MỤC LỤC

DANH MỤC HÌNH VẼ, BẢNG BIỂU

DANH MỤC CÁC TỪ VIẾT TẮT

STT	Từ viết tắt	Diễn giải
1	ENIAC	Electronic Numerical Integrator and Computer
2	IAS	Princeton Institute for Advanced Studies
3	ALU	Arithmetic And Logic Unit
4	IC	Integrated Circuit
5	SSI	Small Scale Integration
6	MSI	Medium Scale Integration
7	LSI	Large Scale Integration
8	VLSI	Very Large Scale Integration
9	ASIMO	Advanced Step Innovative Mobility
10	RISC	Reduced Instructions Set Computer
11	CISC	Complex Instruction Set Computer
12	MAR	Memory Address Register
13	MBR	Memory Buffer Register
14	PCU	Power Control Unit
15	IA	Intel Itanium

LỜI NÓI ĐẦU

Kiến trúc máy tính là học phần chuyên môn trong chương trình đào tạo của ngành Công nghệ thông tin. Mục đích của môn học: trang bị cho sinh viên các kiến thức cơ bản về kiến trúc máy tính, kiến trúc tập lệnh, nguyên lý hoạt động và tổ chức của máy tính cũng như những vấn đề cơ bản trong thiết kế một hệ thống máy tính. Trên cơ sở đó có thể đánh giá được hiệu năng của máy tính, khai thác và sử dụng hiệu quả các loại máy tính hiện hành. Với những kiến thức thu được, sinh viên có thể giải quyết được các bài toán về thiết kế mạch Logic, các hệ lệnh, xây dựng bộ nhớ, thay thế khung trang,… và sau này làm tiền đề giúp sinh viên có thể đi sâu tìm hiểu những môn học chuyên ngành khác như Hệ điều hành, Mạng máy tính, An toàn thông tin...

Học phần Kiến trúc máy tính được học trước môn Hệ điều hành, để cung cấp các kiến thức về tập lệnh, bộ nhớ, hệ thống vào ra.. cho sinh viên trước khi tiếp cận phần kiến thức của môn học hệ điều hành. Ngoài ra Kiến trúc máy tính giúp sinh viên hiểu rõ cấu trúc, chức năng, mối liên kết và quy trình vận hành của hệ thống máy tính nên là kiến thức nền cho các môn học chuyên ngành như Lập trình hướng đối tượng, Kỹ thuật mô phỏng... và trợ giúp sinh viên có kỹ năng tốt trong việc vận hành, sử dụng máy tính khi học các môn chuyên ngành về thiết kế , lập trình trên máy tính.

Để đáp ứng nhu cầu học tập của sinh viên chuyên ngành Công nghệ thông tin; Mạng máy tính và truyền thông dữ liệu, trường Đại học Kinh Tế Kỹ Thuật Công Nghiệp - Khoa Công Nghệ Thông Tin tổ chức biên soạn tài liệu học tập "Kiến Trúc Máy Tính". Tài liệu này được biên soạn theo đề cương chi tiết môn học Kiến trúc máy tính, nhằm giúp sinh viên khoa Công nghệ thông tin có một tài liệu cô đọng, sát với chương trình đào tạo, làm tài liệu học tập bổ ích và cần thiết. Ngoài ra tài liệu Kiến trúc máy tính còn là kiến thức nền tảng trợ giúp các em tiếp thu những kiến thức trong các môn học khác của chương trình đào tạo.

Nội dung của tài liệu bao gồm sáu chương, trong mỗi chương bao gồm các phần nội dung chủ yếu:

- Mục đích của chương
- Yêu cầu đối với sinh viên
- Nội dung bài giảng lý thuyết
- Câu hỏi hướng dẫn ôn tập, thảo luận
- Câu hỏi trắc nghiệm, bài tập áp dụng,.

Chương 1 - TỔNG QUAN: trình bày những kiến thức, những khái niệm chung về kiến trúc máy tính, hiệu năng máy tính và toán logic đại số boolen.

Chương 2 - THIẾT KẾ HỆ LỆNH: trình bày kiến thức về các dạng lệnh, kích thước mã lệnh, các phương pháp xác định địa chỉ và số lượng tham số trong một lệnh.

Chương 3 - BỘ XỬ LÝ TRUNG TÂM CPU: trình bày kiến thức về cách tổ chức, vận hành, chức năng của bộ xử lý trung tâm CPU.

Chương 4 - THIẾT KẾ BỘ NHỚ: trình bày kiến thức về cách tổ chức, xây dựng bộ nhớ, bộ nhớ cache, bộ nhớ ảo và các kỹ thuật phân trang, phân đoạn.

Chương 5 - HỆ THỐNG VÀO RA: trình bày các kiến thức về cách tổ chức, vận hành hệ thống vào ra.

Chương 6 - MỘT SỐ KIẾN TRÚC HIỆN ĐẠI: trình bày các kiến thức về một số kiến trúc máy tính hiện đại và trong tương lai.

Do thời gian và trình độ có hạn nên tài liệu học tập khó có thể tránh khỏi những thiếu sót nhất định. Chúng tôi luôn mong nhận được sự góp ý của bạn đọc để tài liệu học tập được tái bản hoàn thiện hơn trong những lần sau.

Xin chân thành cảm ơn!

Nhóm biên soạn

Th.s Nguyễn Thu Hiền

Th.s Trần Thanh Đại

CHƯƠNG 1: TỔNG QUAN

***Mục đích**: Giới thiệu cách phân loại máy tính. Phân tích các thông số đo khả năng xử lý của máy tính, làm quen với toán logic - đại số boolean, tìm hiểu một số mạch logic đơn giản, tìm hiểu kiến trúc tuần tự Vonneuman và nguyên lý hoạt động của kiến trúc này.*

***Yêu cầu**: Sinh viên biết cách phân loại máy tính, biết rõ đặc điểm của từng loại. Nắm vững các thông số đo khả năng xử lý của máy tính. Biết cách thiết kế một số mạch logic cơ bản. Áp dụng vào làm bài tập.*

1.1 Máy tính và phân loại máy tính

1.1.1 Máy tính

1.1.1.1 Khái niệm

Máy tính là thiết bị điện tử thực hiện các công việc: nhận dữ liệu vào, xử lý dữ liệu theo dãy các lệnh được nhớ sẵn bên trong và đưa dữ liệu (thông tin) ra. Thông tin lưu trữ trên máy tính là thông tin số hoặc biểu diễn dưới dạng quy luật logic. Hoạt động của máy tính được điều khiển bằng một phần mềm gọi là hệ điều hành. Máy tính được lắp ghép bởi các thành phần có thể thực hiện các chức năng đơn giản đã định nghĩa trước. Quá trình tác động tương hỗ phức tạp của các thành phần này tạo cho máy tính một khả năng xử lý thông tin. Nếu được thiết lập chính xác (thông thường bởi các chương trình máy tính) máy tính có thể mô phỏng lại một số khía cạnh của một vấn đề hay của một hệ thống. Trong trường hợp này, khi được cung cấp một bộ dữ liệu thích hợp nó có thể tự động giải quyết vấn đề hay dự đoán trước sự thay đổi của hệ thống. [1]

Từ "máy tính" (computers), đầu tiên được dùng cho những người tính toán số học, có hoặc không có sự trợ giúp của máy móc, nhưng hiện nay nó hoàn toàn có nghĩa là một loại máy móc. Đầu tiên máy tính chỉ giải các bài toán số học, nhưng máy tính hiện đại làm được nhiều hơn thế. Đến những năm 1990, khái niệm máy tính đã thực sự tách rời khỏi khái niệm điện toán và trở thành một ngành khoa học riêng biệt với nhiều lĩnh vực đa dạng và khái niệm hơn hẳn ngành điện toán thông thường và được gọi là công nghệ thông tin.

1.1.1.2 Các nguyên lý cơ bản của máy tính

Máy tính có thể làm việc thông qua sự chuyển động của các bộ phận cơ khí, điện tử (electron), photon, hạt lượng tử hay các hiện tượng vật lý khác đã biết. Mặc dù máy tính được xây dựng từ nhiều công nghệ khác nhau song gần như tất cả các máy tính hiện nay đều là máy tính điện tử. Máy tính có thể trực tiếp mô hình hóa các vấn đề cần được giải quyết. Trong khả năng của nó các vấn đề cần được giải quyết sẽ được mô phỏng gần giống nhất với những hiện tượng vật lý đang khai thác. Ví dụ, dòng chuyển động của các điện tử có thể được sử dụng để mô hình hóa sự chuyển động của nước trong đập… Trong phần lớn các máy tính ngày nay, trước hết, mọi vấn đề sẽ được chuyển thành các yếu tố toán học bằng cách diễn tả mọi thông tin liên quan thành các số theo hệ nhị phân (hệ thống đếm dựa trên các số 0 và 1 hay còn gọi là hệ đếm cơ số hai). Sau đó, mọi tính toán trên các thông tin này được tính toán bằng đại số Boole (Boolean algebra). Các mạch điện tử được sử

dụng để miêu tả các phép tính Boole. Vì phần lớn các phép tính toán học có thể chuyển thành các phép tính Boole nên máy tính điện tử đủ nhanh để xử lý phần lớn các vấn đề toán học.

Máy tính không thể giải quyết tất cả mọi vấn đề của toán học. Alan Turing đã sáng tạo ra khoa học lý thuyết máy tính trong đó đề cập tới những vấn đề mà máy tính có thể hay không thể giải quyết. Khi máy tính kết thúc tính toán một vấn đề, kết quả của nó được hiển thị cho người sử dụng thấy thông qua thiết bị xuất như: màn hình, máy in… Máy tính chỉ đơn giản thi hành các tìm kiếm cơ khí trên các bảng màu và đường thẳng đã lập trình trước, rồi sau đó thông qua các thiết bị đầu ra (màn hình, máy in,…) chuyển đổi chúng thành những ký hiệu mà con người có thể cảm nhận được thông qua các giác quan (hình ảnh trên màn hình, chữ trên văn bản được in ra). Chỉ có bộ não của con người mới nhận thức được những ký hiệu này tạo thành các chữ hay số và gắn ý nghĩa cho chúng.

Máy tính số (Digital computer) 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.

Tập lệnh của máy tính tạo thành một ngôn ngữ giúp con người có thể tác động lên máy tính, gọi là ngôn ngữ máy (machine language). Tuy nhiên, hầu hết các ngôn ngữ máy đều đơn giản nên để thực hiện một yêu cầu nào đó, người thiết kế phải thực hiện một công việc phức tạp. Đó là chuyển các yêu cầu thành các chỉ thị có chứa trong tập lệnh của máy. Vấn đề này có thể giải quyết bằng cách thiết kế một tập lệnh mới thích hợp cho con người hơn tập lệnh đã cài đặt sẵn trong máy (built-in). Ngôn ngữ máy sẽ được gọi là ngôn ngữ cấp một (L1) và ngôn ngữ vừa được hình thành gọi là ngôn ngữ cấp hai (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. [4]

1.1.2 Phân loại máy tính

1.1.2.1 Theo mục đích sử dụng

Siêu máy tính

Một siêu máy tính là một máy tính vượt trội trong khả năng và tốc độ xử lý. Thuật ngữ Siêu Tính Toán được dùng lần đầu trong báo New York World vào năm 1920 để nói đến những bảng tính (tabulators) lớn của IBM làm cho trường Đại học Columbia. Siêu máy tính hiện nay có tốc độ xử lý hàng nghìn teraflop (một teraflop tương đương với hiệu suất một nghìn tỷ phép tính/giây) hay bằng tổng hiệu suất của 6.000 chiếc máy tính hiện đại nhất hiện nay gộp lại (một máy có tốc độ khoảng từ 3-3,8 gigaflop). Có thể hiểu siêu máy tính là hệ thống những máy tính làm việc song song.

Siêu máy tính cỡ nhỏ

Siêu máy tính cỡ nhỏ (minisupercomputers) là một dòng máy tính xuất hiện vào giữa thập kỉ 1980. Khi việc tính toán khoa học dùng bộ xử lý vector trở nên phổ biến hơn, nhu cầu sử dụng hệ thống giá thành thấp để dùng ở cấp độ phòng ban thay vì ở cấp độ doanh nghiệp mang đến cơ hội cho các nhà kinh doanh máy tính mới bước vào thị trường. Nhìn chung, mục tiêu về giá cả của các máy tính nhỏ hơn này là một phần mười các siêu máy tính lớn hơn. Đặc trưng của các máy tính này là sự kết hợp giữa xử lý vector và đa xử lý cỡ nhỏ (small-scale).

Máy tính lớn

Máy tính lớn (Mainframe) là loại máy tính có kích thước lớn được sử dụng chủ yếu bởi các công ty lớn như các ngân hàng, các hãng bảo hiểm,... để chạy các ứng dụng xử lý khối lượng lớn dữ liệu. Ví dụ: kết quả điều tra dân số, thống kê khách hàng, doanh nghiệp, và xử lý các giao tác thương mại. So với các máy tính loại nhỏ như máy tính cá nhân, máy tính lớn có thể nhận hàng ngàn lệnh cùng một lúc.

Máy chủ doanh nghiệp

Máy chủ doanh nghiệp là hệ thống máy tính chủ yếu phục vụ cho một doanh nghiệp lớn. Ví dụ các loại máy chủ như máy chủ web, máy chủ in ấn, và máy chủ cơ sở dữ liệu. Tính chất chủ yếu để phân biệt một máy chủ doanh nghiệp là tính ổn định vì ngay cả một sự cố ngắn hạn cũng có thể gây thiệt hại hơn cả việc mua mới và cài đặt mới hệ thống. Lấy ví dụ, một hệ thống máy tính trong thị trường chứng khoán cấp quốc gia có trục trặc, chỉ cần ngưng hoạt động trong vòng vài phút có thể cho thấy việc thay thế toàn bộ hệ thống hiện tại bằng một hệ thống đáng tin cậy hơn vẫn là giải pháp tốt hơn.

Máy tính mini

Thuật ngữ máy tính mini được phát triển vào những năm 1960 để mô tả các máy tính nhỏ hơn sử dụng các bóng bán dẫn và công nghệ bộ nhớ lõi. Máy tính mini còn được gọi là máy tính tầm trung. Chúng được sử dụng trong kiểm soát quá trình sản xuất, chuyển mạch điện thoại và kiểm soát thiết bị phòng thí nghiệm. Trong những năm 1970, chúng là

phần cứng được sử dụng để khởi động ngành công nghiệp thiết kế hỗ trợ máy tính (CAD) và các ngành công nghiệp tương tự khác mà cần có một hệ thống dành riêng nhỏ hơn.

Máy trạm

Workstation là một Microcomputer được thiết kế dành để chạy các ứng dụng kỹ thuật hoặc khoa học. Mục đích chính cho việc tạo ra máy tính này là để phục vụ cho một người tại một thời điểm, có thể kết nối với nhau qua mạng máy tính và phục vụ nhiều người cùng lúc. Một nhóm các máy trạm có thể xử lý các công việc của một máy tính lớn Main Frame nếu như được kết nối mạng với nhau.

Các máy trạm cung cấp hiệu suất cao hơn máy tính để bàn, đặc biệt là về CPU, đồ họa, bộ nhớ và khả năng xử lý đa nhiệm. Nó được tối ưu hóa cho việc xử lý các loại dữ liệu phức tạp như các bản vẽ 3D trong cơ khí, các mô phỏng trong thiết kế, vẽ và tạo ra các hình ảnh động, các logic toán học. Thông thường các bộ phận giao tiếp với máy trạm bao gồm: màn hình với độ phân giải cao, bàn phím và chuột.

Máy tính cá nhân

Máy tính cá nhân (personal computer) là một loại máy tính mà giá cả, kích thước và sự tương thích của nó hữu dụng cho từng đối tượng cá nhân. Máy tính cá nhân bao gồm các loại như:

Máy tính để bàn (Desktop): máy vi tính để bàn hay máy tính cố định là một máy tính cá nhân được thiết kế để sử dụng thường xuyên tại một vị trí duy nhất trên bàn do kích thước và yêu cầu về điện năng tiêu thụ. Cấu hình thường gặp là vỏ máy chứa nguồn máy, bo mạch chủ (một mạch in với một bộ vi xử lý làm chức năng đơn vị xử lý trung tâm(CPU), bộ nhớ, bus, và các linh kiện điện tử khác); đĩa lưu trữ; bàn phím, chuột làm đầu vào; và màn hình máy tính, loa, máy in làm đầu ra.

Máy tính xách tay: (laptop computer hay laptop PC) là máy tính cá nhân nhỏ gọn có thể mang xách được. Nó thường có trọng lượng nhẹ, tùy thuộc vào hãng sản xuất và kiểu máy dành cho mỗi đối tượng có mục đích sử dụng khác nhau.

Laptop thường có một màn hình LCD hoặc LED mỏng gắn bên trong nắp trên vỏ máy và bàn phím chữ kết hợp số ở bên trong nắp dưới vỏ máy. Laptop khi không dùng đến sẽ được gấp lại, và do đó nó thích hợp cho việc sử dụng khi di chuyển. Một Laptop tiêu chuẩn kết hợp các thành phần, đầu vào (Input), đầu ra (Output) và các thành phần cơ bản của máy tính để bàn, bao gồm màn hình máy tính, loa nhỏ, một bàn phím, thiết bị chuột. Ổ đĩa cứng, ổ đĩa quang, một bộ xử lý, và bộ nhớ máy tính được kết hợp thành một khối. Hầu hết các laptop đều có webcam và microphone sẵn, một số máy tính laptop khác có màn hình cảm ứng. Laptop có thể lấy nguồn từ pin có sẵn bên trong và được sạc lại hay cấp nguồn trực tiếp từ nguồn điện bên ngoài thông qua AC adapter. Các chi tiết phần cứng, chẳng hạn như tốc độ xử lý và dung lượng bộ nhớ, khác nhau theo từng cấu hình Laptop, dòng máy tính, nhà sản xuất và mức giá. [1]

Máy tính bảng: (Tablet computer/tablet PC) có khả năng thực hiện các công việc như máy tính cá nhân (hỗ trợ đầu vào đầu ra hạn chế hơn máy tính cá nhân). Máy tính bảng phần lớn giống với điện thoại thông minh, điểm khác biệt duy nhất là máy tính bảng tương đối lớn hơn điện thoại thông minh, và có thể không hỗ trợ truy cập đến một mạng di động.

Màn hình cảm ứng được vận hành bằng cử chỉ được thực hiện bằng ngón tay hoặc bút kỹ thuật số (bút stylus), thay vì chuột, bàn di chuột và bàn phím của các máy tính lớn hơn. Năm 2010, Apple đã phát hành iPad, máy tính bảng đại chúng đầu tiên đạt được sự phổ biến rộng rãi. Sau đó, máy tính bảng nhanh chóng tăng lên ở khắp mọi nơi và sớm trở thành một loại sản phẩm lớn được sử dụng cho các ứng dụng cá nhân, giáo dục và nơi làm việc...

1.1.2.2 Theo mức cải tiến công nghệ

Một cách phân loại máy tính khác: theo mức độ hoàn thiện của công nghệ. Những chiếc máy tính có mặt sớm nhất thuần túy là máy cơ khí. Trong thập niên 1930, các thành phần rơ - le cơ điện đã được giới thiệu vào máy tính từ ngành công nghiệp liên lạc viễn thông. Trong thập niên 1940, những chiếc máy tính thuần túy điện tử đã được chế tạo từ những đèn điện tử chân không. Trong hai thập niên 1950 và thập niên 1960, bóng điện tử dần được thay thế bởi transistor, và từ cuối thập niên 1960 đầu thập niên 1970 là bởi mạch tích hợp bán dẫn (chíp bán dẫn, hay IC) cho đến hiện nay.

Một hướng nghiên cứu phát triển gần đây là máy tính quang (optical computer) trong đó máy tính hoạt động theo nguyên lý của ánh sáng hơn là theo nguyên lý của các dòng điện; đồng thời, khả năng sử dụng DNA trong công nghệ máy tính cũng đang được thử nghiệm. Một nhánh khác của việc nghiên cứu có thể dẫn công nghiệp máy tính tới những khả năng mới như tính toán lượng tử, tuy rằng nó vẫn còn ở giai đoạn đầu của việc nghiên cứu.

1.1.2.3 Theo đặc trưng thiết kế

Kỹ thuật số và kỹ thuật tương tự

Một quyết định nền tảng trong việc thiết kế máy tính là hoặc sử dụng kỹ thuật số (digital) hoặc sử dụng kỹ thuật tương tự (analog). Các máy tính kỹ thuật số (digital computer) tính toán trên các giá trị số rời rạc (discreet value) hoặc giá trị tượng trưng (symbolic value), trong khi đó máy tính tương tự (analog computer) tính toán trên các tín hiệu dữ liệu liên tục (continuous data signal). Bắt đầu từ thập niên 1940, máy tính kỹ thuật số đã trở nên phổ biến hơn mặc dù máy tính tương tự vẫn được sử dụng cho một số mục đích đặc biệt như trong kỹ thuật robot. Các thiết kế khác dùng tính toán xung lượng và tính toán lượng tử cũng hiện hữu nhưng chúng hoặc được sử dụng cho các mục đích đặc biệt hoặc vẫn đang trong vòng thử nghiệm.

Nhị phân và Thập phân

Một phát triển quan trọng trong thiết kế tính toán kỹ thuật số là việc sử dụng hệ nhị phân như là hệ thống số đếm nội tại. Điều này đã bãi bỏ những yêu cầu cần thiết trong các

cơ cấu kỹ thuật phức tạp của các máy tính sử dụng hệ số đếm khác, chẳng hạn như hệ thập phân. Việc áp dụng hệ nhị phân đã làm cho việc thiết kế trở lên đơn giản hơn để thực hiện các phép tính số học và các phép tính logic.

Khả năng lập trình

Khả năng lập trình của máy tính (programmability), cung cấp một tập hợp các chỉ thị để thực hiện mà không có sự điều khiển vật lý, là một đặc trưng thiết kế nền tảng của phần lớn các máy tính. Đặc trưng này là sự mở rộng đáng kể khi các máy tính đã được phát triển đến mức nó có thể kiểm soát việc thực hiện của chương trình. Điều này cho phép máy tính kiểm soát được thứ tự trong sự thực thi các lệnh trong chương trình dựa trên các dữ liệu đã được tính ra.

Điểm nổi bật chính trong thiết kế này đó là nó đã được đơn giản hóa một cách đáng kể với việc áp dụng các phép tính số học theo hệ đếm nhị phân để có thể mô tả hàng loạt các phép tính logic.

Khả năng lưu trữ

Trong quá trình tính toán, máy tính thông thường cần phải lưu trữ các giá trị trung gian để có thể sử dụng trong các tính toán sau đó. Khả năng thực hiện của máy tính phần lớn phụ thuộc vào tốc độ đọc các giá trị từ bộ nhớ và tốc độ ghi vào bộ nhớ, cũng như dung lượng bộ nhớ. Ban đầu bộ nhớ chỉ được sử dụng cho các giá trị trung gian, nhưng từ thập niên 1940 chương trình có thể được lưu trữ theo cách này.

1.1.2.4 Theo năng lực sử dụng

Máy tính để bàn

Máy tính để bàn hay máy tính cố định là một máy tính cá nhân được thiết kế để sử dụng thường xuyên tại một vị trí duy nhất trên bàn do kích thước và yêu cầu về điện năng tiêu thụ. Máy tính để bàn có từ hệ thống thấp nhất với giá dưới 1000$ đến cao nhất là những trạm làm việc cấu hình lớn giá trên 1000$. Kết hợp hiệu năng (hiệu năng tính toán, hiệu năng đồ hoạ) với giá cả của một hệ thống là vấn đề lớn nhất đối với khách hàng cũng như đối với nhà thiết kế máy tính. Một hệ thống máy để bàn hiệu quả là ở bộ vi xử lý hiệu năng cao nhất, mới nhất cũng như bộ vi xử lý giá hạ gần nhất và hệ thống xuất hiện đầu tiên.

Server

Nhiệm vụ của server là cung cấp sự phát triển của các dịch vụ tính toán và sắp xếp đáng tin cậy với quy mô lớn. Sự phát triển chóng mặt của W.W.W tạo xu hướng tăng trưởng rất lớn nhu cầu về server web và độ tinh vi của các dịch vụ trên cơ sở web. Servers trở thành xương sống trong các xí nghiệp quy mô lớn thay cho máy tính lớn truyền thống. Đối với server, các đặc tính khác nhau là rất quan trọng. Đầu tiên là tính sẵn dùng, hệ thống này có thể cung cấp dịch vụ một cách đáng tin cậy và hiệu quả. Một phần nào đó của các hệ thống quy mô lớn không thể tránh khỏi hỏng hóc; thách thức đối với một server là duy trì tính sẵn dùng dù cho các thành phần có thiếu khả năng bằng cách sử dụng cấu hình dư.

Đặc trưng chìa khoá thứ hai là khả năng co dãn. Khả năng dãn rộng của dung lượng tính toán, bộ nhớ, lưu trữ, băng thông vào ra của một dịch vụ là cốt yếu. Sau cùng, các server được thiết kế sao cho thông lượng có hiệu quả. Hiệu năng tổng thể của một server trong điều kiện số tương tác/phút hoặc số trang web/giây - là cốt yếu. Đáp ứng yêu cầu của một cá nhân vẫn là quan trọng, nhưng hiệu quả tổng thể và giá trị hiệu quả - xác định bởi bao nhiêu yêu cầu được xử lý trong một đơn vị thời gian - là thước đo chìa khoá cho hầu hết mọi server.

Các máy tính nhúng

Để chỉ những máy tính được đặt vào trong các thiết bị khác nơi mà sự hiện diện của máy tính này không rõ ràng một cách trực tiếp, là phần phát triển nhanh nhất của thị trường máy tính. Vùng ứng dụng của các thiết bị đó tạo từ các vi xử lý nhúng đơn xuất hiện trong các máy móc thường ngày (phần lớn là các lò viba, máy giặt, máy in, chuyển mạch mạng, ôtô..)

Các máy nhúng có phạm vi rất rộng về khả năng xử lý và giá cả. Từ mức thấp là bộ xử lý 8bit và 16bit có giá dưới 1$ tới mức cao nhất là bộ xử lý 32bit có khả năng thực hiện 50 triệu lệnh 1 giây có giá dưới 10$ và mức cao nhất có thể thực hiện một tỷ lệnh một giây có giá hàng trăm $. Thông thường, hiệu năng yêu cầu trong một ứng dụng nhúng là đòi hỏi trong thời gian thực. Một đòi hỏi hiệu năng thời gian thực là cho phép một đoạn của ứng dụng có thời gian thực hiện chính xác tuyệt đối. Hiệu năng thời gian thực phục vụ các ứng dụng phụ thuộc cao cùng với sự phát triển trong việc sử dụng các bộ vi xử lý nhúng, có đòi hỏi cho phép đo lường tiêu chuẩn có phạm vi rộng, từ khả năng chạy những đoạn mã giới hạn nhỏ đến khả năng thực hiện tốt các ứng dụng gồm hàng chục đến hàng trăm ngàn dòng mã.

Xu hướng quan trọng khác trong hệ thống nhúng là việc sử dụng bộ sử lý hạt nhân cùng với máy ứng dụng đặc biệt. Trong nhiều trường hợp các chức năng của ứng dụng và các yêu cầu về hiệu năng được thoả mãn bởi việc kết hợp một giải pháp phần cứng đặt hàng với phần mềm chạy trên bộ xử lý (lõi) nhúng được tiêu chuẩn hoá, được thiết kế giao diện với phần cứng có hiệu quả đặc biệt. Trong thực tế, vấn đề nhúng thường được giải quyết bằng 3 phương pháp:

1. Sử dụng kết hợp giải pháp phần cứng/phần mềm gồm một vài phần cứng đặt hàng và bộ xử lý nhúng tiêu chuẩn.

2. Sử dụng phần mềm đặt hàng chạy trên một bộ xử lý nhúng sẵn dùng

3. Sử dụng một bộ xử lý tín hiệu số và phần mềm đặt hàng. (Bộ xử lý tín hiệu số là bộ xử lý đặc biệt dành cho các ứng dụng xử lý tín hiệu)

1.2 Kiến trúc máy tính

Kiến trúc máy tính (Computer architecture) là một khái niệm trừu tượng của một hệ thống tính toán dưới quan điểm của người lập trình hoặc người viết chương trình dịch. Nói cách khác, kiến trúc máy tính được xem xét theo khía cạnh mà người lập trình có thể

can thiệp vào mọi mức đặc quyền, bao gồm các thanh ghi, ô nhớ, các ngắt ... có thể được thâm nhập thông qua các lệnh. Kiến trúc máy tính là những thuộc tính ảnh hưởng trực tiếp đến quá trình thực hiện logic của chương trình. Bao gồm: tập lệnh, biểu diễn dữ liệu, các cơ chế vào ra, kỹ thuật đánh địa chỉ,…

Tổ chức máy tính có thể hiểu là các khối chức năng trong máy tính và sự kết nối giữa chúng để thực hiện các đặc tính của kiến trúc. Ví dụ, các thuộc tính của kiến trúc bao gồm tập lệnh, số bit để biểu diễn các kiểu dữ liệu khác nhau (ví dụ: ký tự, số,…), cơ chế vào/ra (I/O) và cách kỹ thuật định địa chỉ bộ nhớ. Các thuộc tính của tổ chức bao gồm các đặc điểm phần cứng như: các tín hiệu điều khiển, giao diện giữa máy tính và các thiết bị ngoại vi và các công nghệ bộ nhớ được sử dụng. Ví dụ, về mặt thiết kế kiến trúc máy tính: ta muốn thực hiện lệnh Nhân. Vậy, về mặt tổ chức sẽ như sau: lệnh này sẽ được thực hiện bởi một đơn vị phần cứng đặc biệt hoặc một cơ chế cho phép thực hiện lặp đi lặp lại phép cộng. Việc lựa chọn một trong hai phương án trên phụ thuộc vào tần suất dự kiến lệnh nhân được sử dụng, tương quan tốc độ tính toán của hai phương pháp, chi phí và kích thước vật lý mỗi một phương án tổ chức.

Hiện nay, sự khác nhau giữa kiến trúc và tổ chức vẫn còn khá rõ nét. Nhiều nhà sản xuất máy tính đưa ra một họ các model máy tính, tất cả chúng đều có kiến trúc tương tự nhau nhưng lại khác nhau về tổ chức. Các model có giá cả và hiệu suất khác nhau. Một kiến trúc đặc biệt có thể tồn tại trong nhiều năm và với nhiều model máy tính khác nhau nhưng tổ chức của nó thay đổi theo sự thay đổi của công nghệ.

Với một lớp máy tính khác gọi là máy vi tính, mối quan hệ giữa kiến trúc và tổ chức rất chặt chẽ. Những thay đổi trong công nghệ không chỉ ảnh hưởng đến tổ chức mà còn dẫn đến việc các kiến trúc máy tính mạnh hơn và phức tạp hơn. Tuy nhiên, yêu cầu tương thích giữa các thế hệ máy tính này khá thấp vì vậy việc thiết kế tổ chức và kiến trúc máy tính có nhiều tương tác hơn. [1]

1.2.1 Kiến trúc tập lệnh

Thuật ngữ kiến trúc tập lệnh dùng để chỉ tập lệnh mà người lập trình có thể nhìn thấy thực sự. Kiến trúc tập lệnh là cấu trúc của một máy tính mà người lập trình ngôn ngữ máy phải hiểu để viết một chương trình chuẩn cho máy đó. Ngôn ngữ duy nhất mà một máy tính nhận dạng được khi chạy là ngôn ngữ máy của nó hoặc kiến trúc tập lệnh. Kiến trúc tập lệnh cũng mô tả máy mà một người thiết kế phần cứng cần phải hiểu để thiết kế ra được một sản phẩm máy tính chuẩn.

Kiến trúc tập lệnh phục vụ như một giao dịch giữa phần cứng và phần mềm. Việc thực hiện trong máy tính gồm có hai phần: tổ chức và phần cứng. Thuật ngữ tổ chức gồm các bộ phận mức cao của một thiết kế máy tính như: hệ thống bộ nhớ, cấu trúc bus và thiết kế CPU (bộ xử lý trung tâm nơi thực hiện các phép tính số học, logic, rẽ nhánh và truyền dự liệu). Phần cứng dùng để chỉ những đặc trưng của máy tính bao gồm thiết kế logic chi tiết và công nghệ đóng gói máy. Thông thường một dòng máy tính gồm các máy có cùng kiến trúc tập lệnh và tổ chức gần giống nhau nhưng chúng khác nhau về thực hiện phần

cứng chi tiết. Ví dụ Pentium và Celeron gần giống hệt nhau nhưng tốc độ đồng hồ khác nhau và hệ thống bộ nhớ khác nhau làm cho Celeron hiệu qủa hơn đối với những máy tính loại thấp.

Nhà thiết kế cần phải thiết kế một máy tính thoả mãn những đòi hỏi như những mục tiêu về năng lượng và hiệu năng. Thông thường, họ cũng phải xác định những đòi hỏi đó và đó có thể là nhiệm vụ chủ yếu. Những đòi hỏi đó có thể là những điểm đặc trưng cụ thể qua tác động của thị trường. Phần mềm ứng dụng thường dẫn đến sự lựa chọn là một vài yêu cầu bằng việc xác định máy tính sẽ được sử dụng thế nào. Nếu có một khối lượng lớn phần mềm cho một kiến trúc tập lệnh nào đó, nhà thiết kế có thể quyết định một cái máy tính mới cần thực hiện một tập lệnh hiện có.

Một kiến trúc tập lệnh thành công cần phải được thiết kế để tồn tại qua những thay đổi nhanh chóng của công nghệ máy tính. Sau tất cả, kiến trúc tập lệnh thành công trong mấy thập kỉ vừa qua, bộ lõi của máy tính lớn IBM đã được sử dụng hơn ba mươi năm qua. Nhà thiết kế cần định kế hoạch cho sự thay đổi công nghệ. Để định kế hoạch cho cuộc cách mạng về máy tính, nhà thiết kế cần nhất là nhận rõ sự thay đổi nhanh chóng trong công nghệ thực hiện. Có bốn công nghệ thực hiện, thay đổi với tốc độ đáng kinh ngạc, giới hạn trong những công nghệ hiện đại:

- Công nghệ mạch tích hợp logic: mật độ Transitor tăng khoảng 35% mỗi năm, hiệu quả kết hợp là tốc độ tăng trưởng trong tổng số Transitor trên một chip là 55% mỗi năm.
- Công nghệ đĩa từ: gần đây, mật độ đĩa được cải tiến hơn 100% mỗi năm, gấp bốn lần trong hai năm. Trước 1990, mật độ tăng khoảng 30% mỗi năm, gấp hai lần trong ba năm. Tốc độ tăng mật độ sẽ nhanh hơn trong thời gian tới. Thời gian truy nhập được cải thiện 1/3 trong mười năm.
- Công nghệ mạng: hiện nay mạng phụ thuộc vào hiệu năng chuyển mạch và hiệu năng của hệ thống truyền dẫn, cả độ trễ và băng thông có thể được cải thiện, gần đây băng thông được chú ý nhiều hơn. Cơ sở hạ tầng Internet ở Mỹ phát triển nhanh hơn (băng thông gấp đôi mỗi năm) nhờ sử dụng cáp quang và sự triển khai nhiều phần cứng chuyển mạch.

Các công nghệ thay đổi nhanh chóng này tác động đến việc thiết kế một bộ vi xử lý, với tốc độ và công nghệ nâng cao, có thể tồn tại năm năm hoặc nhiều hơn. Thực tế, tổng số Transistor cải thiện theo bậc hai còn hiệu năng cải thiện theo tuyến tính vừa là thách thức vừa là cơ hội cho kiến trúc máy tính. Trong thời gian gần đây, mật độ được cải tiến nhanh, các bộ xi xử lý chuyển từ 4 bit tới 8 bit, 16 bit, 32 bit và gần đây là bộ xử lý 64 bit có nhiều cách tân công nghệ Pipeline và Cache.

Kiến trúc CISC tập lệnh lớn và có nhiều lệnh phức tạp. Định hướng thiết kế CISC: xuất hiện ra từ những năm 1960, các chương trình dịch khó dùng các thanh ghi, các vi lệnh được thực hiện nhanh hơn các lệnh và cần thiết giảm độ dài các chương trình. 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 nên các lệnh có chiều dài thay đổi. Mỗi lệnh có thể thực hiện nhiều chức năng. Ưu

điểm của kiến trúc CISC là chương trình ngắn hơn so với kiến trúc RISC. Số lệnh để thực hiện chương trình ít hơn. Khả năng thâm nhập bộ nhớ dễ dàng hơn. Các bộ xử lý CISC trợ giúp mạnh hơn các ngôn ngữ cao cấp nhờ có tập lệnh phức tạp. Tuy nhiên kiến trúc CISC cũng có nhược điểm là diện tích của bộ xử lý dùng cho bộ điều khiển lớn. Giảm khả năng tích hợp thêm vào vi xử lý. Tốc độ tính toán còn chậm. Thời gian xây dựng xong bộ vi xử lý lâu hơn do các câu lệnh phức tạp nên khả năng xảy ra rủi ro nhiều.

Kiến trúc RISC là một phương pháp thiết kế các bộ vi xử lý theo hướng đơn giản hóa tập lệnh, trong đó thời gian thực thi tất cả các lệnh đều như nhau. Định hướng thiết kế vi xử lý RISC: tổ chức lại quá trình thực thi trong vi xử lý nhằm giảm bớt số lần truy xuất bộ nhớ, quá trình thực thi sẽ nhanh hơn. Cắt giảm bộ vi xử lý chỉ còn lại bộ phận thiết yếu, không lãng phí tài nguyên. Các chức năng thích hợp thực hiện bằng phần mềm hơn là bằng phần cứng. Ưu điểm của kiến trúc RISC là tốc độ xử lí cao hơn CISC. Chi phí thiết kế vi xử lý RISC giảm. Độ tin cậy cao và hỗ trợ ngôn ngữ bậc cao. Nhược điểm của kiến trúc RISC là các chương trình dài hơn so với chương trình CISC. Cần thiết phải tính các địa chỉ hiệu dụng vì không có nhiều cách định vị. Có ít lệnh trợ giúp cho ngôn ngữ cấp cao. Cấm thâm nhập bộ nhớ đối với tất cả các lệnh ngoại trừ các lệnh đọc và ghi vào bộ nhớ nên cần phải dùng nhiều lệnh để làm một công việc nhất định. [2]

1.2.2 Mô hình máy tính cơ bản

Máy tính bao gồm các khối chức năng chính sau:

1) Bộ nhớ chính (*Main Memory*): là nơi lưu giữ chương trình và dữ liệu trước khi chương trình được thực hiện. Đây là một tập hợp các ô nhớ, mỗi ô nhớ có một số bit nhất định và chứa một thông tin được mã hoá thành số nhị phân, không quan tâm đến kiểu của dữ liệu mà nó đang chứa. Các thông tin này là các lệnh hay số liệu. Bộ nhớ chính bao gồm hai loại: RAM và ROM.

Bộ nhớ truy cập ngẫu nhiên (RAM - Random Access Memory): mỗi ô nhớ của bộ nhớ trong đều có một địa chỉ để CPU có thể định vị khi cần đọc hay ghi dữ liệu. Thời gian thâm nhập vào một ô nhớ bất kỳ trong bộ nhớ là như nhau. 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). Tốc độ truy cập bộ nhớ nhanh, lưu trữ thông tin được sử dụng hiện hành. Thông tin lưu trữ trên RAM là tạm thời, chúng sẽ mất khi không còn nguồn điện cung cấp.

Bộ nhớ chỉ đọc (ROM - Read Only Memory): là vùng bộ nhớ chỉ đọc, thông tin không bị mất đi khi không còn nguồn điện cung cấp. Nội dung của bộ nhớ ROM thông thường là lưu trữ các chương trình hệ thống và được cài đặt tại nơi sản xuất thiết bị. Ngày nay còn có công nghệ FlashROM có nghĩa là bộ nhớ ROM không chỉ đọc mà còn ghi lại được. Nhờ có công nghệ này BIOS được cải tiến thành FlashBIOS.

2) Bộ xử lý trung tâm (CPU - Central Processing Unit): có thể coi là não bộ, một trong những phần tử cốt lõi nhất của máy tính. CPU có nhiệm vụ điều khiển mọi hoạt động của máy tính, xử lý dữ liệu, thi hành lệnh, hoạt động theo chương trình nằm trong bộ nhớ

chính. CPU lấy lệnh từ bộ nhớ trong và lấy các số liệu mà lệnh đó xử lý. Cấu trúc cơ bản của bộ xử lý trung tâm CPU gồm các thành phần chức năng:

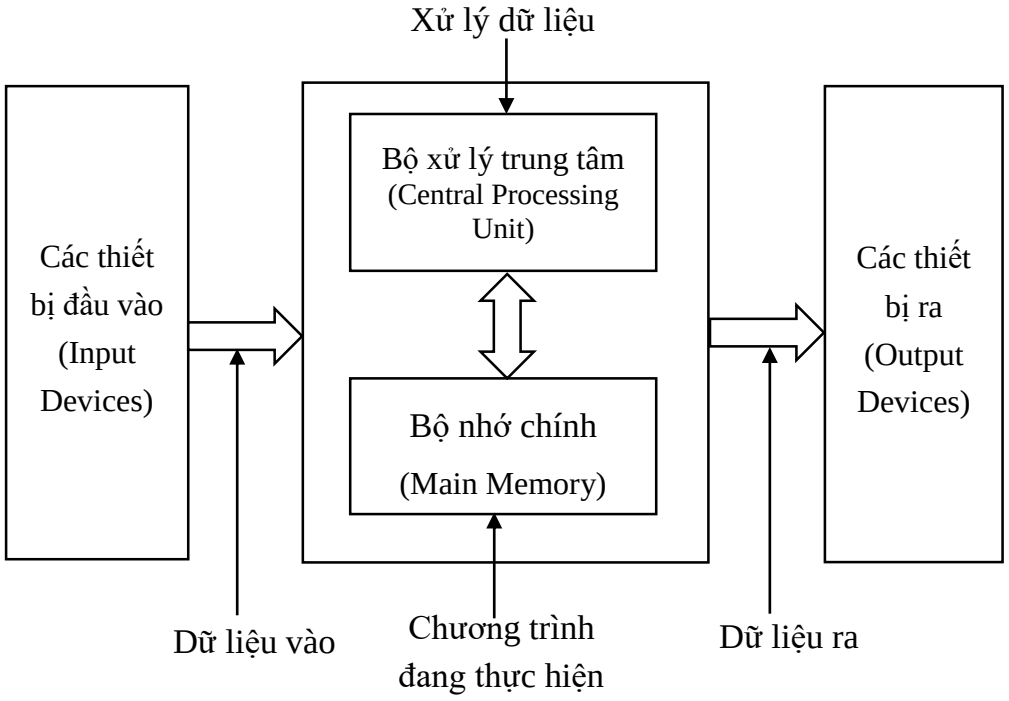
Đơn vị điều khiển (CU - Control Unit): điều khiển hoạt động của máy tính theo chương trình định sẵn. Đơn vị điều khiển CU có chức năng lấy lệnh tuần tự được lưu trữ trong bộ nhớ, giải mã lệnh, tạo các tín hiệu điều khiển hoạt động của các khối chức năng bên trong và bên ngoài CPU. Lệnh đọc từ ô nhớ sẽ được đưa vào thanh ghi lệnh IR, được giải mã tại khối giải mã lệnh ID để xác định công việc CPU cần thực hiện.

Đơn vị số học và Logic (ALU - Arithmetic & Logic Unit): thực hiện các thao tác xử lý dữ liệu thông qua các phép toán số học và Logic trên dữ liệu cụ thể, theo sự điều khiển của đơn vị điều khiển CU.

Tập các thanh ghi (Registers): lưu giữ các thông tin tạm thời phục vụ cho hoạt động của CPU.

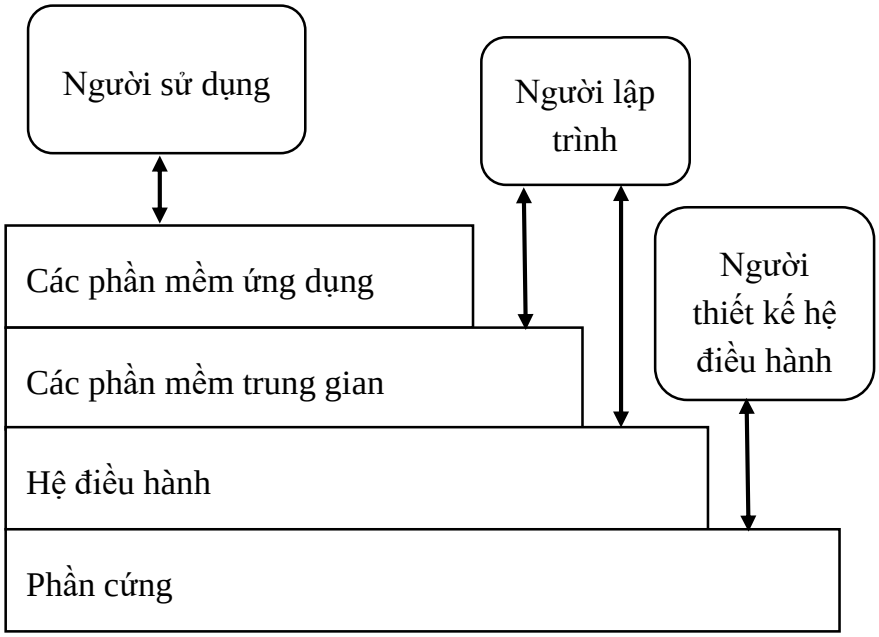
3) Thiết bị vào (Input Device): thực hiện nhiệm vụ thu nhận các thông tin, dữ liệu từ thế giới bên ngoài, biến đổi thành dạng tương thích với phương thức biểu diễn trong máy tính, đưa vào CPU xử lý hoặc ghi vào bộ nhớ. Ví dụ: bàn phím là thiết bị nhập chuẩn. Ngoài ra còn có các thiết bị nhập khác như chuột, scanner, thiết bị đọc mã vạch,…

4) Thiết bị ra (Output Device): thực hiện nhiệm vụ đưa thông tin, dữ liệu từ CPU hoặc bộ nhớ ra ngoài dưới các dạng thức được người sử dụng yêu cầu. Ví dụ: màn hình là thiết bị xuất dữ liệu chuẩn. Ngoài ra còn có các thiết bị xuất khác như: máy in, máy chiếu,…



Hình 1.1 Mô hình máy tính cơ bản

1.2.3 Mô hình phân lớp máy tính



Hình 1.2 Mô hình phân lớp của máy tính

Trong mô hình phân lớp máy tính, người sử dụng ở lớp trên cùng - là lớp các phần mềm ứng dụng. Các phần mềm ứng dụng được thiết kế theo ngôn ngữ bậc cao, người sử dụng chỉ cần cài đặt trong máy tính, biết cách sử dụng là có thể khai thác được các tính năng của phần mềm ứng dụng.

Người lập trình có vai trò quan trọng trong lớp các phần mềm trung gian và lớp hệ điều hành. Trong lớp các phần mềm trung gian, bao gồm các chương trình dịch, dịch mã ngôn ngữ bậc cao thành ngôn ngữ máy. Lớp hệ điều hành có chức năng lập lịch cho các tiến trình, chia sẻ tài nguyên, quản lý bộ nhớ, lưu trữ và quản lý vào ra.

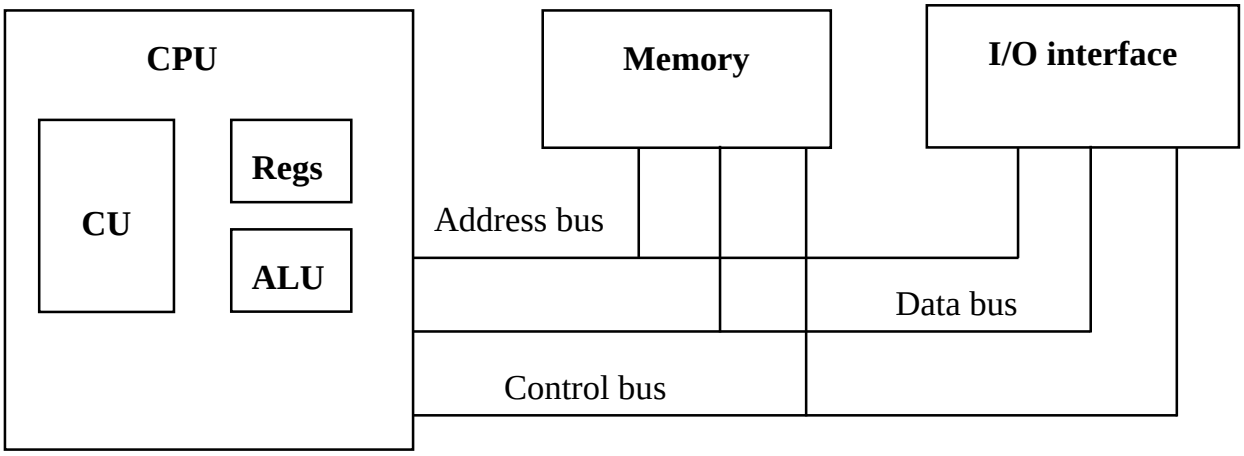
Người thiết kế hệ thống có vai trò quan trọng trong lớp phần cứng, họ tạo ra khung xương cho hệ thống máy tính. Phần cứng bao gồm các linh kiện điện tử, cơ khí trong hệ thống máy tính, bộ xử lý, bộ nhớ, mô đun vào ra,…[4]

1.2.4 Sơ đồ kiến trúc máy tính Von Neumann

John von Neumann (Neumann János, 28 tháng 12, 1903 - 8 tháng 2, 1957) là một nhà toán học người Hungary và là một nhà bác học thông thạo nhiều lĩnh vực, đã có nhiều đóng góp vào các chuyên ngành vật lý lượng tử, giải tích hàm, lý thuyết tập hợp, kinh tế, khoa học máy tính, giải tích số, thủy động lực học, thống kê và nhiều lĩnh vực toán học khác. Đáng chú ý nhất, von Neumann là nhà tiên phong của *máy tính kỹ thuật số* hiện đại và áp dụng lý thuyết toán tử (*operator theory*) vào cơ học lượng tử.

Năm 1945, ông đã đưa ra một đề nghị về kiến trúc máy tính như sau: lệnh (Instruction) và dữ liệu (Data) phải được lưu giữ trong một bộ nhớ ghi/đọc được. Từng ô nhớ trong bộ nhớ phải được định vị bằng địa chỉ. Sự định địa chỉ là tuần tự và không phụ

thuộc vào nội dung của từng ô nhớ. Chương trình xử lý, giải bài toán phải thực hiện tuần tự từ lệnh này đến lệnh tiếp theo, từ lệnh bắt đầu đến lệnh cuối cùng.



Hình 1.3 Sơ đồ kiến trúc máy tính Von Neumann

Nguyên lý làm việc của hệ thống máy tính theo kiến trúc Von Neumann: các lệnh và dữ liệu đều là mã nhị phân được lưu trong bộ nhớ và được nhập vào CPU qua bus số liệu. Chương trình là một tập hợp các lệnh được mã hóa thành các bit 0, 1 và được sắp xếp theo một trật tự nhất định, CPU sẽ thực hiện các lệnh này một cách tuần tự nối tiếp nhau.

Kiến trúc máy tính Von Neumann bao gồm các thành phần:

Bộ xử lý trung tâm CPU - central processing unit: đơn vị quan trọng nhất của máy tính. Nó điều khiển mọi hoạt động của máy tính, điều khiển các quá trình nạp và xử lý dữ liệu. Cấu tạo: CPU bao gồm các khối chính như sau:

Khối điều khiển CU - control unit: thực hiện vai trò kiểm soát toàn bộ các bộ phận khác trong máy tính. Đồng thời nó làm nhiệm vụ phiên dịch các lệnh thành các tín hiệu hoặc thành các vi lệnh nhỏ hơn để gửi tới các bộ phận khác trong máy tính.

Khối tính toán số học và logic ALU - Arithmetic & Logic Unit: sử dụng để thực hiện các phép toán +,-,*,/…

Các thanh ghi - Registers: đây là các ô nhớ đặc biệt có tốc độ trao đổi dữ liệu rất nhanh -từ 3 đến 10 ns - thường dùng làm nhiệm vụ lưu các kết quả trung gian. Ví dụ: Lưu trữ lệnh, dữ liệu trước khi CPU xử lý, hoặc có thể lưu trữ các kết quả trung gian khi ALU thực hiện tinh toán.

Bộ nhớ: là nơi chứa các chương trình hay dữ liệu, được tổ chức từ nhiều ô nhớ hợp thành. Mỗi ô nhớ được gán một địa chỉ nhất định để CPU có thể quản lý và truy cập dữ liệu. Bộ nhớ chính chia làm hai loại ROM và RAM. ROM là bộ nhớ chỉ đọc tốc độ truy cập dữ liệu chậm hơn RAM, RAM là bộ nhớ truy cập ngẫu nhiên cho phép đọc ghi dữ liệu, khi bị mất nguồn cung cấp điện thì dữ liệu không bị mất.

Giao diện vào ra - I/O interface: cho phép CPU, bộ nhớ chính có thể kết nối với các thiết bị bên ngoài. Mỗi thiết bị ngoại vi cũng được gán một địa chỉ nhất định để CPU

trao đổi DL. Để phân biệt với các địa chỉ bộ nhớ người ta thường gọi địa chỉ của các thiết bị ngoại vi này là các cổng vào ra (port I/O).

Các loại bus: bus là một đường kết nối giữa hai hoặc nhiều thiết bị, là các kênh truyền cho phép các tín hiệu điều khiển, dữ liệu, địa chỉ có thể được di chuyển từ bộ phận này đến bộ phận khác. Đặc điểm chính của bus là một phương tiện truyền dẫn chia sẻ: nhiều thiết bị kết nối với bus, khi một thiết bị gửi tín hiệu trên bus, tất cả các thiết bị khác gắn vào bus đều nhận được. Nếu hai thiết bị cùng truyền dữ liệu cùng một thời điểm, các tín hiệu sẽ bị chồng chéo lên nhau và bị méo. Do đó, tại một thời điểm chỉ một thiết bị có thể truyền dữ liệu thành công. Thông thường, một bus gồm nhiều đường truyền. Mỗi đường có khả năng truyền các tín hiệu được biểu diễn dưới dạng số nhị phân 1 và 0. Các tín hiệu có thể được truyền nhiều đợt trên một dòng (truyền nối tiếp). Kết hợp nhiều đường trong bus ta có thể truyền nhiều số nhị phân tại cùng một thời điểm (truyền song song). Ví dụ, một khối 8 bit ta có thể truyền đồng thời trong 8 đường bus. Thông thường, một hệ thống máy tính sẽ có nhiều bus khác nhau cung cấp các đường liên kết thành phần ở cấp độ khác nhau của hệ thống máy tính. Bus kết nối các thành phần chính của máy tính (gồm có: bộ xử lý, bộ nhớ, các module vào/ra) được gọi là bus hệ thống. Các cấu trúc kết nối phổ biến nhất thường sử dụng một hoặc nhiều bus hệ thống. Hệ thống bus bao gồm bus địa chỉ (address bus), bus dữ liệu (data bus), bus điều khiển (control bus).

Bus dữ liệu: cho phép truyền dữ liệu theo hai chiều nhưng không đồng thời.

Bus địa chỉ: vận chuyển địa chỉ xác định ngăn nhớ hay cổng vào ra. Chỉ cho phép truyền một chiều, hoặc từ CPU đến bộ nhớ chính, hoặc từ CPU đến các thiết bị vào ra.

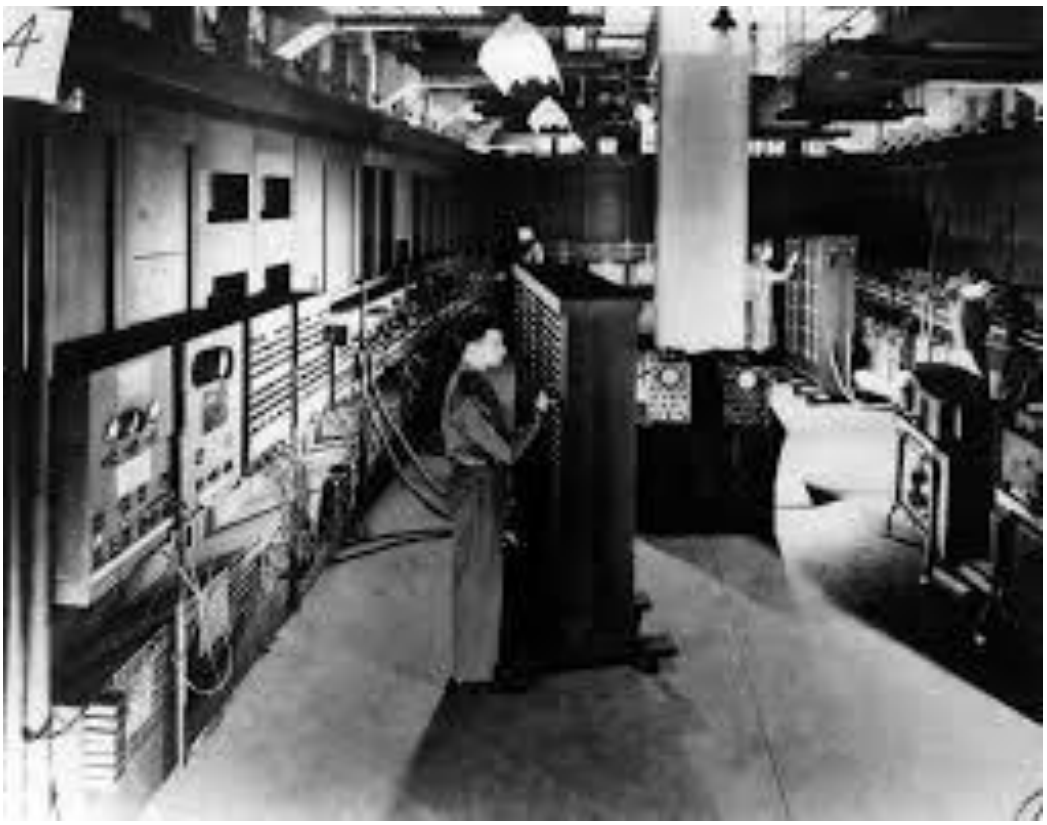
Bus điều khiển: Truyền tín hiệu điều khiển từ CPU đến bộ nhớ chính hoặc tới các giao diện vào ra.

1.3 Sự phát triển của máy tính

1.3.1 Thế hệ đầu tiên (1946 - 1957)

ENIAC là máy tính điện tử số đầu tiên do Giáo sư Mauchly và người học trò Eckert tại Đại học Pennsylvania thiết kế vào năm 1943 và được hoàn thành vào năm 1946. Đây là một máy tính khổng lồ với thể tích dài 20 mét, cao 2,8 mét và rộng vài mét. ENIAC bao gồm: 18.000 đèn điện tử, 1.500 công tắc tự động, cân nặng 30 tấn, và tiêu thụ 140KW giờ. Nó có 20 thanh ghi 10 bit (tính toán trên số thập phân). Có khả năng thực hiện 5.000 phép toán cộng trong một giây. Công việc lập trình bằng tay, đấu nối các đầu cắm điện và dùng các ngắt điện.

Giáo sư toán học John Von Neumann đã đưa ra ý tưởng thiết kế máy tính IAS: chương trình được lưu trong bộ nhớ, bộ điều khiển sẽ lấy lệnh và biến đổi giá trị của dữ liệu trong phần bộ nhớ, bộ làm toán và luận lý (ALU) được điều khiển để tính toán trên dữ liệu nhị phân, điều khiển hoạt động của các thiết bị vào ra. Đây là một ý tưởng nền tảng cho các máy tính hiện đại ngày nay. Máy tính này còn được gọi là máy tính Von Neumann.



Hình 1.4 Máy tính ENIAC (1946)

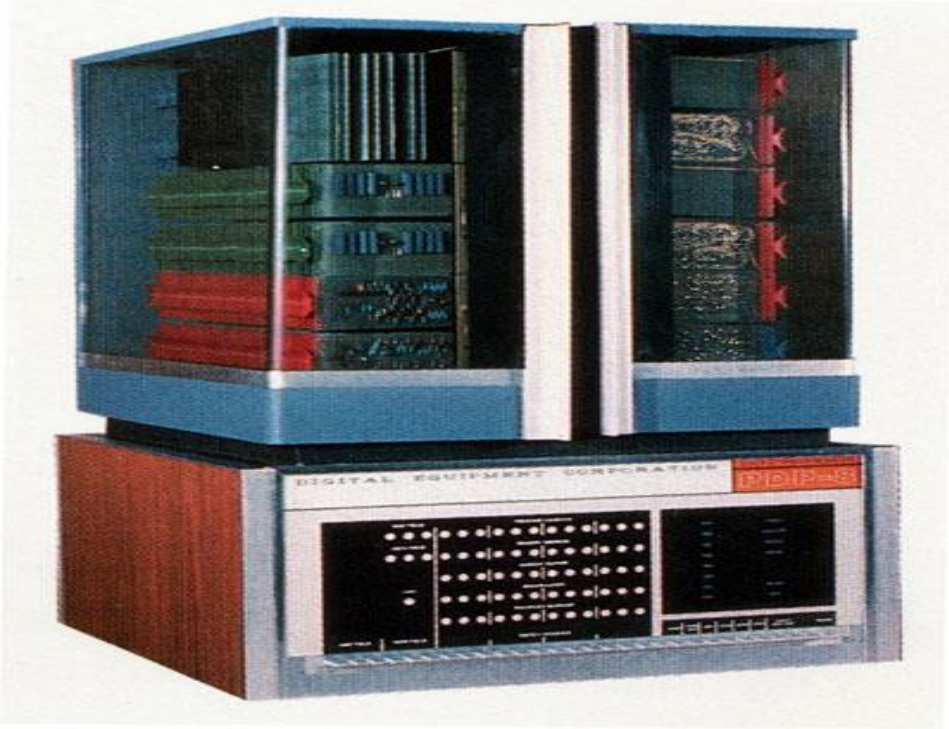
1.3.2 Thế hệ thứ hai (1958 - 1964)

Công ty Bell đã phát minh ra transistor vào năm 1947 và do đó thế hệ thứ hai của máy tính được đặc trưng bởi sự thay thế các đèn điện tử bằng các transistor lưỡng cực. Tuy nhiên, đến cuối thập niên 50, máy tính thương mại dùng transistor mới xuất hiện trên thị trường. Kích thước máy tính giảm, rẻ tiền hơn, tiêu tốn năng lượng ít hơn. Vào thời điểm này, mạch in và bộ nhớ bằng xuyến từ được dùng. Ngôn ngữ cấp cao xuất hiện (như FORTRAN năm 1956, COBOL năm 1959, ALGOL năm 1960) và hệ điều hành kiểu tuần tự (Batch Processing) được dùng. Trong hệ điều hành này, chương trình của người dùng thứ nhất được chạy, xong đến chương trình của người dùng thứ hai và cứ thế tiếp tục.

Hình 1.5 Máy tính DEC PDP-1

1.3.3 Thế hệ thứ ba (1965 - 1971)

Thế hệ thứ ba được đánh dấu bằng sự xuất hiện của các *mạch tích hợp* (IC). Các mạch tích hợp mật độ thấp (SSI: Small Scale Integration) có thể chứa vài chục linh kiện và mạch tích hợp mật độ trung bình (MSI: Medium Scale Integration) chứa hàng trăm linh kiện. Mạch in nhiều lớp xuất hiện, bộ nhớ bán dẫn bắt đầu thay thế bộ nhớ bằng xuyến từ. Máy tính đa chương trình và hệ điều hành chia thời gian được dùng.



Hình 1.6 Máy tính DEC PDP-8 (1965)

1.3.4 Thế hệ thứ tư (1972 - nay)

Thế hệ thứ tư được đánh dấu bằng các IC có mật độ tích hợp cao (LSI: Large Scale Integration) có thể chứa hàng ngàn linh kiện. Các IC mật độ tích hợp rất cao (VLSI: Very Large Scale Integration) có thể chứa hơn mười nghìn linh kiện trên mạch. Hiện nay, các chip VLSI chứa hàng triệu linh kiện. Với sự xuất hiện của bộ vi xử lý (microprocessor) chứa cả phần thực hiện và phần điều khiển của một bộ xử lý, sự phát triển của công nghệ bán dẫn các máy vi tính đã được chế tạo và khởi đầu cho các thế hệ máy tính cá nhân. Các bộ nhớ bán dẫn, bộ nhớ cache, bộ nhớ ảo được dùng rộng rãi. Các kỹ thuật cải tiến tốc độ xử lý của máy tính không ngừng được phát triển: kỹ thuật ống dẫn, kỹ thuật vô hướng,,…

Hình 1.7 Máy tính MITS Altair (1975)

Hình 1.8 Máy vi tính để bàn FPT eLead T7100 (2015)

Hình 1.9 Máy tính bảng iPad Air 2019

1.3.5 Khuynh hướng hiện tại

Việc chuyển từ thế hệ thứ tư sang thế hệ thứ năm 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 năm của máy tính, thế hệ của những máy tính thông minh, dựa trên các ngôn ngữ trí tuệ nhân tạo như LISP, 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. 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 năm là thế hệ các máy tính xử lý song song.

1.4 Hiệu năng máy tính

Làm sao để có thể đo đạc, đánh giá hiệu năng (performance) và định ra được những yếu tố quyết định đến hiệu năng của một máy tính? Lý do chính để khảo sát về hiệu năng là vì hiệu năng của phần cứng máy tính thường là yếu tố mấu chốt quyết định đến tính hiệu quả trong hoạt động của một một hệ thống bao gồm cả phần cứng lẫn phần mềm. Hiệu năng luôn là một thuộc tính quan trọng đối với các nhà thiết kế máy tính và trong việc lựa chọn, mua bán các máy tính mà được cả người bán lẫn người mua quan tâm.

Việc đánh giá hiệu năng máy tính không hề đơn giản. Hiệu năng không chỉ có được do các cải tiến phần cứng mà cũng có thể nhờ vào các phần mềm thông minh hay cả hai tùy góc độ ứng dụng khác nhau. Hiệu năng hoàn toàn có thể được đánh giá theo những phương cách, những chỉ số khác nhau. Ở góc độ nhà thiết kế máy tính (phần cứng/phần mềm), chúng ta cần nắm rõ các vấn đề liên quan đến việc đánh giá hiệu năng máy tính. Trong mỗi ứng dụng cụ thể, xác định phương pháp đánh giá hiệu năng phù hợp.

1.4.1 Hiệu năng máy tính P (Performance)

$$Hiệu\ năng = \frac{1}{thời\ gian\ thực\ hiện} = \frac{1}{t}$$

Ví dụ 1: Máy tính A nhanh hơn máy tính B k lần

$$\frac{P_A}{P_B} = \frac{t_B}{t_A} = k$$

Ví dụ 2: Thời gian chạy chương trình trên máy A và B tương ứng là 10s và 15s.

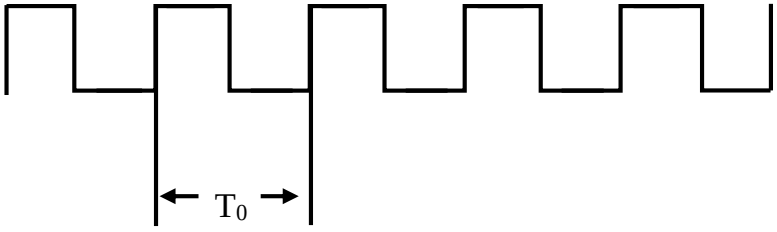
$$k = \frac{t_B}{t_A} = \frac{15s}{10s} = 1.5\ (lần)$$

Vậy máy A nhanh hơn máy B 1.5 lần.

Thời gian được sử dụng làm thước đo cho hiệu suất máy tính. Tuy nhiên thời gian ở đây được định nghĩa theo nhiều cách khác nhau, tùy theo mục đích đo đạc, bao gồm: thời gian theo đồng hồ, thời gian đáp ứng (*response time*), thời gian trôi qua (*elapsed time*). Các máy tính hoạt động theo nguyên lý chia thời gian (*timesharing*), bộ xử lý làm việc đồng thời cho nhiều chương trình. Thời gian thực thi chương trình bao gồm thời gian thực thi bởi CPU lẫn các thiết bị khác (bộ nhớ, đĩa cứng,...). Ở đây ta chỉ giới hạn xem xét đối với thời gian CPU để đo hiệu năng của máy tính.

1.4.2 Tốc độ xung nhịp của CPU

Về mặt thời gian, CPU hoạt động theo một xung nhịp (clock) có tốc độ xác định.



Hình 1.10 Chu kỳ xung nhịp của CPU

Chu kỳ xung nhịp T_0 (Clock period): là thời gian của một chu kỳ.

