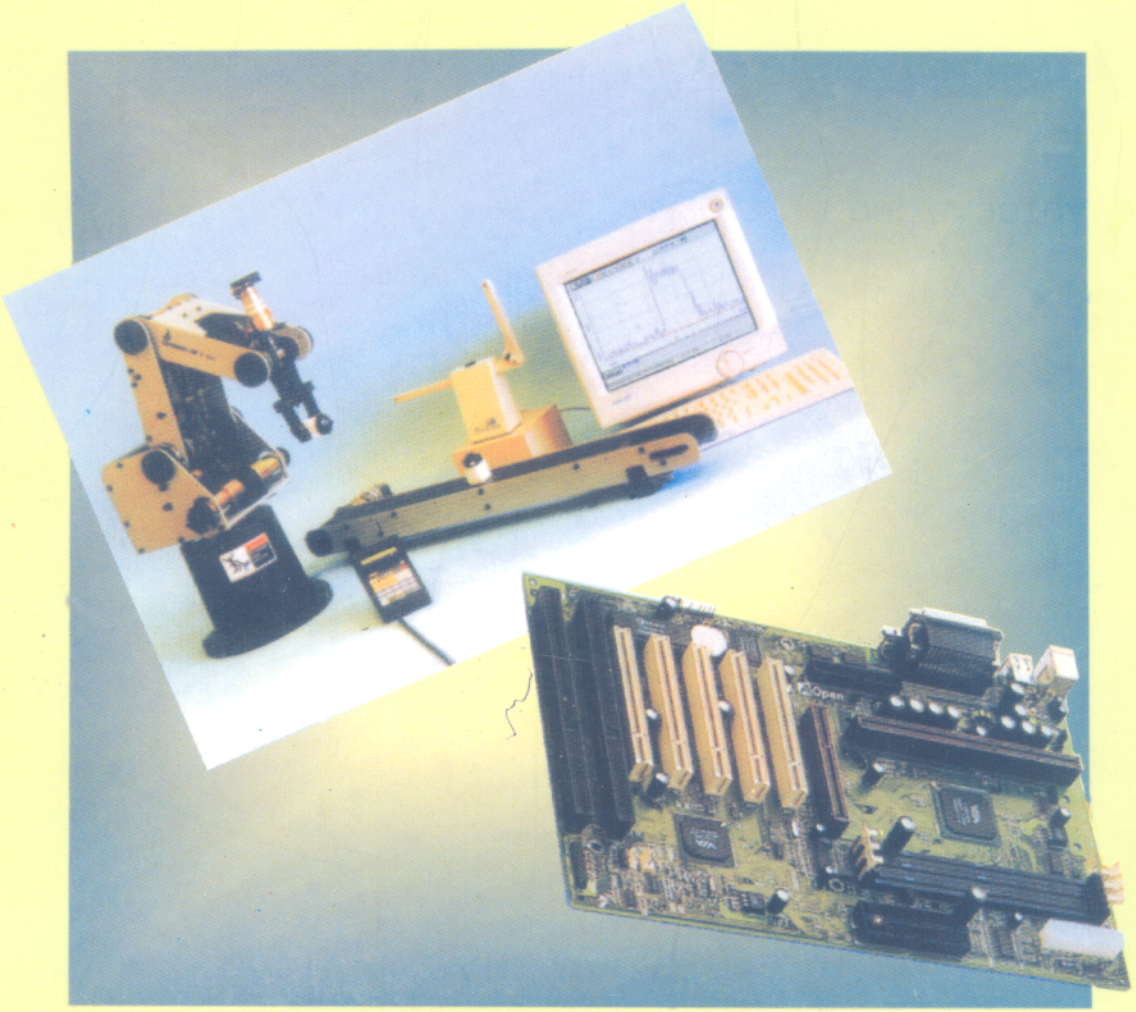
TRẦN QUANG VINH

NGUYÊN LÍ PHẦN CỨNG và KĨ THUẬT GHÉP NỐI MÁY VI TÍNH



NHÀ XUẤT BẢN GIÁO DỤC

TS. TRẦN QUANG VINH

NGUYÊN LÍ PHẦN CỨNG
VÀ
KĨ THUẬT GHÉP NỐI MÁY VI TÍNH

(Tái bản lần thứ nhất)

NHÀ XUẤT BẢN GIÁO DỤC

Bản quyền thuộc HEVOBCO – Nhà xuất bản Giáo dục.

11 – 2007/CXB/128 – 2119/GD

Mã số : 7B594T7 – DAI

Lời nói đầu

Các máy vi tính hiện nay có khả năng xử lí dữ liệu rất mạnh, thậm chí trong nhiều trường hợp không thua kém gì các máy tính mini và máy mainframe. Chúng đang được ứng dụng rộng rãi trong nhiều lĩnh vực khoa học công nghệ và cuộc sống.

Bên cạnh một khối lượng phần mềm hệ thống và ứng dụng đồ sộ trên thế giới hiện nay, công nghệ phần cứng cũng luôn được cải thiện hàng ngày hàng giờ nhằm tạo ra được các thế hệ máy vi tính có tốc độ nhanh hơn nữa, với khả năng nhớ lớn hơn nữa và giá thành có thể chấp nhận được. Đây thực sự là một cuộc cạnh tranh quyết liệt giữa các nhà sản xuất có tên tuổi trên thế giới, giữa các sản phẩm chính thống và không chính thống. Vì vậy, kiến thức về thực hành khảo sát cũng như kiểm tra các thiết bị phần cứng của một máy vi tính điển hình hiện nay là cần thiết đối với mỗi sinh viên trong các ngành có liên quan đến công nghệ thông tin.

Như đã biết, riêng bộ vi xử lí và bộ nhớ chính cùng phần mềm hệ điều hành đã tạo nên một máy vi tính. Nhưng chúng sẽ chỉ là một hệ khép kín nếu như không có cửa sổ mở ra thế giới bên ngoài với các thiết bị ngoại vi đang ngày càng được sử dụng vô cùng phong phú. Cửa sổ đó là các mạch ghép nối vào/ra. Vì vậy, tuy bản thân các mạch ghép nối là một nhân tố cấu thành nên hệ thống máy vi tính chuẩn nhưng cũng có thể coi chúng cùng các thiết bị ngoại vi là những đối tượng quan trọng trong một kĩ thuật có đặc thù riêng là kĩ thuật ghép nối máy tính. Dưới góc độ ứng dụng, có thể tạm phân kĩ thuật này làm hai lĩnh vực chính:

- Kĩ thuật ghép nối máy tính - cụ thể là bộ vi xử lí - với các thiết bị ngoại vi trong một hệ máy vi tính chuẩn. Đó là việc thiết kế, khảo sát các quá trình thông tin giữa bộ vi xử lí với các ngoại vi như màn hình, bàn phím, chuột, máy in v.v... là những thành phần tạo nên một hệ máy vi tính chuẩn.

- Kĩ thuật ghép nối hệ thống máy vi tính chuẩn (nói tắt là máy vi tính, PC) với những thiết bị bên ngoài dùng cho đo lường và điều khiển các quá trình công nghệ, thực nghiệm khoa học v.v... Những thiết bị điện tử bên ngoài này bao gồm cả hai loại: loại hoạt động với các tín hiệu số và loại hoạt động với tín hiệu tương tự hoặc cả hai.

Với những mục đích kể trên, tài liệu được chia thành 3 phần với 19 chương; có thể coi mỗi chương là một bài thực hành lớn cùng các câu hỏi và bài tập liên quan đến một đối tượng cụ thể của phần cứng máy vi tính và một ứng dụng của kĩ thuật

ghép nối. Phần một, từ chương 1 đến chương 4 mô tả về phần cứng máy vi tính và những vấn đề liên quan đến các hoạt động bên trong của vi xử lí. Phần hai, từ chương 5 đến chương 17 có nội dung là kĩ thuật ghép nối máy vi tính. Phần ba gồm 2 chương 18 và 19 mô tả các công cụ (thiết bị điện tử và phần mềm) dùng cho đo đạc kiểm tra máy vi tính và kĩ thuật ghép nối. Riêng phần này, tuỳ trình độ người đọc, có thể không cần xem trước 2 phần trên. Mỗi chương đều có phần đầu trình bày các cơ sở lí thuyết có liên quan. Việc giới thiệu các chương trình phần mềm trong các thí dụ, các chương trình chẩn đoán được dẫn ra ở tài liệu này không thể coi là đầy đủ được; khi muốn biết sâu hơn độc giả cần đọc thêm các sách chuyên khảo hiện đang có sẵn về chúng.

Tài liệu này đã được dùng làm cơ sở cho giảng dạy về cấu trúc và kĩ thuật ghép nối máy vi tính cùng các bài tập thực hành có liên quan của khoa Công nghệ Đại học Quốc gia Hà Nội. Nó cũng có thể được dùng như tài liệu tham khảo cho sinh viên các ngành Công nghệ Thông tin, Điện tử - Viễn thông trong các trường Đại học và những kĩ thuật viên quan tâm đến máy vi tính.

Cuốn sách chắc không tránh khỏi các thiếu sót, vì vậy chúng tôi mong nhận được ý kiến đóng góp của bạn đọc. Các ý kiến xin gửi về :

Bộ môn Điện tử và Kĩ thuật Máy tính, Khoa Công nghệ, Đại học Quốc gia Hà Nội, 144 Đường Xuân Thuỷ, Quận Cầu Giấy, Hà Nội.

hoặc

Nhà Xuất bản Giáo dục, 81 Trần Hưng Đạo, Hà Nội.

Tác giả

PHẦN MỘT

MÁY VI TÍNH
VÀ
BỘ VI XỬ LÍ

CHƯƠNG 1. PHẦN CỨNG VÀ CÁC PHÉP TÍNH SỐ TRONG MÁY VI TÍNH

1.1. VAI TRÒ CỦA MÁY VI TÍNH TRONG THỰC TẾ

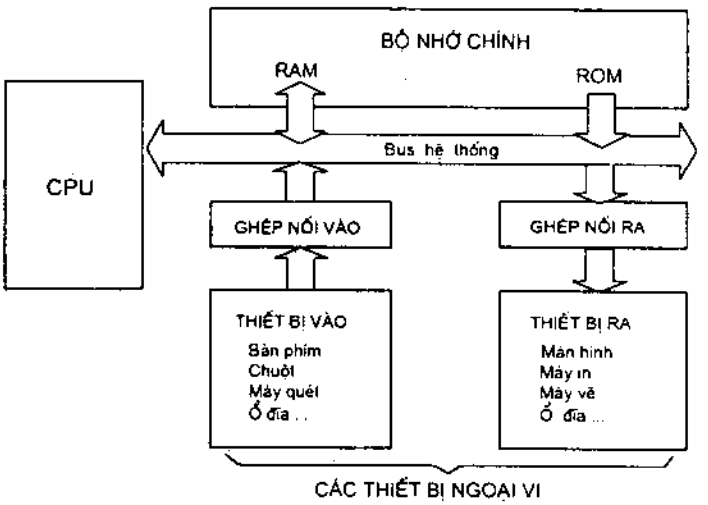
Bộ vi xử lí là thành phần cơ bản để tạo nên một *máy vi tính* hay *máy tính cá nhân* PC. Máy vi tính thực ra là một trong những ứng dụng cụ thể của hệ thống gọi là *hệ vi xử lí* có các khối chức năng như hình 1.1.

Khối xử lí trung tâm CPU (Central Processing Unit).

Bộ nhớ chính.

Các mạch ghép nối vào/ra (cho phép CPU liên lạc với các thiết bị ngoại vi).

Các khối này được nối với nhau qua một tập đường dây truyền tín hiệu gọi là bus hệ thống. Bus hệ thống bao gồm 3 bus thành phần là: bus địa chỉ, bus dữ liệu và bus điều khiển.



Hình 1.1 Sơ đồ khối chức năng của một hệ máy tính.

Khi CPU được chế tạo bởi một mạch vi điện tử có độ tích hợp rất cao thì được gọi là bộ *vi xử lí* (μP). Nó có nhiệm vụ *đọc mã lệnh* là các bit thông tin 0 hoặc 1 từ bộ nhớ chính, *giải mã* các lệnh này thành một chuỗi các xung điều khiển để điều khiển các khối khác *thực hiện từng bước các thao tác trong lệnh*. Bên trong vi xử lí có các *thanh ghi* (registers) để chứa địa chỉ của các lệnh sắp thực hiện (lệnh kế tiếp) như *thanh ghi con trỏ lệnh* IP (Instruction Pointer) hay còn gọi là *bộ đếm chương trình* PC (Program Counter) và các thanh ghi khác dùng để lưu trữ dữ liệu tạm thời hoặc các trạng thái của hệ thống. Các thanh ghi này cùng với *bộ tính số học và lôgic* ALU (Arithmetical and Logic Unit) cho phép thực hiện các thao tác với dữ liệu. Trong vi xử lí thì *đơn vị điều khiển* CU (Control Unit) là phần phức tạp nhất vì nó có chức năng giải mã lệnh và tạo các xung điều khiển toàn hệ thống. *Bộ nhớ chính* (hay còn gọi là bộ nhớ trong) nằm trên bản mạch chính máy tính được chia thành các ô nhớ, trong máy tính IBM/PC mỗi ô nhớ là 1 byte (gồm 8 bit nhỏ). Bộ nhớ này gồm các vi mạch nhớ bán dẫn ROM và RAM có tốc độ truy cập nhanh dùng để chứa các chương trình điều khiển hoạt động của hệ thống. Nó cho phép

CPU nhận lệnh từ đây để khởi động hệ thống. Các chương trình ứng dụng cũng như các dữ liệu cùng các kết quả tạm thời được lưu trữ trong RAM. Ngoài ra còn có *bộ nhớ khối* (còn gọi là bộ nhớ ngoài) là các môi trường lưu trữ tin kiểu khác như đĩa từ, đĩa quang v.v.. có dung lượng nhớ lớn hơn dung lượng bộ nhớ chính nhiều nhưng tốc độ truy cập chậm hơn.

Các mạch ghép nối vào/ra là các mạch điện tử cho phép CPU thông tin được với các thiết bị ngoại vi như bàn phím, màn hình, ổ đĩa từ, chuột v.v.. (để tạo nên một hệ máy vi tính) hoặc các thiết bị ngoại vi như *bộ chuyển đổi số - tương tự* D/A, *tương tự - số* A/D, *mạch vào/ra số* I/O v.v... (để tạo nên một hệ đo lường điều khiển bằng máy vi tính với các đối tượng thực nghiệm bên ngoài). Các thông tin này được trao đổi qua các mạch điện tử cụ thể gọi là các *cổng vào/ra* (I/O port). Tuỳ theo yêu cầu mà chúng có chức năng hoặc là *cổng vào* nếu cho phép CPU nhận thông tin từ ngoài hoặc là *cổng ra* nếu cho phép CPU xuất thông tin ra ngoài hoặc cả hai.

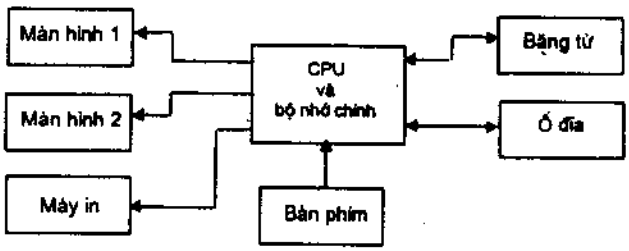
Máy tính còn có một mạch tạo các xung điện gọi là *đồng hồ hệ thống* (system clock) để duy trì hoạt động và đồng bộ hoá CPU cùng các bộ phận liên quan với nhau. Tần số đồng hồ này quyết định tốc độ hoạt động của CPU.

Tất nhiên, để cho các khối trên hoạt động được phải có một *nguồn nuôi*, thường là nguồn cấp năng lượng từ mạng điện thành phố hoặc ắc-quy.

Để người sử dụng có thể làm việc được trên máy tính (thí dụ để viết và cho chạy các chương trình ứng dụng) thì chỉ với các bộ phận phần cứng trên là chưa đủ, một phần mềm gọi là *hệ điều hành* (operating system) phải được cài đặt sẵn trong máy tính. Đó là một tập hợp các chương trình sử dụng để điều hành các chức năng cơ bản của các bộ phận phần cứng hay phần mềm trên hệ thống máy tính.

Trong các lĩnh vực khoa học và đời sống, máy tính số có thể giữ những vai trò như sau:

Máy tính dùng để xử lí dữ liệu: Hình 1.2 là một máy tính được coi như là một bộ xử lí dữ liệu, thí dụ để tính tổng số tiền công phải trả cho từng nhân viên trong cơ quan. Ta có thể gọi toàn bộ các khối trong hình là một máy tính (hay một hệ máy tính) trong khi người nào đó có thể nói: CPU và bộ nhớ chính là một máy tính trong khi các bộ phận còn lại như bàn phím, màn hình, máy in v.v... là các thiết bị ngoại vi. Đâu đó trong bộ nhớ là đoạn chương trình được nhập vào. Đó là một tập hợp các lệnh ở dạng mã máy. Mục đích của chương trình là dùng các dữ liệu nhập vào, xử lí tính toán chúng rồi lưu trữ các kết quả vào bộ nhớ, thí dụ là các mục chi tiết về công phải trả cho từng người ... Các kết quả đó có thể được hiện lên màn hình, in ra giấy in hoặc cất vào các file đĩa trong bộ xử lí dữ liệu này tuỳ người chạy chương trình. Các máy tính được dùng trong bộ xử lí dữ liệu thường được đánh giá trên cơ sở bộ nhớ ngoài lớn bao nhiêu và các máy in cũng như màn hình được tăng cường chất lượng như thế nào.



Hình 1.2 Máy tính là bộ xử lí dữ liệu.

