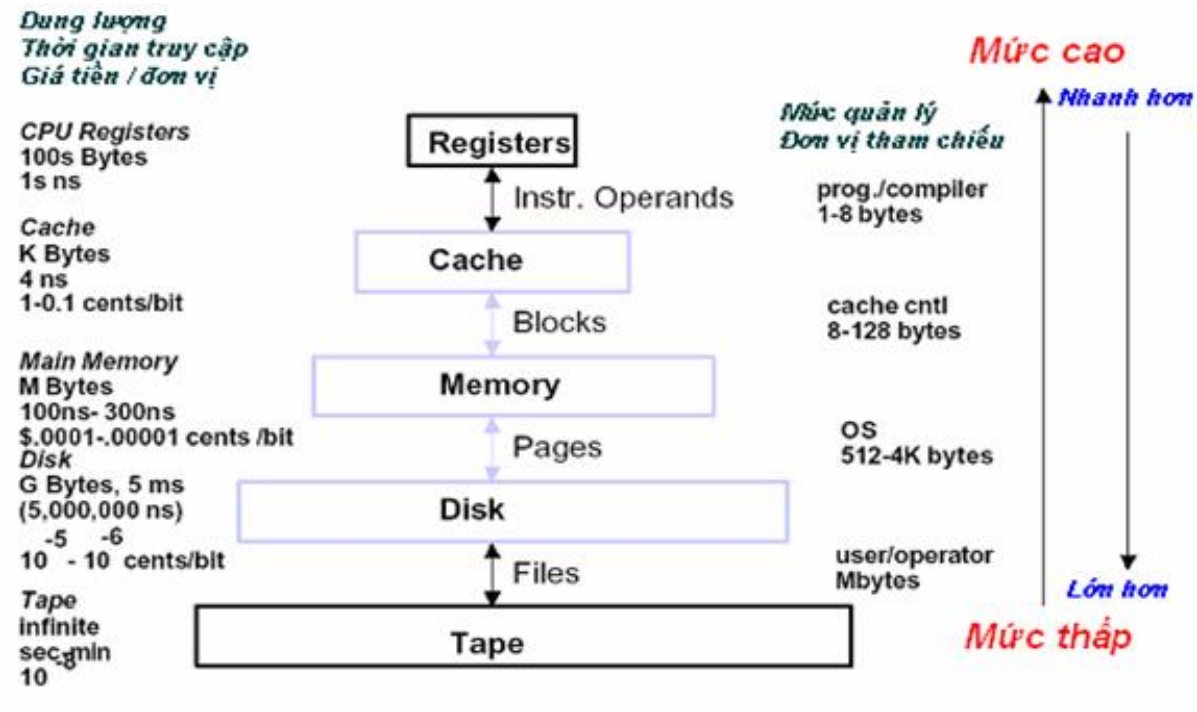
EEPROM (Eletrically Erasable Programable ROM): Chế tạo bằng công nghệ bán dẫn. Chương trình nằm trong ROM có thể được viết vào và có thể xóa (bằng điện) để viết lại bởi người sử dụng.

<table>
<tr><th>Kiểu bộ nhớ</th><th>Loại</th><th>Cơ chế xóa</th><th>Cơ chế ghi</th><th>Tính bay hơi</th></tr>
<tr><td>RAM</td><td>Đọc/ghi</td><td>Điện, mức byte</td><td>Điện</td><td>Có</td></tr>
<tr><td>ROM</td><td rowspan="2">Đọc</td><td rowspan="2">Không</td><td>Mặt nạ</td><td rowspan="5">Không</td></tr>
<tr><td>PROM</td><td rowspan="4">Điện</td></tr>
<tr><td>EPROM</td><td rowspan="3">Hầu hết chỉ đọc</td><td>Tia cực tím, mức chip</td></tr>
<tr><td>EEPROM</td><td>Điện, mức byte</td></tr>
<tr><td>Flash memory</td><td>Điện, mức khối</td></tr>
</table>

Bảng 4.1. Các kiểu bộ nhớ bán dẫn

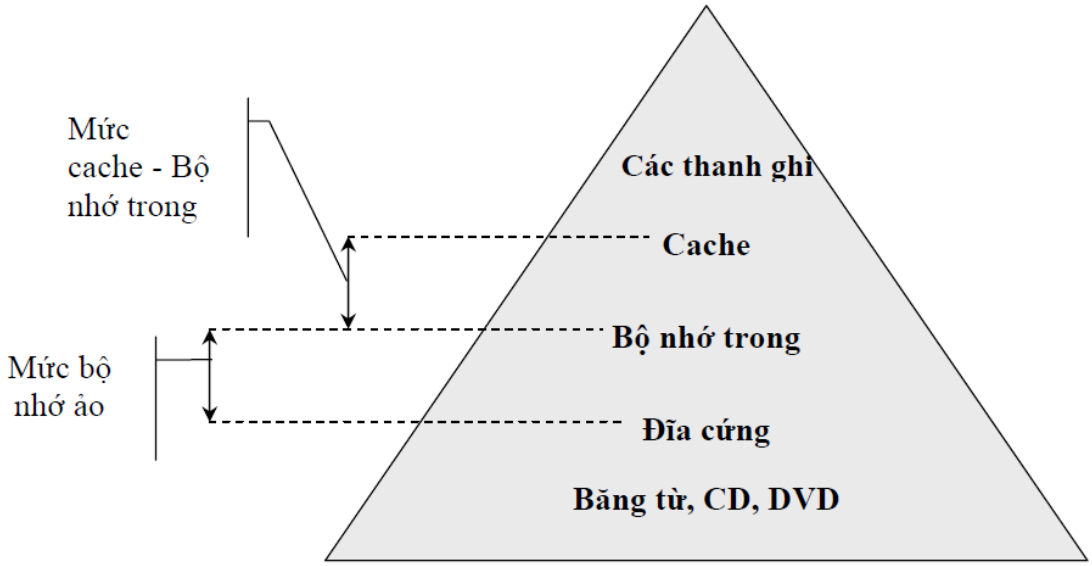
4.2. CÁC CẤP BỘ NHỚ

Các đặc tính như lượng thông tin lưu trữ, thời gian thâm nhập bộ nhớ, chu kỳ bộ nhớ, giá tiền mỗi bit nhớ khiến ta phải phân biệt các cấp bộ nhớ: các bộ nhớ nhanh với dung lượng ít đến các bộ nhớ chậm với dung lượng lớn.



Hình 4.2. Các cấp bộ nhớ

Các đặc tính chính của các cấp bộ nhớ dẫn đến hai mức chính là: mức cache - bộ nhớ trong và mức bộ nhớ ảo (bao gồm bộ nhớ trong và không gian cấp phát trên đĩa cứng). Cách tổ chức này trong suốt đối với người sử dụng. Người sử dụng chỉ thấy duy nhất một không gian định vị ô nhớ, độc lập với vị trí thực tế của các lệnh và dữ liệu cần thâm nhập.



Hình 4.3. Hai mức bộ nhớ

Các cấp bộ nhớ giúp ích cho người lập trình muốn có một bộ nhớ thật nhanh với chi phí đầu tư giới hạn. Vì các bộ nhớ nhanh đắt tiền nên các bộ nhớ được tổ chức thành nhiều cấp, cấp có dung lượng ít thì nhanh nhưng đắt tiền hơn cấp có dung lượng cao hơn. Mục tiêu của việc thiết lập các cấp bộ nhớ là người dùng có một hệ thống bộ nhớ rẻ tiền như cấp bộ nhớ thấp nhất và gần nhanh như cấp bộ nhớ cao nhất. Các cấp bộ nhớ thường được lồng vào nhau. Mọi dữ liệu trong một cấp thì được gặp lại trong cấp thấp hơn và có thể tiếp tục gặp lại trong cấp thấp nhất.

Chúng ta có nhận xét rằng, mỗi cấp bộ nhớ có dung lượng lớn hơn cấp trên mình, ánh xạ một phần địa chỉ các ô nhớ của mình vào địa chỉ ô nhớ của cấp trên trực tiếp có tốc độ nhanh hơn, và các cấp bộ nhớ phải có cơ chế quản lý và kiểm tra các địa chỉ ánh xạ.

4.3. BỘ NHỚ CACHE

4.3.1. Khái niệm

Cache là loại bộ nhớ tốc độ nhanh, hoạt động trung gian giữa CPU và bộ nhớ trong. Nó chứa lệnh và dữ liệu thường xuyên dùng đến. Việc lựa chọn lệnh và dữ liệu cần đặt vào cache dựa vào các nguyên tắc sau đây:

Một chương trình mất 90% thời gian thi hành lệnh của nó để thi hành 10% số lệnh của chương trình.

Nguyên tắc trên cũng được áp dụng cho việc thâm nhập dữ liệu, nhưng ít hiệu nghiệm hơn việc thâm nhập lệnh. Như vậy có hai nguyên tắc: nguyên tắc về không gian và nguyên tắc về thời gian.

Nguyên tắc về thời gian: cho biết các ô nhớ được hệ thống xử lý thâm nhập có khả năng sẽ được thâm nhập trong tương lai gần. Thật vậy, các chương trình được cấu tạo với phần chính là phần được thi hành nhiều nhất và các phần phụ dùng để xử lý các trường hợp ngoại lệ. Còn số liệu luôn có cấu trúc và thông thường chỉ có một phần số liệu được thâm nhập nhiều nhất mà thôi.

Nguyên tắc về không gian: cho biết, bộ xử lý thâm nhập vào một ô nhớ thì có nhiều khả năng thâm nhập vào ô nhớ có địa chỉ kế tiếp do các lệnh được sắp xếp thành chuỗi có thứ tự.

Tổ chức các cấp bộ nhớ sao cho các lệnh và dữ liệu thường dùng được nằm trong bộ nhớ cache, điều này làm tăng hiệu quả của máy tính một cách đáng kể.

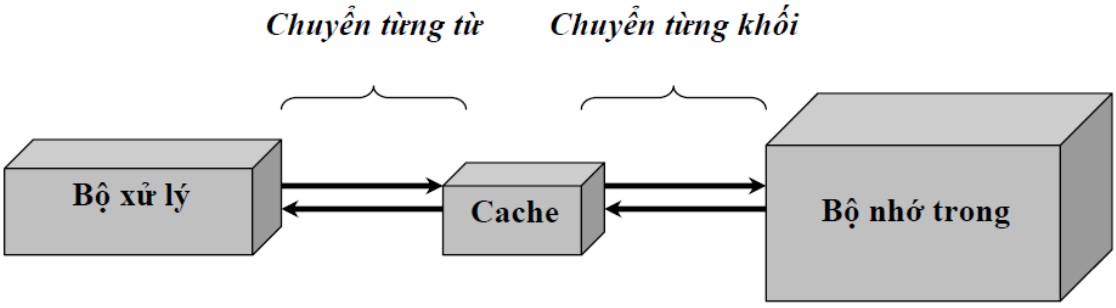
4.3.2. Vận hành của cache

Mức cache -bộ nhớ trong trong bảng các cấp bộ nhớ có cơ cấu vận hành trong suốt đối với bộ xử lý. Với thao tác đọc bộ nhớ, bộ xử lý gởi một địa chỉ và nhận một dữ liệu từ bộ nhớ trong. Với thao tác ghi bộ nhớ, bộ xử lý viết một dữ liệu vào một ô nhớ với một địa chỉ được chỉ ra trong bộ nhớ. Để cho chương trình vận hành bình thường thì cache phải chứa một phần con của bộ nhớ trong để bộ xử lý có thể thâm nhập vào các lệnh hoặc dữ liệu thường dùng từ bộ nhớ cache. Do dung lượng của bộ nhớ cache nhỏ nên nó chỉ chứa một phần chương trình nằm trong bộ nhớ trong. Để đảm bảo sự đồng nhất giữa nội dung của cache và bộ nhớ trong thì cache và bộ nhớ trong phải có cùng cấu trúc. Việc chuyển dữ liệu giữa cache và bộ nhớ trong là việc tải lên hay ghi xuống các khối dữ liệu. Mỗi khối chứa nhiều từ bộ nhớ tùy thuộc vào cấu trúc bộ nhớ cache. Sự lựa chọn kích thước của khối rất quan trọng cho vận hành của cache có hiệu quả.

Trước khi khảo sát vận hành của cache, ta xét đến các khái niệm liên quan:

- Thành công cache (cache hit): bộ xử lý tìm gặp phần tử cần đọc (ghi) trong cache.
- Thất bại cache (cache miss): bộ xử lý không gặp phần tử cần đọc (ghi) trong cache.

- Trừng phạt thất bại cache (cache penalty): Thời gian cần thiết để xử lý một thất bại cache. Thời gian bao gồm thời gian thâm nhập bộ nhớ trong cộng với thời gian chuyển khối chứa từ cần đọc từ bộ nhớ trong đến cache. Thời gian này tùy thuộc vào kích thước của khối.



Hình 4.4. Trao đổi dữ liệu giữa các thành phần CPU-cache-bộ nhớ trong

Để hiểu được cách vận hành của cache, ta lần lượt xem xét và trả lời bốn câu hỏi liên quan đến các tình huống khác nhau xảy ra trong bộ nhớ trong.

Câu hỏi 1: Phải để một khối bộ nhớ vào chỗ nào của cache (sắp xếp khối)?

Câu hỏi 2: Làm sao để tìm một khối khi nó hiện diện trong cache (nhận diện khối)?

Câu hỏi 3: Khối nào phải được thay thế trong trường hợp thất bại cache (thay thế khối)?

Câu hỏi 4: Việc gì xảy ra khi ghi vào bộ nhớ (chiến thuật ghi)?

Trả lời câu hỏi 1: Phải để một khối bộ nhớ vào chỗ nào của cache (sắp xếp khối)?

Một khối bộ nhớ được đặt vào trong cache theo một trong ba cách sau:

1) Kiểu tương ứng trực tiếp: Nếu mỗi khối bộ nhớ chỉ có một vị trí đặt khối duy nhất trong cache được xác định theo công thức: **K= i mod n**

Trong đó:

K: vị trí khối đặt trong cache

i: số thứ tự của khối trong bộ nhớ trong

n: số khối của cache

Như vậy, trong kiểu xếp đặt khối này, mỗi vị trí đặt khối trong cache có thể chứa một trong các khối trong bộ nhớ cách nhau **xn** khối (x: 0,1,...,m; n: số khối của cache)

Ví dụ:

Số thứ tự khối cache	Số thứ tự của khối trong bộ nhớ trong
0	0, n, 2n, …, mn
1	1, n+1, 2n+1, …, mn+1
…	
n-1	n-1, 2n-1, …, mn-1

2) Kiểu hoàn toàn phối hợp: trong kiểu đặt khối này, một khối trong bộ nhớ trong có thể được đặt vào vị trí bất kỳ trong cache.

Như vậy, trong kiểu xếp đặt khối này, mỗi vị trí đặt khối trong cache có thể chứa một trong tất cả các khối trong bộ nhớ.

3) Kiểu phối hợp theo tập hợp: với cách tổ chức này, cache bao gồm các tập hợp của các khối cache. Mỗi tập hợp của các khối cache chứa số khối như nhau. Một khối của bộ nhớ trong có thể được đặt vào một số vị trí khối giới hạn trong tập hợp được xác định bởi công thức: **K= i mod s**

Trong đó:

K: vị trí khối đặt trong cache

i: số thứ tự của khối trong bộ nhớ trong

s: số lượng tập hợp trong cache.

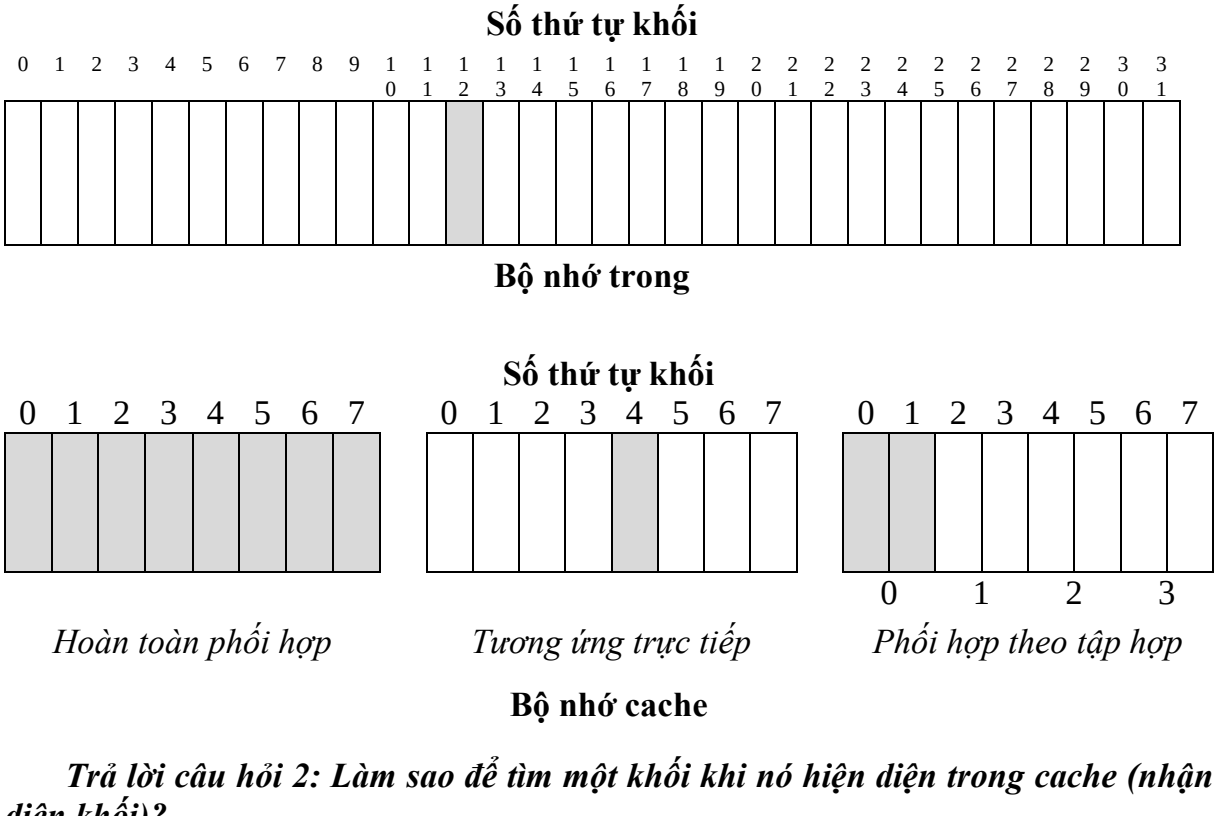
Trong cách đặt khối theo kiểu phối hợp theo tập hợp, nếu tập hợp có m khối, sự tương ứng giữa các khối trong bộ nhớ trong và các khối của cache được gọi là phối hợp theo tập hợp **m** khối.

Nếu m=1 (mỗi tập hợp có 1 khối), ta có kiểu tương ứng trực tiếp.

Nếu m=n (n: số khối của cache), ta có kiểu tương hoàn toàn phối hợp.

Hiện nay, phần lớn các cache của các bộ xử lý đều là kiểu tương ứng trực tiếp hay kiểu phối hợp theo tập hợp (mỗi tập hợp gồm 2 hoặc 4 khối).

Ví dụ: Bộ nhớ trong có 32 khối, cache có 8 khối, mỗi khối gồm 32 byte, khối thứ 12 của bộ nhớ trong được đưa vào cache.



Trả lời câu hỏi 2: Làm sao để tìm một khối khi nó hiện diện trong cache (nhận diện khối)?

Mỗi khối của cache đều có một nhãn địa chỉ cho biết số thứ tự của các khối bộ nhớ trong đang hiện diện trong cache. Nhãn của một khối của cache có thể chứa thông tin cần thiết được xem xét để biết được các khối nằm trong cache có chứa thông tin mà bộ

xử lý cần đọc hay không. Tất cả các nhãn đều được xem xét song song (trong kiểu tương ứng trực tiếp và phối hợp theo tập hợp) vì tốc độ là yếu tố then chốt. Để biết xem một khối của của cache có chứa thông tin mà bộ xử lý cần tìm hay không, người ta thêm một bit đánh dấu (valid bit) vào nhãn để nói lên khối đó có chứa thông tin mà bộ xử lý cần tìm hay không.

Như đã mô tả ở phần đầu, với thao tác đọc (ghi) bộ nhớ, bộ xử lý đưa ra một địa chỉ và nhận (viết vào) một dữ liệu từ (vào) bộ nhớ trong. Địa chỉ mà bộ xử lý đưa ra có thể phân tích thành hai thành phần: phần nhận dạng số thứ tự khối và phần xác định vị trí từ cần đọc trong khối.

Tương ứng với ba kiểu lắp đặt khối đã xét, ta có:

a) Căn cứ vào tổ chức số từ trong khối bộ nhớ mà số bit trong địa chỉ xác định vị trí từ cần đọc trong khối. Cách này đúng với cả ba cách xếp đặt khối đã xét.

b) Phần nhận dạng số thứ tự khối sẽ khác nhau tùy thuộc vào cách xếp đặt khối, trường chỉ số khối được so sánh với nhãn của cache để xác định khối trong cache.

Dữ liệu được bộ xử lý đọc cùng lúc với việc đọc nhãn. Phần chỉ số khối của khối trong bộ nhớ trong được so sánh với bảng tương quan để xác định khối có nằm trong cache hay không. Để chắc rằng nhãn chứa thông tin đúng đắn (tức là khối có chứa từ mà bộ xử lý cần đọc-ghi), nếu việc so sánh nhãn của khối cache giống với số thứ tự khối, bit đánh dấu (Valid bit) phải được bật lên. Ngược lại, kết quả so sánh được bỏ qua. Bộ xử lý căn cứ vào phần xác định từ trong khối để đọc (ghi) dữ liệu từ (vào) cache.

Đối với kiểu tương ứng trực tiếp, phần nhận dạng chỉ số khối được chia thành hai phần:

- Phần chỉ số khối cache: chỉ ra số thứ tự khối cache tương ứng cần xem xét.
- Phần nhãn: so sánh tương ứng với nhãn của khối cache được chỉ ra bởi phần chỉ số khối

<table>
<tr><td colspan="2">Chỉ số khối trong bộ nhớ</td><td rowspan="2">Địa chỉ
từ cần đọc trong khối</td></tr>
<tr><td>Nhãn</td><td>Chỉ số khối cache</td></tr>
</table>

Đối với kiểu hoàn toàn phối hợp, phần nhận dạng chỉ số khối trong địa chỉ sẽ được so sánh với nhãn của tất cả các khối cache.

<table>
<tr><td>Chỉ số khối trong bộ nhớ</td><td>Địa chỉ
từ cần đọc trong khối</td></tr>
</table>

Đối với kiểu phối hợp theo tập hợp, phần nhận dạng chỉ số khối được chia thành hai phần:

- Phần chỉ số tập hợp: chỉ ra số thứ tự tập hợp trong cache cần xem xét.
- Phần nhãn: so sánh tương ứng với nhãn của các khối cache thuộc tập hợp được chỉ ra bởi phần chỉ số tập hợp.

Chỉ số khối trong bộ nhớ		Địa chỉ từ cần đọc trong khối
Nhãn	Chỉ số tập hợp	

Ví dụ: phân tích địa chỉ một từ trong được cho ở trên, địa chỉ xác định một từ trong bộ nhớ có 10 bit, tùy theo cách xếp đặt khối mà ta có thể phân tích địa chỉ này thành các thành phần như sau:

- Đối với kiểu tương ứng trực tiếp:

10 bit		
Chỉ số khối trong bộ nhớ (5 bit)		Địa chỉ từ cần đọc trong khối (5 bit)
Nhãn (2 bit)	Chỉ số khối cache (3 bit)	

- Đối với kiểu hoàn toàn phối hợp:

10 bit	
Chỉ số khối trong bộ nhớ (5 bit)	Địa chỉ từ cần đọc trong khối (5 bit)

- Đối với kiểu phối hợp theo tập hợp, giả sử cache gồm 4 tập hợp, mỗi tập hợp gồm hai khối:

10 bit		
Chỉ số khối trong bộ nhớ (5 bit)		Địa chỉ từ cần đọc trong khối (5 bit)
Nhãn (3 bit)	Chỉ số tập hợp (2 bit)	

Trả lời câu hỏi 3: Khối nào phải được thay thế trong trường hợp thất bại cache (thay thế khối)?

Khi có thất bại cache, bộ điều khiển cache thâm nhập bộ nhớ trong và chuyển khối mà bộ xử lý cần đọc (ghi) vào cache. Như vậy, khối nào trong cache sẽ bị thay thế bởi khối mới được chuyển lên. Đối với kiểu tương ứng trực tiếp, vị trí đặt khối không có sự lựa chọn, nó được xác định bởi trường chỉ số khối cache trong địa chỉ của từ cần đọc (ghi). Nếu cache là kiểu hoàn toàn phối hợp hay phối hợp theo tập hợp thì khi thất bại phải chọn lựa thay thế trong nhiều khối. Có bốn chiến thuật chủ yếu dùng để chọn khối thay thế trong cache:

- Thay thế ngẫu nhiên: để phân bố đồng đều việc thay thế, các khối cần thay thế trong cache được chọn ngẫu nhiên.
- Khối xưa nhất (LRU: Least Recently Used): các khối đã được thâm nhập sẽ được đánh dấu và khối bị thay thế là khối không được dùng từ lâu nhất.
- Vào trước ra trước (FIFO: First In First Out): khối được đưa vào cache đầu tiên, nếu bị thay thế, khối đó sẽ được thay thế trước nhất.

- Tần số sử dụng ít nhất (LFU: Least Frequently Used): khối trong cache được tham chiếu ít nhất

Điều này sử dụng hệ quả của nguyên tắc sử dụng ô nhớ theo thời gian: nếu các khối mới được dùng có khả năng sẽ được dùng trong tương lai gần, khối bị thay thế là khối không dùng trong thời gian lâu nhất.

Trả lời câu hỏi 4: Việc gì xảy ra khi ghi vào bộ nhớ (chiến thuật ghi)?

Thông thường bộ xử lý thâm nhập cache để đọc thông tin. Chỉ có khoảng 15% các thâm nhập vào cache là để thực hiện thao tác ghi (con số này là 33% với các tính toán vectơ-vectơ và 55% đối với các phép dịch chuyển ma trận). Như vậy, để tối ưu hoá các hoạt động của cache, các nhà thiết kế tìm cách tối ưu hoá việc đọc bởi vì các bộ xử lý phải đợi đến khi việc đọc hoàn thành nhưng sẽ không đợi đến khi việc ghi hoàn tất. Hơn nữa, một khối có thể được đọc, so sánh và như thế việc đọc một khối có thể được bắt đầu khi chỉ số khối được biết. Nếu thao tác đọc thành công, dữ liệu ô nhớ cần đọc sẽ được giao ngay cho bộ xử lý. Chú ý rằng, khi một khối được ánh xạ từ bộ nhớ trong vào cache, việc đọc nội dung của khối cache không làm thay đổi nội dung của khối so với khối còn nằm trong bộ nhớ trong.

Đối với việc ghi vào bộ nhớ thì không giống như trên, việc thay đổi nội dung của một khối không thể bắt đầu trước khi nhãn được xem xét để biết có thành công hay thất bại. Thao tác ghi vào bộ nhớ sẽ tốn nhiều thời gian hơn thao tác đọc bộ nhớ. Trong việc ghi bộ nhớ còn có một khó khăn khác là bộ xử lý cho biết số byte cần phải ghi, thường là từ 1 đến 8 byte. Để đảm bảo đồng nhất dữ liệu khi lưu trữ, có hai cách chính để ghi vào cache:

- Ghi đồng thời: Thông tin được ghi đồng thời vào khối của cache và khối của bộ nhớ trong. Cách ghi này làm chậm tốc độ chung của hệ thống. Các ngoại vi có thể truy cập bộ nhớ trực tiếp
- Ghi lại: Để đảm bảo tốc độ xử lý của hệ thống, thông tin cần ghi chỉ được ghi vào khối trong cache. Để quản lý sự khác biệt nội dung giữa khối của cache và khối của bộ nhớ trong, một bit trạng thái (Dirty bit hay Update bit) được dùng để chỉ thị. Khi một thao tác ghi vào trong cache, bit trạng thái (Dirty bit hay Update bit) của khối cache sẽ được thiết lập. Khi một khối bị thay thế, khối này sẽ được ghi lại vào bộ nhớ trong chỉ khi bit trạng thái đã được thiết lập. Với cách ghi này, các ngoại vi liên hệ đến bộ nhớ trong thông qua cache.

Khi có một thất bại ghi vào cache thì phải lựa chọn một trong hai giải pháp sau:

- Ghi có nạp: khối cần ghi từ bộ nhớ trong được nạp vào trong cache như mô tả ở trên. Cách này thường được dùng trong cách ghi lại.
- Ghi không nạp: khối được thay đổi ở bộ nhớ trong không được đưa vào cache. Cách này được dùng trong cách ghi đồng thời.