Tốc độ xung nhịp f_0 (Clock rate) hay còn gọi là tần số xung nhịp: là số chu kỳ trong một giây.

$$f_0 = \frac{1}{T_0}$$

Ví dụ: Bộ xử lý có $f_0 = 4GHz = 4 * 10^9 Hz$

$$T_0 = \frac{1}{4*10^9} = 0.25 * 10^{-9} s = 0.25\ ns$$

1.4.3 Thời gian thực hiện của CPU

Thời gian CPU thực hiện chương trình - CPUtime:

$$Thời\ gian\ thực\ hiện\ của\ CPU = số\ chu\ kỳ\ xung\ nhịp * thời\ gian\ một\ chu\ kỳ$$

$$t_{CPU} = n * T_0 = \frac{n}{f_0}$$

Trong đó n là số chu kỳ xung nhịp. Hiệu năng được tăng lên bằng cách giảm số chu kỳ xung nhịp n, tăng tốc độ xung nhịp f_0.

Ví dụ: hai máy tính A và B cùng chạy một chương trình. Máy tính A có: tốc độ xung nhịp của CPU f_A = 2GHz. Thời gian CPU thực hiện chương trình t_A = 10s. Máy tính B có: thời gian CPU thực hiện chương trình t_B = 6s. Số chu kỳ xung nhịp khi chạy chương trình trên máy B (n_B) nhiều hơn 1.2 lần số chu kỳ xung nhịp khi chạy chương trình trên máy A (n_B).

Hãy xác định tốc độ xung nhịp cần thiết cho máy B.

Bài giải:

$$t = \frac{n}{f}$$

Số chu kỳ xung nhịp khi chạychương trình trên máy A:

$$n_A = t_A * f_A = 10s * 2GHz = 20 * 10^9$$

Số chu kỳ xung nhịp khi chạychương trình trên máy B:

$$n_A = 1.2 * n_A = 24 * 10^9$$

Tốc độ xung nhịp cần thiết cho máy B:

$$f_B = \frac{n_B}{t_B} = \frac{24 * 10^9}{6} = 4 * 10^9 Hz = 4GHz$$

1.4.4 Số chu kỳ cần thiết để thực hiện một lệnh

CPI (Cycle Per Instruction):

$$CPI = \frac{\mathrm{ET} * \mathrm{CR}}{\mathrm{EI}}$$

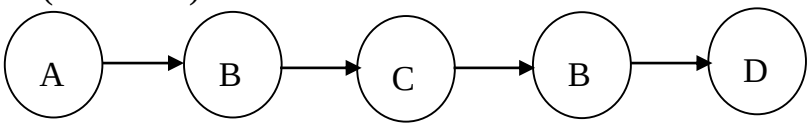
Trong đó:

EI: số lệnh được thực hiện (executed instructions)

ET: thời gian thực hiện lệnh (execution time)

CR: tần số (clock rate)

Ví dụ:



Giả sử: Thực hiện lệnh A có CPI = 2

B có CPI = 3

C có CPI = 1

D có CPI = 4

Biết 1 CPI = 2ns (nano giây) ⇒ làm hết lệnh trên cần 26s

1.4.5 Số lệnh và số chu kỳ trên một lệnh

Số chu kỳ xung nhịp của chương trình:

$$Số\ chu\ kỳ = số\ lệnh\ của\ chương\ trình * số\ chu\ kỳ\ trên\ một\ lệnh$$

$$n = IC * CPI$$

Trong đó:

n: số chu kỳ xung nhịp

IC: số lệnh của chương trình (Instruction Count)

CPI: số chu kỳ trên một lệnh (Cycles per Instruction)

Vậy thời gian thực hiện CPU

$$t_{CPU} = IC * CPI * T_0 = \frac{IC * CPI}{f_0}$$

Trong trường hợp các lệnh khác nhau có CPI khác nhau cần tính CPI trung bình. (làm nốt CPI trung bình)

Nếu các loại lệnh khác nhau, có số chu kỳ khác nhau, ta có tổng số chu kỳ:

$$n = \sum_{i=1}^{K} (CPI_i * IC_i)$$

CPI trung bình:

$$CPI_{TB} = \frac{n}{IC} = \frac{1}{IC} \sum_{i=1}^{K} (CPI_i * IC_i)$$

1.4.6 Số triệu lệnh được thực hiện trong một giây

MIPs (Million Instruction Per Second):

$$MIPs = \frac{1}{\sum (IF * CPI * \tau)} * 1000$$

Trong đó:

IF: tần suất xuất hiện lệnh

CPI: số chu kỳ

τ : thời gian một chu kỳ máy

Ví dụ:

Loại lệnh	**IF (%)**	**CPI**
Nạp và ghi dữ liệu	30.4	1.5
Cộng trừ số nguyên	10	1
Nhân chia số nguyên	3.8	10
Cộng trừ số thực	9.5	7
Nhân chia số thực	6.5	15
Các phép tính logic	3	1
Lệnh rẽ nhánh	20	1.5
So sánh và dịch ở dữ liệu	16.8	2

Cho τ = 10 ns. Tính MIPs = ?

$$MIPs = \frac{100}{\sum (30.4*1.5) + (10*1) + (3.8*10) + (9.5*7) + (6.5*15) + (3*1) + (20*1.5) + (16.8*2)} *1000$$

$$= 30.845 \text{ (mips)}$$

1.5 Các phương pháp vào ra dữ liệu máy tính

1.5.1 Phương pháp thăm dò trạng thái thiết bị ngoại vi

CPU sẽ liên tục gửi các tín hiệu hỏi xem có thiết bị nào sẵn sàng gửi dữ liệu cho mình hay không. Nếu có một thiết bị ngoại vi nào đó muốn trao đổi dữ liệu với CPU thì nó sẽ gửi lại tín hiệu sẵn sàng trao đổi. Khi đó CPU sẽ thực hiện việc trao đổi dữ liệu với thiết bị ngoại vi này.

1.5.2 Phương pháp sử dụng ngắt

CPU vẫn làm công việc của mình, chỉ khi nào có một thiết bị ngoại vi hay một thành phần nào đó của máy vi tính có yêu cầu trao đổi dữ liệu với CPU thì thiết bị ngoại vi này (thành phần của máy tính này) sẽ gửi một yêu cầu ngắt tới CPU, khi đó CPU sẽ tạm dừng công việc hiện tại để quay ra trao đổi dữ liệu với thiết bị ngoại vi này. Trao đổi dữ liệu xong, CPU quay trở lại thực hiện tiếp công việc đang làm dở trước đó.

1.5.3 Phương pháp truy cập bộ nhớ trực tiếp DMA

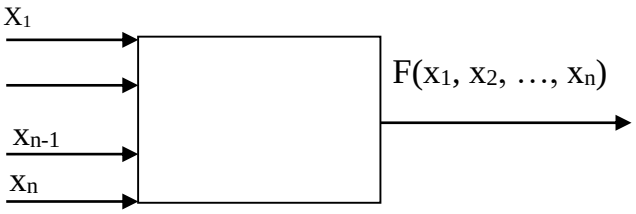
Thường được sử dụng khi có yêu cầu trao đổi dữ liệu lớn giữa CPU và thiết bị ngoại vi. CPU trao quyền quản lý cho bộ điều khiển truy nhập trực tiếp, lúc này việc trao đổi dữ liệu giữa thiết bị ngoại vi và bộ nhớ chính sẽ do bộ điều khiển này thực hiện.

1.5.4 Phương pháp sử dụng kênh dữ liệu

Xây dựng các đường bus dữ liệu riêng, nối trực tiếp thiết bị ngoại vi và bộ nhớ chính, mọi hế thống bus lại có một bộ điều khiển riêng được gọi là kênh dữ liệu.

1.6 Thiết kế một số mạch logic đơn giản

1.6.1 Cổng Logic cơ sở



Hình 1.11 Cổng Logic cơ sở

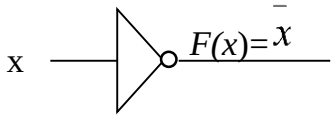
Xét một thiết bị như hình trên, có một số đường vào (dẫn tín hiệu vào) và chỉ có một đường ra (phát tín hiệu ra). Giả sử các tín hiệu vào x_1, x_2, …, x_n (ta gọi là đầu vào hay input) cũng như tín hiệu ra F (đầu ra hay output) đều chỉ có hai trạng thái khác nhau, tức là mang một bit thông tin, mà ta ký hiệu là 0 và 1.

Ta gọi một thiết bị với các đầu vào và đầu ra mang giá trị 0, 1 như vậy là một mạch logic. Đầu ra của một mạch logic là một hàm Boole F của các đầu vào x_1, x_2, …, x_n. Ta nói mạch logic trong hình trên thực hiện hàm F. Các mạch logic được tạo thành từ một số mạch cơ sở, gọi là cổng logic. Các cổng logic sau đây thực hiện các hàm phủ định, hội và tuyển…

1.6.1.1 Cổng NOT

Cổng NOT thực hiện hàm phủ định. Cổng chỉ có một đầu vào. Đầu ra *F(x)* là phủ định của đầu vào *x*.

$$F(x) = \bar{x} = \begin{cases} 0 \; khi & x = 1, \\ 1 \; khi & x = 0. \end{cases}$$



Bảng sự thật hàm phủ định NOT: $F = \bar{x}$

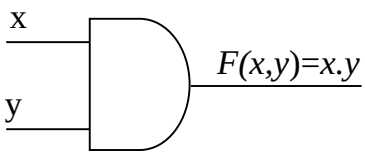
A	NOT A
0	1
1	0

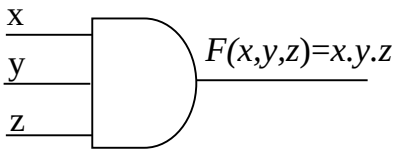
Chẳng hạn, xâu bit 100101011 qua cổng NOT cho xâu bit 011010100.

1.6.1.2 Cổng AND

Cổng AND thực hiện hàm hội. Đầu ra *F(x,y)* là hội (tích) của các đầu vào.

$$F(x,y) = xy = \begin{cases} 1 & khi\ x = y = 1 \\ 0 & \text{trong các trường hợp khác.} \end{cases}$$





Bảng sự thật hàm AND: $F(x,y) = x.y$

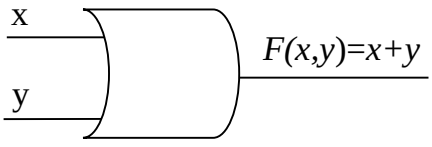
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

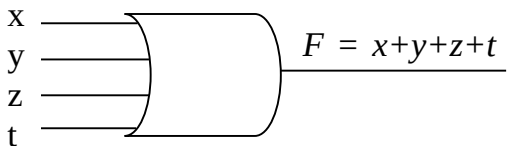
Ví dụ: hai xâu bit 101001101 và 111010110 qua cổng AND cho 101000100.

1.6.1.3 Cổng OR

Cổng OR thực hiện hàm tuyển (tổng). Đầu ra *F(x,y)* là tuyển (tổng) của các đầu vào.

$$F(x,y) = x + y = \begin{cases} 1 & khi\ x = 1\ hay\ y = 1, \\ 0 & khi\ x = y = 0. \end{cases}$$





Bảng sự thật hàm OR: $F(x,y) = x + y$

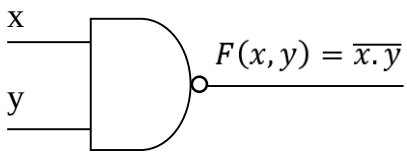
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

Ví dụ: hai xâu bit 101001101 và 111010100 qua cổng OR cho 111011101.

1.6.1.4 Cổng NAND

Xét hàm Sheffer $F(x,y)=\overline{x.y}=\begin{cases}0 & khi\ x=y=1,\\ 1 & khi\ x=0\ hay\ y=0.\end{cases}$

Mạch logic thực hiện hàm Sheffer gọi là cổng NAND, được vẽ như hình dưới đây.



Bảng sự thật hàm NAND: $F(x,y)=\overline{x.y}$

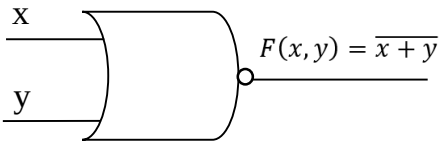
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

Bất kỳ một hàm Boole nào cũng có thể thực hiện được bằng một mạch logic chỉ gồm có cổng NAND.

1.6.1.5 Cổng NOR

Xét hàm Vebb $F(x,y)=\overline{x+y}=\begin{cases}0 & khi\ x=1\ hay\ y=1,\\ 1 & khi\ x=y=0.\end{cases}$

Mạch logic thực hiện hàm Vebb gọi là cổng NOR, được vẽ như hình dưới đây.



Bảng sự thật hàm NOR: $F(x,y)=\overline{x+y}$

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

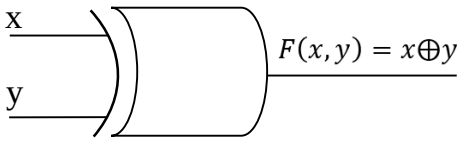
Bất kỳ hàm Boole nào cũng có thể thực hiện được bằng một mạch logic chỉ gồm cổng NOR.

1.6.1.6 Cổng XOR

Một phép toán logic quan trọng khác là phép tuyển loại:

$$F(x,y) = x \oplus y = \begin{cases} 0 & khi\ x = y, \\ 1 & khi\ x \neq y. \end{cases}$$

Mạch logic này là một cổng logic, gọi là cổng XOR, được vẽ như hình dưới đây



Bảng sự thật hàm XOR: $F = (x,y) = x \oplus y$

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

1.6.2 Thiết kế mạch Logic

Các cổng logic có thể lắp ghép để được những mạch logic thực hiện các hàm Boole phức tạp hơn. Như ta đã biết rằng một hàm Boole bất kỳ có thể biểu diễn bằng một biểu thức chứa các toán tử và toán hạng. Từ đó suy ra có thể lắp ghép thích hợp các cổng NOT, AND, OR, NAND, NOR, XOR để được một mạch logic thực hiện một hàm Boole bất kỳ.

Để thiết kế mạch logic thực hiện hàm Boole bất kỳ gồm ba bước chính:

Bước 1: xây dựng bảng chân lý gồm các tín hiệu vào và tín hiệu ra

Bước 2: xác định hàm đầu ra theo dạng tổng chuẩn tắc hoàn toàn

Bước 3: vẽ mạch tương ứng với hàm đầu ra.

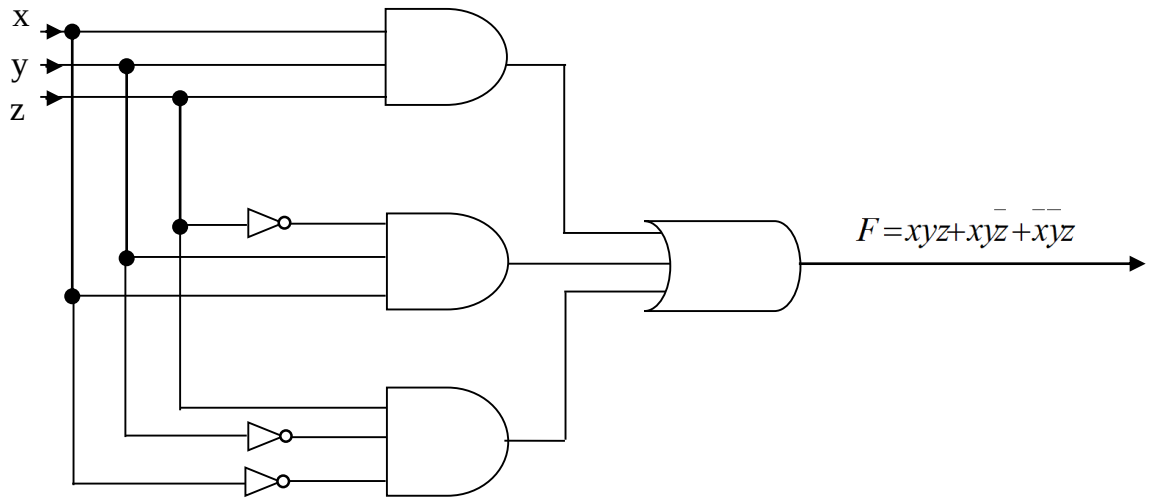
Ví dụ: Xây dựng một mạch logic thực hiện hàm Boole cho bởi bảng sau.

x	y	z	F(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Theo bảng này, hàm F có dạng tổng (tuyển) chuẩn tắc hoàn toàn là:

$F(x,y,z) = xyz + xy\bar{z} + \bar{x}\bar{y}z$.

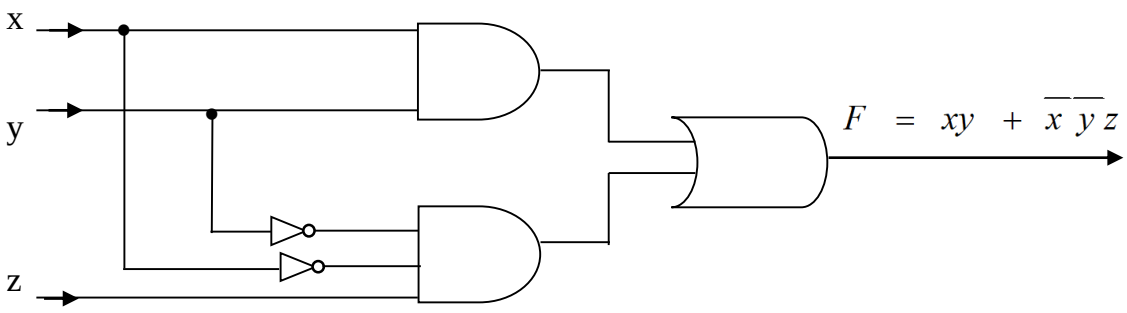
Hình dưới đây vẽ mạch logic thực hiện hàm F đã cho.



Biểu thức của *F(x, y, z)* có thể rút gọn:

$xyz + xy\bar{z} + \bar{x}\bar{y}z = xy(z + \bar{z}) + \bar{x}\bar{y}z = xy + \bar{x}\bar{y}z$.

Hình dưới đây cho ta mạch logic thực hiện hàm $xy + \bar{x}\bar{y}z$.



Hai mạch logic trong hai hình trên thực hiện cùng một hàm Boole, ta nói đó là hai mạch logic tương đương, nhưng mạch logic thứ hai đơn giản hơn. [1]

1.7 Hệ tổ hợp và hệ dãy

1.7.1 Khái niệm hệ tổ hợp và hệ dãy

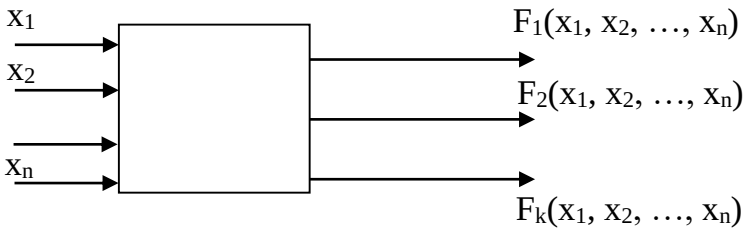
Hệ tổ hợp: là hệ mà tín hiệu ra chỉ phụ thuộc tín hiệu vào tại thời điểm hiện tại. Hệ tổ hợp được gọi là hệ không nhớ. Hệ tổ hợp thực hiện bằng những phần tử logic cơ bản.

Hệ dãy: là hệ mà tín hiệu ra không chỉ phụ thuộc tín hiệu vào tại thời điểm hiện tại, mà còn phụ thuộc quá khứ của tín hiệu vào. Hệ dãy được gọi là hệ có nhớ. Mạch của hệ dãy bắt buộc phải có các phần tử nhớ. Ngoài ra có thể thêm các phần tử logic cơ bản.

1.7.2 Xây dựng bộ cộng, bộ trừ

1.7.2.1 Bộ cộng

Nhiều bài toán đòi hỏi phải xây dựng những mạch logic có nhiều đường ra, cho các đầu ra $F_1, F_2, ..., F_k$ là các hàm Boole của các đầu vào $x_1, x_2, \ldots, x_n$.



Hình 1.12 Mạch Logic

Bộ cộng bán phần HA (Haft Adder)

Ta xét phép cộng hai số tự nhiên từ các khai triển nhị phân của chúng. Trước hết, ta sẽ xây dựng một mạch có thể duợc dùng để tìm x+y với x, y là hai số 1-bit. Đầu vào mạch này sẽ là x và y. Đầu ra sẽ là một số 2-bit $\overline{cs}$, trong đó s là bit tổng và c là bit nhớ.

0+0 = 00

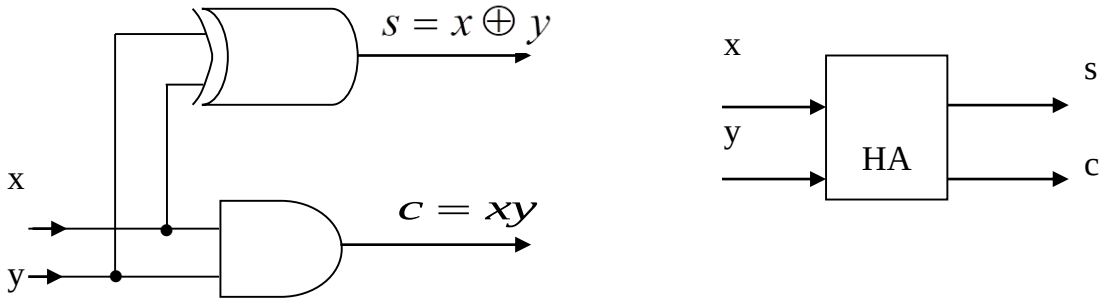
0+1 = 01

1+0 = 01

1+1 = 10

x	*y*	*c*	*s*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Từ bảng trên, ta thấy ngay $s = x \oplus y, c = x.y$. Ta vẽ được mạch thực hiện hai hàm $s = x \oplus y$ và $c = x.y$ như hình dưới đây. Mạch này gọi là mạch cộng hai số 1-bit hay mạch cộng bán phần, ký hiệu là HA.



Hình 1.13 Mạch cộng bán phần HA

Bộ cộng toàn phần FA (Full Adder)

Xét phép cộng hai số 2-bit $\overline{a_2 a_1}$ và $\overline{b_2 b_1}$, $a_2\ a_1$

$b_2\ b_1$

Thực hiện phép cộng theo từng cột, ở cột thứ nhất (từ phải sang trái) ta tính $a_1 + b_1$ được bit tổng s_1 và bit nhớ c_1; ở cột thứ hai, ta tính $a_2 + b_2 + c_1$, tức là phải cộng ba số 1-bit. Cho *x, y, z* là ba số 1-bit. Tổng *x+y+z* là một số 2-bit $\overline{cs}$, trong đó *s* là bit tổng của *x+y+z* và *c* là bit nhớ của *x+y+z*. Các hàm Boole *s* và *c* theo các biến *x, y, z* được xác định bằng bảng sau:

x	y	z	c	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Từ bảng này, dễ dàng thấy rằng:

$$s = x \oplus y \oplus z$$

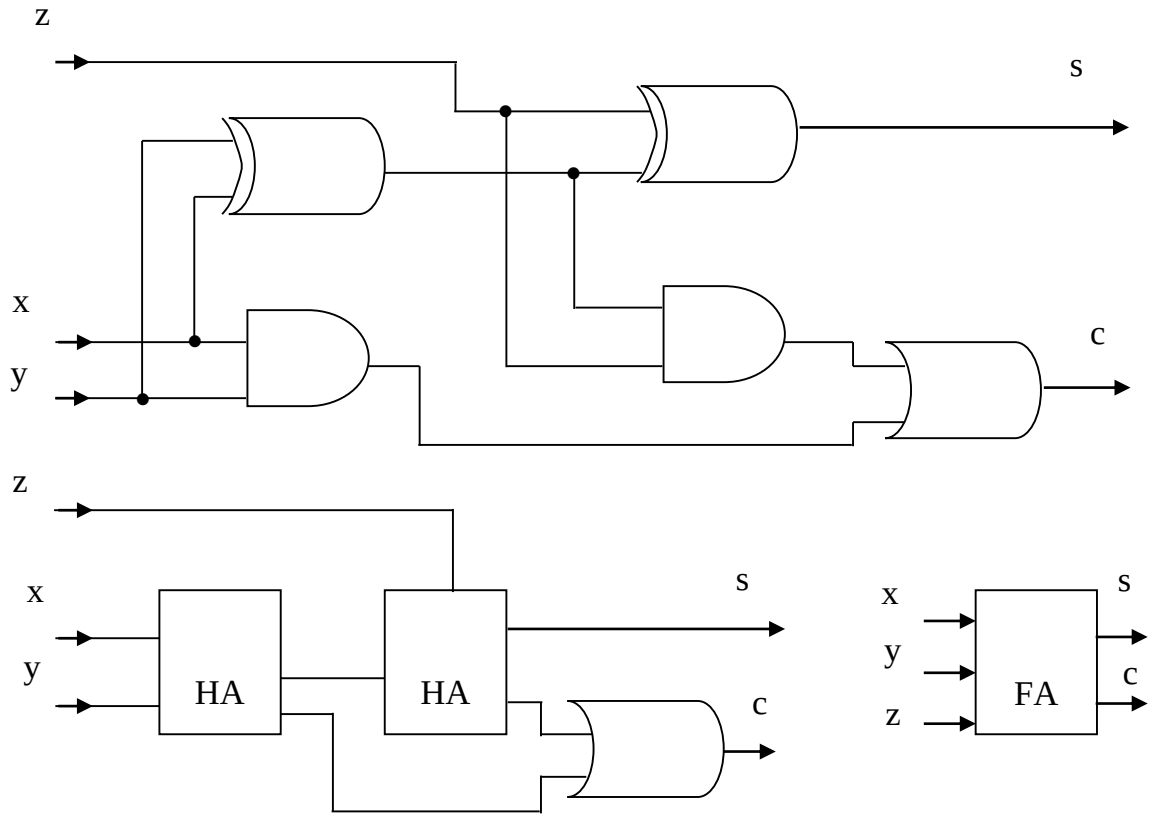
Hàm c có thể viết dưới dạng tổng chuẩn tắc hoàn toàn là:

$$c = \overline{x}.y.z + x.\overline{y}.z + x.y.\overline{z} + x.y.z$$

Công thức của *c* có thể rút gọn:

$$c = z.(\overline{x}.y + x.\overline{y}) + x.y.(\overline{z} + z) = z.(x \oplus y) + x.y\,.$$

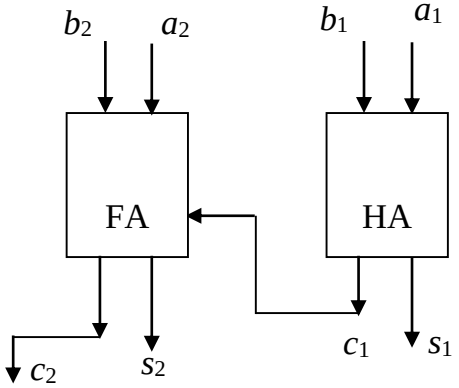
Ta vẽ được mạch thực hiện hai hàm Boole $s = x \oplus y \oplus z$ và $c = z.(x \oplus y) + x.y$ như hình dưới đây, mạch này là ghép nối của hai mạch cộng bán phần (HA) và một cổng OR. Đây là mạch cộng ba số 1-bit hay mạch cộng toàn phần, ký hiệu là FA.



Hình 1.14 Mạch cộng toàn phần FA

Trở lại phép cộng hai số 2-bit $\overline{a_2 a_1}$ và $\overline{b_2 b_1}$. Tổng $\overline{a_2 a_1} + \overline{b_2 b_1}$ là một số 3-bit $\overline{c_2 s_2 s_1}$, trong đó s_1 là bit tổng của a_1+b_1: $s_1 = a_1 \oplus b_1$, s_2 là bit tổng của $a_2+b_2+c_1$, với c_1 là bit nhớ của a_1+b_1: $s_2 = a_2 \oplus b_2 \oplus c_1$ và c_2 là bit nhớ của $a_2+b_2+c_1$.

Ta có được mạch thực hiện ba hàm Boole s_1, s_2, c_2 như hình dưới đây.



1.7.2.2 Bộ trừ

Bộ trừ bán phần HS (Haft Subtractor)

Ta xét phép trừ hai số tự nhiên từ các khai triển nhị phân của chúng. Trước hết, ta sẽ xây dựng một mạch có thể duợc dùng để tìm a - b với a, b là hai số 1-bit, a là số bị trừ

và b là số trừ. Đầu vào mạch này sẽ là a và b. Đầu ra sẽ là một số 2-bit $\overline{DB}$, trong đó D là bit hiệu và B là bit mượn.

0-0 = 0 mượn 0

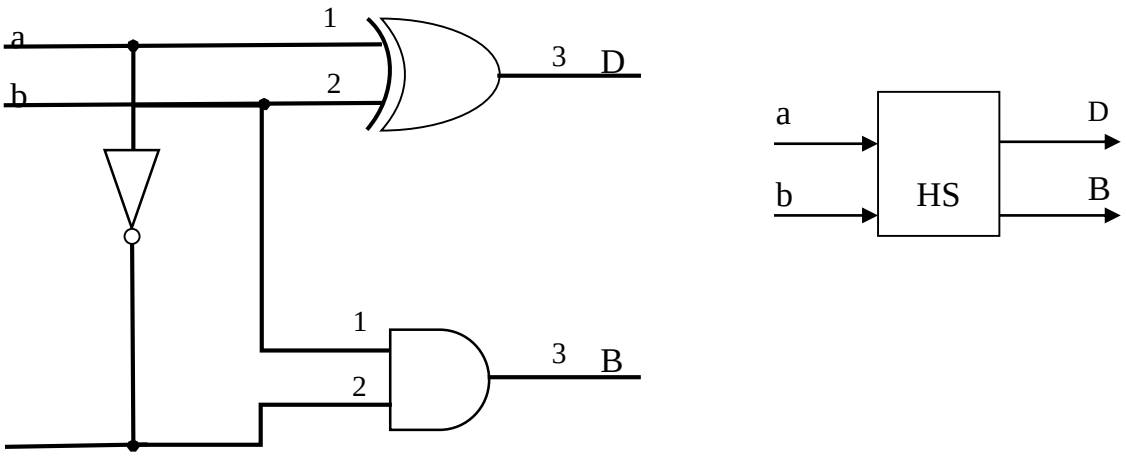
1-0 = 0 mượn 0

0-1 = 1 mượn 1

1-1 = 0 mượn 0

A	*b*	*D*	*B*
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

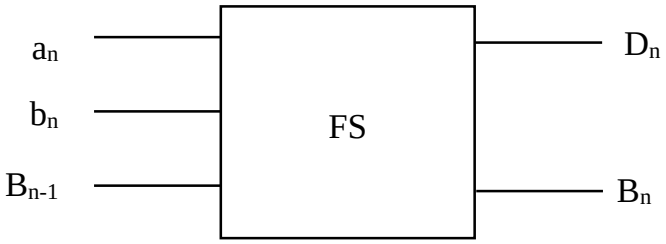
Từ bảng trên, ta thấy ngay $D = a.\bar{b} + \bar{a}.b = a \oplus b; B = \bar{a}.b$. Ta vẽ được mạch thực hiện hai hàm $D = a \oplus b$ và $B = \bar{a}.b$ như hình dưới đây. Mạch này gọi là mạch trừ hai số 1-bit hay mạch trừ bán phần, ký hiệu là HS.



Hình 1.15 Mạch trừ bán phần HS

Bộ trừ toàn phần FS (Full Subtractor)

Sơ đồ mô phỏng



Trong đó: B_{n-1}: số mượn của lần trừ trước đó

B_n: số mượn của lần trừ hiện tại

D_n: hiệu số hiện tại

a_n: số bị trừ

b_n: số trừ

Bảng chân lý mô tả hoạt động của mạch:

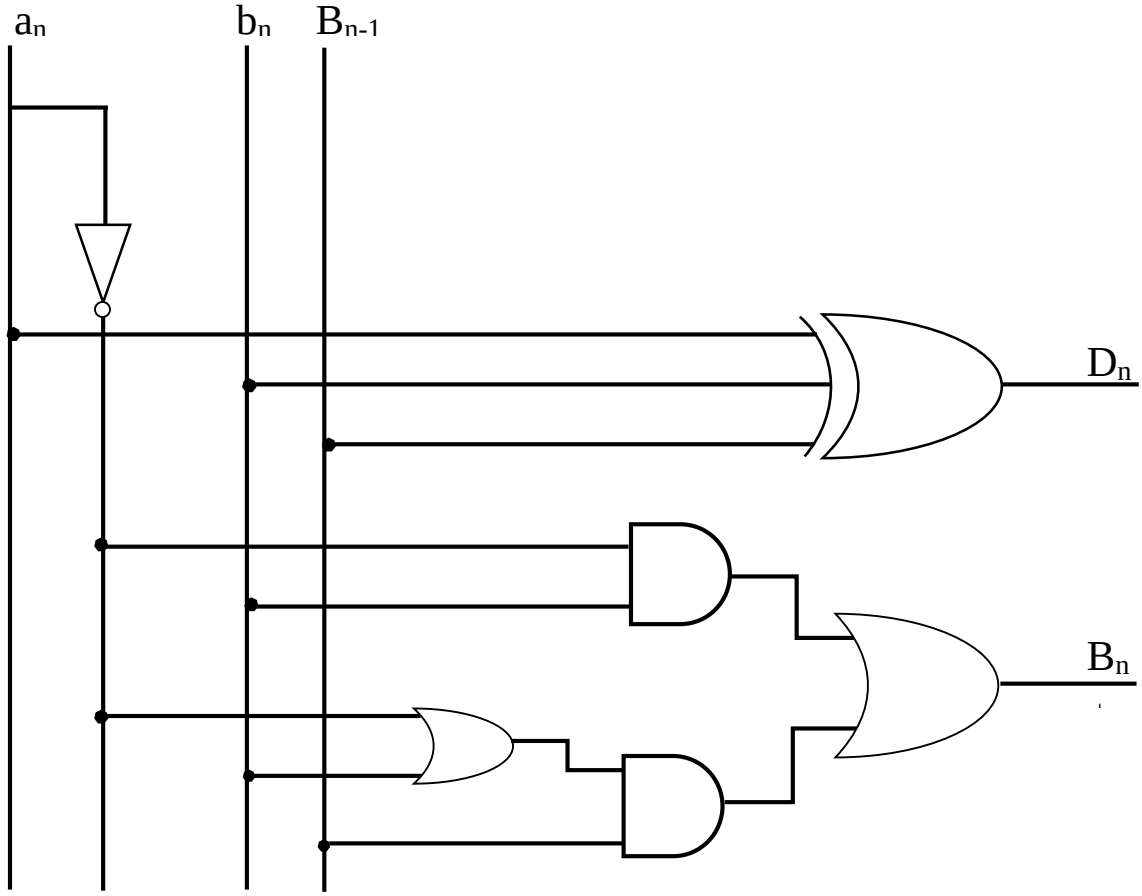
a_n	b_n	B_{n-1}	D_n	B_n
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Từ bảng sự thật của mạch trừ toàn phần FS, ta có hàm đầu ra của 2 bít D và B như sau:

$$D_n = \overline{a_n}.\overline{b_n}.B_{n-1} + a_n.b_n.B_{n-1} + \overline{a_n}.b_n.\overline{B_{n-1}} + a_n.\overline{b_n}.\overline{B_{n-1}} = a_n \oplus b_n \oplus B_{n-1}$$

$$B_n = \overline{a_n}.B_{n-1} + b_n.B_{n-1} + \overline{a_n}.b_n = \overline{a_n}.b_n + B_{n-1}.(\overline{a_n} + b_n)$$

Sơ đồ mạch Logic như sau:



Hình 1.16 Mạch trừ toàn phần FS

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 1

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Hãy nêu những đặc điểm cơ bản của các loại máy tính theo các tiêu chí phân loại?

Câu 2. Mô hình máy tính cơ bản gồm các thành phần chính nào. Hãy nêu chức năng cơ bản của các thành phần đó?

Câu 3. Vẽ mô hình phân lớp máy tính?

Câu 4. Nêu cấu tạo chung và vẽ sơ đồ kiến trúc máy tính Von Neumann?

Câu 5. Có mấy loại bus dùng trong kiến trúc máy tính. Hãy kể tên và nêu chức năng của từng loại? Vẽ sơ đồ hoạt động của máy tính ứng với các đường bus kể trên?

Câu 6. Nêu cách tính các thông số điển hình để đánh giá hiệu năng của máy tính?

Câu 7. Nêu các phương pháp vào ra dữ liệu của máy tính?

Câu 8. Kể tên các thông số cơ bản ảnh hưởng đến hiệu năng của máy tính?

Câu 9. CPI là gì? Nêu các tính thông số CPI?

Câu 10. MIPs là gì? Nêu cách tính MIPs?

Câu hỏi trắc nghiệm

Câu 1. Kiến trúc máy tính là gì?

A. Là một môn học về việc thiết kế các chức năng của hệ thống. Nghiên cứu tính khả thi, giá thành, tốc độ và các yếu tố kỹ thuật khác liên quan đến việc thiết kế

B. Nó liên quan đến ngôn ngữ lập trình

C. Nó liên quan đến các khía cạnh lý thuyết cao cấp của việc thiết kế máy tính

D. Nó là một môn học lý thuyết về máy tính.

Câu 2. Cấu trúc máy tính là thuật ngữ:

A. Là một môn học về việc thiết kế các chức năng của hệ thống. Nghiên cứu tính khả thi, giá thành, tốc độ và các yếu tố kỹ thuật khác liên quan đến việc thiết kế.

B. Nó liên quan đến ngôn ngữ lập trình máy tính.

C. Nó liên quan đến các khía cạnh lý thuyết cao cấp của việc thiết kế máy tính.

D. Quan tâm đến các đơn vị vận hành và sự kết nối giữa chúng nhằm hiện thực hóa những đặc tả về kiến trúc, chẳng hạn như tín hiệu điều khiển, giao diện giữa máy tính với các thiết bị ngoại vi,...

Câu 3. Bộ xử lý trung tâm CPU có chức năng:

A. Lấy chỉ thị từ bộ nhớ chính, giải mã và điều khiển ALU

B. Thi hành các chương trình được chứa trong bộ nhớ chính

C. Thực hiện các thao tác đơn giản

D. Chứa các kết quả tạm thời và thông tin điều khiển nhất định.

Câu 4. Cấu trúc phần cứng của máy tính

A. CPU, ALU, Bộ nhớ trong
C. CPU, hệ thống vào ra, Bộ nhớ trong
B. CPU, CU, ALU
D. CPU, CU, Bộ nhớ trong

Câu 5. Hệ thống BUS dùng để làm gì?

A. Nối các bộ phận của máy tính lại với nhau
C. Nối bộ nhớ ngoài với bộ nhớ trong
B. Nối CPU với bộ nhớ ngoài
D. Nối bộ xử lý với các bộ phận bên ngoài

Câu 6. Chức năng của BUS địa chỉ?

A. Vận chuyển địa chỉ để xác định ngăn nhớ hay cổng vào ra.

B. Vận chuyển lệnh từ bộ nhớ đến CPU.

C. Vận chuyển các tín hiệu điều khiển.

D. Vận chuyển dữ liệu giữa CPU, các Modul nhớ và Modul vào ra với nhau.

Câu 7. Chức năng của BUS điều khiển?

A. Vận chuyển lệnh từ bộ nhớ đến CPU.

B. Vận chuyển dữ liệu giữa CPU, các Modul nhớ và Modul vào ra với nhau.

C. Vận chuyển các tín hiệu điều khiển.

D. Vận chuyển địa chỉ để xác định ngăn nhớ hay cổng vào ra.

Câu 8. Chức năng của BUS dữ liệu?

A. Vận chuyển địa chỉ để xác định ngăn nhớ hay cổng vào ra.

B. Vận chuyển lệnh từ bộ nhớ đến CPU và vận chuyển dữ liệu giữa CPU, các Modul nhớ và Modul vào ra với nhau.

C. Vận chuyển các tín hiệu điều khiển.

D. Vận chuyển dữ liệu giữa CPU, các Modul nhớ và Modul vào ra với nhau.

Câu 9. CPI (Cycle Per Instruction) là gì?

A. Số lệnh được thực hiện

B. Thời gian thực hiện lệnh

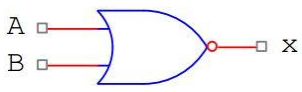
C. Tần số

D. Số chu kỳ cần thiết để thực hiện một lệnh.

Câu 10. MIPs (Million Instruction Per Second) là gì?

A. Tần suất xuất hiện lệnh
C. Số lệnh được thực hiện
B. Số chu kỳ
D. Số triệu lệnh được thực hiện trong 1 giây

Câu 11. Hãy cho biết hình vẽ sau là của cổng Logic cơ sở nào?



A. Cổng NAND
B. Cổng XOR
C. Cổng OR
D. Cổng NOR

Câu 12. Hãy cho biết bảng chân lý sau của cổng logic cơ sở nào?

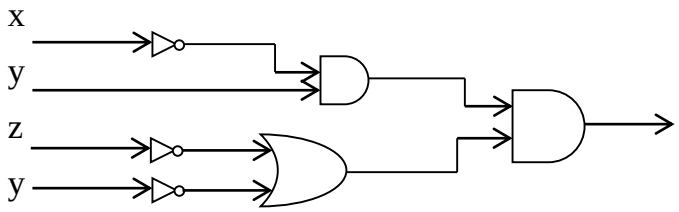
A	B	F(A,B)
0	0	0
0	1	0
1	0	0
1	1	1

A. Cổng NOT
B. Cổng NOR
C. Cổng OR
D. Cổng AND

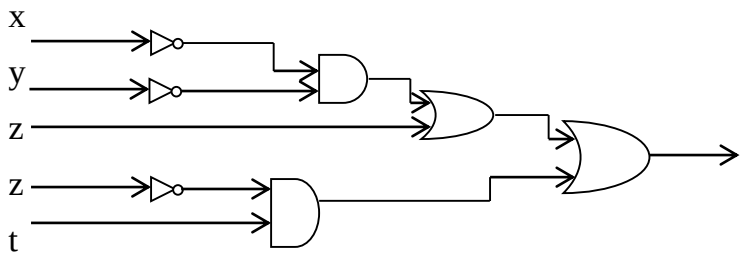
Bài tập áp dụng

Câu 1. Tìm hàm đầu ra của các mạch sau:

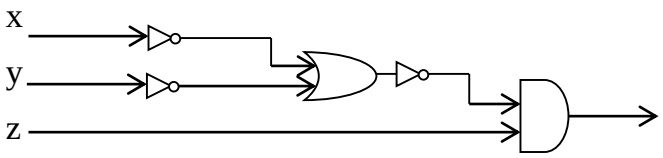
a.



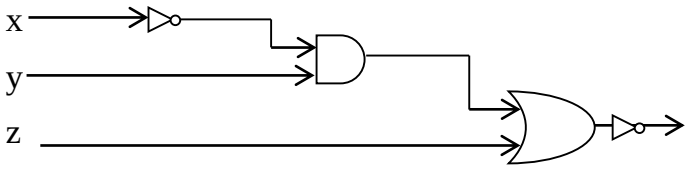
b.



c.

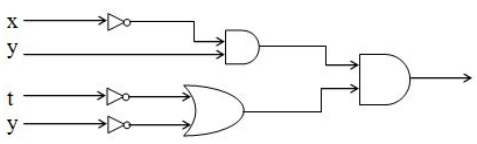


d.

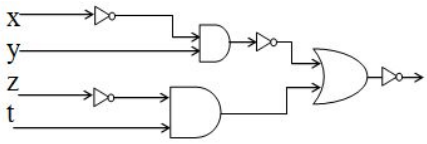


Câu 2. Tìm hàm đầu ra của các mạch sau:

a.



b.



Câu 3. Xây dựng các mạch logic tương ứng với các hàm đầu ra sau:

a. $\overline{(\bar{x}+z)(y+\bar{z})}$ b. $\overline{(x+y)}x$

c. $x + y\overline{(x+z)}$ d. $(x\bar{z} + \bar{y}z) + \bar{x}$

Câu 4. Xây dựng các mạch logic tương ứng với các hàm đầu ra sau:

a. $(\bar{x} + y) + \overline{\bar{z}.t}$ b. $\overline{\overline{x + \bar{y}} + \bar{t}}$

c. $(\bar{y}.\bar{z}).\bar{t}$ d. $\overline{\bar{y} + z} + \bar{t}$

Câu 5: Vẽ sơ đồ kiến trúc máy tính tuần tự Von neumann?

Câu 6. Thiết kế bộ cộng bán phần HA và bộ cộng toàn phần FA?

Câu 7. Thiết kế bộ trừ bán phần HS và bộ trừ toàn phần FS?

Câu 8. Vẽ bảng chân lý cho các cổng cơ sở AND, OR, XOR, NOT?

Câu 9. Vẽ bảng chân lý cho các cổng NAND, NOR?

Câu 10. Thiết kế bộ cộng hai số n bit?

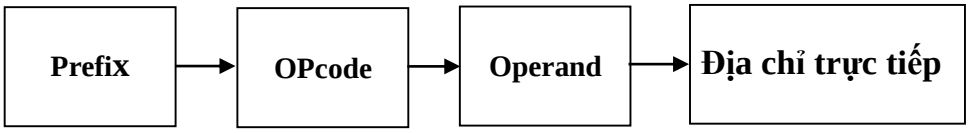
Chương 2: THIẾT KẾ HỆ LỆNH

***Mục đích**: Giới thiệu khái niệm về dạng lệnh, kích thước mã lệnh, các kiểu định địa chỉ được dùng trong kiến trúc máy tính. Số lượng các tham số trong một lệnh, loại và chiều dài của toán hạng, tác vụ mà máy tính có thể thực hiện. Giới thiệu tổng quát tập lệnh của các kiến trúc máy tính. Biết bản chất quy trình thực hiện một lệnh trong máy tính.*

***Yêu cầu**: Sinh viên hiểu được kiến thức về tập lệnh. Nắm vững các kiểu định địa chỉ được dùng trong kiến trúc máy tính, số lượng các tham số trong một lệnh, loại và chiều dài của toán hạng, tác vụ mà máy tính có thể thực hiện. Áp dụng vào làm bài tập viết các đoạn chương trình tính giá trị biểu thức bằng các hệ lệnh.*

2.1. Giới thiệu dạng lệnh, kích thước mã lệnh.

Một lệnh mô tả bằng mã nhị phân có thể dài từ 1 đến 6 byte. Cấu trúc chung của một mã lệnh tuân theo sơ đồ sau:



Hình 2.1 Cấu trúc chung của một lệnh

Prefix (Tiền tố) đi trước mã lệnh. Phần này có thế có, có thể không. Ví dụ tiền tố 3Eh báo hiệu vô hiệu hóa đoạn DS.

Operation code (Mã toán) chỉ ra các thao tác mà CPU cần thực hiện, phân biệt đó là lệnh gì. Đối với CPU mã lệnh là một chuỗi các bit 0, 1. Đối với người sử dụng mã lệnh là một chuỗi các từ gợi nhớ. Ví dụ với lệnh dịch chuyển MOV có mã toán là 100010.

Toán hạng (operand) dùng để xác định những đối tượng mà ở đó phép toán được thực hiện (nội dung của thanh ghi hay bộ nhớ). Các toán hạng thường được ngăn cách nhau bởi dấu phẩy. Toán hạng có thể có hoặc không.

Ví dụ lệnh cộng hay nhân hai số hạng với nhau yêu cầu phải có hai toán hạng, lệnh dịch chuyển cần một toán hạng, lệnh xóa cờ không cần toán hạng nào.

Địa chỉ trực tiếp: Xác định địa chỉ của nơi chứa các toán hạng.

2.2. Số lượng các lệnh cho bộ Vi xử lý

Khi thiết kế một hệ lệnh cho bộ VXL nảy sinh một số vấn đề sau:

- Số lượng lệnh là bao nhiêu.
- Số lượng tham số trong hệ lệnh là bao nhiêu.
- Phân chia các lệnh theo dạng nào?

Để thực hiện việc mã hóa các lệnh người ta sử dụng một dãy số nhị phân. Độ dài của dãy số này sẽ quy định số lượng các lệnh mà bộ xử lý có thể xử lý được. Nếu số lượng lệnh ít, người ta sẽ phải cần nhiều lệnh để biểu diễn một thao tác hoặc một phép xử lý. Điều

này kéo theo kích thước chương trình sẽ dài hơn. Tuy nhiên bộ xử lý loại này sẽ dễ dàng thiết kế hơn.

Nếu số lượng của lệnh nhiều, dẫn đến phải sử dụng nhiều bit hơn để biểu diễn một lệnh, nhưng mỗi lệnh lại có thể thực hiện được những thao tác phức tạp, do vậy một chương trình có thể ngắn hơn hoặc một chương trình có thể sử dụng ít lệnh hơn so với chương trình cùng loại nếu viết trên bộ xử lý ít lệnh. Tuy nhiên các bộ xử lý này rất khó thiết kế.

Trong thực tế tồn tại hai dòng vi xử lý tương ứng với hai loại số lượng lệnh khác nhau:

Vi xử lý có ít lệnh: RISC - reduced instruction set computer. Máy tính với tập lệnh rút gọn.

Vi xử lý có nhiều lệnh: CISC - complex instruction set computer. Máy tính với tập lệnh phức tạp. [2]

2.3. Phân loại lệnh

Phần kiến thức về các lệnh Asembly đã được học trong học phần Vi xử lý, trong tài liệu học tập này sẽ chỉ trình bày khái quát, cú pháp và ý nghĩa các loại lệnh asembly.

Mỗi bộ vi xử lý có một tập lệnh xác định, các bộ vi xử lý thế hệ sau thường có tập lệnh được bổ sung, mở rộng hơn so với các bộ vi xử lý thế hệ trước nó, điều đó có nghĩa các bộ vi xử lý thế hệ sau có thể chạy được các chương trình viết cho các bộ vi xử lý trước. Nhưng ngược lại thì không hoàn toàn đúng. Như đã nói trên đây, chúng ta lấy bộ vi xử lý Intel 8088 làm cơ sở để nghiên cứu những vấn đề kỹ thuật của các bộ vi xử lý khác. Vì vậy ở đây chúng ta cũng sẽ nghiên cứu tập lệnh của chính bộ vi xử lý này.

Tập lệnh của 8086/8088 gồm hơn 100 ký hiệu gợi nhớ (mnemonic) của lệnh ngôn ngữ assembler cơ sở, để quy định cho bộ vi xử lý phải làm gì. Mỗi lệnh cơ sở có thể có nhiều biến cách.

Ví dụ: có tới hai mươi tám biến cách khác nhau cho lệnh dịch chuyển cơ sở (MOV). Tuy nhiên trong chương trình môn học này, chúng ta chỉ xem xét một số lệnh cần thiết theo mục tiêu của môn học. Các lệnh mà chúng ta sẽ nghiên cứu được chia làm sáu nhóm:

1. Nhóm lệnh truyền dữ liệu.
2. Nhóm lệnh số học.
3. Nhóm lệnh logic.
4. Nhóm lệnh so sánh.
5. Nhóm lệnh điều khiển chương trình.
6. Các lệnh đặc biệt.

2.3.1 Nhóm lệnh truyền dữ liệu

MOV lệnh di chuyển dữ lệu cơ bản. Lệnh này có thể sử dụng để di chuyển byte (8 bit) hoặc từ (16 bit) của dữ liệu.

Cấu trúc lệnh : **MOV đích, nguồn.**

Trong đó toán hạng đích và gốc có thể tìm theo các địa chỉ khác nhau, nhưng phải có cùng độ dài và không được phép đồng thời là hai ô nhớ hoặc hai thanh ghi đoạn.

Đích	Nguồn	Ví dụ	Giải thích
1 Bộ nhớ	Thanh ghi	**MOV 100H, AX**	- Chuyển nội dung trong AX vào vị trí nhớ 100H.
2 Thanh ghi	Bộ nhớ	**MOV AX, MEM1**	- Chuyển nội dung trong vị trí nhớ do nhãn MEM1 chỉ ra vào thanh ghi AX.
3 Thanh ghi	Thanh ghi	**MOV AX, BX**	- Chuyển nội dung trong BX vào thanh ghi AX.
4 Thanh ghi	Tức thời	**MOVAX, 0FFFFH**	- Chuyển giá trị hằng số FFFFH vào thanh ghi AX; số 0 ở đầu được dùng để phân biệt và chỉ rõ FFFFH là một giá trị hằng chứ không phải là một nhãn.

Bảng 2.1 Các ví dụ về lệnh MOV

XCHG -exchange two operands (hoán đổi nội dung 2 toán hạng).

Cấu trúc lệnh: **XCHG Đích, Nguồn**

Trong đó toán hạng đích và nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải có cùng độ dài, không được phép đồng thời là hai ô nhớ, và cũng không được là thanh ghi đoạn.

Ví dụ:

```
XCHG AH, AL      ; tráo nội dung AH và AL.
XCHG AL, [BX]    ; tráo nội dung AL với ô nhớ có địa chỉ DS:BX.
```

IN- Input data from a port (Đọc dữ liệu từ cổng vào thanh Acc)

Cấu trúc lệnh: **IN Acc, Port**

Port là địa chỉ 8 bit của cổng, nó có thể có giá trị trong khoảng 00H..FFH.

Nếu Acc là AL thì dữ liệu 8 bit được đưa vào từ cổng Port.

Nếu Acc là AX thì dữ liệu 16 bit được đưa vào từ cổng Port và Port+1. Có thể biểu diễn địa chỉ cổng thông qua thanh ghi DX và như vậy địa chỉ cổng được địa chỉ hoá linh hoạt hơn. Lúc này địa chỉ cổng nằm trong dải 0000H..FFFFH và lệnh được viết như sau:

IN Acc, DX

Trong đó DX phải được gán từ trước giá trị ứng với cổng.

OUT- Output a byte or word to a port (Đưa dữ liệu ra cổng từ Acc).

Cấu trúc lệnh: **OUT Port, Acc**

Nếu Acc là AL thì dữ liệu 8 bit được đưa ra cổng Port

Nếu Acc là AH thì dữ liệu 16 bit được đưa ra cổng Port và cổng Port+1.

Tương tự với lệnh **IN**, ở đây cũng có thể dùng thanh ghi DX để chứa địa chỉ cổng. Khi đó lệnh được viết như sau:

OUT DX, Acc.

Thanh ghi DX phải được nạp địa chỉ cổng từ trước.

LEA (load effective address). Lệnh nạp địa chỉ hiệu dụng vào thanh ghi, nó không di chuyển nội dung chứa trong địa chỉ đó. Đây là lệnh để tính địa chỉ lệch hoặc địa chỉ của ô nhớ chọn làm gốc rồi nạp vào thanh ghi đã chọn.

Cấu trúc lệnh: **LEA Đích, nguồn.**

Trong đó :

- Đích thường là một trong các thanh ghi BX, CX, DX, BP, SI, DI.
- Nguồn là tên biến trong đoạn DS được chỉ rõ trong lệnh hoặc ô nhớ cụ thể.

Ví dụ:

```
LEA DX, MSG          ; Nạp địa chỉ lệch của bản tin MSG vào DX.
LEA CX, [BX] [DI] ; Nạp vào CX địa chỉ hiệu dụng do  BX và DI chỉ ra: EA=BX+DI.
```

PUSH/POP Thanh ghi ngăn xếp là nơi rất thuận tiện để cất giữ tạm dữ liệu và các toán hạng cần nhớ của chương trình.

Ví dụ, một chương trình có thể muốn cất lại các nội dung trong thanh ghi AX để dùng trong một số thao tác sau này. Để thực hiện nhiệm vụ đó có thể dùng các lệnh **PUSH** và **POP**.

- PUSH Cất dữ liệu vào ngăn xếp.

Cấu trúc lệnh: **PUSH nguồn**

Mô tả: SP ← SP - 2

Nguồn → {SP}.

Trong đó toán hạng gốc có thể tìm được theo các chế độ địa chỉ khác nhau: có thể là các thanh ghi đa năng, thanh ghi đoạn hoặc ô nhớ. Lệnh này thường dùng với lệnh POP như một cặp đối ngẫu để xử lý các dữ liệu và trạng thái của chuương trình chính khi vào/ra chương trình con.

Ví dụ:

```
PUSH BX              ; cất BX vào ngăn xếp, tại vị trí do SP chỉ ra.
PUSH Table[BX]    ; cất 2 byte của vùng dữ liệu DS có địa chỉ đầu tại (Table+BX).
```

- POP Lấy dữ liệu từ ngăn xếp.

Cấu trúc lệnh: **POP Đích**

Mô tả: Đích → {SP}.

SP ← SP + 2

Trong đó toán hạng gốc có thể tìm được theo các chế độ địa chỉ khác nhau: có thể là các thanh ghi đa năng, thanh ghi đoạn (nhưng không được là thanh ghi đoạ mã CS) hoặc ô nhớ.

Dữ liệu để tại ngăn xếp không thay đổi. Giá trị của SS không thay đổi.

Ví dụ:

POP DX ; lấy 2 byte từ đỉnh ngăn xếp, đưa vào DX.

PUSH Table[BX] ; lấy 2 byte ở đỉnh ngăn xếp rồi để tại vùng DS có địa chỉ đầu tại (Table+BX).

PUSHF/POPF Các nội dung của thanh ghi cờ có thể được gửi vào hay lấy ra khỏi ngăn xếp bằng các lệnh **PUSPF** và **POPF**.

- PUSHF Cất nội dung thanh ghi cờ vào ngăn xếp.

Cấu trúc lệnh: **PUSHF**

Mô tả: SP ← SP - 2

RF → {SP}.

Dữ liệu để tại thanh ghi cờ không thay đổi. SS không thay đổi.

- POPF Lấy 1 từ, từ đỉnh ngăn xếp đưa vào thanh ghi cờ.

Cấu trúc lệnh: **POPF**

Mô tả: RF → {SP}.

SP ← SP + 2

Sau lệnh này dữ liệu để tại ngăn xếp không thay đổi. SS không thay đổi.

2.3.2 Nhóm lệnh số học.

Các lệnh số học bao gồm bốn phép tính số học cơ bản là cộng, trừ , nhân, chia và đảo dấu toán hạng.

ADD/SUB Cộng (add)/trừ (subtract):

Cấu trúc tổng quát của các lệnh cộng, trừ là:

ADD đích, nguồn

SUB đích, nguồn

Mô tả: ADD: Đích ← Đích + Nguồn

SUB : Đích ← Đích -Nguồn

Trong đó các toán hạng đích, nguồn có thể tìm được theo các địa chỉ khác nhau, nhưng phải chứa dữ liệu có cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn.

Bảng sau tóm tắt các loại khác nhau của các toán hạng đích và nguồn dùng trong các lệnh cộng và trừ:

Đích (nơi đến)	Nguồn (gốc)
Thanh ghi	Thanh ghi
Thanh ghi	Bộ nhớ
Bộ nhớ	Thanh ghi
Bộ nhớ	Tức thời (hằng số)
Thanh ghi	Tức thời(hằng số)

Bảng 2.2 Các dạng toán hạng trong lệnh ADD/SUB

Ví dụ 1:

```
ADD AX, BX          ;  AX ← AX+BX
ADD AL, 74H         ;  AX ← AX+ 74H
SUB CL, AL          ; CL ← CL - AL
SUB AX, 0405H       ; AX ← AX - 0405H.
```

Ví dụ 2: Viết đoạn chương trình ngôn ngữ assembly để cộng 5H với 3H, dùng các thanh ghi AL, BL.

```
MOV AL, 05H     ; AL ← 05H
MOV BL, 03H     ; BL ← 03H
ADD AL, BL; AL ← 05H+03H =08H
MOV 100H, AL    ; Di chuyển kết quả từ AL vào vị trí nhớ DS:100H.
```

MUL/DIV Lệnh nhân (multiply, MUL) và chia (divide, DIV):

Cấu trúc lệnh: **MUL số nhân nguồn**

DIV số chia nguồn

Trong đó số nhân nguồn (toán hạng gốc) có thể tìm được theo các chế độ địa chỉ khác nhau. Khi dùng lệnh nhân, số được nhân phải được chuyển vào thanh ghi AX hoặc AL. Còn số nhân thì có thể chuyển vào thanh ghi khác bất kỳ hoặc một địa chỉ nhớ.

Ví dụ 3:

```
MUL BX          ; số nhân nằm trong thanh ghi BX
MUL MEM1        ; số nhân nằm trong địa chỉ nhớ mang nhãn MEM1
```

Khi hai byte nhân với nhau thì kết quả được gửi lưu vào thanh ghi AX.

Ví dụ 4: Viết đoạn chương trình nhân 5H với 3H, dùng thanh ghi CL.

```
MOV AL, 05H     ; AL ← 05H (số được nhân)
MOV CL, 03H     ; CL ← 03H (số nhân)
```

```
MUL CL          ; AL ← 0FH (kết quả)
MOV MEM1, AL    ; chuyển kết quả (0FH) từ AL vào vị trí nhớ có nhãn MEM1.
```

Khi nhân hai từ (16 bit) với nhau thì số được nhân phải chuyển vào thanh ghi AX, còn số nhân có thể ở trong một thanh ghi khác bất kỳ hoặc trong vị trí nhớ 16 bite. Kết quả sẽ là con số 32 bit (hoặc hai từ) và được chứa trong các thanh ghi DX và AX. Từ có trọng số lớn sẽ ở trong thanh ghi DX và từ có trọng số nhỏ sẽ ở trong thanh ghi AX.

Ví dụ 5: Viết đoạn chương trình để nhân 3A62H với 2B14H.

```
MOV AX, 3A62H       ; AX ← 3A62H
MOV CX, 2B14H       ; CX ← 2B14H
MUL CX              ; DXAX ← tích = 289C63A8H
```

Các lệnh chia, về cơ bản, cũng giống như các lệnh nhân. Trong phép chia cỡ byte, số chia là một byte có thể ở trong một thanh ghi hoặc một vị trí nhớ. Số bị chia phải là một số không dấu 16 bit chứa trong thanh ghi AX. Kết quả thương số sẽ ở trong thanh ghi AL, còn số dư thì ở trong thanh ghi AH. Đối với phép chia cỡ từ thì số chia 16 bit có thể đặt trong thanh ghi hoặc một vị trí nhớ. Còn số bị chia phải là một số không dấu 32 bit được đặt trong các thanh ghi DX và AX. Thanh ghi DX sẽ giữ từ có trọng số cao, thanh ghi AX sẽ giữ từ có trọng số thấp. Kết quả thương đặt trong thanh ghi AX, còn số dư đặt trong thanh ghi DX.

Ví dụ 6: Viết đoạn chương trình để chia 6H cho 3H, dùng thanh ghi CL.

```
MOV AX, 0006H   ; AX ← 6H
MOV CL, 03H     ; CL ← 3H
DIV CL          ; AH ← 00H (số dư), AL chứa 02H (thương số)
```

Chú ý: 6H được đưa vào thành 0006H để lấp đầy toàn bộ thanh ghi AX. Như vậy các byte trọng số cao của AX sẽ bị xoá để tránh bị lỗi.

Ví dụ 7: Viết đoạn chương trình để chia 1A034H cho 1002H, dùng thanh ghi BX

```
MOV AX, 0A034H ; AX ← 0A034H
MOV DX, 0001H       ; DX ← 0001H
MOV BX, 1002H       ; BX ← 1002H
DIV BX              ; DXAX ← 00H (số dư)1AH (thương số)
```

INC/DEC Lệnh tăng (increment) và giảm (decrement).

Lệnh tăng sẽ cộng thêm một đơn vị vào toán hạng, còn lệnh giảm sẽ trừ một đơn vị vào toán hạng. Các lệnh này rất cần đối với thao tác đếm.

Cấu trúc tổng quát của các lệnh INC và DEC là:

INC đích Mô tả: Đích ← Đích +1

DEC đích Mô tả: Đích ← Đích -1

Toán hạng đích có thể là một thanh ghi hoặc một vị trí nhớ bất kỳ, có thể là một từ 16 bit hoặc 1 byte; có thể tìm được theo các chế độ địa chỉ khác nhau.

Chú ý:

- Trong lệnh tăng, nếu Đích = FFH (hoặc FFFFH) thì Đích + 1 = 00H (hoặc 0000H) mà không ảnh hưởng đến cờ nhớ. Lệnh này cho kết quả tương đương như lệnh ADD Đích, 1 nhưng chạy nhanh hơn.

- Trong lệnh giảm, nếu đích là 00H (hoặc 0000H) thì Đích -1 = FFH (hoặc FFFFH) mà không ảnh hưởng đến cờ nhớ CF. Lệnh này cho kết quả tương đương với lệnh SUB Đích, 1 nhưng chạy nhanh hơn.

NEG- Negative a Operand (lấy bù hai của một toán hạng hay đảo dấu toán hạng).

Cấu trúc lệnh: **NEG Đích**

Ví dụ 8:

```
NEG AH              ; AH ← 0 - (AH)
NEG BYTE PTR[BX]    ; lấy bù 2 của ô nhớ do BX chỉ ra trong DS.
```

2.3.3 Nhóm lệnh logic.

Các lệnh logic nhằm thực hiện các phép tính Boolean NOT, AND và OR. Lệnh NOT thì đảo tất cả các bit trong toán hạng (byte boặc từ). Các lệnh AND/OR thực hiện các phép tính AND/OR đối với một đôi bit trong toán hạng nguồn và toán hạng đích. Các lệnh này có thể dùng với các toán hạng cỡ từ hoặc cỡ byte.

NOT Lấy bù của một toán hạng, đảo bit của một toán hạng.

Cấu trúc lệnh: **NOT Đích.**

Mô tả: **Đích ← (Đích)**

Trong đó toán hạng đích có thể tìm được theo các chế độ địa chỉ khác nhau. Lệnh này không tác động đến cờ.

Ví dụ 1: Xác định kết quả của đoạn chương trình sau:

```
MOV BL, 00110011B
NOT BL
MOV MEM1, BL
```

Nội dung của thanh ghi BL được nạp vào là 00110011B. Sau khi thực hiện phép NOT thì nội dung của thanh ghi BL là 11001100B và giá trị này được đưa vaò vị trí nhớ được chỉ ra bởi nhãn MEM1.

AND/OR: Và/Hoặc hai toán hạng.

Cấu trúc tổng quát của lệnh AND/OR là:

AND Đích, Nguồn

OR Đích, Nguồn

Trong đó toán hạng đích và nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn.

AND/OR sẽ thực hiện phép tính Boolean đối với các toán hạng nguồn và đích. Phép AND thường dùng để che đi/giữ lại một vài bit nào đó của một toán hạng bằng cách nhân logic toán hạng đó với toán hạng tức thời có các bit 0/1 tại các vị trí cần che/giữ lại tương ứng. Phép OR thường dùng để lập một vài bit nào đó của toán hạng bằng cách cộng lôgic toán hạng đó với toán hạng tức thời có các bit một tại các vị trí tương ứng cần thiết lập (toán hạng tức thời trong những trường hợp này còn được gọi là mặt nạ).

Ví dụ 2:

AND AL, BL ; nội dung thanh ghi BL được giao với nội dung trong thanh ghi AL và kết quả được lưu trong thanh ghi AL(AX). Nếu con số trong AL là 00001101B và trong BL là 00110011B thì kết quả trong thanh ghi AL sau phép AND là: **AL 0000001B**.

OR AL, BL ; nội dung thanh ghi BL được hợp với nội dung trong thanh ghi AL từng bit một và kết quả được lưu trong thanh ghi AL(AX). Nếu con số trong AL là 00001101B và trong BL là 00110011B thì kết quả trong thanh ghi AL sau phép AND là: AL 0011111B.

Ví dụ 3:

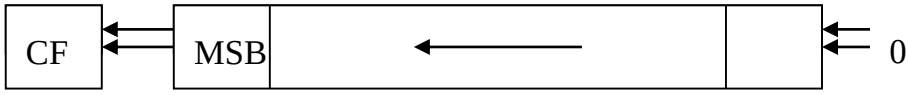
```
AND BL, 0FH  ; che 4 bit cao của BL.
OR BL, 30H   ; lập 4 bit b4 và b5 của BL lên 1.
```

SAL- Shift arithmetically Left (Dịch trái số học)/ **SHL- Shift (Logically) Left** (Dịch trái logic).

Cấu trúc lệnh: **SAL Đích, CL**

SHL Đích, CL

Mô tả:



Mỗi lần dịch MSB sẽ được đưa qua cờ CF và 0 được đưa vào LSB. Thao tác kiểu này được gọi là dịch logic. CL phải được chứa sẵn số lần dịch mong muốn. Thực chất mỗi lần dịch trái tương đương với một lần làm phép nhân với hai của số không dấu. Vì vậy ta có thể làm phép nhân số bị nhân không dấu với 2^i bằng cách dịch trái số học số bị nhân i lần. Chính vì vậy thao tác này còn được gọi là dịch trái số học.

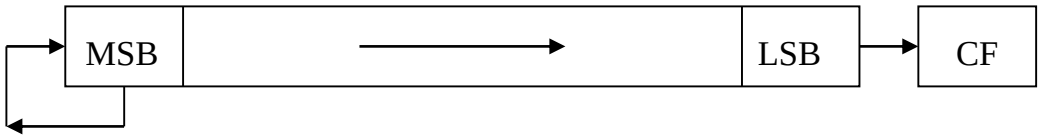
Sau lệnh SAL/SHL, cờ CF mang giá trị cũ của MSB, vì vậy lệnh này cò dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho các lệnh nhảy có điều kiện. Còn cờ OF ← 1 nếu sau khi dịch một lần mà bit MSB bị thay đổi so với trước khi dịch, cờ này không được xác định sau nhiều lần dịch.