Máy tính là một bộ xử lí số: Trong chức năng kể trên, máy tính dành nhiều thời gian cho việc xử lí các dữ liệu dưới dạng các kí hiệu biểu diễn cho các thông tin, thí dụ thông tin về

các nhân viên trong cơ quan. Lúc này thời gian cần để tính toán các phép tính số là rất nhỏ so với thời gian xử lí dữ liệu. Ngược lại, có khi máy tính lại được sử dụng trong những ứng dụng liên quan tới một số lượng rất lớn các phép tính toán học đôi khi lên tới hàng tỉ phép tính cho một việc. Lúc này máy tính đóng vai trò là một bộ xử lí số. Thí dụ, xét trường hợp ứng dụng máy tính để mô hình hoá qúa trình theo dõi diễn biến của khí hậu. Khí quyển là một môi trường liên tục, ba chiều; được tạo bởi một số rất lớn phân tử của các loại khí khác nhau. Việc tính bài toán trong một môi trường liên tục như vậy rất khó. Tuy vậy có một cách dễ dàng hơn khi giải bài toán này bằng cách coi khí quyển bao gồm một số rất lớn các vật khối lập phương. Nhiệt độ, nồng độ và áp suất trong mỗi khối này được xem như là đồng đều. Sự biến đổi chỉ xảy ra khi các khối hộp ở liền kề nhau. Vì một khối hộp có 6 mặt nên có thể tạo ra một mô hình biểu diễn sự tương tác của 6 mặt đó với các mặt lân cận của các khối hộp khác. Có thể giả thiết rằng ở thời điểm ban đầu tất cả các khối hộp là như nhau (tức là giữa chúng không có sự tương tác từ trước), rồi sau đó bắt đầu tính toán xem cái gì sẽ xảy ra nếu có một nguồn năng lượng nào đó, thí dụ mặt trời, tác động lên mô hình này. Tác động của mỗi khối hộp lên khối hộp bên cạnh sẽ được tính toán và toàn bộ xử lí được lặp đi lặp lại. Để nhận được kết quả chính xác, kích thước của các khối hộp phải đủ nhỏ, nếu không giả thiết về tính đồng nhất các tham số trong mỗi khối hộp sẽ bị phá vỡ. Hơn nữa, số tương tác cần thiết để nhận được kết quả đồng quy về một giá trị trạng thái dừng (tĩnh) thường phải rất lớn. Kết quả là chương trình của những bài toán thuộc loại này đòi hỏi phải được chạy trong thời gian rất dài ở những máy tính rất mạnh thường được gọi là *siêu máy tính* (supercomputer).

Một thí dụ khác về bộ xử lí số là việc ứng dụng máy tính trong các hệ đo, điều khiển theo thời gian thực (real time). Thời gian thực ở đây có nghĩa là các kết quả tính toán phải được có ngay trong một khoảng quy định từ khi khởi phát tính. Thí dụ, xét việc ứng dụng máy tính trong hệ điều khiển không lưu. Một anten ra-đa quay vòng sẽ đo vị trí phương hướng và khoảng cách của mỗi máy bay đang bay trên bầu trời - gọi là các đích. Tại thời điểm t, đích thứ i sẽ ở vị trí P(t,i) cho một xung phản xạ trên màn ra-đa là khoảng cách r(i,t) và phương hướng b(i,t). Do có các can nhiễu nên cần phải tính đến ảnh hưởng của chúng tới các xung phản xạ nhận được từ các đích. Máy tính nhận được các số liệu từ máy thu ra-đa của n đích (n máy bay), các số liệu được cập nhật p lần trong mỗi phút. Từ các số liệu ban đầu này, nó phải tính toán vị trí của mỗi máy bay, vẽ đường đi của chúng và cảnh báo cho người điều khiển không lưu về các khả năng va chạm có thể xảy ra. Điều này đòi hỏi phải có một máy tính có tốc độ tính toán nhanh đáng kể.

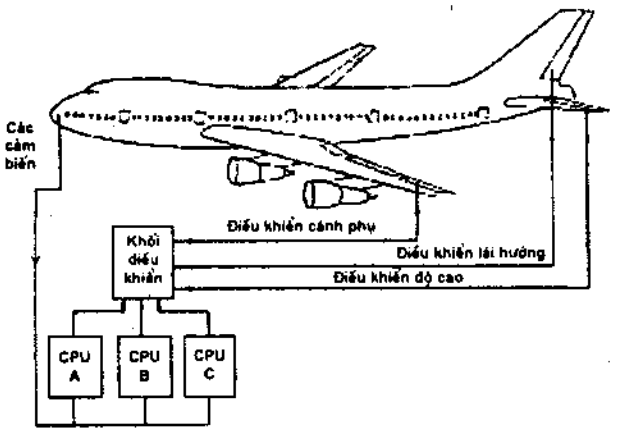
Các máy tính dùng trong xử lí số thường là các máy mạnh, có CPU rất đắt, bộ nhớ chính có tốc độ truy cập rất cao và các thiết bị ngoại vi cùng bộ nhớ ngoài tương đối hiện đại.

Máy tính dùng để đo lường và điều khiển tự động: Do giá thành ngày càng rẻ, máy tính ngày nay còn được đặt ở trung tâm của nhiều hệ thống điều khiển tự động. Khi được sử dụng như một phần tử điều khiển, máy tính thường được gắn vào một hệ thống lớn hơn và người sử dụng thường không thấy được cụ thể là đã có một máy tính trong đó. Thí dụ, một máy tính on-chip (máy tính được chế tạo trên một chip) được lắp trong một máy giặt tự động để tính toán thời gian cần thiết cho các quá trình làm sạch, giũ và điều khiển tốc độ động cơ quay vắt nước quần áo. Trong trường hợp này bà nội trợ thường không quan tâm tới việc cái máy giặt được điều khiển bởi một cơ cấu định thời bằng cơ khí của những năm trước đây hay đã được điều khiển bởi một bộ vi xử lí hiện đại nhất, miễn là nó hoạt động tốt ! Một thí dụ điển hình

của việc sử dụng máy tính trong đo lường điều khiển tự động là hệ thống điều khiển hạ cánh tự động trên máy bay được minh hoạ đơn giản trong hình 1.3.

Ở đây vị trí của máy bay (chiều cao, khoảng cách hạ cánh, khoảng cách lệch khỏi tâm đường bay) và tốc độ được xác định bởi các kĩ thuật vô tuyến khác nhau liên quan tới hệ thống điều khiển hạ cánh trên mặt đất. Các thông tin về vị trí của máy bay được cấp tới 3 máy tính riêng biệt để xác định các độ lệch trong quá trình bay. Đó là các độ lệch giữa vị trí đo được của máy bay với vị trí mà nó cần phải đạt được.

Các kết quả tính toán được xuất ra khỏi máy tính là các tín hiệu điều khiển các bộ phận lái của máy bay (như cánh phụ, cánh lái độ cao, cánh lái hướng) hoặc nếu cần thiết điều chỉnh công suất phát ra của động cơ. Tất cả nhằm điều chỉnh vị trí của máy bay theo chiều hướng sao cho các độ lệch kể trên trở về không. Chương trình máy tính ở đây được lưu trữ trong bộ nhớ chỉ đọc ROM. Việc sử dụng 3 máy tính trong sơ đồ này nhằm tăng tính tin cậy của hệ thống, đề phòng khả năng một trong những máy tính bất ngờ bị hỏng trong quá trình điều khiển.



Hình 1.3 Điều khiển hạ cánh tự động trên máy bay bằng vi tính.

Căn cứ vào các chỉ tiêu về kích thước, giá thành, các thiết bị ngoại vi kèm theo và hiệu suất; các máy tính ngày nay được phân làm 3 loại: *máy tính lớn* (mainframe), *máy tính nhỏ* (minicomputer) và *máy vi tính* hay máy tính cá nhân (micro computer hay personal computer).

Máy tính lớn được xây dựng để giải các bài toán lớn với tốc độ rất nhanh. Nó thường làm việc với các bus dữ liệu lên tới 64 bit hoặc hơn nữa cùng với một bộ nhớ rất lớn. Do vậy nó cũng có kích thước lớn. Các máy tính này thường được dùng trong những ứng dụng quân sự, trong các ngành hàng không, hàng hải, khí tượng thuỷ văn, ngân hàng, bảo hiểm v.v...

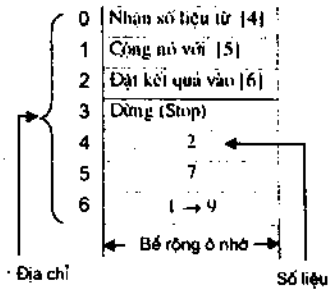
Máy tính nhỏ là một dạng thu hẹp về tính năng cũng như kích thước của máy tính lớn. Nó đáp ứng được cho các ứng dụng vừa phải mà nếu dùng máy tính lớn sẽ gây lãng phí. Bus dữ liệu ở đây thường có độ dài từ 32 bit và bộ nhớ nhỏ hơn. Máy tính nhỏ thường dùng cho các mục đích tính toán khoa học kĩ thuật hay điều khiển các quá trình công nghệ.

Máy vi tính sử dụng bộ vi xử lí làm đơn vị xử lí trung tâm, kích thước cũng như công suất tiêu tán nhỏ cho phép dùng cho một người. Cũng có khi máy vi tính là một bộ vi điều khiển trong một hệ thống lớn. Độ dài của bus dữ liệu thường là 32 (có thể lên tới 64 bit) với dung lượng bộ nhớ vừa phải. Do sự phát triển rất nhanh chóng của công nghệ những năm gần đây ranh giới giữa máy tính nhỏ và máy vi tính ngày càng bị xoá nhoà đi.

1.2. QUAN HỆ GIỮA PHẦN CỨNG VÀ PHẦN MỀM - CẤU TRÚC PHÂN MỨC CỦA MÁY TÍNH

1.2.1. Quan hệ giữa phần cứng và phần mềm

Trong thực tế, máy tính số hiện nay thường được *tổ chức* theo nguyên tắc xây dựng *các hoạt động xảy ra nối tiếp nhau*. Ở một vùng nào đó trong bộ nhớ là một chương trình được lưu trữ. Chương trình trong máy tính bao gồm một tập hợp các lệnh được sắp xếp theo một trật tự nào đó. Khi chạy chương trình, CPU sẽ đọc lần lượt từng lệnh một từ bộ nhớ và thực hiện nó. Các số liệu ban đầu thuộc về chương trình cũng như các số liệu tính toán trung gian cũng được lưu trữ ngay trong bộ nhớ như các lệnh; chỉ có khác là các lệnh được lưu trữ ở một vùng còn các số liệu được lưu trữ ở một vùng khác. Một máy tính được tổ chức như vậy được gọi là máy kiểu Von Neumann.



Hình 1.4 Chương trình và số liệu trong bộ nhớ.

Hình 1.4 biểu diễn một thí dụ cho thấy chương trình gồm các lệnh và số liệu cùng được tồn tại trên một bộ nhớ. Ở đây bộ nhớ gồm 7 vùng kế tiếp nhau được đánh số từ 0 đến 6 gọi là được *đánh địa chỉ*. Mỗi địa chỉ được chứa các con số có ý nghĩa hoặc là lệnh hoặc là số liệu. Như trên hình cho thấy chương trình được lưu trữ trong các địa chỉ từ [0] đến [3] gồm 4 lệnh. Vùng nhớ chứa số liệu có các địa chỉ từ [4] đến [6]. Mỗi lần chỉ có một địa chỉ được thâm nhập. Lệnh đầu tiên nằm trong địa chỉ [0] có nghĩa là: lấy số liệu là nội dung ở địa chỉ [4] từ bộ nhớ, trong trường hợp này là số 2. Lệnh tiếp theo ở địa chỉ [1] là: cộng nó với số liệu là nội dung ở địa chỉ [5], tức là số 7. Sau khi cộng xong 2 số này được kết quả 2 + 7 = 9, máy tính tiếp tục thực hiện lệnh tiếp theo ở địa chỉ [2] là: lưu trữ kết quả là số 9 vào địa chỉ [6]. Lệnh cuối cùng ở địa chỉ [3] cho máy kết thúc chạy chương trình. Số liệu 1 ở địa chỉ [6] chỉ là một giá trị ngẫu nhiên mà ô nhớ có trước khi chạy chương trình.

Phần cứng máy tính bao gồm tất cả những linh kiện vật lí tạo nên hệ thống máy tính. Đó là bộ xử lí trung tâm CPU, bộ nhớ, các thiết bị vào/ra v.v... Các chương trình điều khiển hoạt động của máy tính bao gồm các lệnh gọi là *phần mềm*. Một dạng trung gian giữa phần cứng và phần mềm gọi là *phần dẻo* (firmware). Đó là các phần mềm được đặt sẵn bên trong các bộ phận điện tử trong hệ thống khi sản xuất. Thí dụ như các chương trình đặt trong các vi mạch nhớ EPROM của các thiết bị máy tính, các vi chương trình v.v... Trong những máy tính đầu tiên, giữa phần cứng và phần mềm được phân biệt rất rõ ràng. Nhưng cùng với sự phát triển của công nghệ thông tin, ranh giới giữa chúng nhiều khi được xoá nhoà đi do một thao tác bất kì được thực hiện bởi phần mềm cũng có thể được thực hiện trực tiếp bởi phần cứng là các mạch cổng điện tử số và ngược lại một hoạt động bất kỳ được thực hiện trên sơ đồ nguyên lí các mạch điện tử cũng có thể được mô phỏng bằng một chương trình phần mềm. Việc người thiết kế sắp xếp bao nhiêu phần thao tác dành cho phần cứng hay phần mềm là tuỳ thuộc vào các yêu cầu về tốc độ, giá thành, độ tin cậy, v.v... của máy tính. Như vậy, mặc dù phần cứng và phần mềm là những thực thể rất khác nhau nhưng lưu ý rằng giữa chúng cần phải có một sự sắp xếp nào đó nhằm cho toàn hệ thống máy tính đạt được tính hiệu quả trong hoạt động. Thí dụ, một vài phép toán có thể được thực hiện hoặc bằng các hệ thống phần cứng đặc biệt hoặc bằng các

chương trình lưu trữ trong bộ nhớ. Nói chung cách nhanh nhất để thực hiện một nhiệm vụ là xây dựng một mạch điện dùng riêng cho nhiệm vụ đó. Nhưng điều này có thể phải trả một giá thành sản phẩm rất cao. Việc viết chương trình để thực hiện một vài nhiệm vụ với một máy tính có sẵn thì rẻ đáng kể. Nhưng khi phần cứng của máy tính không được thiết kế tối ưu cho nhiệm vụ đó thì chương trình sẽ trở nên quá dài.

Có một vài lí do để người sử dụng máy tính cần phải hiểu biết về phần cứng:

- Việc lập trình tự nó đã tiềm ẩn các hoạt động về phần cứng. Thí dụ như việc vào/ra dữ liệu. Khi chương trình gửi hay nhận dữ liệu qua các cổng vào/ra là nó đang sử dụng các thiết bị phần cứng mà chương trình phải hiểu rõ trước.

- Phần cứng xác định khả năng của một máy tính. Bất kì người sử dụng máy tính nào đều phải biết trước khả năng đó. Thí dụ, máy tính dẫn đường trong máy bay cần phải có mạch điện cho phép một tốc độ tính toán đủ nhanh để điều chỉnh một thông số nào đó trước khi nó trệch xa khỏi giá trị cần thiết.

- Việc lập trình không thể tách rời khỏi phần cứng được. Thí dụ về việc xây dựng một hệ xử lí được dùng để điều khiển nhiệt độ của phản ứng hoá học. Một hệ thống như vậy sẽ liên quan đến việc biến đổi đại lượng nhiệt độ thành điện thế qua các cảm biến như *cặp nhiệt điện*; tiếp đó biến đổi điện thế đó ra thành giá trị số bằng *bộ biến đổi tương tự-số (ADC)*, đọc nó vào máy tính, xử lí rồi xuất một số khác ra một thiết bị biến đổi ngược lại thành điện thế là *bộ biến đổi số-tương tự (DAC)*; và cuối cùng sử dụng điện thế đó để điều khiển bếp đun. Để thực hiện tốt các hoạt động đó cần phải có kiến thức của hai người: một là kĩ thuật viên điện tử và một là người lập trình. Đôi khi chúng có ở cả trong một người, vừa là một lập trình viên giỏi vừa tinh thông về kĩ thuật phần cứng điện tử.

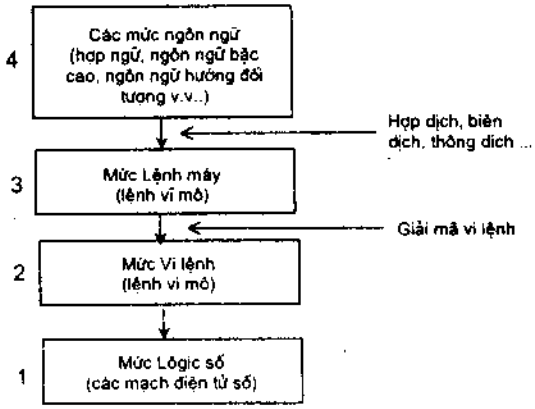
- Việc nắm bắt được phần cứng máy tính cho phép người lập trình hiểu biết chính xác cái gì đang xảy ra sau chương trình của mình không như người chỉ biết máy tính như một cái hộp đen !

1.2.2. Cấu trúc phân mức của máy tính

Máy tính là một thiết bị thông minh rất phức tạp với hàng triệu linh kiện điện tử và rất nhiều các chương trình. Nó là đối tượng cho nhiều loại người dùng để nghiên cứu, thiết kế và sử dụng. Có những người quan tâm đến cấu trúc phần cứng, đó là những người làm trong lĩnh vực nghiên cứu chế tạo máy tính hay kỹ thuật ghép nối máy tính với các thiết bị ngoại vi. Nhưng có nhiều người lại chỉ quan tâm đến cấu trúc phần mềm và một chút ít phần cứng vì họ là những lập trình viên. Còn những người dùng các chương trình ứng dụng thì chỉ nhìn nhận máy tính qua các chương trình ứng dụng như là một công cụ để sử dụng cho một mục đích nào đó mà thôi. Như vậy, có lẽ cách tổng quát là nhìn nhận một máy vi tính như là một hệ thống bao gồm phần cứng và phần mềm có tính tương đương nhau về mặt lôgic. Giống như đã làm trong kỹ thuật thông tin máy tính về mô hình phân tầng, ta có thể coi máy tính về mặt lôgic được phân thành các mức như hình 1.5.