Trong các tổ chức có nhiều hơn một bộ xử lý với các tổ chức cache và bộ nhớ chia sẻ, các vấn đề liên quan đến tính đồng nhất của dữ liệu cần được đảm bảo. Sự thay đổi dữ liệu trên một cache riêng lẻ sẽ làm cho dữ liệu trên các hệ thống cache và bộ nhớ liên quan không đồng nhất. Vấn đề trên có thể được giải quyết bằng một trong các hệ thống cache tổ chức như sau:

- Mỗi bộ điều khiển cache sẽ theo dõi các thao tác ghi vào bộ nhớ từ các bộ phận khác. Nếu thao tác ghi vào phần bộ nhớ chia sẻ được ánh xạ vào cache của nó quản lý, bộ điều khiển cache sẽ vô hiệu hoá sự thâm nhập này. Chiến lược này phụ thuộc vào cách ghi đồng thời trên tất cả các bộ điều khiển cache.
- Một vi mạch được dùng để điều khiển việc cập nhật, một thao tác ghi vào bộ nhớ từ một cache nào đó sẽ được cập nhật trên các cache khác.
- Một vùng nhớ chia sẻ cho một hay nhiều bộ xử lý thì không được ánh xạ lên cache. Như vậy, tất cả các thâm nhập vào vùng nhớ chia sẻ này đều bị thất bại cache.

4.4. HIỆU QUẢ CỦA CACHE VÀ CÁC MỨC CACHE

Thông thường người ta dùng thời gian thâm nhập trung bình bộ nhớ trong để đánh giá hiệu quả của cache.

Thời gian thâm nhập trung bình được cho bởi công thức:

$$\begin{Bmatrix} Thời\ gian\ thâm\ nhập \\ trung\ bình\ bộ\ nhớ \end{Bmatrix} = \begin{Bmatrix} Thời\ gian\ thâm\ nhập \\ thành\ công \end{Bmatrix} + \begin{Bmatrix} Tỷ\ lệ \\ thất\ bại \end{Bmatrix} * \begin{Bmatrix} Trừng\ phạt \\ thất\ bại \end{Bmatrix}$$

Thời gian thâm nhập thành công là thời gian thâm nhập vào một thông tin trong một thành công cache. Tỉ số thất bại là tỉ số giữa số thất bại cache và tổng số thâm nhập cache. Thời gian thâm nhập thành công và trừng phạt thất bại được đo bằng đơn vị thời gian hoặc bằng chu kỳ xung nhịp (clock cycle).

Trong việc tìm kiếm thông tin trong cache phải chú ý làm giảm tỉ lệ thất bại mà các nguyên nhân chính là như sau:

- Khởi động: trong lần thâm nhập cache đầu tiên, không có thông tin cần tìm trong cache nên phải chuyển khối chứa thông tin đó vào cache.
- Khả năng: vì cache không thể chứa tất cả các khối cần thiết cho việc thi hành một chương trình nên gặp thất bại do cache thiếu khả năng, do đó một khối bị lấy ra khỏi cache rồi lại được đưa vào sau này.
- Tranh chấp: Nếu chiến thuật thay thế các khối là phối hợp theo tập hợp hay tương ứng trực tiếp, các thất bại do tranh chấp xảy ra vì một khối có thể bị đưa ra khỏi cache rồi được gọi vào sau đó nếu có nhiều khối phải được thay thế trong các tập hợp.

Ba nguyên nhân trên cho ta ý niệm về nguyên nhân thất bại, nhưng mô hình đơn giản trên có những hạn chế của nó. Mô hình này giúp ta thấy một số liệu trung bình nhưng chưa giải thích được từng thất bại một. Ví dụ, nếu tăng kích thước cache thì giảm thất bại do tranh chấp và thất bại do khả năng vì cache càng lớn thì nhiều khối có thể được đưa vào. Tuy nhiên, một thất bại có thể đi từ thất bại do khả năng đến thất bại do tranh chấp khi kích thước của cache thay đổi. Khi nêu ba nguyên nhân trên ta đã không lưu ý đến cách thức thay thế các khối. Cách thức này có thể dẫn đến những vận hành bất thường như là tỉ lệ thất bại cao lên khi độ phối hợp lớn lên.

Cache duy nhất chứa đồng thời lệnh và dữ liệu.

Cache riêng lẻ phân biệt cache lệnh và cache dữ liệu.

Giải pháp sau có lợi là tránh các khó khăn do kiến trúc, khi thi hành các lệnh dùng kỹ thuật ống dẫn.

Với một cache duy nhất, sẽ có tranh chấp khi một lệnh muốn thâm nhập một số liệu trong cùng một chu kỳ của giai đoạn đọc một lệnh khác. Cache riêng lẻ còn giúp tối ưu hóa mỗi loại cache về mặt kích thước tổng quát, kích thước các khối và độ phối hợp các khối.

Việc dùng cache trong có thể làm cho sự cách biệt giữa kích thước và thời gian thâm nhập giữa cache trong và bộ nhớ trong càng lớn. Người ta đưa vào nhiều mức cache:

- Cache mức một (L1 cache): thường là cache trong (on-chip cache; nằm bên trong CPU)
- Cache mức hai (L2 cache) thường là cache ngoài (off-chip cache; cache này nằm bên ngoài CPU).
- Ngoài ra, trong một số hệ thống (PowerPC G4, IBM S/390 G4, Itanium của Intel) còn có tổ chức cache mức ba (L3 cache), đây là mức cache trung gian giữa cache L2 và một thẻ bộ nhớ.

4.5. BỘ NHỚ TRONG

Bộ nhớ trong thoả mãn các yêu cầu của cache và được dùng làm đệm vào ra vì bộ nhớ trong vừa là nơi chứa các thông tin từ ngoài đưa vào, vừa là nơi xuất ra các thông tin cho cache. Việc đo hiệu quả của bộ nhớ trong dựa vào thời gian thâm nhập và bề rộng dãi thông. Thông thường thời gian thâm nhập bộ nhớ trong là phần tử quan trọng cho cache trong lúc dãi thông bộ nhớ là phần chính cho các tác vụ xuất nhập. Với việc dùng phổ biến các cache ngoài, dãi thông của bộ nhớ trong cũng trở thành quan trọng cho cache.

Mặc dù cache cần bộ nhớ trong có thời gian thâm nhập nhỏ, nhưng thường thì dễ cải thiện dãi thông bộ nhớ nhờ nhiều cách tổ chức bộ nhớ mới, hơn là giảm thời gian thâm nhập cho cache. Cache thụ hưởng các tiến bộ về dãi thông bằng cách tăng kích thước của mỗi khối của cache mà không tăng đáng kể trừng phạt thất bại cache.

Người ta dùng các kỹ thuật sau đây để nới rộng dãi thông của bộ nhớ trong:

- Nới rộng chiều dài ô nhớ trong: đây là kỹ thuật đơn giản để tăng giải thông bộ nhớ. Thông thường cache và bộ nhớ trong có chiều rộng ô nhớ là chiều rộng 1 từ vì bộ xử lý thâm nhập vào một từ ô nhớ. Nhân đôi, nhân bốn chiều rộng ô nhớ của cache và bộ nhớ trong làm lưu lượng thâm nhập bộ nhớ trong được nhân đôi hay nhân bốn. Vậy cũng phải chi tiêu thêm để nới rộng bus bộ nhớ (là bus nối bộ xử lý với bộ nhớ).

Một ví dụ bộ xử lý có chiều dài ô nhớ trong lớn là bộ xử lý ALPHA AXP 21064 (Hãng DEC). Cache ngoài, bộ nhớ trong và bus bộ nhớ đều có độ rộng là 256 bit.

- Bộ nhớ đan chéo đơn giản: các IC bộ nhớ có thể được tổ chức thành dãi để đọc hay viết nhiều từ cùng một lúc thay vì chỉ đọc một từ, độ rộng của bus và của cache không thay đổi. Khi gởi nhiều địa chỉ đến nhiều dãi thì ta đọc được nhiều từ cùng một lúc. Bộ nhớ đan chéo cũng cho phép ghi vào bộ nhớ nhiều từ cùng một lúc. Tổ chức bộ nhớ đan chéo đơn giản không rắc rối nhiều so với tổ chức bình thường của bộ nhớ trong vì các dãi có thể dùng chung các đường địa chỉ với bộ điều khiển ô nhớ, và như thế mỗi dãi có thể dùng phần số liệu của bus bộ nhớ. SDRAM và DDR SDRAM là các loại RAM dùng kỹ thuật này

- Bộ nhớ đan chéo tổ chức thành dãi độc lập: một tổ chức bộ nhớ đan chéo hiệu quả hơn, là cho phép nhiều thâm thập bộ nhớ và như thế cho phép các dãi làm việc độc lập với nhau. Mỗi dãi cần có các đường địa chỉ riêng biệt và đôi khi cần bus số liệu riêng biệt: Trong trường hợp này bộ xử lý có thể tiếp tục công việc của mình trong lúc chờ đợi số liệu (trường hợp thất bại cache). RDRAM là bộ nhớ loại này
- Tránh xung đột giữa các dãi bộ nhớ. Trong các máy tính đa xử lý và máy tính vectơ, hệ thống bộ nhớ được thiết kế nhằm cho phép nhiều yêu cầu thâm nhập độc lập nhau. Sự hiệu quả của hệ thống tùy thuộc vào tần số các trường hợp có yêu cầu độc lập thâm nhập vào các dãi khác nhau. Với sự đan chéo bình thường (hình IV.6), các thâm nhập tuần tự hoặc tất cả các thâm nhập vào các địa chỉ cách biệt nhau một số chẵn, thì vận hành tốt nhưng sẽ gặp rắc rối nếu sự cách biệt giữa các địa chỉ là một số lẻ. Một biện pháp mà các máy tính lớn dùng là làm giảm bớt các trường hợp xung đột tĩnh bằng cách tăng số lượng các dãi. Thí dụ, máy NEC SX/3 chia bộ nhớ trong ra 128 dãi.

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

Hình 4.5. Bộ nhớ đan chéo bậc 4

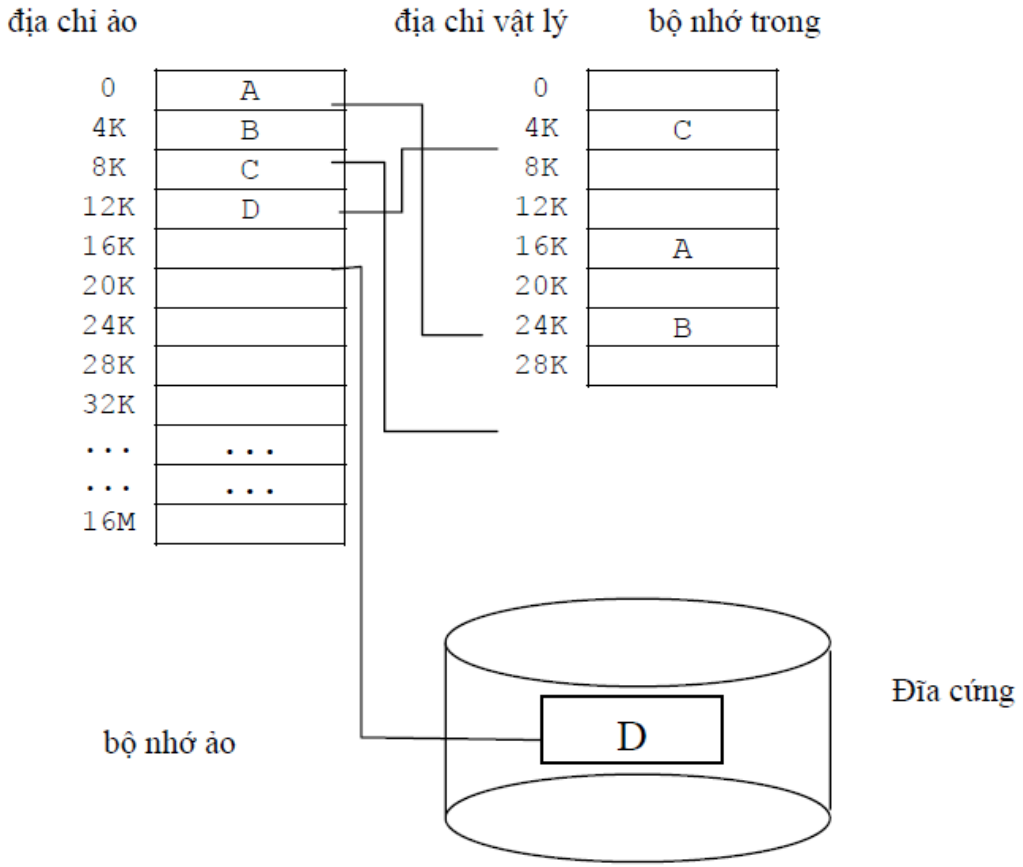
Dãy thứ i chứa tất cả các từ có địa chỉ thỏa mãn công thức ***(địa chỉ) mod 4 = i***

4.6. BỘ NHỚ ẢO

Bộ nhớ ảo xác định một cơ chế vận chuyển tự động số liệu giữa bộ nhớ trong và bộ nhớ ngoài (đĩa từ).

Trước đây, khi độ dài của chương trình vượt quá giới hạn dung lượng bộ nhớ thì người lập trình phải phân chia chương trình của mình thành từng phần tự loại bỏ nhau (overlays) và phải tự quản lý việc trao đổi thông tin giữa bộ nhớ và đĩa từ. Bộ nhớ ảo làm nhẹ trách nhiệm của các nhà lập trình bằng cách làm cho việc trao đổi thông tin này được thực hiện một cách tự động.

Trong các bộ xử lý hiện đại, bộ nhớ ảo được dùng để cho phép thực hiện cùng lúc nhiều tiến trình (process), mỗi tiến trình có một không gian định vị riêng. Nếu tất cả các không gian định vị này đều thuộc không gian định vị bộ nhớ trong thì rất tốn kém. Bộ nhớ ảo bao gồm bộ nhớ trong và bộ nhớ ngoài được phân tích thành khối để có thể cung cấp cho mỗi chương trình một số khối cần thiết cho việc thực hiện chương trình đó. Hình 4.6 cho thấy một chương trình chứa trong bộ nhớ ảo gồm 4 khối, 3 trong 4 khối nằm ở bộ nhớ trong, khối thứ tư nằm trên đĩa.



Hình 4.6. Một chương trình gồm 4 trang A, B, C, D
trong đó trang D nằm trong đĩa

Ngoài việc phân chia không gian bộ nhớ, cần bảo vệ và quản lý tự động các cấp bộ nhớ, bộ nhớ ảo đơn giản hoá việc nạp chương trình vào bộ nhớ để thi hành nhờ một cơ chế được gọi là sự tái định địa chỉ (address relocation). Cơ chế này cho phép một chương trình có thể được thi hành khi nó nằm ở bất cứ vị trí nào trong bộ nhớ.

Tham số	Cache	Bộ nhớ ảo
Chiều dài mỗi khối (trang)	16 – 128 bytes	4096 – 65536 bytes
Thời gian thâm nhập thành công	1 – 2 xung nhịp	40 – 100 xung nhịp
Trừng phạt khi thất bại + Thời gian thâm nhập + Di chuyển số liệu	8 – 100 xung nhịp 6 – 60 xung 2 – 40 xung	700.000 – 6 triệu xung 500.000 – 4 triệu xung 200.000 – 2 triệu xung
Tỷ số thất bại	0,5% - 10%	0,00001% - 0,001%
Dung lượng	8 KB – 8MB	16 MB – 8GB

Bảng 4.2. So sánh các thông tin điển hình giữa bộ nhớ cache và bộ nhớ ảo

Ngoài sự khác biệt trong bảng 4.2, có những khác biệt khác giữa bộ nhớ cache và bộ nhớ ảo là:

- Khi thất bại cache, sự thay thế một khối trong cache được điều khiển bằng phần cứng, trong khi sự thay thế trong bộ nhớ ảo là chủ yếu do hệ điều hành.

- Không gian định vị mà bộ xử lý quản lý là không gian định vị của bộ nhớ ảo, trong lúc đó thì dung lượng bộ nhớ cache không tùy thuộc vào không gian định vị bộ xử lý.
- Bộ nhớ ngoài còn được dùng để lưu trữ tập tin ngoài nhiệm vụ là hậu phương của bộ nhớ trong (trong các cấp bộ nhớ).

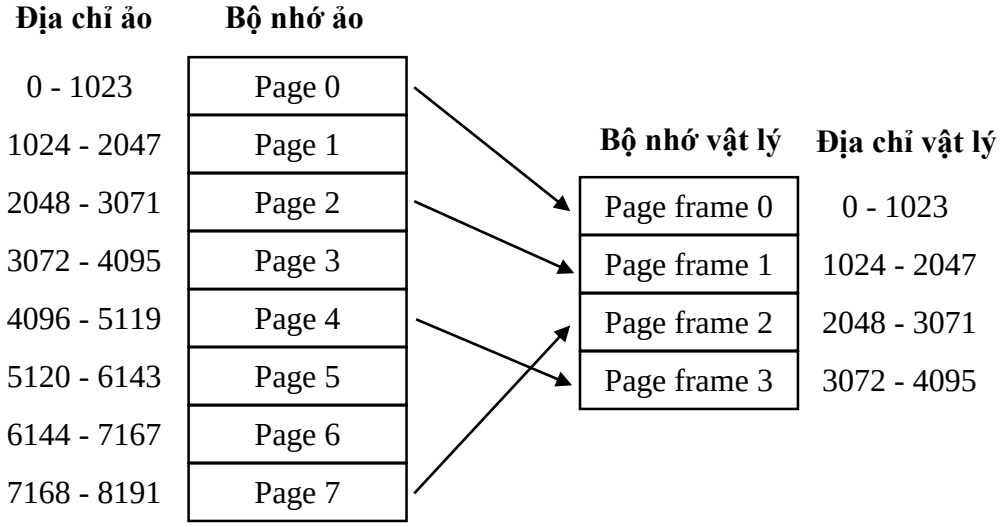
Bộ nhớ ảo cũng được thiết kế bằng nhiều kỹ thuật đặc thù cho chính nó.

Các hệ thống bộ nhớ ảo có thể được chia thành 2 loại: loại với khối có dung lượng cố định gọi là trang, và loại với khối có chiều dài thay đổi gọi là đoạn. Định vị trang xác định một địa chỉ trong trang, giống như định vị trong cache. Trong định vị đoạn cần 2 từ: một từ chứa số thứ tự đoạn và một từ chứa độ dời trong đoạn. Chương trình dịch gặp khó khăn nhiều hơn trong định vị đoạn.

Do việc thay thế các đoạn, ngày nay ít máy tính dùng định vị đoạn thuần tuý. Một vài máy dùng cách hỗn hợp gọi là đoạn trang. Trong đó mỗi đoạn chứa một số nguyên các trang. Bây giờ chúng ta trả lời 4 câu hỏi đặt ra trong các cấp bộ nhớ cho bộ nhớ ảo.

Câu hỏi 1: Một khối được đặt tại đâu trong bộ nhớ trong?

Việc trừng phạt bộ nhớ ảo khi có thất bại, tương ứng với việc phải thâm nhập vào ổ đĩa. Việc thâm nhập này rất chậm nên người ta chọn phương án hoàn toàn phối hợp trong đó các khối (trang) có thể nằm ở bất kỳ vị trí nào trong bộ nhớ trong. Cách này cho tỉ lệ thất bại thấp.



Hình 4.7. Ánh xạ các trang ảo vào bộ nhớ vật lý

Câu hỏi 2: Làm thế nào để tìm một khối khi nó đang nằm trong bộ nhớ trong?

Định vị trang và định vị đoạn đều dựa vào một cấu trúc dữ liệu trong đó số thứ tự trang hoặc số thứ tự đoạn được có chỉ số. Cho định vị trang, dựa vào bảng trang, địa chỉ trong bộ nhớ vật lý được xác lập cuối cùng là việc đặt kề nhau số thứ của trang vật lý với địa chỉ trong trang. Cho định vị đoạn, dựa vào thông tin trên bảng đoạn, việc kiểm tra tính hợp lệ của địa chỉ được tiến hành. Địa chỉ vật cuối cùng được xác lập bằng cách cộng địa chỉ đoạn và địa chỉ trong đoạn (độ dời trong đoạn).

Câu hỏi 3: Khối nào phải được thay thế khi có thất bại trang?

Hầu hết các hệ điều hành đều cố gắng thay thế khối ít dùng gần đây nhất (LRU: Least Recent Utilized) vì nghĩ rằng đây là khối ít cần nhất.

Câu hỏi 4: Việc gì xảy ra khi cần ghi số liệu?

Chiến thuật ghi luôn là một sự ghi lại nghĩa là thông tin chỉ được viết vào trong khối của bộ nhớ trong. Khối có thay đổi thông tin, được chép vào đĩa từ nếu khối này bị thay thế.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Sự khác nhau giữa SRAM và DRAM? Trong máy tính chúng được dùng ở đâu?
2. Mục tiêu của các cấp bộ nhớ?
3. Nêu hai nguyên tắc mà cache dựa vào đó để vận hành.
4. Cho một bộ nhớ cache tương ứng trực tiếp có 8 khối, mỗi khối có 16 byte. Bộ nhớ trong có 64 khối. Giả sử lúc khởi động máy, 8 khối đầu tiên của bộ nhớ trong được đưa lên cache.
    a. Viết bảng nhãn của các khối hiện đang nằm trong cache
    b. CPU lần lượt đưa các địa chỉ sau đây để đọc số liệu: O4AH, 27CH, 3F5H. Nếu thất bại thì cập nhật bãng nhãn.
    c. CPU dùng cách ghi lại. Khi thất bại cache, CPU dùng cách ghi có nạp. Mô tả công việc của bộ quản lý cache khi CPU đưa ra các từ sau đây để ghi vào bộ nhớ trong: 0C3H, 05AH, 1C5H.
5. Các nguyên nhân chính gây thất bại cache?
6. Các giải pháp đảm bảo tính đồng nhất dữ liệu trong hệ thống bộ đa xử lý có bộ nhớ chia sẻ dùng chung?
7. Các cách nới rộng dãy thông của bộ nhớ trong?
8. Tại sao phải dùng bộ nhớ ảo?
9. Sự khác biệt giữa cache và bộ nhớ ảo?

Chương 5

THIẾT BỊ NHẬP XUẤT

5.1. DẪN NHẬP

Bộ xử lý của máy tính điện tử liên hệ với bên ngoài nhờ các bộ phận xuất nhập (I/O) mà ta còn gọi là ngoại vi.

Các ngoại vi thông dụng là:

- Màn hình, bàn phím, chuột, máy in, thẻ mạng... là những bộ phận giúp con người sử dụng máy tính dễ dàng.
- Các đĩa từ, băng từ, đĩa quang, các loại thẻ nhớ là những bộ phận lưu trữ thông tin trữ lượng lớn.

Tất cả các ngoại vi đều được nối vào bộ xử lý và bộ nhớ trong bằng một hệ thống dây nối phức tạp vì tính đa dạng của các ngoại vi.

Trong chương này chúng ta tập trung nói đến các bộ phận lưu trữ số liệu có trữ lượng cao (đĩa từ, đĩa quang, băng từ) và sự kết nối các bộ phận này vào máy tính.

5.2. ĐĨA TỪ

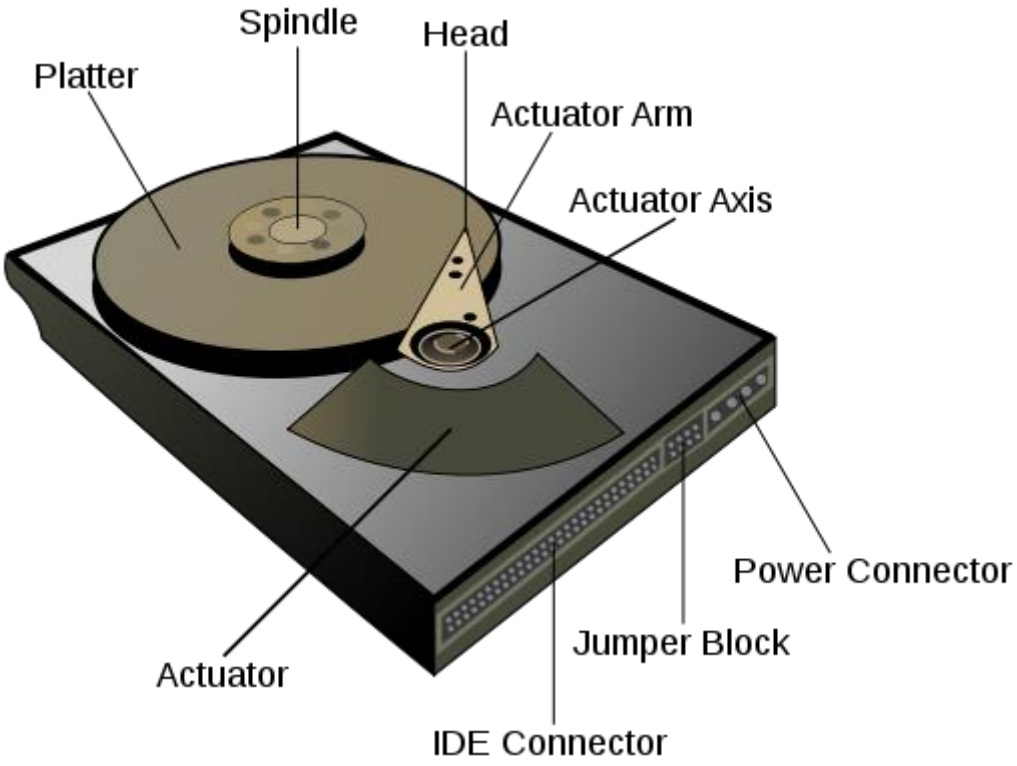
Dù rằng công nghệ mới không ngừng phát minh nhiều loại bộ phận lưu trữ một lượng thông tin lớn nhưng đĩa từ vẫn giữ vị trí quan trọng từ năm 1965. Đĩa từ có hai nhiệm vụ trong máy tính.

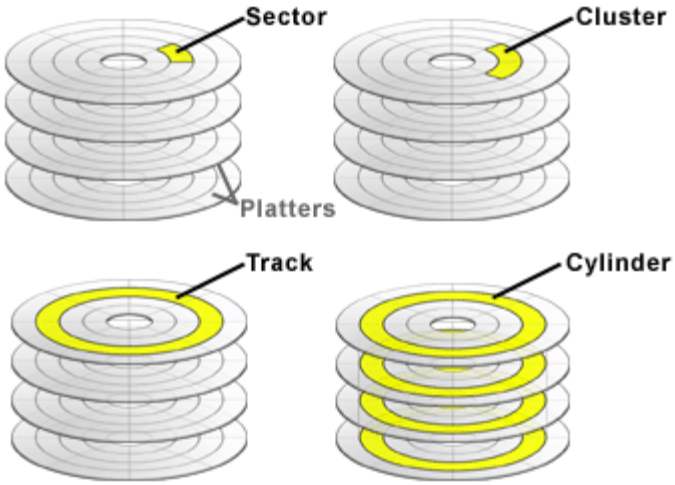
- Lưu trữ dài hạn các tập tin.
- Thiết lập một cấp bộ nhớ bên dưới bộ nhớ trong để làm bộ nhớ ảo lúc chạy chương trình.

Do đĩa mềm dần được các thiết bị lưu trữ khác có các tính năng ưu việt hơn nên chúng ta không xét đến thiết bị này trong chương trình mà chỉ nói đến đĩa cứng. Trong tài liệu này mô tả một cách khái quát cấu tạo, cách vận hành cũng như đề cập đến các tính chất quan trọng của đĩa cứng.

Một đĩa cứng chứa nhiều lớp đĩa (từ 1 đến 4) quay quanh một trục khoảng 3.600-15.000 vòng mỗi phút. Các lớp đĩa này được làm bằng kim loại với hai mặt được phủ một chất từ tính (hình V.1). Đường kính của đĩa thay đổi từ 1,3 inch đến 8 inch. Mỗi mặt của một lớp đĩa được chia thành nhiều đường tròn đồng trục gọi là rãnh. Thông thường mỗi mặt của một lớp đĩa có từ 10.000 đến gần 30.000 rãnh. Mỗi rãnh được chia thành nhiều cung (sector) dùng chứa thông tin. Một rãnh có thể chứa từ 64 đến 800 cung. Cung là đơn vị nhỏ nhất mà máy tính có thể đọc hoặc viết (thông thường khoảng 512 bytes). Chuỗi thông tin ghi trên mỗi cung gồm có: số thứ tự của cung, một khoảng trống, số liệu của cung đó bao gồm cả các mã sửa lỗi, một khoảng trống, số thứ tự của cung tiếp theo.

Với kỹ thuật ghi mật độ không đều, tất cả các rãnh đều có cùng một số cung, điều này làm cho các cung dài hơn ở các rãnh xa trục quay có mật độ ghi thông tin thấp hơn mật độ ghi trên các cung nằm gần trục quay.