Lệnh này cập nhật các cờ SF, ZF, PF. Trong đó PF chỉ có ý nghĩa khi toán hạng là 8 bit; cờ AF không xác định.

SAR - Shift Arithmetically Right (Dịch phải số học).

Cấu trúc lệnh: **SAR Đích, CL**

Mô tả:

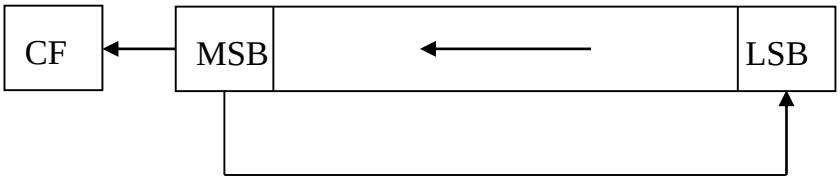


Sau mỗi lần dịch phải, MSB được giữ nguyên (nếu đây là bit dấu thì dấu luôn không đổi sau các lần dịch. Còn LSB được đưa vào cờ CF, CL phải được chứa sẵn số lần dịch mong muốn. Kiểu dịch này tương đương với một lần chia cho hai của số có dấu. Vì vậy có thể thay phép chia cho hai

ROL - Rotate All Bit to the Left (Quay vòng sang trái).

Cấu trúc lệnh: **ROL Đích, CL**

Mô tả:



Lệnh này dùng để quay toán hạng sang trái, MSB sẽ được đưa qua cờ CF và LSB. CL phải chứa số lần quay mong muốn.

Sau lệnh ROL cờ CF mang giá trị cũ của MSB, vì vậy lệnh này cò dùng để tạo cờ CF từ giá trị của MSB làm điều kiện cho các lệnh nhảy có điều kiện. Còn cờ OF ← 1 nếu sau khi dịch một lần mà bit MSB bị thay đổi so với trước khi dịch, cờ này không được xác định sau nhiều lần dịch. Lệnh này tác động vào các cờ CF, OF.

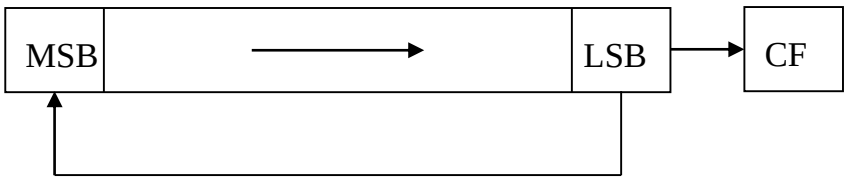
Ví dụ 4:

```
ROL BX, 1       ; quay vòng sang trái thanh ghi BX.
MOV CL, 4       ; đặt số lần quay vào thanh ghi CL.
ROL AL, CL ; quay vòng sang trái thanh ghi AL 4 lần.
```

ROR - Rotate All Bit to the Right (Quay vòng sang phải).

Cấu trúc lệnh: **ROR Đích, CL**

Mô tả:



Lệnh này dùng để quay toán hạng sang phải, LSB sẽ được đưa qua cờ CF và MSB. CL phải chứa số lần quay mong muốn.

2.3.4 Nhóm lệnh so sánh.

CMP - Compare Byte or Word (so sánh 2 byte hay 2 từ).

Cấu trúc lệnh: **CMP Đích, Gốc**

Trong đó toán hạng đích và gốc có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chữa dữ liệu có cùng độ dài và không được phép đồng thời là hai ô nhớ.

Lệnh này chỉ tạo các cờ, không lưu kết quả so sánh; sau lệnh so sánh, các toán hạng không bị thay đổi. lệnh này thường được dùng để tạo cờ cho các lệnh nhảy có điều kiện.

Các cờ chính theo quan hệ đích và nguồn khi so sánh hai số không dấu:

	CF	ZF
Đích = Nguồn	0	1
Đích > Nguồn	0	0
Đích < Nguồn	1	0

TEST - And Operands to Update Flag (và 2 toán hạng để tạo cờ).

Cấu trúc lệnh: **TEST Đích, Nguồn**

Trong đó toán hạng đích và nguồn có thể tìm được theo các chế độ địa chỉ khác nhau, nhưng phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ và cũng không được là thanh ghi đoạn. Sau lệnh này các toán hạng không bị thay đổi và kết quả không được lưu giữ. Các cờ được tạo ra sẽ được dùng làm điều kiện cho các lệnh nhảy có điều kiện. Lệnh này cũng có tác dụng che như một mặt nạ.

Tác động: Xoá: CF, OF

Cập nhật: PF, SF, ZF (PF chỉ liên quan đến 8 bit thấp)

Không xác định: AF.

Ví dụ :

```
TEST AH, AL          ; Và AH với AL để tạo cờ.
TEST AH, 01H         ; Bit 0 của AH = 0?
TEST BP, [BX][DI]    ; Và BP với ô nhớ DS:BX+DI.
```

2.3.5 Nhóm lệnh điều khiển chương trình.

2.3.5.1 Lệnh nhảy

Lệnh nhảy không điều kiện: Lệnh này khiến bộ vi xử lý bắt đầu thực hiện một lệnh mới tại địa chỉ được mô tả trong lệnh.

Cấu trúc lệnh: **JMP Nhãn**

Lệnh mới bắt đầu tại địa chỉ ứng với nhãn. Chương trình dịch sẽ căn cứ vào vị trí nhãn để xác định giá trị dịch chuyển.

Lệnh nhảy có điều kiện: Lệnh này biểu diễn thao tác: nhảy (có điều kiện) tới nhãn, tức là chỉ thực hiện nhảy tới nhãn nếu điều kiện chỉ ra đúng. Nhãn phải nằm cách xa (dịch đi một khoảng) -128.. +127 byte so với lệnh tiếp theo sau lệnh nhảy có điều kiện. Chương trình

dịch sẽ căn cứ vào vị trí của nhãn để xác định giá trị dịch chuyển. Các lệnh này không tác động đến cờ. Người ta phân biệt các kiểu nhảy có điều kiện:

Nhảy theo kiểu không dấu:

JA/JNBE - Jump if Above/ Jump if Not Below or Equal.

Cấu trúc lệnh: JA Nhãn

JNBE Nhãn

JAE/JNB- Jump if Above or Equal/ Jump if Not Below.

Cấu trúc lệnh: JAE Nhãn

JNB Nhãn

JB/JNAE- Jump if Below/ Jump if Not Above or Equal.

Cấu trúc lệnh: JB Nhãn

JNAE Nhãn.

Ví dụ 1:

```
CMP AL, 10H     ; so sánh AL với 10H.
JA MEM1         ; nhảy đến nhãn MEM1 nếu AL cao hơn 10H.
JB MEM2         ;nhảy đến nhãn MEM2 nếu AL thấp hơn 10H.
```

Nhảy theo kiểu có dấu:

JG/JNLE- Jump if Greater than/ Jump if Not Less than or Equal.

Cấu trúc lệnh: JG Nhãn

JNLE Nhãn.

JGE/JNL- Jump if Greater than or Equal/ Jump if Not Less than.

Cấu trúc lệnh: JGE Nhãn

JNL Nhãn.

JL/JNGE- Jump if Less than/ Jump if Not Greater than or Equal.

JLE/JNG- Jump if Less than or Equal/ Jump if Not Greater than.

Nhảy theo kiểu đơn:

JE/JZ- Jump if Equal/ Jump if Zero.

JNE/JNZ- Jump if Not Equal/ Jump if Not Zero.

JC- Jump if Carry

JNC- Jump if Not Carry

JO- Jump if Overflow

JNO- Jump if Not Overflow

JS- Jump if Sign

JNS- Jump if Not Sign

JP/JPE- Jump if Parity/ Jump if Parity Even

JNP/JPO- Jump if Not Parity/ Jump if Parity Odd

2.3.5.2 Lệnh lặp:

Lệnh này dùng để lặp lại đoạn chương trình (bao gồm các lệnh nằm trong khoảng tử nhãn đến hết lệnh **LOOP Nhãn** cho đến khi số lần lặp CX=0. Điều này có nghĩa là trước khi vào vòng lặp, ta phải đưa số lần lặp mong muốn vào thanh ghi CX và sau mỗi lần thực hiện lệnh **LOOP Nhãn** thì CX tự động giảm đi một.

Nhãn phải nằm cách xa (dịch một khoảng) -128 byte so với lệnh tiếp theo sau lệnh LOOP. Lệnh này không tác động đến cờ.

Cấu trúc lệnh: **LOOP Nhãn**

Ví dụ 2:

```
        XOR AL, Al ; xoá AL
        MOV CX, 16; số lần lặp đưa vào CX
Lap:    INC AL          ; tăng AL lên 1
        LOOP Lap        ; lặp lại 16 lần, AL =16.
```

Lệnh JCXZ- Jump if CX is Zero (nhảy nếu CX = 0).

Cấu trúc lệnh: **JCXZ Nhãn**

Đây là lệnh nhảy có điều kiện tới nhãn nếu nội dung thanh đếm bằng 0 và không có liên hệ gì với cò ZF. Nhãn phải nằm cách xa (dịch đi một khoảng) -128.. +127 byte so với lệnh tiếp theo sau lệnh JCXZ. Chương trình dịch sẽ căn cứ vào vị trí nhãn để xác định giá trị dịch chuyển.

2.3.5.3 Lệnh gọi chương trình con CALL:

Lệnh này dùng để chuyển hoạt động của bộ vi xử lý từ chương trình chính (CTC) sang chương trình con (ctc). Nếu ctc ở cùng một đoạn mã với CTC thì ta có gọi gần. Nếu CTC và ctc nằm trong hai đoạn mã khác nhau thì ta có gọi xa. Gọi gần và gọi xa khác nhau về cách tạo địa chỉ trở về. Địa chỉ trở về là địa chỉ tiếp theo ngay sau lệnh CALL. Khi gọi gần thì chỉ cần cất IP của địa chỉ trở về, khi gọi xa thì phải cất cả CS và IP của địa chỉ trở về. Địa chỉ trở về được tự động cất vào ngăn xếp khi bắt đầu thựuc hiện lệnh gọi và được tự động lấy ra khi gặp lệnh trở về RET.

RET - Return from Procedure to Calling Program (Trở về CTC từ ctc).

Cấu trúc lệnh: **RET**

Khi gặp lệnh trở về RET, vi xử lý kết thúc ctc lấy lại địa chỉ trở về, bao gồm địa chỉ IP (trường hợp gọi gần) hoặc IP và CS (trong trường hợp gọi xa) của lệnh tiếp theo sau lệnh CALL, được đặt trong ngăn xếp.

INT - Interrupt Program Excution (Ngắt, gián đoạn chương trình đang chạy).

Cấu trúc lệnh: **INT N, N = 0.. FFH**

Mô tả: Các thao tác của bộ vi xử lý khi chạy lệnh INT :

1. SP ← SP - 2. {SP} ← FR

2. IF ← 0 (cấm các ngắt khác tác động), TF ← 0 (chạy suốt).

3. SP ← SP - 2, {SP} ← CS.

4. SP ← SP - 2, {SP} ← IP.

5. {N x 4} → IP, {5N x 4 + 2} → CS.

Mỗi lệnh ngắt ứng với một chương trình phục vụ ngắt khác nhau có địa chỉ lấy từ bảng véc tơ ngắt. Bảng này gômg 256 vec tơ, chứa địa chỉ của các chương trình phục vụ ngắt tương ứng và chiếm 1 Kb RAM có địa chỉ thấp nhất. Ví dụ như các chương trình phục vụ ngắt của BIOS, của DOS như IO.SYS, MSDOS.SYS.

Ví dụ 3: INT 21H

2.3.6 Các lệnh đặc biệt.

NOP - No Operation (CPU không làm gì)

Cấu trúc lệnh: **NOP**

Lệnh này không thực hiện một công việc gì ngoài việc làm tăng nội dung của IP và tiêu tốn ba chu kỳ đồng hồ. Nó thường được dùng để tính thời gian trễ trong các vòng trễ hoặc để chiếm chỗ cho các lệnh cần thêm vào chương trình sau này mà không làm ảnh hưởng đến độ dài chương trình. Các cờ không bị thay đổi.

STC - Set the Carry Flag (lập cờ nhớ)

Cấu trúc lệnh: **STC**

Mô tả: **CF←1 STC**

Thiết lập cờ nhớ bằng một và không ảnh hưởng đến các cờ khác. Các cờ bị thay đổi: CF=1.

2.4. Các phương pháp xác định địa chỉ

Toán hạng của lệnh có thể là một giá trị cụ thể nằm ngay trong lệnh, nội dung của thanh ghi hay nội dung của ngăn nhớ hoặc cổng vào ra.

Những phương pháp định địa chỉ hay còn gọi là chế độ định địa chỉ (addressing mod) được dùng để vi xử lý tìm ra (định vị, addressing) các toán hạng cần thiết cho một lệnh nào đó. Một bộ vi xử lý có thể có nhiều chế độ địa chỉ, các chế độ địa chỉ này được xác định ngay từ khi chế tạo bộ vi xử lý và sau này không thể thay đổi được. Họ vi xử lý Intel có sáu chế độ địa chỉ như sau:

1. Chế độ định địa chỉ tức thì
2. Chế độ định địa chỉ thanh ghi
3. Chế độ định địa chỉ trực tiếp
4. Chế độ định địa chỉ gián tiếp qua thanh ghi
4. Chế độ định địa chỉ gián tiếp qua ngăn nhớ
6. Chế độ định địa chỉ dịch chuyển

2.4.1 Định địa chỉ tức thì

Đây là phương pháp địa chỉ hóa đơn giản nhất. Trong chế độ địa chỉ này toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số và ta có thể tìm thấy toán hạng này ở ngay sau mã lệnh (chính vì vậy chế độ địa chỉ này gọi là chế độ địa chỉ tức thì). Ta có thể dùng chế độ này để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (trừ các thanh ghi đoạn và các thanh ghi cờ) hoặc vào bất kỳ ô nhớ nào trong đoạn dữ liệu.

Ưu điểm: Các tham số sẽ chứa dữ liệu của chính nó, vì vậy nó không yêu cầu cần phải truy cập bộ nhớ thêm khi thực hiện lệnh. Do vậy tốc độ sẽ nhanh. [1]

Nhược điểm: Giá trị của các tham số sẽ bị giới hạn bởi kích thước của lệnh.

Ví dụ:

```
MOV AX, 4EH          ; chuyển giá trị 4EH vào thanh ghi AX.
MOV AX, 0FFOH        ; chuyển 0FF0H vào thanh ghi AX
MOV DS, AX           ; để đưa vào DS.
MOV [BX], 4EH        ; chuyển 4EH vào địa chỉ ô nhớ DS:BX
ADD R1, 5            ; R1← R1 + 5
```

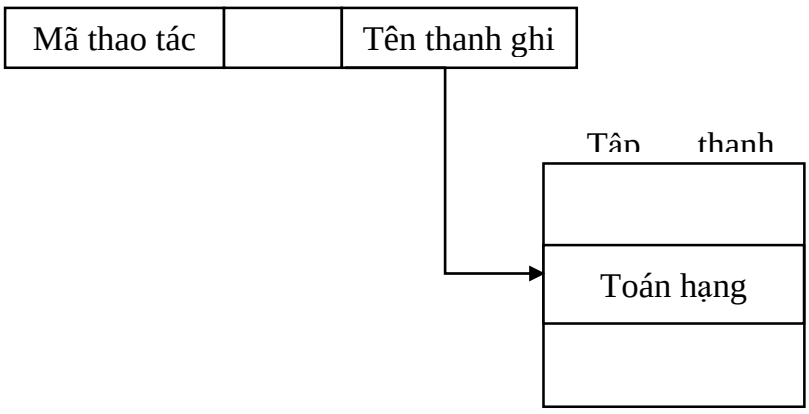
Hình 2.2 Phương pháp định địa chỉ tức thì

2.4.2 Định địa chỉ thanh ghi

Trong chế độ địa chỉ này người ta dùng các thanh ghi bên trong CPU như là các toán hạng để chứa dữ liệu cần thao tác. Vì vậy khi thực hiện lệnh có thể đạt tốc độ truy nhập cao hơn so với các lệnh có truy nhập đến bộ nhớ.

Ví dụ:

```
MOV AX, BX       ; chuyển nội dung BX vào AX.
ADD  DS, DL      ; cộng nội dung AL và DL , kết quả giữ trong AL.
ADD R1,R2    ; R1←R1+R2
```



Hình 2.3 Phương pháp định địa chỉ thanh ghi

2.4.3 Định địa chỉ trực tiếp

Trong phương pháp này trường tham số sẽ chứa địa chỉ đến vùng bộ nhớ chứa giá trị của tham số. tức là Trong chế độ địa chỉ này một toán hạng chứa địa chỉ lệch của ô nhớ dùng để chứa dữ liệu, còn toán hạng kia chỉ có thể là một thanh ghi mà không thể là một vị trí nhớ. Nếu tham số mà chứa địa chỉ của thanh ghi thì người ta gọi là địa chỉ hóa thanh ghi trực tiếp Register directed addressing

Nếu so sánh với chế độ địa chỉ tức thì ta thấy ở đây ngay sau mã lệnh không phải là một toán hạng mà là một địa chỉ lệch của toán hạng. Xét về phương diện địa chỉ thì đó là địa chỉ trực tiếp.

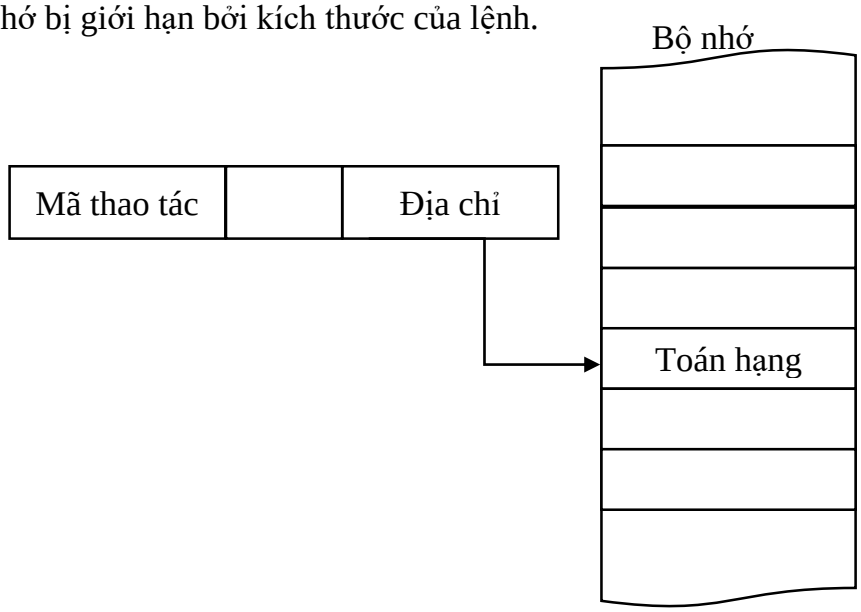
Ví dụ: MOV AL, [1234H] ; chuyển nội dung ô nhớ DS:1234H vào AL.

ADD R1,A ; R1←R1+(A) (Tìm toán hạng trong bộ nhớ có địa chỉ A)

MOV [4321H], CX ; chuyển nội dung CX vào hai vị trí nhớ liên tiếp là DS:4321 và DS:4322.

Ưu điểm : Khắc phục hạn chế về giá trị của dữ liệu

Nhược điểm : Phải cần thêm các phép truy nhập bộ nhớ để nạp tham số, không gian truy nhập bộ nhớ bị giới hạn bởi kích thước của lệnh.



Hình 2.4 Phương pháp định địa chỉ trực tiếp

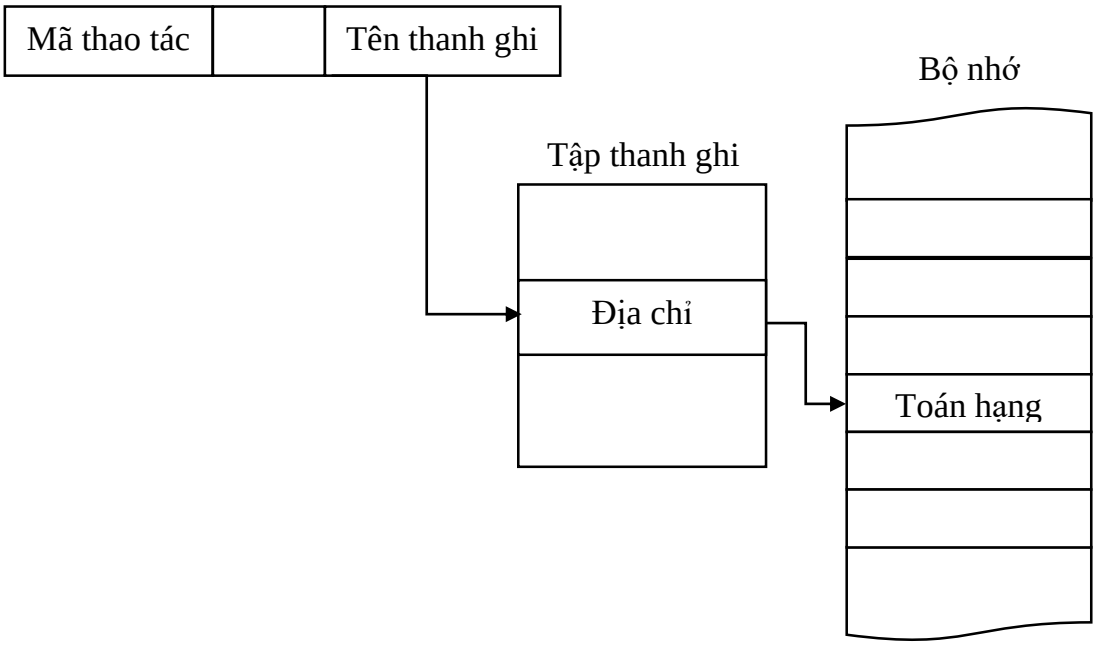
2.4.4 Định địa chỉ gián tiếp qua thanh ghi

Trong chế độ địa chỉ này một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệch của ô nhớ chứa dữ liệu, còn toán hạng kia chỉ có thể là một thanh ghi mà không được là ô nhớ.

Ví dụ: MOV AL, [BX] ; chuyển nội dung tại ô nhớ DS:BX vào AL.

MOV [SI], CL ; chuyển nội dung CL vào ô nhớ DS:SI.

MOV AL,[BX] ; (chuyển nội dung tại ô nhớ DS:BX vào AL)

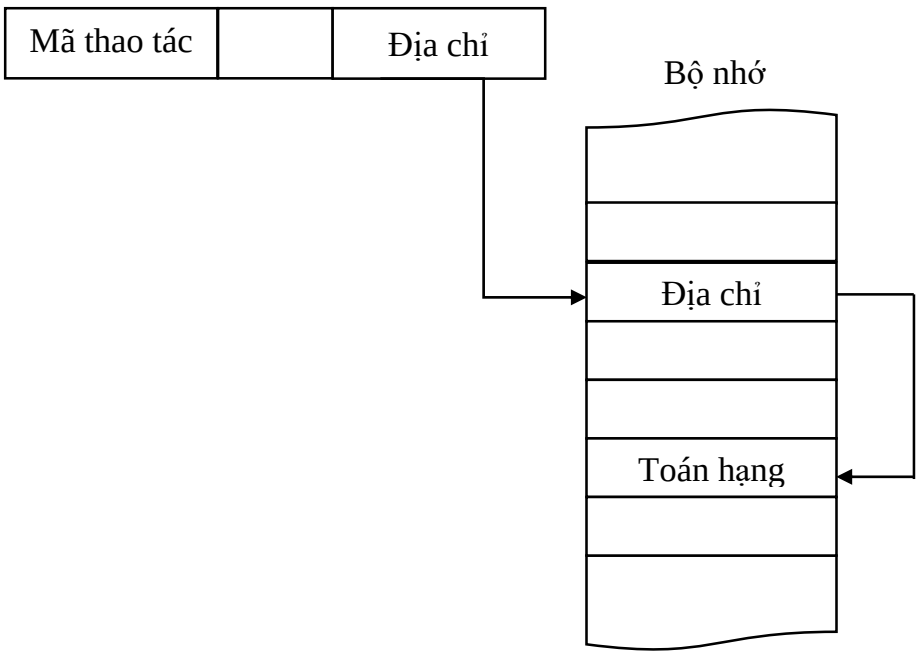


Hình 2.5 Phương pháp định địa chỉ gián tiếp qua thanh ghi

2.4.5 Định địa chỉ gián tiếp qua ngăn nhớ

Ngăn nhớ được trỏ bởi trường địa chỉ của lệnh chứa địa chỉ của toán hạng, có thể gián tiếp nhiều lần. Phương pháp định địa chỉ qua gián tiếp qua ngăn nhớ giống như khái niệm biến con trỏ và biến động trong lập trình. CPU phải thực hiện tham chiếu bộ nhớ nhiều lần để tìm toán hạng nên sẽ mất thời gian. Vùng nhớ có thể được tham chiếu là lớn.

Ví dụ: ADD AH, [a]



Hình 2.6 Phương pháp định địa chỉ gián tiếp qua ngăn nhớ

2.4.6 Định địa chỉ dịch chuyển

Để xác định toán hạng, trường địa chỉ chứa hai thành phần: tên thanh ghi và hằng số. Địa chỉ toán hạng = nội dung thanh ghi + hằng số. Thanh ghi có thể được ngầm định.

Các dạng của định địa chỉ dịch chuyển:

Địa chỉ hóa tương đối với PC: thanh ghi là bộ đếm chương trình, toán hạng có địa chỉ cách ngăn nhớ được trỏ bởi PC một độ lệch xác định.

Định địa chỉ cơ sở: sử dụng các thanh ghi cơ sở như BX và BP và các hằng số biểu diễn các giá trị dịch chuyển (displacement values) được dùng để tính địa chỉ hiệu dụng của toán hạng trong các vùng nhớ DS và SS. Sự có mặt của các giá trị diạch chuyển xác định tính tương đối (so với cơ sở) của địa chỉ.

Ví dụ:

MOV CL, [BX] + 10; chuyển nội dung 2 ô nhớ liên tiếp có địa chỉ DS:(BX+10) và DS:(BX+11) vào CX.

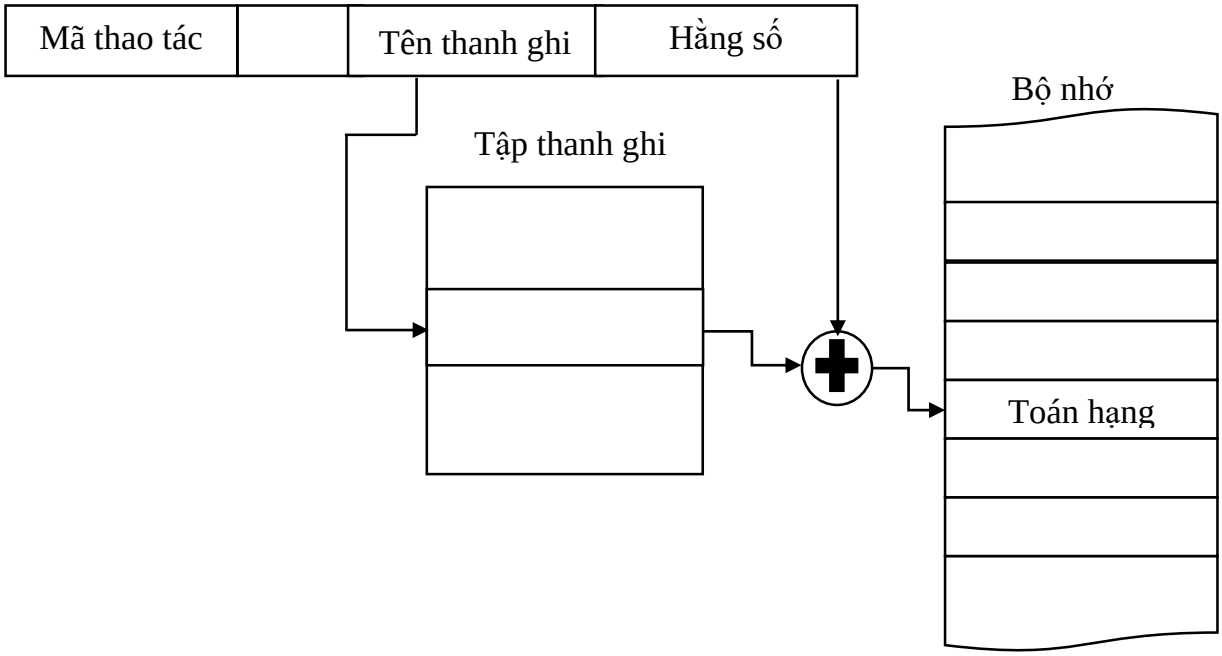
MOV AL, [BP] + 10; chuyển nội dung ô nhớ SS:(BP+10) vào AL

Định địa chỉ chỉ số: sử dụng các thanh ghi chỉ số như SI và DI và các hằng số biểu diễn các giá trị dịch chuyển (displacement values) được dùng để tính địa chỉ của toán hạng trong vùng nhớ DS.

Ví dụ:

MOV AL, [SI]+10 ; chuyển nội dung ô nhớ DS:(SI+10) vào AL.

MOV AL, [SI+10] ; tương tự như trên.



Hình 2.7 Phương pháp định địa chỉ dịch chuyển

2.5. Số lượng các tham số trong một lệnh

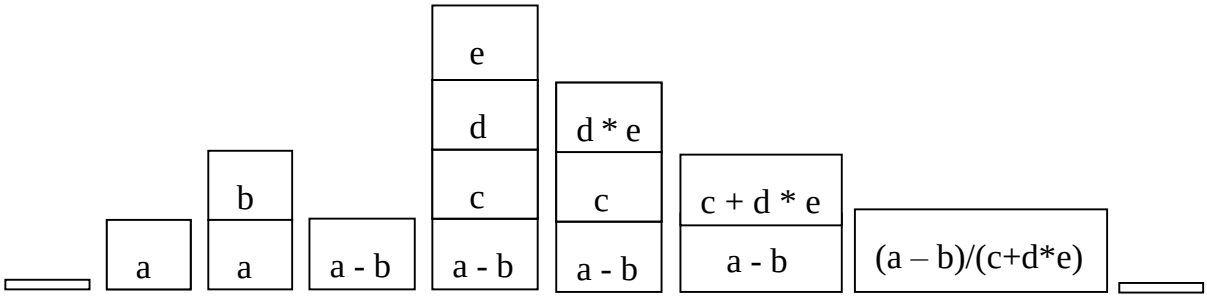
2.5.1 Hệ lệnh không có tham số

Loại máy này chỉ có hai lệnh một tham số đó là PUSH và POP. PUSH đưa dữ liệu vào stack, POP lấy dữ liệu từ stack ra). Còn các lệnh khác, không có tham số nào, việc tính toán các dữ liệu hoàn toàn dựa vào ngăn xếp.

Ví dụ: f = (a- b)/(c+d*e) được thực hiện qua hệ lệnh không tham số như sau.

Push a	Đưa a vào
Push b	Đưa b vào
Sub	Trừ không có tham số
Push c	Đưa c vào
Push d	Đưa d
Push e	Đưa e
Mul	Thực hiện nhân không có tham số
Add	Thực hiện cộng không có tham số
Div	Thực hiện chia không có tham số
Pop f	Đưa kết quả ra f

Trạng thái của stack như sau:



Ưu điểm: Đây là loại đơn giản, dễ chế tạo. Lệnh ngắn và ít mã máy.

Nhược điểm: Các lệnh không cho phép sử dụng các thanh ghi. Do vậy tốc độ chậm. Các quy trình tính toán không được tối ưu và người lập trình khó thực hiện.

2.5.2 Hệ lệnh một tham số

Loại máy sử dụng một thanh ghi dùng chung đóng vai trò làm tham số thứ hai cho hệ thống. Thanh ghi dùng chung này được gọi là thanh ghi tích lũy (accumulator Reg). Hệ lệnh này sử dụng hai lệnh:

LOAD - ACC: Nạp dữ liệu vào thanh ghi tích lũy.

STORE: ghi dữ liệu từ thanh ghi tích lũy ra bộ nhớ.

Ví dụ: Y= X+Z

LOAD - ACC X : Nạp X vào

ADD Z : Được ghi vào thanh ghi tích lũy

STORE Y : Đưa Y từ thanh ghi tích lũy ra bộ nhớ.

Ưu điểm: Lệnh ngắn và làm tối thiểu trạng thái bên trong máy.

Nhược điểm: cho phép sử dụng ít thanh ghi nên trong quá trình tính toán hầu hết phải truy nhập bộ nhớ. Hệ lệnh một tham số thường sử dụng địa chỉ hóa trực tiếp nhưng gây hạn chế vùng không gian bộ nhớ truy nhập thấp.

Ví dụ: Tính giá trị biểu thức f = (a- b)/(c+d*e) bằng hệ lệnh một tham số.

Bài giải:

```
LOAD   d
MUL    e
STORE  X
LOAD   c
ADD    X
STORE  Y
LOAD   a
SUB    b
DIV    Y
STORE  f
```

2.5.3 Hệ lệnh hai tham số

Thường sử dụng hai thanh ghi làm nơi chứa toán hạng. Nếu lệnh có 2 tham số, thường một tham số vừa đóng vai trò là nguồn (giá trị đem tính toán) vừa đóng vai trò là đích (nơi chứa kết quả). Giá trị cũ của một toán hạng nguồn bị mất vì phải chứa kết quả.

Ví dụ: Y = X+ Z

```
LOAD R1, X  (nhập X vào thanh ghi 1)
LOAD R2, Z  (nhập Z vào thanh ghi 2)
ADD R1, R2
STORE Y, R1
```

Ưu điểm: sử dụng ít phép truy nhập bộ nhớ khi thực hiện lệnh. Dạng lệnh thường ngắn gọn vì người ta cần ít bit để địa chỉ hóa thanh ghi. Các lệnh thường có độ dài cố định và chúng có thời gian thực hiện như nhau. Các chương trình sẽ dễ dàng sinh ra các mã lệnh tối ưu.

Nhược điểm: do phần lớn các lệnh đều phải truy nhập thanh ghi nên hạn chế trong việc áp dụng các phương pháp trong địa chỉ hóa. Phải dùng nhiều lệnh để thực hiện một thao tác đơn giản.

Chú ý: Với các bộ xử lý hệ lệnh gồm hai tham số người ta còn thiết kế các lệnh vừa có thể kết hợp các thanh ghi và địa chỉ bộ nhớ trong cùng một lệnh.

Ví dụ: LOAD R1, X
ADD R1, Z
STORE y, R1

Với cách thức này: ưu điểm là có thể sử dụng nhiều phương pháp địa chỉ hóa khác nhau và có thể truy nhập trực tiếp bộ nhớ để lấy dữ liệu mà không cần phải thông qua thanh ghi. Nhưng nhược điểm là kích thước của lệnh thường thay đổi do vậy rất phức tạp trong quá trình thiết kế. Phần lớn thời gian thực hiện lệnh là do truy nhập bộ nhớ để lấy dữ liệu. Sử dụng ít thanh ghi nên tốc độ chậm.

Ví dụ: Viết đoạn chương trình thực hiện tính giá trị biểu thức f = (a - b)/(c + d * e) bằng hệ lệnh hai tham số:

Bài giải:

Hai tham số

MOV AX, a
SUB AX, b
MOV DX, d
MUL DX, e
MOV CX, c
ADD CX, DX
DIV AX, CX
STORE f, AX

2.5.4 Hệ lệnh ba tham số

Thường sử dụng các thanh ghi hoặc ngăn nhớ làm nơi chứa toán hạng. Hệ lệnh ba tham số có hai toán hạng nguồn và một toán hạng đích. Sau khi thực hiện xong các lệnh, kết quả được lưu trên thanh ghi hoặc bộ nhớ

Ưu điểm: Kiểu rất tổng quát để tạo các mã hữu hạn.

Nhược điểm: Phần lớn thời gian thực hiện lệnh phải dành cho việc truy nhập bộ nhớ và kích thước của lệnh thay đổi dẫn đến thiết kế bộ giải mã khó khăn.

Ví dụ:

Viết đoạn chương trình thực hiện tính giá trị biểu thức f = (a - b)/(c + d * e) bằng hệ lệnh ba tham số:

Ba tham số

SUB Y,a,b

MUL T,d,e

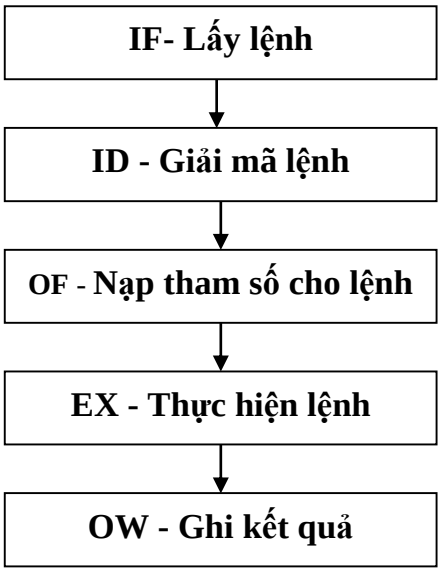
ADD X,c,T

DIV f,Y,X

2.6. Quy trình thực hiện lệnh

Quy trình thực hiện một lệnh thông thường được chia làm năm giai đoạn chính:

1) Lấy lệnh (Instruction fetch- IF).
2) Giải mã lệnh (Instruction decording- ID).
3) Nạp các tham số cho lệnh (Operands Fetch-OF).
4) Xử lý lệnh (Excute - EX)
5) Ghi kết quả (Operand Write-OW).



Hình 2.8 Quy trình thực hiện lệnh

Giải thích sơ đồ: [1]

Lấy lệnh (Instruction fetch): Trước hết chương trình và số liệu ban đầu được đưa vào bộ nhớ trong (RAM), các chương trình và số liệu này được lưu trữ dưới dạng nhị phân. Khi bắt đầu thi hành chương trình, lệnh đầu tiên sẽ được nạp từ bộ nhớ vào thanh ghi lệnh instruction register- IR qua các bus, đồng thời nó sẽ gán địa chỉ của lệnh kế tiếp vào bộ đếm chương trình - program counter.

Giải mã lệnh (Instruction decording): Trong bước này khối điều khiển CU sẽ giải mã lệnh và nhận biết các lệnh. Nếu lệnh cần một hay nhiều toán hạng thì CU sẽ xác định xem toán hạng đó nằm ở đâu trong bộ nhớ, công việc này thường gọi là tính địa chỉ các toán hạng và sang bước ba.

Nạp các tham số cho lệnh (Operands Fetch): Sau khi xác định địa chỉ của các toán hạng xong, CU sẽ phát ra tín hiệu điều khiển để lấy dữ liệu từ bộ nhớ nạp vào các thanh ghi tương ứng với các tham số của lệnh để chuẩn bị tính toán.

Xử lý lệnh (excute): Thực hiện lệnh. Sau khi nạp các lệnh, CU sẽ tùy theo loại lệnh để tiến hành thực hiện. Nếu lệnh là lệnh tính toán thì CU phát tín hiệu điều khiển tới ALU (khối tính toán) thực hiện các phép toán trên toán hạng lấy về, Nếu lệnh là lệnh điều khiển thì CU sẽ sinh ra các tín hiệu điều khiển tương ứng.

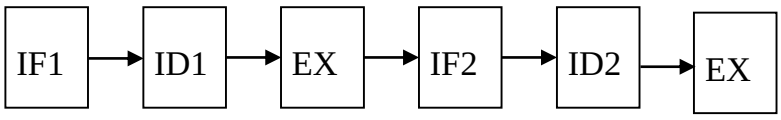
Và ghi kết quả (Operand Write): Sau khi thực hiện lệnh xong tùy theo yêu cầu của lệnh mà các kết quả thực hiện sẽ được lưu trữ trong thanh ghi, ổ cứng hay thiết bị ngoại vi. Sau khi thực hiện lưu trữ xong CPU sẽ chuyển sang lệnh kế tiếp và lại thực hiện tuần tự như trên.

Chú ý: Những bộ vi xử lý cổ điển 8 bit tiến hành năm giai đoạn trên một cách tuần tự - nghĩa là chỉ có một dòng chảy của lệnh và một bộ nhớ chứa dữ liệu của nó, lệnh đầu tiên được nhận từ bộ nhớ về rồi được thực hiện, sau đó đến lệnh tiếp theo và cứ tiếp tục như vậy.

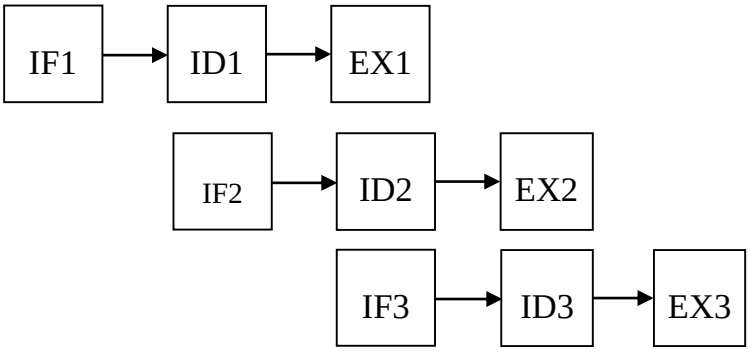
Từ các bộ vi xử lý 16 bit trở đi, bộ vi xử lý dùng pipeline (xen kẽ dòng lệnh) để tiết kiệm thời gian xử lý - đây chính là thực hiện cơ chế song song ở một chừng mực nhất định bằng cách khi đang thực hiện một lệnh thì lấy về lệnh tiếp theo..

Nếu chỉ quan tâm đến ba giai đoạn chính IF, ID, EX. Ta có sơ đồ tuần tự và pipeline như sau:

a) Tuần tự



b) Pipeline



Hình 2.9 Sơ đồ quy trình thực hiện lệnh tuần tự và pipeline

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 2

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Nêu cấu trúc tổng quát của một lệnh và quy trình thực hiện một lệnh (không phải giải thích).

Áp dụng: sử dụng lệnh 0 và 2 toán hạng (tham số) để thực hiện biểu thức sau:

f = ((a*b)+(c*d))/e

Câu 2. Quy trình thực hiện một lệnh gồm mấy giai đoạn. Vẽ sơ đồ giải thích.

Sử dụng lệnh 1 và 3 toán hạng (tham số) để thực hiện biểu thức sau:

f = ((a*b)+(c*d))/e

Câu 3. Phương pháp xác định địa chỉ là gì?

Câu 4. Trình bày chế độ định địa chỉ tức thì? Cho ví dụ minh họa?

Câu 5. Trình bày chế độ định địa chỉ thanh ghi? Cho ví dụ minh họa?

Câu 6. Trình bày chế độ định địa chỉ trực tiếp? Cho ví dụ minh họa?

Câu 7. Trình bày chế độ định địa chỉ gián tiếp qua thanh ghi? Cho ví dụ minh họa?

Câu 8. Trình bày chế độ định địa chỉ gián tiếp qua ngăn nhớ? Cho ví dụ minh họa?

Câu 9. Trình bày chế độ định địa chỉ dịch chuyển? Cho ví dụ minh họa?

Câu 10. Trình bày rõ các bước trong quy trình thực hiện lệnh?

Câu hỏi trắc nghiệm

Câu 1. Cấu trúc chung của một mã lệnh gồm:

A. Mã toán, Toán hạng
C. Toán hạng, Địa chỉ trực tiếp
B. Tiền tố, Mã toán, Toán hạng
D. Tiền tố, Mã toán, Toán hạng, Địa chỉ trực tiếp

Câu 2. Bộ đếm chương trình PC có nhiệm vụ:

A. Giữ địa chỉ của lệnh tiếp theo sẽ được nhận vào
C. Chứa địa chỉ của ngăn nhớ dữ liệu
B. Chứa địa chỉ của ngăn nhớ đỉnh ngăn xếp
D. Giữ địa chỉ của lệnh đã thực hiện

Câu 3. Ngăn xếp (stack) là vùng nhớ có cấu trúc:

A. ILO B. FIFO C. FILO D. LIFO

Câu 4. Trong chế độ địa chỉ gián tiếp thanh ghi, địa chỉ của toán hạng được chứa trong:

A. Ngăn xếp B. Bộ nhớ C. Lệnh D. Thanh ghi

Câu 5. Phương pháp định địa chỉ là gì?

A. Là cách thức địa chỉ hóa trong trường địa chỉ để xác định đỉnh của ngăn xếp

B. Là cách thức địa chỉ hóa trong trường địa chỉ để xác định địa chỉ của thanh ghi

C. Là cách thức địa chỉ hóa trong trường địa chỉ của lệnh để xác định nơi chứa toán hạng

D. Là cách thức địa chỉ hóa trong trường địa chỉ để xác định ngăn nhớ

Câu 6. Chế độ định địa chỉ gián tiếp thanh ghi là gì?

A. Toán hạng là ngăn nhớ có địa chỉ được chỉ ra trực tiếp trong trường địa chỉ của lệnh

B. Toán hạng là ngăn nhớ có địa chỉ nằm trong thanh ghi

C. Toán hạng nằm ngay trong trường địa chỉ của lệnh

D. Dùng các thanh ghi bên trong CPU như là các toán hạng để chứa dữ liệu cần thao tác

Câu 7. Hãy cho biết câu lệnh sau các toán hạng được xác định bởi chế độ địa chỉ nào?MOV AL, [1234H]

A. Chế độ địa chỉ tức thì
B. Chế độ địa chỉ trực tiếp
C. Chế độ địa chỉ gián tiếp qua thanh ghi
D. Chế độ địa chỉ thanh ghi

Câu 8. Trong hệ lệnh 1 tham số, một toán hạng được chỉ ra trong lệnh, còn một toán hạng ngầm định là:

A. Thanh ghi đoạn
B. Thanh ghi lệch
C. Thanh chứa Acc (Accumulator Reg)
D. Thanh chứa Bcc

Câu 9. Hãy chọn đoạn lệnh đúng để tính giá trị biểu thức c = a + b bằng hệ lệnh 2 toán hạng:

A.
```
load a
add b
store c
```

B.
```
load R1, a
add R1, b
store c
```

C.
```
add c, a, b
```

D.
```
push a
push b
add
pop c
```

Câu 10. Hình vẽ sau của chế độ định địa chỉ nào?

Mã thao tác		Toán hạng

A. Chế độ địa chỉ gián tiếp qua thanh ghi
B. Chế độ địa chỉ tức thì
C. Chế độ địa chỉ trực tiếp
D. Chế độ địa chỉ tương đối chỉ số

Bài tập áp dụng

Câu 1. Sử dụng các hệ lệnh 0, 1, 2, 3 tham số (toán hạng) để thực hiện biểu thức sau:

a) F1 = ((a*b+c)-d)/(a+b)
b) F2 = ((a*b+c)-d)/(a+b)
c) F3 = a/b*c+(d-e*g)
d) F4 = a+b-c*d/e

Câu 2. Nêu đặc điểm của các hệ lệnh 0, 1, 2, 3 tham số? Cho ví dụ minh họa?

Câu 3. Sử dụng các hệ lệnh 0, 1, 2, 3 tham số để thực hiện các biểu thức sau:

a) T1 = a / b * c
b) T2 = (a – b) / c + d

c) T3 = a + b / c * d
d) T4 = (a * b) /(c + d)

Câu 4. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ tức thì?

Câu 5. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ thanh ghi?

Câu 6. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ tức thì?

Câu 7. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ trực tiếp?

Câu 8. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ gián tiếp qua thanh ghi?

Câu 9. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ gián tiếp qua ngăn nhớ?

Câu 10. Vẽ sơ đồ khối thể hiện nguyên lý làm việc của phương pháp định địa chỉ dịch chuyển?

Chương 3: BỘ XỬ LÝ TRUNG TÂM CPU

***Mục đích**: Trình bày kiến trúc, cách tổ chức và chức năng của bộ xử lý trung tâm CPU. Phân tích các thành phần của bộ xử lý trung tâm CPU như: đơn vị số học và logic ALU, đơn vị điều khiển CU và tập các thanh ghi. Tìm hiểu kỹ thuật đường ống lệnh pipeline - là cơ sở để hiểu được các hoạt động xử lý lệnh trong các kỹ thuật ống dẫn, siêu ống dẫn, siêu vô hướng,... Kỹ thuật xử lý thông tin: song song mức lệnh. Bộ xử lý đa luồng và đa lõi.*

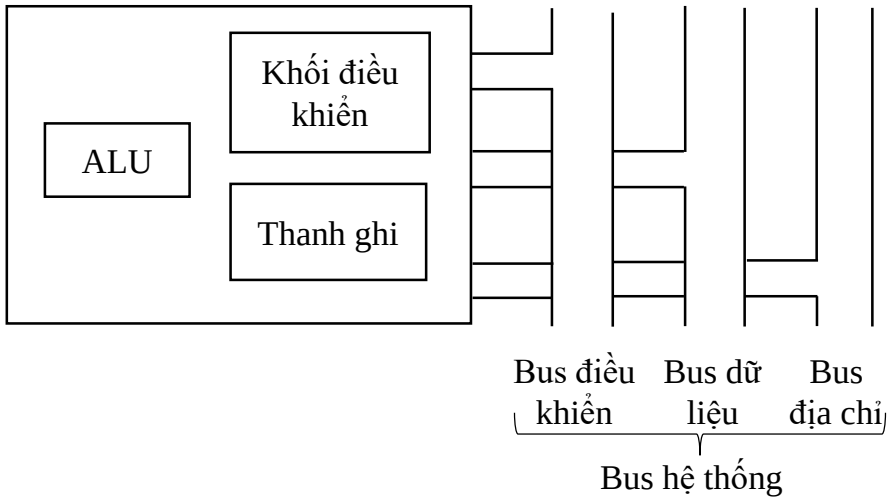
***Yêu cầu**: Sinh viên nắm được những kiến thức về bộ xử lý trung tâm CPU và các thành phần cấu tạo nên CPU. Hiểu được kỹ thuật đường ống lệnh - pipeline, áp dụng vào làm bài tập về hiệu năng của pipeline như: khắc phục khó khăn khi sử dụng pipeline, tính hệ số tăng tốc theo lý thuyết và theo thực tế của hệ thống pipeline.*

3.1 Tổ chức của CPU (Central Processing Unit)

Để hiểu hơn về tổ chức bộ xử lý, ta xét các nhiệm vụ mà nó phải thực hiện:

- **Truy xuất lệnh**: Bộ xử lý đọc lệnh từ bộ nhớ (thanh ghi, bộ nhớ cache, bộ nhớ chính).
- **Giải mã lệnh**: Lệnh được giải mã để xác định hành động nào được yêu cầu.
- **Truy xuất dữ liệu**: Việc thực thi một lệnh có thể yêu cầu đọc dữ liệu từ bộ nhớ hoặc một mô-đun I/O.
- **Xử lý dữ liệu**: Việc thực thi một lệnh có thể yêu cầu thực hiện một số phép tính số học hoặc logic trên dữ liệu.
- **Ghi dữ liệu**: Kết thúc việc thực hiện có thể yêu cầu ghi dữ liệu vào bộ nhớ hoặc một mô-đun I / O.

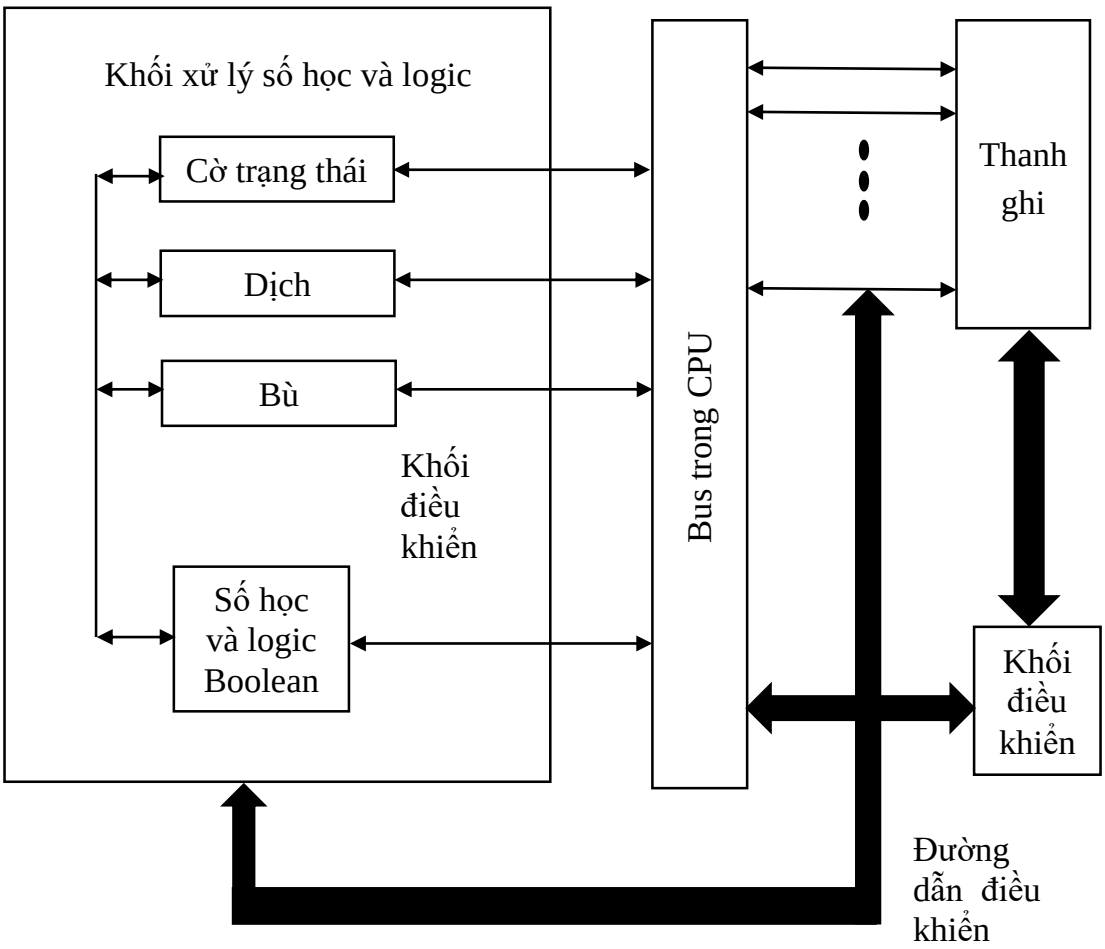
Để thực hiện những việc này, rõ ràng là bộ xử lý cần phải lưu tạm thời một số dữ liệu. Nó phải nhớ vị trí của lệnh gần nhất vừa thực thi để có thể biết được nơi truy xuất lệnh tiếp theo. Nó cần lưu trữ tạm thời các lệnh và dữ liệu trong khi một lệnh đang được thực thi. Nói cách khác, bộ xử lý cần một bộ nhớ nhỏ bên trong nó và đó chính là tập các thanh ghi.



Hình 3.1 CPU và bus hệ thống

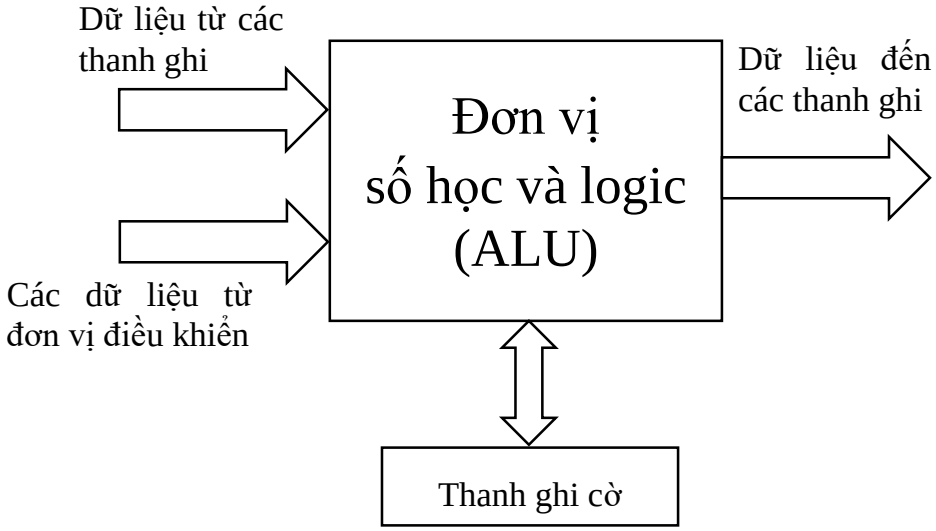
Hình trên là sơ đồ khối đơn giản của một bộ xử lý, cho thấy kết nối của nó với phần còn lại của hệ thống thông qua bus hệ thống. Các thành phần chính của bộ xử lý là khối số học và logic (ALU), khối điều khiển (CU), tập các thanh ghi (Registers). ALU thực hiện tính toán hoặc xử lý dữ liệu. Khối điều khiển kiểm soát việc di chuyển dữ liệu, lệnh vào ra khỏi bộ xử lý và điều khiển hoạt động của ALU. Ngoài ra, hình vẽ này cũng thể hiện bộ nhớ bên trong CPU, bao gồm một tập hợp các vị trí lưu trữ, được gọi là thanh ghi.

Hình bên dưới mô tả sơ đồ khối chi tiết hơn của bộ xử lý. Các đường truyền dữ liệu và các đường điều khiển logic được thể hiện, bao gồm một thành phần được gọi là bus trong bộ xử lý. Thành phần này là cần thiết để truyền dữ liệu giữa các thanh ghi khác nhau với ALU vì trên thực tế, ALU chỉ hoạt động trên dữ liệu nằm trong bộ nhớ nội bộ của bộ xử lý. Hình vẽ này cũng cho thấy các thành phần cơ bản điển hình của ALU. Lưu ý sự tương đồng giữa cấu trúc bên trong của máy tính và cấu trúc bên trong của bộ xử lý. Cả hai trường hợp đều có một tập hợp nhỏ vài thành phần chính (máy tính: bộ xử lý, I/O, bộ nhớ; bộ xử lý: khối điều khiển, ALU, thanh ghi) được kết nối với nhau bằng đường dẫn dữ liệu.



Hình 3.2 Cấu trúc bên trong CPU

Kiến trúc ALU:



Thanh ghi cờ: hiển thị trạng thái của kết quả phép toán

Hình 3.3 Mô hình kết nối ALU

Chức năng của đơn vị số học và logic là: thực hiện các phép toán số học và logic. ALU là thành phần của máy tính chịu trách nhiệm thực hiện các phép toán số học và logic trên các dữ liệu được cung cấp. Mọi thành phần khác như bộ điều khiển, các thanh ghi, bộ nhớ, thiết bị vào ra cung cấp dữ liệu để ALU xử lý và nhận dữ liệu đã xử lý từ đó. ALU thực chất là tổ hợp các vi mạch được thiết kế để lưu trữ dữ liệu và thực hiện các tính toán logic đơn giản. Các tín hiệu điều khiển từ đơn vị điều khiển xác định tác vụ mà ALU phải thực hiện, dữ liệu cần xử lý có thể được nạp từ nội dung của các thanh ghi. Kết quả xử lý sẽ được lưu vào thanh ghi hoặc bộ nhớ, hoặc đưa ra thiết bị ngoại vi. Kết quả xử lý của ALU cũng tác động lên các cờ trạng thái kết quả phép toán.

Như vậy, dữ liệu được cung cấp cho ALU từ các thanh ghi, và kết quả xử lý cũng được lưu trong các thanh ghi. Các thanh ghi này là các thiết bị lưu trữ tạm thời bên trong CPU và được kết nối với ALU. ALU cũng cho các tín hiệu cờ để ghi lại trạng thái của một hoạt động xử lý, ví dụ như tín hiệu cờ Overflow được đặt bằng một, khi kết quả tính toán phải biểu diễn dưới dạng có độ dài vượt quá độ dài của thanh ghi được sử dụng để lưu kết quả. Các giá trị cờ cũng được lưu bên trong CPU. Cuối cùng, bộ điều khiển cung cấp các tín hiệu điều khiển các hoạt động xử lý và di chuyển dữ liệu của ALU.[4]

Tập các thanh ghi (Registers):

Dùng để chứa thông tin tạm thời phục vụ cho các hoạt động hiện tại của CPU. Trong kiến trúc máy tính, một thanh ghi là một bộ nhớ dung lượng nhỏ và rất nhanh được sử dụng để tăng tốc độ xử lý của các chương trình máy tính bằng cách cung cấp các truy cập trực tiếp đến các giá trị cần dùng. Hầu hết, nhưng không phải tất cả, các máy tính hiện đại hoạt động theo nguyên lý chuyển dữ liệu từ bộ nhớ chính vào các thanh ghi, tính toán trên chúng, sau đó chuyển kết quả vào bộ nhớ chính

3.2 Thiết kế đơn vị điều khiển

Khái quát:

CU - Control Unit là đơn vị điều khiển, điều phối mọi hoạt động của các bộ phận chức năng trong CPU thông qua Control BUS. Có thể coi CU là khối dịch lệnh của CPU, nó tạo ra các tín hiệu tương ứng làm đầu vào cho Controller để điều khiển hoạt động của các khối chức năng. Các tín hiệu do CU tạo ra có thể phân thành hai loại: tín hiệu định thời và tín hiệu điều hành hoạt động của CPU. Các tín hiệu định thời do CU tạo ra xác định trạng thái của CPU làm việc:

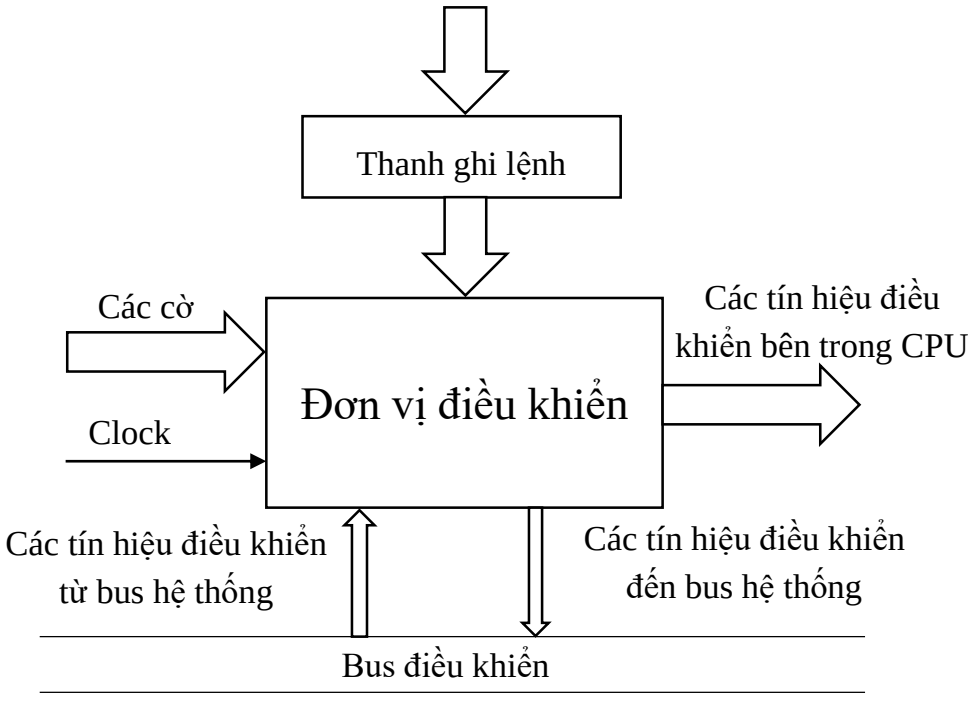
- Đang ở chế độ đọc dữ liệu vào (Input mode)
- Đang đưa dữ liệu ra (Output mode)
- Đang bắt đầu một tác vụ khác (Beginning another operation).

Các tín hiệu trạng thái của CPU xác định CPU đang:

- Đọc dữ liệu từ bộ nhớ (Memory Read)
- Ghi dữ liệu vào bộ nhớ (Memory Write)
- Nhận lệnh (Instruction Fetch)
- Đọc dữ liệu từ I/O (I/O Read)
- Đưa dữ liệu ra I/O (I/O Write)

Các tín hiệu để phân biệt các tác vụ trên gồm: IO/M, RD/W và DBIN, DBOT. Cần hiểu rằng việc sử dụng mạch Logic Controller tạo các tín hiệu điều khiển dựa vào các tín hiệu trạng thái của CPU và tín hiệu định thời, có nghĩa là tạo tín hiệu gì và vào thời điểm nào.

Mô hình kết nối đơn vị điều khiển:



Hình 3.4 Mô hình kết nối đơn vị điều khiển

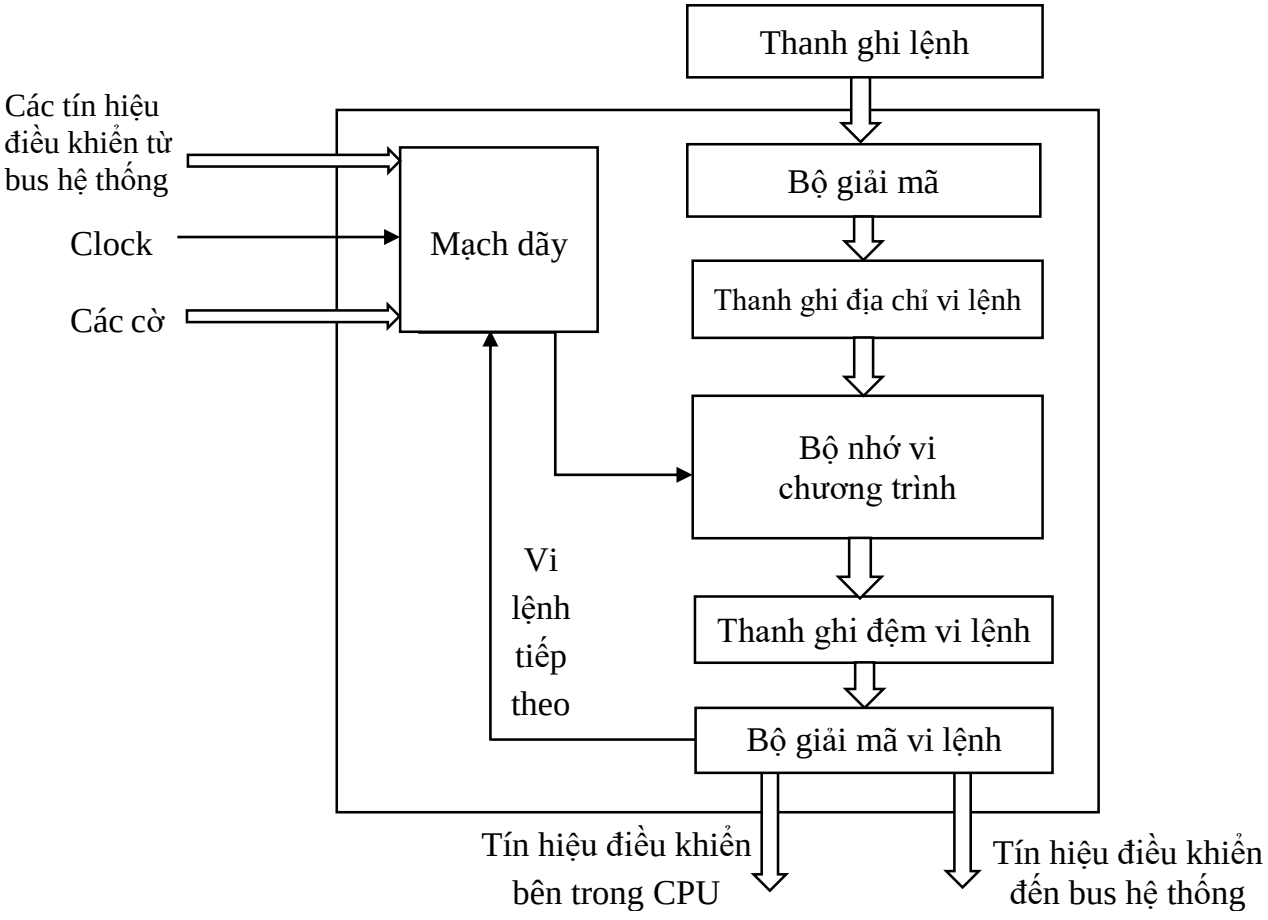
Chức năng:

- Điều khiển nhận lệnh từ bộ nhớ đưa vào bộ xử lý trung tâm CPU
- Tăng nội dung của PC để trỏ sang lệnh kế tiếp
- Giải mã lệnh đã được nhận để xác định thao tác mà lệnh yêu cầu
- Phát ra các tín hiệu điều khiển thực hiện lệnh
- Nhận các tín hiệu yêu cầu từ bus hệ thống và đáp ứng với các yêu cầu đó.