Mức 1 ở đây thực sự là phần cứng của máy vi tính, đó là các mạch điện tử cổng, các trigger, mạch giải mã, chuyển kênh, các bộ đếm v.v...Các mạch này được gắn kết với nhau qua các cổng liên kết để tạo thành các nhóm lớn hơn có những chức năng riêng phục vụ cho việc thực thi một

hoặc vài vi lệnh của mức 2. Các máy tính ban đầu thực ra không có mức lôgic thứ 2 này vì các mạch điện tử số ở mức 1 được thiết kế để có thể thực hiện được ngay các lệnh máy ở mức 3.



Hình 1.5 Cấu trúc phân mức của máy vi tính.

Các lệnh máy này là các tập lệnh như tập lệnh của các vi xử lý mà các nhà sản xuất cung cấp cho người dùng. Nhưng rồi tập lệnh này (gọi là lệnh vĩ mô, macro-instructions) ngày càng nhiều và phức tạp (thí dụ là hơn 300 lệnh trong họ 80x86) mà nếu thiết kế để mức 1 thực hiện ngay chúng thì số mạch điện tử bị kéo theo trở nên rất nhiều và phức tạp thêm. Do vậy, ý tưởng xây dựng một khâu trung gian biến đổi các lệnh máy "muôn hình muôn vẻ" thành các tổ hợp của một tập lệnh đơn giản hơn là lệnh vi mô (micro-instructions) thường cỡ 20 vi lệnh) ra đời. Chỉ có các lệnh vi mô này mới gắn kết thực sự với các mạch điện tử số ở mức 1. Như vậy, một lệnh máy ở mức 3 phía trên qua khâu giải mã sẽ trở thành một số các vi lệnh ở mức 2. Sự xuất hiện mức lệnh máy và mức vi lệnh là do các yêu cầu thiết kế về phần cứng trong khi các mức trên nữa lại xuất hiện với lý do bởi người lập trình và sử dụng. Các lệnh máy là tập hợp các con số "vô nghĩa" 0 và 1 quá buồn tẻ và phức tạp với các lập trình viên nên một hệ thống các từ gợi nhớ được sử dụng và như vậy *hợp ngữ* ra đời. Các ngôn ngữ bậc cao hơn với cú pháp câu lệnh ngắn gọn, sáng sủa gần với ngôn ngữ đời thường của con người được xây dựng nên các mức lôgic cao hơn trong mô hình cấu trúc phân mức này của máy tính. Tuy nhiên một quá trình thực hiện lệnh hoặc chạy chương trình trong máy tính phải được thực hiện từ trên xuống đến tận mức 1 và do vậy các ngôn ngữ phải qua các bước như biên dịch, hợp dịch ... đến giải mã vi lệnh. Nếu nhìn nhận kỹ hơn nữa phải thêm vào một mức dưới mức 1, đó là mức linh kiện điện tử được mô tả trên các nguyên lý hoạt động của các linh kiện tạo nên các mạch cổng như transistor bán dẫn, điện trở, dây dẫn v.v... vì các tính chất vật lý của chúng cũng quyết định đến hiệu năng của toàn máy tính.

Việc phân máy tính thành các mức như vậy thực ra là sự khái quát lại cách nhìn nhận của riêng mỗi người. Người đó dưới góc độ sử dụng của mình sẽ nghiên cứu sử dụng máy tính ở một "độ cao", ở một mức nào đó và chỉ nghĩ về máy tính như một máy vật lý thực sự như họ nhìn thấy ở mức này và không cần biết các chi tiết phức tạp của các mức dưới. Tuy nhiên, những ai cần tìm hiểu cách thức một máy tính làm việc thực sự (như các nhà thiết kế máy tính, các lập trình viên muốn tìm ảnh hưởng của phần cứng đến tốc độ làm việc của các chương trình trong kỹ thuật ghép nối chẳng hạn) thì nên phải chú ý tới tất cả các mức.

1.3. CÁC PHÉP TÍNH SỐ TRONG MÁY TÍNH

Lượng thông tin nhỏ nhất được lưu trữ và thao tác trong máy tính là *bit*. Máy tính số lưu

trữ thông tin trong các bộ nhớ thành dạng các nhóm bit gọi là *từ nhớ* (word). Số bit trong một từ tuỳ từng trường hợp có thể khác nhau. Một từ dài 8 bit gọi là một *byte*. Một từ dài 4 bit gọi là một *nibble*. Máy tính tương thích IBP/PC truy xuất bộ nhớ mỗi lần ít nhất là 1 byte hay có thể nói 1 ô nhớ trong nó là chứa 1 byte thông tin. Ở đây thường gọi nhóm chứa 2 byte là một từ (word) , và một từ kép (double word) gồm 4 byte...

Các từ nhớ, là các số trong máy tính; ngoài việc *biểu diễn định lượng cho một con số* còn có thể *biểu diễn cho các lệnh* (thí dụ từ "1111 1000" là mã lệnh xoá cờ nhớ trong vi xử lý 8086 của hãng Intel), có thể *biểu diễn cho các ký tự* (gồm các chữ cái và ký hiệu in ra được hoặc ký tự điều khiển không in ra được, thí dụ từ "0100 0001" là mã ASCII của chữ "A") hoặc có thể *biểu diễn cho các điểm ảnh* (pixel) trong một ảnh đồ hoạ về độ sáng, màu, v.v...

1.3.1. Các hệ đếm liên quan đến máy tính

Trong cuộc sống hàng ngày ta thường dùng hệ đếm *cơ số mười* hay *hệ thập phân* (Decimal number system). Trong hệ này dùng một tổ hợp các chữ số (digit) từ 0 đến 9 để biểu diễn giá trị các số. Các mạch điện trong máy tính ở từng thời điểm chỉ ở một trong hai trạng thái đóng hoặc mở hay gọi là các trạng thái lôgic "0" hoặc "1" của các linh kiện điện tử tạo nên nó. Do đó, để biểu diễn một số trong máy tính phải dùng *hệ cơ số hai* hay *hệ nhị phân* (Binary number system). Trong hệ này ta dùng một tổ hợp các chữ số 0 hoặc 1 để biểu diễn giá trị các số. Mỗi chữ số trong một số nhị phân đại diện cho một *bit* thông tin.

Thường các số được biểu diễn bằng cách *ký hiệu theo vị trí*, nghĩa là giá trị hay "trọng lượng" của mỗi chữ số trong một số tuỳ thuộc vào vị trí của nó bên trong một số. Tổng quát, một số N khi được biểu diễn theo vị trí với cơ số b sẽ được viết là:

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 a_{-1} a_{-2} \ldots a_{-m}$$

ở đây:

$$a_n b^n + a_{n-1} b^{n-1} + \ldots + a_1 b^1 + a_0 b^0 + a_{-1} b^{-1} + a_{-2} b^{-2} + \ldots + a_{-m} b^{-m} = \sum_{i=-m}^{n} a_i b^i$$

Các chữ a trong biểu thức trên là các *digit* và có thể nhận một trong b giá trị. Khi cần phân biệt rõ hệ đếm thường viết thêm chỉ số dưới ký hiệu cơ số vào các số. Thí dụ, số 65 trong hệ thập phân được viết là 65_{10}. Cũng là số đó trong hệ nhị phân được viết là 1000001_2 . Dấu phẩy (dấu chấm trong tiếng Anh) dùng để phân tách giữa phần nguyên và phân số. Trong hệ thập phân ta có dấu chấm thập phân, trong hệ nhị phân ta cũng có dấu chấm nhị phân v.v...

Thí dụ, số thập phân 123,25 sẽ có giá trị số bằng tổng của các tích giữa các hệ số 1, 2, 3, 2 và 5 với các trọng lượng khác nhau:

$$123{,}25 = 1\times10^2 + 2\times10^1 + 3\times10^0 + 2\times10^{-1} + 5\times10^{-2}.$$

Và, một số nhị phân 101,01 sẽ có giá trị số với các trọng số 2 mũ tương ứng lần lượt như sau:

$$101{,}01 = 1\times2^2 + 0\times2^1 + 1\times2^0 + 0\times2^{-1} + 1\times2^{-2}.$$

Như lý do kể trên, theo cách viết thông thường bit tận cùng bên trái của một số nhị phân gọi là *bit có nhiều ý nghĩa nhất MSB* (Most Significal Bit); ngược lại, bit tận cùng bên phải gọi là *bit có ít ý nghĩa nhất* LSB (Least Significal Bit). Đôi khi, khi xử lý một số gồm nhiều byte thì người ta cũng thường gọi các byte tận cùng bên trái hoặc bên phải là MSB (byte nhiều ý nghĩa nhất, byte cao) hoặc LSB (byte ít ý nghĩa nhất, byte thấp).

Hiện nay có 4 hệ đếm hay được dùng trong những bài toán liên quan đến máy tính, đó là:

Hệ mười (decimal)	b = 10	a = {0,1,2,3,4,5,6,7,8,9}
Hệ nhị phân (binary)	b = 2	a ={0,1}
Hệ tám (octal)	b = 8	a = {0.1.2.3.4.5.6.7}
Hệ mười sáu (hexaldecimal)	b = 16	a = {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}

Lưu ý rằng các chữ A, B, C, D, E, F trong hệ mười sáu biểu diễn cho các chữ số có giá trị tương ứng với các số 10, 11, 12, 13, 14, 15 trong hệ thập phân.

Giữa các hệ đếm thường xuyên phải có sự chuyển đổi qua lại trong các thiết bị nhập/xuất của máy vi tính.

• Đổi số thập phân sang nhị phân: Quy tắc là chia liên tiếp số đó cho 2 và sau mỗi phép chia ghi lấy phần dư cho tới khi được thương bằng 0. Đảo ngược thứ tự dãy các số dư sẽ được các chữ số của số hệ hai cần tìm. Thí dụ, đổi số 11 sang số nhị phân

Thí dụ, đổi số 11_{10}

11 : 2 = 5 dư R = 1 ↑
5 : 2 = 2 R = 1 |
2 : 2 = 1 R = 0 | vậy kết quả là $11_{10} = 1011_2$
1 : 2 = 0 R = 1 |

Khi số hệ mười có phần thập phân sau dấu phảy thì đầu tiên phải đổi riêng rẽ từng phần rồi sau đó cộng các kết quả lại. Đối với phần nguyên ta có thể làm theo cách vừa nêu trên, riêng đối với phần thập phân đổi theo quy tắc sau: Lấy số cần đổi nhân với 2, tích nhận được sẽ gồm phần nguyên và phần lẻ nhị phân. Lấy phần lẻ đó nhân tiếp với 2. Làm như vậy cho tới khi được tích chẵn bằng 1 hoặc tới giá trị chính xác mong muốn. Chọn riêng các phần nguyên của các tích thu được và sắp xếp lại sẽ được các chữ số sau dấu phảy của số hệ nhị phân cần tìm.

Thí dụ, đổi số $0,125_{10}$ ra số nhị phân

0, 125 × 2 = [0], 250
0, 250 × 2 = [0], 500 ⇒ $0,125 = 0,001_2$
0, 500 × 2 = [1], 000

Kết hợp các quy tắc nêu trên ta được kết quả đổi số

$$34,125_{10} = 100010,001_2.$$

• Đổi số thập phân ra các hệ khác cũng theo quy tắc kể trên, thí dụ đổi số 53241_{10} ra hệ mười sáu như sau:

53241 : 16 = 3327 R = $9_{10} = 9_{16}$
3327 : 16 = 207 R = $15_{10} = F_{16}$
207 : 16 = 12 R = $15_{10} = F_{16}$
12 : 16 = 0 R = $12_{10} = C_{16}$

vậy, $53241_{10} = CFF9_{16}$ hay CFF9h

• Đổi số hệ nhị phân sang số hệ mười: bằng cách cộng các số mũ 2 ứng với các digit bằng 1 trong số nhị phân. Thí dụ:

1	0	1	0	1	1	1	←	số nhị phân
64	32	16	8	4	2	1	←	các trọng số

64 + 16 + 4 + 2 + 1 = 87_{10} kết quả

Khi số bit lớn, thường dùng thuật giải đệ quy như sau: Lấy bít bằng 1 nhiều ý nghĩa nhất (tận cùng bên trái) nhân với 2 rồi cộng với bít tiếp theo (bên phải nó). Cứ stiếp tục như vậy cho tới bit ít ý nghĩa nhất được cộng vào. Thuật toán này có thể được biểu diễn như sau:

$(a_0 + 2(a_1 + 2(a_2 + ...)))$ ở đây bit ít ý nghĩa nhất LSB của số cần đổi là a_n.

Thí dụ, 10111_2 → 1 0 1 1 1

⇒ ⇒ 2→2 ⇒ 4→5 ⇒ 10→11 ⇒ 22→23

Kết quả là $10111_2 = 23_{10}$.

• Chuyển đổi giữa các số nhị phân và số hệ mười sáu: nếu lưu ý rằng một nhóm số nhị phân 4 bit vừa đủ biểu diễn một chữ số trong hệ mười sáu ta sẽ nhóm số nhị phân cần chuyển đổi thành các nhóm 4 bit và đổi mỗi nhóm đó ra 1 chữ số mười sáu.

Thí dụ 11001011101_2 → nhóm: 110 0101 1101

6 5 D

Kết quả $11001011101_2 = 65D_{16}$

Ngược lại, khi chuyển đổi một số hệ mười sáu sang số nhị phân chỉ việc đổi mỗi chữ số hệ mười sáu sang một nhóm 4 bit.

Thí dụ, số $AB4C_{16} = 1010\ 1011\ 0100\ 1100_2$

A B 4 C

• Giữa hệ thập phân và nhị phân tồn tại một hệ lai *BCD* (Binary Coded Decimal). Hệ này mã hoá mỗi con số trong hệ thập phân bằng một số nhị phân. Thường dùng một số nhị phân 4 bit để mã hoá cho các chữ số từ 0 đến 9. Như vậy không dùng hết các tổ hợp có thể có của 4 bit ($2^4 = 16$).

Thí dụ, số 520_{10} nếu biểu diễn theo số BCD sẽ là:

Số thập phân	5	2	0
Các số BCD	0101_{BCD}	0010_{BCD}	0000_{BCD}

Mã nhị phân được mô tả ở trên thuộc loại mã nhị phân tự nhiên hay loại mã có trọng số 8421. Tuy nhiên trong một số trường hợp đặc biệt, thí dụ như để phát hiện và sửa lỗi trong quá trình truyền các từ dữ liệu, còn có các loại mã khác không có trọng số như mã Gray, mã Hamming v.v...

1.3.2. Các mã phát hiện và sửa lỗi

Trong quá trình truyền dữ liệu có thể gặp phải sự thay đổi giá trị các bit thông tin do can nhiễu hoặc sai hỏng của các thiết bị thu, phát. Các lỗi này có thể được phát hiện hoặc sửa nhờ phương pháp đặc biệt là truyền đi các mã phát hiện lỗi EDC (error detecting code) hoặc mã sửa lỗi ECC (error correcting code). Một trong những phương pháp phát hiện và sửa lỗi là: khi muốn kiểm tra lỗi một từ dữ liệu m bit người ta ghép thêm r bit kiểm tra được mã hoá theo một cách nào đó rồi truyền từ ghép dài $n = m + r$ bit đi. Bằng cách kiểm tra từ ghép này tại bên

thu, có thể phát hiện được đã có lỗi hay không xảy ra trong từ *m* bit và trong một số trường hợp có thể biết chính xác lỗi đó xảy ra ở các bit nào để sửa. Vì rằng các các bit kiểm tra không mang thông tin nên chúng thường được gọi là các *bit dư thừa* (redundant bits). Có nhiều loại mã EDC và ECC, ta chỉ xét một vài loại

1.3.2.1. *Phát hiện lỗi với bit chẵn lẻ*

Mã EDC đơn giản nhất là một bit chẵn lẻ đơn (parity bit). Đó là một bit kiểm tra được ghép thêm vào các bit của xâu dữ liệu. Có 2 loại bit chẵn lẻ: chẵn lẻ chẵn và chẵn lẻ lẻ. Nếu dùng loại chẵn lẻ chẵn, bit này có giá trị là 0 hoặc 1 sao cho tổng số các con số 1 trong xâu ghép là một số chẵn. Nếu dùng loại chẵn lẻ lẻ thì ngược lại chọn sao cho tổng số các con số 1 trong xâu ghép là một số lẻ. Thí dụ, nếu dùng bít chẵn lẻ-chẵn, với từ số liệu 8 bit (1001 0110). giá trị của bit chẵn lẻ-chẵn ở đây sẽ bằng 0 vì số bit có giá trị 1 trong xâu số liệu bằng 4 là một số chẵn rồi. Ngược lại, nếu dùng bít chẵn lẻ-lẻ thì bit chẵn lẻ ở đây là 1.

Xâu ghép (gồm các bit thông tin và bit kiểm tra chẵn lẻ) được phát đi và ở bên thu nếu phát hiện thấy tổng số các bit có giá trị bằng 1 không theo các quy luật trên có nghĩa rằng đã có bit lỗi trong bộ nhớ hoặc trên đường truyền dữ liệu.

Phương pháp đơn giản kể trên có nhược điểm là không định vị được bit lỗi và không thể phát hiện được các lỗi không phải là bit đơn (thí dụ như khi có 2 bit trong xâu cùng bị lỗi thì giá trị của bit chẵn lẻ vẫn không thay đổi). Để khắc phục có thể dùng thêm phương pháp kiểm tra chẵn lẻ cho từng khối số liệu được truyền. Ở đây dùng mã phát hiện lỗi EDC khối. Một mã phát hiện lỗi khối chẵn lẻ dùng 2 loại bit kiểm tra: bit chẵn lẻ dọc VRC (vertical parity bit) và bit chẵn lẻ ngang LRC (horizontal – longitudinal parity bit).