Hình 5.1. Cấu tạo của một đĩa cứng

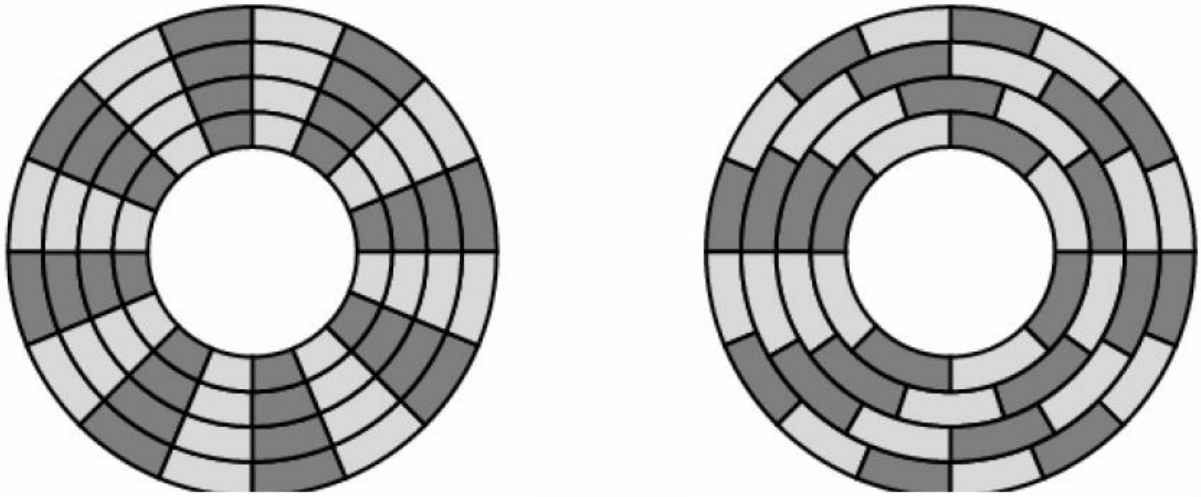
Với công nghệ ghi với mật độ đều, người ta cho ghi nhiều thông tin hơn ở các rãnh xa trục quay. Công nghệ ghi này ngày càng được dùng nhiều với sự ra đời của các chuẩn giao diện thông minh như chuẩn SCSI.

Để đọc hoặc ghi thông tin vào một cung, ta dùng một đầu đọc ghi di động áp vào mỗi mặt của mỗi lớp đĩa. Các đầu đọc/ghi này được gắn chặt vào một thanh làm cho chúng cùng di chuyển trên một đường bán kính của mỗi lớp đĩa và như thế tất cả các đầu này đều ở trên những rãnh có cùng bán kính của các lớp đĩa. Từ "trụ" (cylinder) được dùng để gọi tất cả các rãnh của các lớp đĩa có cùng bán kính và nằm trên một hình trụ.

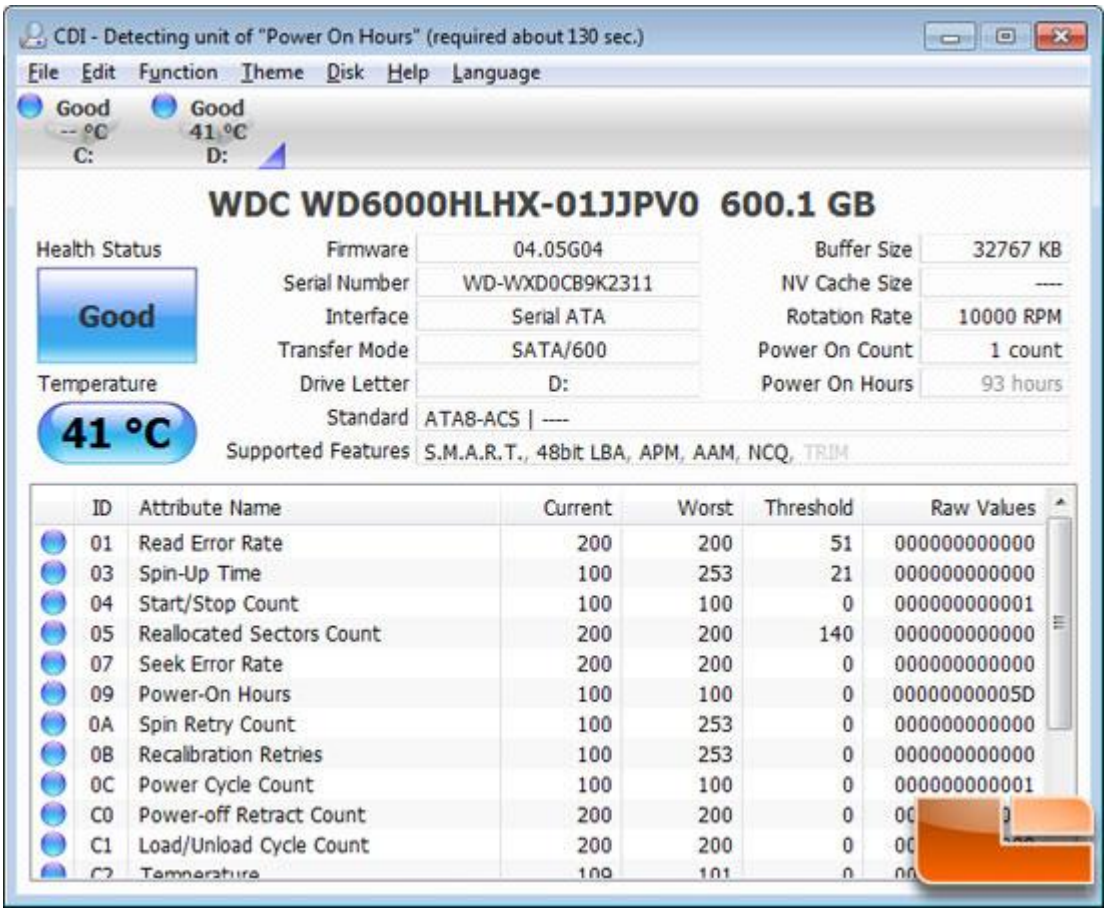
Người ta luôn muốn đọc nhanh đĩa từ nên thông thường ổ đĩa đọc nhiều hơn số dữ liệu cần đọc; người ta nói đây là cách đọc trước. Để quản lý các phức tạp khi kết nối

(hoặc ngưng kết nối) lúc đọc (hoặc ghi) thông tin, và việc đọc trước, ổ đĩa cần có bộ điều khiển đĩa.

Công nghiệp chế tạo đĩa từ tập trung vào việc nâng cao dung lượng của đĩa mà đơn vị đo lường là mật độ trên một đơn vị bề mặt.



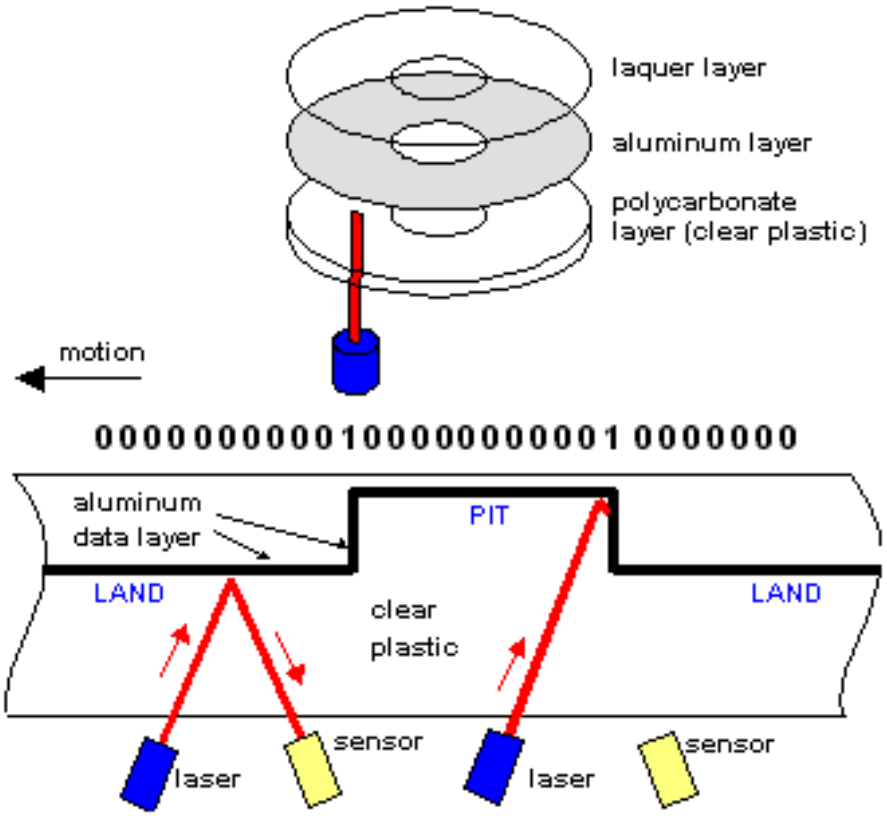
Hình 5.2. Mật độ ghi dữ liệu trên các loại đĩa cứng



Hình 5.3. Thông số kỹ thuật của một đĩa cứng

5.3. ĐĨA QUANG

Các thiết bị lưu trữ quang rất thích hợp cho việc phát hành các sản phẩm văn hoá, sao lưu dữ liệu trên các hệ thống máy tính hiện nay. Ra đời vào năm 1978, đây là sản phẩm của sự hợp tác nghiên cứu giữa hai công ty Sony và Philips trong công nghiệp giải trí. Từ năm 1980 đến nay, công nghiệp đĩa quang phát triển mạnh trong cả hai lĩnh vực giải trí và lưu trữ dữ liệu máy tính. Quá trình đọc thông tin dựa trên sự phản chiếu của các tia laser năng lượng thấp từ lớp lưu trữ dữ liệu. Bộ phận tiếp nhận ánh sáng sẽ nhận biết được những điểm mà tại đó tia laser bị phản xạ mạnh hay biến mất do các vết khắc (pit) trên bề mặt đĩa. Các tia phản xạ mạnh chỉ ra rằng tại điểm đó không có lỗ khắc và điểm này được gọi là điểm nền (land). Bộ nhận ánh sáng trong ổ đĩa thu nhận các tia phản xạ và khuếch tán được khúc xạ từ bề mặt đĩa. Khi các nguồn sáng được thu nhận, bộ vi xử lý sẽ dịch các mẫu sáng thành các bit dữ liệu hay âm thanh. Các lỗ trên CD sâu 0,12 micron và rộng 0,6 micron (1 micron bằng một phần ngàn mm). Các lỗ này được khắc theo một track hình xoắn ốc với khoảng cách 1,6 micron giữa các vòng, khoảng 16.000 track/inch. Các lỗ (pit) và nền (land) kéo dài khoản 0,9 đến 3,3 micron. Track bắt đầu từ phía trong và kết thúc ở phía ngoài theo một đường khép kín các rìa đĩa 5mm. Dữ liệu lưu trên CD thành từng khối, mỗi khối chứa 2.352 byte. Trong đó, 304 byte chứa các thông tin về bit đồng bộ, bit nhận dạng (ID), mã sửa lỗi (ECC), mã phát hiện lỗi (EDC). Còn lại 2.048 byte chứa dữ liệu. Tốc độ đọc chuẩn của CD-ROM là 75 khối/s hay 153.600 byte/s hay 150KB/s (1X).



Hình 5.4. Cơ chế lưu trữ thông tin trên đĩa quang

Dưới đây là một số loại đĩa quang thông dụng.

CD (Compact Disk): Đĩa quang không thể xoá được, dùng trong công nghiệp giải trí (các đĩa âm thanh được số hoá). Chuẩn đĩa có đường kính 12 cm, âm thanh phát từ đĩa khoảng 60 phút (không dừng).

CD-ROM (Compact Disk Read Only Memory): Đĩa không xoá dùng để chứa các dữ liệu máy tính. Chuẩn đĩa có đường kính 12 cm, lưu trữ dữ liệu hơn 650 MB. Khi phát hành, đĩa CD-ROM đã có chứa nội dung. Thông thường, dĩa CD-ROM được dùng để chứa các phần mềm và các chương trình điều khiển thiết bị.

CD-R (CD-Recordable): Giống như đĩa CD, đĩa mới chưa có thông tin, người dùng có thể ghi dữ liệu lên đĩa một lần và đọc được nhiều lần. Dữ liệu trên đĩa CD-R không thể bị xoá.

CD-RW (CD-Rewritable): Giống như đĩa CD, đĩa mới chưa có thông tin, người dùng có thể ghi dữ liệu lên đĩa, xoá và ghi lại dữ liệu trên đĩa nhiều lần.

DVD (Digital Video Disk - Digital Versatile Disk): Ra đời phục vụ cho công nghiệp giải trí, đĩa chứa các hình ảnh video được số hoá. Ngày nay, DVD được sử dụng rộng rãi trong các ứng dụng công nghệ thông tin. Kích thước đĩa có hai loại: 8cm và 12 cm. Đĩa DVD có thể chứa dữ liệu trên cả hai mặt đĩa, dung lượng tối đa lên đến 17GB. Các thông số kỹ thuật của đĩa DVD-ROM (loại đĩa chỉ đọc) so với CD-ROM. Tốc độ đọc chuẩn (1X) của DVD là 1.3MB/s (1X của DVD tương đương khoảng 9X của CDROM).

DVD-R (DVD-Recordable): Giống như đĩa DVD-ROM, người dùng có thể ghi dữ liệu lên đĩa một lần và đọc được nhiều lần. Đĩa này chỉ có thể ghi được trên một mặt đĩa, dung lượng ghi trên mỗi mặt tối đa là 4.7 GB.

DVD-RW (DVD-Rewritable): Giống như đĩa DVD-ROM, người dùng có thể ghi, xoá và ghi lại dữ liệu lên đĩa nhiều lần.. Đĩa này cũng có thể ghi được trên một mặt đĩa, dung lượng ghi trên mỗi mặt tối đa là 4.7 GB.

Với các đặc tính của đĩa quang, giá thành ngày càng thấp, được xem như một phương tiện thích hợp để phân phối các phần mềm cho máy vi tính. Ngoài ra, đĩa quang còn được dùng để lưu trữ lâu dài các dữ liệu thay thế cho băng từ.

5.4. CÁC LOẠI THẺ NHỚ

Hiện nay, thẻ nhớ là một trong những công nghệ mới nhất được dùng làm thiết bị lưu trữ. Thẻ nhớ flash là một dạng bộ nhớ bán dẫn EEPROM (công nghệ dùng để chế tạo các chip BIOS trên các vỉ mạch chính), được cấu tạo bởi các hàng và các cột. Mỗi vị trí giao nhau là một ô nhớ gồm có hai transistor, hai transistor này cách nhau bởi một lớp ô-xít mỏng. Một transistor được gọi là floating gate và transistor còn lại được gọi là control gate. Floating gate chỉ có thể nối kết với hàng (word line) thông qua control gate. Khi đường kết nối được thiết lập, bit có giá trị 1. Để chuyển sang giá trị 0 theo một qui trình có tên Fowler-Nordheim tunneling. Tốc độ, yêu cầu về dòng điện cung cấp thấp và đặc biệt với kích thước nhỏ gọn của các loại thẻ nhớ làm cho kiểu bộ nhớ này được dùng rộng rãi trong công nghệ lưu trữ và giải trí hiện nay.

Tên	Ký hiệu	Kích thước
PC Card	PCMCIA	85.6 × 54 × 3.3 mm
CompactFlash I	CF-I	43 × 36 × 3.3 mm
CompactFlash II	CF-II	43 × 36 × 5.5 mm
SmartMedia	SM / SMC	45 × 37 × 0.76 mm
Memory Stick	MS	50.0 × 21.5 × 2.8 mm

Memory Stick Duo	MSD	31.0 × 20.0 × 1.6 mm
Memory Stick PRO Duo	MSPD	31.0 × 20.0 × 1.6 mm
Memory Stick PRO-HG Duo	MSPDX	31.0 × 20.0 × 1.6 mm
Memory Stick Micro M2	M2	15.0 × 12.5 × 1.2 mm
Miniature Card		37 × 45 × 3.5 mm
Multimedia Card	MMC	32 × 24 × 1.5 mm
Reduced Size Multimedia Card	RS-MMC	16 × 24 × 1.5 mm
MMCmicro Card	MMCmicro	12 × 14 × 1.1 mm
P2 card	P2	
Secure Digital card	SD	32 × 24 × 2.1 mm
SxS	SxS	
Universal Flash Storage	UFS	
miniSD card	miniSD	21.5 × 20 × 1.4 mm
microSD card	microSD	15 × 11 × 0.7 mm
xD-Picture Card	xD	20 × 25 × 1.7 mm
Intelligent Stick	iStick	24 × 18 × 2.8 mm
Serial Flash Module	SFM	45 × 15 mm
µ card	µcard	32 × 24 × 1 mm
NT Card	NT NT+	44 × 24 × 2.5 mm
XQD card	XQD	38.5 × 29.8 × 3.8 mm

Bảng 5.1. Các loại thẻ nhớ

5.5. BĂNG TỪ

Băng từ có cùng công nghệ với các đĩa từ nhưng khác đĩa từ hai điểm:

- Việc thâm nhập vào đĩa từ là ngẫu nhiên còn việc thâm nhập vào băng từ là tuần tự. Như vậy việc tìm thông tin trên băng từ mất nhiều thời gian hơn việc tìm thông tin trên đĩa từ.
- Đĩa từ có dung lượng hạn chế còn băng từ gồm có nhiều cuộn băng có thể lấy ra khỏi máy đọc băng nên dung lượng của băng từ là rất lớn (hàng trăm GB). Với chi phí thấp, băng từ vẫn còn được dùng rộng rãi trong việc lưu trữ dữ liệu dự phòng.

Các băng từ có chiều rộng thay đổi từ 0,38cm đến 1,27 cm được đóng thành cuộn và được chứa trong một hộp bảo vệ. Dữ liệu ghi trên băng từ có cấu trúc gồm một số các rãnh song song theo chiều dọc của băng.

Có hai cách ghi dữ liệu lên băng từ:

- Ghi nối tiếp: với kỹ thuật ghi xoắn ốc, dữ liệu ghi nối tiếp trên một rãnh của băng từ, khi kết thúc một rãnh, băng từ sẽ quay ngược lại, đầu từ sẽ ghi dữ liệu trên rãnh mới tiếp theo nhưng với hướng ngược lại. Quá trình ghi cứ tiếp diễn cho đến khi đầy băng từ.
- Ghi song song: để tăng tốc độ đọc-ghi dữ liệu trên băng từ, đầu đọc - ghi có thể đọc-ghi một số rãnh kề nhau đồng thời. Dữ liệu vẫn được ghi theo chiều dọc băng từ nhưng các khối dữ liệu được xem như ghi trên các rãnh kề nhau. Số rãnh ghi đồng thời trên băng từ thông thường là 9 rãnh (8 rãnh dữ liệu - 1byte và một rãnh kiểm tra lỗi).

5.6. CÁC CHUẨN BUS

Số lượng và chủng loại các bộ phận vào/ra không cần định trước trong các hệ thống xử lý thông tin. Điều này giúp cho người sử dụng máy tính dùng bộ phận vào/ra nào đáp ứng được các yêu cầu của họ. Vào/ra là giao diện trên đó các bộ phận (thiết bị) được kết nối vào hệ thống. Nó có thể xem như một bus nới rộng dùng để kết nối thêm ngoại vi vào máy tính. Các chuẩn làm cho việc nối kết các ngoại vi vào máy tính được dễ dàng; bởi vì, trong khi các nhà thiết kế-sản xuất máy tính và các nhà thiết kế-sản xuất ngoại vi có thể thuộc các công ty khác nhau. Sự tồn tại các chuẩn về bus là rất cần thiết. Như vậy, nếu nhà thiết kế máy tính và nhà thiết kế ngoại vi tôn trọng các chuẩn về bus này thì các ngoại vi có thể kết nối dễ dàng vào máy tính. Chuẩn của bus vào/ra là tài liệu quy định cách kết nối ngoại vi vào máy tính.

Các máy tính quá thông dụng thì các chuẩn về bus vào/ra của chúng có thể được xem là chuẩn cho các hãng khác (ví dụ: trước đây, UNIBUS của máy PDP 11, các chuẩn về bus của máy IBM PC, AT và hiện nay là các chuẩn của hãng Intel liên quan đến các máy vi tính). Các chuẩn về bus phải được các cơ quan về chuẩn như ISO, ANSI và IEEE công nhận.

5.7. MỘT SỐ BIỆN PHÁP AN TOÀN DỮ LIỆU TRONG VIỆC LƯU TRỮ THÔNG TIN

Người ta thường chú trọng đến sự an toàn trong lưu giữ thông tin ở đĩa từ hơn là sự an toàn của thông tin trong bộ xử lý. Bộ xử lý có thể hư mà không làm tổn hại đến thông tin. Ổ đĩa của máy tính bị hư có thể gây ra các thiệt hại rất to lớn.

Một phương pháp giúp tăng cường độ an toàn của thông tin trên đĩa từ là dùng một mảng đĩa từ. Mảng đĩa từ này được gọi là Hệ thống đĩa dự phòng (RAID - Redundant Array of Independent Disks). Cách lưu trữ dư thông tin làm tăng giá tiền và sự an toàn (ngoại trừ RAID 0). Cơ chế RAID có các đặc tính sau:

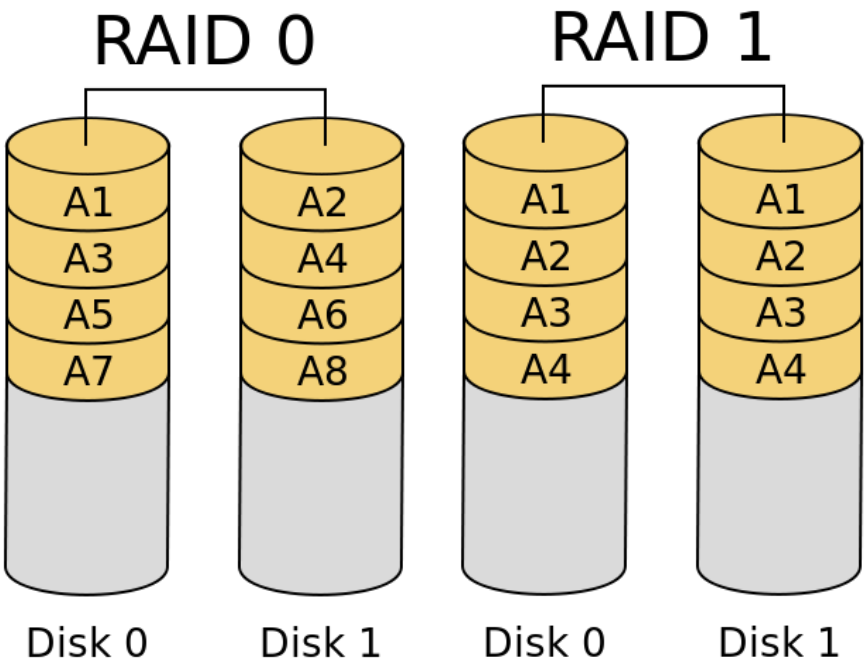
1. RAID là một tập hợp các ổ đĩa cứng (vật lý) được thiết lập theo một kỹ thuật mà hệ điều hành chỉ "nhìn thấy" chỉ là một ổ đĩa (logic) duy nhất.
2. Với cơ chế đọc/ghi thông tin diễn ra trên nhiều đĩa (ghi đan chéo hay soi gương).
3. Trong mảng đĩa có lưu các thông tin kiểm tra lỗi dữ liệu; do đó, dữ liệu có thể được phục hồi nếu có một đĩa trong mảng đĩa bị hư hỏng .

Tùy theo kỹ thuật thiết lập, RAID có thể có các mức sau:

RAID 0: Thực ra, kỹ thuật này không nằm trong số các kỹ thuật có cơ chế an toàn dữ liệu. Khi mảng được thiết lập theo RAID 0, ổ đĩa logic có được (mà hệ điều hành

nhận biết) có dung dượng bằng tổng dung lượng của các ổ đĩa thành viên. Điều này giúp cho người dùng có thể có một ổ đĩa logic có dung lượng lớn hơn rất nhiều so với dung lượng thật của ổ đĩa vật lý cùng thời điểm. Dữ liệu được ghi phân tán trên tất cả các đĩa trong mảng. Đây chính là sự khác biệt so với việc ghi dữ liệu trên các đĩa riêng lẻ bình thường bởi vì thời gian đọc-ghi dữ liệu trên đĩa tỉ lệ nghịch với số đĩa có trong tập hợp (số đĩa trong tập hợp càng nhiều, thời gian đọc – ghi dữ liệu càng nhanh). Tính chất này của RAID 0 thật sự hữu ích trong các ứng dụng yêu cầu nhiều thâm nhập đĩa với dung lượng lớn, tốc độ cao (đa phương tiện, đồ hoạ,…). Tuy nhiên, như đã nói ở trên, kỹ thuật này không có cơ chế an toàn dữ liệu, nên khi có bất kỳ một hư hỏng nào trên một đĩa thành viên trong mảng cũng sẽ dẫn đến việc mất dữ liệu toàn bộ trong mảng đĩa. Xác suất hư hỏng đĩa tỉ lệ thuận với số lượng đĩa được thiết lập trong RAID 0. RIAD 0 có thể được thiết lập bằng phần cứng (RAID controller) hay phần mềm (Stripped Applications).

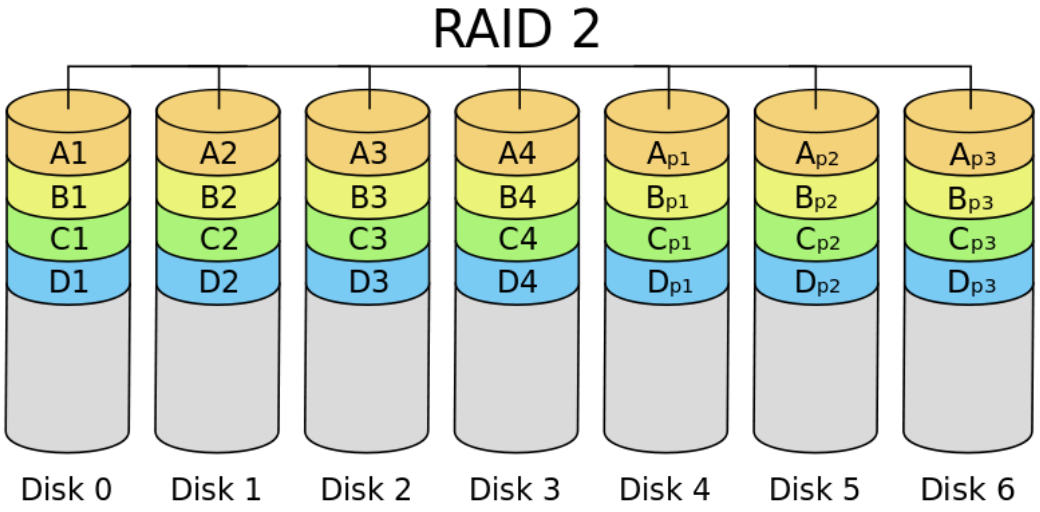
RAID 1 (Mirror - Đĩa gương): Phương cách thông thường tránh mất thông tin khi ổ đĩa bị hư là dùng đĩa gương, tức là dùng 2 đĩa. Khi thông tin được viết vào một đĩa, thì nó cũng được viết vào đĩa gương và như vậy luôn có một bản sao của thông tin. Trong cơ chế này, nếu một trong hai đĩa bị hư thì đĩa còn lại được dùng bình thường. Việc thay thế một đĩa mới (cung thông số kỹ thuật với đĩa hư hỏng) và phục hồi dữ liệu trên đĩa đơn giản. Căn cứ vào dữ liệu trên đĩa còn lại, sau một khoảng thời gian, dữ liệu sẽ được tái tạo trên đĩa mới (rebuild). RAID 1 cũng có thể được thiết lập bằng phần cứng (RAID controller) hay phần mềm (Mirror Applications) với chi phí khá lớn, hiệu suất sử dụng đĩa không cao (50%).



Hình 5.5. RAID 0 và RAID 1

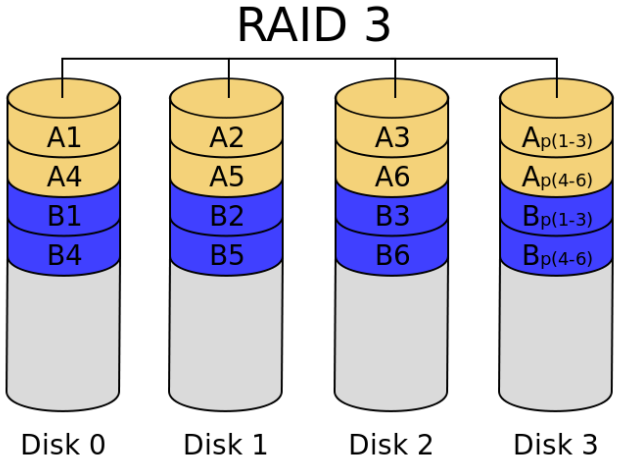
RAID 2: Dùng kỹ thuật truy cập đĩa song song, tất cả các đĩa thành viên trong RAID đều được đọc khi có một yêu cầu từ ngoại vi. Một mã sửa lỗi (ECC) được tính toán dựa vào các dữ liệu được ghi trên đĩa lưu dữ liệu, các bit được mã hóa được lưu trong các đĩa dùng làm đĩa kiểm tra. Khi có một yêu cầu dữ liệu, tất cả các đĩa được truy cập đồng thời. Khi phát hiện có lỗi, bộ điều khiển nhận dạng và sửa lỗi ngay mà không làm giảm thời gian truy cập đĩa. Với một thao tác ghi dữ liệu lên một đĩa, tất cả các đĩa dữ liệu và

đĩa sửa lỗi đều được truy cập để tiến hành thao tác ghi. Thông thường, RAID 2 dùng mã Hamming để thiết lập cơ chế mã hoá, theo đó, để mã hóa dữ liệu được ghi, người ta dùng một bit sửa lỗi và hai bit phát hiện lỗi. RAID 2 thích hợp cho hệ thống yêu cầu giảm thiểu được khả năng xảy ra nhiều đĩa hư hỏng cùng lúc.