Các tín hiệu đưa đến đơn vị điều khiển bao gồm: tín hiệu Clock là tín hiệu nhịp từ mạch tạo dao động bên ngoài. Lệnh từ thanh ghi lệnh đưa đến để giải mã. Các cờ từ thanh ghi cờ cho biết trạng thái của bộ xử lý trung tâm CPU. Các tín hiệu yêu cầu từ bus điều khiển. Các tín hiệu phát ra từ đơn vị điều khiển bao gồm: các tín hiệu điều khiển bên trong bộ xử lý trung tâm CPU như điều khiển các thanh ghi, điều khiển khối tính toán số học và logic ALU; và các tín hiệu điều khiển bên ngoài bộ xử lý trung tâm CPU như điều khiển bộ nhớ, điều khiển các modul vào-ra.[4]

3.2.1 Thực hiện bằng vi chương trình

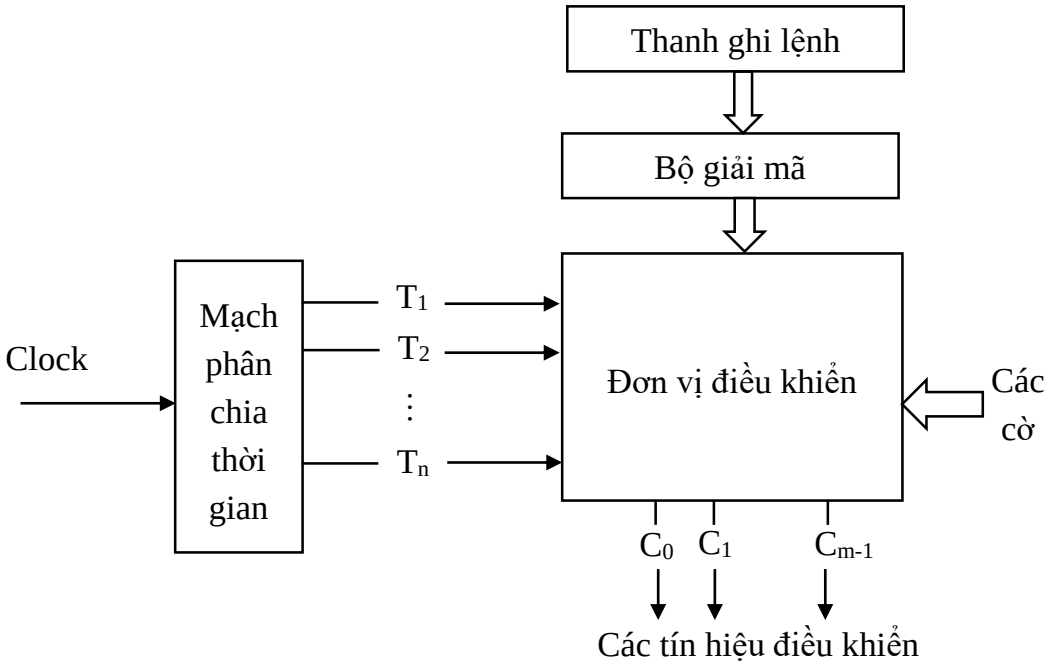
Bộ nhớ vi chương trình (ROM) lưu trữ các vi chương trình (microprogram). Một vi chương trình bao gồm các vi lệnh (microinstruction). Mỗi vi lệnh lại mã hóa cho một vi thao tác (microoperation). Để hoàn thành một lệnh cần thực hiện một hoặc một vài vi chương trình. Thực hiện bằng vi chương trình thì tốc độ thường chậm.



Hình 3.5 Đơn vị điều khiển vi chương trình

3.2.2 Thực hiện bằng kết nối cứng

Sử dụng mạch cứng để giải mã và tạo các tín hiệu điều khiển thực hiện lệnh. Thực hiện bằng kết nối cứng tốc độ sẽ nhanh hơn nhưng đơn vị điều khiển sẽ phức tạp hơn.

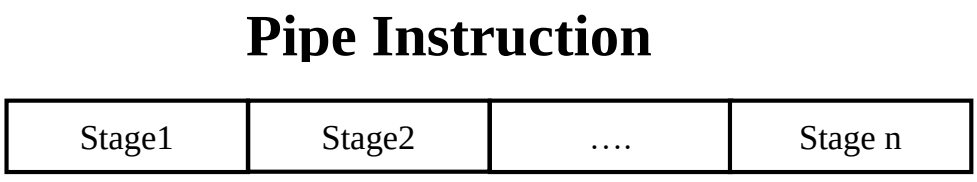


Hình 3.6 Đơn vị điều khiển kết nối cứng

3.3. Kỹ thuật đường ống Pipeline

Với những ý tưởng nhằm cải tiến hiệu suất cho các CPU, tăng tốc độ xử lý, kỹ thuật đường ống lệnh pipeline xử lý lệnh bằng việc gối chồng thực hiện nhiều lệnh đồng thời đã đáp ứng được những ý tưởng trên.

Pipelinning chia chu trình lệnh thành các công đoạn và cho phép thực hiện gối lên nhau theo kiểu dây chuyền. Các đoạn đó được gọi là pipe stage hay pipe segment



Hình 3.7 Công đoạn trong pipeline

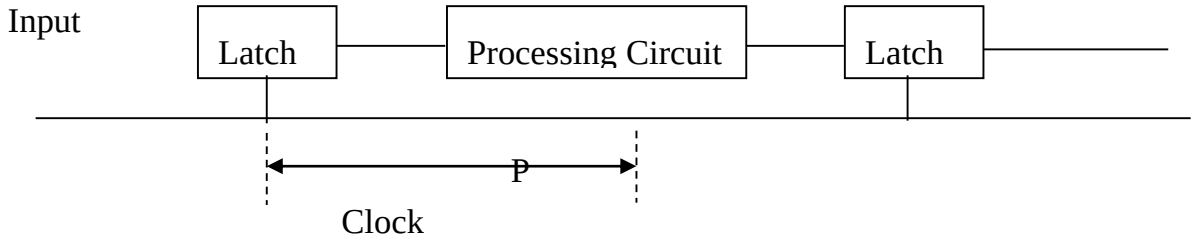
Trong các máy tính tiên tiến sử dụng kỹ thuật đường ống lệnh pipeline, CPU được tổ chức để song song hóa nhiều công đoạn trong một chu kỳ xử lý lệnh. Khối thanh ghi được tổ chức phân cấp có khối lượng lớn (gọi là cache). CPU không chỉ lấy từng lệnh ở bộ nhớ mà lấy cả khối lệnh đặt sẵn trên cache để giảm thiểu thời gian do truy cập bộ nhớ nhiều lần. Khi nhiều lệnh đã được đưa lên bộ nhớ cache thì trong khi đang thực hiện một lệnh, có thể đồng thời đọc dữ liệu cho một lệnh thứ hai và giải mã một lệnh thứ ba theo thứ tự.

Kỹ thuật này một phương pháp cải thiện toàn bộ hiệu suất xử lý của bộ xử lý, gần như cho phép thực hiện nhiều lệnh một cách đồng thời. Pipeline dễ hiểu đối với nhà lập

trình, nó được thực hiện xử lý các lệnh chồng lên nhau. Thiết kế pipeline làm giảm đáng kể thời gian rảnh rỗi của CPU khi thực hiện liên tiếp của các câu lệnh. [1]

3.3.1 Khái niệm Pipeline

Pipeline là một kỹ thuật thực hiện lệnh trong đó các lệnh được thực hiện theo kiểu gối đầu nhau nhằm tận dụng những khoảng thời gian dỗi giữa các công đoạn, qua đó làm tăng tốc độ thực hiện lệnh của vi xử lý.



Hình 3.8 Cấu trúc pipeline

P = tb + tl

Hoạt động: các dữ liệu hoặc các lệnh khi cần xử lý sẽ được đưa vào input. Sau một xung đồng hồ Clock thì các dữ liệu sẽ được chuyển vào bộ xử lý, đến xung clock tiếp theo thì đầu ra của bộ xử lý đó sẽ được chuyển vào bộ xử lý ở công đoạn sau.

Quá trình này cứ được thực hiện liên tiếp sau n xung clock thì ta được kết quả của lệnh mới được đưa vào. Khi lệnh đầu tiên được thực hiện ở công đoạn hai thì lệnh hai sẽ được đưa vào công đoạn thứ nhất theo phương pháp dây chuyền. Như vậy với các lệnh từ thứ hai trở đi thì người ta chỉ cần một xung nhịp clock đã có kết quả của lệnh.

Khoảng thời gian của xung nhịp clock sẽ được tính bằng khoảng thời gian của công đoạn thực hiện lâu nhất + thời gian chuyển dịch từ công đoạn này sang công đoạn khác. Thời gian thực hiện lâu nhất ký hiệu là tb (t Bottle neck). tl là khoảng thời gian cho dữ liệu chảy ra (Tlatch).

Các thông số đo khả năng thực hiện của piepline:

a) Thời gian thực hiện lệnh

Giả sử hệ thống piepline gồm m công đoạn, thực hiện n lệnh.

$T_{pipeline}$ = m*p + (n-1)*p (thời gian thực hiện pipeline)

Trong khi đó nếu thực hiện tuần tự ta có :

$T_{seq} = n\sum_{i=1}^{m} t_i$ t_i: thời gian thực hiện công đoạn thứ i

Giả sử $t_i = t$ với mọi i. ($t_i = t_b = t$)

Ta có T_{seq} = m*n*t

Giả sử $t_l = 0$ => $T_{pipeline} = m*(t_b + t_l) + (n-1)*(t_b + t_l) = m*t + (n-1)*t$

b) Hệ số tăng tốc

Gọi S (speedup) là hệ số tăng tốc. Ta có $s = \frac{T_{seq}}{T_{pipe}} = \frac{m*n*t}{(m+(n-1))*t} = \frac{m*n}{m+n-1}$

Nếu $n \longrightarrow \infty \Rightarrow S_{n \longrightarrow \infty} = m$

Ví dụ: Hệ thống pipeline có 5 công đoạn, thực hiện 5 lệnh

$$s = \frac{5*5}{5+5-1} = \frac{25}{9}$$

	1	2	3	4	5	6	7	8	9
I_1	-	-	-	-	-|				
I_2		-	-	-	-	-|			
I_3			-	-	-	-	-|		
I_4				-	-	-	-	-|	
I_5					-	-	-	-	-|

Hiệu quả (Efficiency) và thông lượng (Through put)

$$E = \frac{S}{m} = \frac{n}{m+n-1}$$

$$H = \frac{n}{T_{pipe}} = \frac{n}{m*p+(n-1))*p}$$

Trong kỹ thuật pipeline thì một lệnh sẽ được chia nhỏ thành nhiều công đoạn để thực hiện, nhìn chung sẽ được chia thành năm công đoạn như sau:

1) IF- instruction fetch: lấy lệnh từ bộ nhớ hoặc cache.
2) ID- instruction decode: giải mã lệnh và đọc các toán hạng.
3) EX- execute: thực hiện lệnh, nếu là lệnh truy cập bộ nhớ thì tính toán địa chỉ ô nhớ.
4) MEM- memory access: đọc hoặc ghi bộ nhớ.
5) WB- write back: ghi/ lưu kết quả vào các thanh ghi.

Chức năng thực hiện tại mỗi giai đoạn:

Instruction Fetch - IF (Đọc lệnh từ bộ nhớ):

- Gọi lệnh từ bộ nhớ cache và nạp chúng vào thanh ghi lệnh (IR).
- Sử dụng địa chỉ lưu trong thanh ghi PC để giải mã ra mã máy của câu lệnh tiếp theo và lưu vào thanh ghi trung gian IF/ID.
- Giá trị PC được cộng thêm và lưu vào thanh ghi trung gian IF/ID.

Intruction Decode - ID (Giải mã lệnh và đọc các thanh ghi):

Sử dụng mã máy của câu lệnh lưu trong thanh ghi IF/ID làm đầu vào cho khối Regfile. Khối Control sử dụng phần opcode của mã máy của câu lệnh để giải mã thành các tín hiệu điều khiển, ngoài tín hiệu SignEx được sử dụng cho khối mở rộng, các tín hiệu khác được lưu vào thanh ghi trung gian ID/EX. Đọc các thanh ghi rs, rt từ bộ thanh ghi và lưu vào

thanh ghi trung gian. ID/EXk - khối mở rộng sử dụng tín hiệu SignEx từ khối Control để mở rộng dấu hay mở rộng zero của 16 bit thấp của mã máy thành 32 bit và lưu vào thanh ghi ID/EX. Địa chỉ các thanh ghi rs, rt, rd được lưu vào thanh ghi ID/EX.

Execution - EX (Tính toán kết quả của câu lệnh hoặc địa chỉ):

Khối ALU sử dụng các đầu vào đã được lưu trong thanh ghi ID/EX để tính toán và lưu kết quả vào thanh ghi trung gian EX/MEM. Một bộ mux được dùng để lựa chọn thanh ghi đích từ hai thanh ghi Rt, Rd và lưu địa chỉ vào thanh ghi EX/MEM. Địa chỉ mới của PC sau câu lệnh cũng được tính toán trong khối này.

Một số bộ mux được dùng để lựa chọn giá trị mới cho PC từ các câu lệnh rẽ nhánh. Các tín hiệu điều khiển MemWrite, MemtoReg và RegWrite được lưu tiếp vào thanh ghi EX/MEM.

Memory access - MEM (Đọc hoặc ghi dữ liệu trên bộ nhớ dữ liệu):

Sử dụng kết quả tính toán từ khối ALU và tín hiệu điều khiển MemWrite từ thanh ghi EX/MEM để thực hiện đọc hoặc ghi vào bộ nhớ dữ liệu. Kết quả đọc ghi vào thanh ghi trung gian MEM/WB.

Các giá trị đầu ra của ALU, địa chỉ thanh ghi đích cùng với hai tín hiệu điều khiển MemtoReg và RegWrite được ghi lại vào thanh ghi MEM/WB.

Write back - WB (Ghi kết quả vào thanh ghi):

Sử dụng tín hiệu MemtoReg từ thanh ghi MEM/WB để lựa chọn dữ liệu cần ghi vào thanh ghi RF (Register File). Chú ý rằng khi lệnh đầu tiên trong đường ống được ghi kết quả trả về đến tập thanh ghi, lệnh thứ tư trong thứ tự năm lệnh trong đường ống được đọc từ thanh ghi.

Sử dụng địa chỉ thanh ghi đích và tín hiệu cho phép ghi RegWrite để thực hiện công việc ghi dữ liệu vào bộ thanh ghi.

Tóm lại, tiến trình thực hiện một lệnh bao gồm một số bước. Trước tiên, bộ điều khiển của bộ vi xử lý lấy ra lệnh từ bộ nhớ cache (hoặc từ bộ nhớ). Sau đó, bộ phận điều khiển giải mã lệnh để xác định loại hoạt động sẽ được thực hiện. Khi thao tác yêu cầu toán hạng, bộ điều khiển cũng xác định địa chỉ của mỗi toán hạng và lấy chúng từ bộ nhớ cache (hoặc bộ nhớ). Tiếp theo, thao tác được thực hiện trên các toán hạng và cuối cùng, kết quả được lưu trữ ở vị trí xác định.

Pipeline lệnh làm tăng hiệu suất của bộ xử lý bằng việc gối chồng quá trình xử lý của một vài lệnh khác nhau. Pipeline lệnh gối chồng tiến trình của các phân đoạn trước lên các lệnh khác để thu được tổng thời gian hoàn tất thấp hơn nhiều đối với một chuỗi các lệnh.

Quy trình thực hiện lệnh trong đường ống lệnh pipeline cũng có thể được phân chia thành sáu giai đoạn như sau:

- Truy xuất lệnh (FI - Fetch instruction): đọc lệnh mong đợi tiếp theo vào bộ đệm.
- Giải mã lệnh (DI - Decode instruction): Xác định opcode và nhận diện toán hạng.

- Tính toán các toán hạng (CO - Calculate operands): Tính toán địa chỉ hiệu dụng của từng toán hạng nguồn tùy thuộc vào các phương pháp định địa chỉ khác nhau.
- Truy xuất toán hạng (FO - Fetch operands): Truy xuất từng toán hạng từ bộ nhớ. Không cần truy xuất toán hạng từ thanh ghi.
- Thực thi lệnh (EI - Execute instruction): Thực hiện hành động được chỉ định và lưu trữ kết quả (nếu có) trong vị trí toán hạng đích đã định.
- Ghi toán hạng (WO - Write operand): Lưu kết quả vào bộ nhớ.

3.3.2 Phân loại Pipeline

Pipeline thường được phân làm hai loại: pipeline lệnh và pipeline số học. Pipeline trong mỗi loại này lại có thể được thiết kế theo hai cách tĩnh hoặc động.

Pipeline tĩnh có thể chỉ thực hiện một thao tác như cộng hay nhân tại một thời điểm. Thao tác của các pipeline tĩnh chỉ có thể được thay đổi sau khi pipeline được giải phóng (pipeline được giải phóng khi dữ liệu đầu vài cuối cùng đi ra khỏi pipeline). Ví dụ xem xét một pipeline tĩnh có khả năng thực hiện cộng và nhân. Tại mỗi thời điểm mà pipeline chuyển từ thao tác nhân sang thao tác cộng, nó phải được giải phóng và thiết lập cho thao tác mới. Hiệu suất của pipeline tĩnh giảm mạnh khi các thao tác thay đổi thường xuyên vì điều này yêu cầu pipeline phải được giải phóng và nạp lại tại mỗi thời điểm.

Pipeline động có thể thực hiện nhiều hơn một thao tác tại một thời điểm. Để thực hiện một thao tác riêng biệt trên một dữ liệu đầu vào, dữ liệu phải đi qua một chuỗi các phân đoạn nào đó. Trong pipeline động cơ chế điều khiển khi nào dữ liệu được nạp vào pipeline phức tạp hơn nhiều trong pipeline tĩnh.

3.3.3 Các hazards trong kỹ thuật Pipeline

3.3.3.1 Vấn đề tăng thông lượng của lệnh.

Ba nguồn gốc của các vấn đề về kiến trúc có thể ảnh hưởng tới pipeline lệnh là vấn đề nạp dữ liệu, vấn đề nghẽn cổ chai, và vấn đề issuing. Một số giải pháp được dưa ra cho từng vấn đề.

Vấn đề nạp dữ liệu:

Nói chug việc cung cấp các lệnh nhanh chóng qua pipeline là đắt giá trong giới hạn diện tích của chíp. Đệm dữ liệu để gửi tới pipeline là một cách đơn giản để tăng toàn bộ khả năng sử dụng của pipeline được định nghĩa như phần tram thời gian mà các phân đoạn của pipeline được sử dụng trên chu kỳ thời gian đủ lớn. Một dòng pipeline được sử dụng 100% khi mọi phân đoạ được sử dụng trong từng chu kỳ đồng hồ.

Đôi khi một pipeline được giải phóng và nạp lại ví dụ mỗi khi xảy ra ngắt hoặc rẽ nhánh. Thời gian để nạp lại pipeline có thể được tối thiểu hoá bằng việc nạp trước các lệnh và dữ liệu vào các bộ đệm như cache trên chip để có thể truyền ngay tức thì vào pipeline. nếu các lệnh và dữ liệu cho một hành động thông thường được nạp trước khi chúng là cần thiết và lưu trong các bộ đệm pipeline sẽ có một nguồn thông tin liên tục để thực hiện công việc. Các thuật toán tiền nạp được sử dụng để đánh giá khả năng chắc chắn của các llệnh cần

thiết là có sẵn trong hầu hết thuời gian. Trễ do xung đột truy nhập bộ nhớ có thể được giảm nếu sử dụng các thuật toán này, do thời gian yêu cầu để truyền dữ liệu từ bộ nhớ chính lớn hơn nhiều thời gian yêu cầu để truyền dữ liệu từ bộ đệm.

Vấn đề nghẽn cổ chai:

Liên quan đến dung lượng nạp ấn định cho một phân đoạn trong pipeline. Nếu quá nhiều công việc được gán cho một phân đoạn thời gian cần để hoàn tất một thao tác tại phân đoạn này có thể trở nên dài không thể chấp nhận được. Khoảng thời gian khá dài này được sử dụng bởi một lệnh trong một phân đoạn chắc chắn sẽ tạo nên nghẽn cổ chai trong hệ thống pipeline. Trong một hệ thống như thế tốt hơn là phải loại bỏ nghẽn cổ chai đó là nguồn gốc của sự tắc nghẽn. Một giải pháp cho vấn đề này là tiếp tục phân nhỏ phân đoạn. Giải pháp khác là xây dựng nhiều bản sao của phân đoạn này trong pipeline.

Vấn đề issuing:

Nếu một lệnh sẵn sàng nhưng không thể thực hiện do một số nguyên nhân thì có một hazard tồn tại với lệnh đó. Các hazard này tạo nên vấn đề issuing. Chúng ngăn cản việc đưa một lệnh ra thực hiện. có ba loại hazard được thảo luận ở đây đó là hazard cấu trúc, dữ liệu và điều khiển.

Hazard cấu trúc nói đến trạng thái trong đó nguồn yêu cầu không sẵn sàng (hoặc đang bận) đối với việc thực hiện lệnh

Hazard dữ liệu nói đến trạng thái trong đó tồn tại sự phụ thuộc dữ liệu (xung đột toán hạng) với lệnh trước đó.

Hazard điều khiển nói đến trạng thái trong đó lệnh rẽ nhánh gây ra sự thay đổi trong luồng chương trình.

3.3.3.2 Các kiểu xung đột trong đường ống lệnh Pipeline.

Xung đột cấu trúc (Structural Hazard).

Một xung đột cấu trúc xảy ra đó là kết quả của việc xung đột tài nguyên giữa các lệnh. Loại xung đột (hazard) này có thể xảy ra do thiết kế các đơn vị thực hiện. Nếu một đơn vị thực hiện yêu cầu nhiều hơn một chu kỳ đồng hồ (chẳng hạn như lệnh nhân) không hoàn toàn pipeline hoặc không được tái nạp, khi đó một chuỗi lệnh mà sử dụng đơn vị đó không thể được đưa ra thực hiện tiếp sau (một lệnh mỗi chu kỳ đồng hồ). Việc tái tạo và hoặc trong pipeline các đơn vị thực hiện làm tăng số lệnh có thể được đưa ra đồng thời.

Một loại xung đột (hazard) cấu trúc khác có thể xảy ra do thiếu tập thanh ghi. Nếu một tập thanh ghi không có nhiều cổng ghi/đọc, việc ghi/đọc đồng thời tới thanh ghi là không thể thực hiện được. Ví dụ như hình bên dưới trong tình trạng nào đó pipeline lệnh muốn thực hiện hai lệnh ghi lên thanh ghi trong một chu kỳ đồng hồ. Điều này là không thể khi tập thanh ghi chỉ có một cổng ghi.

CLOCK							
	1	2	3	4	5	6	7
Load	IF	ID	EX	*MEM*	WB		
Instr1		IF	ID	EX	MEM	WB	
Instr2			IF	ID	EX	MEM	WB
Instr3				*IF*	ID	EX	MEM WB

Bảng 3.1 Xung đột cấu trúc

Cách khắc phục cho xung đột này:

Tác động của xung đột này có thể được giảm đi một cách đơn giản bằng cách bổ sung thực hiện nhiều đơn vị thực hiện như hình sau, hoặc bổ sung thêm phần cứng sử dụng các tập thanh ghi đa cổng ghi/đọc.

CLOCK										
	1	2	3	4	5	6	7			
Load	IF	ID	EX	MEM	WB					
Instr1		IF	ID	EX	MEM	WB				
Instr2			IF	ID	EX	MEM	WB			
Stall				Buble	Buble	Buble	Buble	Buble		
Instr3					IF	ID	EX	MEM	WB	

Bảng 3.2 Thêm "Bubble" xử lý xung đột cấu trúc

Xung đột dữ liệu (Data Hazard)

Một xung đột được tạo ra bất cứ khi nào có một sự phụ thuộc giữa các lệnh và chúng thì gần như đủ để sự chồng chéo trong quá trình thực hiện sẽ thay đổi thứ tự tiếp cận với các toán hạng tham gia vào sự phụ thuộc. Do sự phụ thuộc, chúng ta phải đảm bảo được điều gọi là trật tự chương trình, có nghĩa là, những chỉ lệnh sẽ thực hiện theo thứ tự nếu chúng đã thực hiện một cách liên tục một lần tại một thời điểm xác định bởi chương trình nguồn ban đầu. Mục tiêu của những kỹ thuật phần mềm và phần cứng là để khai thác song song bằng cách bảo đảm trật tự chương trình nơi mà nó ảnh hưởng đến kết quả của chương

trình. Phát hiện và tránh những rủi ro đảm bảo rằng trật tự cần thiết của chương trình được bảo toàn.

Những rủi ro dữ liệu, có thể được phân thành một trong ba loại, tùy theo thứ tự của những truy xuất đọc và ghi trong những lệnh. Theo quy ước, các rủi ro được đặt tên theo thứ tự trong chương trình mà phải được bảo toàn bằng các đường ống. Hãy xem xét hai chỉ lệnh i_1 và i_2, với i_1 trước i_2 theo thứ tự chương trình. Các rủi ro dữ liệu có thể là:

RAW: Loại dữ liệu nguy hiểm này đã được thảo luận trước đây; Nó đề cập đến tình huống trong đó i_2 đọc một nguồn dữ liệu trước khi i_1 viết cho nó. Điều này có thể tạo ra một kết quả không hợp lệ vì đọc phải được thực hiệnsau khi ghi để có được một kết quả hợp lệ.

Ví dụ 1: kết quả sai có thể được đưa ra nếu i_2 đọc R_2 trước khi i_1 ghi nó.

i_1: Add R_2, R_3, R_4 --$R_2=R_3+R_4$

i_2: Add R_5, R_2, R_1 --$R_5=R_2+R_1$

WAR: nó đề cập đến tình huống trong đó i_2 ghi vào một vị trí trước khi i_1 đọc nó. Ví dụ một kết quả sai có thể được đưa ra nếu i_2 ghi vào R_4 trước khi i_1 đọc nó tức là lệnh i_1 sử dụng giá trị sai của R_4.

i_1: Add R_2, R_3, R_4 -- $R_2=R_3+R_4$

i_2: Add R_4, R_5, R_6 -- $R_4=R_5+R_6$

WAW: nó đề cập đến tình trạng trong đó i2 ghi vào một vị trí trước khi i1 ghi vào đó. Ví dụ giá trị của r2 được tính lại bởi i1. nếu thứ tự thực hiện được đảo ngược tức là i2 ghi vào r2 trước khi i1 ghi vào đó một giá trị sai cho r2 có thể được đưa ra.

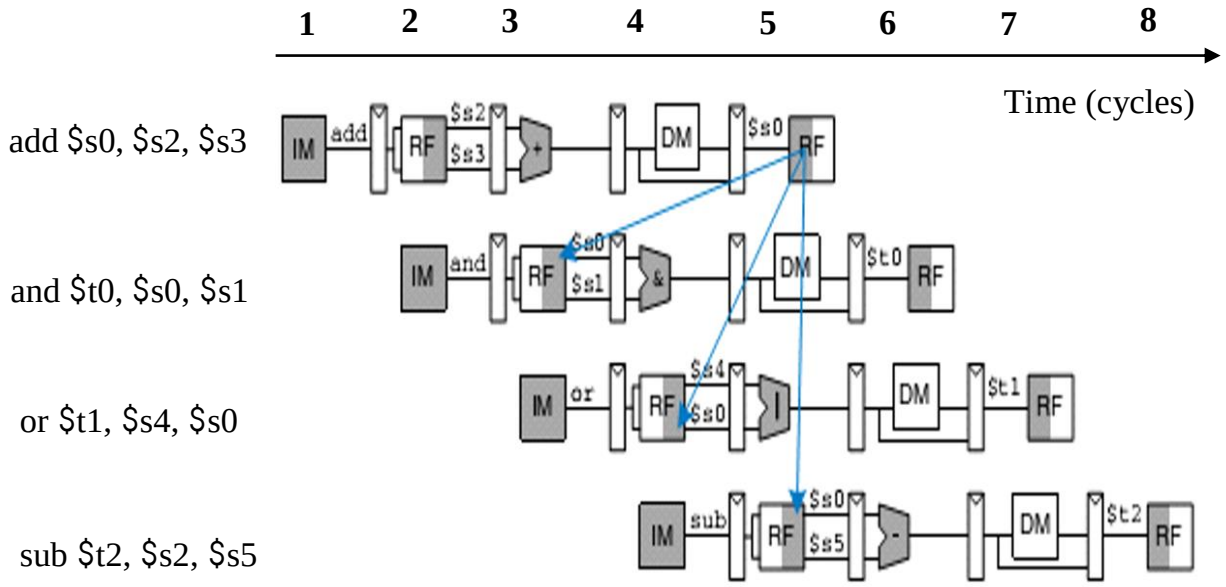
i_1: Add R_2, R_3, R_4 -- $R_2=R_3+R_4$

i_2: Add R_2, R_5, R_6 -- $R_2=R_5+R_6$

Chú ý: với hai kiểu WAR và WAW, mặc dầu các lệnh chạy đồng thời, nhưng nếu chúng kết thúc vẫn đúng thứ tự thì hoàn toàn không xảy ra hazard. Nhưng mà, chính kiến trúc pipeline lại nhằm mục đích xử lý song song nhiều lệnh, bằng cách chia các câu lệnh thành những đơn vị chức năng khác nhau, không bị phụ thuộc lẫn nhau. Để chúng có thể được thực hiện và kết thúc không cần tuân theo thứ tự. Trong kiến trúc như vậy đã làm cho hazard có thể xảy ra.

Cách khắc phục xung đột:

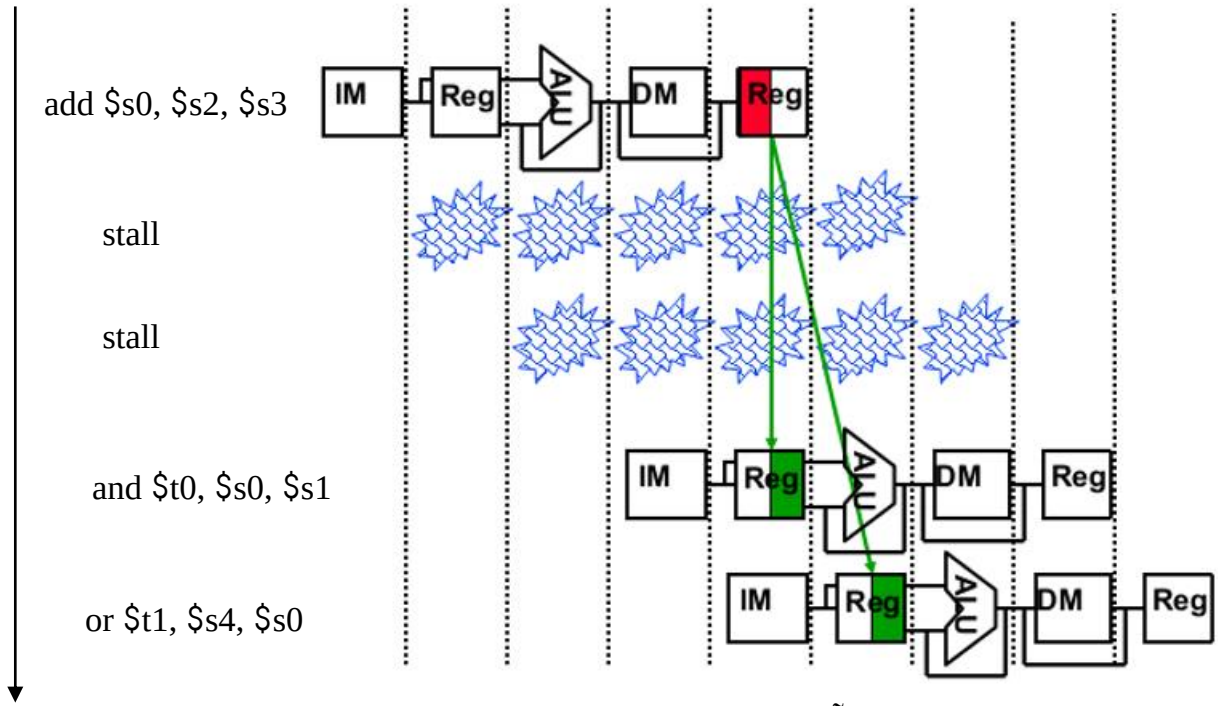
Để giải quyết vấn đề xung đột dạng này ta xét đến một ví dụ như hình sau:



Hình 3.9 Mô hình xung đột dữ liệu

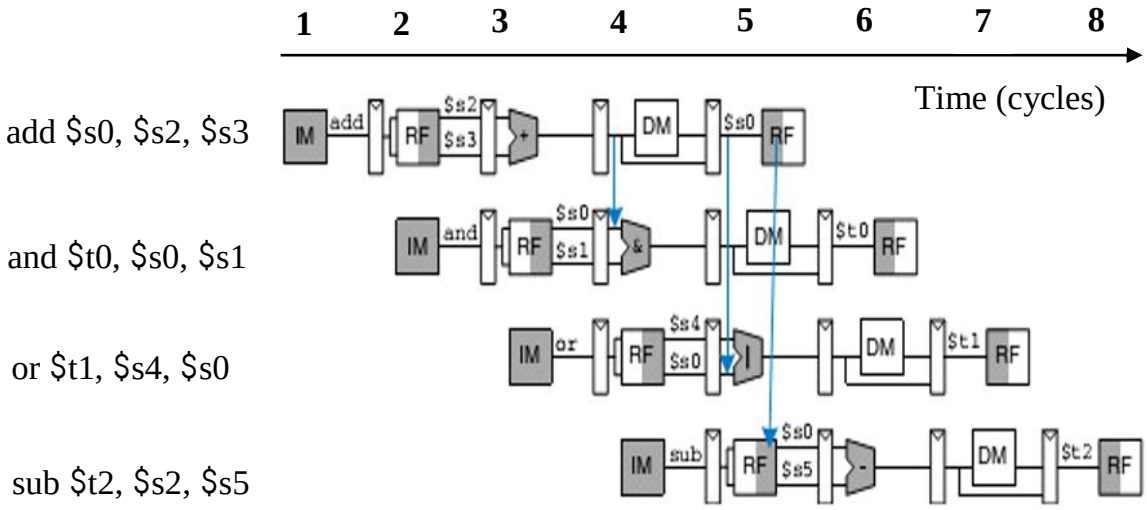
Có hai cách để giải quyết loại xung đột này là chờ dữ liệu tính xong rồi thực hiện lệnh kế tiếp, hoặc chuyển dữ liệu sau khi được tính toán ở giai đoạn MEM hoặc WB về giai đoạn EX.

- Phương pháp chờ: phần cứng sẽ phát hiện sự phụ thuộc dữ liệu và dừng những lệnh nào có dữ liệu phụ thuộc vào lệnh trước đó cho tới khi dữ liệu được sẵn sàng.



Hình 3.10 Hình ảnh chèn trễ

- Phương pháp chuyển tiếp dữ liệu: các lệnh thường được tính toán ở giai đoạn EX rồi chuyển đến các giai đoạn MEM và WB, do vậy ta có thể chuyển dữ liệu trờ về giai đoạn EX cho các lệnh phụ thuộc dữ liệu phía sau.



Hình 3.11 Chuyển tiếp dữ liệu

Ngoài ra, để giải quyết xung đột này, trong những kiến trúc như ngày nay, các lệnh phụ thuộc được kiểm tra bởi phần mềm lúc biên dịch (kiểm tra tĩnh - static), và cũng có thể được kiểm tra bởi phần cứng khi chạy (kiểm tradộng - dynamic). Làm cho thứ tự thực hiện của các câu lệnh phụ thuộc được giữ nguyên, để đảm bảo trả về kết quả đúng.

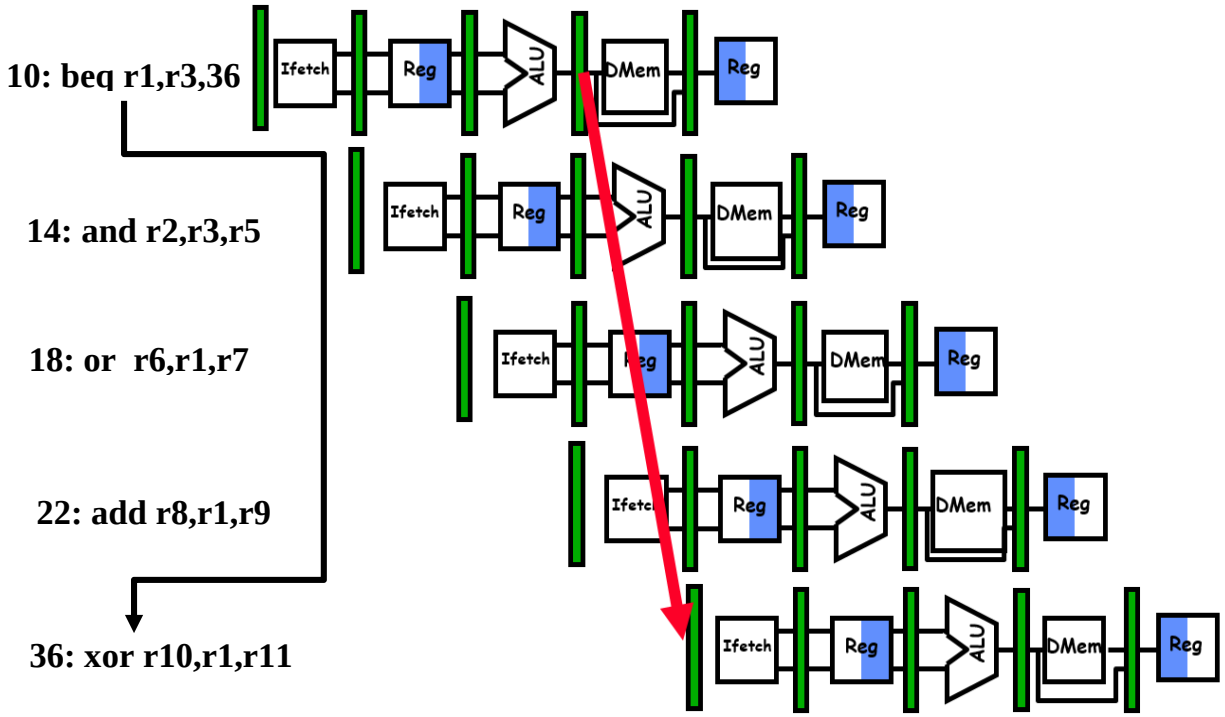
Đã có nhiều kỹ thuật kiểm tra tĩnh khác nhau, chúng đều có những tính năng tiên tiến để có thể tìm ra hầu hết những phụ thuộc trong chương trình. Tuy nhiên có những phụ thuộc không thể xác định được trong lúc biên dịch, chỉ đến khi chạy chương trình thì mới biết được. Ví dụ, khi thực hiện chương trình thì các câu lệnh mới được nạp vào bộ nhớ, khi đó mới biết được địa chỉ của chúng trong bộ nhớ, và khi đó mới có thể giải quyết được vấn đề bằng kỹ thuật kiểm tra động.

Nói chung, kỹ thuật kiểm tra động sẽ bổ sung cho những trường hợp mà kỹ thuật kiểm tra tĩnh không giải quyết được. Ngược lại, kỹ thuật kiểm tra động lại bị hạn chế bởi khả năng hỗ trợ của phần cứng. Trong thực tế, cả hai kỹ thuật này được kết hợp với nhau để giải quyết vấn đề.

Xung đột điều khiển (Control Hazard).

Như chúng ta đã biết bất kỳ một tập lệnh nào của máy tính cũng có những dạng lệnh dùng để điều khiển dòng chương trình theo hướng khác chứ không phải chỉ là tuần tự. Trong ngôn ngữ lập trình người ta gọi đó là lệnh rẽ nhánh. Thông thường trong một chương trình có khoảng 30% lệnh rẽ nhánh. Các lệnh rẽ nhánh sẽ làm giảm thông lượng của Pipeline rất khủng khiếp nếu chúng ta phối hợp chúng không thoả đáng.

Mỗi một lệnh rẽ nhánh thường yêu cầu tạo ra một địa chỉ mới trong bộ đếm chương trình do vậy rất có thể nó sẽ làm hỏng các lệnh đang sẵn sàng trong Pipeline hoặc là phải lấy lại chúng từ buffer. Điều đó sẽ làm thông lượng trong Pipeline giảm xuống như là xử lý tuần tự.



Hình 3.12 Xung đột điều khiển

Các lệnh rẽ nhánh có thể chia ba loại: Rẽ nhánh không điều kiện; rẽ nhánh có điều kiện và rẽ nhánh lặp.

Lệnh rẽ nhánh không điều kiện thường được thực hiện nếu trong chương trình có nó, khi đó nó thiết lập một địa chỉ đích mới trong bộ đếm chương trình chứ ít khi nó tăng lên một để trỏ vào lệnh tiếp theo.

Lệnh rẽ nhánh có điều kiện thì lại khác, nghĩa là khi điều kiện đúng thì nó thiết lập một địa chỉ đích mới trong bộ đếm chương trình, ngược lại thì nó tăng một để trỏ vào lệnh tiếp theo. Hay nói cách khác, một đường dẫn sẽ được chọn và ta gọi đó là đường dẫn đích nếu điều kiện đúng, ngược lại thì đường dẫn lúc này là đường dẫn tuần tự hay chính là lệnh tiếp theo.

Lệnh rẽ nhánh lặp, lệnh này thường lặp lại việc nhảy trở về điểm vào ban đầu của chính nó, tuy nhiên có một cơ chế để đếm số lần hay là bằng cách đánh dấu nào đó để thoát khỏi vòng lặp.

Trong các lệnh rẽ nhánh thì lệnh rẽ nhánh có điều kiện là khó kết hợp hơn cả. Ví dụ sau đây đưa ra lệnh lặp có điều kiện trong dãy các lệnh:

i1

i2 (rẽ nhánh có đk tới ik)

i3

…

ik

ik+1

Để nhìn thấy sự ảnh hưởng của việc bất lợi này trong việc tính toán của Pipeline ta cần xác định trung bình số chu kỳ trên một lệnh. Gọi t_{ave} là trung bình số chu kỳ cần thiết để thực hiện một lệnh.

T_{ave} = P_b *(trung bình số chu kỳ trên một lệnh rẽ nhánh) + (1-P_b)*(trung bình số chu kỳ trên một lệnh không rẽ nhánh). P_b là xác suất xuất hiện lệnh rẽ nhánh.

Trung bình số chu kỳ trên một lệnh rẽ nhánh có thể được xem xét theo hai khả năng:

Nếu đường dẫn đích được chọn thì trung bình số chu kỳ là: 1+ c cycles (c = branch penalty). Ngược lại thì trung bình số chu kỳ là 1.

Như vậy trung bình số chu kỳ trên một lệnh rẽ nhánh là: P_t*(1+c)+(1-P_t)*1. Ở đây P_t là xác suất để đường dẫn đích t được chọn.

Còn trung bình số chu kỳ trên một lệnh không rẽ nhánh là: l

Thay vào công thức trên ta có:

$t_{ave} = P_b [P_t (1+c) + (1-P_t)(1)] + (1- P_b)(1) = 1 + cP_b P_t$.

Theo kết quả của Lee và Smith giả sử ta lấy: P_b=0.2, P_t = 0.65 và c=3.

Thay vào công thức trên ta có:

$$t_{ave} = 1 + 3 (0.2)(0.65) = 1.39.$$

Hay nói cách khác khi có lệnh rẽ nhánh trong Pipeline thì nó chỉ hoạt động được khoảng 72% công suất tối đa. Đôi khi người ta còn quan tâm đến thông lượng của Pipeline khi có lệnh rẽ nhánh, lúc đó nó được xác định như sau:

$$H = 1/t_{ave} = 1/(1+cP_bP_t).$$

Cách khắc phục xung đột:

Để làm giảm ảnh hưởng của việc rẽ nhánh lên hiệu năng bộ vi xử lý, nhiều kỹ thuật đã được đưa ra. Một số ký thuật được biết đến nhiều hơn đó là: dự đoán rẽ nhánh, làm trễ việc rẽ nhánh và nhận trước nhiều nhánh. Các kỹ thuật này sẽ được trình bày dưới đây.

Branch Prediction (dự đoán rẽ nhánh)

Với thiết kế loại này, việc quyết định kết quả của một nhánh được dự đoán trước khi nhánh thực sự được thi hành. Bởi vậy, trên cơ sở của việc dự đoán đó, chuỗi đường dẫn hoặc đường dẫn đích sẽ được chọn để thực thi. Mặc dầu, việc chọn đường dẫn thường

làm giảm BranchPenalty, nhưng nó có thể tăng penalty trong trường hợp dự đoán không chính xác.

Có hai loại dự đoán trước: tĩnh và động.

Trong dự đoán tĩnh, một quyết định cố định về việc nhận trước một trong hai đường dẫn được hoàn thành trước khi chạy chương trình. Ví dụ, một kỹ thuật đơn giản sẽ luôn được giả sử rằng nhánh đã được lấy. Kỹ thuật này nạp vào bộ đếm chương trình địa chỉ đích khi một nhánh được bắt gặp. Một kỹ thuật khác tương tự đó là tự động chọn một đường dẫn (chuỗi đường dẫn hoặc đường dẫn đích) cho một số loại nhánh và đường dẫn khác cho những loại nhánh còn lại. Nếu việc chọn đường dẫn là sai, ống dẫn sẽ được drain và lệnh phù hợp với đường dẫn chính xác sẽ được tìm và nạp. Penalty sẽ được trả lại.

Đối với dự đoán động, trong suốt thời gian thực thi chương trình bộ vi xử lý sắp đặt một quyết định dựa vào các thông tin của việc thực thi các nhánh trước đó. Ví dụ, một kỹ thuật đơn giản sẽ ghi history của hai đường dẫn đã lấy bởi mỗi lệnh rẽ nhánh. Nếu hai lần thực thi trước đó của một lệnh nhánh đều được chọn bởi cùng một đường dẫn thì đường dẫn đó sẽ được chọn cho việc thực thi của lệnh nhánh hiện thời. Nếu cả hai đường dẫn trước đó không phù hợp với nhau (không giống nhau), một trong các đường dẫn sẽ được chọn ngẫu nhiên.

Đối với dự đoán tĩnh, thường đòi hỏi ít hơn về phần cứng, nhưng chúng có thể làm tăng sự phức tạp của trình biên dịch. Trái lại, phương pháp dự đoán động lại làm tăng sự phức tạp phần cứng, giảm sự làm việc của trình biên dịch. Nhìn chung, phương pháp dự đoán động thu được kết quả tốt hơn phương pháp dự đoán tĩnh và cũng cung cấp độ tương thích của đoạn mã đối cao hơn, khi mà quyết định được hoàn thành sau thời gian biên dịch.

Để nhận thấy được tác động về hiệu năng của phương pháp dự đoán rẽ nhánh, chúng ta cần phải định lượng lại số chu kỳ trung bình trên một lệnh nhánh bằng công thức toán học. Có hai trường hợp có thể xảy ra: đường dẫn được dự đoán hoặc chính xác hoặc không chính xác.

Trong trường hợp một đường dẫn dự đoán là chính xác thì penalty = d nếu đường dẫn là đường dẫn đích, penalty = 0 nếu đường dẫn là chuỗi đường dẫn. (Chú ý rằng, địa chỉ của đường dẫn đích được chứa sau giai đoạn giải mã. Tuy nhiên, khi một bộ đệm của nhánh đích được dùng trong kế hoạch này, địa chỉ đích có thể được chứa trong suốt thời gian hoặc sau giai đoạn tìm và nạp lệnh).

Trong trường hợp một dự đoán đường dẫn không chính xác cho cả hai dự đoán đường dẫn đích và dự đoán chuỗi đường dẫn, penalty là c. Chúng ta có: số chu kỳ trung bình/lệnh nhánh = P_r [P_t(1+d) + (1- P_t)(1)] + (1- P_r)[P_t (1+c) + (1- P_t)(1+c)]

Trong đó P_r là xác suất của một dự đoán đúng.

$t_{ave} = P_b[P_r(P_td + 1) + (1- P_r)(1+c)] + (1- P_b)(1)$

$= 1 + P_bc - P_b P_rc + P_bP_rP_td.$

Giả sử rằng pb = 0.2, pt = 0.65, c = 3 và d = 1 và giả sử rằng dự đoán đường dẫn chính xác 70% về mặt thời gian (ví dụ: pr = 0.70), thì: t_{ave} = 1.27. Như vậy, đối với dự đoán rẽ nhánh thì pipeline hoạt động đạt tới 78% công suất tối đa của nó.

Delayed Branching (làm trễ việc rẽ nhánh)

Delayed braching vạch ra kế hoạch loại bỏ hoặc làm giảm đáng kể ảnh hưởng của Branch penalty. Với thiết kế loại này, một số chắc chắn các câu lệnh sau lệnh rẽ nhánh sẽ được tìm nạp và thực thi bất chấp đường dẫn nào sẽ được chọn cho lệnh rẽ nhánh. Ví dụ, một bộ xử lý với một nhánh làm trễ k lần sẽ thực thi một đường dẫn chứa k lệnh liên tiếp theo sau và sau đó hoặc tiếp tục trên cùng đường dẫn, hoặc bắt đầu một đường dẫn mới từ một địa chỉ đích mới. Thông thường có thể xảy ra, trình biên dịch cố gắng điền tiếp đó k "khe chứa lệnh" sau nhánh với các lệnh là độc lập với lệnh rẽ nhánh. Lệnh NOP (không làm gì cả) được đặt vào bất kỳ khe trống còn lại nào. Ví dụ, giả sử có đoạn mã lệnh như sau:

```
i1: Load R1, A
i2: Load R2, B
i3: BrZr R2, i7                    - - rẽ nhánh tới lệnh i7 nếu R2=0
i4: Load R3, C
i5: Add R4, R2, R3          - - R4 = R2+R3
i6: Mul R5, R1, R2          - - R5 = R1*R2
i7: Add R4, R1, R2          - - R4 = R1+R2
```

Giả sử k = 2, trình biên dịch thay đổi đoạn mã này bằng việc di chuyển lệnh i1 và chèn một lệnh NOP sau lệnh nhánh i3. Đoạn mã được sửa là:

```
i2: Load R2, B
i3: BrZr R2, i7
i1: Load R1, A
NOP
i4: Load R3, C
i5: Add R4, R2, R3
i6: Mul R5, R1, R2
i7: Add R4, R1, R2
```

Có thể thấy được trong đoạn mã thay đổi, lệnh i1 sẽ được thực thi bất chấp kết quả của nhánh.

Multiple prefetching (nhận (tìm và nạp) trước nhiều nhánh)

Với thiết kế này, bộ xử lý sẽ tìm và nạp cả hai đường dẫn có thể xảy ra. Khi mà việc quyết định nhánh được hoàn thành, đường dẫn thừa sẽ được loại bỏ đi. Bằng việc nhận trước cả hai đường dẫn có thể, việc nhận penalty sẽ được tránh trong trường hợp một dự đoán không chính xác.

Để nhận cả hai đường dẫn, hai bộ đệm được sử dụng để phục vụ cho pipeline. Việc thực thi thông thường là bộ đệm thứ nhất được nạp với những lệnh từ chuỗi địa chỉ tiếp sau của lệnh nhánh. Nếu một nhánh xảy ra, nội dung của bộ đệm thứ nhất được vô hiệu hoá, và bộ đệm thứ hai, cái mà đã được nạp với những lệnh từ địa chỉ đích của lệnh nhánh, sẽ được sử dụng như bộ đệm chính.

Bộ đệm đôi này lập kế hoạch đảm bảo một lưu lượng không đổi các câu lệnh và dữ liệu đi tới pipeline và làm giảm thời gian trễ được sinh ra do drain và refilling đường ống. Thực chất của việc làm giảm hiệu năng là không thể tránh được bất kỳ thời gian nào pipeline bị drain.

Tóm lại, mỗi kỹ thuật đã được xem xét ở trên đều làm giảm sự suy biến công suất pipeline. Tuy nhiên, việc lựa chọn bất kỳ phương pháp nào cho việc thiết kế cụ thể phụ thuộc vào các nhân tố như công suất đòi hỏi và chi phí phải trả. Trong thực tế, vì hai nhân tố này, sẽ hiếm khi xem xét pha trộn các phương pháp này để thi hành trên một bộ xử lý đơn.

3.4 Kỹ thuật song song mức lệnh

3.4.1 Khái niệm và vai trò của kỹ thuật song song mức lệnh

Trong trường hợp các lệnh là độc lập với nhau, pipelining có thể thực hiện các lệnh gối lên nhau và việc gối lên nhau giữa các lệnh được gọi là Instruction level parallelism (ILP). ILP cho phép tăng khả năng song song hóa thực hiện các lệnh, đồng thời giảm ảnh hưởng của các data hazard và control hazard, giúp tăng khả năng xử lý để khai thác hiệu quả cơ chế song song hóa. Việc song song hóa có thể được thực hiện bằng phần cứng và phần mềm, nếu được thực hiện bằng phần cứng thì được gọi là dynamic techniques, được thực hiện bằng phần mềm gọi là static techniques.

Mục đích của bộ biên dịch và bộ xử lý là phải nhận biết và thực hiện song song hóa ở mức tối đa có thể. Các chương trình thông thường đều được viết theo mô hình thực hiện tuần tự, trong đó, các lệnh được thực hiện lần lượt, lệnh này sau lệnh khác với thứ tự được chỉ ra trong chương trình. ILP cho phép bộ biên dịch và bộ xử lý gối lên nhau khi thực hiện các lệnh, thậm chí là thay đổi thứ tự các lệnh được thực hiện. Ví dụ với chương trình sau:

1. e = a + b
2. f = c + d
3. g = e * f

Thao tác thứ ba được thực hiện dựa trên kết quả của thao tác thứ nhất và thứ hai, do đó nó không thể được thực hiện cho đến khi một, hai hoàn thành. Tuy nhiên, thao tác một

và hai không phụ thuộc lẫn nhau, nên có thể được thực hiện đồng thời. Giả sử mỗi thao tác được hoàn thành trong một đơn vị thời gian, thì cả chương trình có thể được thực hiện chỉ trong hai đơn vị thời gian, do đó ILP được tính là 3/2.

Dạng đơn giản nhất và phổ biến nhất của song song mức lệnh là song song mức lệnh trong vòng lặp (loop-level parallelism).

Ví dụ 1:

```
for (i=1; i<=1000; i= i+1)
        x[i] = x[i] + y[i];
```

Trong vòng lặp này, mỗi bước lặp là không có liên quan đến nhau, và các bước lặp có thể được thực hiện song song với nhau.

Ví dụ 2:

```
for (i=1; i<=100; i= i+1)
{
        a[i] = a[i] + b[i];          //s1
        b[i+1] = c[i] + d[i];      //s2
}
```

Trong mỗi bước lặp, s1 phụ thuộc vào s2, nhưng s2 không phụ thuộc vào s1. Để xóa đi sự phụ thuộc này, nhằm thực hiện song song trong mỗi vòng lặp, ta biến đổi như sau:

```
a[1] = a[1] + b[1];
for (i=1; i<=99; i= i+1)
{
        b[i+1] = c[i] + d[i];
        a[i+1] = a[i+1] + b[i+1];
}
b[101] = c[100] + d[100];
```

Ví dụ 3:

```
for (i=1; i<=100; i= i+1)
{
        a[i+1] = a[i] + c[i];          //s1
        b[i+1] = b[i] + a[i+1];     //s2
}
```

Trong ví dụ này, s1 phụ thuộc vào s2, đồng thời s2 cũng phụ thuộc vào s1. Ở đây không có cách biến đổi nào để mỗi vòng lặp có thể thực hiện song song với nhau được.

3.4.2 Phụ thuộc trong song song mức lệnh

Dưới đây chúng ta đi tìm hiểu kỹ hơn về việc phụ thuộc trong song song mức lệnh.

Data dependencies

Lệnh J được gọi là có phụ thuộc dữ liệu với lệnh I nếu một trong hai điều kiện sau được thỏa mãn: (1) Instruction I sinh ra một kết quả được sử dụng trong instruction J, hoặc (2) Instruction J là độc lập dữ liệu với instruction K, và instruction K là phụ thuộc dữ liệu với instruction I. Nếu hai lệnh là phục thuộc dữ liệu, chúng không thể được thực hiện đồng thời hoặc không thể thực hiện gối lên nhau hoàn toàn. Sự phụ thuộc chỉ ra rằng sẽ có một chuỗi một hoặc nhiều RAQ hazard giữ hai instruction. Vì phụ thuộc dữ liệu làm hạn chế số lượng các ILP chúng ta có thể thực hiện, vì vậy cần phải nghiên cứu cách thức khắc phục nhược điểm này. Điều này có thể được thực hiện theo hai cách khác nhau: duy trì sự phụ thuộc và khắc phục hazard, hoặc giảm sự phụ thuộc bằng cách điều chỉnh, chuyển hóa code ban đầu để có được đoạn code tối ưu. Lập lịch thực hiện code (Shedungling the code) là phương pháp cơ bản nhất được sử dụng để tránh hazard mà không cần phải thay đổi sự phụ thuộc. Xóa bỏ sự phụ thuộc dữ liệu đòi hỏi có những thông tin về cấu trúc tổng thể của một chương trình và thường cần đến rất nhiều các xử lý phức tạp. Tuy nhiên, các kỹ thuật để thực hiện việc tối ưu hóa như thế thường được đảm nhận bởi các bộ compiler. Điều này ngược với việc tránh các hazard bằng scheduling, vì theo cách này, cả phần cứng và phần mềm đều có thể đảm nhận được.

Name dependencies

Loại phụ thuộc thứ hai là name dependence. Một name dependence xuất hiện khi hai instruction sử dụng cùng một thanh ghi hay một vùng bộ nhớ mà không có dòng dữ liệu giữa những instruction kết hợp với tên đó. Có hai loại dependences giữa một lệnh I xuất hiện trước lệnh j trong thứ tự chương trình: (1) Antidependence giữa lệnh j và lệnh j xuất hiện khi lệnh j thực hiện write vào một thanh ghi hay một vùng bộ nhớ mà lệnh i đọc và lệnh i được thực hiện trước. Một Antidependence tương ứng với một WAR hazard. (2) Output dependence xuất hiện khi lệnh i và lệnh j viết vào cùng một thanh ghi hay một vùng bộ nhớ. Thứ tự giữa các lệnh phải được giữ nguyên

Cả Antidependence và Output dependence đều là name dependences, ngược với data dependencies, khi mà không có giá trị nào được truyền giữa các instructions. Điều này có nghĩa là các instruction liên quan tới một phụ thuộc tên có thể thực hiện đồng thời hoặc được thực hiện theo một thứ tự khác, nếu name (thanh ghi hoặc vùng bộ nhớ) được sử dụng trong các lệnh này thay đổi, do đó không có xung đột giữa các instruction. Renaming có thể được thực hiện dễ dàng nhờ các thanh ghi toán hạng và được gọi là các register renaming. Register renaming có thể được thực hiện tĩnh bởi bộ biên dịch hoặc được thực hiện động bởi phần cứng.

Name dependences ép buộc các lệnh trong vòng lặp phải được thực hiện theo đúng thứ tự như thế. Thanh ghi được sử dụng cho mỗi lệnh copy của thân vòng lặp và chỉ được đổi tên khi các phụ thuộc được giữ lại

Với renaming, việc copy trong mỗi thân vòng lặp trở nên độc lập với nhau và có thể được thực hiện gối lên nhau, hoặc có thể được thực hiện song song hóa. Quá trình thực

hiện renaming được thực hiện bởi hoặc bộ biên dịch, hoặc phần cứng. Trong đó, thông thường chúng ta thấy renaming trong phần cứng là chủ yếu.

Control Dependences

Loại phụ thuộc cuối cùng là control dependence. Một control dependence quyết định thứ tự thực hiện của một lệnh, tuân theo lệnh rẽ nhánh, do đó lệnh không rẽ nhãnh chỉ được thực hiện khi cần. Tất cả các lệnh, trừ những lệnh trong khối đầu tiên của chương trình là độc lập, là phụ thuộc điều khiển trên một số nhánh, và nhìn chung, những control dependences này phải được giữ nguyên. Một trong những ví dụ đơn giản nhất của control dependence là sự phụ thuộc của các statements trong phần "then" của mỗi lệnh if statement. Có hai loại ràng buộc cho các control dependences: (1) Một lệnh là phụ thuộc điều khiển vào một branch không thể được chuyển đến trước branch, do việc thực hiện của nó không còn được điều khiển bởi branch. Ví dụ, chúng ta không thể thực hiện một lệnh từ phần then của một lệnh if và chuyển nó trước lệnh if. (2) Một lệnh không là phụ thuộc điều khiển vào một branch không thể chuyển đến sau một branch vì nếu thế việc thực hiện nó sẽ được điều khiển bởi một branch. Ví dụ, chúng ta không thể thực hiện một lệnh trước lệnh if và chuyển nó vào phần then.

3.5 Bộ xử lý đa luồng và đa lõi

CPU đa nhân, **CPU đa lõi** (multi-core) là một có nhiều đơn vị vi xử lý (thường được gọi là "core") được tích hợp và đóng gói trên cùng một nền mạch tích hợp (chip) vật lý duy nhất. Mỗi core đều có thể thực hiện việc xử lý tuần tự từng gói dữ liệu và sự kết hợp nhiều core trên một hệ CPU giúp làm tăng tốc độ xử lý chung của hệ thống khi mà dữ liệu được phân thành nhiều gói nhỏ và phân cho các core xử lý song song cùng một lúc.

Vậy tại sao các kiến trúc sư phần cứng lại muốn đặt nhiều CPU vào cùng một chip? Lý do lớn nhất là việc đặt nhiều lõi lên cùng một vi mạch sẽ giúp giảm không gian trên bản mạch chính khi có nhu cầu muốn sử dụng với số lượng CPU lõi đơn tương đương. Thêm nữa, lợi thế của việc sử dụng đa lõi trên cùng một vi mạch đương nhiên sẽ làm việc kết hợp cùng nhau chặt chẽ và nâng cao được hiệu quả hơn.

Khả năng tiết kiệm năng lượng cũng được phát huy thấy rõ đối với thiết kế này. Khi nhiều lõi cùng nằm trên một chip, xung tín hiệu truyền giữa các lõi sẽ ngắn hơn. Ngoài ra, đặc trưng của CPU đa lõi là chạy với điện năng thấp hơn vì công suất tiêu tốn để tín hiệu truyền trên dây bằng với bình phương điện áp chia cho điện trở trong dây, do đó điện năng thấp hơn sẽ dẫn đến kết quả là nguồn điện sử dụng đi. [4]

Một lý do khác đối với việc tiết kiệm nguồn điện là tốc độ đồng hồ. Như bạn thấy, CPU đa lõi có thể thực thi các hoạt động nhiều lần hơn trong một giây trong khi tần số thấp hơn. Ví dụ bộ xử lý MIT RAW 16 lõi hoạt động ở tần số 425MHz có thể thực thi gấp 100 lần các hoạt động trong một giây đối với Intel Pentium 3 đang chạy ở tần số 600 MHz. Vậy tần số như vậy ảnh hưởng như thế nào với sự tiêu thụ điện năng của CPU? Đây quả là vấn đề khá phức tạp, nhưng bạn phải hiểu một quy tắc đơn giản là mỗi một phần trăm tăng

thêm tốc độ đồng hồ sẽ tăng 3% điện năng tiêu thụ. Và tất nhiên là điều đó còn chưa tính tới tác động của các nhân tố khác có ảnh hưởng tới sự tiêu thụ điện năng. CPU đa lõi còn có thể chia sẻ một mạch ghép nối bus tốt như mạch lưu trữ. Hình 1 là lược đồ của chip Core 2 dual của Intel - có tính năng là một L2 cache được chia sẻ. Kết quả là tiết kiệm được lượng không gian đáng kể. Theo Intel, CPU Core 2 dual có thể lên tới 4MB được chia sẻ L2 Cache.

Đa luồng

Kiến trúc này có tác dụng gì với các nhà phát triển phần mềm? Một nhân tố khác giới hạn lợi ích thực thi của CPU đa lõi là phần mềm chạy trên nó. Đối với người dùng bình thường, hiệu suất lớn nhất mà họ đạt được khi lựa chọn một CPU đa lõi là tính đa nhiệm được cải thiện. Ví dụ, với một CPU đa lõi bạn sẽ thấy sự cải thiện lớn khi xem DVD trong lúc máy vẫn đang được quét virus mà tốc độ không bị ảnh hưởng, bởi vì từng ứng dụng sẽ được gán trên các lõi khác nhau.

Nếu người dùng đang chạy một ứng dụng đơn trên máy tính đa lõi thì sẽ không thấy rõ được việc tăng hiệu suất đáng kể lắm. Bởi hầu hết các ứng dụng không được xử lý đa luồng. Chính vì vậy các ứng dụng cũng cần phải thay đổi trong thiết kế. Ví dụ một chương trình quét virus chạy trên một tuyến mới trong khi GUI lại chạy trên một tuyến khác. Việc xử lý đa luồng đúng cách là khi khối lượng công việc được phân chia thành nhiều luồng khác nhau. Việc quét virus là một ví dụ, luồng GUI làm việc rất ít, trong khi luồng quét virus thực hiện một nhiệm vụ rất nặng và không có khả năng chia nhỏ ra và gửi đến các lõi khác.

Việc phát triển một ứng dụng đa luồng đích thực yêu cầu rất nhiều công việc phức tạp. Điều này rõ ràng cũng tốn khá nhiều chi phí vào một chu trình thiết kế phần mềm. Đó là lý do tại sao phần lớn các ứng dụng phần mềm sẽ không được phát triển như các ứng dụng thực sự đa luồng cho đến khi số lượng lõi đủ cao để thực hiện nhiều tác vụ mà không làm ảnh hưởng tới hiệu suất. Và điều này sẽ đạt được khi người dùng có nhu cầu. Tuy vậy các mạng của bạn còn có nhiều vấn đề khác. Các router có thể trở thành các thiết bị được chấp nhận rộng rãi trước tiên với kiến trúc đa lõi cũng như việc xử lý đa luồng. Các máy chủ cũng sẽ tăng hiệu suất đáng kể từ công nghệ mới này. Trong số các bạn có ai cho rằng vẫn chưa có các sản phẩm đa lõi? Điều này hoàn toàn có thể được cũng như điều tôi đã nói về bước tiến quan trọng đối với tầm quan trọng của lõi.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 3

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Nêu cấu tạo, chức năng của bộ xử lý trung tâm CPU?

Câu 2. Nêu cấu tạo, chức năng của đơn vị điều khiển CU?

Câu 3. Trình bày các phương pháp thiết kế đơn vị điều khiển CU?

Câu 4. Nêu cấu tạo, chức năng của đơn vị tính toán số học và logic ALU?

Câu 5. Vẽ hình để mô tả kỹ thuật đường ống. Kỹ thuật đường ống làm tăng tốc độ CPU lên bao nhiêu lần (theo lý thuyết)? Tại sao trên thực tế sự gia tăng này lại ít hơn?

Câu 6. Các khó khăn trong kỹ thuật đường ống pipeline?

Câu 7. Trình bày cách giải quyết khó khăn trong kỹ thuật đường ống lệnh pipeline?

Câu 8. Trình bày về sự khác biệt giữa bộ xử lý đa luồng và đa lõi?

Câu 9. Trình bày kỹ thuật song song mức lệnh?

Câu 10. Trình bày các dạng phụ thuộc song song mức lệnh?

Câu hỏi trắc nghiệm

Câu 1. Bộ xử lý trung tâm CPU gồm có:

A. Khối điều khiển và bộ nhớ trong.

B. Khối tính toán số học, Logic và bộ nhớ trong.

C. Khối điều khiển, khối tính toán số học và Logic, tập các Thanh ghi.

D. Tập các Thanh ghi và bộ nhớ.

Câu 2. Cấu trúc phần cứng của máy tính

CPU, ALU, Bộ nhớ trong. CPU, hệ thống vào ra, Bộ nhớ trong.

CPU, CU, ALU. CPU, CU, Bộ nhớ trong.

Câu 3. Đơn vị điều khiển (CU) có nhiệm vụ:

A. Làm các phép tính trên các số liệu.

B. Lấy chỉ thị từ bộ nhớ chính, giải mã và điều khiển ALU.

C. Thi hành các chương trình được chứa trong bộ nhớ chính.

D. Tác động vào các mạch điện để thi hành các lệnh.

Câu 4. Đơn vị tính toán số học và Logic (ALU) có nhiệm vụ;

A. Làm phép tính trên các số liệu

B. Đảm bảo thi hành các lệnh một cách tuần tự

C. Tác động vào các mạch điện để thi hành các lệnh

D. Lấy lệnh từ bộ nhớ trong

Câu 5. Kỹ thuật ống dẫn có đặc điểm:

A. Làm giảm tốc độ thực hiện các lệnh

B. Làm tăng tốc độ thực hiện các lệnh

C. Có ít thanh ghi khác nhau dùng cho các tác vụ đọc viết

D. Tất cả các giai đoạn của lệnh không được thi hành cùng một lúc

Câu 6. T_{seq} thời gian thực hiện tuần tự, $T_{pipeline}$ thời gian thực hiện pipeline. Gọi S (speedup) là hệ số tăng tốc

A. S = Tseq/Tpipeline

B. S = Tseq + 1/Tpipeline

C. S = Tseq/Tpipeline - 1

D. S = Tseq + Tpipeline/Tpipeline

Câu 7. Cho Hệ thống pipeline có 5 công đoạn, thực hiện 5 lệnh, hệ số tăng tốc là

A. 9/25 B. 9/24 C. 25/9 D. 34/9

Câu 8. Cho hệ thống pipeline, có 5 công đoạn, biết hệ số tăng tốc là 20/8. Hỏi hệ thống đã thực hiện bao nhiêu lệnh?