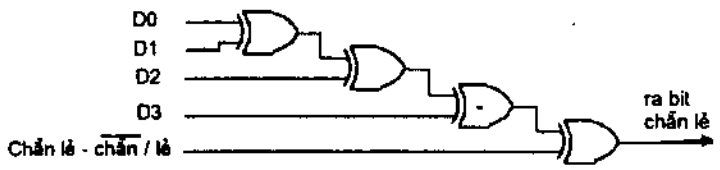
Thí dụ, mỗi từ số liệu gồm 3 bit D0, D1 và D2 cần phát và thu với kiểm tra chẵn lẻ – chẵn trong hệ thống được viết thành một cột dọc và một chuỗi gồm 7 từ được xắp xếp liền kề nhau thành khối theo chiều ngang như sau:

Bit	từ 1	từ 2	từ 3	từ 4	từ 5	từ 6	từ 7	LRC
D0	0	1	1	0	0	0	1	1
D1	1	0	0	1	0	1	1	0
D2	1	1	0	1	1	0	1	1
D3 (VRC)	0	0	1	0	0	1	0	0

√ √

Bit chẵn lẻ dọc VRC cho mỗi từ được thêm vào đáy mỗi cột tạo thành bit D3. Trên cơ sở một hàng các bit D0 của 7 từ sẽ tính được 1 bit chẵn lẻ ngang LRC. Cũng như vậy, tính được các bit chẵn lẻ ngang cho các hàng bit D1, D2 và cả các bit VRC trong khối. Nếu khi thu phát hiện có lỗi ở 2 vị trí đánh dấu √ trên hình ta có thể biết ngay bit D1 của từ mã thứ 3 bị lỗi.

Các phép thống kê chỉ ra rằng việc sử dụng đồng thời cả hai phương pháp VRC và LRC làm tăng xác xuất phát hiện lỗi so với khi chỉ dùng VRC lên 2 đến 4 bậc.Việc tạo các bit chẵn lẻ và kiểm tra lỗi chẵn lẻ thường được thực hiện bằng phần cứng là các mạch lôgic tổ hợp (thí dụ như các mạch hoặc tuyệt đối XOR). Hiện có sẵn những vi mạch chuyên dụng như 74AS280 cho phép tạo và kiểm tra lỗi chẵn lẻ với 9 bit dữ liệu vào. Hình 1.6 là một sơ đồ một mạch tạo các bit chẵn lẻ cho các từ dài 4 bit D0-D3 với 1 lối vào tuỳ chọn chẵn lẻ-chẵn (0) và chẵn lẻ-lẻ (1).



Hình 1.6 Mạch tạo bit chẵn lẻ bằng các cổng XOR.

1.3.2.2. *Phát hiện lỗi bằng mã dư thừa vòng CRC*

Kiểm tra bằng mã CRC (Cycle Redundancy Check) được dùng để phát hiện lỗi khi đọc số liệu từ đĩa từ mềm. Từ dữ liệu được ghép thêm vào các byte kiểm soát lỗi CRC khi phát.

Nguyên lí tạo mã CRC như sau: Một xâu bít bất kì có thể được coi là một tập các hệ số của một đa thức B(x), trong đó "x" là hư số. Thí dụ, byte 1000 0011 ứng với đa thức $B(x) = 1.x^7 + 0.x^6 + 0.x^5 + ... +1.x^1 + 1 = x^7 + x + 1$. Chọn một đa thức G(x) nào đó gọi là *đa thức sinh* (generation polynomial). Khi chia modulo-2 đa thức B(x) cho G(x) ta được thương số Q(x) và số dư R(x):

$B(x) = Q(x).G(x) \oplus R(x)$, ở đây $\oplus$ là kí hiệu của phép cộng modulo-2,

hay: $Q(x) = [B(x) \oplus R(x)] / G(x)$.

Như vậy dãy bít số liệu tạo bởi xâu ghép B(x) và số dư R(x) sẽ chia hết cho G(x). Các byte kiểm soát lỗi CRC được tạo bởi các hệ số của số dư R(x) đó. Khi đọc dữ liệu bên thu, mạch lôgic kiểm soát lỗi sẽ lấy xâu ghép đó chia modulo-2 cho đa thức sinh G(x). Nếu phép chia có dư thì nghĩa là đã có lỗi trong xâu dữ liệu.

Đa thức sinh do tổ chức viễn thông quốc tế CCITT quy định là $x^{16} + x^{12} + x^5 + 1$.

Vì các phép modulo-2 tương đương với việc thực hiện phép *hoặc tuyệt đối* nên các việc tạo mã CRC cho quá trình ghi và kiểm tra lỗi cho quá trình đọc dữ liệu thường được thực hiện rất đơn giản bằng các mạch điện tử số.

1.3.2.3. *Mã sửa lỗi Hamming*

Đây là loại mã sửa lỗi đơn giản. Một từ mã Hamming gồm *m* bit dữ liệu và *r* bit kiểm tra chẵn lẻ. Các bit kiểm tra được chọn như thế nào đấy để một lỗi bit đơn có thể được phát hiện chính xác vị trí và do đó có thể sửa được. Các mã Hamming được gán cho ký hiệu là $H_{n, m}$ trong đó n là độ dài từ mã bằng (m + r). Thí dụ, từ mã $H_{7,4}$ như sau:

7	6	5	4	3	2	1	← vị trí bit
I_4	I_3	I_2	C_3	I_1	C_2	C_1	I: các bit dữ liệu, C: các bit kiểm tra

Lưu ý rằng vị trí của các bit kiểm tra nhất thiết phải ở các vị trí nhị phân (1, 2, 4, ...) và các bit này được mã hoá theo quy luật sau:

$C_3 = I_2 \oplus I_3 \oplus I_4$

$C_2 = I_1 \oplus I_3 \oplus I_4$

$C_1 = I_1 \oplus I_2 \oplus I_4$

Giả thử rằng từ dữ liệu cần truyền là $I_4I_3I_2I_1 = 1101$.

Các bít kiểm tra tính như trên bằng $C_3C_2C_1 = 010$. Như vậy từ mã truyền đi là 1100110. Nếu như có lỗi, thí dụ bit I_3 chuyển từ 1 tới 0. Lúc đó từ mã nhận được sẽ là 1000110. Bên thu sẽ tính các bit kiểm tra

$C_3 = 0 \oplus 0 \oplus 1 = 1$

$C_2 = 1 \oplus 0 \oplus 1 = 0$

$C_1 = 1 \oplus 0 \oplus 1 = 0$

Từ bit kiểm tra sẽ là 100 trong khi chúng là 010.

Nếu lấy modulo-2 hai số này ta có: $100 \oplus 010 = 110$ chính là vị trí thứ 6. Hay bit thứ 6 tức là I_3 đã có lỗi và cần phải sửa lại từ 0 về 1! Các công thức tính các bit kiểm tra ở đây được thực hiện như vậy nhằm cho bất kỳ một lỗi bit đơn nào đó xảy ra sẽ dẫn tới thay đổi tổ hợp của các bit kiểm tra này và trỏ tới chính vị trí bit lỗi đó. Bằng cách thêm vào một bit kiểm tra nữa có thể tạo được mã Hamming cho phép phát hiện 2 lỗi và sửa 1 lỗi...

1.3.3. Biểu diễn các số nguyên không dấu và có dấu

Một số nhị phân n bit có thể biểu diễn được 2^n số nguyên không dấu có giá trị từ 0 đến 2^n-1, thí dụ số nhị phân 1 byte có thể biểu diễn cho các số nguyên dương từ 0 đến 255 và số 2 byte biểu diễn cho các số từ 0 đến 65535.

Các số nguyên có dấu dùng bit tận cùng bên trái làm bit dấu: bit bằng 0 biểu diễn cho dấu (+) và bit bằng 1 cho dấu (-). Một cách tổng quát, số nhị phân dài n bit sẽ biểu diễn được các số nguyên có dấu trong khoảng từ $-(2^{n-1})$ đến $+(2^{n-1}-1)$. Thí dụ, số nhị phân dài 8 bit có thể biểu diễn được các số nguyên âm và dương trong khoảng từ -128 ... 0 ... +127.

Một số nguyên dương trong khoảng này sẽ được biểu diễn như mã nhị phân thông thường có số 0 ở bít cao nhất. Thí dụ: 65d = 0100 0001b

Một số nguyên âm trong khoảng này sẽ được biểu diễn theo mã *số bù hai* của số nguyên dương tương ứng. Số bù hai B' của số B được xác định sao cho: B' + B = 0. Tức là nếu muốn tìm số bù hai của B ta chỉ việc lấy đảo của nó rồi cộng thêm 1. Thí dụ: (-65d) = mã bù hai của số (+ 65d) = 1011 1110 + 0000 0001 = 1011 1111b.

1.3.4. · Các số có dấu chấm động

Do cách viết các số có phần phân số, dấu phảy phân số (thí dụ dấu thập phân trong các số hệ mười) theo tiếng Việt được thay bằng dấu chấm theo tiếng Anh trong các máy tính. Vì vậy, trong mục này ta sẽ gọi *số có dấu phảy* (phân số) là *số có dấu chấm*.

Xét 2 phép tính sau:

```
  7 6 3 2 1 3 5          7 6 3. 2 1 3 5
+ 1 7 9 4 8 2 1        + 1 7 9. 4 8 2 1
-------------          ---------------
9 4 2 6 9 5 6          9 4 2. 6 9 5 6
```

Giống như cách tính tay của người, máy tính sẽ tính toán phép tính bên phải như với các số nguyên rồi chỉ việc thêm dấu chấm thập phân vào lưu trữ ở một ô nhớ xác định thích hợp. Cách thức tính toán như vậy gọi là cách tính với *dấu chấm tĩnh*. Thí dụ khác, cộng hai số 3,625 và 6,5 rồi in ra kết quả. Chương trình nhập dữ liệu sẽ chuyển 2 số đó ra dạng nhị phân:

3.625 → 11.101b → 0011.1010

6.5 → 110.1b → 0110.1001

Đến đây máy tính coi các số này là 2 số nhị phân 1 byte 0011 1010 và 0110 1000 rồi cộng theo quy tắc thông thường:

```
  0 0 1 1 1 0 1 0
+ 0 1 1 0 1 0 0 0
-----------------
  1 0 1 0 0 0 1 0   = 162₁₀ (số nhị phân không dấu)
```

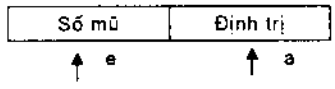
Chương trình đến đây sẽ nhận kết quả và phân tách nó ra thành 2 phần: phần nguyên 1010 và phần phân số .0010 rồi in kết quả chính xác là $1010.0010_2 = 10.125_{10}$.

Nhưng, khi cần biểu diễn và xử lí những phân số và những số rất dài, thí dụ các số bao gồm rất nhiều con số 0 mà chỉ có một ít con số có ý nghĩa như hai số sau:

Khối lượng mặt trời: 1990000000000000000000000000000000 g

Khối lượng điện tử: 0.000000000300000000000000000910956 g

Máy tính sẽ phải dùng đến một số lớn byte nhớ để lưu trữ các con số không như vậy. Như đã biết rằng 1 byte biểu diễn được 256 số (từ 0 đến 255) thì để biểu diễn một số trong dải rất rộng từ lớn đến nhỏ như vậy cần đến cỡ 14 byte cho phần nguyên và 12 byte cho phần thập phân; vậy tổng cộng cần tới 28 byte = 208 bit cho một số ! Rõ ràng trong trường hợp này dùng loại số có dấu chấm tĩnh sẽ rất bất tiện. Máy tính số thường biểu diễn và lưu trữ những số trên trong dạng *dấu chấm động* (floating point) như sau: 1.990×10^{33} g và $0,910956 \times 10^{-27}$ g hay theo cách *viết số khoa học* là 1.99E33 g và 0.910956E-27 g. Tổng quát, một số dấu chấm động sẽ có dạng $a \cdot r^e$ trong đó a là *phần định trị*, r là *cơ số* và e là *số mũ*. Nếu cơ số r đã được biết trước (thường là cơ số 2 cho số nhị phân trong máy tính) thì máy sẽ lưu trữ một số dấu chấm động thành 2 trường số liệu như hình sau:



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

Số có độ chính xác đơn dài 32 bit (Single)

Số có độ chính xác kép dài 64 bit (Double)

Số có độ chính xác mở rộng dài 128 bit (Quadruple).

Thường dạng biểu diễn các số có dấu chấm động *được chuẩn hoá*. IEEE định nghĩa một số dấu chấm động có cơ số ngầm định là 2, được chuẩn hoá sao cho phần định trị F nằm trong dải $1 \le F < 2$ (tương ứng với phần nguyên bằng 1). Một số X sẽ được định nghĩa là:

$$X = -1^S \times 2^{E-B} \times 1.F$$

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

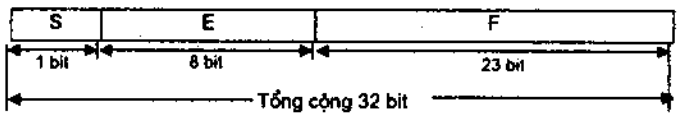
E là số mũ được chuyển dịch đi B ; F là phần định trị.

Thường độ dịch B được chọn bằng 2^{m-1} trong đó m là độ dài bit của trường số mũ. Điều đó cho phép lưu trữ các số mũ trong dạng được chuyển dịch chỉ gồm toàn các số nhị phân không âm vì số mũ âm nhất đã được chuyển dịch thành 0. Thí dụ, trường số mũ dài 8 bit, m =8, do vậy dải số mũ thực sự là từ -128 tới +127 . Nếu chọn $B = 2^{8-1} = 128$ ta sẽ có số mũ âm nhất được dịch sẽ là -128 + 128 = 0 và số mũ dương lớn nhất là +127 + 128 = + 255.

Bảng sau cho các định dạng của các loại số dấu chấm động theo IEEE.

Loại	Single	Double	Quadruple
Bề rộng của trường (bit)			
S	1	1	1
E	8	11	15
F	23	52	111
Tổng cộng	32	64	128
Bit dấu	0=+, 1=-	0=+, 1=-	0=+, 1=-
Số mũ			
E cực đại	255	2047	32767
E cực tiểu	0	0	0
Độ dịch	127	1023	16383

Thí dụ: một số chính xác đơn dài 32 bit có độ dịch là 127 và phần phân số của định trị là 23 bit có định dạng như sau:



Do vậy số -2345,125 trong hệ mười ở máy tính sẽ là một số dài 16 bit :

- 2345.125d = - 1001 0010 1001. 001b (số nhị phân bình thường)

= - $1.00100101001001 \times 2^{11}$ (số đã được chuẩn hoá)

Phần định trị âm, do vậy bit dấu S là 1;

Số mũ có độ dịch được cho là +11+127 = 138 = 10001010;

Phần phân số của định trị là .00100101001001000000000 (số 23 bit) ;

Do đó, dạng IEEE của số này là:

110001010 00100101001001000000000. Nó được lưu trữ bởi 2 từ 16 bit là:
1100010100010010 và 1001001000000000

Khác với số nguyên và số có dấu chấm tĩnh, các phép tính số học với số dấu chấm động có phức tạp hơn do các số khi được chuẩn hoá sẽ không có cùng số mũ như nhau. Để thực hiện một phép cộng (hoặc trừ) hai số dấu chấm động phải tiến hành theo các bước sau:

- Tăng số mũ của số có số mũ nhỏ hơn cho bằng số mũ lớn hơn.
- Chia phần định trị của số nhỏ hơn cho cùng một hệ số nhân ở bước trên.
- Cộng (hoặc trừ) các phần định trị.
- Nếu cần thiết, chuẩn hoá lại kết quả.

Thí dụ đơn giản, cộng hai số trong hệ thập phân A = 0.12345×10^5 và B = 0.56789×10^3. Số mũ của B nhỏ hơn nên cần tăng lên số mũ lên 2 tức là nhân với 100 và do vậy phần định trị của nó phải chia cho 100, tức là B = 0.0056789×10^5. Vậy:

A = 0. 1 2 3 4 5 0 0 × 10^5
B = 0. 0 0 5 6 7 8 9 × 10^5
\+
0. 1 2 9 1 2 8 9 × 10^5 = 0.1291289E5

1.3.5. Các phép toán số học đối với các số nhị phân

Quy tắc về các phép toán cộng, trừ, nhân 2 số nhị phân 1 bit cho trong bảng sau:

Cộng	Trừ	Nhân
0 + 0 = 0	0 - 0 = 0	0 x 0 = 0
0 + 1 = 1	0 - 1 = 1 mượn 1	0 x 1 = 1
1 + 0 = 1	1 - 0 = 1	1 x 0 = 1
1 + 1 = 1 nhớ 1	1 - 1 = 1	1 x 1 = 1

Các phép toán với số nhị phân n bit có thể được thực hiện như theo các cách như sau.

• ***Phép cộng*** được thực hiện giống cách tính tay thông thường. Thí dụ, cộng 2 số 0101b = 5d và 0111b = 7d như sau: 0101 + 0111 = 1100 = 12d.

Các mạch điện tử thực hiện phép cộng là các mạch tổng toàn phần FA (full adder).

```
    0 1 0 1
+   0 1 1 1
  ---------
  1 1 0 0
```

• ***Phép trừ*** trong máy tính được thực hiện bằng việc cộng số bị trừ với *số bù hai* của số trừ. Thí dụ, thực hiện phép trừ 2 số A - B với A = 1001 ; B = 0011; số đảo của B là 1100. Vậy B' = 1101. Do đó A - B = 1001 + 1101 = (1)0110 = 6_{10} . Số 1 trong ngoặc ( ) được coi là số nhớ (carry)

• ***Phép nhân***

• *Phép nhân các số không có dấu*:

Về nguyên tắc, phép nhân hai số chính là phép cộng lặp nhiều lần. Số lần lặp bằng số nhân trừ đi 1. Đặc biệt nếu số nhân là một số 2^n (n nguyên) thì kết quả chính là phép dịch trái số bị nhân đi n bit. Một trong những quy tắc nhân 2 số nhị phân được thực hiện giống như cách tính tay thông thường ở hệ thập phân như sau:

- Bắt đầu từ bit ít ý nghĩa nhất LSB, nhân từng bit của số nhân n bit với số bị nhân rồi dịch trái đi 0, 1, 2, ... bit để được n các tích thành phần.
- Kết quả nhân (tổng tích luỹ) là tổng các tích thành phần.