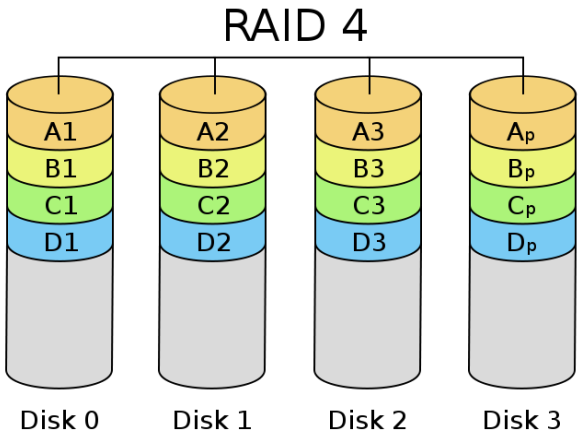
Hình 5.6. RAID 2

RAID 3: Dùng kỹ thuật ghi song song, trong kỹ thuật này, mảng được thiết lập với yêu cầu tối thiểu là 3 đĩa có các thông số kỹ thuật giống nhau, chỉ một đĩa trong mảng được dùng để lưu các thông tin kiểm tra lỗi (parity bit). Như vậy, khi thiết lập RAID 3, hệ điều hành nhận biết được một đĩa logic có dung lượng n-1/n (n: số đĩa trong mảng). Dữ liệu được chia nhỏ và ghi đồng thời trên n-1 đĩa và bit kiểm tra chẵn lẻ được ghi trên đĩa dùng làm đĩa chứa bit parity – chẵn lẻ đan chéo ở mức độ bít. Bít chẵn lẻ là một bít mà người ta thêm vào một tập hợp các bít làm cho số bít có trị số 1 (hoặc 0) là chẵn (hay lẻ). Thay vì có một bản sao hoàn chỉnh của thông tin gốc trên mỗi đĩa, người ta chỉ cần có đủ thông tin để phục hồi thông tin đã mất trong trường hợp có hỏng ổ đĩa. Khi một đĩa bất kỳ trong mảng bị hư, hệ thống vẫn hoạt động bình thường. Khi thay thế một đĩa mới vào mảng, căn cứ vào dữ liệu trên các đĩa còn lại, hệ thống tái tạo thông tin. Hiệu suất sử dụng đĩa cho cách thiết lập này là n-1/n. RAID 3 chỉ có thể được thiết lập bằng phần cứng (RAID controller).



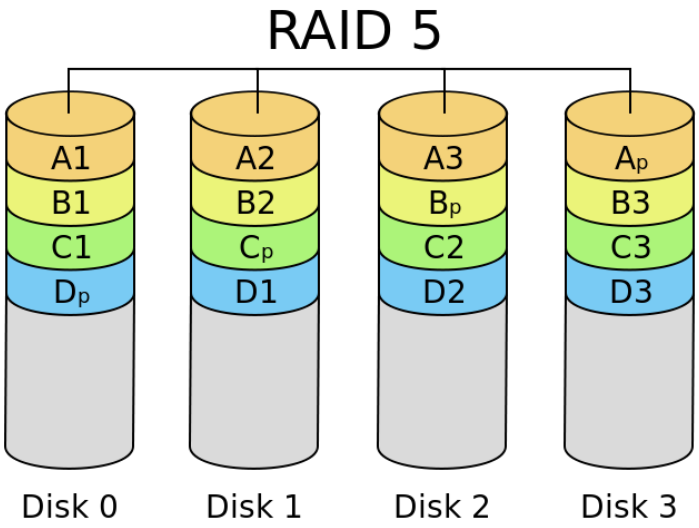
Hình 5.7. RAID 3

RAID 4: từ RAID 4 đến RAID 6 dùng kỹ thuật truy cập các đĩa trong mảng độc lập. Trong một mảng truy cập độc lập, mỗi đĩa thành viên được truy xuất độc lập, do đó mảng có thể đáp ứng được các yêu cầu song song của ngoại vi. Kỹ thuật này thích hợp với các ứng dụng yêu cầu nhiều ngoại vi là các ứng dụng yêu cầu tốc độ truyền dữ liệu cao. Trong RAID 4, một đĩa dùng để chứa các bit kiểm tra được tính toán từ dữ liệu được lưu trên các đĩa dữ liệu. Khuyết điểm lớn nhất của RAID 4 là bị nghẽn cổ chai tại đĩa kiểm tra khi có nhiều yêu cầu đồng thời từ các ngoại vi.



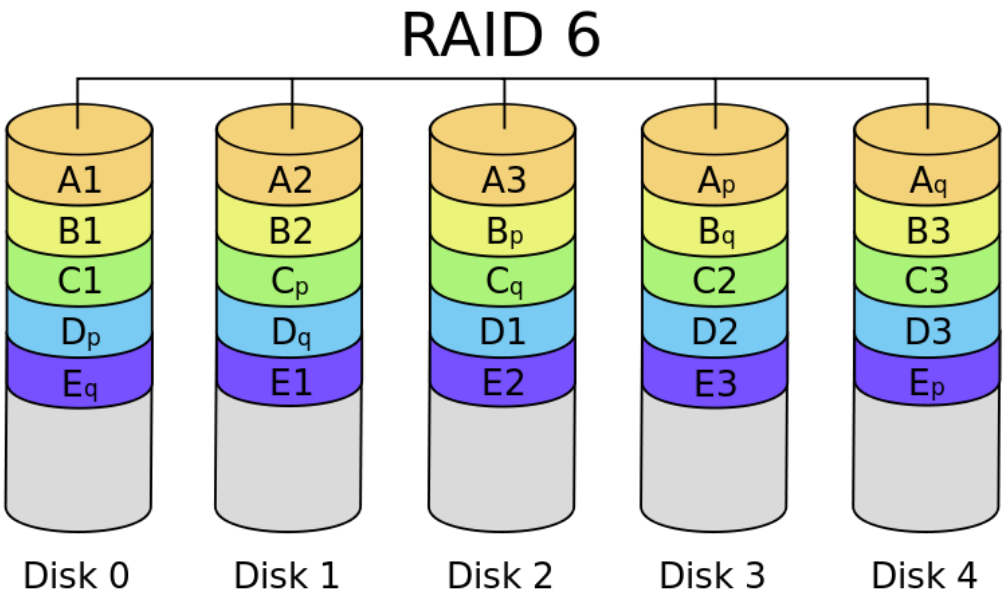
Hình 5.8. RAID 4

RAID 5: yêu cầu thiết lập giống như RAID 4, dữ liệu được ghi từng khối trên các đĩa thành viên, các bit chẵn lẻ được tính toán mức độ khối được ghi trải đều lên trên tất cả các ổ đĩa trong mảng. Tương tự RAID 4, khi một đĩa bất kỳ trong mảng bị hư hỏng, hệ thống vẫn hoạt động bình thường. Khi thay thế một đĩa mới vào mảng, căn cứ vào dữ liệu trên các đĩa còn lại, hệ thống tái tạo thông tin. Hiệu suất sử dụng đĩa cho cách thiết lập này là n-1/n. RAID 5 chỉ có thể được thiết lập bằng phần cứng (RAID controller). Cơ chế này khắc phục được khuyết điểm đã nêu trong cơ chế RAID 4.



Hình 5.9. RAID 5

RAID 6: Trong kỹ thuật này, cần có n+2 đĩa trong mảng. Trong đó, n đĩa dữ liệu và 2 đĩa riêng biệt để lưu các khối kiểm tra. Một trong hai đĩa kiểm tra dùng cơ chế kiểm tra như trong RAID 4&5, đĩa còn lại kiểm tra độc lập theo một giải thuật kiểm tra. Qua đó, nó có thể phục hồi được dữ liệu ngay cả khi có hai đĩa dữ liệu trong mảng bị hư hỏng.



Hình 5.10. RAID 6

Hiện nay, RAID 0,1,5 được dùng nhiều trong các hệ thống. Các giải pháp RAID trên đây (trừ RAID 6) chỉ đảm bảo an toàn dữ liệu khi có một đĩa trong mảng bị hư hỏng. Ngoài ra, các hư hỏng dữ liệu do phần mềm hay chủ quan của con người không được đề cập trong chương trình. Người dùng cần phải có kiến thức đầy đủ về hệ thống để các hệ thống thông tin hoạt động hiệu quả và an toàn.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Mô tả vận hành của ổ đĩa cứng. Cách lưu trữ thông tin trong ổ đĩa cứng
2. Mô tả các biện pháp an toàn trong việc lưu trữ thông tin trong đĩa cứng.
3. Nguyên tắc vận hành của đĩa quang. Ưu khuyết điểm của các loại đĩa quang.
4. Thông thường có bao nhiêu loại bus? Tại sao phải có các chuẩn cho các bus vào ra?
5. Thế nào là chủ nhân của bus? Khi bus có nhiều chủ nhân thì làm thế nào để giải quyết tranh chấp bus?
6. Giải thích việc nới rộng dãi thông bằng cách sử dụng các gói tin.
7. Sự khác biệt giữa bộ xử lý vào ra và bộ xử lý trung tâm của máy tính.
8. Tìm hiểu các thông tin liên quan đến ổ cứng
    a. Giao tiếp: loại giao tiếp (PATA, SATA, SCSI), các phiên bản, tốc độ
    b. Công nghệ S.M.A.R.T
    c. Khái niệm Partition, cách tổ chức phân vùng trên đĩa
    d. Format: khái niệm, các loại định dạng (FAT, NTFS, …)
    e. Tìm hiểu về ổ cứng SSD
    f. Tìm hiểu về ổ cứng lai
9. Tìm hiểu các thông tin liên quan đến đĩa quang
    a. Cách đọc, ghi thông tin

b. Các chuẩn CD

c. Các chuẩn DVD

10. Tìm hiểu các thông tin liên quan đến thẻ nhớ

a. Các loại thẻ nhớ: CompactFlash Type I/II (CF), Microdrive, Secure Digital (SD), miniSD, Micro SD, MultiMediaCard (MMC), RS-MMC, Micro MMC, Memory Stick, Memory Stick PRO, Memory Stick Duo, Memory Stick PRO Duo, xD

b. Các tiêu chuẩn tốc độ của thẻ SD (SD Class)

11. Tìm hiểu thông tin các chuẩn bus

a. PCI

b. PCI-Express

c. USB

d. eSATA

e. IEEE 1394

f. Thunderbolt

12. Tìm hiểu thông tin về các mức RAID

Chương 6

TỔNG QUAN VỀ LẬP TRÌNH HỢP NGỮ

6.1. NGÔN NGỮ MÁY

Chương trình là tập các lệnh được đưa vào bộ nhớ cho máy thực hiện, chương trình có nhiều dạng ngôn ngữ khác nhau. Dạng cơ bản nhất mà máy (CPU) có thể hiểu và thực thi được gọi là ngôn ngữ máy (Machine Language). Tùy theo CPU mà ngôn ngữ máy có một dạng nhất định, có nghĩa là mỗi loại CPU có một loại ngôn ngữ riêng.

Ví dụ: một đoạn chương trình ngôn ngữ máy thuộc họ CPU Intel 8006/8088 được biểu diễn dưới dạng số nhị phân là một dãy các byte như sau:

```
10110100 00000010 10000000 11000010 00110010 01010000
```

Đoạn chương trình trên gồm có 3 lệnh có chiều dài lần lượt là 2, 3, 1 byte. Chiều dài của lệnh là do CPU qui định.

Vì là ngôn ngữ riêng của máy nên chương trình viết bằng ngôn ngữ thực hiện rất nhanh và chiếm ít chỗ trong bộ nhớ, tuy nhiên chương trình rất khó viết và rất khó nhớ các lệnh.

6.2. HỢP NGỮ (ASSEMBLY LANGUAGE)

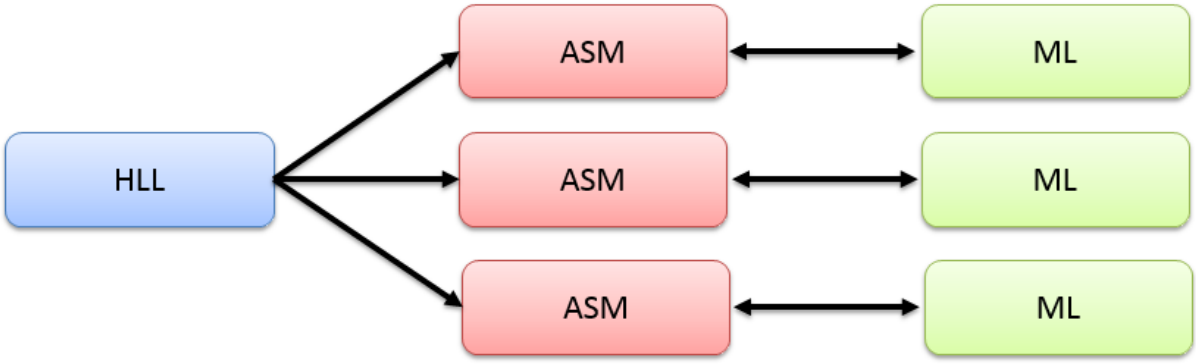
6.2.1. Khái niệm

Hợp ngữ là loại ngôn ngữ giúp lập trình viên viết chương trình dễ dàng hơn thay cho ngôn ngữ máy. Hợp ngữ có dạng như ngôn ngữ máy tức là một lệnh hợp ngữ tương đương với một lệnh ngôn ngữ máy (có thể có một lệnh hợp ngữ tương đương với nhiều lệnh ngôn ngữ máy gọi là lệnh vĩ mô), nhưng khác với ngôn ngữ máy ở chỗ thay vì viết chương trình dưới mã máy (thường là dạng nhị phân), người ta dùng một số ký hiệu tượng trưng cho dễ nhớ.

Ví dụ: Lệnh hợp ngữ: MOV AH, 2

Tương đương với lệnh sau của CPU Intel 8086/8088 dưới dạng mã máy: 10110100

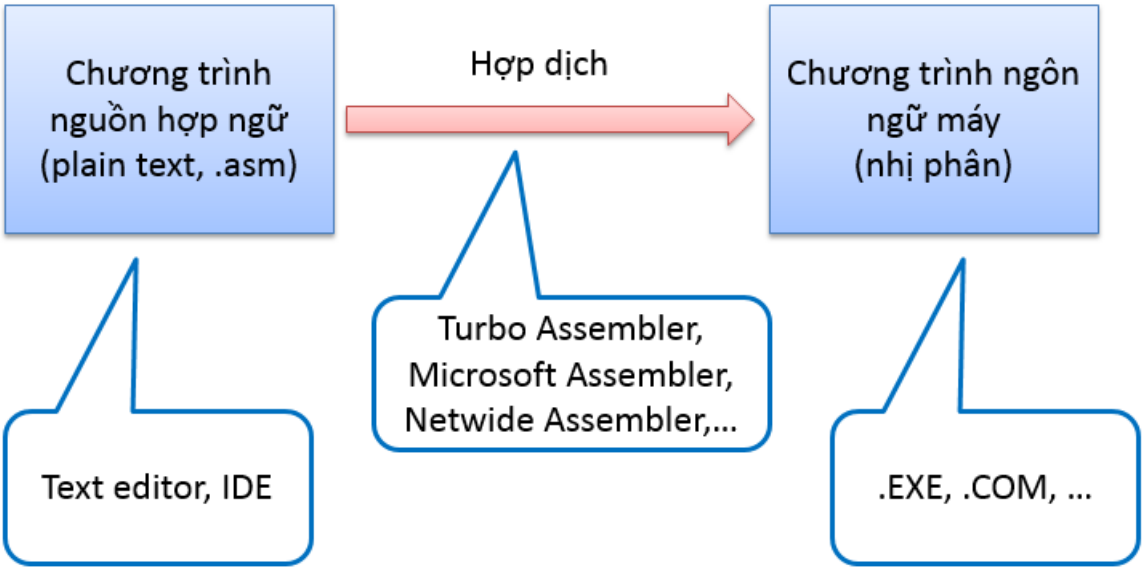
Hợp ngữ được gọi là ngôn ngữ cấp thấp bởi vì nó sát với ngôn ngữ máy theo cấu trúc và chức năng. Mỗi một lệnh hợp ngữ tương ứng với một lệnh ngôn ngữ máy (tương quan một – một). Ngược lại, mỗi một lệnh của ngôn ngữ cấp cao – High Level Language (Pascal, Basic, C, C++, …) tương ứng với nhiều lệnh mã máy.



Hình 6.1. Phát sinh ngôn ngữ máy bởi các chương trình hợp ngữ và ngôn ngữ cấp cao

6.2.2. Trình hợp dịch (Assembler)

Do máy không thể hiểu được chương trình viết bằng hợp ngữ nên phải qua giai đoạn để dịch chương trình từ hợp ngữ ra ngôn ngữ máy. Chương trình làm nhiệm vụ dịch các chương trình viết bằng hợp ngữ ra ngôn ngữ máy gọi là trình hợp dịch (Assembler). Chương trình viết bằng hợp ngữ gọi là chương trình nguồn và chương trình dưới dạng ngôn ngữ máy được dịch từ chương trình nguồn gọi là chương trình đích.



6.3. ỨNG DỤNG CỦA HỢP NGỮ

Hợp ngữ có sự tương ứng trực tiếp với ngôn ngữ máy nên viết chương trình bằng hợp ngữ có những ưu điểm sau:

- Tốc độ: chương trình viết bằng hợp ngữ luôn đạt tốc độ cao nhất do được viết rất sát phần cứng máy mà nó thực hiện.'
- Tiết kiệm bộ nhớ: chương trình hợp ngữ rất nhỏ gọn do rất ít có các lệnh dư thừa và người lập trình kiểm soát trực tiếp dữ liệu trong bộ nhớ.
- Khả năng: lập trình bằng hợp ngữ cho phép ta kiểm soát trực tiếp phần cứng máy tính do đó không gặp phải những giới hạn thường gặp trong ngôn ngữ cấp cao khi sử dụng phần cứng.
- Ngoài ra lập trình bằng hợp ngữ giúp ta tìm hiểu sâu thêm về kiến trúc máy tính, hệ điều hành, các thiết bị phần cứng. Điều này giúp ích cho ta rất nhiều khi viết chương trình, ngay cả khi viết bằng ngôn ngữ cấp cao.

Tuy nhiên, ngày nay rất khó thấy các chương trình được viết hoàn toàn bằng hợp ngữ. Điều đó xuất phát từ những nguyên nhân sau:

- Hợp ngữ rất khó học, rất khó để đọc và hiểu một chương trình hợp ngữ.
- Hợp ngữ đòi hỏi phải tập trung chú ý vào từng chi tiết nhỏ. Do đó phải mất quá nhiều thời gian để viết và bảo trì chương trình.

- Hợp ngữ phụ thuộc vào nền phần cứng mà nó thực hiện do đó một chương trình hợp ngữ viết trên hệ phần cứng này không thể đem sang thực hiện trên một hệ phần cứng khác.
- Tốc độ máy tính và dung lượng bộ nhớ hiện nay rất cao cùng với sự phát triển của các chương trình dịch (compiler), cải tiến các giải thuật cho phép các chương trình có thể chạy nhanh và sinh mã máy hiệu quả mà không cần thiết phải sử dụng hợp ngữ.

Thay vào đó, hợp ngữ thường được dùng để tối ưu hóa một số đoạn nhất định của các trình ứng dụng để tăng tốc và để truy cập phần cứng máy tính (dưới dạng các chương trình con). Hợp ngữ cũng được dùng khi viết các chương trình trong hệ thống nhúng (embedded system) – là những chương trình lưu trữ trong các chip ROM của các thiết bị phần cứng, hoặc được dùng để viết các chương trình điều khiển thiết bị (device driver) của hệ điều hành.

6.4. CÁC KIỂU DỮ LIỆU

Mục đích chính của máy tính là lưu trữ, truy tìm và xử lý dữ liệu. Do đó ta sẽ khảo sát cách thức máy tính lưu trữ dữ liệu trong bộ nhớ.

6.4.1. Số học

Hầu hết các hệ thống máy tính hiện nay đều không sử dụng hệ thống số thập phân để biểu diễn các con số. Thay vào đó là hệ thống số nhị phân (binary), trong hệ thống này mỗi ký số chỉ có thể là 0 hay 1, tương ứng với trạng thái on/off của mạch điện tử.

Ghi chú:

- Trong hợp ngữ, để biểu diễn hằng số ở hệ 2, 16 ta ghi giá trị kèm theo ký tự b/B, h/H). Ví dụ: 11001010b, 3Fh
- Nếu hằng hệ 16 bắt đầu bằng ký tự thì phải thêm 0 vào trước. Ví dụ: 0F8H

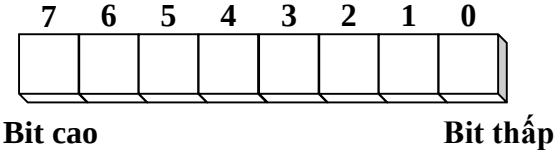
Bit

Biểu diễn một ký số nhị phân 0 hoặc 1.

Byte

Có chiều dài là 8 bit, các bit được đanh số từ 0 đến 7 từ phải sang trái, bit 7 là bit cao, bit 0 là bit thấp, dùng để biểu diễn:

- Một số nguyên không dấu có giá trị từ 0 đến 255 (nhị phân không dấu).
- Một số nguyên có dấu có giá trị từ -128 đến 127 (nhị phân có dấu theo kiểu bù hai).

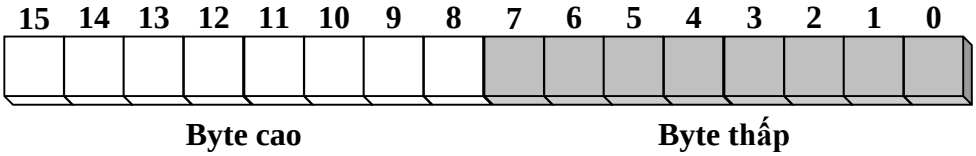


Word

Có chiều dài 16 bit, các bit được đánh số từ 0 đến 15, bit 15 là bit cao, bit 0 là bit thấp, byte bên phải gọi là byte thấp, byte bên trái gọi là byte cao dùng để biểu diễn:

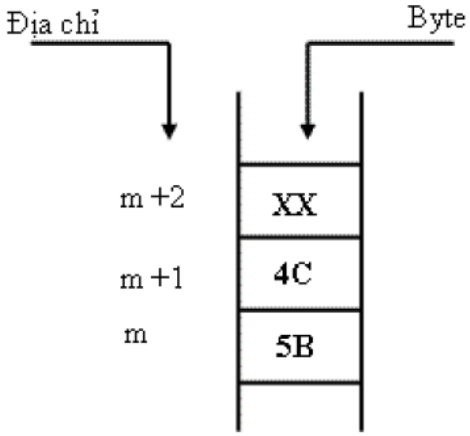
- Một số nguyên không dấu có giá trị từ 0 đến 65535.

- Một số nguyên có dấu có giá trị từ -32768 đến +32767.



Khi lưu trữ trong bộ nhớ, byte thấp sẽ lưu ở địa chỉ thấp, byte cao được lưu ở địa chỉ cao.

Ví dụ: word 4C5B được lưu trong bộ nhớ tại địa chỉ bắt đầu là m như sau: byte thấp 5B được lưu tại ô nhớ m, byte cao 4C được lưu tại ô nhớ m+1.



Double Word

Có chiều dài 32 bit, các bit được đánh số từ 0 đến 31, bao gồm 2 word 16 bit. Word thấp chiếm các bit từ 0 đến 15, word cao chiếm các bit từ 16 đến 31. Double word có thể chứa một số nguyên có dấu từ -2147483648 đến 2147483647 hoặc một số nguyên không dấu từ 0 đến 4294967295.

Số BCD không nén (Binary – Coded Decimal unpacked)

Gồm nhiều byte, mỗi byte biểu diễn một ký số thập phân.

Ví dụ: số 1234 thập phân lưu trong bộ nhớ theo dạng BCD không nén sẽ chiếm 4 byte liên tiếp nhau có giá trị lần lượt là 1, 2, 3, 4.

Số BCD nén (Binary – Coded Decimal packed)

Gồm nhiều byte, mỗi byte biểu diễn hai ký số thập phân, mỗi 4 bit (nible) biểu diễn một ký số thập phân.

Ví dụ: số 1234 thập phân lưu trong bộ nhớ theo dạng BCD nén sẽ chiếm 2 byte liên tiếp nhau có giá trị như sau:

```
0001 0010 0011 0100
```

6.4.2. Ký tự

Các máy tính chỉ lưu trữ các số nhị phân, do đó để lưu trữ các ký tự máy tính phải sử dụng một lược đồ mã hóa ký tự phiên dịch các ký tự thành các con số và ngược lại. Bảng mã được sử dụng phổ biến nhất của các máy tính tên là ASCII (American Standard Code for Information Interchange). Trong bảng mã ASCII, mỗi ký tự chiếm 1 byte và

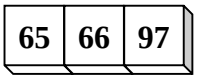
được gán một trị số duy nhất, kể cả các ký tự điều khiển dùng khi in hoặc truyền dữ liệu giữa các máy tính.

Ví dụ: các ký tự A, B, a trong bảng mã ASCII có các trị số lần lượt là 65, 66, 97

Một chuỗi ký tự là một dãy các byte liên tiếp, mỗi byte chứa mã ASCII biểu diễn cho một ký tự trong chuỗi. Chuỗi được lưu trong bộ nhớ từ địa chỉ thấp đến địa chỉ cao.

Hằng chuỗi nằm giữa hai dấu nháy đơn hay nháy kép.

Ví dụ: chuỗi ký tự "Aba" lưu trong bộ nhớ gồm 3 byte liên tiếp nhau như sau:



6.4.3. Chú ý

Trong lĩnh vực lập trình ngôn ngữ cấp cao, ta có khuynh hướng xem dữ liệu và mã lệnh là khác nhau. Tuy nhiên, về mặt vật lý, cả hai đều là những chuỗi bit nhị phân trong bộ nhớ. Ví dụ: chuỗi bit 010000110101110010001001 có thể là 3 byte biểu diễn cho một chuỗi, 3 byte nhớ biểu diễn cho 3 số nguyên hoặc cũng có thể là 3 byte mã của một lệnh máy.

Mặt khác, trong hợp ngữ sự phân biệt giữa các kiểu dữ liệu cũng chỉ có tính tương đối. Ví dụ: một chuỗi bit 1010001100010010 có thể xem là byte thấp và byte cao của 1 word, 2 byte số nguyên riêng lẻ, hoặc là một chuỗi 2 ký tự.

Máy tính không thể tự phân biệt được những trường hợp trên. Do đó khi lập trình hợp ngữ, trách nhiệm của lập trình viên là phải giữ tách biệt được mã và dữ liệu đồng thời kiểm soát được dữ liệu trong bộ nhớ - đây là phần việc đã được các ngôn ngữ lập trình cấp cao tự thực hiện.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Nêu hai kiểu chương trình hoặc ứng dụng thích hợp với hợp ngữ hơn so với ngôn ngữ cấp cao.
2. Cho biết ý nghĩa của mối quan hệ một – nhiều khi so sánh ngôn ngữ cấp cao với ngôn ngữ máy.
3. Tại sao việc tìm hiểu hệ điều hành lại quan trọng khi ta nghiên cứu về hợp ngữ?
4. Tính năng kiểm tra dữ liệu tỏ ra mạnh hơn trong hợp ngữ hay trong ngôn ngữ cấp cao?
5. Có phải chương trình viết trên bộ xử lý Intel 8086/8088 đều chạy được trên các bộ xử lý 386, 486, Pentium và ngược lại? Tại sao?
6. Tại sao hợp ngữ không được dùng khi viết các ứng dụng lớn?
7. Thực hiện chuyển đổi số nguyên giữa các cơ hệ thập phân, nhị phân, thập lục phân.
8. Sử dụng ngôn ngữ cấp cao (Pascal, C, …) viết các chương trình chuyển đổi số giữa các cơ hệ thập phân, nhị phân, thập lục phân.
9. Thực hiện các phép tính cơ bản (cộng, trừ) ở hệ nhị phân, thập lục phân.
10. Tính số lượng bit nhị phân cần thiết để biểu diễn một số nguyên.