A. 4 B. 5 C. 6 D. 3

Câu 9. Các khó khăn (xung đột) gặp phải khi sử dụng đường ống lệnh pipeline?

A. Xung đột điều khiển

B. Xung đột điều khiển, xung đột cấu trúc

C. Xung đột cấu trúc, xung đột dữ liệu, xung đột điều khiển

D. Xung đột CPU

Câu 10. Xung đột dữ liệu bao gồm?

A. RAW B. WAR C. WAW D. RAW, WAR, WAW

Bài tập áp dụng

Câu 1. Cho hệ thống Pipeline gồm 5 giai đoạn: IF (nạp lệnh), ID (giải mã lệnh), OF (nạp tham số), EX (thực hiện lệnh), OS (lưu kết quả). Cho một chuỗi các lệnh như sau:

1) MOV R1,x
2) MOV R2,y
3) SUB R3,R2,R1
4) MOV R4,z
5) MUL R5,R1,R4
6) SUB R6,R2,R3

Tính hệ số tăng tốc $S=T_t/T_p$ theo lý thuyết và theo thực tế khi thực hiện chuỗi lệnh trên?

Câu 2. Cho hệ thống Pipeline gồm 5 giai đoạn: IF (nạp lệnh), ID (giải mã lệnh), OF (nạp tham số), EX (thực hiện lệnh), OS (lưu kết quả). Cho một chuỗi các lệnh như sau:

7) MOV R_1, x
8) MOV R_2, y
9) SUB R_2, R_2, R_1
10) MOV R_3, z
11) MUL R_4, R_1, R_3
12) SUB R_5, R_2, R_4

Tính hệ số tăng tốc $S=T_t/T_p$ theo lý thuyết và theo thực tế khi thực hiện chuỗi lệnh trên?

Câu 3. Cho hệ thống Pipeline gồm 5 giai đoạn: IF (nạp lệnh), ID (giải mã lệnh), OF (nạp tham số), EX (thực hiện lệnh), OS (lưu kết quả). Cho một chuỗi các lệnh như sau:

1) MOV R_1, x
2) MOV R_2, y
3) ADD R_2, R_2, R_1
4) MOV R_3, z
5) DIV R_4, R_1, R_3
6) ADD R_5, R_2, R_4

Tính hệ số tăng tốc $S=T_t/T_p$ theo lý thuyết và theo thực tế khi thực hiện chuỗi lệnh trên?

Câu 4. Cho hệ thống pipeline, có 5 công đoạn, biết hệ số tăng tốc là 20/8. Hỏi hệ thống đã thực hiện bao nhiêu lệnh?

Câu 5. Giả sử hệ thống pipeline, thực hiện 6 lệnh, biết hệ số tăng tốc là 24/9. Hỏi hệ thống có mấy công đoạn?

Câu 6. Giả sử hệ thống pipeline, thực hiện 6 lệnh, Mỗi lệnh gồm 5 công đoạn. Hỏi hệ thống có hệ số tăng tốc là bao nhiêu?

Câu 7. Giả sử hệ thống pipeline, thực hiện 10 lệnh, Mỗi lệnh gồm 6 công đoạn. Hỏi hệ thống có hệ số tăng tốc là bao nhiêu?

Câu 8. Giả sử hệ thống pipeline thực hiện n lệnh, Mỗi lệnh gồm 6 công đoạn. Hệ số tăng tốc là 24/9. Hỏi hệ thống thực hiện bao nhiêu lệnh?

Câu 9. Giả sử hệ thống pipeline thực hiện n lệnh, Mỗi lệnh gồm 5 công đoạn. Hệ số tăng tốc là 25/9. Hỏi hệ thống thực hiện bao nhiêu lệnh?

Câu 10. Vẽ sơ đồ khối của bộ xử lý trung tâm CPU và bus hệ thống?

Chương 4: THIẾT KẾ BỘ NHỚ

Mục đích: *Giới thiệu mô hình phân cấp bộ nhớ, cấu trúc một phần tử nhớ, cách thiết kể bộ nhớ có dung lượng xác định, tìm hiểu bộ nhớ cache, tác dụng của bộ nhớ cache, các phương pháp thay thế dữ liệu. Tìm hiểu bộ nhớ ảo và kỹ thuật phân trang, bộ nhớ ảo và kỹ thuật phân đoạn, các phương pháp thay thế khung trang. Hiện tượng ngắt và loại trừ.*

Yêu cầu: *Sinh viên phải nắm vững cấu trúc một phần tử nhớ và biết cách thiết kế bộ nhớ, hiểu được cơ chế của kỹ thuật phân trang và kỹ thuật phân đoạn. Áp dụng lý thuyết thay thế dữ liệu vào làm bài tập.*

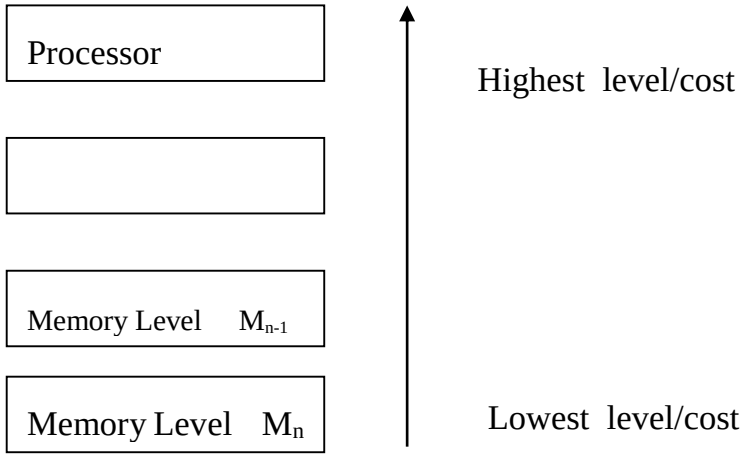
4.1. Mô hình phân cấp bộ nhớ

4.1.1 Sự cần thiết của phân cấp bộ nhớ

Trong quá trình thực hiện các chương trình người ta thống kê thời gian để truy nhập dữ liệu chiếm một phần rất lớn trong tổng thời gian thực hiện của toàn bộ chương trình. Đây chính là thời điểm thắt cổ chai của hệ thống (bottle neck). Vì thế người ta sẽ tìm cách để làm giảm tổng thời gian truy nhập bộ nhớ sao cho dữ liệu hầu như luôn sẵn sàng với Bộ xử lý.

4.1.2 Quy tắc chung của hệ thống phân cấp bộ nhớ

Đưa ra mô hình phân cấp bộ nhớ:



Hình 4.1 Mô hình phân cấp bộ nhớ

Mối quan hệ về thời gian (Temporal Locality): Người ta cho rằng các dữ liệu, các lệnh vừa được sử dụng thì tương lai sẽ được sử dụng tiếp. Vì thế người ta tìm cách để lưu các dữ liệu vào các lệnh này để có thể truy nhập trong tương lai.

Mối quan hệ về không gian (Spatial Locality): Theo quan điểm này người ta cho rằng các lệnh và các dữ liệu nằm gần lệnh hoặc là dữ liệu vừa được sử dụng thì trong tương lai sẽ có thể được sử dụng, do vậy người ta phải tìm cách để lưu trữ hoặc nạp trước dữ liệu này.

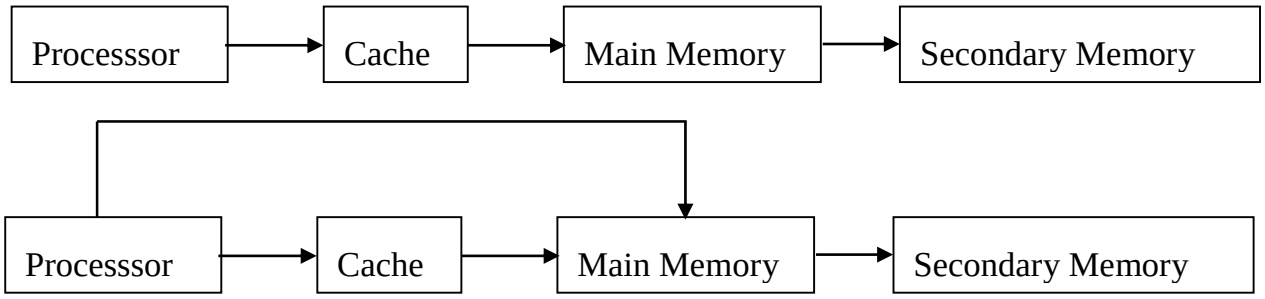
Mối quan hệ tuần tự (Sequential Locality): Người ta đã thống kê được rằng các lệnh trong chương trình hầu hết được thực hiện một cách tuần tự (70-80%), lệnh thực hiện rẽ nhánh và lặp (20-30%) vì thế người ta sẽ tìm cách để nạp hoặc lưu trữ các lệnh nằm trước hoặc sau lệnh đang thực hiện để cho quá trình truy nhập lênh này nhanh hơn.

Trong mô hình phân cấp bộ nhớ ta sẽ tìm cách ghép nối các bộ nhớ có tốc độ truy cập nhanh kết hợp với các bộ nhớ có tốc độ truy cập thấp, sao cho tổng thời gian truy nhập một mục dữ liệu/một lệnh và tổng giá thành của bộ nhớ là thấp nhất. Vì các bộ nhớ càng gần bộ xử lý có tốc độ càng cao giá thành càng đắt, nên không thể sử dụng chúng với dung lượng lớn mà chỉ sử dụng chúng đóng vai trò trung gian, khi đó để nạp hoặc ghi dữ liệu thì ta phải xây dựng những thuật toán, thông thường áp dụng một trong ba quan điểm trên.

Mô hình bộ nhớ hai cấp

Hình 4.2 Mô hình bộ nhớ hai cấp

Mô hình bộ nhớ ba cấp

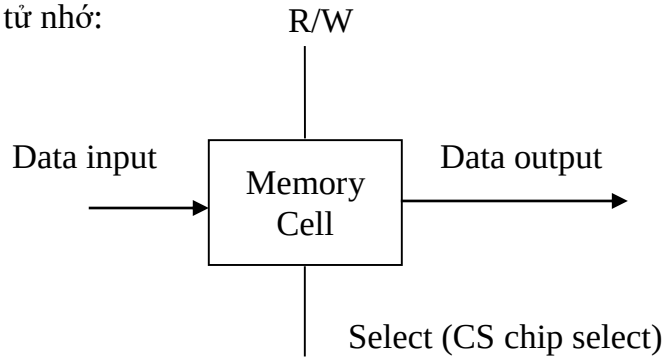


Hình 4.3 Mô hình bộ nhớ ba cấp

4.1.3 Phần tử nhớ và bộ nhớ

Là đơn vị nhỏ nhất được sử dụng để cấu tạo nên bộ nhớ mỗi phần tử nhớ này thường là một mạch Flip-Flop hoặc một tổ hợp các transistor.

Sơ đồ cấu tạo của phần tử nhớ:



Hình 4.4 Cấu tạo phần tử nhớ

Mỗi phần tử nhớ sẽ lưu trữ được một bit dữ liệu, để xây dựng thành bộ nhớ có các dung lượng cao hơn người ta phải kết nối các phần tử nhớ lại với nhau. Khi bộ nhớ cần thiết kế, có một từ nhớ là n bit suy ra phải kết nối n phần tử nhớ lại với nhau và các phần tử nhớ này phải có chung đường tín hiệu select. Khi đó, mỗi khi có tín hiệu lựa chọn thì tất cả các phần tử nhớ này sẽ được lựa chọn đồng thời.

Ví dụ: 16Kilobyte cần số Cell $2^4 * 2^{10}*2^3 = 2^{17}$ Cell.

Ví dụ : Sơ đồ một bộ nhớ gồm có 4 ô nhớ, mỗi ô nhớ hai bit suy ra cần 8 cell.

Nếu bộ giải mã có hai đầu vào và 4 đầu ra đặt là: LS 124.

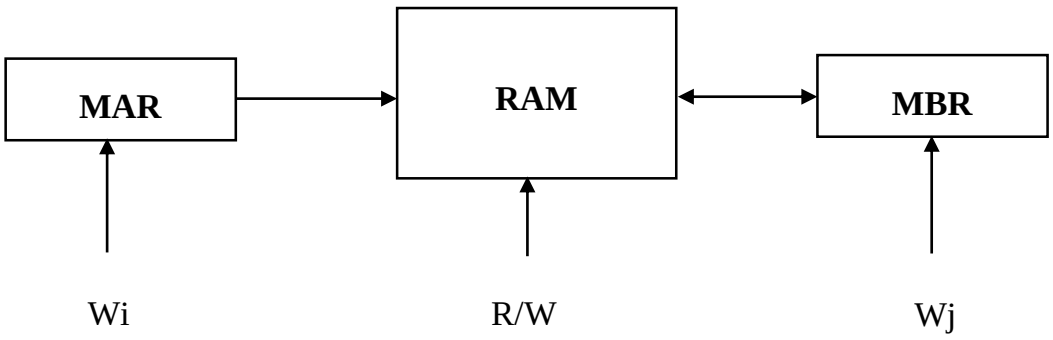
Nếu bộ giải mã có 3 đầu vào và 8 đầu ra đặt là: LS 138.

Phân 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), bộ nhớ chỉ đọc (ROM) đều được chế tạo bằng chất bán dẫn; và **bộ nhớ ngoài** bao gồm: đĩa cứng, đĩa mềm, băng từ, trống từ, các loại đĩa quang, các loại thẻ nhớ,...

a. Bộ nhớ RAM (Random access Memory):

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 4.5 Vận hành của bộ nhớ RAM

Tuỳ 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 huỷ 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 huỷ. 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.

b. Bộ nhớ chỉ đọc ROM (Read only Memory):

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 tắc 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. [1]

Kiểu bộ nhớ	Loại	Cơ chế xóa	Cơ chế ghi	Tính bay hơi
RAM	Đọc/ghi	Bằng điện, mức byte	Bằng điện	Có
ROM	Chỉ đọc	Không thể xóa	Mặt nạ	Không
Programmable ROM (PROM)			Bằng điện	
Erasable PROM	Hầu hết chỉ đọc	Tia cực tím, mức chip		
Electrically Erasable PROM (EEPROM)		Bằng điện, mức byte		
Flash Memory		Bằng điện, mức khối		

Bảng 4.1 Các kiểu bộ nhớ bán dẫn

4.2. Xây dựng bộ nhớ

4.2.1 Các bước tiến hành xây dựng bộ nhớ

Bước 1: Xác định số lượng các IC/ chip cần sử dụng để xây dựng bộ nhớ.

$$A = \frac{\textit{Dung lượng bộ nhớ * số các bit đầu ra}}{\textit{Dung lượng IC * số bit ra của IC}}$$

Bước 2: Xác định số lượng đường địa chỉ cần sử dụng cho bộ nhớ

$$n_1 = \{[\log 2(\text{dung lượng bộ nhớ})]\}.$$

Bước 3: Xác định số lượng đường địa chỉ cần sử dụng cho IC nhớ.

$$n_2 = [\log 2\ (\text{dung lượng IC})].$$

Bước 4 : Xác định số lượng IC cần mở đồng thời :

$$B = \frac{\text{so dau ra cua bo nho}}{\text{so dau ra cua IC}}$$

Bước 5: Thiết kế bộ nhớ giải mã gồm một số đầu vào hoặc đầu ra xác định.

n_1-n_2 đầu vào.

$\frac{A}{B}$ đầu ra.

Bước 6: Vẽ hình.

Chú ý:

- Trong trường hợp số lượng bit đầu ra của IC mà lớn hơn bit đầu ra của bộ nhớ thì ta phải sử dụng cả IC (sử dụng số nguyên lần IC đó).

- Trong trường hợp số lượng các bit đầu ra của bộ nhớ nhiều hơn số đầu ra của IC thì ta phải mở đồng thời nhiều nhiều IC bằng cách nối chung đường CS của chúng với nhau.

- Khi vẽ hình nếu có nhiều IC hoặc nhiều phần giống nhau thì ta có thể vẽ đại diện cho những phần giống nhau đó.

- Các đường địa chỉ sử dụng trong hình vẽ thì được đánh số từ A_0 trở đi. Ưu tiên các đường địa chỉ có chỉ số thấp để nối vào các IC nhớ, các đường địa chỉ có chỉ số cao thì nối vào bộ giải mã.

- Các đường dữ liệu được đánh số từ D_0 đi, nếu ta muốn ghép nhiều đầu ra dữ liệu của các IC thì các đầu ra này phải được đánh số liên tiếp nhau. Ví dụ $D_o \rightarrow D_7$ và $D_8 \rightarrow D_{15}$.

- Khi thiết kế bộ giải mã ta phải tuân thủ các bước sau:

Bước 1: Lập bảng chân lý.

Bước 2: Lập hàm logic.

Bước 3: Vẽ mạch.

Trong trường hợp các bộ giải mã là quen thuộc (bộ giải mã 24, 38) thì ta không cần phải làm bước này.

4.2.2 Ví dụ minh họa

Ví dụ 1: Xây dựng bộ nhớ có dung lượng 1M * 8bit từ IC 256K * 8bit.

D_0-D_7 phụ thuộc vào dung lượng của bộ nhớ

Bước 1: Xác định số lượng các IC/ chip cần sử dụng để xây dựng bộ nhớ.

$$A = \frac{\textit{Dung lượng bộ nhớ * số các bit đầu ra}}{\textit{Dung lượng IC * số bit ra của IC}}$$

$$A = \frac{2^{20} * 8}{2^{18} * 8} = 4IC$$

Bước 2: Xác định số lượng đường địa chỉ cần sử dụng cho bộ nhớ

n_1= {[log 2(dung lượng bộ nhớ)]} $n_1 = [\log 2 (2^{20})] = 20$

Bước 3: Xác định số lượng đường địa chỉ cần sử dụng cho IC nhớ

n_2 = [log2 (dung lượng IC)] $n_2 = [\log 2 (2^{18})] = 18$

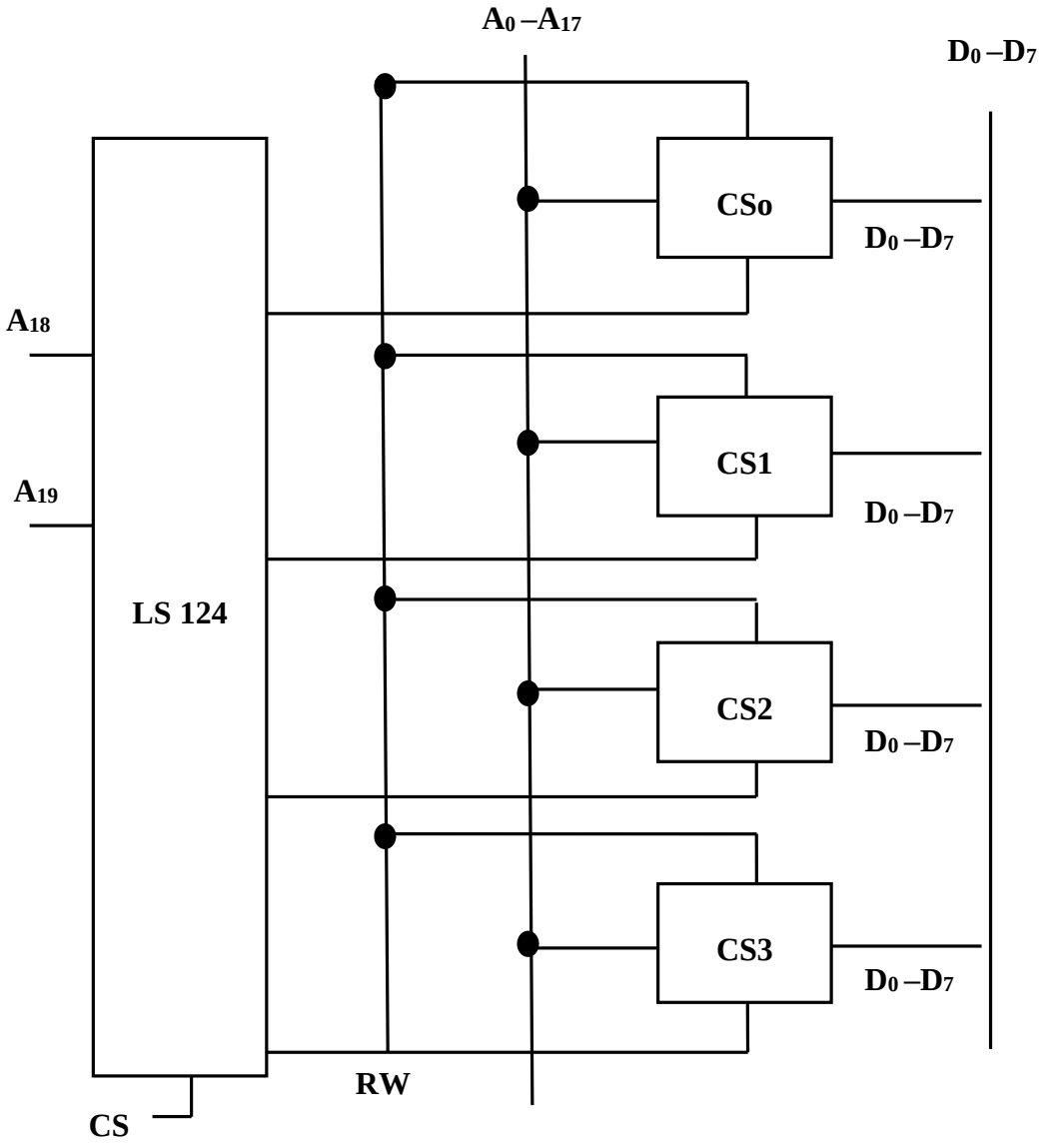
Bước 4 : Xác định số lượng IC cần mở đồng thời :

$$B = \frac{\text{so dau ra cua bo nho}}{\text{so dau ra cua IC}} = \frac{8}{8} = 1$$

Bước 5: Thiết kế bộ nhớ giải mã gồm một số đầu vào hoặc đầu ra xác định.

Đầu vào $n_1 - n_2 = 20-18 = 2$ Đầu ra $\frac{A}{B} = \frac{4}{1} = 4$

Bước 6: Vẽ hình. Sử dụng bộ giải mã LS 124.



Hình 4.6 Xây dựng bộ nhớ 1MB * 8 bit từ IC 256KB * 8 bit

Ví dụ 2 : Xây dựng bộ nhớ có dung lượng 1 M * 32 bit từ IC 256 * 8bit.

Bước 1 - Tính số IC: $A = \frac{2^{20} * 32}{2^{18} * 8} = 16 \ IC$

Bước 2 - Số đường địa chỉ cần sử dụng cho bộ nhớ: $n_1 = [\log 2 (2^{20})] = 20$

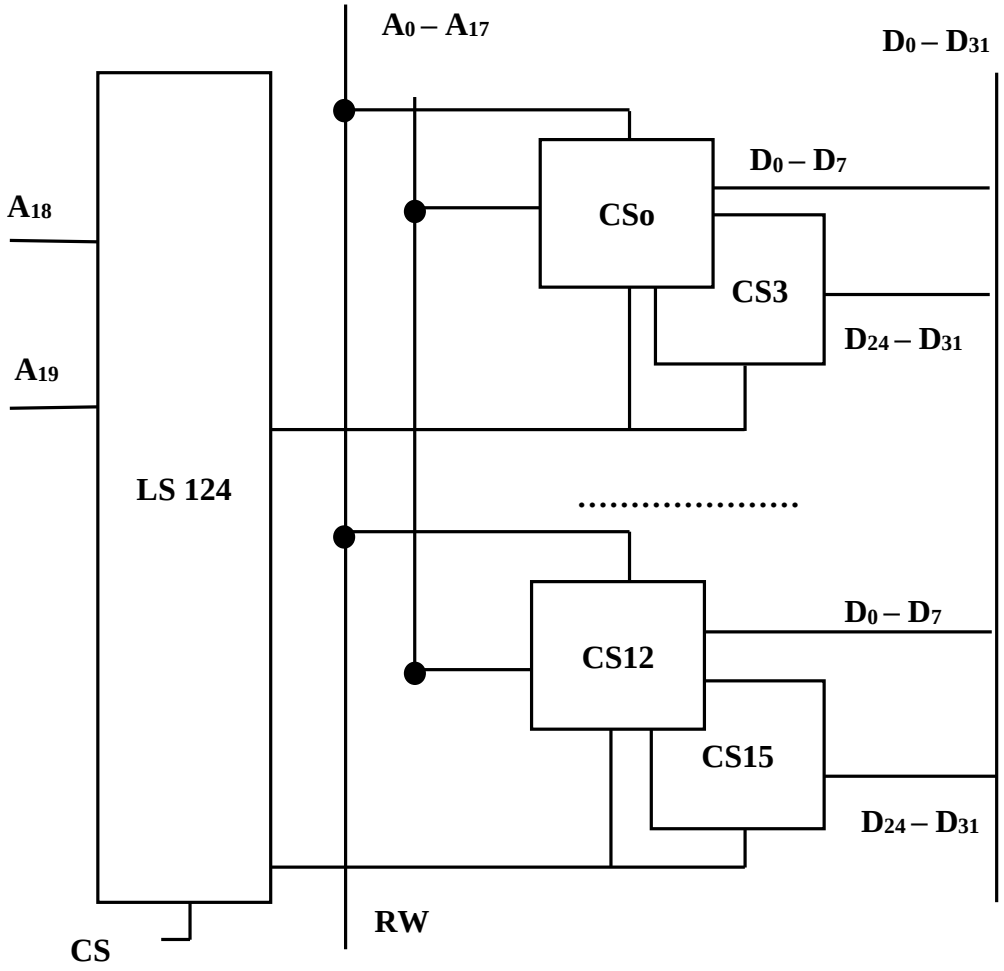
Bước 3 - Số đường địa chỉ cần sử dụng cho IC: $n_2 = [\log 2 (2^{18})] = 18$

Bước 4 - Số IC mở đồng thời: B = 32/ 8 = 4.

Bước 5 - Bộ giải mã gồm: Đầu vào $n_1 - n_2 = 2$.

Đầu ra $\frac{A}{B} = \frac{16}{4} = 4$

Bước 6 - Vẽ hình sử dụng bộ giải mã LS 124



Hình 4.7 Xây dựng bộ nhớ 1MB * 32 bit từ IC 256KB * 8 bit

Ví dụ 3 Thiết kế bộ nhớ có dung lượng 512 K * 8bit, từ IC cơ sở 256 K * 8 bit.

Bước 1 - Tính số IC: $A = \frac{2^9 * 2^{10} * 8}{2^8 * 2^{10} * 8} = 2IC$

Bước 2 - Số đường địa chỉ cần sử dụng cho bộ nhớ: $n_1 = [\log 2 (2^{19})] = 19$

Bước 3 - Số đường địa chỉ cần sử dụng cho IC: $n_2 = [\log 2 (2^{18})] = 18$

Bước 4 - Số IC mở đồng thời: B = 8/ 8 = 1

Bước 5 - Bộ giải mã gồm: Đầu vào $n_1 - n_2 = 1$

Đầu ra $\frac{A}{B} = \frac{2}{1} = 2$

Xây dựng bộ giải mã gồm 1 đầu vào và 2 đầu ra.

Bước 1: bảng chân lý

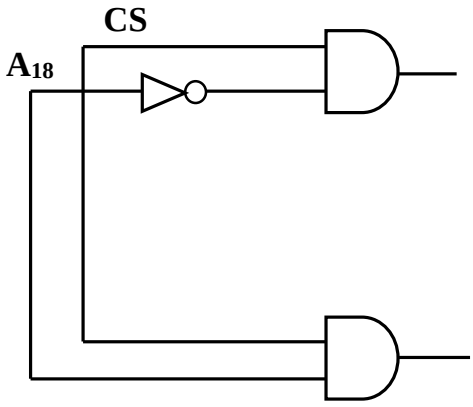
CS	A_{18}	CS_0	CS_1
1	0	1	0
1	1	0	1

Bước 2: Lập hàm Logic

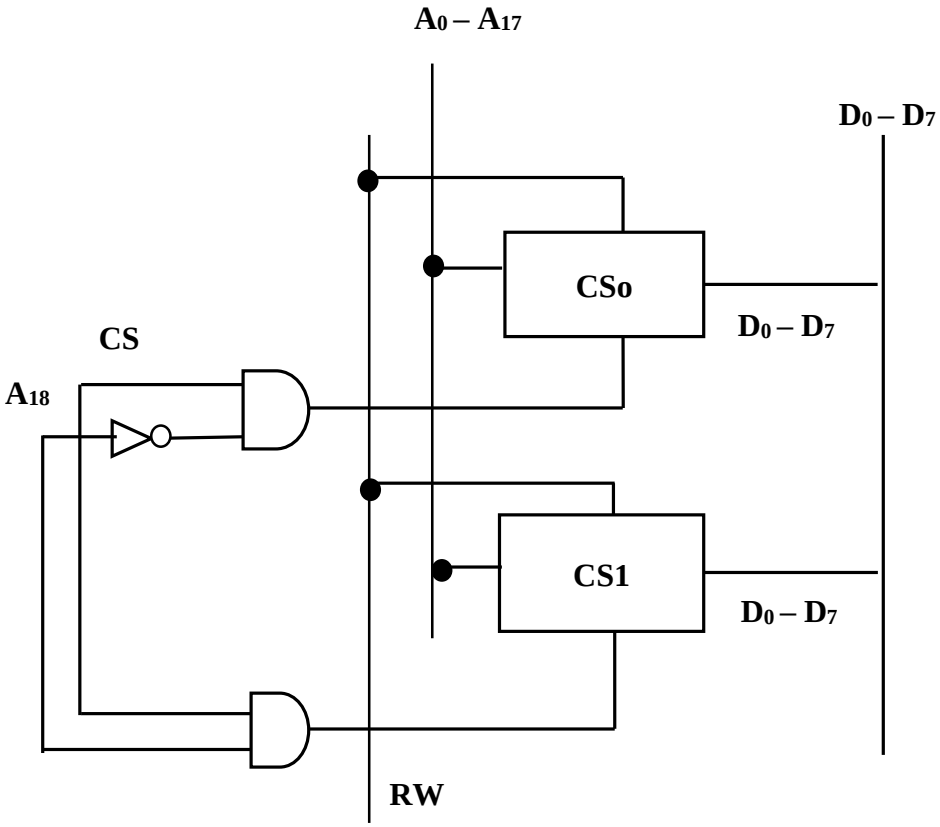
$CS_0 = CS * \overline{A}_{18}$

$CS_1 = CS * A_{18}$

Bước 3: vẽ mạch



Bước 6: vẽ hình sử dụng bộ giải mã 1 đầu vào, 2 đầu ra



Hình 4.8 Xây dựng bộ nhớ 512KB * 8 bit từ IC nhớ 256KB * 8 bit

4.3. Bộ nhớ cache

4.3.1 Một số khái niệm

Cache là thành phần nhớ trong sơ đồ phân cấp bộ nhớ máy tính. Nó hoạt động như một thành phần trung gian, trung chuyển dữ liệu từ bộ nhớ chính về CPU và ngược lại.

Mục đích cơ bản của cache là lưu trữ các lệnh chương trình thường xuyên được tham chiếu lại bởi phần mềm trong khi hoạt động. Truy cập nhanh vào các hướng dẫn này làm tăng tốc độ tổng thể của chương trình phần mềm.

Khi bộ vi xử lý xử lý dữ liệu, nó trông đầu tiên trong bộ nhớ cache, nếu nó tìm thấy các hướng dẫn ở trước đó (về việc đọc dữ liệu trước đó) nó không cần phải đọc nhiều dữ liệu hơn từ bộ nhớ lớn hơn, hoặc các thiết bị lưu trữ dữ liệu khác. Hầu hết các chương trình sử dụng rất ít tài nguyên khi chúng đã được mở và hoạt động trong một thời gian, chủ yếu là do các hướng dẫn thường xuyên tham chiếu lại có xu hướng được lưu trữ. Điều này giải thích lý do tại sao các phép đo hiệu năng hệ thống trong các máy tính có bộ vi xử lý chậm hơn nhưng bộ nhớ cache lớn hơn có xu hướng nhanh hơn các phép đo hiệu năng hệ thống trong các máy tính có bộ vi xử lý nhanh hơn nhưng không gian bộ nhớ bị giới hạn.

4.3.1.1 Vị trí của cache

Với các hệ thống cũ, cache thường nằm ngoài CPU. Với các CPU mới, cache thường được tích hợp vào trong CPU. Nó nằm giữa bộ vi xử lý và bộ nhớ chính.

Hình 4.9 Hình ảnh vị trí của cache

4.3.1.2 Vai trò của cache

Khi tốc độ làm việc của bộ vi xử lý ngày càng vượt xa tốc độ truy nhập bộ nhớ chính (được tính bằng nano giây-ns, DRAM làm việc nhanh nhất chỉ là 60ns) có nghĩa là bộ vi xử lý phải mất thêm vài chu kỳ đợi bộ nhớ hoàn thành quá trình đọc/ghi. Điều này làm giảm hiểu suất làm việc của bộ vi xử lý. Một giải pháp hữu hiệu là sử dụng thêm bộ nhớ đệm cache với tốc độ truy nhập chỉ vài ns đến 10ns. Bộ nhớ cache còn được gọi là bộ nhớ truy cập nhanh. Nâng cao hiệu năng hệ thống.

Cache sẽ tiết kiệm thời gian truy xuất bộ nhớ của CPU bằng cách dự đoán trước các lệnh kế tiếp mà CPU sẽ cần và nạp nó vào trong cache trước khi mà CPU thực sự cần đến nó. Nếu lệnh cần thiết đã có sẵn trong cache thì CPU sẽ truy xuất dữ liệu từ cache, nếu không, CPU mới truy xuất trên bộ nhớ chính. Cache đóng vai trò là một nơi lưu trữ tạm thời những lệnh mà CPU cần xử lý.

Nâng cao hiệu năng hệ thống

Dung hòa giữa CPU có tốc độ cao và bộ nhớ chính có tốc độ thấp (giảm số lượng truy cập trực tiếp của CPU vào bộ nhớ chính). Thời gian trung bình CPU truy cập hệ thống bộ nhớ gần bằng thời gian truy cập cache.

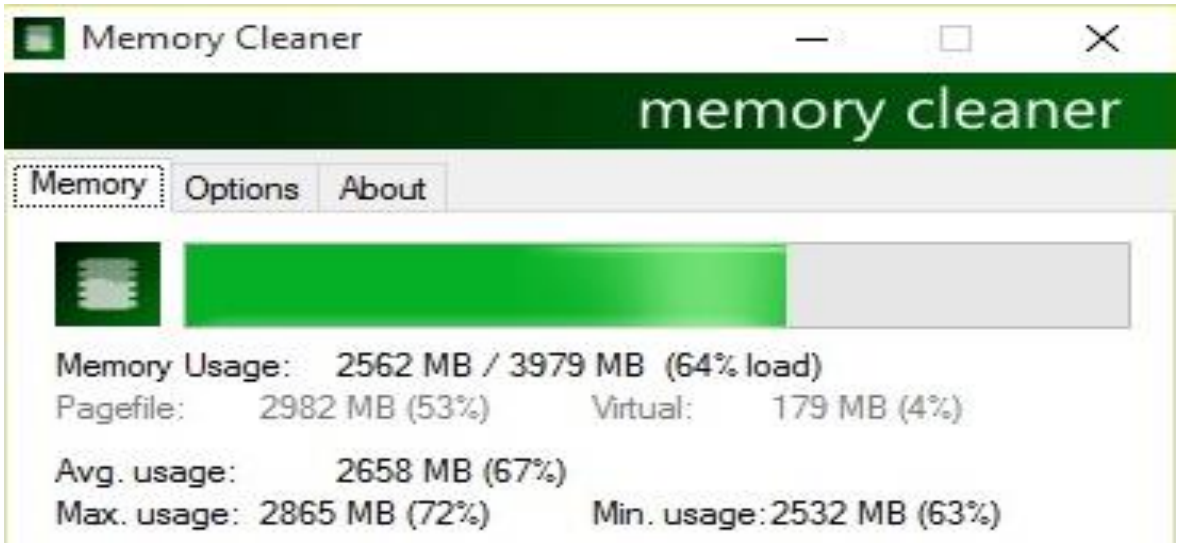
Giảm giá thành sản xuất

Nếu hai hệ thống có cùng hiệu năng, hệ thống có cache sẽ rẻ hơn. Nếu hai hệ thống cùng giá thành, hệ thống có cache sẽ nhanh hơn.

4.3.1.3 Dung lượng

Dung lượng bộ nhớ cache thường nhỏ. Với các hệ thống cũ là 16KB, 32KB,…,128KB. Với các hệ thống mới hiện nay, dung lượng có thể đạt được là 256KB, 512KB, 1MB, 2MB,…

Dung lượng đối với từng loại cache: L1 cache thường có dung lượng chỉ vài chục KB: từ 8KB -32KB. L2 cache thường có dung lượng khoảng vài trăm KB đến vài MB: 256KB, 512KB, 1MB – 8MB. L3 cache thường có dung lượng khoảng vài MB.



Hình 4.10 Hình ảnh về dung lượng cache

4.3.1.4 Phân loại cache

Cache gồm 2 loại: cache sơ cấp (primary cache) và cache thứ cấp.

Cache sơ cấp

Bộ nhớ sơ cấp (primary cache) còn được gọi là cache cấp một (cache L1) hoặc bộ đệm chính, nằm trên CPU và được sử dụng để lưu trữ tạm thời các hướng dẫn và dữ liệu được tổ chức theo các khối 32 byte. Bộ nhớ cache chính là hình thức lưu trữ nhanh nhất. Bởi vì nó được tích hợp vào chip với giao diện chờ đợi (trạng thái chờ) bằng không đến bộ xử lý của bộ xử lý nên nó bị giới hạn về kích thước.

Bộ nhớ cache sơ cấp được triển khải bằng cách sử dụng RAM tĩnh (SRAM) và cho đến gần đây có kích thước 16KB truyền thống. SRAM sử dụng hai bóng bán dẫn cho mỗi bit và có thể chứa dữ liệu mà không cần sự hỗ trợ từ bên ngoài, miễn là nguồn điện được cung cấp cho mạch điện. Bóng bán dẫn thứ hai điều khiển đầu ra của bóng đầu tiên: một mạch được gọi là flip-flop. Điều này tương phản với RAM động (DRAM) phải được làm mới nhiều lần mỗi giây để giữ nội dung dữ liệu của nó.

SRAM được sản xuất theo cách khá giống với cách xử lý, các mẫu transistor tích hợp cao được khắc vào silicon. Mỗi bit SRAM bao gồm từ bốn đến sáu bóng bán dẫn, đó là lý do tại sao SRAM chiếm nhiều không gian hơn so với DRAM, chỉ sử dụng một (cộng với một tụ điện). Điều này, cộng với thực tế là chi phí của SRAM cũng gấp nhiều lần DRAM. Do đó mà nó không được sử dụng rộng rãi hơn trong các hệ thống PC. Tốc độ truy cập cache xấp xỉ bằng tốc độ làm việc của CPU, nhưng dung lượng khá nhỏ. Năm 1988, lần đầu tiên cache L1 được thiết kế cho CPU đời 80386XS, sau đó là 80486DX, 80486DX2. Cuối năm 1994, Intel đã giới thiệu sản phẩm CPU 80486DX4 với tốc độ 75-120MHz. Lúc này, cache L1 được phân thành hai bộ nhớ với hai chức năng khác nhau:

- Data cache: Để lưu trữ dữ liệu với dung lượng 8KB
- Code cache: Để lưu trữ mã lệnh với dung lượng 8KB tiếp theo

Bộ vi xử lý P55 MMX của Intel, ra mắt vào đầu năm 1997, đáng chú ý là việc tăng kích thước bộ nhớ cache L1 lên 32KB.Đối với tất cả các thiết kế L1, logic điều khiển của cache chính giữa giữ liệu và mã được sử dụng thường xuyên nhất trong bộ nhớ đệm và cập nhật bộ nhớ ngoài chỉ khi CPU kiểm soát các bus khác hoặc trong quá trình truy cập bộ nhớ trực tiếp bằng các thiết bị ngoại vi như ổ đĩa quang và cap âm thanh

Cache thứ cấp

Hầu hết các máy tính đều được cung cấp với bộ nhớ cache thứ cấp (còn được gọi là bộ nhớ cache cấp hai) để thu hẹp khoảng cách hiệu suất bộ xử lý/ bộ nhớ. Bộ nhớ cache cấp hai sử dụng một logic điều khiển như bộ nhớ cache cấp một và cũng được triển khai trong SRAM.

Bộ nhớ cache cấp hai thường có hai kích cỡ: 256KB hoặc 512KB, và có thể được tìm thấy hoặc được hàn vào bo mạch chủ, trong ổ cắm thẻ nhớ Low Edge (CELP) hoặc gần đấy hơn trên COAST (cache on a stick) mô-đun. Sau này giống như một SIMM nhưng ngắn hơn một chút và cắm vào một ổ cắm COAST, thường nằm gần bộ vi xử lý và giống như một khe cắm mở rộng PCI, Pentium Pro lệch khỏi sự sắp xếp này, chọn bộ nhớ cache cấp hai trên chip xử lý.

Trước kia, tất cả các cache cấp hai đều được gắn lên mainboard, nhưng bắt đầu từ CPU Pentium, cache mức hai được đưa vào cùng một vỏ bọc với CPU (chứ không nằm ngay bên trong CPU như cache cấp một). Để nối CPU tới cache cấp hai, bắt buộc phải sử dụng bus. Bus này được gọi là Bus tuyến sau (Back side Bus), vì thế không thể thấy được Bus do nó nằm kín trong vỏ bọc CPU, ngược lại Bus nối CPU với bộ nhớ nằm ngoài vỏ bọc cache cấp một.

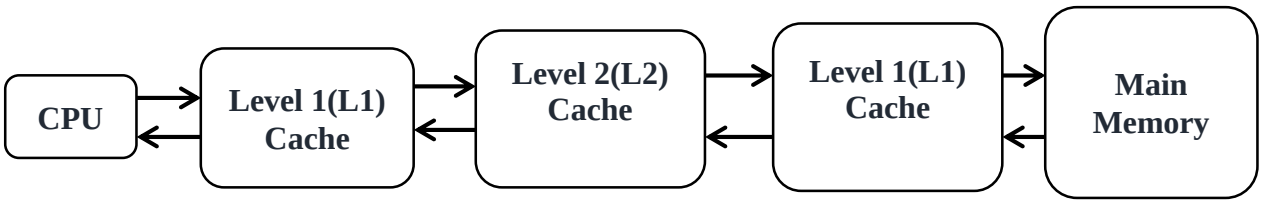
Cache cấp hai có hai loại:

Cache cấp hai rời được gắn trên vi mạch nhỏ và nối với CPU bởi Bus tuyến sau - Back Side Bus có độ rộng là 64 bit. Kiến trúc này làm cho thời gian truy xuất cache cấp hai nhanh hơn so với khi nó được gắn trên bo mạch hệ thống. Nhưng cũng chỉ bằng nửa tốc độ làm việc của CPU, có nghĩa là tốc độ truy xuất của CPU tới cache cấp hai không thể nhanh bằng truy xuất lên cache cấp một.

Cache cấp hai ATC được tích hợp trên bề mặt tại cùng một vị trí của lõi CPU, và nối với CPU bằng Bus có độ rộng là 256 bit. Tốc độ truy xuất dữ liệu bằng với tốc độ làm việc của bộ vi xử lý. Do đó, cache này còn được gọi là Advanced Transfer Cache (ATC). Tuy vậy, ATC cache vẫn không được xem là cache cấp một vì nó không nằm trong một vi chip với CPU mà chỉ nằm trong cùng một vỏ bọc với CPU mà thôi.

Mục đích của bộ nhớ cache cấp hai là cung cấp thông tin được lưu trữ cho bộ xử lý mà không có bất kỳ sự chậm trễ nào (trạng thái chờ). Với mục đích này, giao diện bus của bộ xử lý có giao thức truyền đặc biệt gọi là chế độ chụp liên tục. Chu kỳ bùng nổ bao gồm bốn lần truyền dữ liệu, trong đó chỉ có địa chỉ của 64 bit đầu tiên là đầu ra trên bus địa chỉ. Bộ nhớ cache cấp hai phổ biến nhất là cụm đường ống đồng bộ.

Để có một bộ nhớ đệm đồng bộ, một chipset như Triton được yêu cầu để hỗ trợ nó. Nó có thể cung cấp hiệu suất máy tính tăng 3-5% vì nó được hẹn giờ đến một chu kỳ đồng hồ. Điều này đạt được bằng cách sử dụng công nghệ SRAM chuyên dụng đã được phát triển để cho phép truy cập trạng thái chờ bằng không cho các chu kỳ đọc liên tục. Ram tĩnh có thời gian truy cập trong khoảng từ 4.5ns đến 8ns và cho phép thời gian truyền 3-1-1-1 cho tốc độ bus lên tới 133MHz. Những con số này đề cập đến ba chu kỳ đồng hồ cho mỗi lần truy cập đọc bộ nhớ chế độ chụp (Ví dụ: 3-1-1-1 đề cập đến ba chu kỳ đồng hồ cho từ đầu tiên, một chu kỳ cho mỗi từ tiếp theo). Đối với tốc độ bus lên tới 66MHz, RAM tĩnh đồng bộ cung cấp hiệu suất nhanh hơn có khả năng lặp 2-1-1-1. Tuy nhiên, với tốc độ bus trên 66MHz, hiệu suất của nó giảm xuống còn 3-2-2-2.



Hình 4.11 Phân cấp bộ nhớ cache

Ngoài ra, còn bộ nhớ đệm không đồng bộ, chậm và rẻ hơn rất nhiều vì nó không được định thời gian cho một chu kỳ đồng hồ. Với SRAM không đồng bộ, có sẵn ở tốc độ từ 12ns - 20ns, tất cả các chu trình đọc đều có thời gian là 3-2-2-2 trên bus CPU 50 - 66MHz. Có nghĩa là hai trạng thái chờ cho chu kỳ lead-off và một trạng thái chờ cho ba lần chuyển tiếp của chu kỳ bùng nổ.

4.3.1.5 Hệ số CacheHit và CacheMiss

Cachehit: là sự kiện CPU truy cập tới mục dữ liệu và mục tin mà tìm được trong cache. Xác suất xảy ra Hit được gọi là hệ số hit, ký hiệu là: H

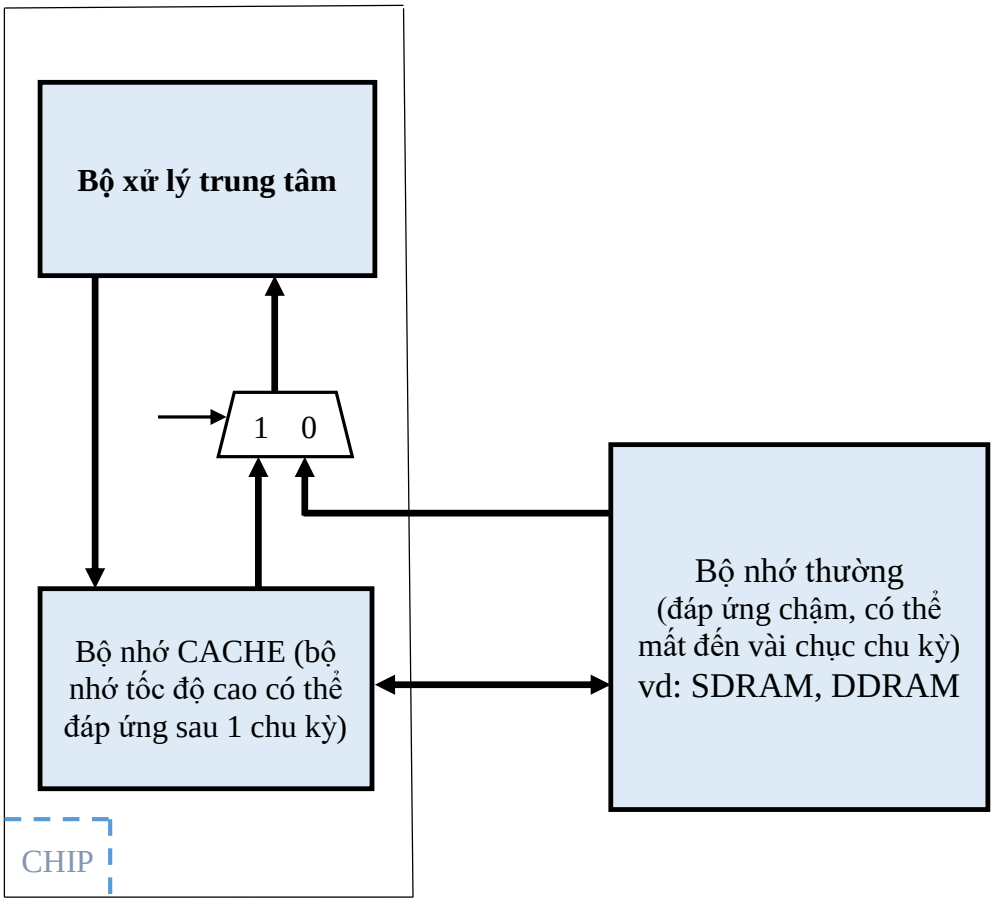
Với điều kiện tồn tại H : 0 <= H <= 1

Giá trị H càng cao thì cache càng tốt.

Cache miss: là sự kiện CPU truy cập tới mục dữ liệu mà không tìm thấy nó trong cache. Khả năng xảy ra Miss được gọi là hệ số miss, ký hiệu là: 1-H

Với điều kiện tồn tại : 0 <= (1-H) <= 1

Hệ số Miss thấp thì hiệu quả cache cao.



Hình 4.12 Mô hình miêu tả Cache hit và Cache miss

Tỷ số trúng cache: $hitratio = \frac{\sum cahehit}{\sum truynhapbncache}$

Chú ý : Trong thực tế hit ratio = 90%.

Ví dụ: Cho Thời gian truy nhập BNC là 100ns. Thời gian truy nhập BN Cache là 10ns. Hit ratio = 90%.

Bài giải: Nếu 10 lần truy nhập Bộ nhớ:

Không có cache : $T_{k\ có\ cache}$ = 10 * 100 =1000 ns.

Có cache : (có 9 lần trúng và một lần trượt)

T_{cache} = 10 * 9 + 100*1 = 190 ns.

→ $\frac{T_{kcache}}{T_{cache}} = \frac{1000}{190}$ = 5.3 lần

4.3.1.6 Thực thi cache

Số lần truy cập có kết quả là cache hit được gọi là hit rate, và có thể được xem là thước đo độ hiệu quả cache của một chương trình hay thuật toán. Lỗi đọc nhầm làm quá trình xử lý bị chậm lại vì cần phải lấy dữ liệu từ bộ nhớ chính nên chậm hơn nhiều so với

đọc từ cache. Lỗi viết nhầm có thể xuất hiện nhưng không nặng đến thế, vì bộ xử lý có thể tiếp tục xử lý khi mà dữ liệu được sao chép vào bộ nhớ chính trong chế độ ngầm.

Để tạo chỗ cho một entry mới khi mà cache miss, cache có thể sẽ hủy một trong số những entry hiện có. Replacement policy (chính sách thay thế) là nguồn gốc của hoạt động trên. Vấn đề cơ bản của bất cứ replacement policy nào là nó phải dự đoán được cache entry đang tồn tại nào sẽ không được dùng nữa. Dự đoán là một chuyện rất khó nên không có cách nào hoàn hảo trong số các replacement policy hiện hữu. Một trong số replacement policy, LRU, thay thế entry được truy cập ít nhất trong thời gian gần đây.

Đánh dấu một số phần bộ nhớ là non-cacheable có thể nâng cao hiệu suất làm việc do tránh được những phần bộ nhớ hiếm khi truy cập. Điều này tránh trường hợp cho một cái gì vào cache mà không tái sử dụng. Cache entries có thể bị tắt hoặc khóa tùy vào trường hợp.

Nếu dữ liệu được viết vào cache, đến lúc nào đó nó cũng sẽ phải được đưa vào bộ nhớ chính; thời điểm viết được biết đến là write policy. Trong write-through cache, mỗi lần viết vào cache thì cũng viết vào bộ nhớ chính. Thay vào đó, trong write-back hay copy-back cache, mỗi lần viết không đồng bộ vào bộ nhớ chính ngay lập tức, mà cache tìm lại những địa điểm đã được viết vào cache, đánh dấu chúng là dirty. Dữ liệu ở những địa điểm này chỉ được viết vào bộ nhớ chính chỉ khi dữ liệu đó bị xóa ở cache. Vì lý do này, một lỗi đọc nhầm trong write-back cache có thể có hai dịch vụ truy cập dữ liệu: một để viết từ dirty location vào bộ nhớ chính, và cái còn lại để đọc vị trí mới từ bộ nhớ chính. Ngoài ra, một lần viết vào một vị trí trong bộ nhớ chính mà chưa được định vị trong write-back cache có thể xóa đi một dirty location, từ đó giải phóng dung lượng cache cho vị trí mới.

Thêm vào đó cũng có nhiều intermediate policy nữa. Cache có thể là write-through, nhưng bản ghi có thể nằm tạm thời trong hàng chờ lưu trữ dữ liệu, thường là để có thể lưu nhiều bản ghi một lúc.

Dữ liệu được cache từ bộ nhớ chính có thể bị thay đổi bởi nhiều yếu tố, trong trường hợp mà bản sao trong cache có thể bị lỗi thời hay mất hiệu lực. Thay vào đó khi một CPU thuộc một hệ thống đa xử lý cập nhật dữ liệu trong cache, các bản sao của dữ liệu trong cache liên quan đến các CPU khác mất hiệu lực. Các quy trình trao đổi giữa các cache manager để thống nhất dữ liệu thì được gọi là cache coherence protocols.

Thời gian lấy một cache line từ bộ nhớ chính rất quan trọng vì CPU sẽ phải đợi cache line. Khi CPU đạt tình trạng này, nó được gọi là stall. Khi CPU trở nên nhanh hơn so với bộ nhớ chính, stalls do có cache misses nên không làm được nhiều phép tính tiềm năng hơn; CPU hiện đại có thể thực hiện hàng trăm chỉ dẫn trong thời gian lấy một cache line từ bộ nhớ chính.

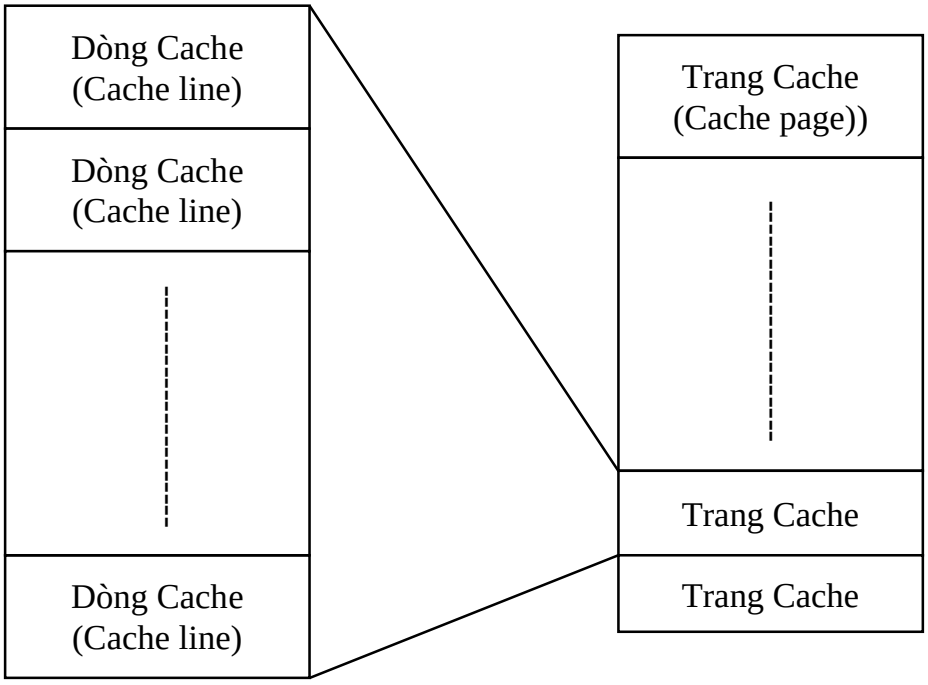
Nhiều phương pháp có thể được sử dụng để giữ CPU làm việc hiệu quả trong thời gian này, bao gồm cả out-of-order execution, mà trong đó CPU cố thực hiện những chỉ dẫn độc lập nằm sau chỉ dẫn đang đợi dữ liệu cache miss. Một công nghệ khác được sử dụng bởi nhiều bộ xử lý là simultaneous multithreading (SMT), hay hyper-threating, cho phép

một thread khác sử dụng lõi CPU trong khi thread đầu tiên đợi tài nguyên của CPU sẵn sàng. [1] [2]

4.3.2 Tổ chức và hoạt động của bộ nhớ cache

4.3.2.1 Tổ chức bộ nhớ cache

Bộ nhớ cache được tổ chức theo kiểu bộ nhớ được địa chỉ hóa bằng chính nội dung dữ liệu của nó (CAM - Content Address Memory). Cụ thể được tổ chức thành các rãnh, mỗi rãnh gồm hai trường: Thẻ và khối dữ liệu.



Hình 4.13 Mô tả Cache page

Cache page: Một bộ nhớ chính được chia làm nhiều phần bằng nhau gọi là cache page. Chú ý rằng, một cache page không liên quan gì đến một trang nhớ (Memory cache) trong chế độ page. Trong cấu trúc word page có nhiều ý nghĩa khác. Kích thước của page phụ thuộc kích thước và sự sắp xếp của Cache.

Cache line: Một cache page được cắt thành những phần nhỏ gọi là cache line. Kích thước cache line được xác định bởi cả CPU và thiết kế của cache.

Giả sử bộ nhớ chính gồm đến 2^n từ nhớ đã được đánh địa chỉ (với mỗi từ nhớ có một địa chỉ duy nhất rộng n bit). Bộ nhớ chính chia thành M khối, mỗi khối có K từ nhớ M = 2n/K. Bộ nhớ cache có C khe mỗi khe có K từ nhớ (C << M). Tại một thời điểm luôn có một tập con các khối nhớ thường trú trong cache. Nếu một từ sẽ được đọc thì khối chứa từ đó sẽ được chuyển vào trong cache. Do số khối nhiều hơn số khe, một khe có thể không được giành cho một khối trong thời gian lâu dài. Vì lý do đó, mỗi khe có một thẻ cho biết khối nào đang được lưu trữ. Thẻ thường là một phần của địa chỉ bộ nhớ chính của khối đang được lưu trữ trong khe. Bộ nhớ cache có thể làm việc với ba cấu hình khác nhau: Ánh xạ

trực tiếp, Liên kết đầy đủ và Liên kết tập hợp. Cấu hình Liên kết tập hợp được dùng hầu hết hiện nay.

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

Bộ nhớ cache được thiết kế với ý định mang lại bộ nhớ có tốc độ xấp xỉ tốc độ của bộ nhớ nhanh nhất hiện có, đồng thời cung cấp một kích thước bộ nhớ lớn với phí tổn ít hơn so với các loại bộ nhớ bán dẫn. Bộ nhớ cache chứa bản sao của một phần bộ nhớ chính. Khi CPU cố gắng đọc một từ ở bộ nhớ, từ này sẽ được kiểm tra xem có trong cache hay không. Nếu có, từ đó sẽ được cung cấp ngay cho CPU. Trong trường hợp ngược lại, một khối bộ nhớ chính, bao gồm một lượng cố định các từ sẽ được đọc vào trong cache và sau đó, từ đó sẽ được cung cấp cho CPU. Do có hiện tượng tham chiếu cục bộ, khi một khối dữ liệu được lấy vào trong cache để đáp ứng một tham chiếu đơn lẻ, có nhiều khả năng các tham chiếu kế tiếp sẽ là các từ khác trong cùng một khối.

Cache có thể được gắn trực tiếp vào các thiết bị ngoại vi. Các đĩa cứng hiện đại được gắn bộ nhớ truy cập nhanh, khoảng 512KB. Máy tính không trực tiếp sử dụng bộ nhớ này mà là chương trình điều khiển ổ cứng. Đối với máy tính, các chip có bộ nhớ này được coi là đĩa. Khi máy tính yêu cầu dữ liệu ổ cứng, chương trình quản lý đĩa sẽ kiểm tra bộ nhớ đó trước khi di chuyển các bộ phận cơ khí của đĩa cứng (tốc độ rất chậm so với bộ nhớ). Nếu nó tìm thấy dữ liệu mà máy tính yêu cầu trong cache, nó sẽ cung cấp thông tin đó mặc dù không thực sự truy cập vào ổ đĩa, tiết kiệm rất nhiều thời gian.

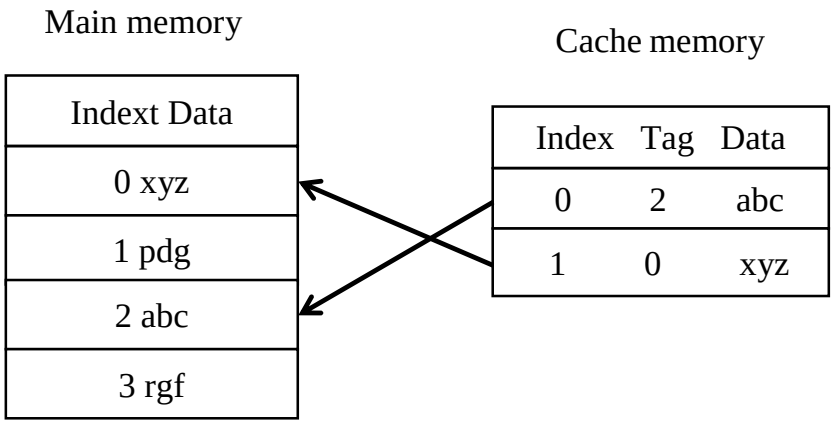
4.3.2.2 Hoạt động của bộ nhớ cache

Cache được coi là bộ nhớ thông minh: cache có khả năng tiên đoán trước yêu cầu về lệnh và dữ liệu của CPU. Dữ liệu và lệnh cần thiết được chuyển trước từ bộ nhớ chính về cache, sau đó CPU truy cập cache. Từ đó giảm thiểu được thời gian truy nhập bộ nhớ.

Cache hoạt động dựa trên hai nguyên lý cơ bản:

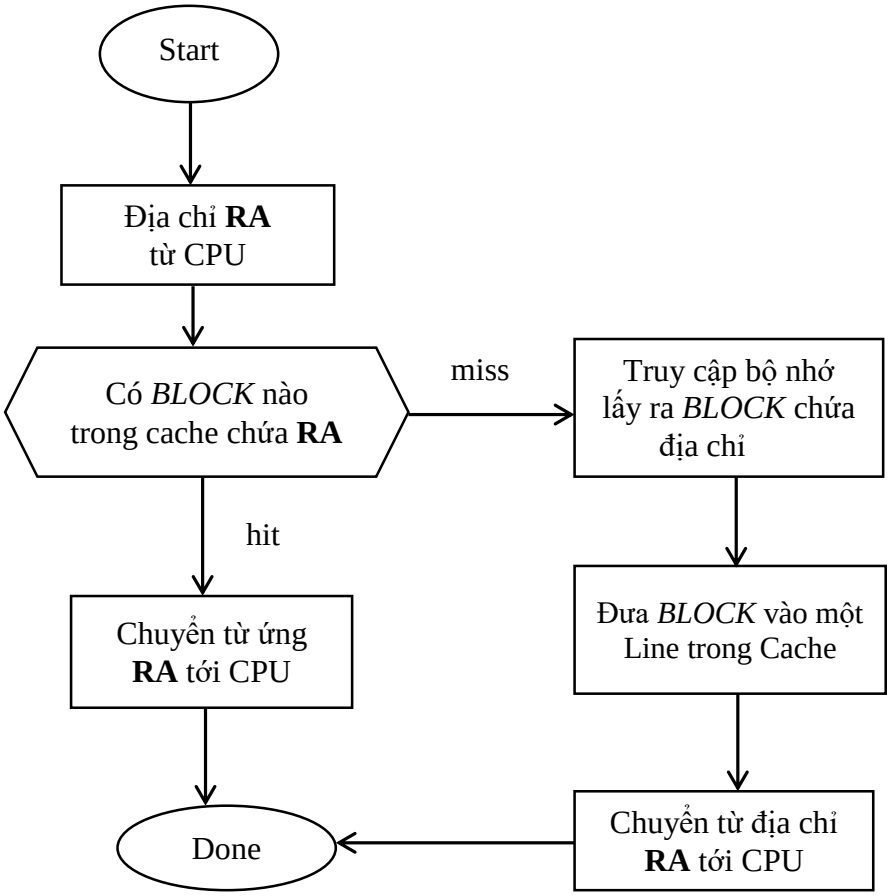
Nguyên lý cục bộ/lân cận về không gian (spatial locatily): Nếu một vị trí bộ nhớ được truy cập, thì khả năng các vị trí gần đó được truy cập trong thời gian gần tới là rất cao. Áp dụng với các mục dữ liệu và các lệnh có thứ tự tuần tự theo chương trình. Hầu hết các lệnh trong chương trình có thứ tự tuần tự, do đó cache đọc một khối lệnh trong bộ nhớ, mà bao gồm các phần tử xung quanh vị trị phần tử hiện tại được truy cập.

Nguyên lý cục bộ/lân cận về thời gian (temporal locatily): Nếu một ị trí bộ nhớ được truy cập, thì khả năng nó sẽ được truy cập trong thời gian gần tới là cao. Áp dụng các mục dữ liệu và các lệnh trong vòng lặp. Cache đọc khối dữ liệu trong bộ nhớ bao gồm tất cả các thành phần trong vòng lặp.



Hình 4.14 Mô hình hoạt động của bộ nhớ Cache

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



Hình 4.15 Sơ đồ thao tác đọc cache

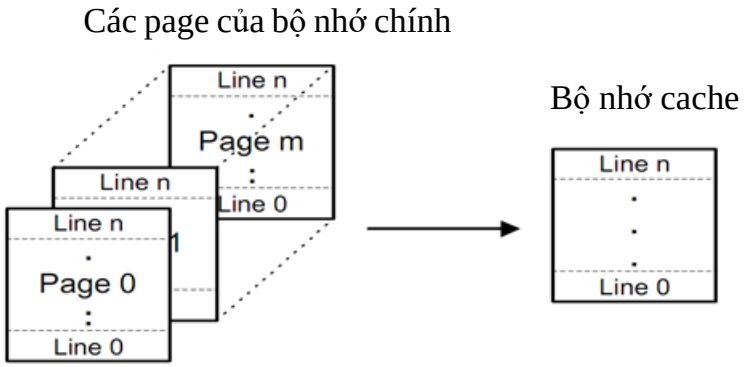
4.3.3 Các phương pháp ánh xạ

Bộ nhớ chính có 2^N byte nhớ. Bộ nhớ chính và bộ nhớ cache được chia thành các khối có kích thước bằng nhau. Bộ nhớ chính: B0, B1, B2,…, Bp-1 (p Blocks). Bộ nhớ cache: L0, L1, L2,…, Lm-1 (m Lines). Kích thước của Block hoặc Line = 8, 16, 32, 64, 128 byte. Mỗi Line trong cache có một thẻ nhớ (Tag) được gắn vào. Một số Block của bộ nhớ chính được nạp vào các Line của cache. Nội dung Tag (thẻ nhớ) cho biết Block nào của bộ nhớ chính hiện đang được chứa ở Line đó. Nội dung Tag được cập nhật mỗi khi Block từ bộ nhớ chính nạp vào Line đó

Chức năng ánh xạ: Là ánh xạ các khối dữ liệu của bộ nhớ chính vào các vị trí của bộ nhớ cache. Có ba kỹ thuật ánh xạ chính:

- Ánh xạ trực tiếp.
- Ánh xạ liên kết toàn phần.
- Ánh xạ liên kết tập hợp.

4.3.3.1 Ánh xạ trực tiếp (Direct Mapping).



Hình 4.16 Cấu tạo cache ánh xạ trực tiếp

Đây là kiểu ánh xạ đơn giản nhất cho phép từng khối dữ liệu (Block) của bộ nhớ chính chỉ được ghi (chỉ được nạp) ở một rãnh nhất định (vào một Line của Cache). Tổng quát: B_j chỉ có thể nạp vào Lj mod m trong đó m là số Line của cache.

Địa chỉ N bit của bộ nhớ chính chia thành ba trường:

- Trường Word gồm W bit xác định một từ nhớ trong Block hay Line: 2W = kích thước của Block hay Line.
- Trường Line gồm L bit xác định một trong số các Line trong cache: 2L = số Line trong cache = m.
- Trường Tag gồm T bit: T = N - (W+L)
- Mỗi thẻ nhớ (Tag) của một Line chứa được T bit
- Khi Block từ bộ nhớ chính được nạp vào Line của cache thì Tag ở đó được cập nhật giá trị là T bit địa chỉ cao của Block đó.
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể. Nhờ vào giá trị L bit của trường Line sẽ tìm ra Line tương ứng. Đọc nội dung Tag ở

Line đó (T bit), rồi so sánh với T bit bên trái của địa chỉ vừa phát ra. Giống nhau: cache hit. Khác nhau: cache miss

Ví dụ: dữ liệu của một khối 4 byte. Tổ chức ánh xạ từ bộ nhớ chính 16MB vào Bộ nhớ cache 64 KB.