Thí dụ, nhân hai số nhị phân 4 bit:

```
                    1 0 0 1     Số bị nhân (9)
                X   0 1 1 0     số nhân (6)
                  -----------
                    0 0 0 0     ← tích thành phần 1
                  1 0 0 1       ← tích thành phần 2
                1 0 0 1         ← tích thành phần 3
              0 0 0 0           ← tích thành phần 4
            -------------------
            0 1 1 0 1 1 0       ← tổng tích luỹ kết quả: 54
```

Trong thực tế máy tính số không dùng thuật giải kể trên vì sẽ đòi hỏi phải lưu trữ n tích thành phần rồi mới cộng đồng thời n từ số liệu đó. Một thuật giải tốt hơn là cộng ngay các tích thành phần khi chúng vừa được tính ra như các bước sau khi dùng cho phép nhân 2 số nhị phân không dấu n bit:

a, Đặt bộ đếm tới n.

b, Xoá thanh ghi 2n bit chứa tích thành phần.

c, Kiểm tra bit LSB của số nhân; nếu là 1 thì cộng số bị nhân vào n bit nhiều ý nghĩa nhất của tích thành phần.

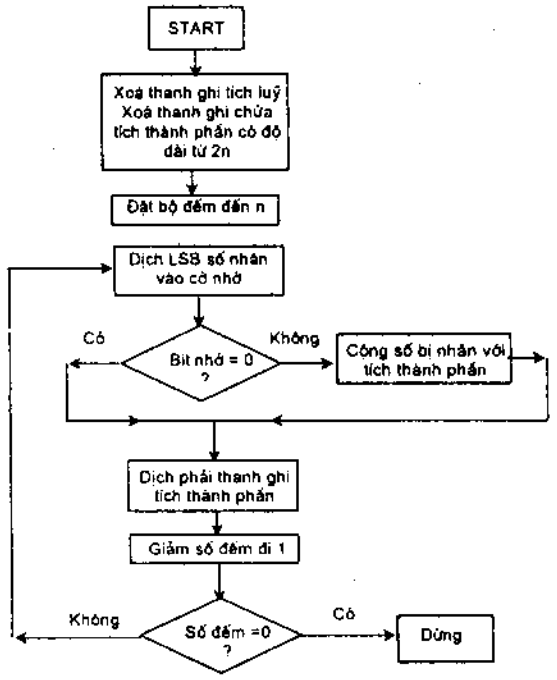
d, Dịch phải 1 bit tích thành phần.

e, Dịch phải số nhân 1 bit (bit LSB sẽ bị mất).

f, Giảm bộ đếm đi 1. Lặp lại các bước từ c cho đến khi bộ đếm bằng không. Đọc kết quả từ thanh ghi tích thành phần.

Lưu đồ của việc nhân các số không dấu theo giải thuật trên được trình bày như dưới đây. Bảng bên cạnh dẫn giải rõ về một trường hợp nhân 2 số không dấu với số nhân là 1101 và số bị nhân là 1010.

Bước	Bộ đếm	Số nhân	Tích thành phần	Chu trình
(a) và (b)	4	1101	0000 0000	
(c)	4	1101	1010 0000	1
(d) và (e)	4	0110	0101 0000	1
(f)	3	0110	0101 0000	1
(c)	3	0110	0101 0000	2
(d) và (e)	3	0011	0010 1000	2
(f)	2	0011	0010 1000	2
(c)	2	0011	1100 1000	3
(d) và (e)	2	0001	0110 0100	3
(f)	1	0001	011 00100	3
(c)	1	0001	1000 0100	4
(d) và (e)	1	0000	1000 0010	4
(f)	0	0000	1000 0010	4



♦ *Phép nhân các số có dấu*: Phương pháp kinh điển để nhân các số có dấu trong dạng bù 2 là dùng giải thuật Booth. Giải thuật này áp dụng cho 2 số dương, một số âm và một dương hoặc cả 2 số đều là số âm. Giải thuật được thực hiện bằng cách kiểm tra 2 bit của số nhân với nhau để quyết định một trong 3 bước tiếp theo:

1. Nếu bit nhân hiện tại bằng 1 và bit nhân bậc thấp hơn tiếp sau bằng 0 thì trừ số bị nhân từ tích thành phần.
2. Nếu bit nhân hiện tại bằng 0 và bit nhân bậc thấp tiếp theo bằng 1 thì cộng số bị nhân với tích thành phần.
3. Nếu bit nhân hiện tại giống như bit nhân bậc thấp hơn tiếp theo thì không làm gì cả.

Lưu ý 1: khi cộng số bị nhân với tích thành phần thì bỏ qua bit nhớ được sinh ra trong phép cộng.

Lưu ý 2: Khi tích thành phần bị dịch thì dùng phép dịch số học và bit dấu cũng được thêm vào.

Lưu ý 3: Khởi đầu, khi bit hiện tại của số nhân là bit ít ý nghĩa nhất thì bit bậc thấp hơn tiếp theo của số nhân được cho bằng 0.

Bảng sau minh hoạ 3 trường hợp nhân các số 13 × 15, -13 × 15 và -13 × (-15).

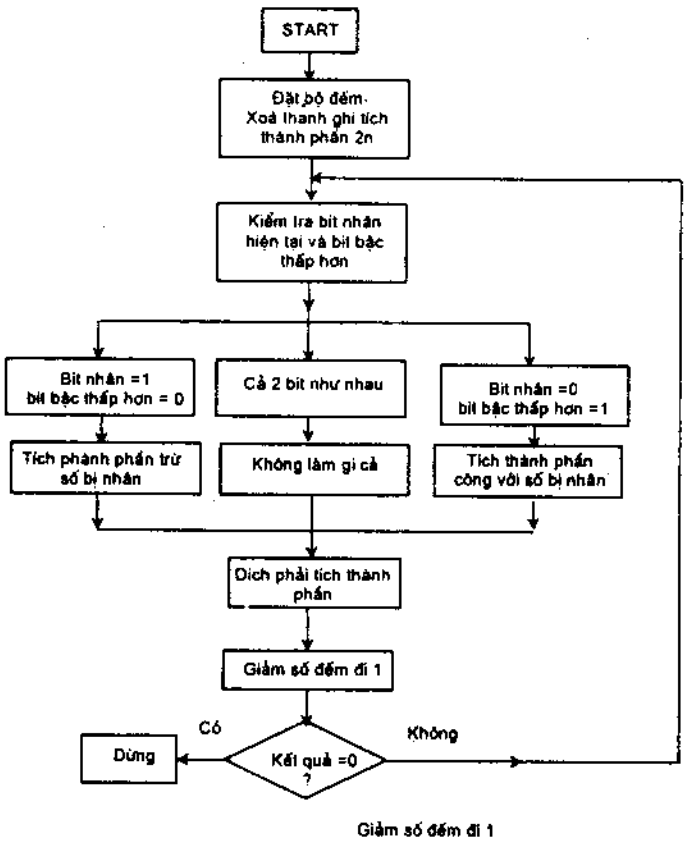
1. Số bị nhân = 01111 = +15 Số nhân = 01101 = +13	Bit nhân	Tích thành phần
Trừ số bị nhân	0 1 1 0 1 0	0000000000
Dịch phải tích thành phần		1000100000
Cộng số bị nhân	0 1 1 0 1	1100010000
Dịch phải tích thành phần		10011110000
Trừ số bị nhân	0 1 1 0 1	0001111000
Dịch phải tích thành phần		1010011000
Không làm gì	0 1 1 0 1	1101001100
Dịch phải tích thành phần		1110100110
Cộng số bị nhân	0 1 1 0 1	10110000110
Dịch phải tích thành phần		0011000011

2. Số bị nhân = 01111 = +15 Số nhân = 10011 = -13	Bit nhân	Tích thành phần
Trừ số bị nhân	1 0 0 1 1 0	0000000000
Dịch phải tích thành phần		1001000000

Không làm gì	10011	1100010000
Dịch phải tích thành phần		1100010000
Cộng số bị nhân	10011	1110001000
Dịch phải tích thành phần		1010110100
Không làm gì	10011	0010110100
Dịch phải tích thành phần		0001011010
Trừ số bị nhân	10011	1001111010
Dịch phải tích thành phần		1100111101

3. Số bị nhân = 10001 = -15 Số nhân = 10011 = -13	Bit nhân	Tích thành phần
Trừ số bị nhân	100110	0111100000
Dịch phải tích thành phần		0011110000
Không làm gì	10011	0011110000
Dịch phải tích thành phần		0001111000
Cộng số bị nhân	10011	1010011000
Dịch phải tích thành phần		1101001100
Không làm gì	10011	1101001100
Dịch phải tích thành phần		1110100110
Trừ số bị nhân	10011	10110000110
Dịch phải tích thành phần		0011000011

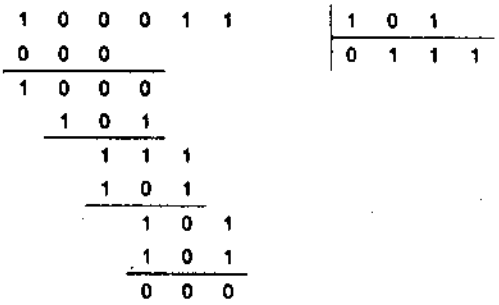
Giải thuật Booth được thực hiện như lưu đồ sau:



Giảm số đếm đi 1

• ***Phép chia*** là phép tính ngược lại của phép nhân nên sẽ là phép trừ lặp đi nhiều lần và phép dịch phải số bị chia đi n bit sẽ cho kết quả thương là 2^n. Về nguyên tắc cũng có thể thực hiện phép chia 2 số nhị phân cho nhau như phép tính tay thông thường như thí dụ sau. Chia hai số 35 : 5 = 7 trong hệ nhị phân như sau.

$100011 : 101 = 111 = 7_{10}$.



Có nhiều thuật toán chia, một trong những thuật toán được thực hiện trong máy tính trên cơ sở các phép cộng và dịch như sau:

a, Đổi số chia ra số bù hai.

b, Lấy số bị chia trừ đi số chia. Nếu kết quả có bit dấu bằng 0 thì bit tương ứng của thương bằng 1; nếu kết quả này có bit dấu bằng 1 thì bit tương ứng của thương bằng 0 và phải khôi phục lại giá trị ban đầu của số bị chia bằng cách cộng kết quả này với số chia ở mã nhị phân.

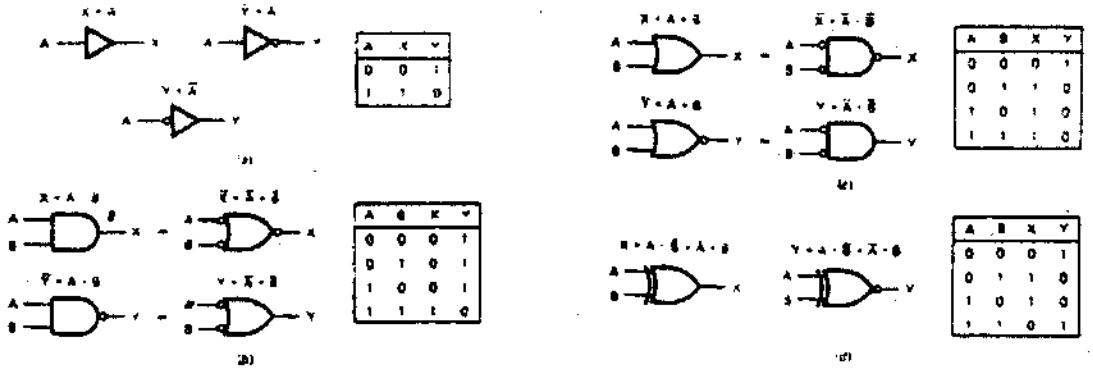
c, Dịch trái kết quả thu được ở trên và làm lại bước 2 cho đến khi nhận được kết quả cuối cùng là 0 hoặc nhỏ hơn số chia (nếu phép chia có dư).

1.4. CÁC LINH KIỆN ĐIỆN TỬ SỐ TRONG MÁY TÍNH

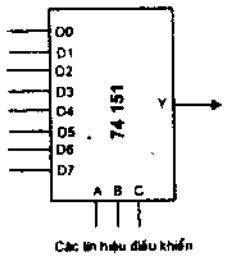
Máy vi tính hiện nay gồm một số chip điện tử số có độ tích hợp lớn LSI và rất lớn VLSI được kết nối với nhau qua các cổng lôgic. Phần này sẽ chỉ nhắc lại một cách hết sức ngắn gọn các linh kiện điện tử số dùng cho vi xử lý và vi tính. Chi tiết về chúng đã được trình bày kỹ tại các giáo trình về điện tử số.

1.4.1. Các cổng lôgic

Hình 1.7a chỉ ra ký hiệu và bảng chân lý của các mạch đệm (buffer) mạch đảo NOT. Các cổng AND và NAND được chỉ ra trên hình 1.7b; các cổng OR và NOR hình 1.7c và cổng XOR hình 1.7d. Các cổng được ứng dụng để tạo nên các mạch lôgic tổ hợp (combination logic) như các bộ lập mã, giải mã, các bộ dồn kênh và phân kênh. Hình 1.8 là một thí dụ vi mạch 74151 dồn kênh 1 từ 8. Hình 1.9 là vi mạch 74 138 phân kênh 3 đường vào ABC thành một trong 8 đường ra Y0-Y7.

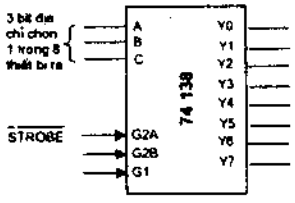


Hình 1.7 Các cổng lôgic.



C	B	A	Y
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

Hình 1.8 Bộ dồn kênh (multiplexor) 74 151.



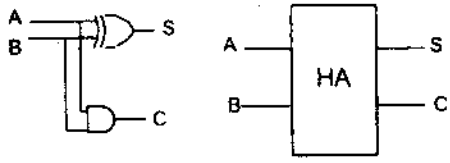
Các lối vào cho phép G1	G2A	G2B	C	B	A	Các lối ra Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
x	1	1	x	x	x	1	1	1	1	1	1	1	1
0	x	x	x	x	x	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Hình 1.9 Bộ phân kênh 74 138.

Bộ bán cộng HA được xây dựng trên các cổng lôgic như hình 1.10.

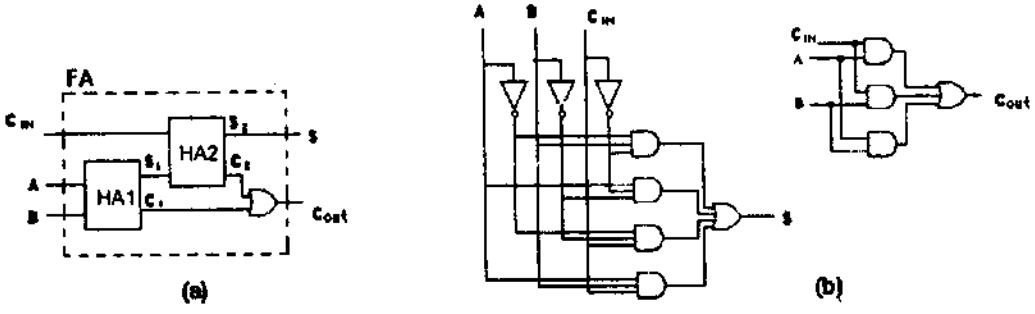
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Sum Cary



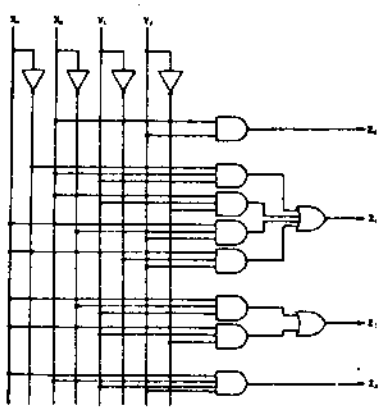
Hình 1.10 Bộ bán cộng.

Bộ cộng toàn phần FA về nguyên tắc được xây dựng trên các bộ bán cộng như hình 1.11a nhưng cũng có thể được lắp trực tiếp như hình 1.11b.



Hình 1.11 Bộ cộng toàn phần.

Như trên đã nói, có khi cần thực hiện các phép nhân có tốc độ cao thì phải sử dụng những mạch điện tử số thực hiện các phép nhân này. Hình 1.12 là một thí dụ về mạch nhân 2 số nhị phân 2 bit cùng bảng chân lí của nó.



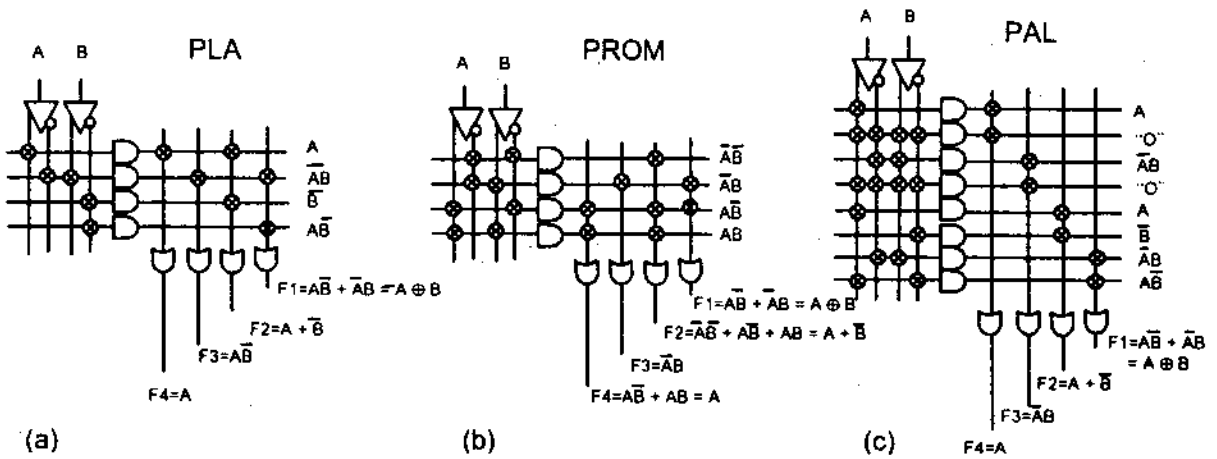
	Inputs				Outputs			
	X		Y		Z			
X*Y=Z	X_1	X_0	Y_1	Y_0	Z_3	Z_2	Z_1	Z_0
0*0=0	0	0	0	0	0	0	0	0
0*1=0	0	0	0	1	0	0	0	0
0*2=0	0	0	1	0	0	0	0	0
0*3=0	0	0	1	1	0	0	0	0
1*0=0	0	1	0	0	0	0	0	0
1*1=1	0	1	0	1	0	0	0	1
1*2=2	0	1	1	0	0	0	1	0
1*3=3	0	1	1	1	0	0	1	1
2*0=0	1	0	0	0	0	0	0	0
2*1=2	1	0	0	1	0	0	1	0
2*2=4	1	0	1	0	0	1	0	0
2*3=6	1	0	1	1	0	1	1	0
3*0=0	1	1	0	0	0	0	0	0
3*1=3	1	1	0	1	0	0	1	1
3*2=6	1	1	1	0	0	1	1	0
3*3=9	1	1	1	1	1	0	0	1

Hình 1.12 Bảng chân lí và mạch nhân 2 số nhị phân 2 bit.