Chương 7

TỔ CHỨC BỘ XỬ LÝ INTEL 8086/8088

7.1. TỔ CHỨC BỘ NHỚ

Bộ nhớ chính của 8086/8088 là một dãy các byte liên tiếp nhau được đánh số từ 0 đến 20^{20}-1 (00000h đến FFFFFh). Số thứ tự đó gọi là địa chỉ (dài 20 bit) và tổng cộng bộ nhớ có 1MB.

Để tận dụng bộ nhớ và thuận lợi trong xử lý, người ta chia bộ nhớ thành từng đoạn (segment), mỗi đoạn bắt đầu từ địa chỉ chia hết cho 16 và có chiều dài tối đa 64KB. Lúc đó vị trí một byte trong bộ nhớ được xác định bằng một segment:offset (gọi là địa chỉ tương đối, vị trí thực trong bộ nhớ gọi là địa chỉ tuyệt đối). Segment và offset là một số nhị phân 16 bit có trị từ 0 đến FFFFh.

Như vậy địa chỉ của một ô nhớ bất kỳ trong bộ nhớ 1MB được xác định bằng các tổ hợp địa chỉ 16 bit địa chỉ segment và 16 bit địa chỉ offset.

Địa chỉ vật lý

Là đại lượng có chiều dài 20 bit xác định vị trí của một ô nhớ trong 1MB bộ nhớ.

Địa chỉ logic

Gồm 2 phần, mỗi phần có chiều dài 16 bit lần lượt biểu diễn địa chỉ segment và offset của một ô nhớ. Ký hiệu:

Segment : Offset

Ví dụ: với địa chỉ vật lý FFFFEh ta có thể định vị theo địa chỉ logic là F000:FFFEh hoặc FFFF:000Eh

Người lập trình chỉ có thể truy xuất các ô nhớ thông qua địa chỉ logic, còn địa chỉ vật lý chỉ dành riêng cho CPU sử dụng.

Cách chuyển từ địa chỉ logic sang địa chỉ vật lý

Bằng cách dịch địa chỉ segment sang trái 4 bit rồi cộng với địa chỉ offset.

Ví dụ: ta có địa chỉ logic 1134:1032h thì địa chỉ vật lý là:

1134h dịch trái 4 bit → 11340h

11340h + 1032h → 12372h

Một ô nhớ chỉ có duy nhất một địa chỉ vật lý, nhưng ta có thể dùng nhiều địa chỉ logic khác nhau để định vị cùng một ô nhớ.

Ví dụ: hai địa chỉ logic sau:

35B7:1000h

36B7:0000h

→ Biểu thị cùng một địa chỉ vật lý 36B70h

Sự phân bố các đoạn trong bộ nhớ

Các đoạn bộ nhớ có thể chồng lên nhau, có thể nối tiếp nhau, có thể tách rời nhau.

Việc phân bộ nhớ thành từng đoạn làm cho việc tổ chức chương trình và dữ liệu khá mềm dẻo. Chương trình không cần phải viết như một dãy liên tục các chỉ thị (instruction) và dữ liệu mà có thể là các đơn thể (module) gồm mã và dữ liệu.

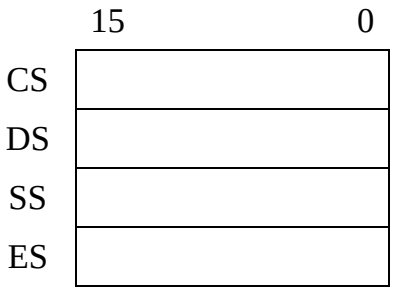
Dữ liệu cũng có thể tổ chức thành các cấu trúc dữ liệu khác nhau: dữ liệu dùng riêng, dữ liệu dùng chung với các chương trình khác trong hệ thống. Mỗi đơn thể mã và dữ liệu này có thể có kích thước khác nhau.

7.2. TỔ CHỨC THANH GHI

Thanh ghi (register) là nơi dữ liệu bên trong CPU tùy theo độ dài 8 bit hay 16 bit và tùy theo chức năng khi đó thanh ghi được dùng để chứa dữ liệu sẽ thao tác hoặc kết quả các phép tính hoặc các địa chỉ dùng để định vị ô nhớ khi cần thiết. Có tất cả 14 thanh ghi, mỗi thanh ghi dài 16 bit chia thành 5 nhóm:

7.2.1. Nhóm thanh ghi đoạn (segment register)

Gồm 4 thanh ghi: đoạn mã CS, đoạn dữ liệu DS, đoạn bổ sung ES và đoạn stack SS. Đó là những thanh ghi chứa địa chỉ segment của các ô nhớ khi cần truy xuất.



Thanh ghi đoạn mã CS (Code Segment): Lưu địa chỉ segment chứa chương trình ngôn ngữ máy đang thực thi.

Thanh ghi đoạn dữ liệu DS (Data Sengment): Lưu địa chỉ segment của đoạn dữ liệu trong chương trình.

Thanh ghi đoạn bổ sung ES (Extra Segmnent): Lưu địa chỉ segment của đoạn dữ liệu bổ sung.

Thanh ghi đoạn Stack SS (Strack Segment): Lưu địa chỉ segment của đoạn stack.

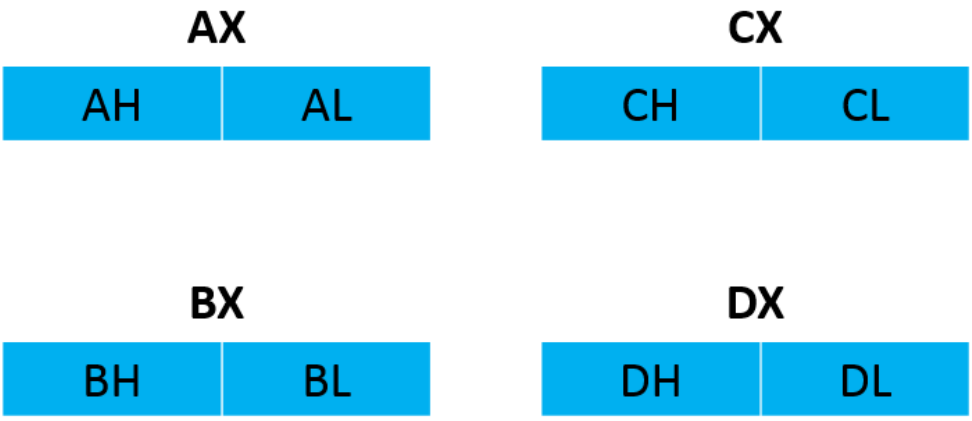
Bốn thanh ghi này có thể truy xuất dữ liệu trên bốn đoạn khác nhau và một chương trình chỉ có thể sử dụng cùng một lúc tối đa bốn đoạn

Trên CPU 80386 còn có hai thanh ghi chức năng tương tự như ES là FS và GS.

7.2.2. Nhóm thanh ghi đa dụng (general segister)

Gồm bốn thanh ghi AX, BX, CX, DX. Các thanh ghi này có thể xem như một thanh ghi 16 bit hoặc hai thanh ghi mỗi thanh ghi 8 bit

Đối với CPU 80386 các thanh ghi đa dụng có thể kéo dài đến 32 bit tạo thành thanh ghi EAX, EBX, ECX, EDX.



Thanh ghi tích lũy AX (Accumulator register): Thường dùng để lưu số nhân, số chia trong các phép toán nhân hoặc chia, các phép tính số học, logic và chuyển dữ liệu.

Ví dụ: `MUL BH ;AX ← AL * BH`

Thanh ghi cơ sở BX (Base register): Thường dùng để định vị bộ nhớ

Ví dụ: `MOV [BX], AX`

→ Lấy nội dung thanh ghi AX đưa vào ô nhớ có địa chỉ segment là DS và địa chỉ offset BX

Thanh ghi đếm CX (Count register): Dùng để định số lần lặp của vòng lặp.

Thanh ghi dữ liệu DX (Data register): Dùng để lưu kết quả của các phép toán nhân và chia, định địa chỉ cổng trong các lệnh nhập xuất cổng.

Ví dụ:
```
MOV AL, 62 ;AL ← 62
MOV DX, 1000 ; DX ← 1000
OUT DX, AL; Đưa nội dung của AL ra cổng 1000
```

7.2.3. Nhóm thanh ghi con trỏ và chỉ mục (pointer and index register)

Gồm bốn thanh ghi: SI, DI, BP, SP. Các thanh ghi này là thanh ghi 16 bit dùng làm con trỏ hay chỉ mục trong các phép toán truy xuất bộ nhớ trong. Đối với CPU 80386, các thanh ghi này có thể kéo dài đến 32 bit thành các thanh ghi ESI, EDI, EBP, ESP.

Chỉ mục nguồn SI (Source Index): Dùng lưu địa chỉ bộ nhớ. Đối với thao tác trên chuỗi nó dùng để ghi địa chỉ của một chuỗi nguồn.

Ví dụ: `MOV AL, [SI]`

→ Lấy nội dung ô nhớ có địa chỉ offset trong SI, địa chỉ segment DS đưa vào thanh ghi AL.

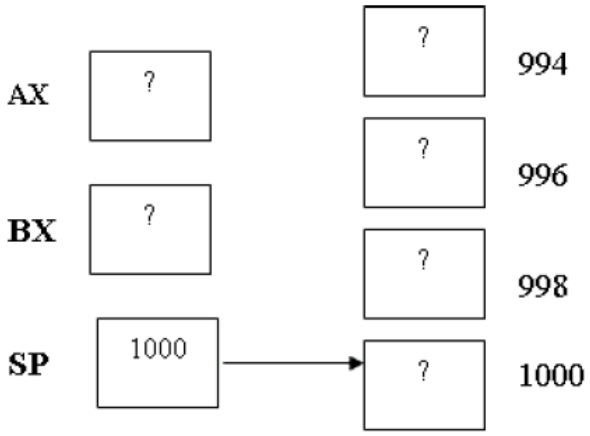
Chỉ mục đích DI (Destination Index): Dùng lưu địa chỉ bộ nhớ. Đối với thao tác trên chuỗi nó dùng để ghi địa chỉ của chuỗi đích.

Con trỏ stack (Stack Pointer): Dùng làm con trỏ để chỉ đến phần tử ở đỉnh stack. Stack là một cấu trúc dữ liệu đặc biệt lưu trữ dữ liệu theo kiểu LIFO (Last In First Out) tức là dữ liệu đưa vào sau lại được lấy ra trước. Các phần tử được đưa vào hoặc lấy ra khỏi stack đều tính theo đơn vị word. Nội dung của SP là địa chỉ của đỉnh stack. Lệnh PUSH dùng đưa trị vào đỉnh stack và POP là lấy trị ra khỏi đỉnh stack.

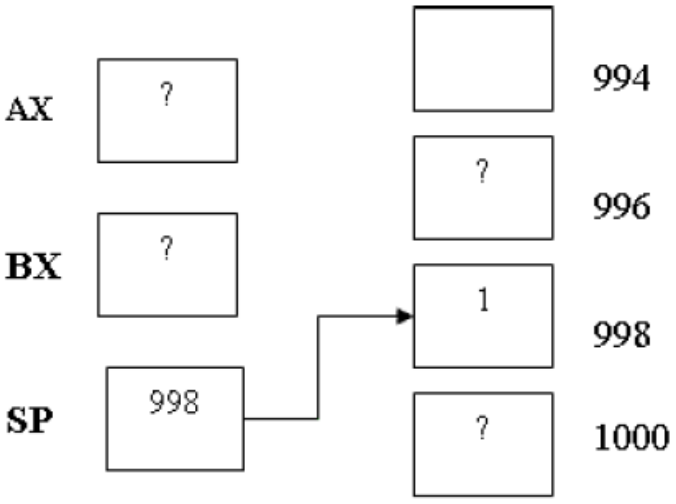
Ví dụ:

```
MOV AX, 1 ; AX ← 1 (1)
PUSH AX ; Đưa AX vào đỉnh stack (2)
MOV BX, 2 ; BX ← 2 (3)
PUSH BX ; Đưa BX vào đỉnh stack (4)
POP AX ; (5)
; Lấy trị ở đỉnh stack đưa vào AX (tức là 2)
POP BX ; (6)
; Lấy trị ở đỉnh stack đưa vào BX (tức là 1)
```

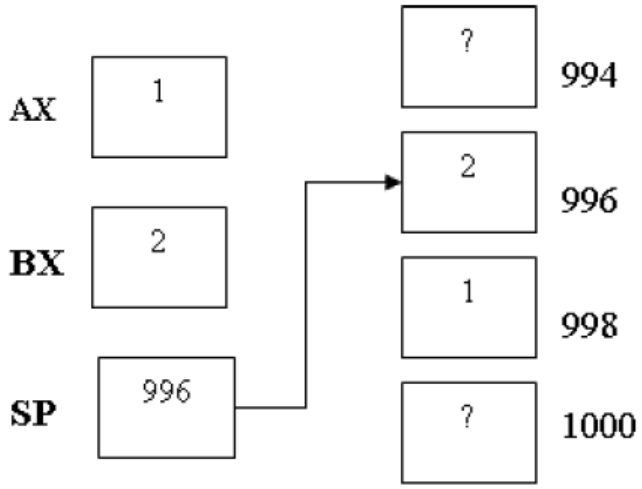
Trạng thái ban đầu:



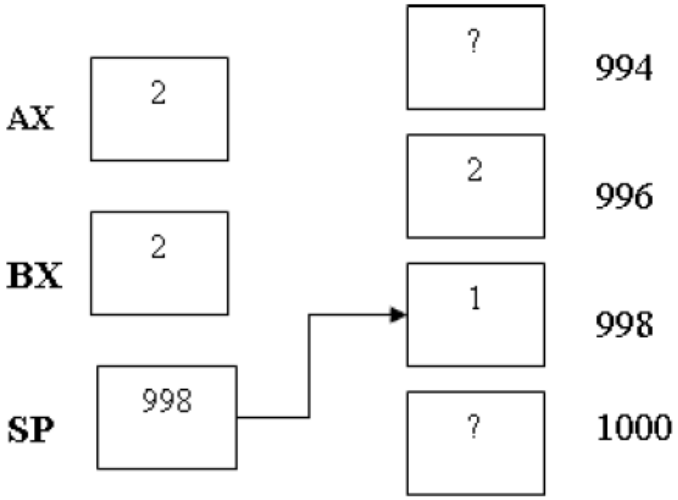
Sau khi thực hiện (1) và (2):



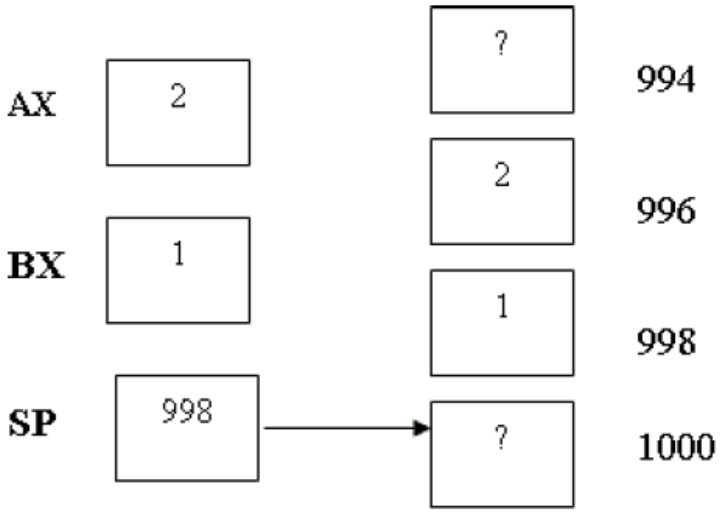
Sau khi thực hiện (3) và (4):



Sau khi thực hiện (5):



Sau khi thực hiện (6):



Con trỏ cơ sở BP (Base Pointer): Dùng trong các phép định địa chỉ cơ sở khi truy xuất stack.

7.2.4. Thanh ghi con trỏ lệnh chương trình IP (Instruction Pointer)

Chứa địa chỉ offset (phối hợp với địa chỉ segment trong CS) của ô nhớ chứa mã lệnh của lệnh kế tiếp sẽ được CPU thi hành. Khi CPU thực hiện một lệnh, IP tự động thay đổi để chỉ đến địa chỉ offset của ô nhớ chứa lệnh được CPU thi hành kế tiếp.

7.2.5. Thanh ghi cờ hiệu (Flag register)

Dài 16 bit ghi nhận các thông tin về trạng thái của CPU và kết quả thực hiện lệnh sau cùng. Mỗi bit là một cờ, cờ có thể có trị 1 gọi là trạng thái Set hoặc có trị 0 gọi là trạng thái Clear. CPU dùng 9 bit làm cờ (các vị trí khác dùng cho các CPU khác hoặc mở rộng về sau).

Để bật trạng thái set ta dùng lệnh:

```
ST <Ký_Tự_Đầu_Của_Cờ>
```

Còn trạng thái clear ta dùng lệnh:

```
CL <Ký_Tự_Đầu_Của_Cờ>
```

Ta có thể chia làm hai nhóm:

a) Nhóm các cờ trạng thái: Sáu cờ hiệu CF, AF, PF, ZF, SF, OF. Các cờ này có thể bị ảnh hưởng mỗi khi CPU thực hiện xong một lệnh.

Cờ nhớ CF (Carry Flag): Được bật lên 1 nếu kết quả của phép toán vừa thực hiện có nhớ hay có mượn

Ví dụ:

```
AL = 10100111 = A7h
BL = 10010101 = 95h
ADD AL, BL ; AL ← AL + BL
 10100111 + 10011110 = 100111100
```

→ Kết quả: Al = 00111100 = 3Ch có nhớ 1 nên cờ CF = 1

Cờ phụ AF (Auxiliary Flag): Được bật lên 1 nếu kết quả phép toán vừa thực hiện có nhớ hay có mượn đối với 4 bit thấp. Thường dùng trong phép tính các số BCD.

Cờ zero ZF (Zero Flag): Được bật lên 1 nếu kết quả của phép toán vừa thực hiện bằng zero.

Cờ dấu SF (Sign Flag): Có giá trị tương ứng với bit cao nhất của kết quả của phép toán vừa thực hiện. Thường dùng để xác định kết quả của phép toán là số âm hay số dương.

Cờ chẵn lẻ PF (Parity Flag): Được bật lên 1 nếu kết quả của phép toán có tổng 8 bit thấp là 1 số chẵn.

Cờ tràn OF (Overflow Flag): Được bật lên 1 nếu kết quả của phép toán các số có dấu bị sai.

b) Nhóm các cờ hiệu điều khiển: Ba cờ hiệu DF, TF, IF.

Cờ định hướng DF (Direction Flag): Dùng để quy định chiều chuyểnvdữ liệu trong các lệnh xử lý chuỗi.

Cờ TF (Trap Flag): Cho phép CPU chuyển sang chế độ thực hiện tuần tự từng lệnh của chương trình

Cờ ngắt IF (Interrupt Flag):

- IF = 1: Các ngắt cứng phát sinh sẽ được tiếp nhận (Enable).
- IF = 0: Các ngắt cứng phát sinh sẽ không được tiếp nhận (Disable).

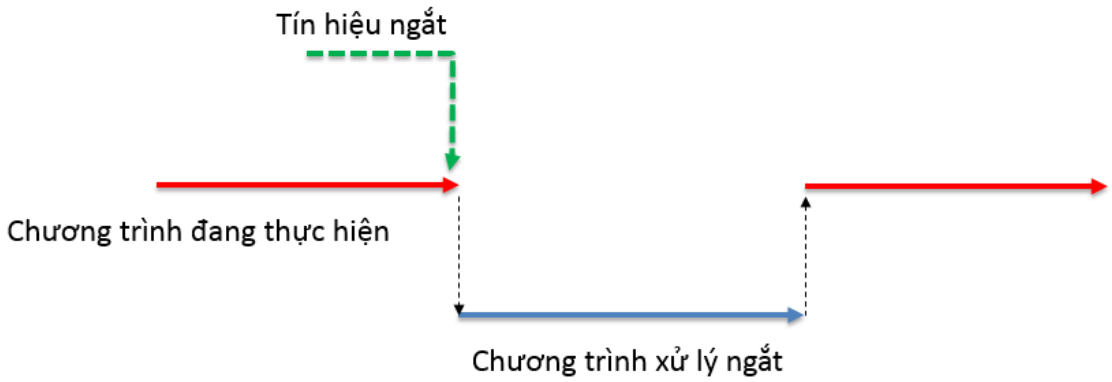
<table>
<tr><td>AH</td><td>AL</td><td>AX</td><td rowspan="4">Nhóm thanh ghi đa dụng</td></tr>
<tr><td>BH</td><td>BL</td><td>BX</td></tr>
<tr><td>CH</td><td>CL</td><td>CX</td></tr>
<tr><td>DH</td><td>DL</td><td>DX</td></tr>
<tr><td colspan="3">CS</td><td rowspan="4">Nhóm thanh ghi đoạn</td></tr>
<tr><td colspan="3">DS</td></tr>
<tr><td colspan="3">SS</td></tr>
<tr><td colspan="3">ES</td></tr>
<tr><td colspan="3">SP</td><td rowspan="2">Nhóm thanh ghi con trỏ</td></tr>
<tr><td colspan="3">BP</td></tr>
<tr><td colspan="3">SI</td><td rowspan="2">Nhóm thanh ghi chỉ mục</td></tr>
<tr><td colspan="3">DI</td></tr>
<tr><td colspan="3">IP</td><td>Thanh ghi con trỏ lệnh</td></tr>
<tr><td colspan="3">Flag</td><td>Thanh ghi cờ hiệu</td></tr>
</table>

Hình 7.1. Các thanh ghi của CPU Intel 8086/8088

7.3. NGẮT QUÃNG (INTERRUPT)

7.3.1. Định nghĩa

Là tín hiệu được gửi đến bộ vi xử lý để tạm ngưng chương trình đang thực hiện và xử lý một nhiệm vụ khác, sau khi xử lý xong nhiệm vụ, thì chương trình bị ngắt sẽ được tiếp tục thực hiện sau vị trí ngắt.



7.3.2. Phân loại ngắt

a. Ngắt cứng (hard interrupt)

Được phát sinh bởi các mạch của máy tính khi đáp lại một sự kiện nào đó. Nó được dùng để điều khiển các thiết bị quan trọng như bàn phím, ổ đĩa, máy in… CPU gồm 8 ngắt cứng IRQ (Interrupt Request)

IRQ0: Timer.

IRQ1: Keyboard.

IRQ2: Dùng trong máy AT.

IRQ3: Serial port 2.

IRQ4: Serial port 1.

IRQ5: Harddisk.

IRQ6: Floppy disk.

IRQ7: Parallet port.

Độ ưu tiên được gắn theo thứ tự IRQ0 xuống IRQ7. Các ngắt này có thể chắn được bằng cách dùng lệnh CLI, lúc đó cờ hiệu IF = 0 và các ngắt này khi phát sinh sẽ không tiếp nhận.

Ví dụ: nếu ta chắn INT (ngắt bàn phím) thì bàn phím sẽ vô tác dụng khi ta gõ phím. Để các ngắt này hoạt động trở lại bình thường ta dùng lệnh STI. Cờ IF chỉ ảnh hưởng đối với ngắt phần cứng.

b. Ngắt không chắn được NMI (Non Maskable Interrupt)

Đây là tín hiệu ngắt không che được bằng phần mềm, nó có độ ưu tiên cao nhất. Nó được phát sinh để báo một sự nguy hiểm như sụt thế hay sự hư hỏng bộ nhớ.

c. Ngắt nội bộ (Interrupt internal)

Được phát sinh bởi CPU khi thực hiện chương trình gồm:

- Divide by zero interrupt: Được tự động phát sinh bởi CPU mỗi khi phép chia có sự sai như chia cho zero.
- Into (Overflow interrupt): Được tự động phát sinh bởi CPU mỗi khi một sự tràn dữ liệu xảy ra trong các phép toán.
- Trap interrupt: Được tự động phát sinh bởi CPU mỗi khi thực hiện xong 1 lệnh trong trường hợp cờ TF = 1.

d. Ngắt mềm (soft interrupt)

Do chương trình yêu cầu để thực hiện một chương trình con nào đó trong ROM hay RAM. Một ngắt mềm là một ngắt được gọi bởi lệnh INT.

Ví dụ: lệnh INT 5 sẽ thực hiện chương trình in toàn bộ nội dung của màn hình máy tính ra máy in.

7.3.3. Bảng vector ngắt

Là một bảng có chiều dài 1KB nằm trong bộ nhớ trong bắt đầu ở địa chỉ 0000:0000 đến 0000:03FF. Bảng này gồm 256 phần tử, mỗi phần tử gồm 4 byte chứa segment và

offset, phần tử thứ i chứa địa chỉ logic của chương trình xử lý ngắt thứ i. Khi một ngắt được tiếp nhận, CPU căn cứ vào số hiệu ngắt để tìm địa chỉ của chương trình con xử lý ngắt tương ứng trong bảng vector ngắt.

- Ngắt từ 00h đến 1 Fh: Ngắt dùng trong ROM.
- Ngắt từ 20h đến 2 Fh: Ngắt mềm dùng trong DOS.
- Ngắt từ 30h đến FFh: Ngắt mềm do người dùng định nghĩa.

7.4. CÁC KIỂU ĐỊNH VỊ

Các lệnh của CPU có thể chia làm ba nhóm:

- Nhóm không có toán hạng
- Nhóm có một toán hạng
- Nhóm có hai toán hạng. Trong trường hợp này toán hạng thứ nhất gọi là toán hạng nguồn, toán hạng 2 gọi là toán hạng đích.

Các toán hạng của một câu lệnh có thể được quy định dưới nhiều dạng khác nhau, được gọi là phép định địa chỉ. Phép định địa chỉ báo cho CPU biết cách tính giá trị thật của toán hạng trong câu lệnh. Có ba phép định địa chỉ:

- Phép định địa chỉ thanh ghi.
- Phép định địa chỉ hằng.
- Phép định địa chỉ bộ nhớ.

7.4.1. Phép định địa chỉ thanh ghi

Trong phép định địa chỉ này, thanh ghi được chỉ rõ trong câu lệnh. Nội dung của thanh ghi này sẽ là giá trị thật của toán hạng trong câu lệnh.

Các thanh ghi được dùng trong phép định địa chỉ thanh ghi là:

AH, AL, BH, BL, CH, CL, DH, DL

AX, BX, CX, DX

SP, BP, SI, DI

CS, DS, ES, SS

Ví dụ: `ADD AX, BX`

7.4.2. Phép định địa chỉ hằng

Trong các phép định vị địa chỉ hằng, một giá trị 8 bit hay 16 bit được chỉ rõ trong câu lệnh. Giá trị này sẽ là giá trị thật của toán hạng trong câu lệnh. Toán hạng hằng chỉ có thể là toán hạng nguồn.

Ví dụ: `MOV AL, 47`

7.4.3. Phép định địa chỉ bộ nhớ

Các lệnh của CPU có thể làm việc trực tiếp với các dữ liệu trong bộ nhớ. Khi một toán hạng bộ nhớ được sử dụng, CPU phải tính địa chỉ vùng nhớ chứa dữ liệu. Địa chỉ này gọi là địa chỉ EA (Effective Address). Dựa vào địa chỉ EA (segment và offset) CPU

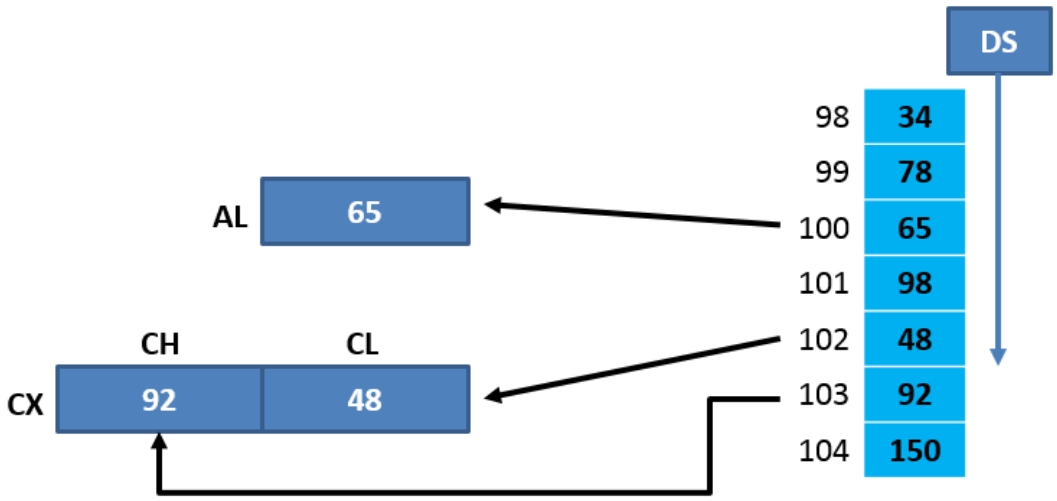
sẽ tính toán và chuyển sang địa chỉ vật lý tương ứng để tiến hành truy xuất vùng nhớ. Nội dung của vùng nhớ là giá trị thật của toán hạng trong câu lệnh.