Gọi C là số rãnh trong bộ nhớ cache.

S là số hiệu rãnh.

A là sô hiệu khối của bộ nhớ chính.

Công thức ánh xạ: S = A modulo C

Đây là kiểu ánh xạ đơn giản nhất cho phép từng khối dữ liệu của bộ nhớ chính chỉ được ghi ở một rãnh nhất định.

Số rãnh trong bộ nhớ cache C = 64 KB : 4 Byte = 16 KB = 2^4. 2^{10} = 2^{14} Byte.

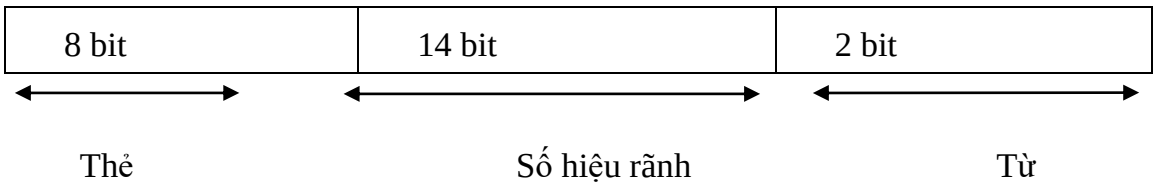
Trong bộ nhớ chính có 16 MB =2^{24} byte => Số khối = $\frac{2^{24}}{2^2}$ =2^{22} có 22 khối.

Có 22 khối =>Số hiệu khối A là 2^{22} - 1(Đánh từ 0)=> Số hiệu rãnh S =2^{22} - 1 modulo 2^{14}

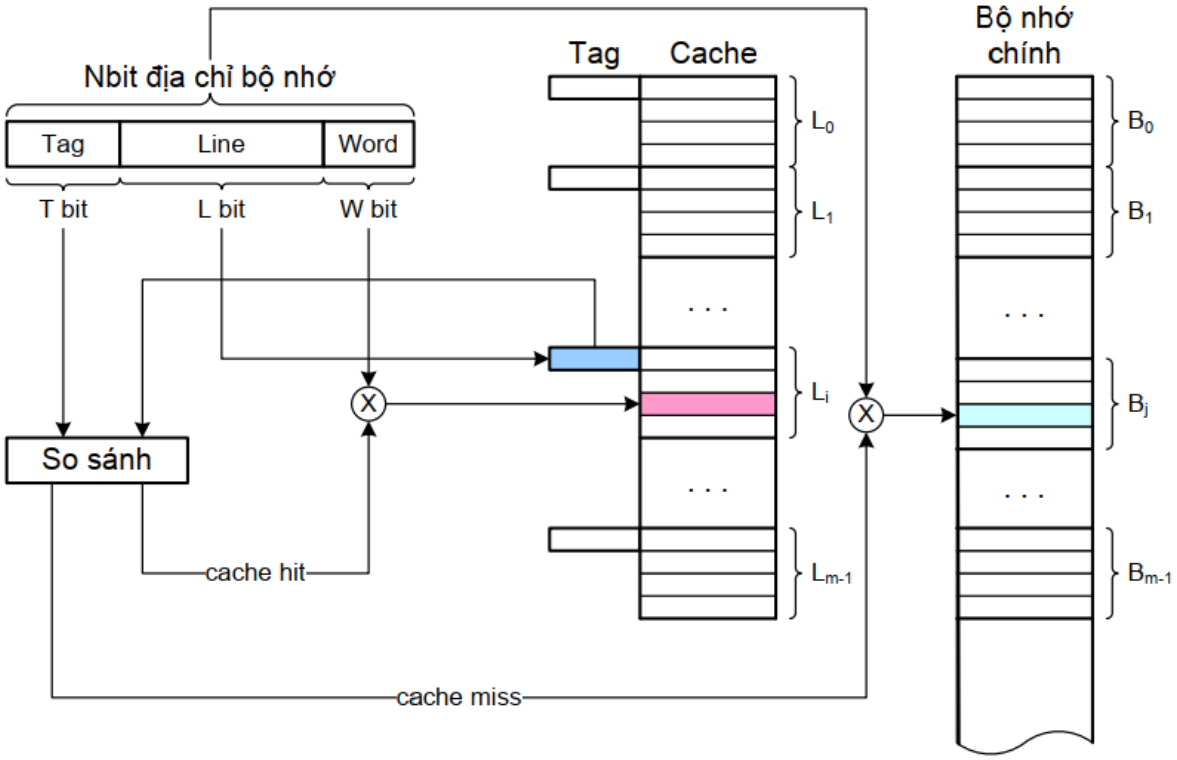
24 bit địa chỉ được phân chia thành:

- 2 bit có trọng số nhỏ nhất để xác định một byte duy nhất trong một khối của bộ nhớ.
- 22 bit còn lại để xác định một khối trong 2^{22} khối của bộ nhớ chính. Cache sẽ dịch 22 bit đó thành một số thẻ 8 bit và số hiệu rãnh là 14 bit. Số hiệu rãnh định nghĩa một rãnh duy nhất trong cache. Nó chính bằng số hiệu khối của bộ nhớ chính chia modulo cho 2^{14}.

Cụ thể 24 bit địa chỉ trong bộ nhớ được phân bổ như sau:



Từ đó xác định sự ánh xạ của các khối trong bộ nhớ chính vào các rãnh của bộ nhớ cache.



Hình 4.17 Phương pháp ánh xạ trực tiếp

Ưu điểm: Giá thành rẻ.

Nhược điểm: Lãng phí bộ nhớ, tỷ lệ hit ratio thấp. Sự cố định các khối trong các line của Cache. Trong trường hợp chương trình muốn truy xuất tới 2 Block liên tục mà 2 Block được phân nằm trong cùng line thì khả năng Cache miss rất cao.

Ví dụ: Giả sử máy tính đánh địa chỉ cho từng byte. Không gian địa chỉ bộ nhớ chính = 4GB Dung lượng bộ nhớ *cache* là 256KB. Kích thước *Line* (*Block*) = 32byte. Xác định số bit của các trường địa chỉ cho phương pháp ánh xạ trực tiếp?

Bài giải:

Bộ nhớ chính = 4GB = 2^{32} byte → N = 32 bit

Cache = 256 KB = 2^{18} byte.

Line = 32 byte = 2^5 byte → W = 5 bit

Số Line trong cache = $2^{18}/ 2^5 = 2^{13}$ Line → L = 13 bit

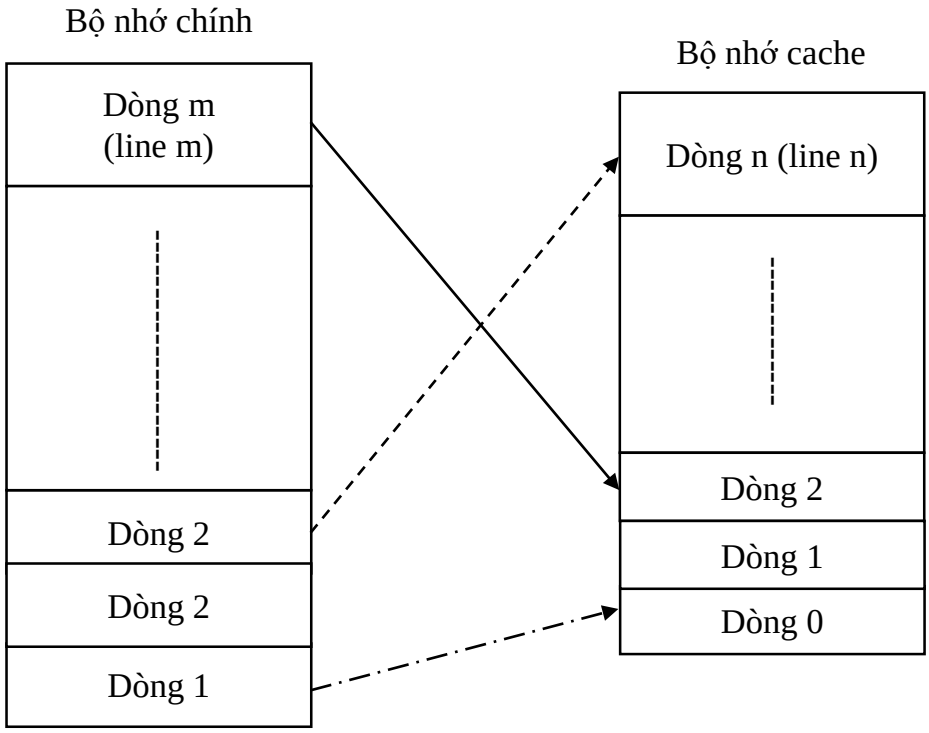
T = 32 - (13 + 5) = 14 bit

Tag	Line	Word
14 bit	13 bit	5 bit

Hình 4.18 Các trường địa chỉ theo phương pháp ánh xạ trực tiếp

4.3.3.2 Ánh xạ liên kết toàn phần (Full Associative Mapping)

Kỹ thuật ánh xạ liên kết hoàn toàn cho phép một khối dữ liệu của bộ nhớ chính có thể được ghi vào bất kỳ rãnh nào của bộ nhớ cache.



Hình 4.19 Cấu tạo cache ánh xạ liên kết toàn phần

Cách sắp xếp này cho phép mỗi line trong bộ nhớ chính có thể được lưu đến bất kỳ vị trí nào trong bộ nhớ Cache. Cấu trúc này không có Cache page mà chỉ có Cache line. Cả bộ nhớ chính và bộ nhớ cache được chia thành các line có kích thước nhau. Trong hình trên, line 1 của bộ nhớ chính được lưu đến line 0 của bộ nhớ cache nhưng không phải lúc nào cũng vậy. Nghĩa là line 1 có thể được lưu ở bất kỳ cache line nào. Trong phương pháp ánh xạ này, một cache line có thẻ lưu bất kỳ line bộ nhớ nào hay một line bộ nhớ có thể lưu bất kỳ đâu trong cache.

Cách kết hợp này cho hiệu suất tốt nhất nhưng cấu trúc thực hiện lại phức tạp. Sự phức tạp này là do việc xác định dữ liệu yêu cầu nằm trong cache. Để đáp ứng các yêu cầu tại một thời điểm, địa chỉ hiện tại phải được so sánh với tất cả các địa chỉ có mặt trong TRAM. Điều này yêu cầu một số lượng rất lớn bộ so sánh, từ đó làm tăng độ phức tạp và giá thành, khi cache làm ra có kích thước lớn. Chính vì vậy, loại cache này chỉ dùng cho các cache dung lượng nhỏ hơn 4KB.

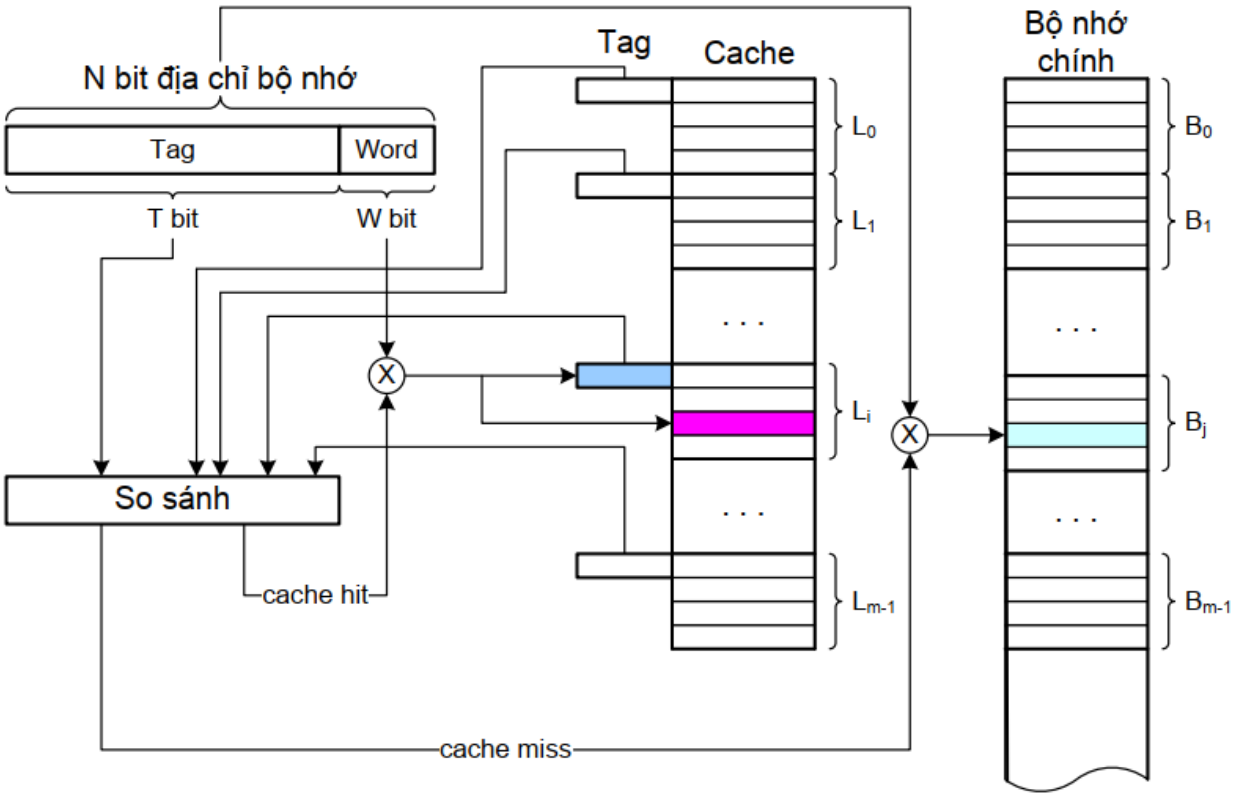
Một Block của bộ nhớ chính có thể nhập vào bất kỳ line nào trong cache.

Địa chỉ CPU phát ra được chia thành hai địa chỉ: Tag và Word

- Địa chỉ Tag xác định khối duy nhất của bộ nhớ nằm trong cache. Mỗi giá trị Tag của line là khác nhau.
- Trường Word để xác định chính xác từ cần truy xuất.

Chi phí phương pháp này đối với cache là cao.

- Mỗi thẻ nhớ (Tag) của một Line chứa được T bit.
- Khi Block từ bộ nhớ chính được nạp vào Line của cache thì Tag ở đó được cập nhật giá trị là T bit địa chỉ cao của Block đó.
- Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể.
- So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong cache. Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line đó. Nếu không có giá trị nào bằng: cache miss



Hình 4.20 Phương pháp ánh xạ liên kết toàn phần

Ưu điểm : Linh hoạt, tận dụng tối ưu dung lượng bộ nhớ kép. Xác suất cache hit cao.

Nhược điểm: Mạch thực hiện phức tạp. Phải so sánh với tất cả các Tag nên mất nhiều thời gian.

Ví dụ: Giả sử máy tính đánh địa chỉ cho từng byte. Không gian địa chỉ bộ nhớ chính bằng 4GB. Dung lượng bộ nhớ *cache* là 256KB. Kích thước *Line* (*Block*) = 32byte. Xác định số bit của các trường địa chỉ cho phương pháp ánh xạ liên kết toàn phần?

Bài giải:

Bộ nhớ chính = 4GB = 2^{32} byte → N = 32 bit

Line = 32 byte = 2^5 byte → W = 5 bit

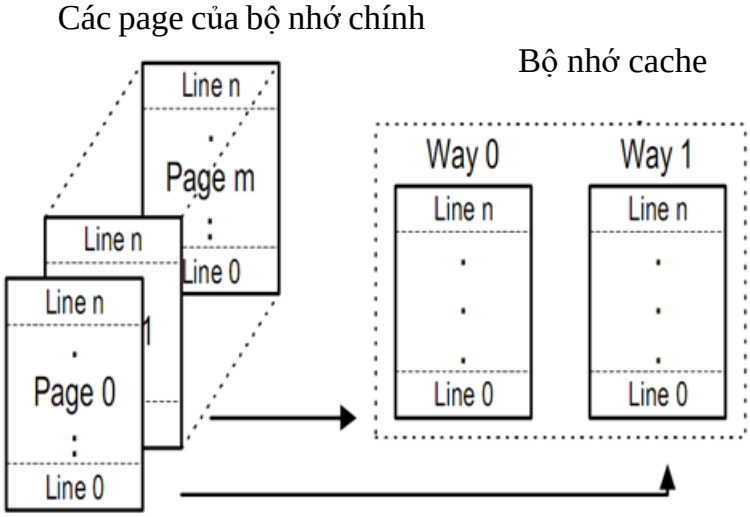
Số bit của trường Tag sẽ là: T = 32 - 5 = 27 bit

Tag	Word
27 bit	5 bit

Hình 4.21 Các trường địa chỉ theo phương pháp ánh xạ liên kết toàn phần

4.3.3.3 Ánh xạ liên kết tập hợp:

Kết hợp của hai phương pháp: Ánh xạ trực tiếp và ánh xạ ánh xạ liên kết hoàn toàn (Set Associative Mapping)



Hình 4.22 Cấu tạo cache ánh xạ liên kết tập hợp

Phương pháp ánh xạ liên kết tập hợp là sự tổ hợp của cách sắp xếp ánh xạ liên kết và ánh xạ trực tiếp. một cấu trúc ánh xạ liên kết tập hợp chia SRAM Cache (Cache có cấu tạo là SRAM) thành các đoạn bằng nhau, gọi là Cache way (Đặc trưng có cache 2-way và cache 4-way). Kích thước Cache page bằng với kích thước một cache way. Mỗi cache way được xem như một ánh xạ trực tiếp nhỏ.

Trong cấu trúc này, hai line của bộ nhớ có thể được lưu bất cứ lúc nào. Điều này giúp giảm thiểu số lần dữ liệu cache line được ghi đến. Cấu trúc này rõ ràng là ít phức tạp hơn cấu trúc ánh xạ liên kết vì số bộ so sánh bằng với số cache way. Một cache thiết lập liên kết 2-way chỉ yêu cầu hai bộ so sánh như vậy làm giảm rất nhiều giá thành khi thực hiện cấu trúc này so với cấu trúc ánh xạ liên kết.

Kích thước *Block* = 2W Word

Trường *Set* có S bit dùng để xác định một trong số các Set trong cache. 2S = Số *Set trong cache*

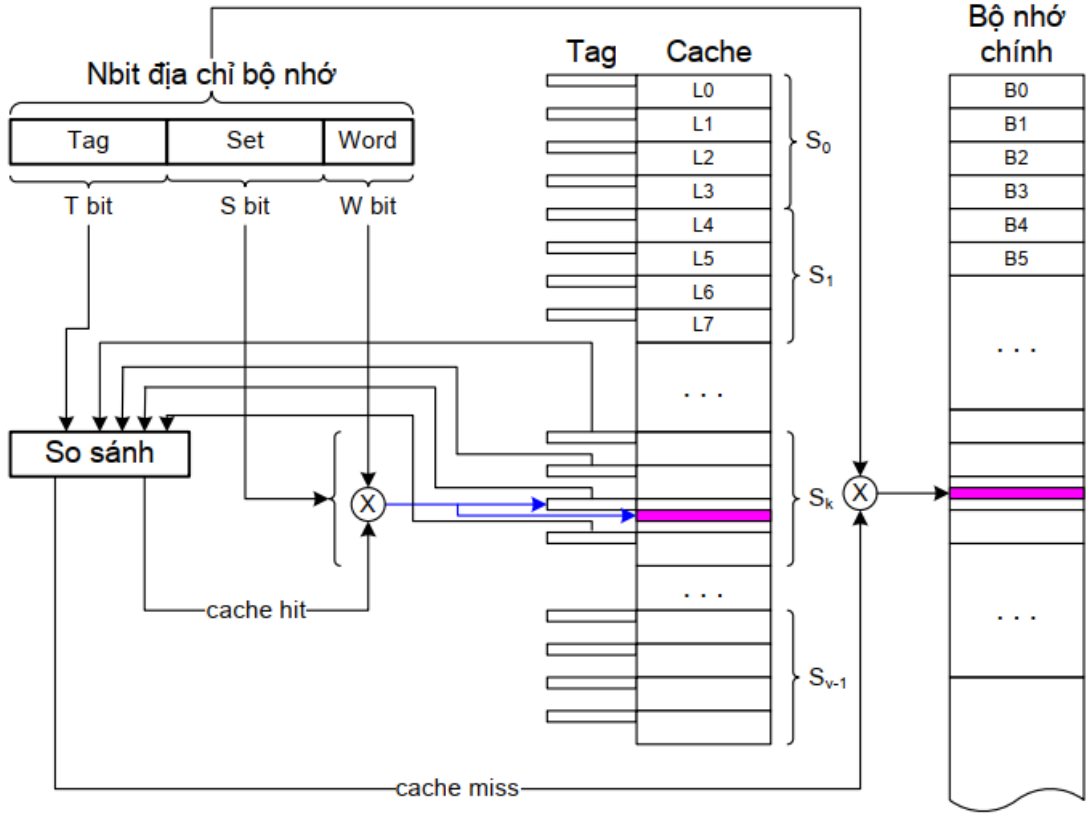
Trường *Tag* có T bit: T = N - (W+S)

Khi CPU muốn truy nhập một từ nhớ thì nó phát ra một địa chỉ N bit cụ thể

- Nhờ vào giá trị S bit của trường Set sẽ tìm ra Set tương ứng

- So sánh T bit bên trái của địa chỉ vừa phát ra với lần lượt nội dung của các Tag trong Set đó. Nếu gặp giá trị bằng nhau: cache hit xảy ra ở Line tương ứng. Nếu không có giá trị nào bằng: cache miss.

Tổng quát cho cả hai phương pháp trên. Thông dụng với: 2, 4, 8, 16Lines/Set.



Hình 4.23 Phương pháp ánh xạ liên kết tập hợp

Ví dụ: Giả sử máy tính đánh địa chỉ cho từng byte. Không gian địa chỉ bộ nhớ chính = 4GB Dung lượng bộ nhớ *cache* là 256KB. Kích thước *Line* (*Block*) = 32byte. Xác định số bit của các trường địa chỉ cho phương pháp ánh xạ liên kết tập hợp 4 đường?

Bài giải:

Bộ nhớ chính = 4GB = 2^{32} byte → N = 32 bit

Line = 32 byte = 2^5 byte → W = 5 bit

Số Line trong cache = $2^{18}/ 2^5 = 2^{13}$ Line

Một Set có 4 Line = 2^2 Line → số Set trong cache = $2^{13}/2^2 = 2^{11}$ Set → S = 11 bit

Số bit của trường Tag: T = 32 - (11 + 5) = 16 bit

Tag	Set	Word
16 bit	11 bit	5 bit

Hình 4.24 Các trường địa chỉ theo phương pháp ánh xạ liên kết tập hợp 4 đường

4.3.4 Các phương pháp thay thế dữ liệu

Có 4 phương pháp

- LFU (least frequency used).
- LRU (least recently used)
- FIFO(first in first out)
- Random (Ngẫu nhiên).

Phương pháp thay thế ngẫu nhiên: Không có quy luật gì cả.

Thay thế ngẫu nhiên: Các dòng trong cache được chọn ngẫu nhiên để thay thế, do vậy phương pháp này đơn giản, tỷ lệ miss cao vì phương pháp này không xét tới dòng cache nào đang thực sự được sử dụng. Nếu một dòng cache đang được sử dụng mà bị thay thế thì sự kiện miss xảy ra và cần phải đọc lại vào cache.

FIFO (first in first out) - Phương pháp thay thế dữ liệu ở rãnh đầu tiên trong cache.

Dựa trên nguyên lý FIFO: Các dòng cache được đọc vào cache trước sẽ được chọn để thay trước. Tỷ lệ miss thấp hơn so với phương pháp ngẫu nhiên, tuy vậy tỷ lệ miss vẫn cao vì phương pháp này vẫn chưa thực sự xem xét tới block nào đang thực sự được sử dụng. Cài đặt phức tạp vì cần thêm mạch để giám sát thứ tự nạp các dòng bộ nhớ vào cache.

Ứng với một rãnh là một bộ đếm. Mỗi khi dữ liệu được nạp vào rãnh thì giá trị bộ đếm của rãnh đó được reset = 0, các rãnh khác tăng lên một.

Mỗi lần truy cập dữ liệu thì giá trị bộ đếm của các rãnh khác tăng lên một.

Khi cần thay thế tìm giá trị bộ đếm lớn nhất để thay thế → reset vị trí đó bằng không, các giá trị khác tăng lên một.

LRU (least recently used) - Phương pháp thay thế dữ liệu ở các rãnh ít được sử dụng gần đây nhất. Các dòng cache ít được sử dụng nhất trong thời gian gần đây sẽ được chọn để thay. Xét tới các dòng đang được sử dụng. Tỷ lệ miss thấp nhất so với phương pháp ngẫu nhiên và FIFO. Cài đặt phức tạp vì cần thêm mạch để giám sát tần suất sử dụng các dòng cache.

Ứng với một rãnh là một bộ đếm. Mỗi khi dữ liệu được nạp vào rãnh thì giá trị bộ đếm của rãnh đó được reset =0, các rãnh khác tăng lên 1. Mỗi lần truy cập dữ liệu thì giá trị của bộ đếm dữ liệu được reset = 0, còn bộ đếm của các rãnh khác tăng lên một.

Khi cần thay thế , tìm giá trị bộ đếm lớn nhất để thay thế.Reset = 0, Các rãnh khác tăng lên một.

LFU (least frequency used) - Phương pháp thay thế ở các rãnh ít được sử dụng nhất. Ứng với một rãnh là một bộ đếm. Mỗi khi dữ liệu được nạp vào rãnh thì giá trị bộ đếm của rãnh đó được reset =0, rãnh khác giữ nguyên.Mỗi lần truy cập dữ liệu trong bộ nhớ cache, nếu dữ liệu đã tồn tại trong cache thì giá trị bộ đếm của rãnh chứa dữ liệu được tăng lên 1, còn bộ đếm của các rãnh khác giữ nguyên.

Khi cần thay thế dữ liệu hệ thống sẽ tìm rãnh có giá trị bộ đếm nhỏ nhất để thay thế. [1][2]

Ví dụ: Cho dãy sử dụng lần lượt là:

1 2 5 1 3 5 1 4

LFU

3	3
2	5
1	4
0	1

LRU

3	3
2	5
1	2
0	1

FIFO

3	3
2	5
1	2
0	1

Giá trị biến đếm : Rãnh 0: 0, 0 , 0, 1, 1, 1, 0, 1, 2, 0, 1, 2, 0, 1, 2, 3, 4, 5.

Rãnh 1: - , 0, 0, 0, 0 , 0 - , 0, 1, 2, 3, 4 - , 0, 1, 2, 3, 4

Rãnh 2 : -, - , 0, 0, 0, 1, 0 -, - ,0, 1, 2, 0, -, -, 0, 1, 2, 3.

Rãnh 3: - , -, -, -, 0 , 0, -, -, -, - , 0, 1, -, -, -, - , 0, 1.

Cache hit, mit: M, M, M, H, M, H, M, M, M, H, M, H M, M, M, M, H, M, H.

Rãnh 0: 2.

Rãnh 1: 0.

Rãnh 2: 1.

Rãnh 3 0 (H).

4.4. Bộ nhớ ảo và kỹ thuật phân trang

4.4.1 Sự cần thiết của bộ nhớ ảo

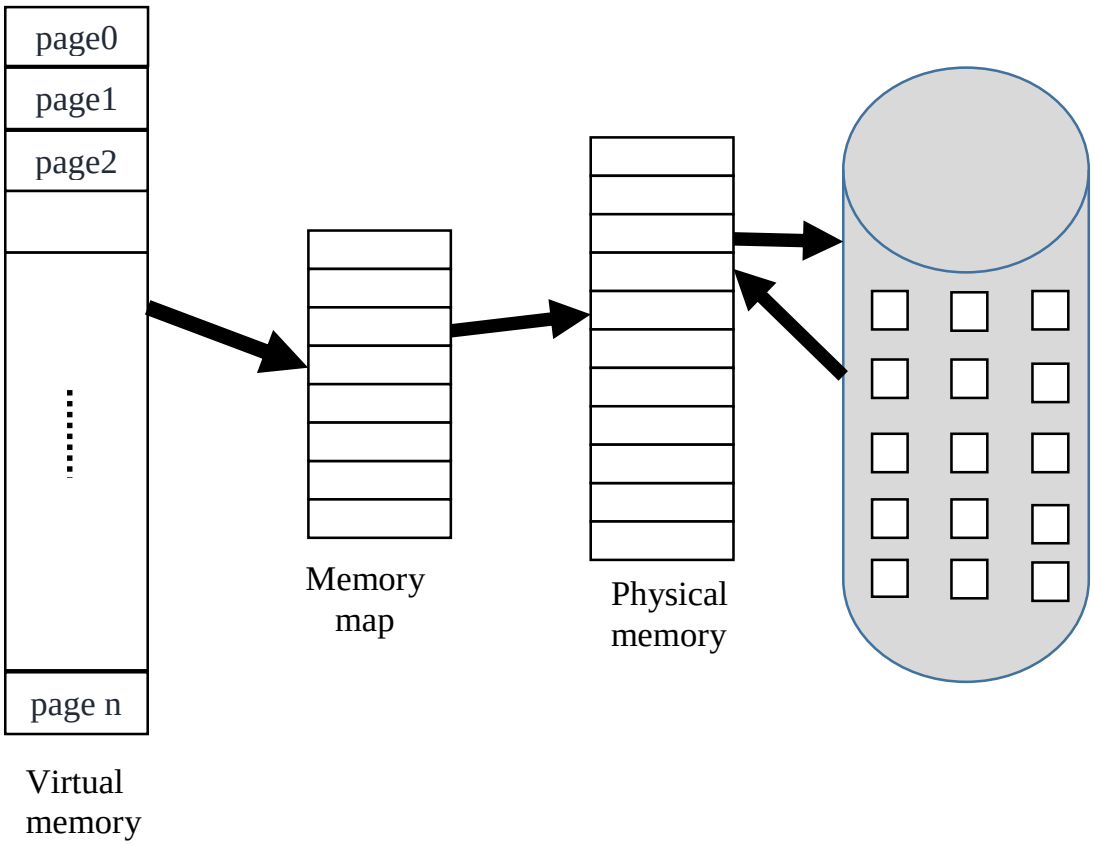
Bộ nhớ ảo là một kỹ thuật cho phép việc thực thi của quá trình mà quá trình có thể không hoàn toàn ở trong bộ nhớ. Một lợi điểm quan trọng của cơ chế này là các chương trình có thể lớn hơn bộ nhớ vật lý. Ngoài ra, bộ nhớ ảo phóng đại bộ nhớ chính thành bộ nhớ luận lý cực lớn khi được hiển thị bởi người dùng. Kỹ thuật này giải phóng người lập trình từ việc quan tâm đến giới hạn kích thước bộ nhớ. Bộ nhớ ảo cũng cho phép các quá trình dễ dàng chia sẻ tập tin và không gian địa chỉ, cung cấp cơ chế hữu hiện cho việc tạo quá trình.

Khi xem xét các chương trình thực thi chúng ta nhận thấy rằng trong nhiều trường hợp toàn bộ chương trình là không cần thiết. Khả năng thực thi chương trình khi chỉ một phần chương trình ở trong bộ nhớ có nhiều lợi điểm:

- Chương trình sẽ không còn bị ràng buộc bởi không gian bộ nhớ vật lý sẵn có. Người dùng có thể viết chương trình có không gian địa chỉ ảo rất lớn, đơn giản hoá tác vụ lập trình.
- Vì mỗi chương trình người dùng có thể lấy ít hơn bộ nhớ vật lý nên nhiều chương trình hơn có thể được thực thi tại một thời điểm. Điều này giúp gia tăng việc sử dụng CPU và thông lượng nhưng không tăng thời gian đáp ứng.

- Yêu cầu ít nhập/xuất hơn để nạp hay hoán vị mỗi chương trình người dùng trong bộ nhớ vì thế mỗi chương trình người dùng sẽ chạy nhanh hơn.

Do đó, chạy một chương trình mà nó không nằm hoàn toàn trong bộ nhớ có lợi cho cả người dùng và hệ thống. Bộ nhớ ảo là sự tách biệt bộ nhớ luận lý từ bộ nhớ vật lý. Việc tách biệt này cho phép bộ nhớ ảo rất lớn được cung cấp cho người lập trình khi chỉ bộ nhớ vật lý nhỏ hơn.



Hình 4.25 Lưu đồ minh họa bộ nhớ ảo lớn hơn bộ nhớ vật lý

Thêm vào đó, việc tách biệt bộ nhớ luận lý từ bộ nhớ vật lý, bộ nhớ ảo cũng cho phép các tập tin và bộ nhớ được chia sẻ bởi những quá trình khác nhau thông qua việc chia sẻ trang. Bộ nhớ ảo thường được cài đặt bởi phân trang theo yêu cầu (demand paging). Nó cũng có thể được cài đặt trong cơ chế phân đoạn. Tuy nhiên, các giải thuật thay thế đoạn phức tạp hơn các giải thuật thay thế trang vì các đoạn có kích thước thay đổi.

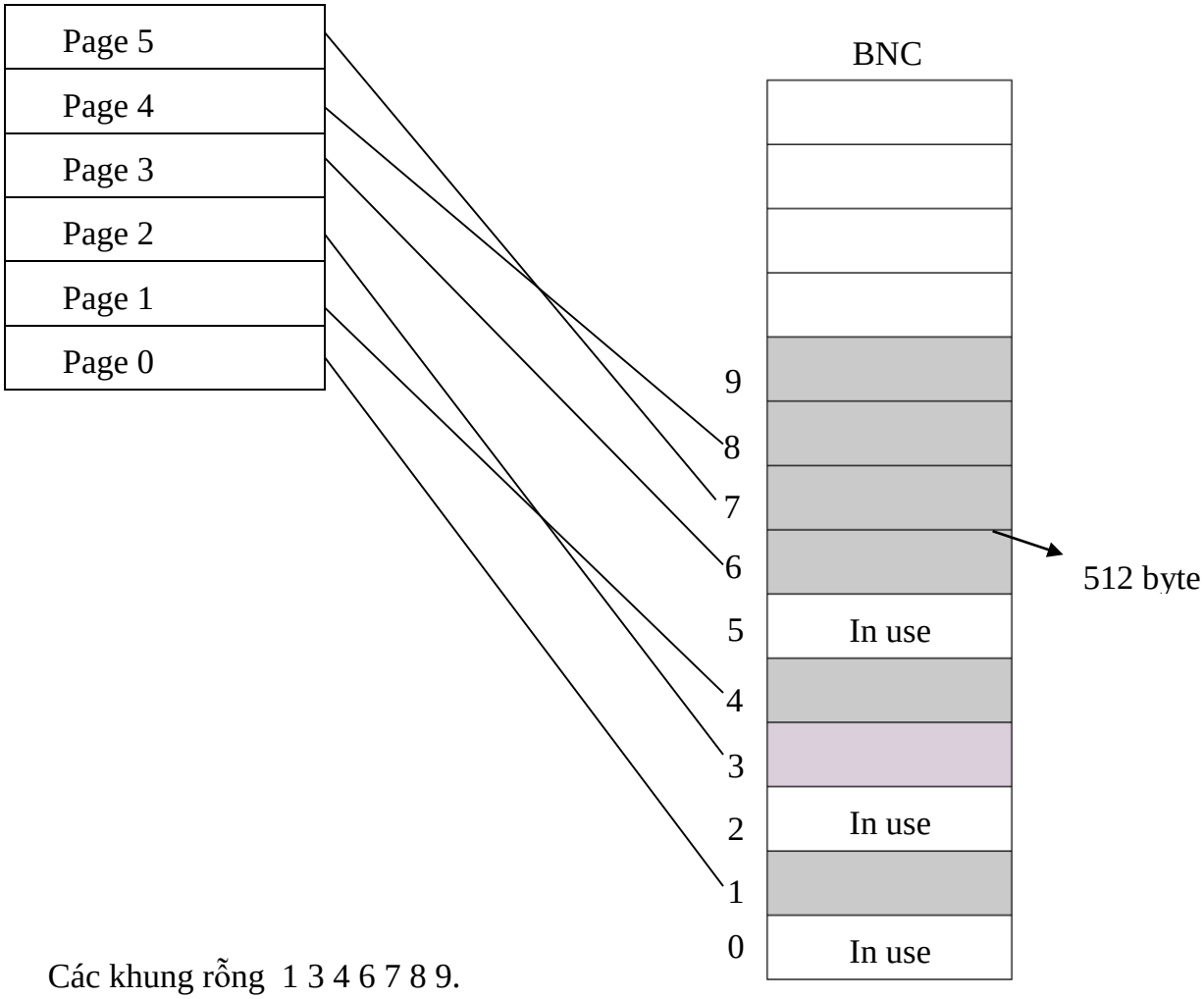
4.4.2 Bộ nhớ ảo và kỹ thuật phân trang

Với kỹ thuật này, một chương trình hay một tiến trình được chia thành nhiều khối nhỏ có kích thước như nhau gọi là các trang, các trang này được lưu trữ ở bộ nhớ phụ (thông thường là đĩa cứng). Khi chương trình (tiến trình) được thực hiện thì hệ điều hành sẽ nạp các trang của chương trình vào các đoạn có kích thước tương ứng của bộ nhớ chính.

Mỗi đoạn nhỏ chứa trang được gọi là khung trang (hay còn gọi là khung - frame). Tại một thời điểm thì trong bộ nhớ chính có một số khung đã được sử dụng, một số khung rỗng. Danh sách các khung rỗng được quản lý bởi Hệ điều hành, các trang của một tiến trình mới khi thực hiện sẽ được nạp vào các khung rỗng này, ứng với mỗi một chương trình

hoặc một tiến trình thì hệ điều hành sẽ duy trì một bảng phân trang cho nó. Kích thước trang = 512 Byte.

Ví dụ: Tiến trình A được chia thành 6 trang, được lưu trên đĩa cứng.



Các khung rỗng 1 3 4 6 7 8 9.

Ký hiệu trang	Trạng thái	Quy trình nạp	Địa chỉ của khung trang sẽ nạp
0	1	R/W	1
1	1	R/W	4
2	1	R/W	3
3	1	R/W	6
4	1	R/W	8
5	1	R/W	7

Một địa chỉ ảo gồm có hai trường.

- Trường số hiệu trang.
- Trường độ dịch chuyển.

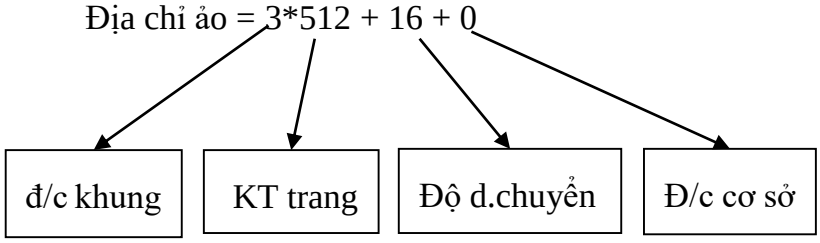
Số hiệu trang	Độ dịch chuyển

Ví dụ:

2	16

Số hiệu trang là 2→ Địa chỉ khung là 3

Địa chỉ ảo = 3*512 + 16 + 0



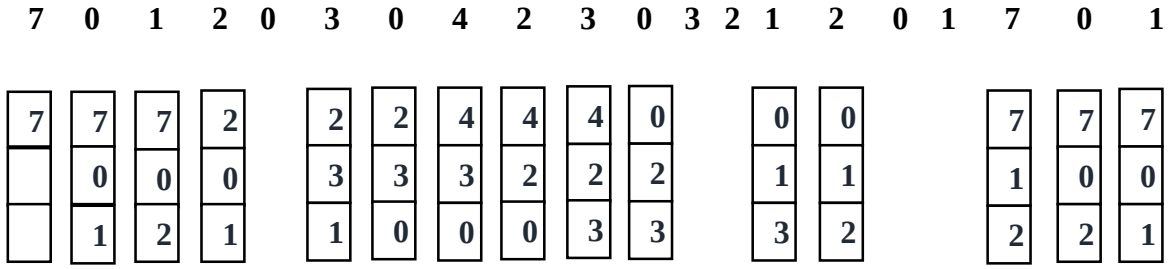
4.4.3 Các phương pháp thay thế khung trang

4.4.3.1 Thay thế trang FIFO

Giải thuật thay thế trang đơn giản nhất là giải thuật FIFO. Giải thuật này gắn với mỗi trang thời gian khi trang đó được mang vào trong bộ nhớ. Khi một trang phải được thay thế, trang cũ nhất sẽ được chọn.

Chú ý rằng, nó không yêu cầu nghiêm ngặt để ghi thời gian khi trang được mang vào. Chúng ta có thể tạo một hàng đợi FIFO để quản lý tất cả trang trong bộ nhớ, ta thay thế trang tại đầu hàng đợi. Khi trang được mang vào bộ nhớ, chúng ta chèn nó vào đuôi của hàng đợi.

Cho một thí dụ về chuỗi tham khảo, ba khung trang ban đầu là rỗng. Ba tham khảo đầu tiên (7, 0, 1) gây ra lỗi trang và được mang vào các khung rỗng này. Tham khảo tiếp theo (2) thay thế trang 7, vì trang 7 được mang vào trước. Vì 0 là tham khảo tiếp theo và 0 đã ở trong bộ nhớ rồi, chúng ta không có lỗi trang cho tham khảo này. Tham khảo đầu tiên tới 3 dẫn đến trang 0 đang được thay thế vì thế nó là trang đầu tiên của ba trang trong bộ nhớ (0, 1, 2) để được mang vào. Bởi vì thay thế này, tham khảo tiếp theo, tới 0, sẽ bị lỗi. Sau đó, trang 1 được thay thế bởi trang 0. Mỗi khi một lỗi xảy ra, chúng ta hiển thị các trang ở trong 3 khung của chúng ta. Có 15 lỗi tất cả.



Hình 4.26 Giải thuật thay thế trang FIFO

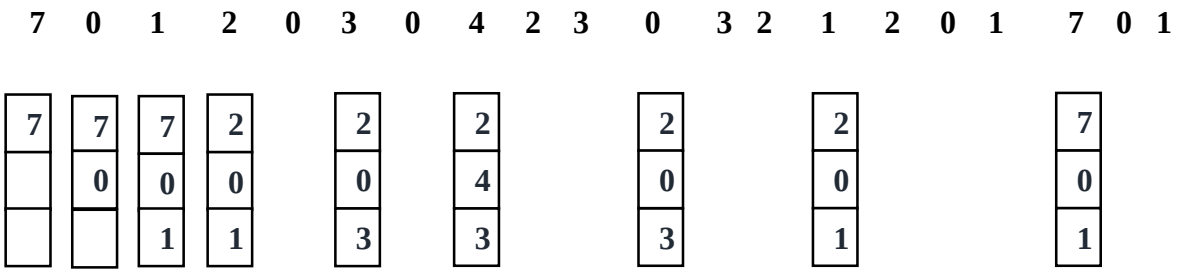
Giải thuật thay thế trang FIFO rất dễ hiểu và lập trình. Tuy nhiên, năng lực của nó không luôn tốt. Trang được cho để thay thế có thể là trang chức nhiều dữ liệu cần thiết, thường xuyên được sử dụng nên được nạp sớm, do vậy khi chuyển ra bộ nhớ phụ sẽ nhanh chóng gây ra lỗi trang.

4.4.3.2 Thay thế trang tối ưu hoá

Giải thuật thay thế trang tối ưu có tỉ lệ lỗi trang thấp nhất trong tất cả các giải thuật Giải thuật như thế tồn tại và được gọi là OPT hay MIN. Nó đơn giản là: thay thế trang mà nó không được dùng cho một khoảng thời gian lâu nhất. Sử dụng giải thuật thay thế trang đảm bảo tỉ lệ lỗi trang nhỏ nhất có thể cho một số lượng khung cố định.

Ví dụ, trên một chuỗi tham khảo mẫu, giải thuật thay thế trang tối ưu sẽ phát sinh 9 lỗi trang, như được hiển thị trong hình bên dưới. Tham khảo đầu tiên gây ra lỗi điền vào 3 khung trống. Tham khảo tới trang 2 thay thế trang 7 vì 7 sẽ không được dùng cho tới khi tham khảo 18, trái lại trang 0 sẽ được dùng tại 5 và trang 1 tại 14. Tham khảo tới trang 3 thay thế trang 1 khi trang 1 sẽ là trang cuối cùng của 3 trang trong bộ nhớ được tham khảo lần nữa. Với chỉ 9 lỗi trang, thay thế tối ưu là tốt hơn nhiều giải thuật FIFO, có 15 lỗi. (Nếu chúng ta bỏ qua 3 lỗi đầu mà tất cả giải thuật phải gặp thì thay thế tối ưu tốt gấp 2 lần thay thế FIFO.) Thật vậy, không có giải thuật thay thế nào có thể xử lý chuỗi tham khảo trong 3 khung với ít hơn 9 lỗi.

Tuy nhiên, giải thuật thay thế trang tối ưu là khó cài đặt vì nó yêu cầu kiến thức tương lai về chuỗi tham khảo. Do đó, giải thuật tối ưu được dùng chủ yếu cho nghiên cứu so sánh. Thí dụ, nó có thể có ích để biết rằng, mặc dù một giải thuật không tối ưu nhưng nó nằm trong 12.3% của tối ưu là tệ, và trong 4.7% là trung bình.



Hình 4.27 Giải thuật thay thế trang tối ưu hóa

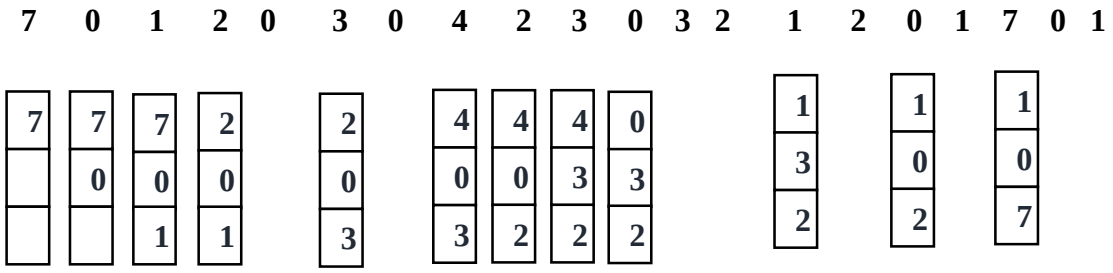
4.4.3.3 Thay thế trang LRU

Nếu giải thuật tối ưu là không khả thi, có lẽ một xấp xỉ giải thuật tối ưu là có thể. Sự khác biệt chủ yếu giữa giải thuật FIFO và OPT là FIFO dùng thời gian khi trang được mang vào bộ nhớ; giải thuật OPT dùng thời gian khi trang được sử dụng. Nếu chúng ta sẽ dụng quá khứ gần đây như một xấp xỉ của tương lai gần thì chúng ta sẽ thay thế trang mà nó không được dùng cho khoảng thời gian lâu nhất như hình bên dưới. Tiếp cận này là giải thuật ít được dùng gần đây nhất (least-recently-used (LRU)).

Thay thế trang LRU gắn với mỗi trang thời gian sử dụng cuối cùng của trang. Khi một trang phải được thay thế, LRU chọn trang không được dùng trong một khoảng thời gian lâu nhất. Chiến lược này là giải thuật thay thế trang tối ưu tìm kiếm lùi theo thời gian hơn là hướng tới. (gọi SR là trình tự ngược của chuỗi tham khảo S thì tỉ lệ lỗi trang cho giải thuật OPT trên S là tương tự như tỉ lệ lỗi trang cho giải thuật OPT trên SR. Tương tự,

tỉ lệ lỗi trang đối với giải thuật LRU trên S là giống như tỉ lệ lỗi trang cho giải thuật LRU trên SR)

Kết quả ứng dụng thay thế LRU đối với chuỗi tham khảo điển hình được hiển thị trong hình sau. Giải thuật LRU sinh ra 12 lỗi. 5 lỗi đầu tiên là giống như thay thế tối ưu. Tuy nhiên, khi tham chiếu tới trang 4 xảy ra thay thế LRU thấy rằng 3 khung trong bộ nhớ, trang 2 được dùng gần đây nhất. Trang được dùng gần đây nhất là trang 0, và chỉ trước khi trang 3 được dùng. Do đó, giải thuật LRU thay thế trang 2, không biết rằng trang 2 để được dùng. Sau đó, khi nó gây lỗi trang 2, giải thuật LRU thay thế trang 3, của 3 trang trong bộ nhớ {0, 3, 4} trang 3 ít được sử dụng gần đây nhất. Mặc dù vấn đề này nhưng thay thế LRU với 12 lỗi vẫn tốt hơn thay thế FIFO với 15.

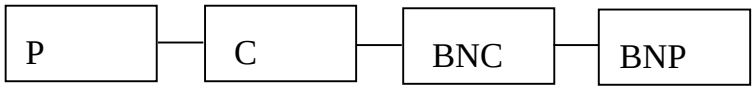


Hình 4.28 Giải thuật thay thế trang LRU

Chính sách LRU thường được dùng như giải thuật thay thế trang và được xem là tốt.

Các Ví dụ minh họa

Ký hiệu S truy cập thành công, F : Lỗi truy cập



Processor Cache

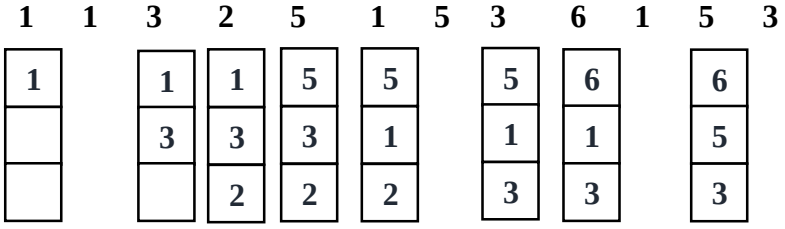
Ta có PF = F /F+S. Tỷ lệ lỗi trang (page fault).

F: Tổng số lần truy cập trang bị lỗi. F + S số lần truy cập trang.

Ví dụ: Tính PF cho một tiến trình truy nhập bộ nhớ theo các trang như sau:

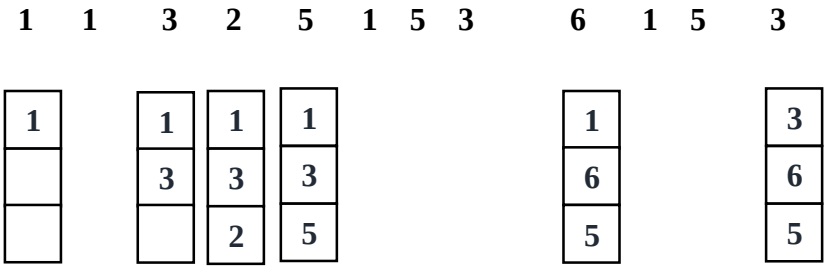
1, 1, 3, 2, 5, 1, 5, 3, 6, 1, 5, 3. Biết bộ nhớ có 3 khung trang và thay thế dữ liệu theo phương pháp: FIFO, OTP, LRU.

Bài giải: Thay thế trang FIFO



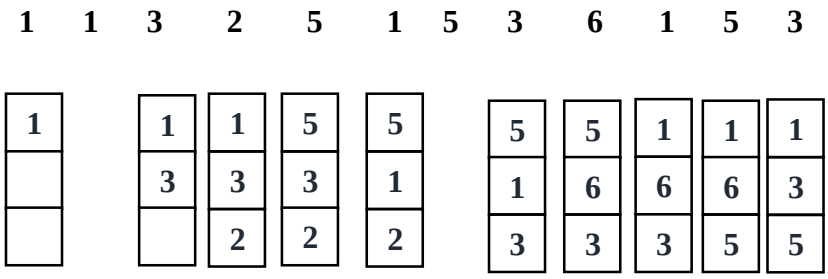
Tỷ lệ lỗi trang: $PF = \frac{F}{F+S} = \frac{8}{12} = 66{,}67\%$

Thay thế trang tối ưu hóa



Tỷ lệ lỗi trang: $PF = \frac{F}{F+S} = \frac{6}{12} = 50\%$

Thay thế trang LRU



Tỷ lệ lỗi trang: $PF = \frac{F}{F+S} = \frac{10}{12} = 83{,}33\%$

4.5. Bộ nhớ ảo và kỹ thuật phân đoạn

4.5.1 Khái niệm phân đoạn

Là một trong các phương pháp tổ chức việc chuyển đổi dữ liệu giữa bộ nhớ ngoài và bộ nhớ trong. Phương pháp này thực hiện tương tự như phân trang nhưng nó có một số đặc điểm sau:

- Kích thước của mỗi đoạn có thể không giống nhau đối với các tiến trình khác nhau hoặc đối với các phần trong một tiến trình.
- Dùng bảng phân đoạn cho mỗi tiến trình để quản lý các đoạn.
- Trong bảng chứa các thông tin: quyền hạn, số lượng đoạn, độ dài đoạn, điểm khởi đầu của đoạn.
- Trong hệ thống lưu một bảng xác định vùng nhớ trống

Để quản lý các đoạn người ta cũng dùng các bảng phân đoạn cho mỗi tiến trình. Trong các bảng này sẽ chứa một số thông tin cơ bản như quyền hạn và số lượng đoạn, độ dài đoạn, điểm khởi đầu của đoạn đó. Ngoài ra trong hệ thống người ta thường lưu một bảng để xác định vùng nhớ còn trống.

4.5.2 Các chiến lược đặt các phân đoạn vào bộ nhớ chính

4.5.2.1 Tìm vị trí vừa nhất (best fit)

Để nạp một đoạn mới vào bộ nhớ chính thì người ta sẽ tìm kiếm trong bảng lưu trữ những đoạn còn trống để lấy ra vị trí có kích thước vừa nhất với kích thước của đoạn cần nạp. Thông thường với phương pháp này người ta sẽ sản xuất bảng lưu các đoạn còn trống tăng dần theo dung lượng của đoạn và sẽ tìm đến phần tử đầu tiên trong bảng mà có kích thước lớn hơn kích thước của đoạn cần nạp.

Sau khi nạp các đoạn vào bộ nhớ thì có khả năng hệ thống sẽ sinh ra các vùng nhớ còn trống mà trong tương lai không vừa cho một đoạn nào người ta gọi là hệ thống phân mảnh (fragment). Để khắc phục trong hệ thống bộ nhớ người ta phải thiết kế các chương trình để chống phân mảnh defragment.

4.5.2.2 Tìm vị trí lớn nhất (Worst fit)

Phương pháp này tương tự như phương pháp trên, người ta sẽ tìm một vùng bộ nhớ còn trống lớn nhất để nạp đoạn. Để nạp một đoạn mới vào bộ nhớ chính thì người ta sẽ tìm kiếm trong bảng lưu trữ những đoạn còn trống để lấy ra vị trí có kích thước lớn nhất vừa với kích thước của đoạn cần nạp. Thông thường với phương pháp này người ta sẽ sản xuất bảng lưu các đoạn còn trống tăng dần theo dung lượng của đoạn và sẽ tìm đến phần tử đầu tiên trong bảng mà có kích thước lớn hơn kích thước của đoạn cần nạp.

4.5.2.3 Tìm vị trí mà vừa đoạn đầu tiên (First fit)

Trong phương pháp này người ta cần phải sắp xếp các vùng bộ nhớ còn trống, hệ thống sẽ kiểm tra các vùng bộ nhớ còn trống để tìm ra vị trí có thể nạp vừa đoạn đầu tiên.

Ví dụ 1: trong kỹ thuật quản lý phân vùng động, các vùng nhớ còn trống có kích thước như sau: 100k, 250k, 260k, 300k, 400k, 95k, 500k, 280k. Vùng nhớ nào sẽ được chọn để nạp chương trình có kích thước 270k theo các giải thuật First Fit, Worst fit, Best fit?

Ví dụ 2: trong kĩ thuật quản lí phân vùng động, các vùng nhớ sau còn trống có kích thước như sau: 100k, 250k, 260k, 300k, 400k, 95k, 500k, 280k. Vùng nhớ nào sẽ được chọn để nạp chương trình có kích thước 150k theo các giải thuật First Fit, Worst fit, Best fit?

4.6. Ngắt và loại trừ

4.6.1 Hiện tượng Ngắt (Interrupt)

Trong thực tế, tốc độ xử lý dữ liệu của CPU cao hơn rất nhiều so với "sự chế biến dữ liệu" của các thiết bị I/O. Vì vậy cần tạo ra một cơ chế vào/ra hợp lý để tăng hiệu suất làm việc của CPU. Ngắt trong hệ thống máy tính nhằm mục đích giải quyết sự bất hợp lý do CPU phải chờ đợi thiết bị ngoại vi. Thiết bị ngoại vi chỉ yêu cầu CPU phục vụ việc nhận hay chuyển giao dữ liệu khi bản thân nó đã sẵn sàng. Để thực hiện tốt yêu cầu này, cơ chế phục vụ ngắt là hợp lý nhất.

Ngắt nghĩa là yêu cầu CPU tạm thời dừng công việc hiện tại để trao đổi hay xử lý dữ liệu không thuộc tuần tự của chương trình đang thực hiện. Ngắt là một hiện tượng xuất

hiện ngẫu nhiên về phương diện thời điểm nhưng được dự đoán trước. Ngắt là hiện tượng thiết bị có yêu cầu ngắt gửi một tín hiệu báo với CPU rằng một sự kiện đã xẩy ra, yêu cầu CPU phải xử lý. Quá trình đang được CPU xử lý sẽ bị tạm thời dừng lại để thực hiện một thao tác khác phục vụ sự kiện có yêu cầu. Khi thao tác này kết thúc, quá trình xử lý vừa bị tạm dừng sẽ được tiếp tục ngay tại nơi nó đã bị tạm dừng. Bản thân sự kiện thông thường là yêu cầu phục vụ của thiết bị ngoại vi đối với CPU. 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.

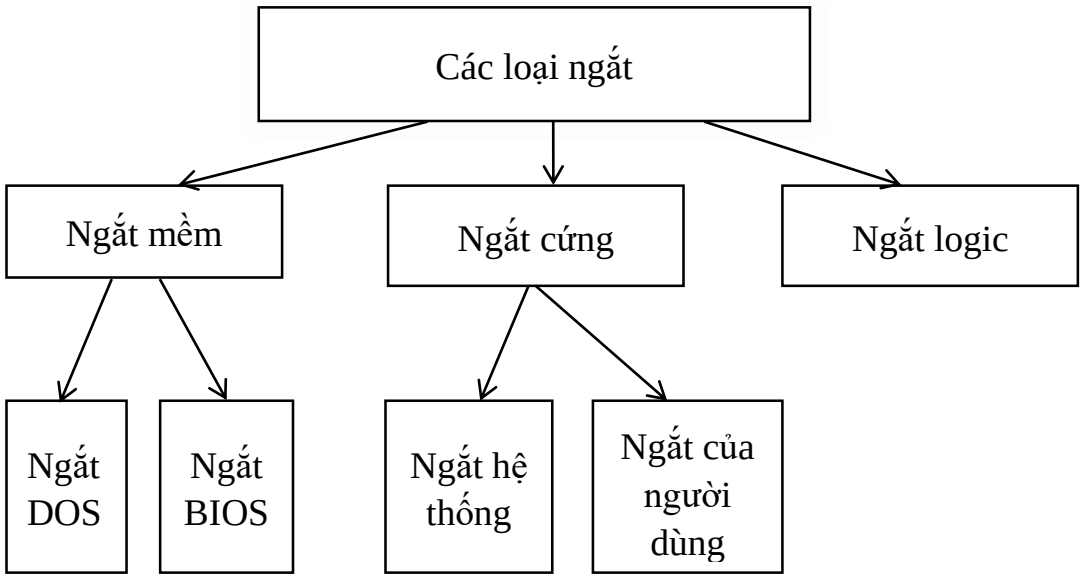
2. Lưu trữ trạng thái hiện tại.

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.

Phân loại ngắt: thuật ngữ "ngắt" xuất phát từ kỹ thuật ngắt cứng. Khi nói đến ngắt cứng, ngắt mềm hoặc ngắt logic (ngoại lệ) là hàm ý nói đến các chương trình con phục vụ hoạt động của hệ thống máy tính và nói đến cách kích hoạt các chương trình con này. Tất cả các chương trình phục vụ ngắt đều có chung đặc điểm: thứ nhất là hầu hết các chương trình con phục vụ ngắt đã được viết sẵn (là các chương trình của hệ điều hành) và được

phép sử dụng; thứ hai là địa chỉ của các chương trình con này phải được đặt ở một vùng xác định là Bảng véc tơ ngắt, nằm trong bộ nhớ chính. Các chương trình con phục vụ ngắt cứng thường được dùng để điều khiển quá trình vào/ra với các thiết bị vào - ra chuẩn ở mức vật lý. Các chương trình con phục vụ ngắt cứng được kích hoạt bởi các tín hiệu vật lý IRQ (Interrupt Request) đến từ thiết bị vào - ra. Các chương trình con phục vụ ngắt mềm là các chương trình hệ thống thực hiện các thao tác vào - ra cơ bản ở mức logic và các hoạt động khác của hệ thống. Các chương trình con phục vụ ngắt mềm được kích hoạt bởi lệnh INT trong tập lệnh của CPU. Các chương trình con phục vụ ngắt logic cũng phục vụ cho hoạt động của hệ thống, nhưng chúng chỉ được kích hoạt khi CPU thực hiện lệnh và do phát sinh một ngoại lệ nào đó.



Hình 4.29 Phân loại ngắt

4.6.2 Hiện tượng Loại trừ

Thường được sản sinh từ các tín hiệu đồng bộ trong hệ thống . Các ngoại trừ thường được sử dụng để kiểm soát các lỗi của lệnh. Chia làm ba loại:

Fault: Có thể được phát hiện và được xử lý trước khi thực hiện lệnh lỗi. Ví dụ chia cho 0, tràn Stack.

Trap (Bẫy lỗi): Được xử lý sau khi lệnh thực hiện mà lệnh đó sinh ra lỗi. Trong thực tế để xử lý trap người ta sử dụng một chương trình con để thông báo cho người sử dụng.

Abort: Thường xảy ra tại các vị trí không thể xác định được và người ta thường sử dụng lỗi loại này mang mục đích thông báo.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 4

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Bộ nhớ máy tính là gì. Trình bày nguyên lý hoạt động và chức năng của bộ nhớ?

Câu 2. Phân loại bộ nhớ và nêu đặc điểm của từng loại?

Câu 3. Trình bày mô hình phân cấp bộ nhớ?

Câu 4. Trình bày công thức xác định số IC cần để xây dựng bộ nhớ?

Câu 5. Trình bày công thức xác định số lượng đường địa chỉ cần sử dụng cho IC nhớ?

Câu 6. Trình bày công thức xác định số lượng đường địa chỉ cần sử dụng cho bộ nhớ?

Câu 7. Trình bày công thức xác định số IC mở đồng thời trong bộ nhớ?

Câu 8. Trình bày khái niệm bộ nhớ cache? Thế nào là cache hit? Cache miss? Hit ratio? Cho ví dụ minh họa?

Câu 9. Trình bày hoạt động của bộ nhớ cache? Cho ví dụ minh họa

Câu 10. Nêu các phương pháp thay thế dữ liệu được sử dụng trong bộ nhớ cache?

Câu 11. Thế nào là bộ nhớ ảo? Đặc điểm của bộ nhớ ảo?

Câu 12. Nêu các phương pháp thay thế dữ liệu được sử dụng trong bộ nhớ ảo?

Câu 13. Thế nào là hiện tượng ngắt quãng? Các giai đoạn thực hiện ngắt quãng của CPU.

Câu 14. Thế nào là hiện tượng loại trừ? Phân biệt các kiểu loại trừ?

Câu hỏi trắc nghiệm

Câu 1. Chọn đáp án sai về qui tắc phân cấp bộ nhớ. Qui tắc chung của hệ thống phân cấp bộ nhớ dựa trên :

A. Mối quan hệ về không gian

B. Mối quan hệ logic

C. Mối quan hệ về thời gian

D. Mối quan hệ tuần tự

Câu 2. Người ta đã thống kê được rằng các lệnh trong chương trình hầu hết được thực hiện một cách tuần tự chiếm:

A. 50-60 % B. 40-50% C. 30-40% D. 70-80%

Câu 3. Mô hình phân cấp bộ nhớ hai cấp:

A. Processor – Secondarymemory - Mainmemory

B. Mainmemory - Processor –Secondarymemory

C. Secondarymemory - Mainmemory - Processor

D. Processor - Mainmemory – Secondarymemory

Câu 4. Khi xây dựng bộ nhớ có dung lượng 1MB * 8bit từ IC 256KB * 8bit thì số IC cần để xây dựng bộ nhớ là

A. 8 B. 4 C. 6 D. 7

Câu 5. Khi xây dựng bộ nhớ có dung lượng 1MB * 8bit từ IC 256KB * 8bit thì số lượng đường địa chỉ cần sử dụng cho bộ nhớ là

A. 20 B. 8 C. 10 D. 12

Câu 6. Khi xây dựng bộ nhớ có dung lượng 512KB * 8bit từ IC 256KB * 8bit thì số IC cần để xây dựng bộ nhớ là

A. 4 B. 2 C. 6 D. 3

Câu 7. Khi xây dựng bộ nhớ có dung lượng 512KB * 8bit từ IC 256KB * 8bit thì số lượng đường địa chỉ cần sử dụng cho bộ nhớ là

A. 22 B. 18 C. 10 D. 19

Câu 8. Khi xây dựng bộ nhớ có dung lượng 512KB * 8bit từ IC 256KB * 8bit thì số lượng đường địa chỉ cần sử dụng cho IC là

A. 18 B. 20 C. 14 D. 16

Câu 9. Khi xây dựng bộ nhớ có dung lượng 512KB * 8bit từ IC 256KB * 8bit thì số lượng IC mở đồng thời là

A. 2 B. 4 C. 1 D. 3

Câu 10. Tính PF cho một tiến trình truy nhập bộ nhớ theo các trang như sau:

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1. Biết bộ nhớ có 3 khung trang và thay thế dữ liệu theo phương pháp FIFO

A. 56 % B. 75% C. 60% D. 80%

Câu 11. Cho Thời gian truy nhập bộ nhớ chính là 150ns. Thời gian truy nhập bộ nhớ Cache là 10ns. Tỷ số trúng cache là = 60%. Truy cập bộ nhớ 10 lần. Tỷ lệ thời gian không cache/ thời gian có cache là

A. 7.72 B. 2.23 C. 2.27 D. 7.27

Câu 12. Công thức xác định số lượng đường địa chỉ cần sử dụng cho bộ nhớ

A. n1 = [log2 (dung lượng IC)]

B. n1 = [ln2 (dung lượng bộ nhớ)]

C. n1 = [log2 (dung lượng bộ nhớ)]

D. n1 = [log10 (dung lượng IC)]

Câu 13. Khi xây dựng bộ nhớ có dung lượng 512 KB * 8bit, từ IC 256 K B* 8 bit thì cần thiết kế bộ giải mã có

A. 1 đầu vào, 2 đầu ra

B. 4 đầu vào, 16 đầu ra

C. 2 đầu vào, 4 đầu ra

D. 3 đầu vào, 8 đầu ra

Câu 14. Cho Thời gian truy nhập bộ nhớ chính là 150ns. Thời gian truy nhập bộ nhớ Cache là 5 ns. Tỷ số trúng cache là = 80%. Truy cập bộ nhớ 10 lần. Tỷ lệ thời gian không cache/ thời gian có cache là:

A. 1.44 B. 1.41 C. 4.41 D. 4.14

Bài tập áp dụng

Câu 1. Nêu các bước thiết kế một bộ nhớ có dung lượng MxN từ các IC nhớ cơ sở PxQ.

Áp dụng:

a) Thiết kế bộ nhớ có dung lượng 16Kx32bit từ các IC nhớ cơ sở 8Kx8bit.
b) Thiết kế bộ nhớ có dung lượng 512Kx32bit từ các IC nhớ cơ sở 16Kx4bit.
c) Thiết kế bộ nhớ có dung lượng 512Kx32bit từ các IC nhớ cơ sở 16Kx16bit.
d) Thiết kế bộ nhớ có dung lượng 1M x 8 bit từ IC 256 K x 8 bit.
e) Thiết kế bộ nhớ có dung lượng 1M x 32 bit từ IC 256 K x 8 bit.
f) Thiết kế bộ nhớ có dung lượng 512K x 8 bit từ IC 256 K x 8 bit.

Câu 2. Tính tỷ suất lỗi trang (PF) cho một tiến trình truy nhập bộ nhớ theo các trang như sau: 1, 1, 3, 2, 5, 3, 6, 1, 5, 3, 4, 5, 6. Biết bộ nhớ có 3 khung trang, thay thế dữ liệu theo phương pháp FIFO.

Câu 3. Tính tỷ suất lỗi trang (PF) cho một tiến trình truy nhập bộ nhớ theo các trang như sau: 1, 1, 3, 2, 5, 3, 6, 1, 5, 3, 4, 5, 6. Biết bộ nhớ có 3 khung trang, thay thế dữ liệu theo phương pháp LRU.

Câu 4. Tính tỷ suất lỗi trang (PF) cho một tiến trình truy nhập bộ nhớ theo các trang như sau: 1, 1, 3, 2, 5, 3, 3, 1, 5, 3, 4, 5, 6, 4. Biết bộ nhớ có 4 khung trang, thay thế dữ liệu theo phương pháp FIFO.