1.4.2. Các mạch lôgic khả trình.

Thay vì cho các cổng rời rạc, các máy vi tính hiện nay thường dùng các mạch lôgic khả trình để liên kết các chip LSI. Hình 1.13 chỉ ra sơ đồ của các mạch kiểu như vậy.

Chúng gồm các ma trận AND và OR có thể chương trình hoá được, do vậy có thể dễ dàng thực hiện một hàm tín hiệu lối ra là là tổng của các tích tín hiệu lối vào. Mỗi cổng AND trong sơ đồ này có 4 lối vào nhưng để đơn giản ta chỉ vẽ một đường dây. Cũng vậy, các cổng OR có một vài lối vào nhưng cũng được vẽ đơn giản chỉ có một dây. Các mạch được chương trình hoá bằng việc làm đứt các cầu chì (được vẽ là dấu x) ở các vị trí bắt chéo nhau giữa 2 dây dọc và ngang. Dấu x chỉ rằng cầu chì được nối, thí dụ giữa lối ra của một cổng AND và một trong các lối vào của cổng OR. Trong mạch PLA (Programmable Logic Array) hoặc FPLA (Field PLA) hình 1.13a; cả hai ma trận AND và OR được chương trình hoá nhờ việc làm đứt các cầu chì.



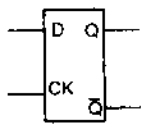
Hình 1.13 Các mạch lôgic khả trình PLA, PROM và PAL.

Với hai ma trận khả trình cho phép sử dụng FPLA một cách rất mềm dẻo nhưng việc lập trình có khó khăn. Trong mạch nhớ ROM khả trình PROM (programmable Read-Only Memory) hình 1.13b, các ma trận AND được chế tạo cố định và chỉ có ma trận OR được lập trình. Các vi mạch

PROM hay được dùng trong các bộ biến đổi mã. Trong mạch lôgic dãy khả trình PAL (Programmable Array Logic) hình 1.13c, các điểm nối trong ma trận OR là cố định và việc nối trong ma trận AND được chương trình hoá. PAL thường được dùng trong các bộ giải mã địa chỉ.

1.4.3. Các mạch chốt, mạch lật, thanh ghi và bộ đếm

• ***Mạch chốt D*** (D-latch): Mạch chốt là một mạch số lưu trữ các giá trị 0 hoặc 1 ở lối ra của nó. Kí hiệu và bảng chân lí cho mạch chốt D như hình 1.14.

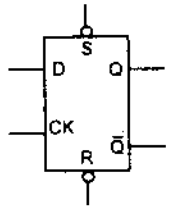


D	CK	Q	Q̄
x	0	Q_N	Q_N
0	1	0	1
1	1	1	0

Hình 1.14 Mạch chốt D.

Nếu lối vào cho phép (enable input) CK ở mức thấp, mức lôgic tại lối vào D sẽ không có tác dụng tới các lối ra Q và $\overline{Q}$. Điều này được ký hiệu bằng dấu x trong cột D. nếu lối vào cho phép ở mức lôgic cao thì mức lôgic cao hoặc thấp từ lối vào D sẽ được truyền tới lối ra Q. Nói cách khác, lối ra Q sẽ bám theo lối vào D chừng nào mà lối vào cho phép ở mức cao. Khi lối vào cho phép CK lại ở mức thấp, trạng thái của Q sẽ được chốt lại từ thời điểm đó và bất kì sự thay đổi trạng thái lối vào D cũng sẽ không ảnh hưởng được tới trạng thái lối ra Q nữa.

• ***Mạch lật*** (flip-flop): Hình 1.15 chỉ ra kí hiệu và bảng chân lí của mạch lật loại D.



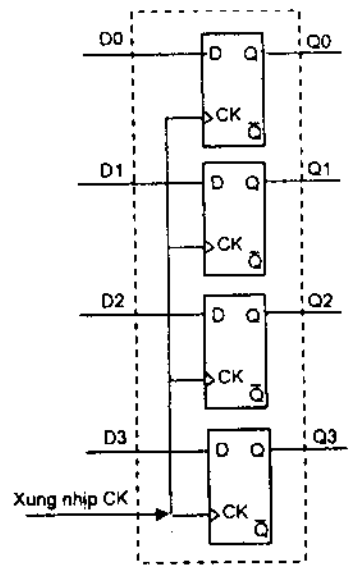
S	R	D	CK	Q	Q̄
1	1	1	↑	1	0
1	1	0	↑	0	1
1	1	x	0	Q_N	$\overline{Q_N}$
1	1	x	1	Q_N	$\overline{Q_N}$
0	1	x	x	1	0
1	0	x	x	0	1
0	0	x	x	-	-

Hình 1.15 Mạch lật D.

Ngoài ra còn có các mạch lật loại RS, JK hoặc T. Giá trị các lối ra Q và $\overline{Q}$ sẽ chỉ được cập nhật khi có một sườn lên của xung điện được đưa tới lối vào CK (kí hiệu bằng dấu ↑ trong cột CK ở bảng chân lí). Mạch lật D còn có các lối vào đặt (set) và xoá (reset). Mạch được gọi là được đặt khi lối ra Q của nó được đặt lên mức lôgic 1 và được xoá (đặt lại) khi lối ra Q được đưa về mức lôgic 0 bất chấp các trạng thái của các lối vào CK và D. Các lối vào này được hoạt động không đồng bộ với nhau.

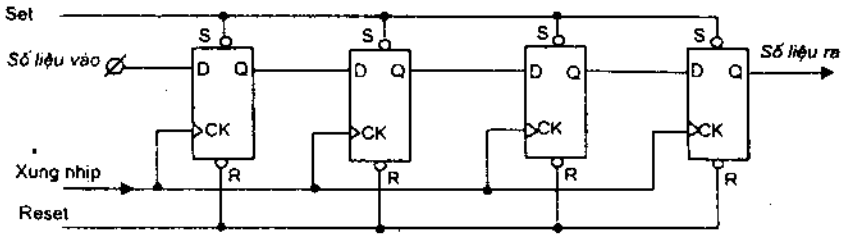
• ***Thanh ghi*** (register) là một nhóm các mạch lật D được nối song song với nhau được ghép lại thành nhóm để lưu trữ các số liệu nhị phân. Hình 1.16 là một thí dụ tạo một thanh ghi 4 bit từ 4 mạch lật D.

Một từ số liệu nhị phân bất kì trên các lối vào từ D3 đến D0 sẽ được chốt sang các lối ra Q3 đến Q0 mỗi khi có



Hình 1.16 Thanh ghi 4 bit.

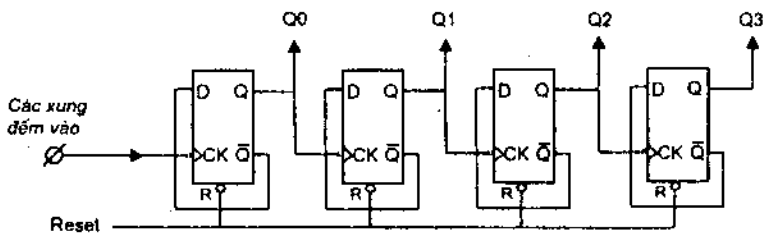
một sườn lên của xung nhịp được đưa tới lối vào xung nhịp. Từ nhị phân này sẽ được lưu trữ ở các lối ra Q cho tới khi một từ mới được đưa tới các lối vào D và lại có một sườn lên của xung nhịp khác được đặt tới lối vào xung nhịp CK. Nếu lối ra Q của một mạch lật được nối tới lối vào D của mạch lật tiếp theo thì ta có một *thanh ghi dịch* (shift register). Hình 1.17 là một thí dụ về thanh ghi dịch 4 bit. Một mức lôgic 0 hoặc 1 từ lối vào D mạch lật thứ nhất sẽ được dịch tới lối ra Q khi tác động một xung nhịp. Xung nhịp thứ 2 sẽ dịch tiếp mức lôgic này tới lối ra Q của mạch lật thứ hai v.v...

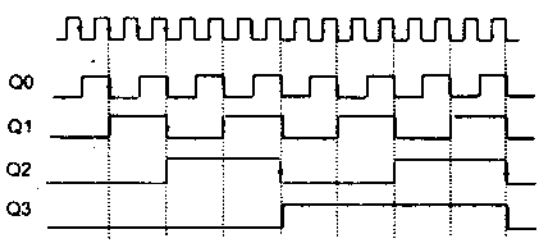


Hình 1.17 Thanh ghi dịch 4 bit.

• ***Các bộ đếm***: Các mạch lật cũng có thể được nối với nhau để tạo thành một mạch đếm mà các lối ra của nó nhảy qua các giá trị của một chuỗi đếm nhị phân mỗi khi có một sườn xuống của xung nhịp được đưa vào mạch. Hình 1.18 là một thí dụ về bộ đếm nhị phân đơn giản 4 bit được lắp trên 4 mạch lật D.

Sau khi bộ đếm được xoá về giá trị không (reset - 0000), mỗi một xung nhịp *CK* sẽ làm cho số đếm nhị phân ở các lối ra Q3 đến Q0 tăng lên 1 cho đến khi đạt giá trị cực đại là (1111) ở xung thứ $(2^4 - 1) = 15$. Tổng quát, nếu số đếm ban đầu được đặt bằng 0 thì một bộ đếm tiến, nhị phân n bit sẽ đếm được $(2^n - 1)$ xung. Giản đồ xung và các giá trị lôgic ở lối ra tương ứng với thứ tự các xung đếm cũng được nêu ra trên hình. Các bộ đếm có thể được xây dựng trên các mạch lật D, J-K, RS ... và được phân loại theo nhiều cách: đếm đồng bộ và không đồng bộ; đếm tiến, đếm lùi; đếm nhị phân, đếm thập phân v.v...



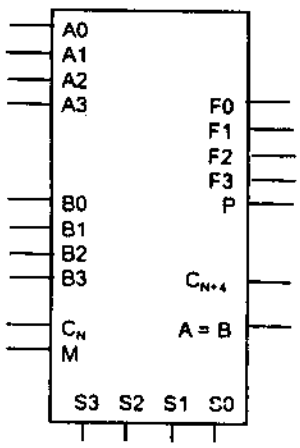


Số xung	Q3	Q2	Q1	Q0	Số xung	Q3	Q2	Q1	Q0
0	0	0	0	0	8	1	0	0	0
1	0	0	0	1	9	1	0	0	1
2	0	0	1	0	10	1	0	1	0
3	0	0	1	1	11	1	0	1	1
4	0	1	0	0	12	1	1	0	0
5	0	1	0	1	13	1	1	0	1
6	0	1	1	0	14	1	1	1	0
7	0	1	1	1	15	1	1	1	1

Hình 1.18 Bộ đếm nhị phân 4 bit.

1.4.4. Mạch tính số học và lôgic ALU

Mạch tính số học và lôgic ALU (Arithmetic Logic Unit) là một mạch điện tử số có thể thực hiện các phép tính AND, OR, cộng, trừ và thực hiện nhiều phép toán trên các từ dữ liệu nhị phân. Hình 1.19 là một thí dụ về kí hiệu và bảng chân lí của một vi mạch ALU loại 74LS181. Trong bảng chân lí, A đại diện cho từ nhị phân 4 bit được đưa tới các lối vào từ A0 đến A3 và B đại diện cho từ nhị phân 4 bit được đưa tới các lối vào từ B0 đến B3. F đại diện cho từ nhị phân 4 bit ở các lối ra từ F0 đến F3. Nếu lối vào chọn mode M ở mức cao, vi mạch sẽ thực hiện một trong 16 hàm của 2 biến vào A và B. Thí dụ, nếu M = 1; S3 = 1, S2 = 0, S1 = 1 và S0 = 1; từ nhị phân 4 bit trên các lối vào A sẽ được AND với từ nhị phân 4 bit trên các lối vào B. Kết quả sẽ được đưa ra tại các lối ra F. ALU là một đơn vị rất quan trọng trong bộ vi xử lí.



Hình 1.19 Vi mạch 74LS181.

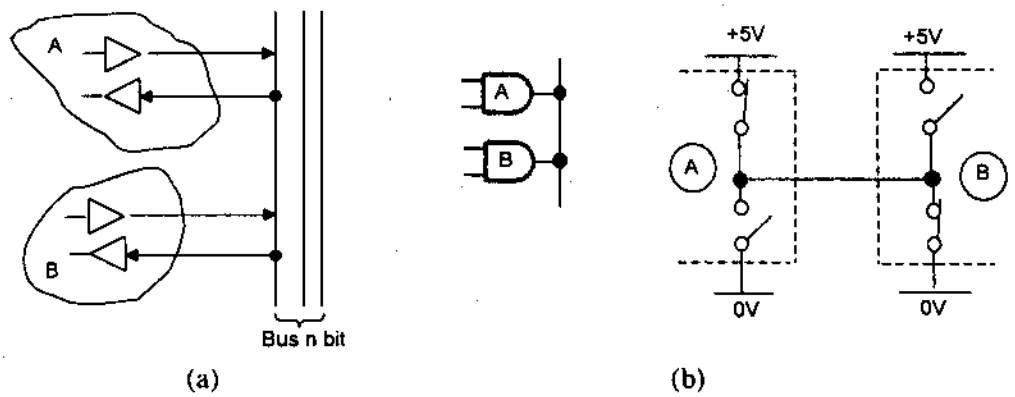
Chọn				Số liệu có mức tích cực cao		
S3	S2	S1	S0	M = H Các phép toán lôgic	$\overline{C_N} = 1$ (không có nhớ)	$\overline{C_N} = 1$ (có có nhớ)
0	0	0	0	F = A	F = A	F = A - 1
0	0	0	1	F = A + B	F = A+ B	F = (A + B) + 1
0	0	1	0	F = A . B	F = A + B	F = (A + B) + 1
0	0	1	1	F = 0	F = - 1 (mã bù 2)	F = 0
0	1	0	0	F = A. B	F = A + A.B	F = A + AB + 1
0	1	0	1	F = B	F = (A + B) + AB	F = (A + B) + AB + 1
0	1	1	0	F = A ⊕ B	F = A - B - 1	F = A - B
0	1	1	1	F = A. B	F = AB - 1	F = AB
1	0	0	0	F = A + B	F = A + AB	F = A + AB + 1
1	0	0	1	F = A ⊕ B	F = A + B	F = A + B + 1
1	0	1	0	F = B	F = (A + B) + AB	F = (A + B) + AB +1
1	0	1	1	F = A. B	F = AB - 1	F = AB
1	1	0	0	F = 1	F = A + A*	F = A + A + 1
1	1	0	1	F = A + B	F = (A + B) +A	F = (A + B) + A + 1
1	1	1	0	F = A + B	F = (A + B) + A	F = (A + B) + A + 1
1	1	1	1	F = A	F = A - 1	F = A

1.4.5. Bus và các cổng ra 3 trạng thái

Trong hệ thống máy vi tính xử lí theo kiểu tuần tự, có rất nhiều các đơn vị cần thông tin (phát và thu tín hiệu) với nhau nhưng ở mỗi thời điểm chỉ có một đơn vị được chọn làm nguồn phát (talker) và một số hạn chế các đơn vị được phép nhận tín hiệu (listener). Do vậy thay vì cho việc nối các đường truyền tín hiệu trực tiếp từng đôi một với nhau cho toàn bộ các cặp đơn vị cần thông tin, các lối ra và lối vào của từng bit trong tất cả các đơn vị đều được nối vào cùng một đường dây. Như vậy nếu máy tính sử dụng n bit dữ liệu cho một mục đích nào đó thì chỉ cần một tập n đường dây thông tin cho mục đích đó. Tập đường dây đó gọi là bus và thường được vẽ đơn giản là một đường nét đậm trên sơ đồ. Nếu phân loại theo mục đích thông tin thì

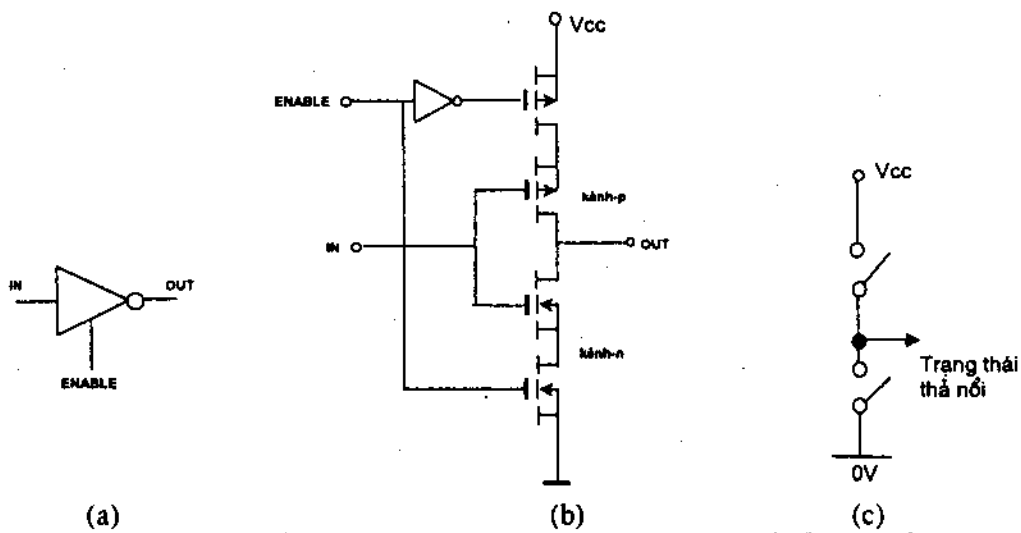
có 3 loại bus: bus số liệu dùng để chuyển tải số liệu, bus địa chỉ dùng để chuyển tải tín hiệu định địa chỉ các đơn vị cần thông tin và bus điều khiển dùng để điều khiển loại thông tin (phát/viết hay thu/đọc số liệu ...). Với các cổng lôgic thông thường (thí dụ TTL) không thể nối trực tiếp chúng vào cùng một đường dây như hình 1.20a được. Lí do là sẽ có trường hợp xảy ra tranh chấp bus (conflic) do đoản mạch.