Chú ý: Trong một câu lệnh không thể cả hai toán hạng nguồn và đích đều là toán hạng bộ nhớ, tuy nhiên thực hiện lệnh này một cách gián tiếp bằng cách chuyển một toán hạng bộ nhớ nguồn vào một thanh ghi rồi chuyển thanh ghi vào toán hạng bộ nhớ đích.

a. Phép định vị bộ nhớ trực tiếp

Trong phép định địa chỉ này, địa chỉ của toán hạng bộ nhớ nằm trực tiếp trong bản thân câu lệnh. Địa chỉ này được ghi giữa hai dấu móc vuông để phân biệt với các dữ liệu hằng.

Ví dụ:
```
MOV AL, DS:[100]
MOV CX, DS:[102]
```



Chú ý: Nếu trong toán hạng không ghi rõ thanh ghi đoạn thì thanh ghi đoạn DS sẽ được dùng.

b. Phép định địa chỉ bộ nhớ gián tiếp

Trong phép định địa chỉ này, địa chỉ EA của toán hạng bộ nhớ không nằm trực tiếp trong câu lệnh mà được xác định gián tiếp thông qua các thanh ghi cơ sở (BP hay BX) và chỉ mục (SI hay DI). Địa chỉ EA của toán hạng bộ nhớ có thể xác định bởi:

- Nội dung của một thanh ghi chỉ mục hay cơ sở (SI, DI, BX, BP) có thể có hoặc không có thêm một đại lượng 8 bit hay 16 bit gọi là độ dời.
- Nội dung của tổng một thanh ghi chỉ mục và một thanh ghi cơ sở có thể có hoặc không có thêm một độ dời 8 bit hay 16 bit.

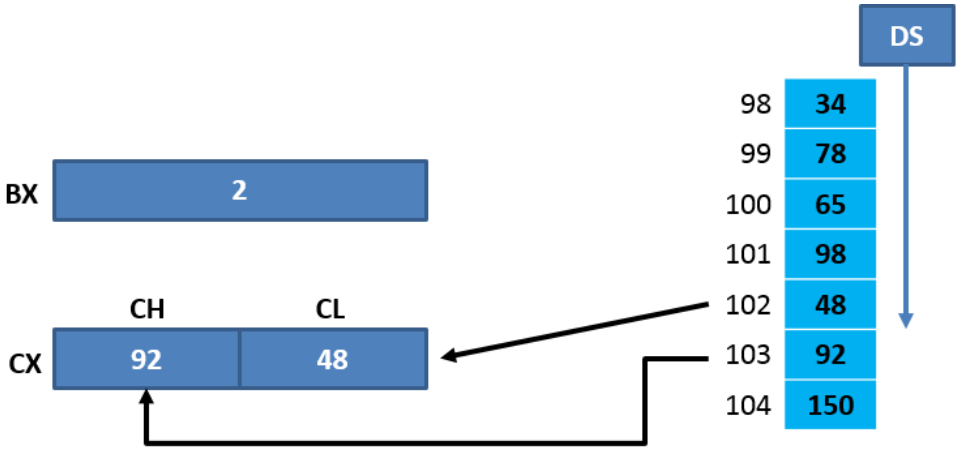
Phép định địa chỉ cơ sở

Phép định vị địa chỉ này xác định địa chỉ EA của toán hạng trong bộ nhớ thông qua nội dung của các thanh ghi cơ sở BX hay BP có thêm hoặc không có sự tham gia của một độ dời 8 bit hay 16 bit.

Phép định địa chỉ cơ sở với thanh ghi BX thường dùng trong việc truy xuất các dữ liệu có dạng cấu trúc kiểu mảng một chiều. Phép định địa chỉ cơ sở với thanh ghi BP thường dùng trong để truy xuất dữ liệu trong STACK.

Ví dụ: `MOV CX, DS:[BX+100]`

Lệnh này lấy nội dung của ô nhớ có địa chỉ offset là nội dung thanh ghi BX + 100 trong đoạn được chỉ bởi DS vào thanh ghi CX.



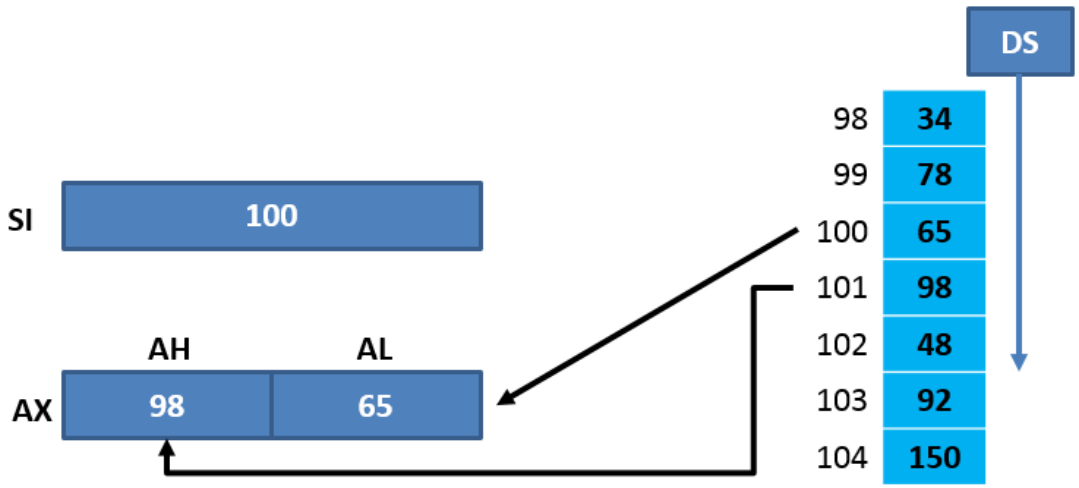
Phép định vị địa chỉ chỉ mục

Phép định vị địa chỉ này xác định địa chỉ EA của toán hạng bộ nhớ thông qua nội dung của các thanh ghi chỉ mục SI hay DI có thêm hoặc không có sự tham gia của một độ dời 8 bit hay 16 bit.

Phép định vị địa chỉ này thường dùng trong việc truy xuất dữ liệu kiểu mảng (SI hay DI xem như chỉ số của mảng) và trong việc xử lý chuỗi.

Ví dụ: `MOV AX, DS:[SI]`

Lấy 16 bit trong bộ nhớ bắt đầu tại địa chỉ offset là SI, địa chỉ segment là DS lưu vào thanh ghi AX.

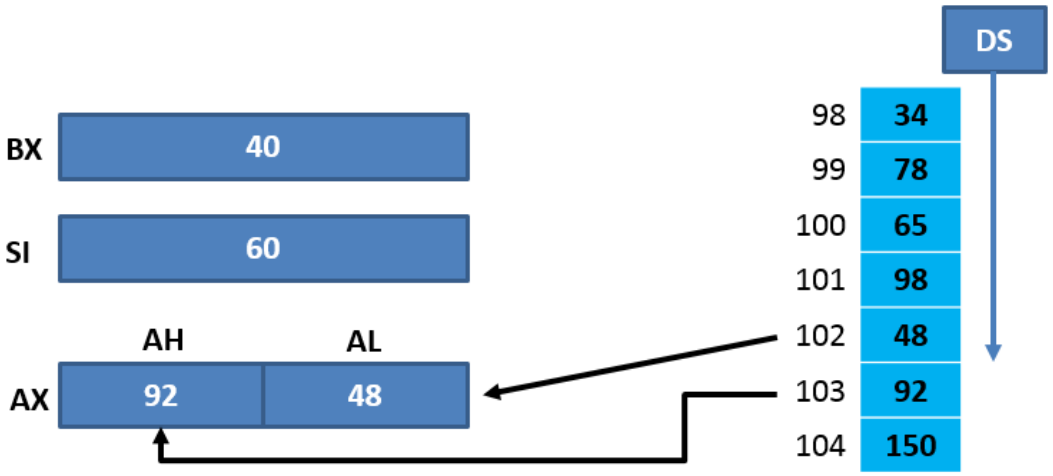


Phép định vị địa chỉ cơ sở chỉ mục

Phép định vị địa chỉ này truy xuất toán hạng bộ nhớ mà địa chỉ EA được tính bằng tổng nội dung của một thanh ghi cơ sở và một thanh ghi chỉ mục có thêm hoặc không có sự tham gia của một độ dời 8 bit hay 16 bit.

Phép định vị này thường dùng cho các cấu trúc dữ liệu phức tạp như mảng nhiều chiều hay mảng mà mỗi thành phần của nó có cấu trúc.

Ví dụ: `MOV AX, DS:[BX + SI + 2]`



Chú ý:

- Trong một câu lệnh chỉ có tối đa một toán hạng bộ nhớ

Ví dụ: `MOV DS:[100], ES:[BX]` → sai

`MOV var1, var2` → sai

- Nếu không chỉ rõ thanh ghi đoạn khi định vị bộ nhớ thì thanh ghi đoạn được sử dụng là SS nếu có thanh ghi BP tham gia định vị, ngược lại là thanh ghi DS.

Ví dụ: `MOV AX, [100]` → DS

`MOV CX, [BP + SI]` → SS

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Vẽ sơ đồ tổ chức bộ nhớ của CPU Intel 8086/8088
2. Giải thích lý do vì sao CPU Intel 8086/8088 phải tổ chức theo kiểu phân đoạn?
3. Mô tả quá trình bộ xử lý tiếp nhận và xử lý tín hiệu ngắt quãng

CHƯƠNG 8

LẬP TRÌNH TURBO ASSEMBLER

8.1. CÁC THÀNH PHẦN CƠ BẢN CỦA HỢP NGỮ

8.1.1. Bộ ký tự của hợp ngữ, từ khoá, tên

Bộ ký tự: Hợp ngữ được xây dựng dựa vào các ký tự sau:

- Bộ chữ cái la tinh: A→ X, a → z
- Bộ chữ số thập phân: 0 → 9
- Các ký tự đặc biệt: ? @ _ $: . [] () <> {} + - * / & % ! ‘ “ ~ \ = # ^ ; ,
- Các ký tự ngăn cách gồm khoảng trắng và tab

Từ vựng:

Từ khoá (key word): Là các từ dành riêng của hợp ngữ như tên các thanh ghi, tên gợi nhớ của các lệnh hợp ngữ, tên các toán tử…

Các từ khoá có thể viết bằng chữ hoa hoặc chữ thường.

Tên (symbol, name): Là một dãy các ký tự dùng để biểu thị tên hằng, tên biến, tên nhãn, tên chương trình con, tên đoạn…

Tên do người lập trình đặt. Một tên có thể chứa:

- Các ký tự chữ: A → Z, a → z
- Các chữ số: 0 → 9
- Các ký tự đặc biệt: . ? @ _ $

Ký tự đầu tiên của tên không thể là ký tự số mà phải là ký tự chữ hoặc các ký tự đặc biệt để Assembler có thể phân biệt sự khác nhau giữa tên và số. Dấu chấm không thể dùng trong một tên chỉ có thể dùng là ký tự đầu của nhãn.

8.1.2. Cấu trúc một lệnh hợp ngữ

Một lệnh hợp ngữ gồm bốn phần:

```
[Label] [Mnemonic] [Operand] [;Comment]
```

Mỗi dòng chỉ chứa một lệnh hợp ngữ và mỗi lệnh phải nằm trên một dòng. Mỗi phần phải cách nhau bằng một hay nhiều ký tự ngăn cách. Mỗi dòng tối đa 128 ký tự.

a. Nhãn (Label): Nhãn chỉ là một tên đại diện cho các số, chuỗi ký tự hoặc các vị trí trong bộ nhớ.

Ví dụ: Tính tổng 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8

```
DOSEG
.MODEL SMALL
.STACK 100h
.DATA
```

```
  Sum DW ?
.CODE
Begin:
  MOV AX, @DATA
  MOV DS, AX
  CALL Sumproc
  MOV AX, 4C00h
  INT 21h
SumProc Proc
  MOV Sum, 0
  MOV AX, 1
  MOV CX, 8
SumLoop:
  ADD Sum, AX
  INC AX
  LOOP SumLoop
  RET
SumProc Endp
END Begin
```

Nhãn Sum tương đương với biến địa chỉ 16 bit

Nhãn SumProc là tên chương trình con chứa mã các phần khác của chương trình có thể gọi tới.

Nhãn SumLoop tương đương với địa chỉ của lệnh ADD Sum, AX để lệnh LOOP ở cuối có thể quay trở lại vị trí lệnh thích hợp.

Mỗi nhãn chỉ định nghĩa một lần trừ một số trường hợp đặc biệt sẽ nói sau. Nhãn có thể dùng làm toán tử. Nhãn có thể xuất hiện một mình trên dòng lệnh trong trường hợp này trị của nhãn là địa chỉ của lệnh ở dòng kế tiếp.

Ví dụ:

```
    .
    .
    Jmp Addition
    .
    .
```

```
Addition :
ADD BX, AX
.
.
```

Hoặc có thể viết:

```
.
.
JMP Addition
.
.
Addition : ADD BX, AX
.
.
```

Không được đặt tên nhãn trùng với các ký hiệu đã quy ước như tên các thanh ghi, các toán tử dùng trong biểu thức (PTR, BYTE, WORLD,…). Cũng không được dùng các chỉ dẫn IFxxx hoặc .ERRxxx để làm tên nhãn.

Chú ý: Sau nhãn có thể có dấu hai chấm hoặc không tuỳ theo cách sử dụng như ta thấy qua các ví dụ trên.

b. Tên gợi nhớ (Mnemonic): Xác định hành động của câu lệnh. Nó có thể là một chỉ thị (instruction) hay một chỉ dẫn (directive).

Chỉ thị hợp ngữ gần giống như các lệnh gợi nhớ của CPU, nó xác định các hành động mà CPU sẽ thực hiện. Khi Assembler dịch, mỗi chỉ thị sẽ được dịch sang một lệnh mã máy tương ứng. Ví dụ như MOV, ADD, MUL, …

Chỉ dẫn là lệnh của Assembler, không phải là lệnh của CPU, các lệnh này tuy xuất hiện trong chương trình nhưng không dịch sang mã máy tương ứng. Các chỉ dẫn dùng để điều khiển cách dịch của Assembler đối với các lệnh khi dịch một chương trình nguồn sang mã máy. Ví dụ như chỉ dẫn END dùng đánh dấu nơi kết thúc chương trình. Khi đọc đến END trình hợp dịch sẽ kết thúc việc dịch chương trình tức những lệnh kế tiếp nếu có sẽ bị bỏ qua. Nếu không có END sẽ bị lỗi sai. Sau END ta có thể quy định nơi bắt đầu thực hiện chương trình. Trường hợp chương trình gồm nhiều đơn thể (nhiều tập chương trình nguồn trong một đơn thể, các đơn thể khác chỉ ghi END, không ghi nhãn ở sau), nếu ghi nhiều thì TLINK chỉ lấy một.

c. Toán hạng (operand): Xác định các dữ liệu mà lệnh (chỉ dẫn hay chỉ thị) cần xử lý. Có thể có một, hai hoặc nhiều hơn hoặc không có toán hạng nào sau lệnh. Số lượng toán hạng có thể là thanh ghi, hằng, nhãn, biến nhớ, chuỗi ký tự.

- Thanh ghi (register): Các toán hạng thường dùng nhất trong chương trình. Một số lệnh chỉ dùng toán tử thanh ghi.

Ví dụ: `MOV DI, AX`

- Hằng (constant): Đôi khi dùng trực tiếp hằng trong toán hạng.

Ví dụ:

```
SUB AX, 4
SUB BX, 'A'
```

- Nhãn (Label): Nhãn có thể dùng làm toán hạng

Ví dụ:

```
.
.
Num DW ?
.
.
MOV Num, AX
.
.
```

- Biểu thức (Expression): Trình hợp dịch cho phép dùng biểu thức với các dấu ngoặc, các toán tử số học, logic, quan hệ,…

d. Ghi chú (Comment): Sau mỗi lệnh ta có thể viết ghi chú sau dấu chấm phẩy.

8.1.3. Chỉ dẫn khai báo dữ liệu

Chỉ dẫn này cho phép khai báo vùng nhớ dành cho các dữ liệu cần sử dụng trong chương trình. Các dữ liệu có thể là số, chuỗi hay một biểu thức có giá trị xác định.

```
[name] DB expression [,expression] …
[name] DW [type PTR] expression[,expression] …
[name] DD expression [,expression] …
[name] DF [type PTR] expression[,expression] …
[name] DQ expression [,expression] …
[name] DT expression [,expression] …
```

Trong đó:

name : đánh dấu địa chỉ offset bắt đầu của dữ liệu trong bộ nhớ

DB : khai báo dữ liệu dạng Byte (8 bit)

DW : khai báo dữ liệu dạng Word (16 bit)

DD : khai báo dữ liệu dạng Double Word (32 bit)

DF : khai báo dữ liệu dạng 48 bit

DQ : khai báo dữ liệu dạng 64 bit

DT : khai báo dữ liệu dạng 80 bit

expression, ... : giá trị ban đầu của các phần tử trong vùng nhớ

Ghi chú:

- Dữ liệu trong đoạn nếu khai báo liên tiếp nhau sẽ cấp phát liên tục trong bộ nhớ.
- Trong phạm vi giáo trình chỉ đề cập đến dữ liệu dạng DB và DW

Ví dụ: Giả sử các dữ liệu sau được khai báo liên tục trong bộ nhớ bắt đầu từ địa chỉ offset 0400h

```
B1  db 10
W1 dw 4906
B2  db 32, 60
W2 dw 1A5Fh
S1  db "a"
S2  db "A", 8, "BC"
    db 100, 200
W3 dw 10, 20, 256
```

→ Giá trị của vùng nhớ từ offset 0400h như sau:

0400	10	B1	040E	10	W3
0401	2Ah	W1	040F	0	
0402	13h		0410	20	
0403	32	B2	0411	0	
0404	60		0412	0	
0405	5Fh	W2	0413	1	
0406	1Ah		0414		
0407	97	S1	0415		
0408	65	S2	0416		
0409	8		0417		
040A	66		0418		
040B	67		0419		
040C	100		041A		
040D	200		041B		

Expression có thể sử dụng ký tự "?" để khai báo giá trị bất kỳ (không quan tâm giá trị ban đầu của phần tử trong bộ nhớ).

Khi muốn khai báo nhiều phần tử có giá trị giống nhau, ta sử dụng toán tử DUP theo cú pháp:

```
<count> DUP(<exp>)
```

→ Khai báo **count** lần giá trị **exp**

Ví dụ:

```
S1 db 5 DUP(10) → 10, 10, 10, 10, 10
S2 db 3 DUP(2 DUP ('a')) → 'a', 'a', 'a', 'a', 'a', 'a'
B1 db ? → B1 nhận giá trị bất kỳ có sẵn trong bộ nhớ
```

Khi truy xuất giá trị trong bộ nhớ trình dịch sẽ căn cứ vào kiểu của vùng nhớ mà xác định kiểu dữ liệu để truy xuất.

Ví dụ: Một số kết quả khi truy xuất vùng nhớ đã khai báo ở ví dụ trên:

```
[0401h] = 2Ah      (8 bit)
[B1] = 10          (8 bit)
[W1] = 132Ah       (16 bit)
[B1 + 2] = 13h     (8 bit)
[W1 + 1] = 3C20h   (16 bit)
[S2 + 6] = 10      (8 bit)
```

8.2. CẤU TRÚC CHƯƠNG TRÌNH HỢP NGỮ

DOS thi hành được hai loại tập tin: dạng .EXE và .COM. Tập tin dạng .EXE thường để xây dựng các chương trình lớn, còn tập tin dạng.com tạo ra các chương trình nhỏ hơn. TASM cho phép tạo cả hai loại tập tin .EXE và .COM, cách viết một chương trình hợp ngữ đối với từng loại .EXE hay .COM là khác nhau.

8.2.1. Tập tin dạng .COM

a. Đặc điểm của tập tin .COM

Chỉ có duy nhất một đoạn. Chương trình, dữ liệu và stack đều dùng chung một đoạn.

Kích thước tối đa của tập tin là 64 KB trừ đi độ dài của PSP là 256 bytes và trừ đi 2 (do PUSH 0 vào stack).

Tập tin dạng .COM nạp vào bộ nhớ và được thực hiện nhanh hơn tập tin dạng.EXE, nhưng nó chỉ áp dụng cho các chương trình nhỏ, chỉ có thể gọi các chương trình con dạng gần, muốn xây dựng các chương trình lớn ta phải viết dưới dạng .EXE.

Tập tin dạng .COM có thể liên kết từ các phần tách rời nhau, mỗi phần tách riêng trên những tập tin khác nhau. Tất cả những phần này phải có cùng tên đoạn, cùng tên với phần chính, phần chính phải chứa điểm bắt đầu của chương trình và phải liên kết trước.

b. Cách thực hiện một tập tin dạng .COM

- Khi DOS thực hiện tập tin .COM, DOS định vị bộ nhớ và tạo một vùng nhớ chiều dài 256 byte ở offset 0000h trong đoạn chương trình, vùng này gọi là PSP (Program Segment Prefix). Nó chứa các thông tin cần thiết cho DOS. Sau đó tập tin .COM sẽ được

nạp ngay sau PSP ở offset 100h và PUSH trị 0 vào stack trước khi DOS chuyển quyền điều khiển cho chương trình .COM.

Tất cả các thanh ghi đoạn CS, DS, ES, SS đều trỏ đến PSP, thanh ghi con trỏ đến lệnh IP được gán trị 100h (địa chỉ offset bắt đầu của chương trình), thanh ghi SP được gán trị FFFEh nếu bộ nhớ cho phép ngược lại nó gán trị lớn nhất của bộ nhớ còn trống trừ đi 2 (do PUSH 0 vào stack).

c. Cấu trúc của chương trình dạng .COM

Cấu trúc đơn giản

```
.MODEL TINY
.CODE
ORG 100h
Begin:
   ; Các lệnh
   .
   .
   ; Kết thúc
   INT 20h
   ; Các lệnh (chương trình con)
   .
   .
   ; Khai báo dữ liệu
   .
   .
END Begin
```

Hoặc:

```
.MODEL TINY
.CODE
ORG 100h
Begin:
   JMP Start
   ; Khai báo dữ liệu
   .
   .
```

```
      Start:
      ; Đoạn mã
      .
      .
      ; Kết thúc
      INT 20h
      ; Các lệnh (chương trình con)
      .
      .
   END Begin
```

Ví dụ:

```
   .MODEL TINY
   .CODE
   ORG 100h
   Begin:
      MOV AH, 09h
      MOV DX, OFFSET Msg
      INT 21h
      INT 20h
      Msg DB 'Hello$'
   END Begin
```

Cấu trúc đoạn:

```
   <tên đoạn> SEGMENT
      ASSUME CS:<tên đoạn>, DS:<tên đoạn>
      ORG 100h
   Begin:
      ; Đoạn mã
      .
      .
      ; Kết thúc
      INT 20h
      ; Các lệnh (chương trình con)
```

```
        .
        .
        ; Khai báo dữ liệu
        .
        .
    <tên đoạn> ENDS
    END Begin
```

Ví dụ:

```
    CSeg SEGMENT
        ASSUME CS:CSeg, DS:CSeg
        ORG 100h
    Begin:
        MOV AH, 09h
        MOV DX, OFFSET Msg
        INT 21h
        INT 20h
        Msg DB 'Hello$'
    CSeg ENDS
    END Begin
```

8.2.2. Tập tin dạng .EXE

a. Đặc điểm của tập tin .EXE

Chương trình có thể được khai báo nhiều đoạn khác nhau. Mỗi chương trình có thể có nhiều đoạn chương trình, nhiều đoạn dữ liệu.

Kích thước của tập tin có thể lớn tuỳ ý tuỳ thuộc vào kích thước bộ nhớ máy tính. Thường dạng .EXE dùng để xây dựng các chương trình lớn có kích thước lớn hơn 64 KB.

Tập tin có một header ở đầu tập tin. Header này chứa các thông tin điều khiển về tập tin để DOS có thể nạp nó vào bộ nhớ và thực hiện. Kích thước header phụ thuộc vào số các lệnh trong chương trình cần để định vị lại các địa chỉ đoạn khi chương trình được nạp, nhưng nó luôn là bội số của 256 bytes.

b. Cách thực hiện tập tin dạng .EXE

Khi DOS thực hiện tập tin .EXE, DOS tạo vùng PSP gồm 256 bytes giống như nó tạo trong tập tin dạng .COM. Kế đó DOS kiểm tra vùng header để xác định vị trí cuối vùng header ở đâu, phần chương trình và dữ liệu nằm ngay sau vùng header sẽ được DOS nạp vào bộ nhớ ngay sau PSP.

Dựa vào thông tin header DOS sẽ định vị lại một số lệnh có liên quan đến địa chỉ đoạn trong chương trình.

Các thanh ghi đoạn DS và ES chỉ đên PSP. Thanh ghi CS và thanh ghi con trỏ lệnh IP được xác định từ các thông tin trong vùng header để chỉ đến điểm bắt đầu của chương trình. Các thanh ghi SS và SP cũng được xác định từ các thông tin trong header để chỉ đến stack.

Tập tin dạng .EXE có thể liên kết từ các phần tách rời nhau, mỗi phần tách riêng trên những tập tin khác nhau. Mỗi phần có thể dùng một tên đoạn riêng, các chương trình con có thể khai báo dưới dạng gần hoặc xa. Tuy nhiên các phần được liên kết với nhau phải chứa duy nhất một đoạn với kiểu Stack. Chương trình chỉ có duy nhất một điểm bắt đầu được xác định bởi lệnh END trong phần chính.

c. Cấu trúc của chương trình dạng .EXE

Cấu trúc đơn giản:

```
.MODEL SMALL
.STACK 200H
.DATA
; Khai báo dữ liệu
.
.
.CODE
Begin:
   MOV AX, @DATA
   MOV DS, AX
   ; Các lệnh
   .
   .
   ; Kết thúc
   MOV AX, 4C00h
   INT 21h
   ; Các lệnh (chương trình con)
   .
   .
END Begin
```

Ví dụ:

```
.MODEL SMALL
```

```
.STACK 200H
.DATA
Msg DB 'Hello$'
.CODE
Begin:
   MOV AX, @DATA
   MOV DS, AX
   MOV AH, 09h
   MOV DX, OFFSET Msg
   INT 21h
   MOV AX, 4C00h
   INT 21h
END Begin
```

Cấu trúc đoạn:

```
<tên đoạn data> SEGMENT
   ; Khai báo dữ liệu
   .
   .
<tên đoạn data> SEGMENT
<tên đoạn code> SEGMENT
   ASSUME CS:<tên đoạn code>, DS:<tên đoạn data>
Begin:
   MOV AX, <tên đoạn data>
   MOV DS, AX
   ; Đoạn mã
   .
   .
   ; Kết thúc
   MOV AX, 4C00h
   INT 21h
   ; Các lệnh (chương trình con)
   .
```

```
    .
<tên đoạn code> ENDS
END Begin
```

Ví dụ:

```
DSeg SEGMENT
   Msg DB 'Hello$'
DSeg SEGMENT
CSeg SEGMENT
   ASSUME CS:CSeg, DS:DSeg
Begin:
   MOV AX, DSeg
   MOV DS, AX
   MOV AX, 09h
   MOV DX, OFFSET Msg
   INT 21h
   MOV AX, 4C00h
   INT 21h
CSeg ENDS
END Begin
```

8.3. CÁC LỆNH CƠ BẢN CỦA BỘ XỬ LÝ INTEL 8086/8088

Các quy ước khi viết cú pháp một lệnh hợp ngữ

- Chữ đậm là tên một lệnh hợp ngữ.
- Chữ nghiêng là tên toán hạng của một lệnh hợp ngữ.
- []: Những tham số đặt trong hai dấu [] là tuỳ chọn.
- Reg: Chỉ toán hạng thanh ghi (8 bit hay 16 bit).
- Reg8: Chỉ toán hạng thanh ghi 8 bit.
- Reg16: Chỉ toán hạng thanh ghi 16 bit.
- Mem: Chỉ toán hạng bộ nhớ (8 bit hay 16 bit).
- Mem8: Chỉ toán hạng bộ nhớ 8 bit.
- Mem16: Chỉ toán hạng bộ nhớ 16 bit.
- Mem32: Chỉ toán hạng bộ nhớ 32 bit.
- Immed: Chỉ toán hạng hằng (8 bit hay 16 bit).

- Immed8: Chỉ toán hạng hằng 8 bit.
- Immed16: Chỉ toán hạng hằng 16 bit.
- SegReg: Chỉ toán hạng thanh ghi đoạn.

8.3.1. Nhóm lệnh chuyển dữ liệu

1. Lệnh MOV (move)

```
MOV dest, soure
```

Toán hạng nguồn soure có thể là Reg, Mem hay Immed.

Toán hạng đích dest chỉ có thể là Reg hay Mem.

Chức năng: Chuyển nội dung toán hạng source vào toán hạng dest.

Chiều dài dữ liệu có thể là 8 bit hay 16 bit.

Trường hợp 1: Chuyển dữ liệu giữa hai thanh ghi.

- Reg8 ← Reg8

Ví dụ:

```
MOV AL, AH
```

- Reg16 ← Reg16

Ví dụ:

```
MOV AX, BX
```

Trường hợp 2: Chuyển dữ liệu giữa thanh ghi và bộ nhớ.