Câu 5. Tính tỷ suất lỗi trang (PF) cho một tiến trình truy nhập bộ nhớ theo các trang như sau: 1, 1, 3, 2, 5, 3, 3, 1, 5, 3, 4, 5, 6, 4. Biết bộ nhớ có 4 khung trang, thay thế dữ liệu theo phương pháp LRU.

Câu 6. Cho Thời gian truy nhập bộ nhớ chính là 150ns. Thời gian truy nhập bộ nhớ Cache là 20ns. Tỷ số trúng cache là = 80%. Truy cập bộ nhớ 10 lần. Tính tỷ lệ thời gian không cache/ thời gian có cache?

Câu 7. Cho biết số lượng IC cần để xây dựng bộ nhớ là 24, số IC cần mở đồng thời là 8, vậy bộ giải mã sẽ có bao nhiêu đầu ra?

Câu 8. Khi xây dựng bộ nhớ có dung lượng 512 KB * 8bit, từ IC 256 K B* 8 bit thì cần thiết kế bộ giải mã với số đầu vào, đầu ra như thế nào?

Câu 9. Khi xây dựng bộ nhớ có dung lượng 1024KB * 128bit từ IC nhớ 512KB * 32bit thì số IC cần để xây dựng bộ nhớ là bao nhiêu?

Câu 10. Thiết kế bộ nhớ có dung lượng 1024KB * 64bit từ các IC nhớ cơ sở 512KB * 32bit thì số lượng IC mở đồng thời là bao nhiêu?

Chương 5: HỆ THỐNG VÀO RA

***Mục đích**: Giới thiệu tổng quan về hệ thống vào ra, các thiết bị ngoại vi và modul vào ra. Phân tích các phương pháp điều khiển vào ra như: vào ra bằng chương trình, vào ra điều khiển bằng ngắt, bằng truy nhập bộ nhớ trực tiếp DMA, kênh vào ra. Tìm hiểu nối ghép thiết bị ngoại vi và các cấu hình nối ghép.*

***Yêu cầu**: Sinh viên nắm được các kiến thức về hệ thống vào ra. Hiểu và phân biệt được các phương pháp điều khiển vào ra, các cấu hình nối ghép thiết bị ngoại vi.*

5.1. Tổng quan về hệ thống vào ra

5.1.1 Giới thiệu chung

Ngoài bộ vi xử lý và bộ nhớ, thành phần quan trọng thứ ba của một hệ thống máy tính là các module vào ra (I/O). Mỗi module giao tiếp với bus hệ thống hoặc bộ chuyển mạch trung tâm và điều khiển một hoặc nhiều thiết bị ngoại vi. Một module I/O không đơn giản chỉ kết nối cơ học các thiết bị với bus hệ thống mà còn chứa các mạch logic thực hiện chức năng truyền thông tin giữa thiết bị ngoại vi và bus.

Vậy tại sao ta không kết nối trực tiếp thiết bị ngoại vi với bus hệ thống. Có một số lý do như sau:

- Có rất nhiều thiết bị ngoại vi với nhiều phương thức hoạt động khác nhau. Nếu bộ xử lý kết nối trực tiếp với thiết bị ngoại vi nó sẽ cần phải được trang bị các mạch logic để điều khiển các thiết bị đó, điều này không cần thiết và gây lãng phí tài nguyên của bộ xử lý.
- Tốc độ truyền dữ liệu của thiết bị ngoại vi thường chậm hơn nhiều so với bộ nhớ hoặc bộ vi xử lý. Do đó, sẽ là không thực tế nếu sử dụng bus tốc độ cao để giao tiếp trực tiếp với thiết bị ngoại vi.
- Mặt khác, tốc độ truyền dữ liệu của một số thiết bị ngoại vi nhanh hơn bộ nhớ hoặc bộ vi xử lý. Một lần nữa, sự chênh lệch này sẽ dẫn đến sự không hiệu quả trong hiệu năng hệ thống nếu ta không có cơ chế quản lý phù hợp.
- Thiết bị ngoại vi thường sử dụng các định dạng dữ liệu và kích thước khác so với máy tính mà chúng được gắn vào.

Do đó, việc sử dụng một module I/O là cần thiết. Module này có hai chức năng chính:

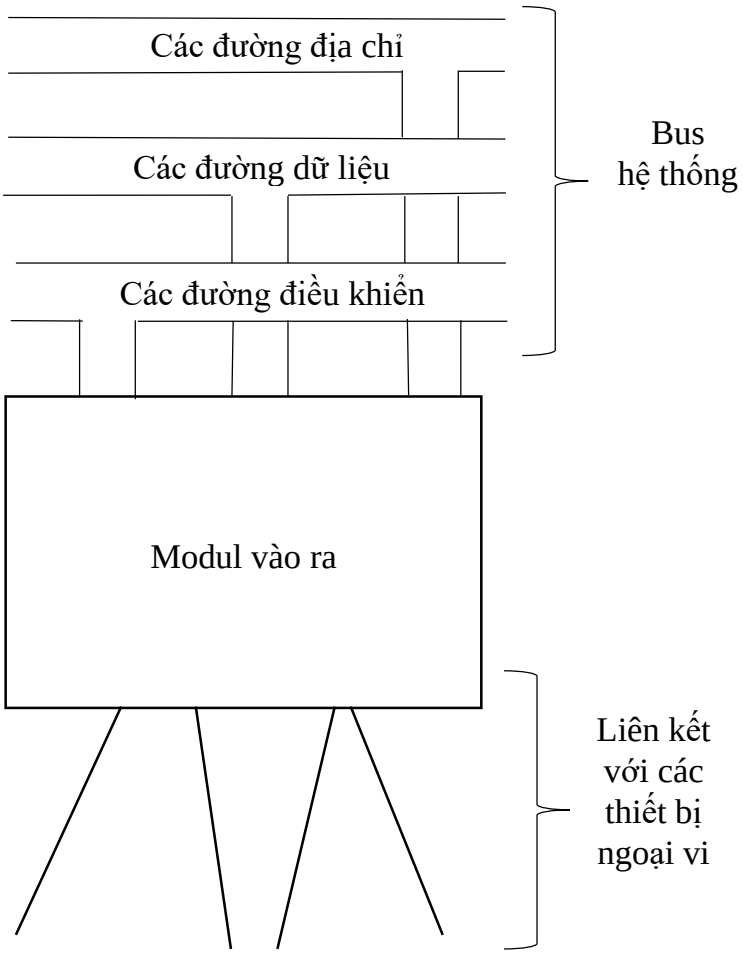
- Giao tiếp với bộ xử lý và bộ nhớ thông qua bus hệ thống hoặc một chuyển mạch trung tâm.
- Giao tiếp với một hoặc nhiều thiết bị ngoại vi bằng các liên kết dữ liệu phù hợp

5.1.2 Các thiết bị ngoại vi

Các hoạt động I/O được thực hiện thông qua một loạt các thiết bị ngoài cung cấp một phương tiện trao đổi dữ liệu giữa môi trường bên ngoài và máy tính. Thiết bị ngoài kết nối với máy tính thông qua một liên kết tới module I/O. Liên kết này được sử dụng để

chuyển tiếp điều khiển, trạng thái và dữ liệu giữa module I/O và thiết bị ngoại vi. Chúng ta có thể phân loại các thiết bị ngoại vi thành ba loại như sau:

- Con người có thể đọc được: thích hợp để giao tiếp với người sử dụng máy tính
- Máy có thể đọc được: thích hợp để giao tiếp với thiết bị
- Truyền thông tin: thích hợp để giao tiếp với các thiết bị từ xa.

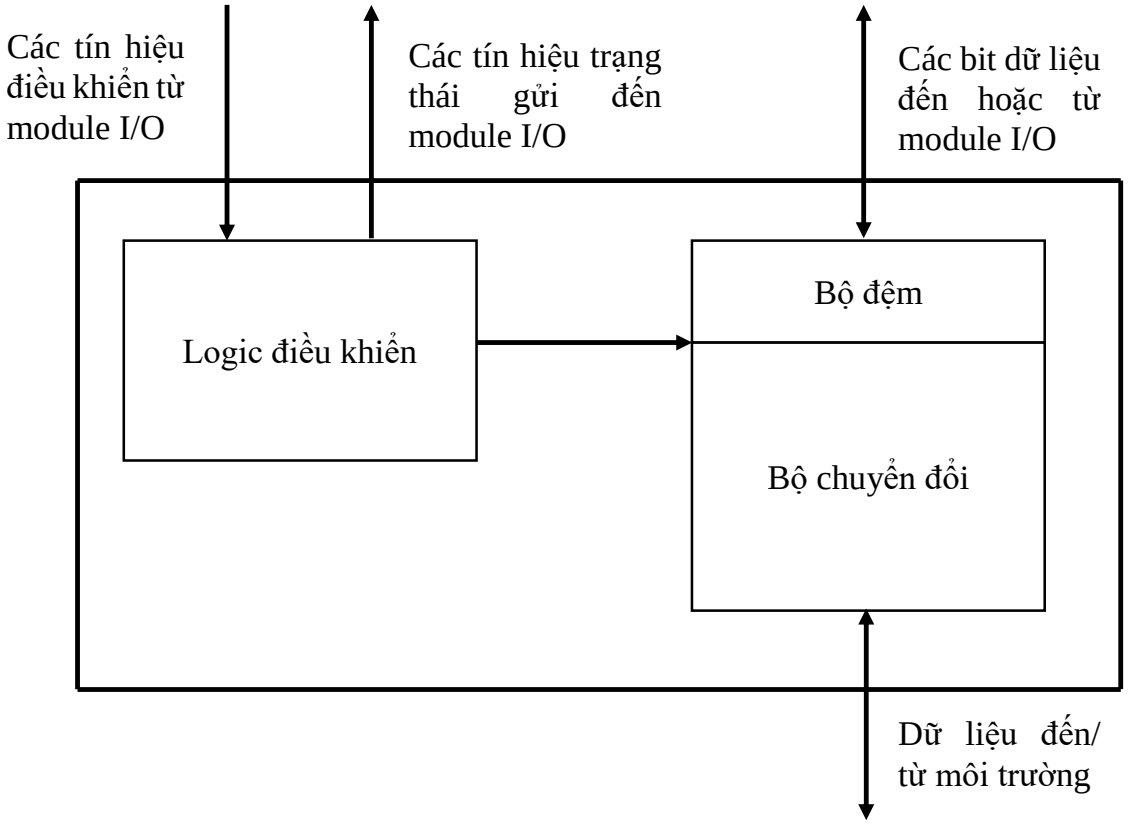


Hình 5.1 Mô hình cơ bản của hệ thống vào ra

Các thiết bị con người có thể đọc được đó là màn hình và máy in. Các thiết bị máy có thể đọc được là các hệ thống đĩa và băng từ, các cảm biến và bộ truyền động được sử dụng trong ứng dụng robot. Về mặt chức năng, các thiết bị này là một phần của hệ thống phân cấp bộ nhớ, tuy nhiên về mặt cấu trúc, các thiết bị này được điều khiển bởi các module I/O và do đó phải được xét đến trong chương này. Thiết bị truyền thông tin cho phép máy tính trao đổi dữ liệu với thiết bị từ xa.

Giao diện với module I/O thông qua các tín hiệu điều khiển, dữ liệu và trạng thái. Các tín hiệu điều khiển xác định chức năng mà thiết bị sẽ thực hiện, chẳng hạn như gửi dữ liệu đến module I/O (INPUT hoặc ĐỌC), tiếp nhận dữ liệu từ module I/O (OUTPUT hoặc GHI), báo cáo trạng thái hoặc thực hiện một số chức năng điều khiển đặc biệt đối với thiết bị (ví dụ: đặt đầu đọc/ghi của đĩa từ vào một vị trí nào đó). Dữ liệu ở dạng một tập bit được gửi tới hoặc nhận từ module I/O.

Các tín hiệu trạng thái cho biết trạng thái của thiết bị. Ví dụ là READY/NOT-READY (sẵn sàng hay chưa sẵn sàng) để thông báo thiết bị có sẵn sàng cho việc truyền dữ liệu hay không. Mạch logic điều khiển tiếp nhận tín hiệu điều khiển từ module I/O và thực hiện việc điều khiển hoạt động thiết bị. Bộ chuyển đổi thực hiện việc chuyển đổi dữ liệu từ dạng tín hiệu điện sang các dạng biểu diễn khác đối với các thiết bị ra và từ các dạng tín hiệu khác nhau thành dữ liệu dạng điện với các thiết bị vào. Bộ chuyển đổi thường đi kèm với một bộ nhớ đệm để lưu trữ dữ liệu tạm thời trong quá trình trao đổi dữ liệu giữa module I/O và môi trường bên ngoài; một kích thước bộ đệm phổ biến từ 8 đến 16 bit.



Hình 5.2 Sơ đồ khối của các thiết bị ngoại vi

5.1.3 Modul vào ra

Các chức năng chính của module vào ra (I/O):

- Điều khiển và định thời
- Giao tiếp với bộ vi xử lý
- Giao tiếp với thiết bị
- Đệm dữ liệu
- Phát hiện lỗi

Trong bất kỳ thời điểm nào, bộ xử lý có thể giao tiếp với một hoặc nhiều thiết bị ngoài theo nhiều cách khác nhau tùy thuộc vào yêu cầu vào/ra dữ liệu của chương trình. Các tài nguyên bên trong, ví dụ như bộ nhớ chính và hệ thống bus sẽ bị chia sẻ với nhiều hoạt động, trong đó gồm cả các hoạt động I/O. Vì vậy, chức năng I/O bao gồm cả các yêu

cầuđiều khiển và định thời để phối hợp luồng lưu lượng giữa các tài nguyên bên trong và các thiết bị ngoài. Ví dụ quá trình điều khiển truyền dữ liệu từ thiết bị ngoại vi đến bộ xử lý gồm các bước sau:

1. Bộ xử lý yêu cầu module I/O kiểm tra trạng thái của thiết bị.

2. Module I/O trả về trạng thái thiết bị.

3. Nếu thiết bị đang hoạt động và sẵn sàng truyền, bộ xử lý yêu cầu truyền dữ liệu bằng cách gửi lệnh cho module I/O.

4. Module I/O nhận một đơn vị dữ liệu (ví dụ, 8 hoặc 16 bit) từ thiết bị ngoài.

5. Dữ liệu được module I/O chuyển sang cho bộ xử lý.

Nếu hệ thống chỉ sử dụng một bus thì tương tác giữa bộ vi xử lý và module I/O sẽ phải sử dụng các cơ chế phân xử bus. Trong đó, module I/O phải thực hiện việc giao tiếp với bộ vi xử lý và thiết bị ngoại vi.

Giao tiếp với bộ xử lý như sau:

- **Giải mã lệnh**: Module I/O nhận lệnh từ bộ xử lý (thường được gửi dưới dạng các tín hiệu trên bus điều khiển). Ví dụ, một module I/O cho ổ đĩa có thể nhận các lệnh sau: READ SECTOR (đọc một sector), WRITE SECTOR (ghi một sector), SEEK track number (tìm kiếm một strack, SCAN record ID (thăm dò một bản ghi). Hai lệnh sau được gửi kèm với một tham số thông qua bus dữ liệu.
- **Dữ liệu**: Dữ liệu được trao đổi giữa bộ xử lý và module I/O qua bus dữ liệu.
- **Báo cáo trạng thái**: Vì các thiết bị ngoại vi thường rất chậm nên bộ xử lý thường phải biết tình trạng của module I/O để yêu cầu thực hiện các hoạt động tiếp theo. Ví dụ, nếu một module I/O được yêu cầu gửi dữ liệu đến bộ xử lý (READ), có thể module này chưa sẵn sàng để thực hiện yêu cầu vì nó vẫn đang thực hiện yêu cầu I/O trước. Điều này cần được báo cáo với bộ xử lý thông qua một tín hiệu trạng thái. Tín hiệu trạng thái thường sử dụng BUSY và READY. Ngoài ra, cũng có thể có một số tín hiệu báo cáo khác với các điều kiện lỗi khác nhau.
- **Nhận dạng địa chỉ**: Giống như mỗi từ nhớ có một địa chỉ, các thiết bị I/O cũng vậy. Một module I/O phải nhận ra một địa chỉ duy nhất cho mỗi thiết bị ngoại vi mà nó điều khiển.

Mặt khác, module I/O phải thực hiện việc giao tiếp với các thiết bị. Giao tiếp này gồm có các lệnh, thông tin trạng thái và dữ liệu. Một nhiệm vụ thiết yếu nữa của module I/O là đệm dữ liệu. Chức năng này hết sức cần thiết do tốc độ trao đổi dữ liệu của bộ nhớ chính và bộ xử lý khá cao so với hầu hết các thiết bị ngoại vi. Dữ liệu từ bộ nhớ chính được gửi đến một module I/O với tốc độ nhanh và cần đươc đệm trong module I/O sau đó gửi đến thiết bị ngoại vi với tốc độ của thiết bị. Ngược lại, dữ liệu được đệm lại để không làm mất thời gian của bộ nhớ nhận truyền chậm. Do đó, module I/O phải hoạt động ở cả tốc độ của bộ nhớ và thiết bị ngoài. Tương tự trong trường hợp nếu thiết bị I/O có tốc độ cao hơn tốc độ truy cập bộ nhớ, module I/O sẽ phải thực hiện hoạt động đệm dữ liệu cần thiết.

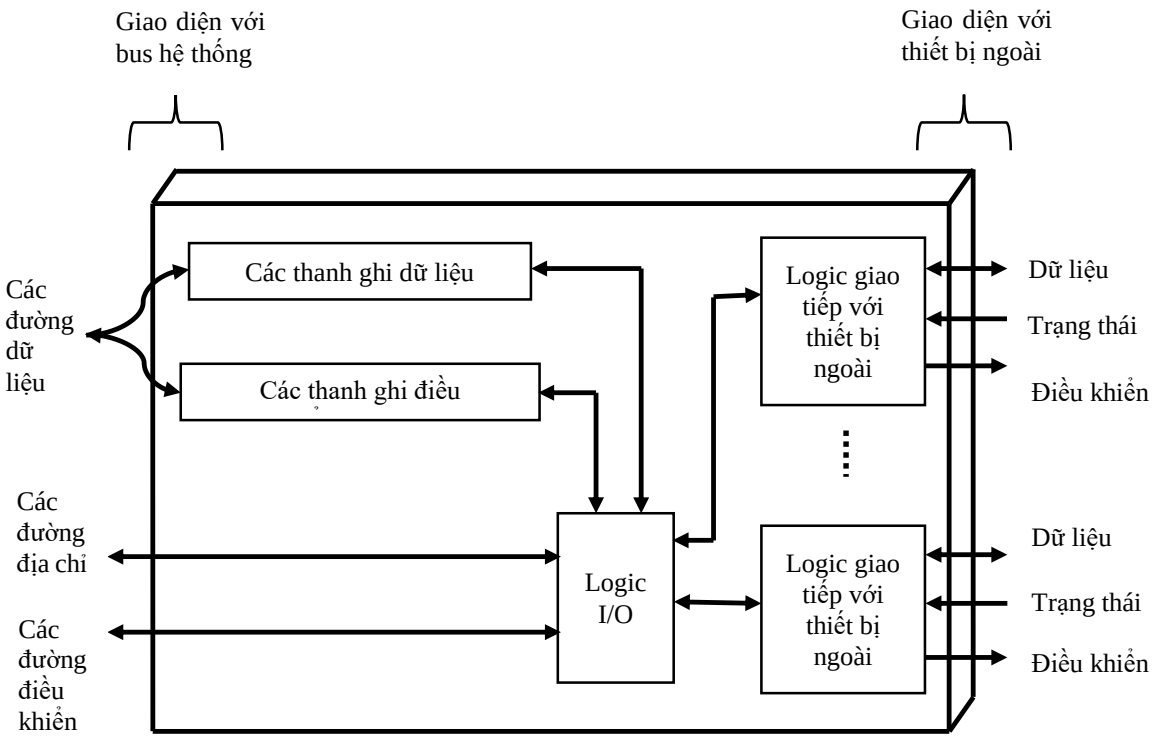
Cuối cùng, một module I/O thường chịu trách nhiệm phát hiện lỗi và sau đó báo lỗi cho bộ vi xử lý. Một số loại bao gồm: hỏng về điện hoặc cơ học (ví dụ: kẹt giấy, một track trong ổ cứng). Một loại lỗi khác có thể do mẫu dữ liệu truyền từ thiết bị đến module I/O vô tình bị thay đổi. Người ta sử dụng một loại mã phát hiện lỗi để phát hiện lỗi truyền tải. Ví dụ như việc truyền mã ký tự IRA, mã này chiếm 7 bit của một byte, vậy bit thứ tám được thiết lập để tổng các bit 1 trong byte là chẵn (parity chẵn) hoặc lẻ (parity lẻ). Khi module I/O nhận được byte, nó sẽ kiểm tra bit chẵn lẻ này để xác định xem có lỗi xảy ra hay không.

Cấu trúc Module I/O:

Các module I/O khác nhau đáng kể về độ phức tạp và số lượng thiết bị bên ngoài mà chúng điều khiển. Tại đây ta sẽ cố gắng chỉ mô tả một cấu trúc chung nhất của các module này. Module này kết nối với phần còn lại của máy tính thông qua một tập hợp các đường tín hiệu (ví dụ: bus hệ thống). Dữ liệu được chuyển đến hoặc từ module được lưu đệm trong các thanh ghi dữ liệu. Các thanh ghi trạng thái cung cấp thông tin về trạng thái hiện tại của thiết bị. Thanh ghi trạng thái cũng có chức năng như một thanh ghi điều khiển: nhận các thông tin điều khiển từ bộ vi xử lý. Logic I/O tương tác với bộ vi xử lý thông qua một tập các đường điều khiển. Bộ vi xử lý sử dụng các đường điều khiển để chuyển các lệnh cho module I/O. Một số đường điều khiển có thể được sử dụng bởi module I/O (ví dụ các tín hiệu phân xử và trạng thái bus).

Module này cũng phải có khả năng nhận dạng và sinh ra các địa chỉ liên kết với các thiết bị mà nó điều khiển. Mỗi module I/O có một (nếu chỉ nối với một TBNV) hoặc một tập địa chỉ (nếu module nối với nhiều TBNV). Cuối cùng, module I/O chứa các logic giao tiếp với từng thiết bị nối vào nó Với hoạt động của module I/O, bộ xử lý có thể nhìn nhận một loạt các thiết bị theo một cách đơn giản. Module I/O có thể ẩn các thông tin về định thời, định dạng, các vấn đề về điện-cơ của thiết bị ngoại vi, bộ xử lý chỉ cần điều khiển hoạt động thông qua các lệnh đọc và ghi đơn giản hoặc có thể là các lệnh đóng, mở tập tin. Tuy nhiên, cũng có những module I/O hoạt động khá đơn giản và vì vậy để lại phần lớn công việc điều khiển thiết bị cho bộ xử lý.

Những module I/O có khả năng xử lý cao, hỗ trợ nhiều cho bộ xử lý thường được gọi là kênh I/O (I/O channel) hoặc bộ xử lý I/O (I/O processor). Những module I/O đơn giản hơn và cần có sự điều khiển chi tiết hơn từ bộ xử lý thường được gọi là bộ điều khiển I/O (I/O controller) hoặc bộ điều khiển thiết bị (device controller). Bộ điều khiển I/O thường được sử dụng trong các máy vi tính, còn kênh I/O thường được sử dụng trên các dòng máy tính lớn (mainframe).



Hình 5.3 Sơ đồ khối của Modul vào ra

5.2 Các phương pháp điều khiển vào ra

Có ba kỹ thuật để thực hiện các hoạt động vào/ra (I/O): vào ra bằng chương trình, vào ra điều khiển bằng ngắt và vào ra bằng cơ chế truy nhập bộ nhớ trực tiếp DMA.

5.2.1 Vào ra bằng chương trình

Dữ liệu được trao đổi giữa bộ vi xử lý và module I/O. Bộ xử lý thực thi một chương trình cho phép nó trực tiếp điều khiển hoạt động vào/ra, bao gồm cảm nhận tình trạng thiết bị, gửi lệnh đọc hoặc ghi, truyền dữ liệu. Khi bộ vi xử lý ra lệnh cho module I/O, nó phải đợi cho đến khi hoạt động I/O hoàn thành. Nếu bộ xử lý nhanh hơn module I/O thì việc chờ đợi này gây lãng phí thời gian của bộ xử lý.

Khi bộ vi xử lý đang thực hiện một chương trình và gặp một chỉ thị (instruction) liên quan đến vào/ra dữ liệu, nó thực hiện chỉ thị đó bằng cách phát ra lệnh (command) cho module I/O thích hợp. Với I/O chương trình, module I/O thực hiện hành động yêu cầu và sau đó thiết lập các bit thích hợp trong thanh ghi trạng thái I/O. Module I/O không có tác vụ nào khác để báo cáo về bộ vi xử lý. Do đó, để biết được công việc đã hoàn thành hay chưa, bộ xử lý phải định kỳ để kiểm tra trạng thái của module I/O cho đến khi hoạt động đã hoàn thành. Để giải thích cho kỹ thuật I/O chương trình, chúng ta xem xét nó từ cách nhìn của các lệnh I/O do bộ vi xử lý gửi đến module I/O và sau đó từ cách nhìn của các chỉ thị được thực hiện bởi bộ vi xử lý.

Các lệnh vào ra:

Để thực hiện chỉ thị liên quan đến vào/ra, bộ xử lý sẽ đưa ra một địa chỉ xác định module I/O và thiết bị bên ngoài cụ thể và và đưa ra một lệnh I/O. Có bốn loại lệnh I/O như sau:

Điều khiển: Được sử dụng để kích hoạt một thiết bị ngoại vi và ra lệnh cho nó phải làm gì. Ví dụ, một ổ đĩa được điều khiển để quay lại hoặc tới trước một bản ghi. Các lệnh điều khiển được thiết kế riêng cho từng loại thiết bị ngoài.

Kiểm tra: Được sử dụng để kiểm tra các điều kiện trạng thái của module I/O và thiết bị ngoại vi. Bộ xử lý cần phải biết liệu các thiết bị ngoại vi có đang bật nguồn và sẵn sàng sử dụng. Nó cũng cần phải biết hoạt động I/O gần nhất đã hoàn thành chưa và có lỗi nào xảy ra không.

Đọc: Yêu cầu module I/O lấy dữ liệu từ thiết bị ngoại vi và đặt nó vào bộ đệm. Bộ xử lý sau đó có thể lấy dữ liệu bằng cách yêu cầu module I/O đặt dữ liệu lên bus dữ liệu.

Ghi: Yêu cầu module I/O chuyển dữ liệu (1 byte hoặc 1 từ) từ bus dữ liệu đến thiết bị ngoại vi. Ví dụ về việc sử dụng I/O chương trình để đọc một khối dữ liệu từ thiết bị ngoại vi (ví dụ: một bản ghi từ băng từ) vào bộ nhớ. Dữ liệu được đọc mỗi lần một từ (16 bit). Đối với mỗi từ được đọc, bộ xử lý phải kiểm tra trạng thái cho đến khi nó xác định được từ đó đã được đọc vào thanh ghi dữ liệu của module I/O. Nhược điểm chính của kỹ thuật này là một quá trình lặp đi lặp lại việc kiểm tra trạng thái làm cho bộ xử lý thêm bận rộn.

Chỉ thị vào ra:

Với I/O chương trình, có sự tương ứng chặt chẽ giữa các chỉ thị (lệnh) liên quan đến vào/ra mà bộ vi xử lý truy xuất từ bộ nhớ và các lệnh I/O ((command) mà bộ xử lý gửi đến module I/O để thực hiện chỉ thị đó. Do đó, các chỉ thị cần được ánh xạ thành các lệnh I/O và chúng thường có quan hệ một-một đơn giản. Dạng của chỉ thị phụ thuộc vào cách các thiết bị ngoài được định địa chỉ.

Thông thường, có nhiều thiết bị I/O được kết nối với hệ thống thông qua các module I/O. Mỗi thiết bị có một số nhận dạng hoặc địa chỉ duy nhất. Khi bộ xử lý đưa lệnh I/O đến thiết bị thì trong lệnh phải chứa thông tin địa chỉ của thiết bị. Sau đó, các module I/O phải dịch các dòng địa chỉ để xác định xem liệu lệnh này có phải gửi cho nó. Khi bộ xử lý, bộ nhớ và module I/O chia sẻ một bus chung, hai chế độ định địa chỉ có thể được thực hiện như sau: I/O ánh xạ bộ nhớ (memory-mapped I/O) và I/O riêng biệt (isolated I/O). I/O ánh xạ bộ nhớ sử dụng một không gian địa chỉ chung cho bộ nhớ và các thiết bị ngoại vi. Bộ xử lý coi thanh ghi trạng thái và dữ liệu của các module I/O giống như vị trí bộ nhớ và sử dụng cùng các lệnh máy để truy cập cả bộ nhớ và các thiết bị ngoại vi. Ví dụ, với 10 đường địa chỉ ta có một không gian địa chỉ gồm 210 = 1024 địa chỉ vị trí bộ nhớ.

Với kỹ thuật I/O riêng biệt, bus được trang bị thêm một đường command line, đường này sẽ cho biết địa chỉ trên bus là địa chỉ của vị trí bộ nhớ hay thiết bị I/O. Như vậy, ta có thể sử dụng toàn bộ không gian địa chỉ cho cả bộ nhớ và thiết vị ngoại vi. Với 10 đường địa chỉ, hệ thống có thể hỗ trợ 1024 vị trí bộ nhớ và 1024 địa chỉ I/O. Giả sử hệ thống có 10-bit địa chỉ, trong đó 512 giá trị dành cho bộ nhớ (từ 0-511) và 512 dành cho I/O (từ 512-1023). Hai địa chỉ được dành riêng cho việc giao tiếp với bàn phím. Địa chỉ 516 được gán cho thanh ghi dữ liệu và 517 được gán cho thanh ghi trạng thái. Thanh ghi trạng thái hoạt động như một thanh ghi điều khiển để nhận các lệnh của bộ vi xử lý. Chương trình trong hình có mục đích đọc 1 byte dữ liệu từ bàn phím vào bộ thanh ghi AC trong bộ xử lý.

Với I/O riêng biệt, các cổng vào/ra chỉ có thể truy cập bằng các lệnh I/O đặc biệt. Các lệnh này sẽ kích hoạt các đường command line trên bus. Hầu hết các bộ xử lý đều có một tập khá lớn các chỉ thị khác nhau để tham chiếu bộ nhớ. Nếu sử dụng I/O riêng biệt, số lượng chỉ thị vào/ra sẽ khá ít. Do đó, ưu điểm của I/O ánh xạ bộ nhớ là số lượng các chỉ thị tham chiếu bộ nhớ cũng có thể được sử dụng trong tham chiếu thiết bị ngoại vi, vì vậy cho phép việc lập trình hiệu quả hơn. Tuy nhiên, nhược điểm của nó là không gian địa chỉ phải chia sẻ với thiết bị ngoại vi. Cả hai kỹ thuật I/O này đều được sử dụng phổ biến.

5.2.2 Vào ra điều khiển bằng ngắt

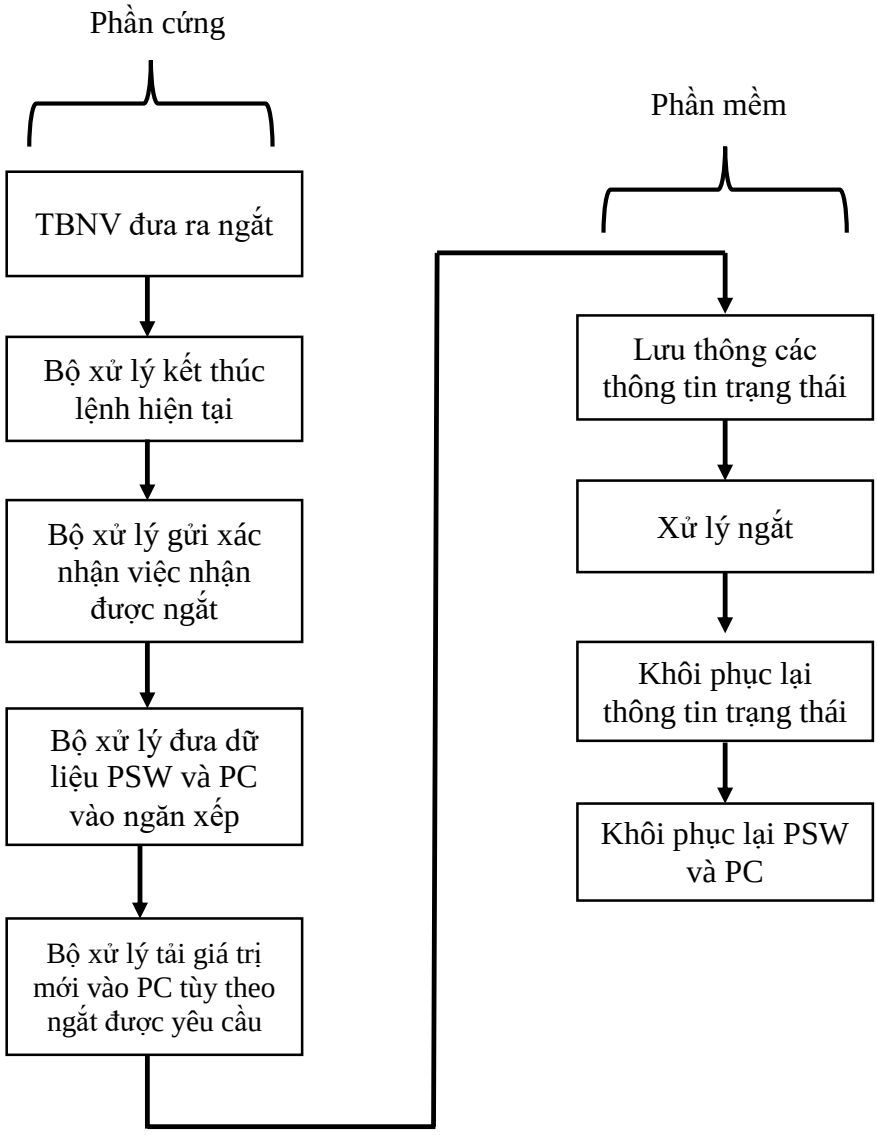
Bộ xử lý đưa ra một lệnh I/O (I/O command) sau đó tiếp tục thực hiện các lệnh (instruction) khác trong chương trình, khi nào module I/O hoàn thành công việc của mình nó sẽ gửi yêu cầu ngắt tới bộ xử lý để bộ xử lý tiếp tục điều khiển hoạt động vào/ra.

Một vấn đề của I/O chương trình là bộ xử lý phải đợi một thời gian dài cho đến khi module I/O sẵn sàng tiếp nhận hoặc truyền dữ liệu. Trong khoảng thời gian đó, bộ xử lý phải liên tục kiểm tra trạng thái của module I/O. Kết quả là hiệu năng của toàn bộ hệ thống suy giảm nghiêm trọng. Để giải quyết vấn đề trên, người ta đưa ra một giải pháp như sau: bộ vi xử lý gửi lệnh I/O đến module và sau đó thực hiện các hoạt động khác. Khi nào module I/O đã sẵn sàng để trao đổi dữ liệu nó sẽ ngắt bộ xử lý để yêu cầu phục vụ. Bộ xử lý thực thi việc truyền dữ liệu rồi quay trở lại công việc của nó. Chúng ta sẽ xem xét cách hoạt động này từ hai phía: bộ xử lý và module I/O. Từ phía module I/O, việc đọc dữ liệu vào như sau: module I/O nhận lệnh READ từ bộ xử lý. Sau đó, nó tiến hành đọc dữ liệu từ một thiết bị ngoại vi được yêu cầu. Khi dữ liệu đã được đọc vào thanh ghi dữ liệu của module, module gửi một báo hiệu ngắt tới bộ xử lý qua một đường điều khiển. Module chờ đợi cho đến khi bộ xử lý gửi yêu cầu dữ liệu. Khi có yêu cầu, module đặt dữ liệu vào bus dữ liệu và được giải phóng, sẵn sàng cho một hoạt động vào/ra khác. Từ phía bộ xử lý, hoạt động Đọc dữ liệu vào như sau: bộ xử lý ra lệnh READ, sau đó nó sẽ thực hiện một công việc khác (ví dụ: bộ xử lý có thể đang chạy trên nhiều chương trình cùng một lúc). Vào cuối mỗi chu kỳ lệnh, bộ xử lý sẽ kiểm tra các ngắt. Khi có ngắt gửi từ module I/O, bộ xử lý lưu lại ngữ cảnh (ví dụ: thanh ghi PC và các thanh ghi khác của bộ xử lý) của chương trình hiện tại và thực hiện việc xử lý ngắt. Khi xử lý ngắt, bộ xử lý đọc dữ liệu từ

module I/O và lưu trữ nó vào bộ nhớ. Sau đó nó khôi phục lại ngữ cảnh trước và tiếp thục thực thi công việc. Việc sử dụng ngắt hiệu quả hơn vì nó giúp loại bỏ khoảng thời gian chờ không cần thiết của bộ xử lý. Tuy nhiên, việc truyền dữ liệu giữa bộ nhớ và module I/O vẫn phải có sự tham gia của bộ xử lý.

Xử lý ngắt:

Ta hãy xem xét chi tiết hơn vai trò của bộ xử lý trong I/O điều khiển ngắt. Sự xuất hiện của một ngắt gây ra một số sự kiện của cả phần cứng và phần mềm. Khi một thiết bị vào/ra thực hiện một hoạt động vào/ra, một chuỗi các hoạt động phần cứng sau đây sẽ xảy ra:



Hình 5.4 Quá trình xử lý ngắt đơn giản

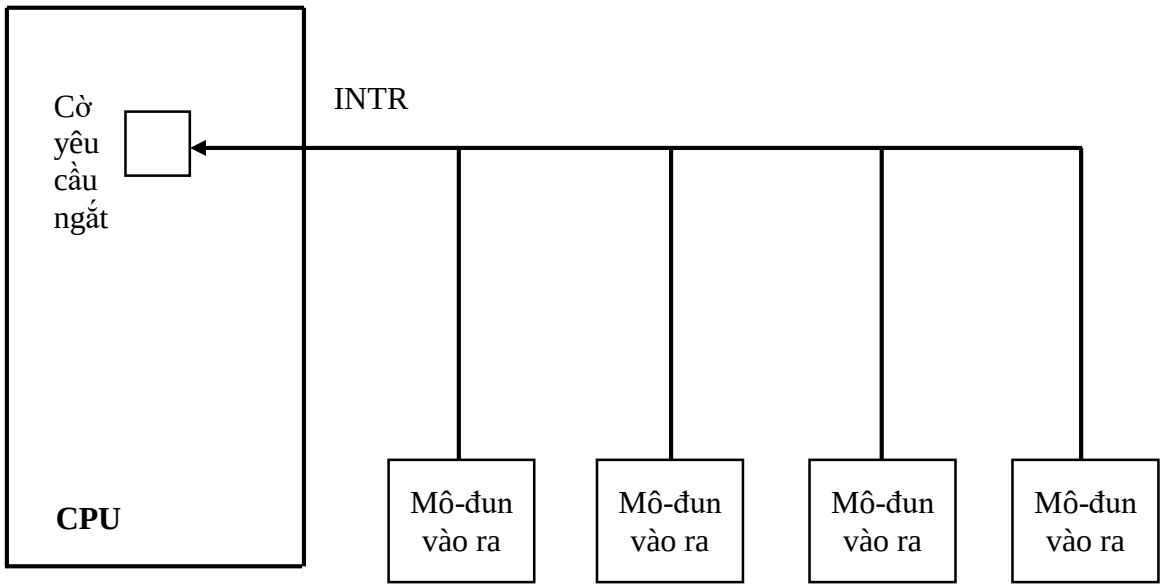
1. Thiết bị phát tín hiệu ngắt cho bộ xử lý.
2. Bộ xử lý hoàn thành lệnh hiện tại trước khi trả lời ngắt.
3. Bộ xử lý kiểm tra xem có ngắt hay không, nếu có một ngắt, và gửi một tín hiệu báo đã nhận (tín hiệu ACK) đến thiết bị đã gửi ngắt. Khi nhận được ACK, thiết bị loại bỏ tín hiệu ngắt.

4. Bộ xử lý cần phải chuyển điều khiển sang chế độ ngắt. Đầu tiên, nó cần phải lưu lại các thông tin của chương trình hiện tại để có thể khôi phục lại công việc sau khi hoàn thành xong việc xử lý ngắt. Các thông tin tối thiểu bắt buộc là (a) trạng thái của bộ xử lý được lưu trữ trong một thanh ghi PSW (thanh ghi trạng thái chương trình), và (b) địa chỉ lệnh tiếp theo sẽ được thực hiện (nội dụng thanh ghi PC). Chúng được đẩy lên vùng bộ nhớ ngăn xếp của hệ thống.
5. Sau đó, bộ xử lý sẽ tải vị trí đầu tiên của chương trình xử lý ngắt vào thanh ghi PC. Tùy thuộc vào kiến trúc hệ thống và thiết kế hệ điều hành, trình xử lý ngắt có thể là một chương trình; một chương trình cho mỗi loại ngắt hoặc một chương trình cho mỗi thiết bị và mỗi loại ngắt. Nếu có nhiều hơn một trình xử lý ngắt, bộ xử lý phải xác định xem ngắt được gửi đến từ đâu. Thông tin đó có thể đã được chứa trong tín hiệu ngắt gửi đến hoặc bộ xử lý phải gửi đến thiết bị gửi ngắt để yêu cầu phản hồi các thông tin cần thiết. Sau khi thanh ghi PC đã được nạp, bộ xử lý bắt đầu thực thi các lệnh trong chương trình xử lý ngắt.
6. Tại thời điểm này, nội dung các thanh ghi PC và PSW của chương trình bị ngắt đã được lưu trên vùng nhớ ngăn xếp của hệ thống. Tuy nhiên, vẫn còn một số thông tin khác liên quan đến trạng thái chương trình cũng cần phải được lưu lại, đặc biệt là nội dung các thanh ghi trong bộ xử lý vì các thanh ghi này có thể được sử dụng bởi trình xử lý ngắt. Do đó, thông thường, trình xử lý ngắt sẽ bắt đầu bằng việc lưu nội dung của tất cả các thanh ghi vào ngăn xếp.
7. Lúc này, ngắt được xử lý bao gồm việc kiểm tra thông tin trạng thái liên quan đến hoạt động I/O hoặc các sự kiện khác gây ra ngắt. Nó cũng có thể thực hiện việc gửi thêm lệnh tiếp theo của một hoạt động I/O đang được thực hiện hoặc gửi tín hiệu ACK đến thiết bị ngoại vi.
8. Khi quá trình xử lý ngắt hoàn tất, giá trị các thanh ghi đã lưu trên ngăn xếp sẽ được lấy ra và khôi phục lại vào các thanh ghi.
9. Hoạt động cuối cùng là khôi phục giá trị PSW và PC từ ngăn xếp. Nhờ đó, chương trinh bị ngắt sẽ được khôi phục lại tiếp tục được thực thi.

Thực hiện ngắt I/O phát sinh hai vấn đề như sau. Thứ nhất, bởi vì trong hệ thống thường có nhiều module I/O, vậy bộ xử lý xác định thiết bị gửi tín hiệu ngắt như thế nào? Thứ hai, nếu nhiều yêu cầu ngắt được cùng gửi đến, bộ xử lý sẽ quyết định phục vụ ngắt nào trước ngắt nào sau? Với vấn đề đầu tiên, có bốn kỹ thuật thường được sử dụng để xác định thiết bị gửi ngắt như sau:

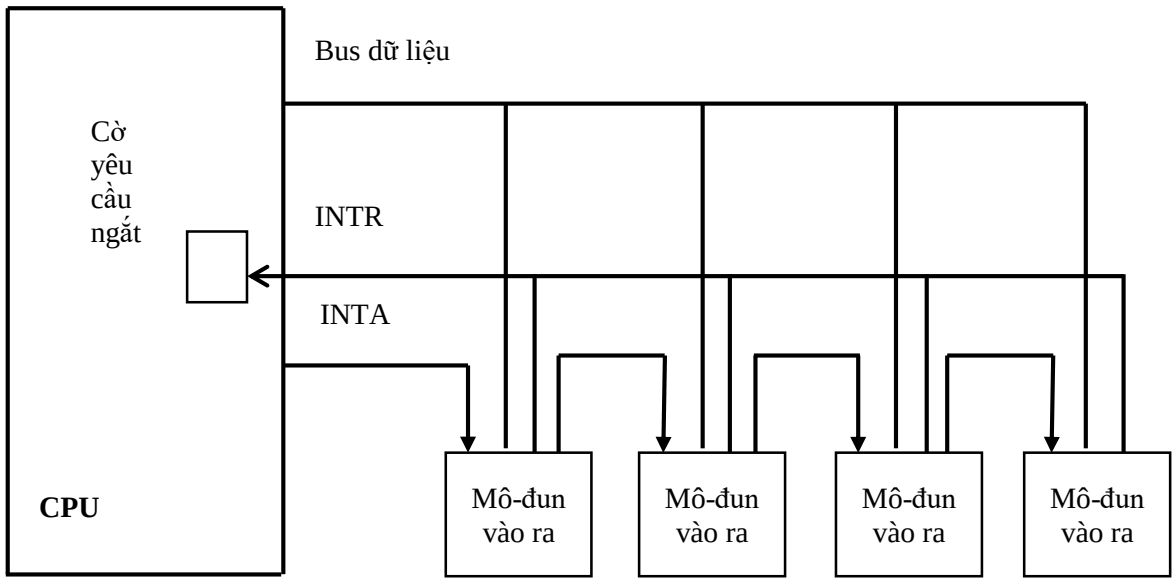
Sử dụng nhiều đường truyền yêu cầu ngắt: giữa bộ xử lý và các module I/O. Tuy nhiên, việc sử dụng các đường bus và các chân của bộ xử lý để làm các đường ngắt là không thực tế. Do đó, ngay cả khi sử dụng kỹ thuật này, có thể mỗi đường ngắt vẫn sẽ nối với nhiều module I/O và vẫn phải sử dụng một trong ba kỹ thuật còn lại cho mỗi đường.

Thăm dò phần mềm: khi bộ xử lý phát hiện ra một ngắt, nó rẽ nhánh sang một trình phục vụ ngắt chung có nhiệm vụ thăm dò từng module I/O để xác định module nào phát ra ngắt. Tín hiệu thăm dò có thể dưới dạng một đường lệnh riêng (ví dụ TESTI/O). Bộ xử lý phát ra tín hiệu TEST I/O và đặt địa chỉ của từng module vào các đường địa chỉ. Module I/O gửi yêu cầu ngắt sẽ trả lời khi nhận được thăm dò. Một cách thăm dò khác là bộ xử lý sẽ đọc các thanh ghi trạng thái của từng module để xác định module nào gửi ngắt. Sau khi xác định được module ngắt, bộ xử lý sẽ chuyển điều khiển sang trình phục vụ ngắt dành riêng cho thiết bị đó. Tuy nhiên, nhược điểm của phương pháp này là tốn thời gian.



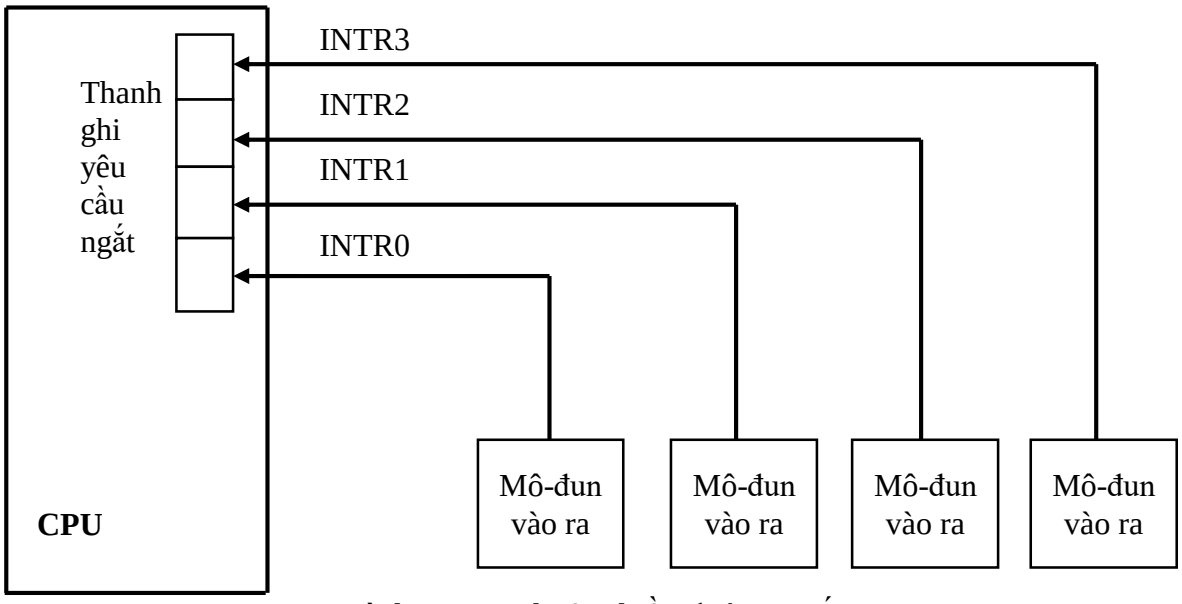
Hình 5.5 Kỹ thuật thăm dò phần mềm

Kỹ thuật chuỗi Daisy thực hiện thăm dò bằng phần cứng. Kỹ thuật này sử dụng một đường ngắt chung (INTR) cho tất cả các module. Khi nhận được yêu cầu ngắt, bộ xử lý gửi một tín hiệu ACK thừa nhận ngắt, tín hiệu này được truyền trên một đường INTA đi qua tất cả các module (nối chuỗi) cho đến khi gặp được module gửi ngắt, module này sẽ gửi lại một tín hiệu trả lời bằng cách đặt một từ lên bus dữ liệu, từ này là có thể là địa chỉ hoặc thông tin nhận diện của module đó, được gọi là *vector*. Khi đó, bộ xử lý sẽ sử dụng vector để trỏ tới trình phục vụ ngắt của module đó. Với kỹ thuật này, bộ xử lý không cần phải rẽ nhánh đến trình phục vụ ngắt chung như trường hợp trên.



Hình 5.6 Kỹ thuật chuỗi Daisy

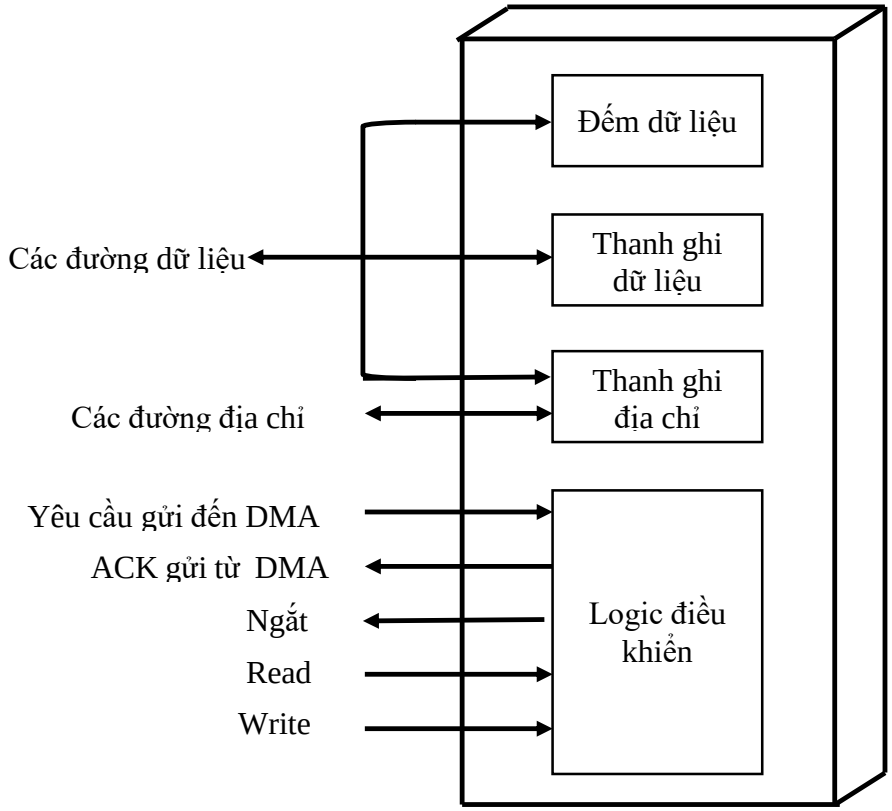
Kỹ thuật phân xử bus: một module I/O muốn gửi yêu cầu ngắt thì cần phải có quyền chiếm bus trước. Vì vậy, tại một thời điểm chỉ có một module có thể sử dụng bus. Khi bộ xử lý phát hiện ra ngắt, nó trả lời bằng đường ACK. Khi nhận được ACK, module yêu cầu đặt các vector của nó vào các đường dữ liệu và bộ xử lý cũng thực hiện việc xử lý ngắt tương tự như với kỹ thuật chuỗi Daisy. Với các kỹ thuật trên, ta cũng có thể giải quyết luôn vấn đề xác định thứ tự ưu tiên trong trường hợp có nhiều ngắt gửi đến bộ xử lý cùng một thời điểm. Với kỹ thuật nhiều đường ngắt, mỗi đường ngắt sẽ được gán một độ ưu tiên nhất định, khi đó, bộ xử lý chỉ cần chọn đường ngắt có độ ưu tiên cao nhất. Với thăm dò phần mềm, bộ xử lý sẽ thực hiện việc thăm dò các module lần lượt theo thứ tự ưu tiên nó. Tương tự, thứ tự các module được nối chuỗi daisy sẽ được xác định theo độ ưu tiên của chúng. Cuối cùng, trong kỹ thuật phân xử bus, module có độ ưu tiên cao hơn sẽ được cho phép chiếm bus trước, vì vậy cũng đã giải quyết được vấn đề này.



Hình 5.7 Kỹ thuật nhiều đường ngắt

5.2.3 Vào ra DMA

Khi cần trao đổi một khối lượng lớn dữ liệu, người ta đưa ra một kỹ thuật hiệu quả hơn, đó là: truy cập bộ nhớ trực tiếp (DMA). Với cả I/O chương trình và I/O điều khiển ngắt, bộ vi xử lý có trách nhiệm lấy dữ liệu từ bộ nhớ chính rồi chuyển cho thiết bị ngoại vi (hoạt động Ghi) hoặc lưu dữ liệu từ thiết bị ngoài vào bộ nhớ chính (hoạt động Đọc). Vì vậy, một cơ chế thay thế được đưa ra là cơ chế DMA (direct memory access - truy cập bộ nhớ trực tiếp) để giảm thiểu sự tham gia của bộ xử lý vào quá trình trao đổi dữ liệu giữa bộ nhớ và thiết bị ngoại vi, tăng hiệu năng của hệ thống.



Hình 5.8 Sơ đồ khối DMA

DMA thường là một module bổ sung trên hệ thống bus. Module DMA có khả năng bắt chước bộ xử lý, tức là tiếp nhận việc điều khiển hệ thống từ bộ xử lý. Nó cần phải thực hiện điều này để trao đổi dữ liệu với bộ nhớ qua bus hệ thống. Với mục đích này, module DMA chỉ sử dụng bus khi bộ xử lý không cần đến nó hoặc DMA phải buộc bộ xử lý phải tạm ngừng hoạt động (gọi là trộm chu kỳ). Kỹ thuật trộm chu kỳ thường phổ biến hơn. Khi bộ xử lý muốn đọc hoặc ghi một khối (block) dữ liệu, nó sẽ đưa ra một lệnh cho module DMA bằng cách gửi tới module DMA các thông tin sau:

- Yêu cầu Đọc hoặc Ghi bằng cách sử dụng các đường điều khiển READ hoặc WRITE bộ xử lý và module DMA. Địa chỉ của thiết bị I/O liên quan, gửi qua các đường dữ liệu.
- Địa chỉ vị trí đầu tiên trong bộ nhớ để đọc hoặc ghi lên, truyền qua các đường dữ liệu và được lưu trữ lại bởi module DMA trong Thanh ghi địa chỉ.

- Số lượng từ cần đọc hoặc ghi, truyền qua các đường dữ liệu và được lưu trữ trong thanh ghi Đếm dữ liệu. Sau đó, bộ xử lý tiếp tục thực hiện các công việc khác. Hoạt động vào/ra được ủy quyền hoàn toàn cho module DMA. Module DMA chuyển toàn bộ khối dữ liệu giữa bộ nhớ và thiết bị ngoại vi mà không cần sự tham gia của bộ xử lý. Khi quá trình trao đổi hoàn tất, module DMA gửi một tín hiệu ngắt tới bộ xử lý. Do đó, bộ xử lý chỉ tham gia vào thời điểm bắt đầu và kết thúc quá trình trao đổi.

Như đã nói ở trên, khi DMA cần chiếm bus để truyền, nó sẽ trộm chu kỳ bus. Trong mỗi trường hợp, bộ xử lý bị treo ngay trước khi nó cần sử dụng bus. Module DMA chuyển một từ sau đó trả lại điều khiển cho bộ xử lý. Lưu ý rằng đây không phải là ngắt; bộ xử lý không cần lưu trữ ngữ cảnh hoặc bất cứ thông tin gì. Đơn giản là bộ xử lý chỉ dừng lại một chu kỳ bus. Về mặt hiệu năng của bộ xử lý thì nó sẽ hoạt động chậm hơn. Tuy nhiên, với việc trao đổi vào/ra nhiều từ, cơ chế DMA hiệu quả hơn rất nhiều so với I/O điều khiển ngắt hoặc I/O chương trình. Cơ chế DMA có thể được cấu hình theo nhiều cách khác nhau. Tất cả các module chia sẻ chung hệ thống bus. Module DMA đại diện cho bộ xử lý sử dụng I/O chương trình để trao đổi dữ liệu giữa bộ nhớ và module I/O. Cấu hình này mặc dù không tốn kém nhưng rõ ràng là không hiệu quả. Giống như I/O chương trình do bộ xử lý điều khiển, mỗi lần truyền một từ sẽ tiêu tốn hai chu kỳ bus. Số lượng chu kỳ bus có thể được giảm bớt bằng cách tích hợp DMA và các module I/O. Module DMA kết nối trực tiếp với một hoặc nhiều module I/O mà không đi qua bus hệ thống. Logic DMA có thể là một phần của module I/O hoặc có thể là một module riêng biệt điều khiển một hoặc nhiều module I/O. Phương thức này có thể được cải tiến thêm một bước nữa bằng cách kết nối các module I/O với module DMA qua một bus I/O. Cấu hình này giảm số giao diện ngoại vi của module DMA thành một và cho phép khả năng mở rộng (tăng thêm số lượng module I/O thì chỉ cần kết nối với bus I/O).

5.2.4 Kênh vào ra hay bộ xử lý vào ra

Quá trình phát triển các chức năng vào ra:

Khi hệ thống máy tính đã trở nên phát triển, mô hình máy tính ngày càng phức tạp và tinh vi. Điều này thể hiện rõ nhất ở chức năng của các module vào/ra. Các bước phát triển của chức năng vào/ra có thể tóm tắt như sau:

1. Ban đầu, CPU trực tiếp điều khiển một thiết bị ngoại vi. Điều này được thấy trong các vi điều khiển đơn giản.

2. Sử dụng các bộ điều khiển hoặc module I/O. CPU sử dụng cơ chế I/O chương trình. Từ thời điểm này, CPU bắt đầu tách ra khỏi việc điều khiển chi tiết thiết bị ngoài.

3. Cấu trúc tương tự như trong giai đoạn hai, tuy nhiên sử dụng cơ chế I/O điều khiển ngắt. CPU không phải tốn nhiều thời gian chờ đợi một hoạt động I/O được thực hiện, do đó tăng hiệu năng của hệ thống.

4. Module I/O được truy cập trực tiếp tới bộ nhớ thông qua cơ chế DMA. Nó có thể thực hiện việc trao đổi một khối dữ liệu đến hoặc đi từ bộ nhớ mà không cần có sự tham gia của CPU ngoại trừ thời điểm bắt đầu và kết thúc quá trình truyền.

5. Module I/O được cải tiến để trở thành một bộ xử lý I/O chuyên thực thi các hoạt động I/O. CPU khi này chỉ cần đưa ra chỉ thị cho bộ xử lý I/O thực hiện một chương trình I/O trong bộ nhớ. Bộ xử lý I/O sẽ nạp và thực hiện các chỉ thị này mà không cần sự can thiệp của CPU. Điều này cho phép CPU có thể chỉ thị một loạt các hoạt động vào/ra và chỉ bị ngắt khi nào toàn bộ công việc đã được thực hiện.

6. Module I/O có bộ nhớ cục bộ của riêng nó, thực chất nó có thể được coi là một máy tính với khả năng nhất định. Với kiến trúc này, một loạt các thiết bị ngoại vi có thể được điều khiển với sự tham gia tối thiểu của CPU. Bộ xử lý I/O sẽ thực hiện hầu hết các nhiệm vụ liên quan đến việc điều khiển các thiết bị đầu cuối.

Với quá trình phát triển này, điều dễ nhận thấy là sự tham gia của CPU vào các tác vụ vào/ra ngày càng ít, do đó giúp cải thiện hiệu suất CPU. Ở giai đoạn năm và sáu, một bước thay đổi lớn đối với khái niệm module I/O, module I/O từ đây có khả năng thực hiện các chương trình. Ở giai đoạn năm, module I/O sẽ được gọi kênh I/O. Còn với bước 6, người ta sử dụng thuật ngữ bộ xử lý I/O. Tuy nhiên, thông thường cả hai thuật ngữ này đều được sử dụng ngang nhau trong cả hai giai đoạn. Ở phần sau chúng tôi sẽ dùng chung thuật ngữ Kênh I/O.

Đặc điểm của các Kênh I/O

Khái niệm Kênh I/O là mở rộng của khái niệm DMA. Kênh I/O có khả năng thực hiện các lệnh vào/ra, cho phép điều khiển hoàn toàn các hoạt động vào/ra. Trong một hệ thống máy tính với các thiết bị như vậy, CPU không thực hiện các lệnh vào/ra. Những chỉ thị này được lưu trữ trong bộ nhớ chính và được bộ xử lý của Kênh I/O thực thi. Do đó, khi có một hoạt động vào/ra, CPU chỉ thị cho Kênh I/O để thực thi một chương trình trong bộ nhớ. Chương trình này sẽ định ra một hoặc nhiều thiết bị, một hoặc nhiều vùng bộ nhớ để lưu trữ, các mức ưu tiên và hoạt động sẽ được thực hiện trong trường hợp có các điều kiện lỗi nhất định. Kênh I/O sẽ theo các chỉ thị này và điều khiển việc truyền dữ liệu. Có hai loại Kênh I/O phổ biến. Kênh Chọn điều khiển nhiều thiết bị tốc độ cao và được dành riêng cho việc truyền dữ liệu với một trong những thiết bị đó tại bất cứ thời điểm nào. Do đó, Kênh I/O lựa chọn một thiết bị và tác động đến việc truyền dữ liệu. Một thiết bị hoặc một nhóm thiết bị được điều khiển bởi một bộ điều khiển hoặc module I/O. Do đó, các Kênh I/O sẽ thay cho CPU trong việc kiểm soát, điều khiển các module I/O. Kênh Ghép kênh có thể xử lý vào/ra nhiều thiết bị cùng một lúc.

5.3 Nối ghép thiết bị ngoại vi

5.3.1 Các kiểu nối ghép vào ra

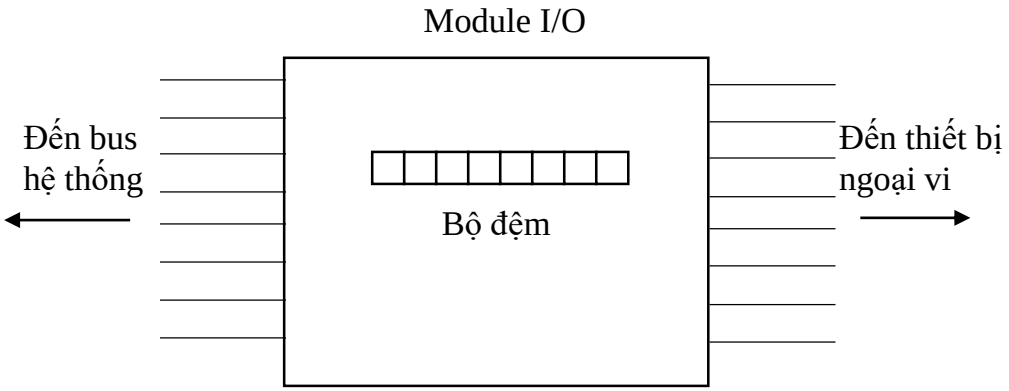
Giao tiếp giữa một thiết bị ngoại vi và một module I/O phải được điều chỉnh phù hợp với tính chất và hoạt động của thiết bị ngoại vi. Đặc điểm chính của giao tiếp này là

nó có thể là giao tiếp nối tiếp hoặc giao tiếp song song. Trong giao tiếp song song, có nhiều đường kết nối giữa module I/O và thiết bị ngoại vi, các bit được truyền đồng thời giống như việc truyền một từ trên bus dữ liệu. Trong một giao tiếp nối tiếp, chỉ có một đường truyền dữ liệu, mỗi bit được truyền một thời điểm. Giao tiếp song song thường được sử dụng cho các thiết bị ngoại vi tốc độ cao như băng từ, đĩa từ, còn giao tiếp nối tiếp thường được sử dụng cho máy in và các thiết bị đầu cuối khác. Ngày nay, với sự phát triển của các giao tiếp nối tiếp tốc độ cao, giao tiếp song song đang trở nên ít phổ biến hơn. Trong cả hai trường hợp, để thực hiện việc trao đổi dữ liệu, module I/O phải đối thoại với thiết bị ngoại vi. Ví dụ với trường hợp thực hiện hoạt động ghi như sau:

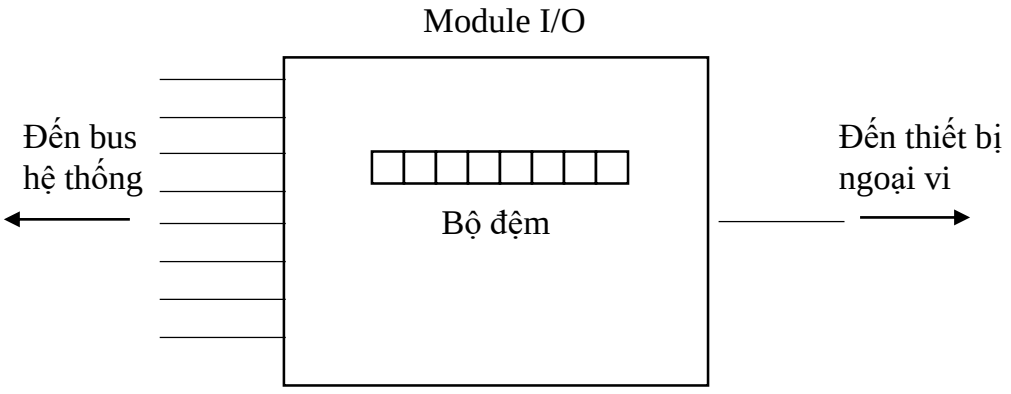
1. Module I/O gửi tín hiệu điều khiển yêu cầu cho phép gửi dữ liệu.

2. Thiết bị ngoại vi chấp nhận yêu cầu Module I/O truyền dữ liệu (một từ hoặc một khối dữ liệu tùy thuộc vào thiết bị ngoại vi).

3. Thiết bị ngoại vi nhận dữ liệu và gửi xác nhận việc đó. Hoạt động đọc cũng được thực hiện với các thao tác tương tự. Một thành phần quan trọng của module I/O là bộ đệm, nó có tác dụng lưu trữ dữ liệu tạm thời để cân bằng sự chênh lệch tốc độ giữa bus hệ thống và các đường kết nối ngoài.



Hình 5.9 Giao tiếp song song



Hình 5.10 Giao tiếp nối tiếp

5.3.2 Các cấu hình nối ghép

Kết nối giữa một module I/O và các thiết bị ngoài có thể là kết nối điểm - điểm hoặc kết nối đa điểm.

Cấu hình Điểm - Điểm

Với giao tiếp điểm - điểm, module I/O được nối với mỗi thiết bị ngoại vi qua một đường riêng. Trên các hệ thống nhỏ (các máy tính cá nhân hoặc máy trạm), các kết nối điểm - điểm điển hình gồm có: kết nối với bàn phím, máy in và modem ngoài.

Cấu hình Điểm - Đa điểm

Giao tiếp đa điểm ngày càng phổ biến và trở nên quan trọng hơn. Giao tiếp này thường được sử dụng để hỗ trợ các thiết bị lưu trữ ngoài (như ổ đĩa và băng từ) và các thiết bị đa phương tiện (CD-ROM, video, audio). Các giao tiếp điểm - đa điểm này là các bus mở rộng và có cùng logic hoạt động giống bus. Ví dụ: USB (Universal Serial Bus): 127 thiết bị. IEEE 1394 (FireWire): 63 thiết bị.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 5

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Tại sao không kết nối trực tiếp thiết bị ngoại vi với bus hệ thống?