- Mem8 ← Reg8

Ví dụ:

```
MOV [BX], AL
MOV DS:[150h], AL
MOV ES:[SI], AL
```

- Reg8 ← Mem8

Ví dụ:

```
MOV AL, [BX]
MOV AH, DS:[10h]
```

- Mem16 ← Reg16

Ví dụ:

```
MOV [BX], AX
MOV Var, AX  ; Var là một biến dạng word
MOV CS:[SI+BX], DX
```

- Reg16 ← Mem16

Ví dụ:

```
MOV CX, Count
```

Trường hợp 3: Chuyển dữ liệu giữa hằng và thanh ghi hay bộ nhớ.

- Reg8 ← Immed8

Ví dụ:

```
MOV AL, 0B5h
```

- Mem8 ← Immed8

Ví dụ:

```
MOV Byte Ptr DS:[150h], 10h
```

- Reg 16 ← Immed16

Ví dụ:

```
MOV AX, 0B800h
```

- Mem16 ← Immed16

Ví dụ:

```
MOV Word PTR [BX], 5AB7h
```

Trường hợp 4: Chuyển dữ liệu giữa thanh ghi đoạn và thanh ghi hay bộ nhớ.

- SegReg ← Reg16

Ví dụ:

```
MOV DS, AX
```

- SegReg ← Mem16

Ví dụ:

```
MOV ES, Screen
```

- Reg16 ← SegReg

Ví dụ:

```
MOV AX, CS
```

- Mem16 ← SegReg

Ví dụ:

```
MOV [BX+DI], ES
```

Chú ý:

- Lệnh MOV không ảnh hưởng thanh ghi cờ hiệu.
- Không thể chuyển dữ liệu trực tiếp giữa hai toán hạng bộ nhớ với nhau, muốn chuyển ta phải dùng một thanh ghi trung gian.

Ví dụ: Muốn chuyển dữ liệu 16 bit từ Var1 vào Var2

```
MOV AX, Var1
```

```
MOV Var2, AX
```

- Không thể chuyển trực tiếp một hằng vào một thanh ghi đoạn, muốn chuyển ta phải dùng một thanh ghi trung gian.

Ví dụ: Muốn chuyển giá trị B800h vào thanh ghi DS

```
MOV AX, 0B800h
MOV DS, AX
```

- Không thể chuyển dữ liệu trực tiếp giữa hai thanh ghi đoạn.

2. Lệnh XCHG (exchange)

```
XCHG dest, source
```

Toán hạng source và dest chỉ có thể là Reg và Mem

Chức năng: Hoán chuyển nội dung của hai toán hạng nguồn và đích.

Ta chỉ có thể hoán chuyển giữa thanh ghi hoặc giữa thanh ghi và bộ nhớ.

Ví dụ:

```
XCHG AX, BX ;Hoán chuyển giữa hai thanh ghi 16 bit
XCHG AX, [BX] ;Hoán chuyển giữa thanh ghi và bộ nhớ
```

Chú ý:

- Lệnh này không ảnh hưởng tới thanh ghi cờ hiệu.
- Không thể dùng lệnh này với các thanh ghi đoạn.

3. Lệnh PUSH

```
PUSH source
```

Toán hạng source chỉ có thể là Reg16 và Mem16.

Chức năng: Dùng để cất một hằng hay nội dung thanh ghi 16 bit hay nội dung một toán hạng bộ nhớ 16 bit v ào STACK. Lệnh PUSH giảm thanh ghi SP 2 đơn vị và chuyển nội dung 16 bit của toán hạng nguồn vào trên đỉnh của STACK. Đỉnh STACK được xác định bởi cặp thanh ghi SS: SP

Ví dụ:

```
PUSH SI
PUSH DS
PUSH [BX + SI]
PUSH 147Eh ;Dùng với CPU 286/386/486
```

4. Lệnh POP

```
POP dest
```

Toán hạng dest chỉ có thể là Reg16 và Mem16.

Chức năng: Ngược với lệnh PUSH, POP dùng để lấy dữ liệu 16 bit đang hiện hành trên đỉnh của STACK (được xác định bởi cặp thanh ghi SS:SP) vào toán hạng dest. Lệnh POP tăng thanh ghi SP 2 đơn vị để chỉ đến đỉnh mới của STACK.

Ví dụ:

```
POP SI
POP [BX+SI]
```

5. Lệnh LEA (load effective address)

```
LEA Reg16, Mem16
```

Chức năng: Chuyển địa chỉ offset của một toán hạng bộ nhớ Mem16 vào thanh ghi Reg16.

Ví dụ:

```
LEA BX, DS:[500h] ; Chuyển 500h vào BX
LEA SI, Table ; Chuyển địa chỉ offset của Table vào thanh ghi SI
```

8.3.2. Nhóm lệnh cờ hiệu

1. Lệnh LAHF (load AH from flag)

```
LAHF
```

Chức năng: Chuyển phần thấp của thanh ghi cờ hiệu gồm các cờ SF, ZF, AF, PF và CF tương ứng vào các bit 7, 6, 4, 2 và 0 của thanh ghi AH, các bit còn lại 5, 3, 1 không đổi.

2. Lệnh SAHF (store AH into flag)

```
SAHF
```

Chức năng: Chuyển các bit 7, 6, 4, 2 và 0 của thanh ghi AH tương ứng vào các cờ SF, ZF, AF, PF và CF của thanh ghi cờ hiệu. Các cờ còn lại OF, DF, IF, TF không bị ảnh hưởng.

3. Lệnh PUSHF

```
PUSHF
```

Chức năng: PUSHF giảm thanh ghi SP 2 đơn vị và chuyển nội dung của thanh ghi cờ hiệu vào trên đỉnh STACK.

4. Lệnh POPF

```
POPF
```

Chức năng: POPF tăng thanh ghi SP 2 đơn vị và chuyển nội dung của phần tử trên đỉnh STACK vào thanh ghi cờ hiệu.

8.3.3. Nhóm lệnh dữ liệu qua cổng

CPU giao tiếp và điều khiển các thiết bị ngoại vi của máy tính thông qua các cổng vào ra (I/O port), mỗi cổng được xác định duy nhất bởi một số 16 bit gọi là địa chỉ cổng. CPU gửi dữ liệu hoặc thông tin điều khiển đến cổng bằng cách chỉ đến địa chỉ cổng đó,

còn cổng nhận thông tin và nếu cần, cổng trả lời bằng cách chuyển dữ liệu hoặc thông tin trạng thái trở lại CPU. Khi chuyển dữ liệu qua cổng ta chỉ có thể truyền trực tiếp một lượng dữ liệu 8 bit.

Tuỳ theo chức năng của từng cổng, các cổng có thể:

- Chỉ đọc được (input port).
- Chỉ ghi được (output port).
- Đọc và ghi được (input/output port).

Ví dụ:

- Bộ điều khiển màn hình đơn sắc dùng địa chỉ cổng từ 03B0h đến 03BFh.
- Bộ điều khiển màn hình màu CGA dùng địa chỉ cổng từ 03D0h đến 03DFh.
- Bộ điều khiển máy in song song thứ nhất dùng các địa chỉ cổng từ 0378h đến 037Fh, máy in song song thứ hai dùng các địa chỉ cổng từ 0278h đến 027Fh.

1. Lệnh IN

```
IN AL, port8
```

Hay

```
IN AL, DX
```

Chức năng: Đọc một lượng 8 bit từ một cổng nhập vào thanh ghi AL. Nếu địa chỉ cổng có giá trị trong khoảng từ 00h đến FFh thì địa chỉ đó có thể viết trực tiếp trong câu lệnh, còn trong trường hợp địa chỉ cổng có giá trị lớn hơn FFh thì phải dùng thanh ghi DX để định địa chỉ cổng.

Ví dụ:

```
IN AL, 61h ; đọc một byte từ cổng 61h
MOV DX, 03F8h
MOV Al, DX ; đọc một byte từ cổng 3F8h
```

2. Lệnh OUT

```
OUT port, AL
```

Hay

```
OUT DX, AL
```

Chức năng: Gởi một lượng 8 bit từ thanh ghi AL ra cổng xuất. Nếu địa chỉ cổng có giá trị trong khoảng từ 00h đến FFh thì địa chỉ đó có thể viết trực tiếp trong câu lệnh, còn trong trường hợp địa chỉ của cổng có giá trị lớn hơn FFh thì phải dùng thanh ghi DX để định địa chỉ cổng.

Ví dụ:

```
OUT 61h, AL ; Gởi một byte ra cổng 61h
MOV DX, 3F8h
OUT DX, AL ; Gởi một byte ra cổng 3F8h
```

8.3.4. Nhóm lệnh tính toán số học

1. Lệnh ADD (addition)

```
ADD dest, source
```

(dest) ← (dest) + (source)

Toán hạng source có thể là Reg, Mem, Immed.

Toán hạng dest có thể là Reg, Mem.

Chức năng: Thực hiện phép cộng không nhớ. Lệnh ADD cộng nội dung toán hạng source và toán hạng dest, kết quả lưu vào toán hạng dest.

Ví dụ:

```
ADD AL, AH
ADD AX, BX
ADD [BX], AX
ADD AL, 0B5h
ADD BYTE PTR DS:[150h], 10h
ADD AX, 0B800h
ADD WORD PTR [BX], 5AB7h
```

Chú ý:

- Không thể cộng trực tiếp thanh ghi đoạn.
- Lệnh ADD ảnh hưởng đến 6 cờ AF, CF, OF, PF, SF, ZF.

2. Lệnh INC (increment)

```
INC dest
```

Toán hạng đích dest chỉ có thể là Reg hay Mem.

(dest) ← (dest) + 1

Chức năng: Tăng nội dung toán hạng dest lên 1.

Ví dụ:

```
INC AL
INC CX
INC BYTE PTR [BX+SI]
```

3. Lệnh SUB (subtract)

```
SUB dest, source
```

(dest) ← (dest) – (source)

Toán hạng nguồn source có thể là Reg, Mem hay Immed.

Toán hạng đích dest có thể là Reg hay Mem.

Chức năng: thực hiện phép trừ không nhớ. Lệnh SUB lấy nội dung toán hạng dest trừ nội dung toán hạng source, kết quả lưu vào toán hạng dest.

Ví dụ:

```
SUB AL, AH
SUB BL, CL
SUB AX, CX
SUB [BX], AL
SUB DS:[150h], AL
SUB ES:[SI], DL
SUB AL, MEM[BX]
SUB [BX], AX
SUB Varw, AX ; Varw là một biến dạng word
SUB SC:[SI+BX], DX
SUB AX, MEM[SI]
SUB AL, 0B5h
SUB Mem[BX], 15h
SUB BYTE PTR DS:[150h], 10h
SUB AX, 0B800h
SUB WORD PTR[BX], 5AB7h
```

Chú ý:

- Không thể trừ trực tiếp thanh ghi đoạn.
- Lệnh SUB ảnh hưởng đến 6 cờ AF, CF, OF, PF, SF, ZF.

4. Lệnh DEC (decrement)

```
DEC dest
```

Toán hạng đích dest chỉ có thể là Reg hay Mem.

(dest) ← (dest) –1

Chức năng: Giảm nội dung toán hạng dest xuống 1.

Ví dụ:

```
DEC AL
DEC CX
DEC BYTE PTR [BX+SI]
DEC WORD PTR DS:[500h]
```

Chú ý: Lệnh này chỉ ảnh hưởng các cờ AF, OF, PF, SF, ZF nhưng không ảnh hưởng cờ CF.

5. Lệnh NEG (negative)

```
NEG dest
```

Toán hạng đích dest chỉ có thể là Reg hay Mem.

(dest) ← - (dest)

Chức năng: Đổi dấu toán hạng dest.

Ví dụ:

```
NEG AX

NEG CL

NEG BYTE WORD [BX+500h]

NEG WORDPTR [BP+SI]
```

Chú ý: Lệnh NEG ảnh hưởng tới tác cờ AF, CF, SF, PF, ZF, OF.

- Cờ CF bật 1 nếu nội dung của kết quả là một số khác không ngược lại CF = 0
- Cờ SF bật 1 nếu nội dung của kết quả là một số âm khác không.
- Cờ PF bật 1 nếu tổng các 8 bit thấp của kết quả là một số chẵn, ngược lại PF = 0
- Cờ ZF bật 1 nếu nội dung của kết quả là 0, ngược lại ZF = 0.
- Cờ OF bật 1 nếu nội dung toán hạng dest có trị 80h (nếu là dạng byte) hay 8000h (nếu là dạng word), ngược lại OF = 0.

Lệnh NEG có thể được ứng dụng để thực hiện phép trừ một hằng với một toán hạng (thanh ghi hay bộ nhớ).

Ví dụ: Muốn lấy 100 trừ đi nội dung của AL, ta không thể dùng lệnh.

```
SUB 100, AL
```

Mà phải viết lại như sau:

```
SUB AL,100

NEG AL
```

6. Lệnh MUL (Multiply)

```
MUL source
```

Toán hạng nguồn source có thể là Reg hay Mem.

Chức năng: Thực hiện phép nhân số không dấu. Có hai trường hợp

Trường hợp 1: source là toán hạng 8 bit thì lệnh MUL nhân nội dung của source với thanh ghi AL, kết quả 16 bit được lưu trong thanh ghi AX.

```
MUL source8
```

AL ← AL * source8

Trường hợp 2: source là toán hạng 16 bit thì lệnh MUL nhân nội dung của source với thanh ghi AX, kết quả 32 bit được lưu trong cặp thanh ghi DX:AX, phần thấp được chứa trong thanh ghi AX và phần cao được chứa trong thanh ghi DX.

```
MUL source16
```

DX:AX ← AX * source16

Chú ý: Lệnh này chỉ ảnh hưởng đến cờ CF và OF. Các cờ CF và OF bật lên 1 nếu phần cao của kết quả khác 0, ngược lại cờ CF và OF bằng 0 nếu phần cao của kết quả bằng 0 (đối với source8 bit thì phần cao là AH, đối với source16 bit thì phần cao là DX).

Ví dụ 1:

```
MUL BH ; AX ← AL * BH

MUL BX ; DX:AX ← AX * BX

MUL WORD PTR [BX+SI] ; DX:AX ← AX * [BX+SI]
```

Ví dụ 2: Nhân hai số 212 và 45

```
MOV AL, 212

MOV BL, 45

MUL BL

;kết quả 9540 lưu trong AX
```

Ví dụ 3: Nhân hai số 26216 và 230

```
MOV AX, 26216

MOV BX, 230

MUL BX

;kết quả 6028760 lưu trong cặp thanh ghi DX:AX
```

7. Lệnh DIV (divide)

```
DIV soure
```

Toán hạng nguồn source có thể là Reg hay Mem.

Chức năng: Thực hiện phép chia số không dấu, lấy nội dung của thanh ghi AX hay cặp thanh ghi DX:AX chia cho toán hạng source.

Trường hợp 1: Nếu source là toán hạng dạng byte thì lệnh DIV lấy nội dung của thanh ghi AX chia cho toán hạng source, thương số và số dư của phép chia tương ứng được lưu trong thanh ghi AL và thanh ghi AH.

```
DIV source8
```

AL ← AX DIV source8

AH ← AX MOD source8

Trường hợp 2: Nếu source là toán hạng dạng word thì lệnh DIV lấy nội dung của cặp thanh ghi DX:AX chia cho toán hạng source, thương số và số dư của phép chia tương ứng được lưu trong thanh ghi AX và trong thanh ghi DX.

```
DIV source16
```

AX ← DX:AX DIV source16

DX ← DX:AX MOD source16

Ví dụ 1:

```
DIV BX ; AX ← DX:AX DIV BX
; DX ← DX:AX MOD BX
```

Ví dụ 2: Chia số 230 cho 23

```
MOV AX, 230
MOV BL, 23
DIV BL ;thương số 10 lưu trong AL
       ;dư số 0 lưu trong AH
```

Ví dụ 3: Chia số 43678 cho 334

```
MOV AX, 43678
MOV DX, 0
MOV BX, 334
DIV BX ;thương số 13 lưu trong AX, dư số 258 lưu
       ;trong DX
```

Chú ý:

- Lệnh DIV không ảnh hưởng tới các cờ hiệu.
- Nếu phép chia cho 0 xảy ra hay thương số của phép vượt quá khả năng lưu trữ của thanh ghi AL hay AX, thì CPU sẽ tự phát sinh ngắt nội bộ là ngắt 0, ngắt này in thông báo lỗi "divide overflow" và chấm dứt chương trình trở về hệ thống. Các trường hợp xảy ra "divide overflow" đối với lệnh DIV:
  - Chia cho 0.
  - Thương số vượt trị 255 đối với phép chia số không dấu dạng byte.
  - Thương số vượt trị 65535 đối với phép chia số không dấu dạng word.

8.3.5. Nhóm các lệnh dịch chuyển và quay (shift and rotate)

1. Lệnh SHL (shift logical left)

```
SHL dest, 1
```

Hay

```
SHL dest, CL
```

Chức năng:

- Lệnh SHL dest, 1: Dịch chuyển nội dung của toán hạng dest sang phải một bit, bit cao của toán hạng dest được đẩy vào CF, bit thấp trở thành 0.

Ví dụ: Dịch chuyển nội dung của thanh ghi AL (có trị 10010110) sang trái 1 bit.

```
SHL AL, 1
```

→ Kết quả: AL = 00101100 CF = 1

- Lệnh SHL dest, CL: Tương đương với thực hiện lệnh SHL dest, 1 n lần với n là nội dung của CL.

Chú ý:

- Nếu toán hạng dest chứa một số nguyên không dấu thì lệnh SHL dest, 1 sẽ nhân nội dung của toán hạng dest với 2, tuy nhiên lệnh này thực hiện nhanh hơn lệnh MUL rất nhiều.
- Lệnh SHL ảnh hưởng tới các cờ CF, OF, PF, SF, ZF. Đối với OF, OF sẽ bật lên 1 nếu toán hạng đổi dấu sang lệnh SHL, ngược lại OF là 0.

2. Lệnh SHR (shift logical right)

```
SHR dest, 1
```

Hay

```
SHR dest, CL
```

Chức năng: giống như lệnh SHL nhưng thay vì dịch chuyển sang trái ta dịch chuyển sang phải.

- Lệnh SHR dest, 1: dịch chuyển nội dung của toán hạng dest sang phải 1 bit, bit thấp của toán hạng dest được đẩy vào CF, bit cao trở thành 0.

Ví dụ: Dịch chuyển nội dung của thanh ghi AL (giá trị 10110011) sang phải 1 bit

```
SHR AL, 1
```

→ Kết quả: AL = 01011001 CF = 1

- Lệnh SHR dest, CL: Tương đương với thực hiện lệnh SHR dest, 1 n lần với n là nội dung của thanh ghi AL.

Chú ý:

- Nếu toán hạng dest chứa một số nguyên không dấu thì lệnh SHR dest, 1 sẽ chia nội dung của toán hạng dest cho 2, tuy nhiên lệnh này thực hiện nhanh hơn lệnh DIV rất nhiều.
- Lệnh này ảnh hưởng tới các cờ CF, OF, PF, SF, ZF. Đối với cờ OF, OF sẽ bật nếu toán hạng đổi dấu sau lệnh SHR, ngược lại OF là 0.

3. Lệnh ROL (rotate left)

```
ROL dest, 1
```

Hay

```
ROL dest, CL
```

Chức năng:

- Lệnh ROL dest, 1: Quay nội dung của toán hạng dest sang trái một bit. Giống như lệnh SHL, nhưng thay vi bit thấp là 0 thì đối với lệnh ROL bit cao được dịch chuyển vào bit thấp.

Ví dụ: Quay các bit của thanh ghi AL (giá trị 10110011) sang trái 1 bit

```
ROL AL, 1
```

→ Kết quả: AL = 01100111 CF = 1

- Lệnh ROL dest, CL dùng cho trường hợp ROL nhiều hơn 1 bit.

Lệnh ROL chỉ ảnh hưởng đến các cờ CF và OF.

4. Lệnh ROR (rotate right)

```
ROR dest, 1
```

Hay

```
ROR dest, CL
```

Chức năng: Giống như lệnh ROL, thay vì quay bit sang trái thì quay bit phải.

- Lệnh ROR dest, 1: Quay nội dung của toán hạng dest sang phải 1 bit. Giống như lệnh SHR, nhưng thay vì bit cao bằng 0 thì đối với lệnh ROR bit thấp được dịch chuyển vào bit cao.

Ví dụ: Quay các bit của thanh ghi AL (giá trị 10110011) sang phải 1 bit

```
ROR AL, 1
```

→ Kết quả: AL = 11011001 CF = 1

- Lệnh ROR dest,CL: Dùng cho trường hợp ROR nhiều hơn 1 bit. Lệnh chỉ ảnh hưởng đến các cờ CF và OF.

8.3.6. Nhóm các lệnh logic

1. Lệnh AND

```
AND dest, source
```

(dest) ← (dest) AND (source)

Toán hạng source thể là Reg , Mem hay Immed.

Toán hạng dest có thể là Reg hay Mem

Chức năng: Lấy nội dung toán hạng source, thực hiện phép toán logic AND từng bit với nội dung của toán hạng dest. Lệnh này ảnh hưởng đến các cờ CF, OF, PF, SF, ZF, không ảnh hưởng cờ AF.

Lệnh này thường dùng để xóa 1 bit nào đó.

Ví dụ: Muốn xoá hai bit cuối của thanh ghi AL ta chỉ cần thực hiện.

```
AND AL, 00111111b
```

2. Lệnh OR

```
OR dest, source
```

(dest) ← (dest) OR (source)

Toán hạng source có thể là Reg, Mem hay Immed.

Toán hạng dest có thể là Reg hay Mem.

Chức năng: Lấy nội dung của toán hạng source, thực hiện phép toán logic OR từng bit với nội dung của toán hạng dest, kết quả lưu vào toán hạng dest. Lệnh này ảnh hưởng đến các cờ CF, OF, PF, SF, ZF, không ảnh hưởng cờ AF. Lệnh này ngược với lệnh AND, dùng để bật 1 bit nào đó lên.

Ví dụ: Muốn bật hai bit cuối của thanh ghi AL lên 1 ta chỉ cần thực hiện.

```
OR AL, 11000000b
```

3. Lệnh XOR

XOR dest, source

(dest) ← (dest) XOR (source)

Chức năng: Lấy nội dung của toán hạng source, thực hiện phép toán logic XOR từng bit với nội dung của toán hạng dest, kết quả lưu vào toán hạng dest. Lệnh này ảnh hưởng đến các cờ CF, OF, PF, SF, ZF không ảnh hưởng tời cờ AF. Lệnh này thường dùng để xác định những bit nào khác nhau giữa hai toán hạng, hoặc muốn đảo ngược một số bit nào đó.

4. Lệnh NOT

```
NOT dest
```

Toán hạng dest có thể là Reg hay Mem.

Chức năng: Đảo ngược từng bit của nội dung toán hạng dest, lệnh NOT hoàn toàn không ảnh hưởng tới các cờ.

Ví dụ:

```
MOV AL, 0
NOT AL ; cho kết quả AL = 0FFh
```

5. Lệnh TEST

```
TEST dest, source
```

Chức năng: Lệnh TEST thực hiện phép toán AND nội dung của toán hạng source và toán hạng dest, nhưng lệnh TEST không lưu kết quả của phép toán vào toán hạng dest như lệnh AND mà chỉ ảnh hưởng đến các cờ CF, OF, PF, SF, ZF.

8.3.7. Nhóm lệnh chuyển điều khiển

Ngoài việc thực hiện tuần tự hết lệnh này đến lệnh khác, trong hợp ngữ ta có thể thay đổi thứ tự thực hiện bằng các lệnh nhảy.

1. Lệnh nhảy không điều kiện JMP (jump)

```
JMP label
```

Hay

```
JMP target
```

Toán hạng target có thể là Reg hay Mem.

Lệnh JMP dùng để chuyển điều khiển chương trình từ vị trí này sang vị trí khác. JMP thực hiện bằng cách thay đổi nội dung cặp thanh ghi CS:IP để chỉ con trỏ lệnh đến vị trí mới.

Lệnh JMP có thể short, near hay far. Với dạng short hay near lệnh JMP chỉ thay đổi nội dung của thanh ghi con trỏ lệnh IP. Còn với dạng far lệnh JMP thay đổi cả thanh ghi đoạn CS và thanh ghi IP.

Ta có thể chia lệnh JMP làm hai dạng:

Dạng 1:

```
JMP label
```

Chức năng: Lệnh JMP chuyển điều khiển đến vị trí được xác định bởi nhãn label. Nhãn label có thể dạng near hay far.

- *Dạng near:* Lệnh nhảy JMP chiếm 3 byte trong bộ nhớ và nhãn label nằm trong cùng đoạn chứa lệnh JMP đó. Nếu nhãn label nằm trong khoảng từ -128 đến +127 byte đối với vị trí lệnh JMP thì ta nên dùng toán tử SHORT trước nhãn label.

```
JMP SHORT label
```

Lệnh này cũng giống như lệnh JMP label, nhưng nó chỉ chiếm 2 byte trong bộ nhớ thay vì 3 byte.

Ví dụ:

```
JMP There
.
.
There:
.
.
JMP SHORT Next ; Nhảy qua vùng dữ liệu đến nhãn Next
Msg DB 'Tin hoc'
Next:
```

- Dạng far: Lệnh nhảy chiếm 5 byte trong bộ nhớ và nhãn label có thể nằm trong đoạn khác đoạn chứa lệnh JMP.

Ví dụ: Đoạn chương trình khởi động lại máy tính trong ROM BIOS bắt đầu tại địa chỉ F000:E05B

```
Bios SEGMENT AT 0F000h
ORG 0E05Bh
Reset LABEL FAR
Bios ENDS
.CODE
```

```
.
.
JMP Reset
.
.
```

Dạng 2:

```
JMP Target
```

Chức năng: Lệnh JMP chuyển điều khiển đến vị trí được xác định trong Target. Target có thể là toán hạng thanh ghi hay bộ nhớ.

- *Target là toán hạng thanh ghi.*

Ví dụ: `JMP AX`

Chuyển nội dung thanh ghi AX → IP, sau đó chuyển điều khiển đến vị trí được xác định bởi cặp thanh ghi CS:IP

- *Target là toán hạng bộ nhớ*: có thể là dạng word (near) hay dạng doubleword (far).

Ví dụ 1:

```
JMP WORD PTR[BX] ; dạng near
JMP DWORD PTR[BX] ; dạng far
```

Ví dụ 2:

```
Reset DD 5BE000F0h
JMP Reset
```

2. Lệnh nhảy có điều kiện

Lệnh nhảy có điều kiện sẽ thực hiện lệnh nhảy hay không tuỳ thuộc vào trạng thái của thanh ghi cờ. Lệnh nhảy có điều kiện có cú pháp tổng quát sau:

```
J<Condition> Short_Label
```

Lệnh nhảy có điều kiện trước tiên kiểm tra điều kiện, sau đó nhảy đến vị trí chỉ bởi nhãn Short_Label nếu điều kiện được thoả hay tiếp tục thực hiện lệnh kế tiếp nếu điều kiện không được thoả.

Mỗi lệnh nhảy chiếm 2 byte trong bộ nhớ, byte đầu tiên biểu thị cho mã lệnh, byte sau biểu thị khoảng cách tương đối từ lệnh đến nhãn, do đó Short_Label trong lệnh nhảy có điều kiện phải nằm trong khoảng –128 đến +127 đối với vị trí lệnh nhảy có điều kiện. Muốn nhảy xa hơn ta dùng lệnh nhảy không điều kiện JMP.

Tên	Ý nghĩa	Cờ được kiểm tra	Ghi chú
JB/JNAE	Nhảy nếu nhỏ hơn/ không lớn hơn hay bằng	CF = 1	Áp dụng cho số không dấu
JBE/JNA	Nhảy nếu nhỏ hơn hay bằng/ không lớn hơn	CF = 1 hay ZF = 1	Áp dụng cho số không dấu

JA/JNBE	Nhảy nếu lớn hơn/ không nhỏ hơn hay bằng	CF = 0 và ZF = 0	Áp dụng cho số không dấu
JAE/ JNB	Nhảy nếu lớn hơn hay bằng/ không nhỏ hơn	CF = 0	Áp dụng cho số không dấu
JE/JZ	Nhảy nếu bằng	ZF = 1	
JNE/JNZ	Nhảy nếu không bằng	ZF = 0	
JL/JNGE	Nhảy nếu nhỏ hơn/ không lớn hơn hay bằng	SF<>OF	Áp dụng cho số có dấu
JLE/JNG	Nhảy nếu nhỏ hơn hay bằng/ không lớn hơn	SF<>OF hay ZF = 1	Áp dụng cho số có dấu
JG/JNLE	Nhảy nếu lớn hơn/ không nhỏ hơn hay bằng	SF = OF và ZF = 0	Áp dụng cho số có dấu
JGE/JNL	Nhảy nếu lớn hơn hay bằng/ không nhỏ hơn	SF = OF	Áp dụng cho số có dấu
JP	Nhảy nếu cờ chẵn lẻ được bật lên	PF = 1	
JNP	Nhảy nếu cờ chẵn lẻ không được bật lên	PF = 0	
JS	Nhảy nếu cờ dấu được bật lên	SF = 1	
JNS	Nhảy nếu cờ dấu không được bật lên	SF = 0	
JO	Nhảy nếu cờ tràn được bật lên	OF = 1	
JNO	Nhảy nếu cờ tràn không được bật lên	OF = 0	
JC	Nhảy nếu cờ nhớ được bật lên	CF = 1	
JNC	Nhảy nếu cờ nhớ không được bật lên	CF = 0	
JCXZ	Nhảy nếu CX bằng 0		

3. Lệnh so sánh CMP (compare)

```
CMP Left, Right
```

Left: Có thể là Reg hay Mem

Right: Có thể Reg, Mem, Immed.