Câu 2. Vẽ sơ đồ khối của các thiết bị ngoại vi?

Câu 3. Chức năng chính của modul vào ra?

Câu 4. Nêu các đặc điểm của các phương pháp điều khiển vào ra?

Câu 5. Phân tích các kiểu nối ghép thiết bị ngoại vi?

Câu 6. Trình bày các cấu hình nối ghép thiết bị ngoại vi?

Câu 7. Vẽ sơ đồ khối các kiểu nối ghép vào ra?

Câu 8. Trình bày sự phát triển các chức năng vào ra?

Câu 9. Vẽ sơ đồ khối cho quá trình xử lý bằng ngắt?

Câu 10. Vẽ sơ đồ khối của mô hình cơ bản của hệ thống vào ra?

Câu hỏi trắc nghiệm

Câu 1. Màn hình có chức năng

A. Xử lý thông tin B. Hiển thị thông tin

C. In thông tin D. Nhập thông tin

Câu 2. Card màn hình có chức năng

A. Xử lý thông tin B. Hiển thị thông tin

C. Xử lý đồ họa D. Nhập thông tin

Câu 3. Bàn phím là được gọi là thiết bị

A. Nhập chuẩn B. In thông tin

C. Xuất thông tin D. Xem thông tin

Câu 4. Bàn phím sử dụng phương pháp truyền

A. Song song B. Đồng bộ C. Bất đồng bộ D. Nối tiếp

Câu 5. Quá trình vào ra dữ liệu giữa thiết bị ngoại vi và bộ nhớ theo phương thức DMA là:

A. Truy cập bộ nhớ gián tiếp qua CPU

B. Vào ra dữ liệu theo ngắt cứng.

C. Truy cập bộ nhớ trực tiếp.

D. Vào ra dữ liệu theo ngắt mềm.

Câu 6. Có các loại ngắt sau trong máy tính:

A. Ngắt cứng, ngắt mềm, ngắt ngoại lệ

B. Ngắt cứng, ngắt mềm, ngắt trung gian

C. Ngắt ngoại lệ, ngắt cứng, ngắt INTR

D. Ngắt mềm, ngắt NMI, ngắt cứng

Câu 7. Khi Bộ xử lý đang thực hiện chương trình, nếu có ngắt (không bị cấm) gửi đến, thì nó:

A. Thực hiện xong chương trình rồi thực hiện ngắt

B. Thực hiện xong lệnh hiện tại, rồi phục vụ ngắt, cuối cùng quay lại thực hiện tiếp chương trình.

C. Từ chối ngắt, không phục vụ

D. Phục vụ ngắt ngay, sau đó thực hiện chương trình

Câu 8. Với phương pháp vào/ra bằng ngắt, phát biểu nào sau đây là đúng:

A. Thiết bị ngoại vi là đối tượng chủ động trong trao đổi dữ liệu

B. Là phương pháp hoàn toàn xử lý bằng phần cứng

C. CPU là đối tượng chủ động trong trao đổi dữ liệu

D. Là phương pháp hoàn toàn xử lý bằng phần mềm

Câu 9. Các cấu hình nối ghép gồm có

A. Điểm - điểm

B. Điểm - đa điểm

C. Điểm - điểm và điểm - đa điểm

D. Đa điểm - đa điểm

Câu 10. Các kiểu nối ghép vào ra?

A. Giao tiếp song song

B. Giao tiếp nối tiếp

C. Giao tiếp đơn

D. Giao tiếp song song và giao tiếp nối tiếp

Chương 6: MỘT SỐ KIẾN TRÚC HIỆN ĐẠI

Mục đích: *Phân loại kiến trúc máy tính dựa trên số lượng bộ xử lý, số lượng các chương trình có thể thực hiện. Giới thiệu kiến trúc song song và mạng liên kết trong. Phân biệt được hai loại kiến trúc: CISC (Complex Instruction Set Computer), RISC (Reduced Instruction Set Computer). Giới thiệu một số kiến trúc máy tính trong tương lai.*

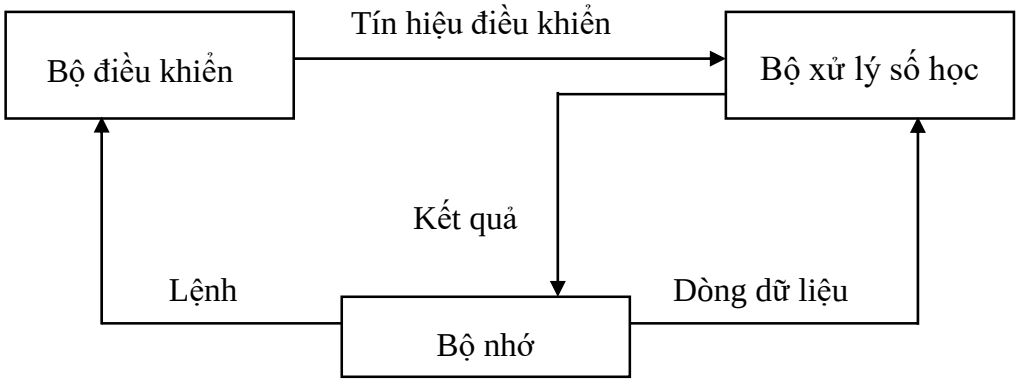
Yêu cầu: *Sinh viên phải nắm được kiến trúc song song, kiến trúc RISC, CISC và mạng liên kết trong. Biết được xu hướng kiến trúc máy tính trong tương lai.*

6.1 Phân loại Kiến trúc máy tính

Có rất nhiều cách phân loại các kiến trúc song song khác nhau trong đó sự phân loại của Micheal Flynn được dùng phổ biến nhất. Flynn phân loại kiến trúc máy tính dựa trên một số đặc tính sau: số lượng bộ xử lý, số lượng các chương trình chúng có thể thực hiện và cấu trúc của bộ nhớ. Sự phân loại này đã dẫn đến bốn mô hình kiến trúc như sau.[1] [4]

6.1.1 SISD (Single Instruction Stream, Single Data Stream)

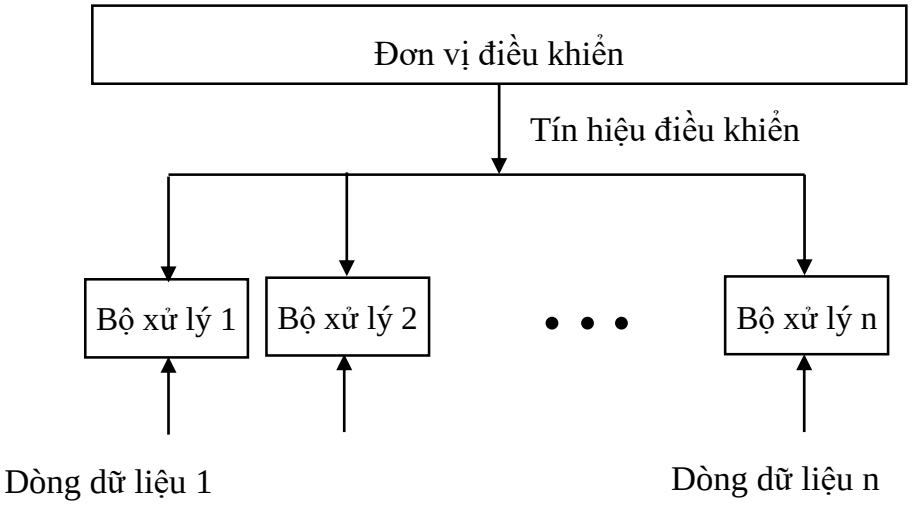
Những máy tính SISD chỉ có một CPU thực hiện duy nhất một lệnh và đồng thời cũng chỉ có thể lấy hoặc lưu trữ một đối tượng dữ liệu tại một thời điểm. Đây chính là cấu trúc tuần tự theo kiểu Von Neumann.



Hình 6.1 Mô hình máy tính SISD

6.1.2 SIMD (Single Instruction Stream Multiple Data Stream)

Các máy SIMD có một đơn điều khiển chỉ thực hiện một lệnh tại một thời điểm, tuy nhiên chúng có nhiều hơn một bộ xử lý. Đơn vị điều khiển phát ra tín hiệu điềukhiển cho tất cả các bộ xử lý và các bộ xử lý này sẽ thực hiện cùng một thao tác trên các đối tượng dữ liệu khác nhau.



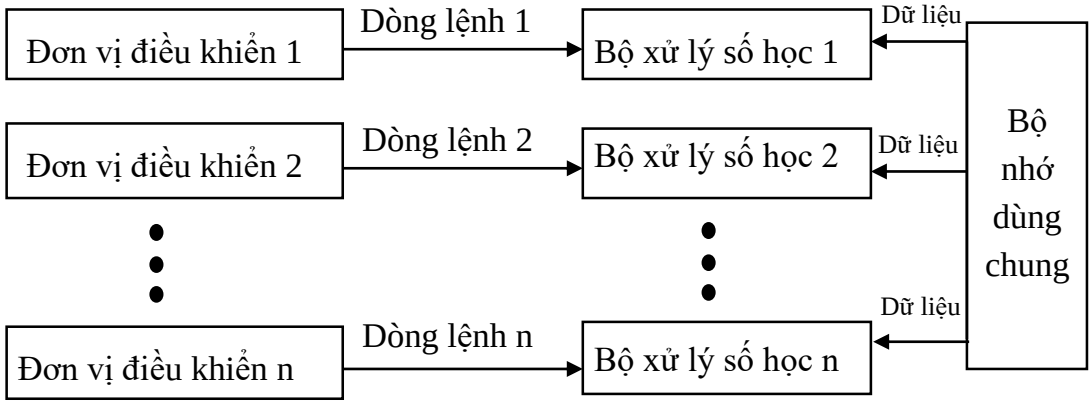
Hình 6.2 Mô hình máy tính SIMD

6.1.3 MISD (Multiple Instruction Stream single Data Stream)

Các máy MISD có thể thực hiện một số chương trình khác nhau trên cùng một đối tượng dữ liệu. Kiến trúc này gồm hai loại sau:

- Lớp các máy yêu cầu mỗi đơn vị xử lý nhận một lệnh riêng và thực hiện trên cùng một đối tượng dữ liệu.
- Lớp các máy mà trong đó dữ liệu lưu chuyển qua một chuỗi các bộ xử lý.

Các kiến trúc pipeline thuộc lớp này, nó làm việc theo nguyên lý pipelining. Nguyên lý này phân chia một tiến trình tính toán thành một số giai đoạn tách rời. Trong xử lý tuần tự, một tiến trình được bắt đầu thực thi chỉ khi tiến trình trước nó đã kết thúc hoàn toàn. Trong khi đó, sử dụng nguyên lý pipelining thì tiến trình ta có thể thực thi một số giai đoạn của hai tiến trình khác nhau, do đó có thể giảm thời gian xử lý.

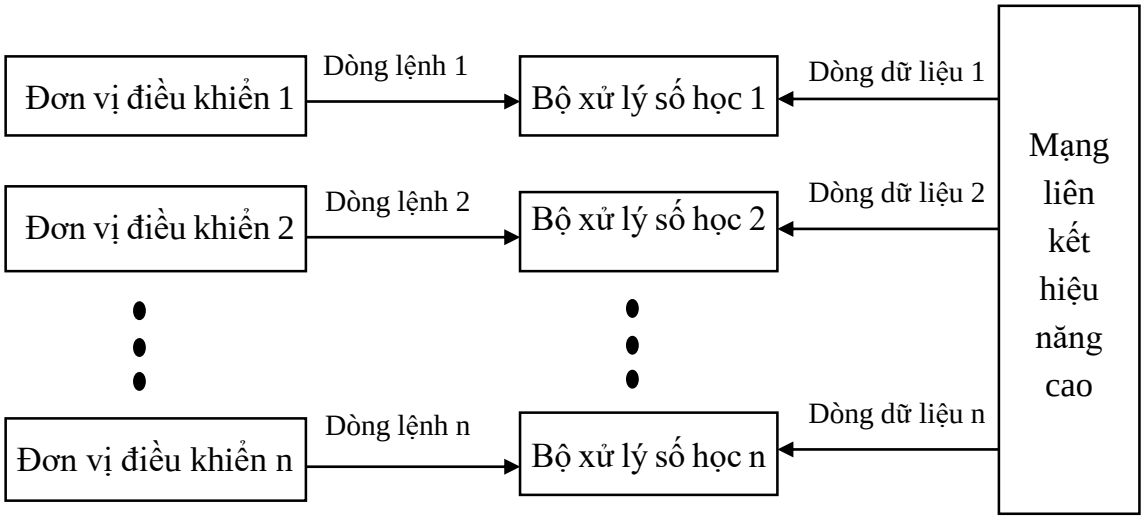


Hình 6.3 Mô hình máy tính MISD

6.1.4 MIMD (Multiple Instruction Stream multiple Data Stream)

MIMD là một hệ thống đa bộ xử lý hoặc đa máy tính.Mỗi bộ xử lý trong hệ thống này có một bộ điều khiển riêng và thực thi những chương trình của riêng chúng. Hệ thống MIMD có các đặc trưng sau:

- Chúng phân tán tiến trình xử lý cho một số các bộ xử lý độc lập.
- Tất cả các bộ xử lý chia sẻ tài nguyên được lưu trữ trong bộ nhớ chính.
- Các bộ xử lý hoạt động đồng thời và độc lập với nhau.
- Mỗi bộ xử lý chạy một chương trình riêng.



Hình 6.4 Mô hình máy tính MIMD

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)*

a) Một 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.

b) Trong hệ thống cluster, 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.

c) Một 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).

6.2 Kiến trúc RISC và CISC

Có thể phân kiến trúc máy tính ra hai lớp cơ bản: máy tính với tập hợp lệnh tối thiểu (RISC - Reduced Instruction Set Computer) và máy tính với tập hợp lệnh phức hợp (CISC - Complex Instruction Set Computer). Các máy tính trước kia thường có một số ít các lệnh đơn giản vì công nghệ ống chân không và bán dẫn lúc bấy giờ quá đắt, kềnh càng và phát nhiều nhiệt, chưa cho phép thiết kế các lệnh phức hợp. Khi công nghệ đạt được những tiến bộ nhất định, máy tính với kiến trúc tập hợp lệnh phức hợp (CISC) được thiết kế để có thêm nhiều lệnh và các lệnh càng ngày càng phức tạp. Chúng được thiết kế để hỗ trợ hệ điều hành và các trình biên dịch. Một lệnh thậm chí có thể thực hiện cả một chương trình con.

Ý tưởng cơ bản để thiết kế các máy tính này là nếu lệnh càng phức hợp thì tập hợp các lệnh cơ bản càng ít và như vậy ít lệnh sẽ được đọc từ bộ nhớ hơn và do đó làm tăng tốc độ xử lý của CPU. Tuy nhiên các lệnh phức hợp lại đòi hỏi thêm linh kiện điện tử, làm cho bộ vi xử lý kềnh càng hơn và cuối cùng lại làm chậm tốc độ của CPU. Trong khi đó nhiều lệnh phức hợp lại hầu như không được sử dụng trong thực tế. Vì thế lớp các máy tính có kiến trúc với tập hợp lệnh tối thiểu RISC xuất hiện với tư tưởng: chứa tập hợp một số ít các lệnh được chọn lọc kỹ lưỡng và các lệnh được thể hiện vật lý một cách đơn giản nhất có thể. Chương trình có thể sử dụng nhiều lệnh, nhưng hiệu suất tổng thể của máy tính tăng lên.

6.2.1 Kiến trúc RISC

(Reduced Instructions Set Computer - Máy tính với tập lệnh đơn giản hóa) là một phương pháp thiết kế các bộ vi xử lý theo hướng đơn giản hóa tập lệnh, trong đó thời gian thực thi tất cả các lệnh đều như nhau. Kiến trúc này có những đặc điểm cơ bản sau:

- Các lệnh đơn giản, một lệnh có một chu kỳ, không có lệnh kết hợp Load/store với số học.
- Nhấn mạnh trên phần mềm.
- Định dạng lệnh đơn giản, có độ dài cố định.

- Ít kiểu dữ liệu, RISC hỗ trợ một vài kiểu dữ liệu đơn giản một cách hiệu quả và các kiểu dữ liệu kết hợp/phức tạp được tổng hợp từ chúng.
- Thiết kế RISC sử dụng chế độ địa chỉ đơn giản và các lệnh có chiều dài cố định tạo điều kiện cho xử lý song song.
- Chú trọng các thao tác với các thanh ghi. Các thanh ghi mục đích chung giống nhau, RISC cho phép bất kỳ thanh ghi nào cũng có thể dùng trong bất kỳ ngữ cảnh nào, giúp cho việc đơn giản hóa thiết kế trình biên dịch.
- Song song hóa thuận tiện.

Đến năm 1986, tất cả các dự án về RISC bắt đầu cho ra đời sản phẩm. Ngày nay hầu hết các chip RISC, đều được thiết kế dựa trên kiến trúc RISC-II của Berkeley. Sun Microsystems với SPARC, hoặc Pyramid Technology. Chính Sun là công ty đầu tiên chứng minh sức mạnh của RISC là có thật trong những hệ thống mới của mình, và cũng nhờ đó họ nhanh chóng chiếm lĩnh thị trường workstation lúc bấy giờ. Ngày nay CPU RISC chiếm một lượng lớn CPU được sử dụng. Kỹ thuật thiết kế RISC đem đến sức mạnh ngay cả ở những kích thước nhỏ, do đó nó nhanh chóng chiếm lĩnh hoàn toàn thị trường CPU nhúng công suất thấp. Đây là một thị trường cực kỳ lớn của CPU, có thể tìm thấy chúng trong xe hơi, điện thoại di động… Người dùng thật ra chỉ quan tâm đến tốc độ, giá cả, và tính tương thích với các phần mềm có sẵn hơn là chi phí để phát triển những chip mới. Cùng với sự phức tạp của CPU tăng lên, chi phí thiết kế và sản xuất cũng tăng lên nhanh chóng. Lợi nhuận thu được từ RISC trở nên quá nhỏ bé so với chi phí đầu tư để phát triển các CPU mới, do đó ngày nay chỉ có những nhà sản xuất lớn mới có đủ khả năng phát triển những CPU mạnh. Kết quả là hầu hết những nền tảng RISC (ngoại trừ IBM POWER/PowerPC) đều thu hẹp quy mô (SPARC và MIPS).

Các hệ thống và các RISC phổ biến:

- Họ MIPS, trong các máy tính SGI, PlayStation và Nintendo 64 game consoles.
- Họ POWER trong các SuperComputers/mainframes của IBM.
- Freescale (trước đây là Motorola SPS) và IBM's PowerPC trong Nintendo Gamecube, Microsoft Xbox 360, Nintendo Wii and Sony PlayStation 3 game consoles, và cho tới gần đây là Apple Macintosh.
- SPARC và UltraSPARC, trong tất cả các hệ thống của Sun.
- Hewlett-Packard PA-RISC.
- DEC Alpha.
- ARM - Palm, Inc. Ban đầu sử dụng (CISC) Motorola 680x0 trong những PDA đầu tiên, nhưng hiện tại là (RISC) ARM; Nintendo sử dụng 1 chip ARM7 trong Game Boy Advance và Nintendo DS. Nhiều nhà sản xuất điện thoại di động, như Nokia cũng dựa trên kiến trúc của ARM. Các hãng di động đang sử chip của ARM như Nokia, Apple, Samsung, ...

6.2.2 Kiến trúc CISC

(Complex Instruction Set Computer - Máy tính với tập lệnh phức tạp). Kiến trúc này có những đặc điểm cơ bản sau:

- Tập lệnh lớn với nhiều lệnh phức tạp. Nguyên lý giải mã lệnh phức tạp do xuất phát từ nhu cầu một câu lệnh hỗ trợ nhiều chế độ địa chỉ.
- Đơn giản hóa trình dịch.
- Chương trình nhỏ và nhanh hơn.
- Số lượng các thanh ghi mục đích chung ít, các lệnh hoạt động trực tiếp trên bộ nhớ, không gian nhớ dùng làm các thanh ghi mục đích chung bị hạn chế. Có nhiều thanh ghi sử dụng với mục đích đặc biệt như con trỏ ngăn xếp, xử lý gián đoạn… Điều này tuy đơn giản hóa việc thiết kế phần cứng nhưng tiêu tốn chi phí do lệnh phức tạp hơn.
- Thông thường một câu lệnh tốn hơn một chu kỳ để xử lý, độ dài câu lệnh không cố định.
- Song song hóa phức tạp.

Một số bộ xử lý dựa trên kiến trúc CISC:

- IBM 370/168 được công bố năm 1970 với bộ xử lý 32 bit và thanh ghi dấu phẩy động 64 bit.
- VAX 11/780 bộ xử lý 32 bit, hỗ trợ nhiều chế độ định địa chỉ và mã máy.
- Intel 80486 được công bố vào năm 1989 với 235 câu lệnh.

Những nhà thiết kế vi xử lý cố gắng để mỗi lệnh có thể thực hiện càng nhiều chức năng càng tốt. Điều này dẫn đến một lệnh sẽ làm tất cả công việc. Cũng lệnh đó lại có thể đọc một số từ thanh ghi và số còn lại từ bộ nhớ sau đó lưu kết quả vào bộ nhớ. Khuynh hướng thiết kế vi xử lý này được gọi là Complex Instruction Set Computer - CISC. Điểm khác biệt thực sự giữa RISC so với CISC là nguyên tắc thực hiện mọi thứ trong các thanh ghi, đọc và lưu dữ liệu vào các thanh ghi. Điểm khác biệt thứ hai: đơn vị logic của một chip RISC bao giờ cũng cần ít transistor hơn so với của một chip CISC. Điều này giúp người thiết kế có rất nhiều sự linh hoạt như: tăng số lượng thanh ghi; sử dụng các phương pháp tối ưu để tăng mức độ xử lý song song bên trong CPU (pipeline, superscalar); tăng kích thước cache; thêm các tính năng như I/O, timer…; thêm các bộ xử lý vector; tận dụng các dây chuyền công nghệ cũ (trong khi với CISC điều này rất khó khăn do kích thước chip lớn hơn); cung cấp những chip cho những ứng dụng có yêu cầu cao về thời gian sử dụng pin hoặc về kích thước chip. Chip CISC được thiết kế nhằm tạo thuận lợi cho các nhà lập trình ứng dụng bằng cách rút gọn nhiều câu lệnh đơn giản, thông dụng thành một câu lệnh thực thi dài. Điều này làm cho CISC xử lý chậm hơn nhưng lại đạt yếu tố thân thiện. Ở mặt khác, RISC nhanh nhưng kém thân thiện hơn, mỗi câu lệnh đơn giản trong RISC phục vụ cho một mục đích hẹp rất cụ thể, thực hiện rất nhanh và các lệnh này được tiến hành song song. RISC đòi hỏi nhà lập trình phải kiên nhẫn, giỏi và một trình biên dịch được tối ưu kỹ lưỡng.

6.2.3 Sự khác biệt chính giữa RISC và CISC

- Trong RISC kích thước tập lệnh là nhỏ trong khi ở CISC kích thước tập lệnh là lớn.
- RISC sử dụng định dạng cố định (32 bit) và chủ yếu là các hướng dẫn dựa trên đăng ký trong khi CISC sử dụng định dạng biến trong phạm vi từ 16-64 bit cho mỗi lệnh.
- RISC sử dụng đồng hồ đơn và chế độ địa chỉ giới hạn. Mặt khác, CISC sử dụng nhiều chế độ địa chỉ và 12 đến 24 đồng hồ.
- Số lượng các thanh ghi mục đích chung mà RISC sử dụng nằm trong khoảng từ 32-192. Ngược lại, kiến trúc CISC sử dụng 8-24 GPR.
- Cơ chế bộ nhớ đăng ký để đăng ký được sử dụng trong RISC với các hướng dẫn LOAD và STORE độc lập. Ngược lại, CISC sử dụng bộ nhớ vào cơ chế bộ nhớ để thực hiện các hoạt động, hơn nữa, kết hợp các hướng dẫn LOAD và STORE.
- RISC đã phân chia dữ liệu và thiết kế bộ đệm hướng dẫn. Đối với, CISC sử dụng bộ đệm hợp nhất cho dữ liệu và hướng dẫn, mặc dù các thiết kế mới nhất cũng sử dụng bộ đệm tách.
- Hầu hết các điều khiển CPU trong RISC đều được gắn cứng mà không có bộ nhớ điều khiển. Ngược lại, CISC được mã hóa và sử dụng bộ nhớ điều khiển (ROM), nhưng CISC hiện đại cũng sử dụng điều khiển cứng.

STT	Cơ sở để so sánh	RISC	CISC
1	Nhấn mạnh về	Phần mềm	Phần cứng
2	Bao gồm	Đồng hồ đơn	Đồng hồ nhiều
3	Kích thước tập lệnh	Nhỏ bé	Lớn
4	Định dạng hướng dẫn	Định dạng cố định (32 bít)	Các định dạng khác nhau (16 - 64 bit mỗi lệnh)
5	Chế độ địa chỉ được sử dụng	Giới hạn 3 - 5 chế độ	12 - 24 chế độ
6	Thiết kế bộ nhớ cache	Tách bộ đệm dữ liệu và bộ đệm hướng dẫn	Bộ nhớ cache thống nhất cho hướng dẫn và dữ liệu
7	Chu kỳ theo hướng dẫn	Chu kỳ đơn cho tất cả các hướng dẫn, CPI trung bình < 1.5	CPI từ 2 - 15
8	Điều khiển CPU	Hardwired mà không kiểm soát bộ nhớ	Mã hóa sử dụng bộ nhớ điều khiển (ROM)

Bảng 6.1 Bảng so sánh kiến trúc RISC và CISC

6.3 Kiến trúc song song và mạng liên kết trong

6.3.1 Song song mức lệnh và song song mức luồng

6.3.1.1 Song song mức lệnh

Với kiến trúc song song mức lệnh số lượng song song có thể trong một khối cơ bản, các dòng lệnh tuần tự không có rẽ nhãnh ngoại trừ đầu vào và các rẽ nhánh ra. Đối với các chương trình MIPS bình thường thì tần suất các nhánh động thường từ 15% đến 25%, có nghĩa là từ bốn đến bảy lệnh được thực hiện giữa hai lệnh rẽ nhánh. Vì các lệnh này khá phụ thuộc vào các lệnh khác và số lượng gối lên nhau mà chúng ta có thể tận dụng trong một khối cơ bản có vẻ ít hơn nhiều so với kích thức trung bình của khối cơ bản. Để đạt được sự tăng hiệu năng đáng kể, chúng ta phải tận dụng ILP (Instruction Level Parallelism) trên nhiều khối cơ bản. Cách đơn giản và phổ biến nhất để tăng số lượng song song có thể giữa các lệnh là tận dụng song song giữa các lần lặp. Kiểu song song này gọi là song song mức lặp (loop-level). Đây là một ví dụ đơn giản về một vòng lặp dùng để cộng hai mảng 1000 phần tử và được thực hiện song song hoàn toàn: tất cả các bước lặp của vòng lặp có thể được gối lên nhau với các bước lặp khác mặc dùng trong một bước lặp các lệnh khó có thể gối lên nhau.

```
for (i=1; i<=1000; i=i+1)
        x[i] = x[i] + y[i];
```

Có một số kỹ thuật mà chúng ta sẽ xem xét để chuyển đổi song song mức lặp thành song song mức lệnh. Về cơ bản, các kỹ thuật này hoạt động bằng cách trải các vòng lặp ra hoặc theo cách tĩnh bởi chương trình dịch hoặc theo cách động bởi phần cứng. Một phương pháp thay thế quan trọng để tận dụng song song mức lặp là sử dụng các lệnh vector. Về cơ bản, một lệnh vector hoạt động dựa trên một chuỗi các mục dữ liệu. Các bộ xử lý có tận dụng ILP phần lớn đã thay thế các bộ xử lý dựa trên vector. Tuy nhiên, các tập lệnh vector có thể được sử dụng lại, ít nhất là trong các xử lý đồ họa, xử lý tín hiệu số, và các ứng dụng đa phương tiện.

6.3.1.2 Kiến trúc song song mức luồng

Với một bộ vi xử lý có nhiều luồng mã lệnh và nhiều luồng dữ liệu, mỗi một bộ vi xử lý có thể chạy các lệnh của chính nó. Trong nhiều trường hợp, các vi xử lý có thể chạy các tiến trình khác nhau (một tiến trình trên một bộ vi xử lý là một đoạn mã có thể chạy độc lập và các trạng thái của tiến trình có chứa toàn bộ các thông tin cần thiết cho việc chạy chương trình trên bộ vi xử lý đó). Trong một môi trường nhiều chương trình, nơi mà các bộ vi xử lý có thể đảm nhận các công việc của riêng nó, mỗi tiến trình trên một bộ vi xử lý là hoàn toàn độc lập với các tiến trình khác trên các bộ vi xử lý khác.

Chúng ta nhận thấy rằng thật là hữu ích nếu có thể chạy một chương trình trên một bộ đa xử lý, chia sẻ mã lệnh và hầu hết các không gian địa chỉ của nó. Khi có nhiều tiến trình cùng chia sẻ mã lệnh và dữ liệu theo kiểu này, chúng được gọi là các luồng. Ngày nay, thuật ngữ luồng thường được dùng để chỉ việc tiến hành chạy mã trên nhiều vùng, có

thể là trên nhiều vi xử lý khác nhau, ngay cả khi chúng không chia sẻ không gian địa chỉ. Để có thể tận dụng hiệu năng của một bộ đa xử lý có n vi xử lý, chúng ta cần phải có tối thiểu là n luồng hoặc n tiến trình chạy đồng thời. Các tiến trình độc lập có thể là do lập trình viên xác định tại thời điểm lập trình hoặc có thể được tạo lập bởi trình biên dịch. Vì rằng khái niệm song song trong trường hợp này được bao hàm trong các luồng nên được gọi là xử lý song song mức luồng.

Xử lý song song song cho phép các luồng có thể chia sẻ các đơn vị chức năng của một bộ vi xử lý đơn theo kiểu xếp chồng. Để có thể cho phép chia sẻ các đơn vị này, bộ vi xử lý cần thiết phải nhân đôi trạng thái độc lập của mỗi luồng. Ví dụ là phải có một bản copy riêng biệt cho file đăng ký, một PC riêng biệt và một bảng phân trang cho mỗi luồng. Bộ nhớ có thể được chia sẻ thông qua cơ chế bộ nhớ ảo, cơ chế này đã hỗ trợ lập trình đa luồng. Thêm vào đó, phần cứng phải hỗ trợ khả năng thay đổi tới các luồng khác nhau đủ nhanh; trong thực tế, cơ chế chuyển luồng phải nhanh hơn cơ chế chuyển tiến trình, và thông thường, cơ chế này đòi hỏi hàng trăm tới hàng ngàn chu kỳ vi xử lý.

Có hai cách tiếp cận tới xử lý đa luồng. Đó là: chuyển mạch đa luồng mịn giữa các luồng (Fine-grained multithreading switch) trên mỗi lệnh, làm cho việc chạy các luồng được tiến hành luân phiên. Việc chạy luân phiên này được thực hiện thông qua cơ chế xoay vòng, bỏ qua mọi luồng bị giữ tại thời điểm đó. Để có thể thực hiện đa luồng theo kiểu này, khối xử lý trung tâm phải có thể chuyển giữa các luồng trong mọi chu kỳ đồng hồ. Một lợi điểm quan trọng của chuyển mạch đa luồng mịn là nó thế thế che giấu thông lượng bị mất phát sinh do quá trình giữ các luồng quá nhanh hoặc quá lâu vì rằng các lệnh từ các luồng khác vẫn có thể chạy trong khi một luồng bị giữ lại. Nhược điểm chính của phương pháp này là nó làm chậm quá trình chạy các luồng riêng rẽ vì rằng mỗi luồng sắp chạy mà không bị giữ lại sẽ phải bị làm trễ bởi các lệnh từ các luồng khác. Một cách tiếp cận khác là xử lý đa luồng thô. Đây là giải pháp được phát minh như là một giải pháp thay thế cho phương pháp đa luồng mịn. Phương pháp đa luồng thô tiến hành chuyển các luồng chỉ khi luồng này bị giữ lại đủ lâu, chẳng hạn như là bị giữ lại ở cache L2. Việc này làm giảm thời gian trễ khi thực hiện các luồng độc lập vì rằng một luồng chỉ bị làm trễ khi mà nó bị giữ lại đủ lâu. Đa luồng thô có một nhược điểm chính là khả năng giảm thông lượng bị mất của nó luôn bị giới hạn, nhất là đối với các luồng có thời gian giữ lại ngắn.

6.3.2 Mạng liên kết trong

Các máy tính song song có hai thành phần chính. Đó là bộ nhớ và mạng liên kết trong. Trong phần này sẽ giới thiệu về một số mô hình của thành phần mạng liên kết trong bởi nó là chủ chốt trong hầu hết các kiến trúc song song. Mạng liên kết đó là sự kết nối giữa các bộ xử lý và các bộ nhớ trong một hệ thống song song. Với mỗi một mô hình mạng liên kết *M* có ba tiêu chuẩn đánh giá. Những tiêu chuẩn này liên quan đến việc truyền thông và độ phức tạp của mạng đó. Những tiêu chuẩn đó là:

- Đường kính của mạng**:**

Giả sử P_1 và P_2 là một cặp bộ xử lý bất kỳ trong mạng *M*. Và trong một thuật toán nào đó các bộ nhớ địa phương của P_1 và P_2 cần phải kết nối. Giả sử đường ngắn nhất kết nối giữa P_1 và P_2 là $Dist(P_1, P_2)$ - là số các kết nối trong đường đó. Khi đó số các bước truyền thông nhỏ nhất được yêu cầu để thông tin được chia sẻ bởi P_1 và P_2 là $Dist(P_1, P_2)/2$. Vì vậy đường kính của mạng được định nghĩa là.

$$D(M) := Max \{ Dist(P_i, P_j): P_i, P_j \in M \}$$

- *Bậc của bộ xử lý:* là số các liên kết tới bộ xử lý đó.
- Thông lượng giữa hai tập hợp các bộ xử lý của mạng**.**

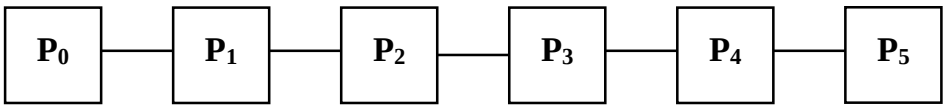
Chia các bộ xử lý của mạng *M* thành hai tập *X* và *Y*. Giả sử *All-Link(X, Y)* là tất cả các liên kết nối một bộ xử lý trong tập *X* tới một bộ xử lý trong tập *Y*. Kích thước của liên kết nhỏ nhất trong *All-Link(X, Y)* được định nghĩa là thông lượng giữa hai phần của mạng. Vì thế tiêu chuẩn thứ ba là:

$$Bisection\text{-}Width(M) := Min\{ \lceil All\text{-}Link(X, Y) \rceil : Abs(\lceil X \rceil - \lceil Y \rceil) > 1 \}$$

Một mạng liên kết *M* là tốt nếu nó có đường kính nhỏ, bậc lớn nhất của các bộ xử lý trong mạng nhỏ và thông lượng giữa hai tập bất kỳ của mạng lớn. Tuy nhiên, trên thực tế không có một mạng liên kết nào có thể đáp ứng đủ ba tiêu chuẩn trên. Có hai nhóm mạng liên kết là: tĩnh và động. Những mạng liên kết tĩnh cung cấp những kết nối cố định giữa các thành phần (các bộ nhớ và các bộ xử lý). Ngược lại, một mạng liên kết động có thể cấu hình lại. Trong loại mạng liên kết này, sự kết nối có thể thiết lập bằng cách tạo ra một tập hợp các hộp chuyển mạch. Sau đây là một số mô hình mạng liên kết cụ thể của hai nhóm mạng trên và đặc điểm của từng mô hình. Mỗi mạng liên kết được thể hiện bởi một đồ thị, trong đó các đỉnh biểu diễn các bộ xử lý và các cạnh biểu diễn cho đường truyền thông giữa từng cặp bộ xử lý.

6.3.2.1 Mạng liên kết tuyến tính (Linear)

Trong mô hình này các bộ xử lý được sắp xếp theo thứ tự tăng dần từ *0* đến *p-1*. Trừ bộ xử lý đầu tiên và cuối cùng, mỗi bộ xử lý đều có hai láng giềng là bộ xử lý liền trước nó và liền sau nó. Mô hình này được dùng phổ biến trong giới những công việc mang tính lý thuyết và đơn giản.

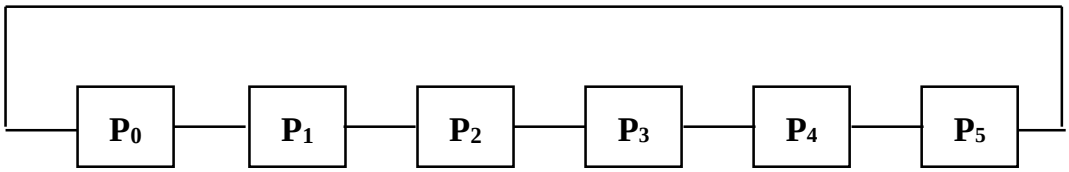


Hình 6.5 Mạng liên kết tuyến tính của 6 bộ xử lý

Đặc điểm: Cấu trúc đơn giản, nhưng dữ liệu phải truyền qua một số bộ xử lý mới đến được đích, gây ra tình trạng trì hoãn và có thể là tắc nghẽn trong liên lạc dài, đặc biệt là bộ xử lý đầu tiên và cuối cùng. Mạng liên kết tuyến tính với *N* bộ xử lý có bậc là 2 và đường kính là *O(N)*

6.3.2.2 Mạng liên kết dạng vòng (Ring)

Mạng liên kết dạng vòng có thể được tổ chức bằng cách nối bộ xử lý đầu tiên với bộ xử lý cuối cùng với nhau. Liên kết trong một vòng có thể theo một hướng duy nhất hoặc theo cả hai hướng. Nghĩa là việc liên lạc trong cấu trúc vòng có thể được thiết lập theo một hướng hoặc cả hai hướng.



Hình 6.6 Mạng liên kết vòng với 6 bộ xử lý

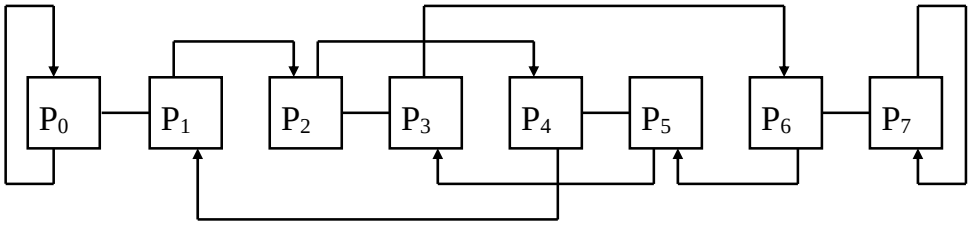
Đặc điểm: cấu trúc vòng có thể vẫn gây sự trì hoãn trong liên lạc dài giữa các thành phần. Mạng liên kết dạng vòng có bậc và đường kính lần lượt là 2 và O(*N*).

6.3.2.3 Mạng liên kết trao đổi phi tuyến (Shuffle Exchange)

Giả sử có *N* bộ xử lý $P_0, P_2, \ldots, P_{N-1}$ với *N* là một luỹ thừa của 2. Trong một mạng liên kết trao đổi phi tuyến hoàn chỉnh, từ bộ xử lý P_i sẽ có một đường truyền thông một chiều tới bộ xử lý P_j với điều kiện:

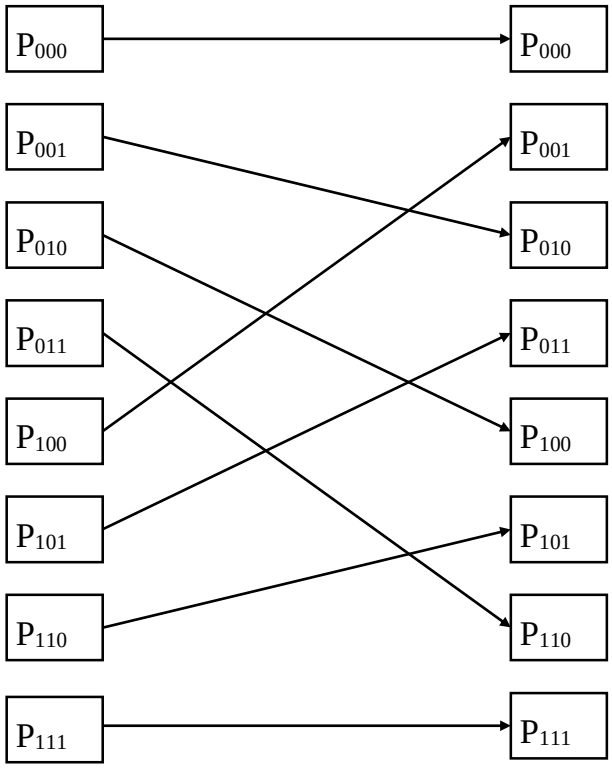
$$j = \begin{cases} 2i & \text{Nếu } 0 \le i \le N/2 - 1 \\ 2i + 1 - N & \text{Nếu } N/2 \le i \le N - 1 \end{cases}$$

Hình sau mô tả một mạng liên kết phi tuyến hoàn chỉnh với *N*=8 trong đó các kết nối phi tuyến được biểu diễn bởi các đường mũi tên nét liền, còn các kết nối trao đổi được biểu diễn bởi các đường nét liền. Nói chung mạng liên kết phi tuyến hoàn chỉnh sẽ kết nối nút *i* với nút *2i* mod *(N*-1), ngoại trừ nút *N*-1 được nối với chính nó.



Hình 6.7 Mạng liên kết phi tuyến với N=8

Một cách khác để biểu diễn mạng liên kết phi tuyến đó là biểu diễn mỗi bộ xử lý bằng số nhị phân, và bộ xử lý P_i sẽ được kết nối với bộ xử lý P_j nếu biểu diễn nhị phân của *j* thu được bằng cách dịch trái 1 bit của biểu diễn nhị phân của *i*. Ví dụ bộ xử lý P_{010} được kết nối với bộ xử lý P_{100} vì 100 thu được bằng cách dịch trái một bit từ 010. Cách biểu diễn này được thể hiện trong hình sau



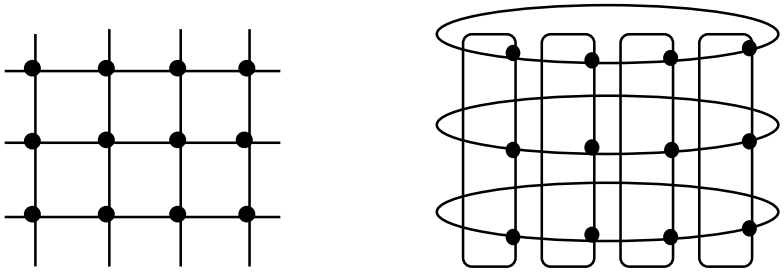
Hình 6.8 Cách biểu diễn khác của mạng liên kết phi tuyến

Đặc điểm: mạng liên kết dạng trao đổi phi tuyến có bậc là 3 và đường kính là *O(log N).*

6.3.2.4 Mạng liên kết lưới hai chiều (Two-Dimensional Mesh)

Trong mạng liên kết dạng này các bộ xử lý được sắp xếp thành một ma trận hai chiều. Mỗi bộ xử lý được kết nối với bốn láng giềng của nó (trên, dưới, trái, phải). Không có một quy luật chung cho những kết nối ở biên. Nó tuỳ thuộc vào từng cấu trúc song song. Trong cùng một cột thì bộ xử lý cuối cùng sẽ được kết nối với bộ xử lý đầu tiên, những bộ xử lý ở lề phải của một hàng sẽ được kết nối với bộ xử lý ở lề trái của hàng tiếp theo. Một số biến thể của mạng lưới hai chiều cho phép những kết nối bao quanh giữa các bộ xử lý trên các cạnh của lưới, ví dụ giữa các bộ xử lý trên cùng một hàng hoặc một cột.

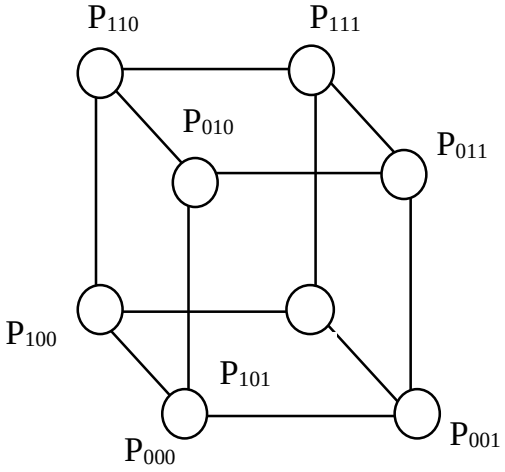
Đặc điểm: cấu trúc lưới có thể tổng quát lên nhiều hơn hai chiều. Trong một lưới *q* chiều, mỗi bộ xử lý được kết nối với hai bộ xử lý láng giềng trong mỗi chiều, còn các bộ xử lý ở biên thì có ít kết nối hơn. Bậc và đường kính của mạng liên kết lưới hai chiều lần lượt là 4 và $O(N^{1/2})$.



Hình 6.9 Lưới hai chiều không có kết nối bao quanh và có kết nối bao quanh

6.3.2.5 Mạng liên kết siêu khối (hypercube or n-cube)

Giả sử có N bộ xử lý với N là một luỹ thừa của 2: $N=2^q$, $q \geq 0$. Nếu mỗi bộ xử lý được kết nối với đúng q láng giềng thì ta sẽ thu được một siêu khối q chiều, nghĩa là mỗi bộ xử lý có bậc là q. Một siêu khối q chiều có thể thu được bằng cách kết nối các bộ xử lý tương ứng trong hai siêu khối $q/2$ chiều. Theo hình thức, mạng liên kết N bộ xử lý được gọi là mạng siêu khối nhị phân, sao cho N bộ xử lý được gắn nhãn là các chuỗi nhị phân có giá trị từ 0 tới N-1, và hai bộ xử lý được gọi là kề nhau (kết nối với nhau) nếu các nhãn của chúng chỉ khác nhau đúng một vị trí bit, tức khác nhau một luỹ thừa của hai. Hình sau mô tả hai mạng liên kết siêu khối với 2^3=8 và 2^4 = 16 bộ xử lý.



Hình 6.10 Mạng liên kết siêu khối với 8 bộ xử lý.

Đặc điểm: hypercube là kiểu tổ chức bộ xử lý được dùng phổ biến trong một số công ty như Intel Corporation, NCUBE Corporation, Thinking Machine Corporation và FPS. Bậc và đường kính của mô hình mạng liên kết siêu khối đều bằng *O(log N*).

6.4 Một số kiến trúc tương lai

6.4.1 Kiến trúc IA - 64

Kiến trúc IA - 64 là một kiến trúc mới được giới thiệu trong những năm gần đây. Kiến trúc này là sản phẩm của sự kết hợp nghiên cứu giữa hai công ty máy tính hàng đầu thế giới là Intel, HP (Hewlett Packard) và một số trường đại học. Kiến trúc mới dựa trên sự phát triển của công nghệ mạch tích hợp và kỹ thuật xử lý song song. Kiến trúc IA-64 giới thiệu một sự khởi đầu mới quan trọng của kỹ thuật siêu vô hướng - kỹ thuật xử lệnh song song (EPIC: Expicitly Parallel Intruction Computing) - kỹ thuật ảnh hưởng nhiều đến sự phát triển của bộ xử lý hiện nay. Sản phẩm đầu tiên thuộc kiến trúc này bộ xử lý *Itanium*.

a) Đặc trưng của kiến trúc IA-64

- Cơ chế xử lý song song là song song các lệnh mã máy (EPIC) thay vì các bộ xử lý song song như hệ thống đa bộ xử lý.
- Các lệnh dài hay rất dài (LIW hay VLIW).
- Các lệnh rẽ nhánh xác định (thay vì đoán các lệnh rẽ nhánh như các kiến trúc trước).

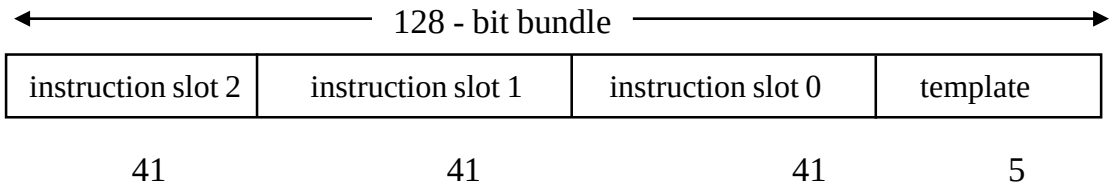
- Nạp trước các lệnh (theo sự suy đoán).

Các đặc trưng của tổ chức của bộ xử lý theo kiến trúc IA-64:

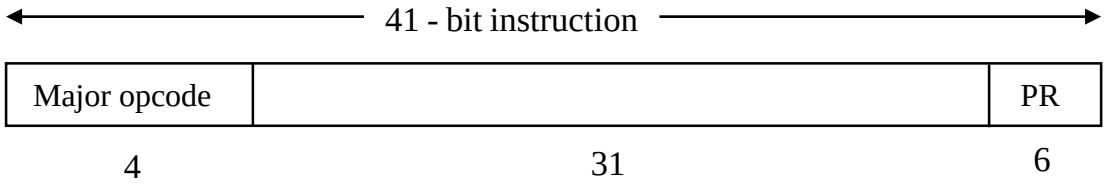
- Có nhiều thanh ghi: số lượng thanh ghi các bộ xử lý kiến trúc IA-64 là 256 thanh ghi. Trong đó, 128 thanh ghi tổng quát (GR) 64 bit cho các tính toán số nguyên, luận lý; 128 thanh ghi 82 bit (FR) cho các phép tính dấu chấm động và dữ liệu đồ hoạ; ngoài ra, còn có 64 thanh ghi thuộc tính (PR)1 bit để chỉ ra các thuộc tính lệnh đang thi hành.
- Nhiều bộ thi hành lệnh: hiện nay, một máy tính có thể có tám hay nhiều hơn các bộ thi hành lệnh song song. Các bộ thi hành lệnh này được chia thành bốn kiểu:
    - Kiểu I (I-Unit): dùng xử lý các lệnh tính toán số nguyên, dịch, luận lý, so sánh, đa phương tiện.
    - Kiểu M (M-Unit): Nạp và lưu trữ giữa thanh ghi và bộ nhớ thêm vào một vài tác vụ ALU.
    - Kiểu B (B-Unit): Thực hiện các lệnh rẽ nhánh.
    - Kiểu F (F-Unit): Các lệnh tính toán số dấu chấm động

b) Định dạng lệnh trong kiến trúc IA-64

Kiến trúc IA-64 định nghĩa một gói (buldle) 128 bit chứa ba lệnh (mỗi lệnh dài 41 bit) và một trường mẫu (template field) 5 bit. Bộ xử lý có thể lấy một hay nhiều gói lệnh thi hành cùng lúc. Trường mẫu (template field) này chứa các thông tin chỉ ra các lệnh có thể thực hiện song song . Các lệnh trong một bó có thể là các lệnh độc lập nhau. Bộ biên dịch sẽ sắp xếp lại các lệnh trong các gói lệnh kề nhau theo một thứ tự để các lệnh có thể được thực hiện song song. Trong một lệnh, mã lệnh chỉ có 4 bit chỉ ra 16 khả năng có thể để thi thi hành một lệnh và 6 bit chỉ ra thanh ghi thuộc tính được dùng với lệnh. Tuy nhiên, các mã tác vụ này còn tuỳ thuộc vào vị trí của lệnh bên trong gói lệnh, vì vậy khả năng thi hành của lệnh nhiều hơn số mã tác vụ được chỉ ra.



Hình 6.11 Định dạng lệnh trong kiến trúc IA - 64



Hình 6.12 Dạng tổng quát của một lệnh trong gói lệnh

Major opcode	other modifying bits	GR 3	GR 2	GR 1	PR
4	10	7	7	7	6

Hình 6.13 Mô tả các trường trong một lệnh (41 bit).

PR: Predicate register. GR: General hay Floating-point

6.4.2 Kiến trúc mạng nơ-ron

Mạng nơ-ron AI- Artifician Neutral Network - Máy tính dưới dạng nơ-ron nhân tạo. Thông thường, một mạng nơ-ron bao gồm một hoặc nhiều nhóm các nơ-ron được kết nối vật lý với nhau hoặc có liên quan với nhau về chức năng. Một nơ-ron đơn có thể được nối với nhiều nơ-ron khác và tổng số nơ-ron và kết nối trong một mạng có thể là một giá trị cực kỳ lớn. Các kết nối, gọi là các khớp thần kinh (synapses), thường nối từ các axon tới các tế bào tua gia thần kinh (dendrite), tuy có thể có các vi mạch dendrodentritic và các kết nối khác. Ngoài tín hiệu điện, còn có các dạng tín hiệu khác phát sinh từ việc khuyếch tán các chất dẫn chuyền xung động thần kinh (neurotransmitter). Chúng có ảnh hưởng đối với tín hiệu điện. Do vậy, cũng như các mạng sinh học khác, mạng nơ-ron vô cùng phức tạp. Trong khi hiện nay, dù chưa đạt được một mô tả chi tiết nào về hệ thần kinh, người ta vẫn ngày càng hiểu rõ hơn về các cơ chế cơ bản.

6.4.3 Kiến trúc Nehalem

Đại diện cho kiến trúc Nehalem chính là hai bộ xử lý điển hình Intel Core i7 965 và Core i7 920. Bên cạnh hiệu năng cao, hai bộ xử lý này còn cung cấp cho người dùng những công nghệ mới mẻ và hữu ích, chúng có cache L2 riêng cho mỗi nhân (dung lượng 256KB, độ trễ thấp). Ngoài ra, các nhân xử lý còn thực hiện việc lấy lệnh và trao đổi dữ liệu trên cache L3 dung lượng cho phép lên đến 8MB. Kiến trúc Nehalem, Intel đã thiết kế hai tuyến bus riêng biệt. Cụ thể, để "liên kết" đến thiết bị ngoại vi như card đồ họa, đĩa cứng... hai BXL Intel Core i7 965 và 920 được cung cấp tuyến bus Intel QuickPath Interconnect (Intel QPI) với tốc độ giao tiếp có thể đạt đến lần lượt là 6,4GT/s (Core i7 965) và 4,8GT/s (Core i7 920). Việc giao tiếp giữa bộ nhớ và chip điều khiển bộ nhớ tích hợp (chỉ hỗ trợ loại DDR3) trong bộ xử lý được thực hiện trên một tuyến bus hoàn toàn độc lập với Intel QPI.

Ngoài ra, kiến trúc Nehalem cũng chuẩn bị cho người dùng một "bộ sưu tập" những công nghệ hữu ích. Đầu tiên, bạn sẽ được gặp lại công nghệ siêu phân luồng (Hyper Threading) đã quen thuộc với người dùng từ thời những bộ xử lý Intel Pentium. Với công nghệ này, một nhân trong bộ xử lý có thể hoạt động đồng thời với hai luồng xử lý, nghĩa là hai bộ xử lý nền Nehalem bốn nhân có thể cung cấp cho người dùng 8 luồng xử lý đồng thời để giải quyết các ứng dụng. Để thực hiện yêu cầu tiết kiệm năng lượng, Intel đã đưa vào bộ xử lý của mình công nghệ Power Gates giúp loại bỏ dòng điện rò trên các nhân đang được đặt ở trạng thái nghỉ và đưa điện áp trên đó về mức gần như bằng 0. Bên cạnh đó, Intel cũng tích hợp vào những bộ xử lý nền Nehalem một chip điều khiển năng lượng PCU (Power Control Unit) để sử dụng hiệu quả điện trên các nhân. Đây chính là điểm

quyết định cho công nghệ Turbo Boost khi PCU có thể điều khiển các thành phần điện năng, xung nhịp và các cảm ứng trạng thái ở mỗi nhân một cách độc lập. Khi yêu cầu của tải công việc cao hơn nhưng chưa tới mức cần phải "bật" thêm nhân mới, các nhân xử lý đang hoạt động có thể được "đẩy" xung nhịp lên mức cao hơn để gánh vác phần việc này.

Do đó, các nhân trong những bộ xử lý Nehalem có thể sẽ hoạt động tại các mức xung nhịp khác nhau. Một số điểm mới khác trên dòng bộ xử lý nền Nehalem cũng cần đề cập đến là tập lệnh đa phương tiện mới SSE4.2 với thêm 7 lệnh bổ sung giúp cải thiện khả năng xử lý văn bản, chuỗi hay một số ứng dụng đặc trưng; cải tiến công nghệ ảo hóa (Intel Vitualization) để các máy ảo có thể hoạt động hiệu quả hơn... Sau sự xuất hiện của những bộ xử lý kiến trúc Core có hiệu năng khá cao, kiến trúc Nehalem mới của Intel tiếp tục đem đến cho người dùng thế hệ bộ xử lý tiếp theo với hiệu năng được cải thiện đáng kể, minh chứng là kết quả thử nghiệm vượt trội so với những bộ xử lý theo nền kiến trúc Core cũ. Bên cạnh đó, những bộ xử lý nền Nehalem cũng đem đến cho người dùng những công nghệ tiết kiệm điện năng, ảo hóa hay những cải tiến trong mặt thiết kế. Nhờ đó, cùng với một bo mạch chủ chipset Intel X58, bộ xử lý nền Nehalem sẽ góp phần đem đến cho người dùng những hệ thống máy tính có ưu thế lớn về sức mạnh xử lý và độ ổn định, đáp ứng tốt nhất yêu cầu làm việc và giải trí của người dùng.

6.4.4 Kiến trúc máy tính lượng tử

Các nhà khoa học phát triển kỹ thuật mới này khai thác hành vi của điện tử, tập trung vào spin của nó thay vì điện tích. Electron, hay còn gọi là điện tử là một hạt sơ cấp có vai trò vô cùng quan trọng. Bạn có biết tất cả các thiết bị điện tử ngày nay, theo cái tên của chúng, đều khai thác tính chất điện của electron. Chúng ta sản xuất điện, chế tạo bóng đèn, điện thoại, máy tính đều dựa vào điều đó. Nhưng electron bản thân chúng không chỉ có tính chất điện. Khai thác các đặc trưng còn lại của điện tử sẽ mở ra nhiều hướng ứng dụng mới vô cùng tuyệt vời. Đó chính là điều mà một nhóm các nhà khoa học quốc tế tại Phòng thí nghiệm Berkeley, Hoa Kì đã làm. Họ phát triển một kỹ thuật mới để khai thác hành vi của điện tử, tập trung vào spin của nó thay vì điện tích. Điều này hứa hẹn mở ra một kiến trúc máy tính và truyền thông tin lượng tử mới. Trong khi gây sốc một vật liệu silic đặc thù bằng vi sóng, các nhà nghiên cứu nhận ra spin điện tử của chúng nhanh chóng bị thay đổi. Nó sẽ nhanh chóng chuyển từ trạng thái kích thích xuống trạng thái cơ bản, đồng thời phát ra một photon ánh sáng. Spin là một khái niệm vật lý đặc trưng cơ bản của điện tử. Nó chỉ có thể nhận một trong hai giá trị.

Vì vậy, có thể hình dung spin giống như một đồng xu hoặc sấp hoặc ngửa. Đặc trưng này dẫn đến ý tưởng sử dụng spin để miên tả các bit 0 và 1 trong máy tính. Đây là một điều rất thú vị. Hãy thử tưởng tượng một điện tử như một đồng xu. Máy tính ngày nay miêu tả 1 bit dữ liệu bằng cách đóng mở dòng điện trong các transistor siêu nhỏ. Mặc dù vậy, nó vẫn giống như bạn phải ném hàng ngàn đồng xu qua một khoảng cách cực lớn chỉ để mô tả 1 bit. Với spin, bạn đơn thuần là lật tại chỗ hai mặt của đồng xu để nó thể hiện các bit khác nhau. Mặc dù vậy, việc lật một đồng xu spin là không đơn giản. Hiệu ứng này chỉ xảy

ra trong tự nhiên mỗi 10.000 năm. Tuy nhiên, bằng nghiên cứu mới này, các nhà khoa học có thể khiến việc này diễn ra chỉ trong 1 giây. “Nó giống như một nghệ sĩ tung hứng ném quả bóng lên trời và nó rơi xuống với tốc độ nhanh gấp 1.000 lần. Cùng với đó phát ra một luồng ánh sáng”, Thomas Schenkel, nhà vật lý làm việc tại Bộ phận Công nghệ gia tốc và Ứng dụng vật lý thuộc Berkeley Lab cho biết. “Kết quả này của chúng tôi rất quan trọng cho việc xử lí thông tin lượng tử”, Patrice Bertet, người dẫn đầu thí nghiệm tại Ủy ban Năng lượng nguyên tử Pháp (CEA) cho biết. “Thật vậy, đây sẽ là nền móng hướng tới sự kết hợp mạnh mẽ của spin điện tử với photon, mà sẽ hình thành cơ sở một kiến trúc máy tính lượng tử mới”.

Trong các máy tính ngày nay, thông tin được lưu trữ trong các bit riêng lẻ. Mỗi bit được gán giá trị 0 hoặc 1. Máy tính lượng tử được hứa hẹn là sẽ mạnh mẽ hơn gấp nhiều lần, bởi nó sử dụng một loại khác của bit có tên qubit. Một qubit có thể cùng lúc nhận hai giá trị 0 và 1, theo một hiệu ứng kì lạ của cơ học lượng tử. Nếu xây dựng được một mảng liên kết các qubit, máy tính lượng tử có khả năng thực hiện rất rất nhiều tính toán cùng lúc. Nghiên cứu mới đã chỉ ra rằng trong khi spin của điện tử có thể đóng vai trò một qubit, photon vi sóng phát ra có thể làm việc như một cách thức truyền thông tin. Kết hợp lại, chúng ta sẽ có một kiến trúc máy tính lượng tử mới. “Những gì chúng tôi cần làm bây giờ là kết nối các spin với nhau”, Morton nói. “Chúng tôi cần phải kết cặp các qubit với nhau nếu muốn thực hiện tính toán”. Để làm được điều này, các nhà nghiên cứu đã thực hiện thí nghiệm tại cơ sở CEA, Pháp. Họ sử dụng những tinh thể siêu tinh khiết của silic, sau đó pha tạp với một số nguyên tử bismuth. Schenkel mô tả công việc này giống như bạn đang cố ép những quả bóng bowling vào một mạng lưới bóng bàn vậy. Sau đó, một mạch siêu dẫn bằng nhôm được tạo ra dể hình thành khoang cộng hưởng cho phép điều chỉnh chính xác vi sóng phát ra. Đồng xu spin của các điện tử bismuth lúc này đều ở trạng thái lật. Khi quá trình phát vi sóng bắt đầu, khoang cộng hưởng nhôm được tinh chỉnh sao cho tới một trạng thái nào đó, các đồng xu spin đều bị lật úp trở lại, đồng thời phát ra một photon ánh sáng. Các khoa học cho biết nghiên cứu mới này của họ không chỉ có ý nghĩa với việc phát triển máy tính lượng tử. Hiệu ứng còn giúp nâng cao chất lượng của kỹ thuật chụp cộng hưởng từ MRI, ứng dụng trong y tế, nghiên cứu cấu trúc vật liệu và phân tử sinh học. Bertet nói nhóm nghiên cứu sẽ tiếp tục rút ngắn thời gian lật những spin này xuống cỡ một mili giây. Khi đó, “điều này sẽ mở đường cho nhiều ứng dụng mới hơn nữa”.

CÂU HỎI ÔN TẬP VÀ BÀI TẬP CHƯƠNG 6

*****

Câu hỏi hướng dẫn ôn tập, thảo luận

Câu 1. Phân loại các kiến trúc song song theo Mycheal Flynn?

Câu 2. Vẽ sơ đồ của kiến trúc Single Instruction Stream, Single Data Stream?

Câu 3. Vẽ sơ đồ của kiến trúc Single Instruction Stream, Multiple Data Stream?

Câu 4. Vẽ sơ đồ của kiến trúc Multiple Instruction Stream, Single Data Stream?

Câu 5. Vẽ sơ đồ của kiến trúc Multiple Instruction Stream, Multiple Data Stream?

Câu 6. Trình bày về kiến trúc song song mức lệnh và song song mức luồng?

Câu 7. Các ví dụ về siêu máy tính dùng kỹ thuật xử lý song song?

Câu 8. Kiến trúc RISC là gì? Phân tích đặc trưng của máy tính dùng kiến trúc RISC?

Câu 9. Kiến trúc CISC là gì? Phân tích đặc trưng của máy tính dùng kiến trúc CISC?

Câu 10. So sánh những đặc tính khác nhau cơ bản của kiến trúc RISC và CISC?

Câu 11. Mạng liên kết trong có những dạng nào? Nêu đặc trưng điển hình của mỗi dạng?

Câu 12. Trình bày một số kiến trúc trong tương lai mà em biết?

Câu hỏi trắc nghiệm

Câu 1. Có mấy mô hình kiến trúc máy tính dựa vào sự phân loại của Flynn

A. 1 B. 2 C. 4 D. 3

Câu 2. Mô hình SISD có mấy đơn vị điều khiển tín hiệu

A. 1 B. 2 C. n D. 3

Câu 3. Mô hình SISD có mấy đơn vị xử lý số học

A. 1 B. 2 C. 3 D. n

Câu 4. Mô hình SIMD có mấy đơn vịđiều khiển

A. 2 B. 1 C. 3 D. n

Câu 5. Mô hình SIMD có mấy đơn vị xử lý số học

A. 2 B. 1 C. 3 D. n

Câu 6. Mô hình MISD có mấy đơn vị điều khiển

A. 2 B. 1 C. 3 D. n

Câu 7. Mô hình MISD có mấy đơn vị xử lý số học

A. 2 B. 1 C. 3 D. n

Câu 8. Mô hình MIMD có mấy đơn vị điều khiển

A. 2 B. 1 C. n D.3

Câu 9. Mô hình MIMD có mấy đơn vị xử lý số học

A. n B. 1 C. 2 D. 3

Câu 10. Mục đích của xử lý song song là

A. Làm giảm thời gian tính toán của hệ thống

B. Giải quyết các bài toán có kích thước dữ liệu lớn

C. Giải quyết các bài toán có độ phức tạp cao

D. Làm giảm thời gian tính toán của hệ thống, giải quyết các bài toán có kích thước dữ liệu lớn, giải quyết các bài toán có độ phức tạp cao.

Câu 11. Để tăng cường hiệu quả của máy tính, người ta đã nghĩ tới giải pháp song song bằng cách:

A. Tăng số lượng bộ nhớ

B. Tăng tốc độ xử lý

C. Tăng số lượng bộ xử lý

D. Tăng thiết bị vào ra

Câu 12. SIMD thuộc loại máy tính:

A. Một dòng lệnh, một dòng số liệu

B. Nhiều dòng lệnh, một dòng số liệu

C. Một dòng lệnh, nhiều dòng số liệu

D. Nhiều dòng lệnh, nhiều dòng số liệu.

Câu 13. MIMD thuộc loại máy tính:

A. Nhiều dòng lệnh, nhiều dòng số liệu

B. Một dòng lệnh, nhiều dòng số liệu

C. Nhiều dòng lệnh, một dòng số liệu

D. Một dòng lệnh, một dòng số liệu.

Câu 14. MISD thuộc loại máy tính:

A. Nhiều dòng lệnh, nhiều dòng số liệu

B. Một dòng lệnh, nhiều dòng số liệu

C. Nhiều dòng lệnh, một dòng số liệu

D. Một dòng lệnh, một dòng số liệu.

Câu 15. SISD thuộc loại máy tính:

A. Nhiều dòng lệnh, nhiều dòng số liệu

B. Một dòng lệnh, nhiều dòng số liệu

C. Nhiều dòng lệnh, một dòng số liệu.

D. Một dòng lệnh, một dòng số liệu.

TÀI LIỆU THAM KHẢO

[1] Tiến sĩ Nguyễn Kim Khánh: bài giảng Kiến trúc máy tính trường Đại học Bách Khoa Hà Nội. 2015

[2] Nguyễn Đình Việt: Kiến trúc máy tính. Nhà xuất bản Đại học Quốc Gia Hà Nội. 2010

[3] Tiến sĩ Trần Công Hùng: Kiến trúc máy tính tiên tiến. Nhà xuất bản Thông tin và truyền thông. 2011

[4] Báo cáo Cấu trúc và bảo trì hệ thống - ĐH Kỹ thuật hậu cần CAND

[5] John L. Henness & David A. Patterson. Computer Architecture. A Quantitative Approach. 2006 (4th edition).

[6] William Stallings. Computer Organization and Architecture. Designing for Performance - 2009 (6th edition).

[7] David A. Patterson & Jonhn L. Hennessy. Computer Organization and design: The hardware/Software Interface. 2005 (3th edition).Patterson, Hennessy - Computer Organization and Design; The Hardware-Software Interface, 2E (Morgan Kaufman, 1997).pdf

[8] David Tarnoff: Computer Organization and Design Fundamentals. 2007. Revised First Edition.pdf