Chức năng: So sánh nội dung toán hạng Left và nội dung toán hạng Right, kết quả của phép so sánh được lưu trong thanh ghi cờ hiệu. Giống như lệnh SUB, lệnh CMP lấy toán hạng Left trừ toán hạng Right. Xoá hoặc bật các cờ OF, SF, ZF, CF, PF, AF dựa trên kết quả của phép trừ, nhưng khác với lệnh SUB ở chỗ nó không thay đổi nội dung của toán hạng Left. *Lệnh CMP thường dùng chung với các lệnh chuyển điều kiện.*

Ví dụ: So sánh nội dung của hai số có dấu hạng Word A và B

- Nếu A<B: AX:= AX - BX
- Nếu A>B: BX:=BX – AX
- Nếu A=B: AX:= AX + 1 và BX:= BX - 1

```
MOV AX, A
CMP AX, B
JP Lab1
JL Lab2
INC AX
DEC BX
JMP SHORT Lab3
Lab1:
  SUB AX, B
  MOV A, AX
  JMP SHORT Lab3
Lab2:
  SUB B, AX
Lab3:
  …
```

4. Lệnh LOOP

```
LOOP Short_Label
```

CX:=CX-1

If CX<>0 Then JMP Short_Label

Chức năng: Lệnh LOOP giảm CX 1 đơn vị và chuyển điều khiển đến nhãn Short_Label nếu thanh ghi CX khác 0, ngược lại lệnh kế tiếp sau lệnh POOP sẽ được thực hiện.

Ví dụ: Cho vùng nhớ Mem có chiều dài 200 byte trong đoạn được chỉ bởi DS, đếm ký tự ‘S’ trong vùng nhớ này

```
   MOV CX, 200
   MOV BX, OFFSET Mem
   MOV Dem, 0
Cont:
   CMP BYTE PTR [BX],'S'
   JNZ NotMatch
```

```
    INC Dem
NotMatch:
    INC BX
    LOOP Cont
    .
    .
    Dem DW 0
    Mem DB 200 DUP(?)
```

5. Lệnh LOOPE (loop while equal)

```
    LOOPE Short_Label
```

CX:= CX-1

If (ZF=1) And (CX<>0) Then JMP Short_Label

Chức năng: Lệnh LOOPE giảm CX 1 đơn vị và chuyển điều khiển đến nhãn Short_Label nếu cờ ZF bằng 1 và thanh ghi CX khác 0, ngược lại lệnh kế tiếp sau lệnh LOOPE sẽ được thực hiện.

6. Lệnh LOOPZ (loop while zero)

Giống như lệnh LOOPE.

7. Lệnh LOOPNE (loop while not equal)

```
    LOOPNE Short_Label
```

CX:= CX-1

If (ZF=0) And (CX<>0) Then JMP Short_Label

Chức năng: Lệnh LOOP giảm CX 1 đơn vị và chuyển điều khiển đến nhãn Short_Label nếu cờ ZF bằng 0 và thanh ghi CX khác 0, ngược lại lệnh kế tiếp sau lệnh LOOPNE sẽ được thực hiện.

8. Lệnh LOOPNZ (loop while not zero)

Giống như lệnh LOOPNE.

Ví dụ: Cho vùng nhớ Mem có chiều dài 200 byte trong đoạn được chỉ bởi DS, tìm ký tự 'S' trong vùng nhớ này.

```
    MOV CX, 200
    MOV BX, OFFSET Mem-1
Cont:
    INC BX
    CMP BYTE PTR [BX], 'S'
    LOOPNZ Cont
```

```
    JZ Found
    ; No find
    .
    .
Found:
    .
    .
    Mem DB 200 DUP(?)
```

8.4. CHƯƠNG TRÌNH CON

8.4.1. Khái niệm

Một chương trình con gồm một tập các lệnh mà có thể gọi thực hiện nhiều lần tại nhiều vị trí khác nhau trong chương trình. Chương trình con cho phép người lập trình xây dựng chương trình một cách có cấu trúc, đặc biệt với những chương trình lớn. Chương trình con có thể nhận các tham số (thường tham số là các thanh ghi) xử lý và xuất kết quả nếu cần. Chương trình con có thể đặt tại vị trí bất kỳ trong đoạn chương trình. Trong chương trình con có thể gọi các chương trình con khác và có thể gọi chính nó (gọi là đệ quy).

8.4.2. Khai báo chương trình con

Cú pháp:

```
<tên_CTC> PROC  [NEAR|FAR]
    ;Mã chương trình
    .
    .
    RET
<tên_CTC> ENDP
```

8.4.3. Gọi chương trình con

```
CALL Label
```

Hoặc

```
CALL Target
```

Trong đó:

- Label: tên chương trình con
- Target: thanh ghi hoặc toán hạng bộ nhớ chứa địa chỉ của chương trình con được gọi

Lệnh CALL có 2 dạng:

- Dạng near: CALL chiếm 3 byte trong bộ nhớ, chương trình con phải nằm cùng đoạn mã chứa lệnh CALL.
- Dạng far: CALL chiếm 5 byte trong bộ nhớ, chương trình con có thể nằm trong đoạn mã khác đoạn mã chứa lệnh CALL.

Cách hoạt động của lệnh CALL near

- Trước tiên lệnh CALL push địa chỉ offset của lệnh kế tiếp sau lệnh CALL vào STACK.
- Sau đó nạp địa chỉ offset của lệnh đầu tiên của chương trình con vào thanh ghi con trỏ lệnh IP và chuyển điều khiển đến chương trình con (lúc này có địa chỉ xác định bởi cặp thanh ghi CS:IP).

Ví dụ:

```
1100:03E0 MOV DX, OFFSET Str
1100:03E3 CALL PrintStr
1100:03E6 Next INT 20h
1100:03E8 Str DB 'Khoa Tin Học'
1100:0600 PrintStr PROC NEAR
            MOV AH, 09h
            INT 21h
            RET
          PrintStr ENDP
```

Giải thích: Khi CPU thực hiện lệnh CALL PrintStr nó push địa chỉ offset của nhãn Next (03E6h) vào STACK và nạp địa chỉ offset của nhãn PrintStr (0600h) vào thanh ghi con trỏ lệnh IP. Cuối cùng chuyển điều khiển đến địa chỉ mới 0600h để thực hiện chương trình con PrintStr.

Sau khi thực hiện xong chương trình con PrintStr, CPU sẽ pop địa chỉ offset của nhãn Next (03E6h) ra khỏi STACK vào thanh ghi con trỏ lệnh IP và quyền điều khiển trả về chương trình chính tại địa chỉ 03E6h.

Cách hoạt động của lệnh CALL far

- Trước tiên lệnh CALL push thanh ghi đoạn CS và địa chỉ offset của lệnh kế sau lệnh CALL vào STACK.
- Sau đó nạp địa chỉ đoạn và địa chỉ offset của nhãn Label vào cặp thanh ghi CS:IP và chuyển điều khiển đến chương trình con (lúc này có địa chỉ xác định bởi cặp thanh ghi CS:IP).

Ví dụ:

```
1100:03E0 MOV DX, OFFSET Str
1100:03E3 CALL PrintStr
1100:03E6 Next INT 20h
```

```
1100:03E8 Str DB ‘Khoa CNTT’
4100:0600 PrintStr PROC FAR
          MOV AH, 09h
          INT 21h
          RET
        PrintStr ENDP
```

Giải thích: Khi CPU thực hiện lệnh CALL FAR PTR PrintStr nó push địa chỉ đoạn (1100h) và địa chỉ offset của nhãn Next (03E6h) lần lượt vào STACK. Rồi nạp địa chỉ đoạn (4100h) và địa chỉ offset (0600h) của chương trình con PrintStr vào cặp thanh ghi CS:IP. Cuối cùng chuyển điều khiển đến địa chỉ mới 4100:0600 để thực hiện chương trình con PrintStr.

Khi CPU thực hiện lệnh RET để kết thúc chương trình con PrintStr, nó pop địa chỉ offset của nhãn Next (03E6h) và địa chỉ đoạn (1100h) ra khỏi STACK và nạp lần lượt vào thanh ghi IP và CS và quyền điều khiển được trả về chương trình chính tại địa chỉ 1100:03E6.

8.4.4. Truyền tham số

Có hai cách để chuyển tham số: trong thanh ghi hoặc trong STACK. Cách chuyển thanh ghi thường dùng trong chương trình hợp ngữ thuần tuý còn trong chương trình ngôn ngữ cấp cao hoặc chương trình con hợp ngữ gọi từ chương trình ngôn ngữ cấp cao thì thường dùng STACK.

Trị trả về: Các chương trình con thường trả về cho chương trình gọi. Nếu chương trình con hợp ngữ được gọi từ ngôn ngữ cấp cao thì phải theo quy định của chương trình chức năng cấp cao. Nếu viết chương trình hợp ngữ thuần tuý thì tuỳ ý, có thể chứa trong các thanh ghi. Sau đây là một số quy ước ta có thể dùng:

Ví dụ: Dùng chương trình con in ra ba chuỗi sau:

Hello, world!

Hello, solar system!

Hello, universe!

```
DOSSEG
.MODEL SMALL
.STACK 200h
.DATA
  WorldMessage DB ‘Hello, world!’, 0Dh, 0Ah,0
  SolarMessage DB ‘Hello, solar system!’, 0Dh, 0Ah,0
  UniverseMessage DB ‘Hello, universe!’, 0Dh, 0Ah,0
.CODE
Begin:
```

```
  MOV AX, @DATA
  MOV DS, AX
  MOV BX, OFFSET WorldMessage
  CALL PrintString ; In 'Hello, world!'
  MOV BX, OFFSET SolarMessage
  CALL PrintString ; In 'Hello, solar system!'
  MOV BX, OFFSET UniverseMessage
  CALL PrintString ; In 'Hello, universe!'
  MOV AX, 4C00h
  INT 21h
; Chương trình con in chuỗi tận cùng bằng 0 lên màn hình
; Input: DS:DB = Địa chỉ chỉ tới chuỗi muốn in
PrintString PROC NEAR
  PrintStringLoop:
    MOV DL, [BX] ; Lấy một ký tự trong chuỗi
    OR DL, DL ; Ký tự vừa lấy ra là 0?
    JZ EndPrintString ; Nếu đúng thì kết thúc
    INC BX ; Nếu không trỏ tới ký tự kế tiếp
    MOV AH, 02h ; In ký tự trong DL
    INT 21h
    JMP PrintStringLoop ; In ký tự kế nếu còn
  EndPrintString:
   RET
PrintString ENDP
END Begin
```

8.5. MACRO

8.5.1. Khái niệm

Macro là một dãy các câu lệnh của Assembler mà có có thể xuất hiện nhiều lần trong một chương trình. Giống như chương trình con đó, mỗi macro có một tên riêng.

Khi một macro được định nghĩa, thì trong chương trình nguồn thay vì viết một dãy các câu lệnh đã được định nghĩa bởi macro, ta chỉ cần viết tên của macro đó.

8.5.2. Khai báo và gọi Macro

```
Name MACRO [Parameter_List]
   ;Mã lệnh trong macro
ENDM
```

Macro gồm 3 phần:

1. Phần header: Dùng để khai báo một macro.

```
Name MACRO [Parameter_List]
```

Trong đó:

- Name: Tên của macro cần định nghĩa.
- Parameter_List: Là một dãy các tham số cách nhau bởi dấu phẩy.

2. Phần thân: Gồm một tập hợp các câu lệnh của Assembler mà macro cần định nghĩa.

3. Phần cuối: Là chỉ dẫn ENDM. Dùng để đánh dấu điểm kết thúc của macro.

Ví dụ: Muốn xuất ký tự 'D' ra màn hình ta viết các lệnh

```
MOV AH, 02h
MOV DL, 'D'
INT 21h
```

Nếu muốn xuất thêm ký tự 'E' ra màn hình ta phải viết thêm các câu lệnh trên lại:

```
MOV AH, 02h
MOV DL, 'E'
INT 21h
```

Vậy nếu chương trình cần xuất nhiều ký tự khác nhau ra màn hình ta phải viết lại đoạn chương trình trên nhiều lần. Để rút gọn chương trình ta có thể viết chương trình con Out_Char.

```
; DL = Chứa ký tự cần xuất
Out_Char PROC
   MOV AH, 02h
   INT 21h
   RET
Out_Char ENDP
```

Khi muốn xuất ký tự 'D' và 'E' ra màn hình ta viết đoạn chương trình:

```
MOV DL, 'D'
CALL Out_Char
MOV DL,'E'
```

```
CALL Out_Char
```

Nếu dùng chương trình con thì chương trình có ngắn gọn hơn, nhưng chúng ta còn phải dùng đến hai câu lệnh. Nếu tham số càng nhiều ta càng viết nhiều câu lệnh hơn trước khi gọi chương trình con. Macro có thể giải quyết vấn để này ngắn gọn hơn như sau:

```
@Out_Char MACRO Char
   MOV AH, 02h
   MOV DL, Char
   INT 21h
ENDM
```

Muốn xuất ký tự 'D' và 'E' ra màn hình ta chỉ cần viết đơn giản như sau:

```
@Out_Char 'D'
@Out_Char 'E'
```

Rõ ràng macro viết ngắn gọn hơn chương trình con nhiều.

8.5.3. Macro và chương trình con

Phân biệt giữa macro và chương trình con

- Đối với chương trình con: Các lệnh của chương trình con khi Assembler dịch chỉ xuất hiện có một lần duy nhất trong chương trình, khi cần gọi chương trình này ra ta chỉ dùng lệnh CALL.
- Đối với macro: Khi dịch Assembler sẽ thay đổi mỗi tên của macro trong chương trình bằng các câu lệnh mà macro đó đã định nghĩa.

Ưu điểm của macro

- Có thể thay đổi các tham số truyền cho macro một cách dễ dàng.
- Macro thực hiện nhanh hơn chương trình con vì CPU không cần thực hiện lệnh CALL và RET.
- Các macro có thể tập hợp thành một thư viện macro.
- Macro có thể lồng nhau, có thể định nghĩa lại và có thể đệ quy.

Tuy nhiên có khuyết điểm là mỗi lần macro được dùng, Assembler sẽ thay tên macro bằng một câu lệnh tương ứng mà macro đã định nghĩa, do đó nó làm cho chương trình mã máy dài hơn, tốn nhiều bộ nhớ hơn.

8.5.4. Chỉ dẫn LOCAL

```
LOCAL LocalName [,LocalName]…
```

Chức năng: Chỉ dẫn này được dùng trong một macro để định nghĩa các ký hiệu mà các ký hiệu này chỉ có thể được dùng trong macro đó.

Xét ví dụ sau:

```
Wait MACRO Count
```

```
    PUSH CX
    MOV CX, Count
    Next:
    LOOP Next
    POP CX
ENDM
```

Macro Wait chỉ có thể được gọi tối đa một lần duy nhất vì ta biết một nhãn chỉ có thể được định nghĩa một lần duy nhất trong chương trình. Muốn giải quyết vấn đề này ta phải dùng chỉ dẫn LOCAL dùng trong phần thân của macro để báo cho Assembler biết các nhãn được khai báo bởi chỉ dẫn LOCAL được đổi thành một tên nhãn mới trong chương trình mỗi khi một macro được gọi. Tên nhãn mới có dạng:

```
??Number
```

Number là một số hex có trị trong khoảng 0000h đến FFFFh. Do đó không nên định nghĩa các ký hiệu có dạng ??Number vì có thể sinh ra các ký hiệu trùng nhau.

Chú ý: Chỉ dẫn này phải được đặt ở đầu macro ngay sau phần khai báo.

Ví dụ: Muốn macro Wait có thể được gọi nhiều trong chương trình ta phải thêm chỉ dẫn LOCAL vào macro Wait.

```
Wait MACRO Count
    LOCAL Next
    PUSH CX
    MOV CX, Count
    Next:
    LOOP Next
    POP CX
ENDM
```

Giải thích:

- Khi gọi macro: Wait 1000

Lần dầu Assembler sẽ dịch macro Wait như sau:

```
PUSH AX
MOV CX, 1000
??0000:
LOOP ??0000
POP CX
```

Ta thấy nhãn Next được thay thế bằng tên một nhãn mới ??0000

- Khi gọi macro: Wait 3000

Lần thứ hai Assembler sẽ dịch vào như sau:

```
PUSH AX
MOV CX, 3000
??0001:
LOOP ??0001
POP CX
```

Ta thấy nhãn Next được thay bằng một tên nhãn mới ??0001

CÂU HỎI ÔN TẬP VÀ BÀI TẬP

1. Giả thiết rằng các số liệu sau đây được nạp vào bộ nhớ bắt đầu tại vị trí offset 0000h:

```
A db 7
B dw 1ABCh
C db "Hello"
```

   a. Hãy cho biết các địa chỉ offset của các biến A, B, C
   b. Hãy cho biết nội dung của byte tại offset 0002h dưới dạng số hex
   c. Hãy cho biết nội dung của byte tại offset 0004h dưới dạng số hex
   d. Hãy cho biết địa chỉ offset của ký tự "o" trong "Hello"

2. Giả thiết rằng các dữ liệu sau đây được khai báo liên tục trong bộ nhớ bắt đầu từ địa chỉ offset 0500h

```
S1 db "ASSEMBLER2011"
S2 db 0, 15 dup('0')
B1 db 10, 13
W1 dw 128
W2 dw 10,13
```

   a. Hãy cho biết địa chỉ offset của các biến trên
   b. Cho biết nội dung của ô nhớ tại địa chỉ 0510h, [B1+3]
   c. Cho biết giá trị của thanh ghi SI, AX sau khi thực hiện các lệnh sau:

```
LEA SI, S2
MOV AH, S2
MOV AL, S2+1
```

   d. Không dùng lệnh STZ, viết các lệnh cần thiết để bật cờ ZF (Zero) lên 1
   e. Cho biết giá trị của các thanh ghi AX, DX sau khi thực hiện các lệnh sau:

```
MOV BX, W1
MOV AX, W2
MUL BX
```

3. Giả thiết rằng các dữ liệu sau đây được khai báo liên tục trong bộ nhớ bắt đầu từ địa chỉ offset 0500h

```
S1 db "ABCDE123456789"
B1 db 10, 13
W1 dw 64
S2 db 0, 10 dup('0')
W2 dw 1013h
```

   a. Hãy cho biết địa chỉ offset của các biến trên
   b. Cho biết nội dung của ô nhớ tại địa chỉ 0510h, [B1+3]
   c. Cho biết giá trị của thanh ghi SI, AX sau khi thực hiện các lệnh sau:

```
LEA SI, S2
MOV AH, S2
MOV AL, S2+1
```

   d. Cho biết giá trị của các thanh ghi AX, DX sau khi thực hiện các lệnh sau:

```
MOV AH, [S1]
MOV AL, [S1+6]
MOV BX, W1
DIV BX
```

   e. Cho biết giá trị của các thanh ghi AX, DX sau khi thực hiện các lệnh sau:

```
MOV BX, W1
MOV AX, W2
MUL BX
```

   f. Cho biết giá trị của các thanh ghi AX, BX sau khi thực hiện các lệnh sau:

```
MOV BX, W1
MOV AX, W2
XOR BX,AX
```

4. Giả thiết rằng các dữ liệu sau đây được khai báo liên tục trong bộ nhớ bắt đầu từ địa chỉ offset 0900h

```
S1 db "ABCDE123456789", 0, 1, 2, 3
B1 db 10, 13
```

```
W1 dw 64, 65, 66
S2 db 0, 10 dup(0)
W2 dw 1013h
```

a. Biễu diễn dữ liệu trên trong bộ nhớ
b. Hãy cho biết địa chỉ offset của các biến trên
c. Cho biết nội dung của ô nhớ tại địa chỉ 0910h, [B1+3], [W1+1]
d. Cho biết giá trị của thanh ghi AX, DX sau khi thực hiện lệnh các lệnh sau:

```
MOV AX, [W1+2]
MOV BH, 0
MOV BL, [B1]
MUL BX
```

5. Giả thiết rằng các dữ liệu sau đây được khai báo liên tục trong bộ nhớ bắt đầu từ địa chỉ offset 0900h

```
S1 db 0, 5 dup('A')
B1 db 10, 13
W1 dw 64, 1000
B2 db 48, '0', 48
S2 db "ABCDE1234567890"
W2 dw 1013h, 10
```

a. Biễu diễn dữ liệu trên trong bộ nhớ
b. Hãy cho biết địa chỉ offset của các biến trên
c. Cho biết nội dung của ô nhớ tại địa chỉ 0920h, [B1+5], [W1+3]
d. Cho biết giá trị của thanh ghi AX, DX sau khi thực hiện lệnh các lệnh sau:

```
MOV AX, [W1+1]
MOV BH, 0
MOV BL, [B1]
MUL BX
```

6. Cho các dữ liệu đã được khai báo như sau:

```
B1 db ?
B2 db ?
W1 dw ?
```

```
W2 dw ?
```

a. Hãy viết các lệnh hợp ngữ để thực hiện các công việc:
   i. Hoán vị nội dung của B1 và B2
   ii. W1:= B1 + B2;
   iii. W1:= W2 + B1*5

b. Cho biết giá trị của các thanh ghi AX, DX sau khi thực hiện các lệnh sau:

```
MOV AX, 8000h
MOV BX, 4
MUL BX
MOV DX,3
MOV AX,8
MOV BX,4
DIV BX
```

7. Cho đoạn chương trình sau:

```
Mov CH,0
Mov CL, a
Lea BX,chuoi
Mov AL,67
nhan_A:
lap:
Mov DL,[BX]
Cmp AL,DL
Jz nhan_B
Inc BX
Loop lap
nhan_B:
Int 20h
du_lieu:
a db 12
chuoi db "THITHANHCONG"
```

Giả sử địa chỉ offset của nhãn du_lieu là 300h, giá trị của các thanh ghi AX, BX, CX, DX khi bắt đầu chạy đoạn chương trình trên bằng 0. Hãy:

a. Cho biết giá trị của các thanh ghi AX, BX, CX, DX khi đoạn chương trình trên thực hiện đến nhan_A

b. Cho biết giá trị của các thanh ghi AX, BX, CX, DX khi đoạn chương trình trên thực hiện đến nhan_B

8. Hãy cho biết mỗi lệnh dưới đây là hợp lệ hay không hợp lệ, trong đó W1 và W2 là các biến kiểu Word; B1 và B2 là các biến kiểu byte:

   a. Mov DS, AX

   b. Mov DS, 100h

   c. Mov DS, ES

   d. Mov W1, DS

   e. XCHG W1, W2

   f. Sub 5, B1

   g. Add B1, B2

   h. Add AL, 256

   i. Mov W1, B1

9. Chỉ dùng các lệnh MOV, ADD, SUB, INC, DEC và NEG. Hãy dịch các dòng lệnh gán của ngôn ngữ cấp cao sau đây sang ngôn ngữ Assembly với các biến kiểu Word:

   a. A:=B-A

   b. A:=-(A-1)

   c. C:=A+B

   d. B:=3*B+7

   e. B:=B-A-1

10. Viết các lệnh Assembly cho mỗi cấu trúc điều khiển sau:

    a. If AX<0 then BX:=1

    b. If AX<0 then AH:=0FFh Else AH:=0

    c. If AX<BX then

       If BX<CX then AX:=0

       Else BX:=0

    d. If (AX<BX) or (BX<CX) then DX:=0 Else DX:=1

    e. If (AX<BX) and (BX<CX) then DX:=0 Else DX:=1

    f. While AX<>0 do AX:=AX-1

    g. For AX:=1 to 100 do BX:=BX+AX

    h. Repeat AX:=AX+1 Until AX>100;

11. Cho biết giá trị của thanh ghi AX, DX sau khi thực hiện các lệnh sau:

    a.

```
MOV AX, 8F3Ch
MOV BX, 32
MUL BX
```

b.

```
MOV AX, 901Ch
MOV DX, 0
MOV BX, 8
DIV BX
```

c.

```
MOV AX, 7000h
MOV DX, 1
MOV BX, 4
DIV BX
```

12. Chỉ dùng các lệnh MOV, ADD, SUB, MUL, DIV, viết các lệnh cần thiết để thực hiện các công việc sau:
    a. W1:= W1 + B1;
    b. W2:= B1*B2 + W2;
    c. Với B1, B2 là biến kiểu byte; W1, W2 là biến kiểu word
13. Cho biết giá trị của thanh ghi AX, DX sau khi thực hiện các lệnh sau:

a.

```
MOV AX, 8D3Ah
MOV SI, 16
MUL SI
```

b.

```
MOV AX, 9A1Eh
MOV DX, 8F4Ch
ROL AX, 1
AND DX, AX
```

c.

```
MOV AX, 8000h
MOV DX, 1
MOV BX, 8
DIV BX
```

14. Viết các chương trình sau:
    a. In một thông báo ra màn hình.
    b. In thông báo và nhập một ký tự từ bàn phím.
    c. Nhập 2 ký tự và in 2 ký tự đó ra màn hình.
    d. Nhập 2 số trong khoảng 0..9, in ra màn hình tổng, tích của 2 số đó.
    e. Nhập 2 ký tự, so sánh 2 ký tự đó với nhau.
    f. Nhập 1 chuỗi
    g. In một chuỗi ra màn hình
    h. Đảo ngược một chuỗi
    i. Chuyển chuỗi thành hoa/thường
    j. Sao chép một chuỗi vào chuỗi khác
    k. Ghép 2 chuỗi với nhau vào chuỗi thứ 3
    l. Tìm vị trí xuất hiện của ký tự trong chuỗi
    m. Đổi một chuỗi số nguyên không dấu thành số
    n. Đổi một chuỗi các ký số nhị phân/thập lục phân thành số
    o. In một số nguyên không dấu ra màn hình dưới hệ k
15. Thực hiện các bài tập ở câu 14 có sử dụng chương trình con, macro.

PHỤ LỤC

CÁC HÀM CỦA DOS VỚI NGẮT 21H

1. HÀM 01H

Nhập từ bàn phím một ký tự (có hiển thị trên màn hình)

Input:

```
AH =01h
```

Output:

```
AL = Mã ASCII của ký tự vừa nhập.
```

Chức năng này chờ nhập một ký tự từ bàn phím và hiển thị ký tự vừa nhập lên màn hình.

2. HÀM 02H

Xuất ra màn hình một ký tự

Input:

```
AH= 02h
DL = Mã ASCII của ký tự cần xuất
```

Output:

```
Không.
```

3. HÀM 08H

Nhập từ bàn phím một ký tự (không hiển thị trên màn hình)

Input:

```
AH = 08h
```

Output:

```
AL= Mã ASCII của ký tự vừa nhập.
```

Chức năng này chờ nhập một ký tự từ bàn phím và không hiển thị ký tự vừa nhập lên màn hình.

4. HÀM 09H

Xuất một chuỗi ký tự ra màn hình

Input:

```
AH= 09h
DS:DX = Địa chỉ của chuỗi ký tự cần xuất
```

Output:

```
Không.
```

Chú ý: Chuỗi ký tự phải kết thúc bằng ký tự '$'

5. HÀM 0AH

Nhập một chuỗi từ bàn phím.

Input:

```
AH= 0Ah
DS:DX = Địa chỉ của vùng đệm ký tự nhận dữ liệu
```

Output:

```
Không.
```

Trước khi gọi hàm này vùng đệm (Buffer) có dạng sau:

Offset	Ý nghĩa
0	Số ký tự tối đa của chuỗi được nhập vào
1	Số ký tự thật sự nhập vào (không kể Enter)
2, 3, 4, …	Các ký tự đã nhập (kể cả ký tự Enter)

Ví dụ:

```
Buffer DB 81 ; Chiều dài tối đa của chuỗi ký tự
       DB 0 ; Số ký tự thực đã nhập của chuỗi
       DB 81 DUP(0) ; Nơi chứa các ký tự nhập vào
```

Hàm này khi nhập vào kết thúc bằng dấu ENTER.

TÀI LIỆU THAM KHẢO

[1] Tống Văn On, Hoàng Đức Hải – *Giáo trình cấu trúc máy tính* – Nhà xuất bản Lao động xã hội

[2] Võ Văn Chín, Nguyễn Hồng Vân, Phạm Hữu Tài – *Giáo trình Kiến trúc máy tính* – Đại học Cần Thơ

[3] Mostafa Abd-El-Barr, Hesham El-Rewin – *Fundamentals of Computer Organization and Architecture* – Wiley

[4] William Stallings – *Computer Organization and Architecture Designing for perfomance* – Prentice Hall

[5] Randall Hyde – *The Art of Assembly Language*