Thực vậy, mỗi lối ra của cổng gồm 2 transistor luân phiên nhau ở 1 trong 2 trạng thái: thông bão hoà hoặc cấm và do vậy có thể coi chúng như 2 công tắc được mắc nối tiếp nhau như hình 1.20b. Transistor thông bão hoà tương đương như công tắc được đóng và transistor cấm tương đương như công tắc được mở. Trường hợp như hình vẽ 1.20b cho thấy khi cổng A ở mức lôgic cao, cổng B ở mức lôgic thấp sẽ xảy ra đoản mạch từ +5V xuống đất.



Hình 1.20 Nối các cổng vào/ra của các đơn vị vào cùng một dây trên bus (a) và xảy ra xung đột bus khi các cổng ra của A và B có mức lôgic như hình (b).

Để giải quyết khó khăn này, một cổng lôgic gọi là cổng 3 trạng thái (tri-state) được sử dụng cho các lối ra của các đơn vị nối vào bus. Thí dụ, một loại cổng đảo 3 trạng thái như hình 1.21a có sơ đồ lắp ráp trên họ lôgic CMOS như hình 1.21b. Khi lối vào cho phép E (enable) ở mức tích cực cao thì lối ra của cổng ở một trong hai trạng thái 0 hoặc 1. Khi lối vào này ở mức thấp, lối ra ở trạng thái thả nổi vì 2 transistor kênh p và n bị cấm.



Hình 1.21 Kí hiệu cổng đảo (NOT) có lối ra 3 trạng thái (a), sơ đồ cổng này lắp trên mạch CMOS với các transistor kênh p và n (b) và sơ đồ tương đương ở trạng thái thả nổi (c).

Hình 1.21c là sơ đồ tương đương khi cổng ở trạng thái trở kháng cao. Điều này dẫn đến kết quả như là đường dây nối mạch điện ra của cổng bị cắt khỏi đường dây bus. Điện thế lối ra của cổng lúc này ở vào trạng thái được thả nổi (floating), nghĩa là nếu đo mức lôgic ở đầu ra của vi mạch 3 trạng thái lúc này ta sẽ nhận được giá trị "trôi nổi" theo giá trị lôgic của bus.Cũng có thể có các cổng 3 trạng thái khác có mức tích cực của lối vào cho phép ngược lại (mức 0 cho phép hoạt động, mức 1 đưa vào trạng thái nổi).

Như vậy, nếu thiết kế các lối ra của các đơn vị nối vào bus là các cổng 3 trạng thái thì khi chọn một đơn vị là nguồn phát và phải buộc các đơn vị còn lại ở vào trạng thái có lối ra thả nổi thì sẽ không xảy ra xung đột bus. Rõ ràng, một hệ thống máy vi tính chỉ có thể làm việc được khi các cổng chốt (latch) lối vào và các cổng lối ra 3 trạng thái (tri-state) được điều khiển một cách hết sức chính xác. Lúc này các cổng ra 3 trạng thái và các bộ đệm vào (input buffer) được nối với cùng một đường dây bus.

CÂU HỎI VÀ BÀI TẬP

1.1 Nêu tóm tắt 3 lĩnh vực ứng dụng của máy tính.

1.2 Theo bạn phần cứng hay phần mềm quan trọng hơn ? Phải xử lí mối quan hệ này như thế nào cho có lợi nhất ?

1.3 Tại sao cần phải biết về phần cứng của máy tính ?

1.4 Đọc và tìm hiểu kĩ thí dụ sau về việc thực hiện một chương trình đơn giản gồm 3 câu lệnh trên máy tính:

- *Nhập vào một giá trị từ bàn phím (bàn phím có địa chỉ ở cổng 05h).*
- *Cộng thêm 7 vào giá trị vừa nhập.*
- *Hiển thị dữ liệu ra màn hình (màn hình có địa chỉ ở cổng 02h).*

Các địa chỉ và nội dung các ô nhớ được liệt kê như sau:

Địa chỉ nhớ (Hex)	Nội dung (nhị phân)	Nội dung (Hex)	Hoạt động
00100h	1110 0100	E4h	Nhập số từ ...
00101h	0000 0101	05h	cổng 05h
00102h	0000 0100	04h	cộng với ...
00103h	0000 0111	07h	07h
00104h	1110 0110	E6h	xuất ra ...
00105h	0000 0010	02h	cổng 02h

Với thí dụ này, CPU nhận các lệnh và dữ liệu từ bộ nhớ 1 byte mỗi lần. Mã nhị phân của các lệnh có vị trí tuần tự trong bộ nhớ, thí dụ bắt đầu từ địa chỉ 00100h. CPU cần một lệnh trước khi nó có thể làm bất cứ điều gì, vì vậy thao tác đầu tiên của nó là tìm và giải mã một byte lệnh từ bộ nhớ. Để làm điều này, CPU gửi đi địa chỉ của byte lệnh đầu tiên, thí dụ là 00100h, tới bộ nhớ trên bus địa chỉ. Sau đó CPU gửi đi tín hiệu đọc bộ nhớ trên bus điều khiển. Bộ nhớ khi nhận được tín hiệu đọc dữ liệu thì gửi đi byte tại địa chỉ nhận được trên bus dữ liệu. CPU đọc byte lệnh đầu tiên (E4h) từ bus dữ liệu và giải mã nó. Giải mã có nghĩa là CPU xác định dựa vào mã nhị phân đọc được thao tác nào CPU sẽ phải làm. Nó sẽ chọn lần lượt các vi lệnh cần thiết để thi hành lệnh đọc được này. Với lệnh ở ví dụ trên, CPU xác định mã đọc được là lệnh vào dữ liệu. Nhờ việc giải mã byte lệnh, CPU biết rằng nó cần thêm thông tin trước khi có thể thi hành các lệnh. Thông tin thêm mà

CPU cần là địa chỉ của cổng mà dữ liệu sẽ được nhập vào từ đó. Địa chỉ cổng được lưu trữ trong ô nhớ tiếp theo sau ô nhớ chứa mã cho lệnh nhập dữ liệu.

Để có thể nhận byte thứ hai này của lệnh, CPU gửi đi địa chỉ tuần tự tiếp theo (00101h) tới bộ nhớ. Để điều khiển thiết bị bộ nhớ, CPU lại gửi đi một tín hiệu đọc bộ nhớ trên bus điều khiển. Bộ nhớ sau đó sẽ gửi lại byte tại địa chỉ tiếp theo này trên bus dữ liệu. Khi CPU vừa mới đọc byte thứ hai, trong ví dụ là 05h, nó sẽ có đầy đủ thông tin cần thiết để thực hiện lệnh.

Để thực hiện lệnh nhập dữ liệu, CPU gửi đi địa chỉ cổng (05h) trên bus địa chỉ và gửi đi tín hiệu đọc vào/ra (I/O) trên bus điều khiển. Tín hiệu đọc vào/ra điều khiển cho thiết bị ở cổng đặt byte dữ liệu lên bus dữ liệu. CPU đọc byte dữ liệu và cất nó trong thanh ghi trong. Đến đây hoàn thành việc tìm, giải mã và thi hành lệnh đầu tiên.

CPU tiếp tục nhận lệnh kế tiếp từ bộ nhớ. Nó gửi đi địa chỉ tuần tự tiếp theo (00102h) trên bus địa chỉ và gửi tín hiệu đọc bộ nhớ trên bus điều khiển. Tín hiệu đọc bộ nhớ sẽ điều khiển thiết bị bộ nhớ đặt byte tại địa chỉ nhận được lên bus dữ liệu. CPU đọc byte lệnh này từ bus dữ liệu và giải mã nó. Nhờ byte lệnh này CPU biết rằng nó phải cộng một số vào số đang được lưu trữ trong thanh ghi trong. CPU cũng xác định được từ việc giải mã byte lệnh này rằng nó lại phải đi vào bộ nhớ để lấy byte tiếp theo của lệnh có chứa số cần cộng. Để thu được byte cần thiết này,CPU gửi địa chỉ tuần tự tiếp theo (00103h) trên bus địa chỉ và một tín hiệu đọc bộ nhớ khác trên bus điều khiển. Bộ nhớ sẽ gửi ra nội dung byte tại địa chỉ này (số 07h) trên bus dữ liệu. Khi CPU nhận được số này sẽ cộng nó với nội dung thanh ghi trong. Kết quả phép cộng sẽ được lưu trong thanh ghi trong. Đến đây hoàn thành việc tìm, giải mã và thi hành lệnh thứ hai.

CPU phải tìm và giải mã lệnh thứ ba. Để làm điều này, nó gửi đi địa chỉ tuần tự tiếp theo (00104h) trên bus địa chỉ và gửi tín hiệu đọc bộ nhớ trên bus điều khiển. Bộ nhớ sau đó sẽ xuất byte dữ liệu tại địa chỉ nhận được (E6h) trên bus dữ liệu. Từ việc giải mã byte dữ liệu này, CPU xác định được rằng nó phải làm thao tác xuất dữ liệu ra cổng. Để xuất dữ liệu ra cổng, CPU sẽ gửi đi địa chỉ tuần tự tiếp theo (00105h) trên bus địa chỉ, gửi đi tín hiệu đọc bộ nhớ trên bus điều khiển, và đọc byte (02h) dữ liệu gửi từ bộ nhớ trên bus dữ liệu. CPU bây giờ đã có đủ thông tin cần thiết để thực hiện lệnh xuất dữ liệu.

Để xuất byte dữ liệu tới cổng, CPU đầu tiên gửi đi địa chỉ của cổng lên bus địa chỉ. Tiếp theo, nó gửi đi byte dữ liệu tại thanh ghi bên trong trên bus dữ liệu. CPU sau đó gửi đi tín hiệu viết I/O trên bus điều khiển. Tín hiệu này điều khiển thiết bị ở cổng để dữ liệu từ các đường bus dữ liệu có thể truyền qua cổng và tới hiển thị LED. Khi CPU xoá bỏ tín hiệu viết I/O để tiếp tục với lệnh kế tiếp, dữ liệu sẽ còn bị chốt trong chân ra của thiết bị cổng. Dữ liệu sẽ còn bị chốt ở đó cho đến khi nguồn bị tắt hay một từ dữ liệu mới được xuất tới cổng. Điều này là rất quan trọng bởi vì nó có nghĩa là máy tính không phải liên tục xuất giá trị để duy trì ở lối ra. Tất cả các bước mô tả ở trên dường như là một khối lượng lớn công việc chỉ để nhập giá trị từ bàn phím, cộng thêm 7 vào giá trị vừa nhập và hiển thị kết quả lên màn hình. Tuy nhiên một máy vi tính đơn giản cũng có thể thực hiện tất cả các bước trên chỉ trong vài micro giây. Có thể tóm tắt vận hành của bus trong quá trình chạy chương trình theo các bước sau:

1. *Máy vi tính nhận từng lệnh chương trình một cách tuần tự, giải mã và thực hiện nó.*
2. *CPU trong máy vi tính nhận lệnh hoặc đọc dữ liệu từ bộ nhớ bằng cách gửi địa chỉ trên bus địa chỉ và tín hiệu đọc bộ nhớ trên bus điều khiển. Bộ nhớ gửi đi lệnh hoặc từ dữ liệu tại địa chỉ nhận được tới CPU trên bus dữ liệu.*
3. *CPU viết một từ dữ liệu lên bộ nhớ bằng cách gửi địa chỉ lên bus địa chỉ, gửi từ dữ liệu trên bus dữ liệu và gửi tín hiệu viết bộ nhớ trên bus điều khiển.*

4. *Để đọc dữ liệu từ cổng, CPU gửi đi địa chỉ cổng trên bus địa chỉ và gửi tín hiệu đọc I/O tới thiết bị cổng trên bus điều khiển. Dữ liệu từ cổng tới CPU trên bus dữ liệu.*
5. *Để viết dữ liệu vào cổng, CPU gửi địa chỉ cổng lên bus địa chỉ, gửi dữ liệu sẽ được viết vào cổng lên bus dữ liệu, và gửi tín hiệu viết I/O tới thiết bị cổng trên bus điều khiển.*

Hãy thử mô tả các bước thực hiện 3 lệnh trên bằng một sơ đồ gồm CPU, bộ nhớ và các cổng vào/ra cùng các đường lệnh.

1.5 Biến đổi các số nguyên thập phân sau thành dạng nhị phân:

(a) 15 (b) 42 (c) 235 (d) 4090

(e) 40 900 (f) 65 530

1.6 Biến đổi các số nhị phân sau thành các số thập phân:

(a) 110 (b) 1110110 (c) 11011 (d) 11111110111

1.7 Biến đổi các số thập phân sau thành các số nhị phân (dấu chấm là dấu phảy trong tiếng Việt):

(a) 1.5 (b) 1024.0625 (c) 3.141592

Biến đổi các số nhị phân sau thành các số thập phân:

(a) 1.1 (b) 0.001 (c) 11011.101010

(d) 111.111111

1.8 Giải mã Huffman sau đây, giả thiết rằng các mã hợp lệ là P = 0, Q = 10, R = 110 và S = 111. Bao nhiêu bit cần được đòi hỏi nếu P, Q, R và S đã được mã hoá lần lượt là 00, 01, 10, 11 ? 000001110111000000101111111101010001111100010

1.9 Biểu diễn các số bù hai 8 bit của các số thập phân sau:

(a) -4 (b) -25 (c) -42

(d) -128 (e) -111 (f) 0

1.10 Cộng 2 số nhị phân 10.125 và 32.1 bằng số có dấu chấm động với từ dài 16 bit gồm: 6 bit mũ, 10 bit định trị, độ dịch bằng $2^{6-1} = 32$.

1.11 Một số dấu chấm động chuẩn IEEE dài 32 bit có định dạng: 1 bit dấu, 8 bit mũ có dịch, 23 bit phần thập phân của định trị. Một số $N = (-1^S) \times 2^{E-127} \times 1F$. Thực hiện phép trừ 42.6875 - 0.09375.

1.12 Một máy tính có độ dài từ máy là 24 bit hoạt động với số có dấu chấm động gồm 8 bit mũ có dịch, 16 bit thập phân định trị. Hãy viết dải các số có thể được biểu diễn và độ chính xác của chúng.

CHƯƠNG 2. CẤU TRÚC CỦA VI XỬ LÍ

2.1. BỘ XỬ LÍ TRUNG TÂM CPU

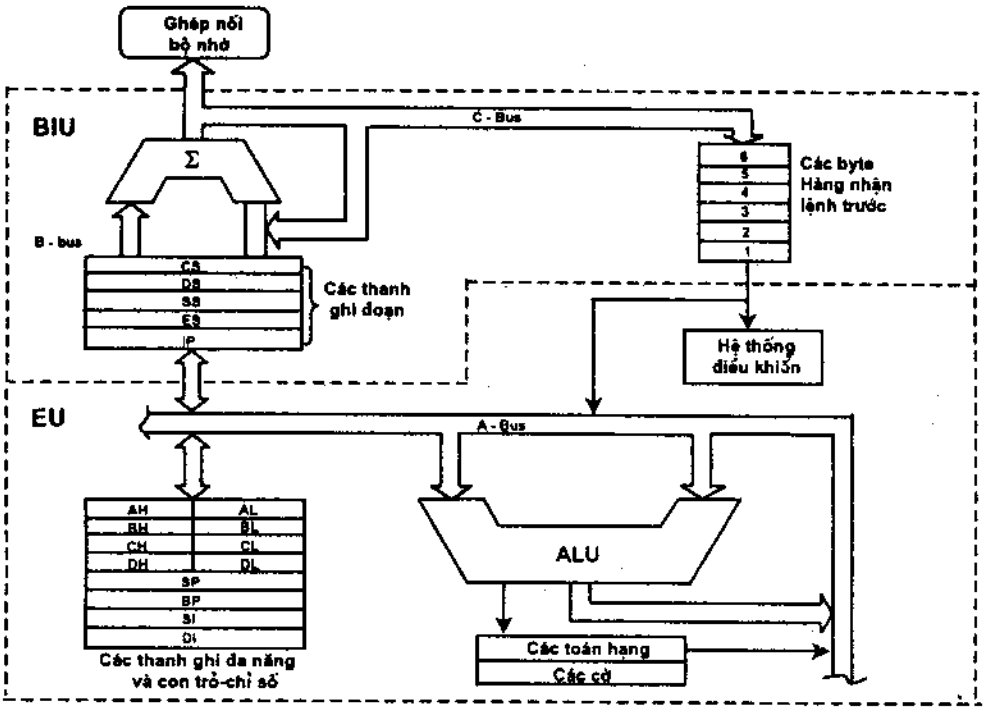
Vi xử lí là bộ xử lí trung tâm CPU được chế tạo trên một chip. Nó là linh kiện quan trọng nhất của hệ máy vi tính. Vì vậy, khảo sát các nguyên tắc hoạt động của nó là điều cần thiết. Đáp ứng lại những yêu cầu ứng dụng trong thực tế, các nhà sản xuất vi xử lí đã phát triển theo ba hướng chính trong suốt những năm qua là:

- *Vi điều khiển chuyên dụng* (embedded or dedicated controllers): Một hướng phát triển là các chíp điều khiển chuyên dụng. Những chíp này được sử dụng để điều khiển các máy thông minh như lò vi sóng, máy giặt quần áo, máy khâu, hệ thống đánh lửa tự động, máy tiện kim loại v.v... Hãng Texas Instrument đã sản xuất hàng triệu vi xử lí 4-bit loại TMS-1000 như vậy cho những ứng dụng kiểu này. Năm 1976, Intel giới thiệu 8048 gồm có CPU 8-bit, RAM, ROM và vài cổng vào/ra trong một chíp 40 chân. Các nhà sản xuất khác cũng theo sau với các sản phẩm tương tự. Các chip kiểu này thường được gọi là các *vi điều khiển* (microcontroller). Vài chíp có bán hiện nay thuộc nhóm này, Intel 8051và Motorola MC6801 chẳng hạn, chứa các bộ đếm có thể lập trình và cổng nối tiếp (UART) cũng như là CPU, ROM, RAM, và các cổng vào/ra song song. Một vi điều khiển như Intel 8096, chứa một CPU 16-bit, các mạch ROM, RAM, UART, các cổng và một bộ chuyển đổi tương tự-số 10-bit.
- *Vi xử lí bit-slice*: Hướng phát triển thứ hai là các vi xử lí *bit slice*. Các chip này cho phép có thể được ghép với nhau để xây dựng một CPU theo ý muốn. Một ví dụ là họ 2900 AMD (Advanced Micro Devices). Họ này có ALU 4-bit, bộ hợp kênh, bộ tuần tự và các phần khác. Các vi xử lí phân mảnh bit này (bit-slice) có thể được nối song song với nhau để làm việc được với các từ dữ liệu 8-bit, 16-bit hay 32-bit. Nói cách khác, các nhà thiết kế có thể thêm vào bao nhiêu bộ vi xử lí kiểu này tuỳ ý nếu cần cho các ứng dụng đặc biệt. Ngoài việc thiết kế theo ý muốn phần cứng của CPU, người dùng còn có thể tạo ra tập lệnh CPU của riêng mình bằng việc sử dụng vi mã (microcode).
- *Vi xử lí đa năng*: Hướng chính thứ ba của tiến trình phát triển vi xử lí là mạch vi xử lí đa năng đã đem lại cho các máy vi tính hiện nay hầu hết hoặc toàn bộ sức mạnh tính toán của máy tính mini trước đó. Sau khi Motorola cho ra đời MC6800; đến năm 1978, Intel cho ra đời 8086 là bộ vi xử lí 16-bit hoàn toàn và tới nay là các chíp vi xử lí Pentium.

2.1.1. Sơ đồ khối của vi xử lí đa năng 8086

Nếu nhìn nhận cấu trúc của máy vi tính theo sơ đồ phân mức hình 1.4 thì có nhiều xuất phát điểm để khảo sát cấu trúc của vi xử lí. Lẽ ra là phải đi từ mức 1 trở lên là mức gồm các mạch điện tử số nhưng để gần gũi hơn với quan điểm của người sử dụng ta xuất phát khảo sát từ mức 3 trở xuống khi sử dụng sơ đồ của vi xử lí 8086 do chính hãng Intel công bố lần đầu như hình 2.1. Nhắc lại, mức 3 là mức lệnh máy hay còn gọi là các lệnh vĩ mô (macro instruction). Chính các hãng sản xuất vi xử lí thường cung cấp cho người dùng các tập lệnh máy của mức này và các lệnh hợp ngữ cũng như các lệnh được viết bằng các

ngôn ngữ bậc cao được các chương trình dịch (giải mã) ra thành các lệnh máy này khi thực thi các lệnh trong chương trình.



Hình 2.1 Sơ đồ khối của vi xử lí 8086 nhìn từ mức 3 trong cấu trúc phân mức.

8086 được chia thành hai phần có chức năng độc lập là *đơn vị ghép nối bus* BIU (bus interface unit) và *đơn vị thi hành lệnh* EU (excution unit). Việc chia sẻ công việc giữa hai khối này làm tăng tốc độ xử lí của vi xử lí lên.

BIU gửi địa chỉ, tìm-nạp và giải mã lệnh từ bộ nhớ, đọc dữ liệu từ các cổng và bộ nhớ, viết dữ liệu lên cổng và bộ nhớ. Nói cách khác, BIU điều khiển và quản lí tất cả quá trình truyền dữ liệu, địa chỉ trên bus cho khối thi hành lệnh EU.

Đơn vị thi hành lệnh EU của 8086 chỉ cho BIU biết nơi phải tìm giải mã lệnh hay dữ liệu và cho thi hành các lệnh đó. Trong EU có mạch điều khiển CU, mạch tính số học-lôgic ALU và các thanh ghi.

Khi chân **RESET** của 8086 được đặt ở mức cao ít nhất bốn nhịp đồng hồ thì vi xử lí được khởi động lại ngay khi chân này trở về mức thấp. Lúc đó các thanh ghi được đặt lại và 8086 thực thi chương trình BIOS cho quá trình khởi động (booting) máy tính.

Như các vi xử lí khác, 8086 có một chức năng mạnh và mềm dẻo trong quá trình chạy chương trình, đó là ***khả năng thực hiện ngắt*** (interrupt). Ngắt là thao tác dừng chương trình chính đang chạy để thực hiện một chương trình khác gọi là *chương trình con xử lí ngắt*. Chương trình con này được kết thúc bằng lệnh IRET (trở về chỗ bị ngắt). Nó ra lệnh cho CPU quay về thực hiện tiếp chương trình chính đúng từ chỗ mà nó bị ngắt. Các ngắt không chỉ có ý nghĩa quan trọng đối với phần mềm mà cả với phần cứng. Có 3 loại ngắt:

Ngắt mềm được thực hiện bằng một lời gọi trong một chương trình ngôn ngữ máy, đó là lệnh INT với số ngắt kèm theo. Thí dụ khi chương trình gặp lời gọi INT 5, vi xử lí sẽ nhảy vào

thực hiện một chương trình con phục vụ ngắt để in trang màn hình. Các ngắt mềm cho phép gọi các chương trình phụ của hệ điều hành. Ngắt mềm còn có thể được gọi từ ngôn ngữ bậc cao lúc đó nó sẽ được dịch ra thành lệnh hợp ngữ INT.

Ngắt cứng được thực hiện khi các chip điện tử trong máy tính hoặc ngoại vi phát ra tín hiệu yêu cầu phục vụ ngắt và truyền nó tới chân INTR hoặc NMI của 8086. Thí dụ, ấn hay nhả bàn phím sẽ gây nên ngắt cứng số 9 (ngắt bàn phím), chương trình xử lí ngắt sẽ phản ứng bằng cách đưa mã phím đó vào vùng đệm của bàn phím. Nếu vùng đệm đầy, sẽ sinh ra tiếng bíp. Ngắt cứng còn được chia làm hai loại: *ngắt có thể bị che* là loại ngắt có thể bị cấm bằng lệnh hợp ngữ CLI và *ngắt không thể bị che* là loại ngắt luôn được thực hiện kể cả khi ngắt này được gọi ngay sau lệnh CLI.

Ngoại lệ được sinh ra bởi chính các hoạt động bên trong vi xử lí (thí dụ như xảy ra lỗi tính toán). Nếu xuất hiện ngoại lệ, một ngắt mềm với số ngắt được vi xử lí cung cấp sẽ xảy ra.

Xét một vài phần chức năng trong khối thi hành lệnh EU.

a, Mạch điều khiển, bộ giải mã lệnh và khối tính số học - lôgic ALU

EU có mạch điều khiển chỉ đạo các hoạt động bên trong của CPU. Bộ giải mã trong EU dịch các lệnh nhận từ bộ nhớ thành chuỗi các hoạt động mà EU sẽ thực hiện. EU có khối tính số học-lôgic ALU 16-bit cho phép thực hiện các phép toán cộng, trừ, AND, OR, XOR, tăng, giảm, bù, dịch các số nhị phân.

b, Thanh ghi cờ : Cờ là một mạch lật (flip-flop) được dùng để chỉ thị trạng thái sinh ra bởi quá trình thực hiện lệnh hay điều khiển các phép toán của đơn vị thi hành lệnh EU. Thanh ghi cờ 16-bit trong EU chứa 9 cờ tích cực. Sáu trong 9 cờ này được sử dụng để chỉ thị trạng thái tạo ra bởi câu lệnh. Ví dụ, một mạch lật được gọi là cờ nhớ sẽ được đặt lên 1 nếu tổng của hai số nhị phân 16-bit có nhớ ở vị trí bit cao nhất. Nếu không có nhớ ở bit cao nhất trong phép cộng thì cờ nhớ sẽ được đặt giá trị 0. Như vậy, nhờ vào cờ EU xác định rất nhanh là phép toán có nhớ.

Sáu cờ trạng thái trong nhóm này là *cờ nhớ* (CF), *cờ chẵn lẻ* (PF), *cờ nhớ phụ* (AF), *cờ zero* (ZF), *cờ dấu* (SF) và *cờ tràn* (OF). Tên của những cờ này cũng đem lại gợi ý về trạng thái nào sẽ ảnh hưởng đến các cờ. 8086 sẽ kiểm tra những cờ này để xác định xem hai hành động luân phiên nào sẽ được thi hành trong quá trình thực hiện lệnh.

Ba cờ còn lại trong thanh ghi cờ được sử dụng để điều khiển các phép toán khác nào đó của bộ xử lí. Ba cờ này khác so với sáu cờ miêu tả ở trên trong cách đặt hoặc đặt lại giá trị của chúng. Sáu cờ trạng thái được đặt (set) hay xoá (clear) bởi EU, cơ bản dựa trên kết quả của vài phép toán lôgic hay phép toán số học. Cờ điều khiển được đặt hay xoá một cách có chủ tâm với các lệnh riêng được đặt trong chương trình của người dùng. Ba cờ điều khiển là *cờ bẫy* (TF), được sử dụng cho chạy từng bước chương trình; *cờ ngắt* (IF), dùng để cho phép hay cấm các ngắt của chương trình và *cờ hướng* (DF), dùng với các lệnh chuỗi.

c, Các thanh ghi đa năng: EU có 8 thanh ghi đa năng với tên là AH, AL, BH, BL, CH, CL, DH và DL. Những thanh ghi này có thể được sử dụng riêng rẽ cho việc lưu trữ tạm thời dữ liệu 8-bit. Các cặp xác định thanh ghi đa năng có thể được sử dụng cùng nhau để lưu trữ từ dữ liệu 16-bit. Các cặp thanh ghi chấp nhận được là AH và AL, BH và BL, CH và CL, DH và DL. Cặp AH-AL được xem như thanh ghi AX, BH-BL là BX, CH-CL là CX và DH-DL là DX.

Ưu điểm của việc sử dụng thanh ghi bên trong cho việc lưu trữ tạm thời dữ liệu là cho phép CPU có thể truy cập dữ liệu nhanh hơn rất nhiều so với việc phải truy cập bộ nhớ ngoài.

Xét đặc điểm của đơn vị ghép nối bus BIU.

d, Hàng nhận lệnh trước PQ: Trong khi EU đang giải mã một lệnh hay thực hiện một lệnh mà không cần sử dụng bus, BIU có thể nhận tới 6 byte lệnh cho lệnh tiếp theo. Nó sẽ lưu trữ các byte lệnh nhận trước này trong một *thanh ghi vào trước-ra trước* FIFO (first-in—first-out) gọi là *Hàng nhận lệnh trước* PQ (Prefetch Queue). Khi EU đã sẵn sàng thực hiện lệnh tiếp theo, nó chỉ đơn giản đọc byte lệnh từ PQ trong BIU. Điều này nhanh hơn nhiều so với việc gửi địa chỉ tới bộ nhớ hệ thống và đợi bộ nhớ gửi lại byte lệnh tiếp theo. Quá trình này tương tự như cách người phụ xây nhận gạch trước và giữ thành hàng gạch nối nhau để người thợ xây chính có thể chỉ vươn tay ra và tóm viên gạch khi cần. Ngoại trừ trường hợp lệnh CALL và JMP, ở đây dữ liệu trong PQ phải bị "tống ra" và sau đó được nạp bắt đầu từ địa chỉ mới. Sự sắp xếp theo kiểu hàng nhận lệnh trước (Prefet Queue) làm tăng tốc đáng kể quá trình xử lí. Nhận lệnh tiếp theo trong khi đang thực hiện lệnh hiện thời được gọi là *cơ chế đường ống* (pipelining).

e, Các thanh ghi đoạn: BIU của 8086 gửi đi địa chỉ 20 bit, vì vậy nó có thể định địa chỉ lên tới 2^{20} byte hay 1.048.576 byte nhớ vật lí trong bộ nhớ hoặc ngoại vi. Tuy nhiên, 8086 lại tổ chức không gian nhớ này ra thành các *đoạn nhớ lôgic*. Mỗi đoạn dài cố định 64 kbyte, địa chỉ của byte nhớ thấp nhất trong đoạn được lấy làm địa chỉ đoạn. Hai đoạn nhớ kế cận như vậy cách nhau 16 byte. Mỗi byte nhớ trong đoạn được xác định bởi một *độ lệch* (offset) là khoảng cách được tính từ byte nhớ đó đến địa chỉ đoạn.

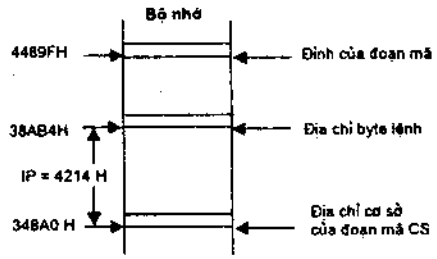
Như vậy, một cặp thông số *địa chỉ đoạn* : *độ lệch* (segment : offset) xác định địa chỉ lôgic của một byte nhớ vật lí. Do vậy, địa chỉ vật lí thực sự của byte nhớ đó sẽ được BIU tính bằng *segment* × *16* + *offset*. Vì mỗi đoạn dài 2^{16} = 64 k nên các thanh ghi đoạn 16 bit trong BIU được sử dụng để lưu giữ 16 bit cao của địa chỉ đoạn nhớ. Còn các thanh ghi đa năng hay thanh ghi IP sẽ lưu giữ 16 bit địa chỉ độ lệch của byte nhớ cần truy cập trong đoạn đó. 8086 dùng 4 thanh ghi đoạn để thông tin với 4 loại đoạn nhớ tại các thời điểm khác nhau. Bốn thanh ghi đoạn là *thanh ghi đoạn mã* CS (code segment), *thanh ghi đoạn ngăn xếp* SS (stack segment), *thanh ghi đoạn phụ* ES (extra segment) và *thanh ghi đoạn dữ liệu* DS (data segment).

Bốn thanh ghi đoạn có thể tách biệt nhau, hay với chương trình nhỏ mà không cần cả 64 kbyte trong toàn đoạn thì chúng có thể chồng lên nhau. Nhắc lại, thanh ghi đoạn được sử dụng để giữ 16 bit cao của địa chỉ bắt đầu cho mỗi đoạn. Ví dụ, thanh ghi đoạn mã chứa 16 bit cao của địa chỉ bắt đầu cho đoạn mã mà từ đó BIU nhận byte mã lệnh. BIU luôn luôn chèn thêm các số 0 cho 4 bit thấp nhất của 20 bit địa chỉ bắt đầu của đoạn. Nếu thanh ghi đoạn mã có giá trị 348Ah thì đoạn mã sẽ bắt đầu ở địa chỉ 348A0h. Nói cách khác, một đoạn 64 kbyte có thể được đặt ở vị trí bất kì trong không gian địa chỉ 1 Mbyte, nhưng đoạn sẽ luôn luôn bắt đầu ở địa chỉ với 4 bit thấp nhất bằng 0. Sự ép buộc này về vị trí của đoạn là cần thiết cho việc lưu trữ và thao tác các số 16-bit khi làm việc với địa chỉ bắt đầu của đoạn. Phần địa chỉ bắt đầu của đoạn được lưu trữ trong thanh ghi đoạn thường được gọi là *địa chỉ cơ sở*.

f, Ngăn xếp: là một phần của bộ nhớ được đặt ra ngoài để lưu trữ địa chỉ và dữ liệu trong khi thực hiện chương trình con. Thanh ghi đoạn ngăn xếp được dùng để giữ 16 bit cao của địa chỉ bắt đầu cho đoạn ngăn xếp của chương trình. Thanh ghi đoạn mở rộng (ES) và thanh ghi đoạn dữ liệu (DS) được dùng để giữ 16 bit cao của địa chỉ bắt đầu của hai đoạn bộ nhớ dùng cho dữ liệu.

g, Con trỏ lệnh: Xét đặc điểm tiếp theo trong BIU là thanh ghi con trỏ lệnh IP. Cũng giống như đã được thảo luận trước đây, thanh ghi đoạn mã giữ 16 bit cao của địa chỉ bắt đầu

thanh ghi mà từ vị trí này BIU nhận byte mã lệnh. Thanh ghi con trỏ lệnh giữ địa chỉ 16-bit hay còn gọi là địa chỉ offset (độ lệch) của byte lệnh tiếp theo nằm bên trong đoạn mã này. Giá



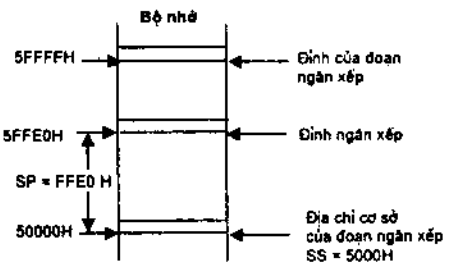
CS		3	4	8	A	0
IP	+		4	2	1	4
Địa chỉ vật lí		3	8	A	B	4

Hình 2.2 Tính địa chỉ vật lí từ địa chỉ lôgic CS:IP.

trị được chứa trong IP gọi là offset vì giá trị này cộng với địa chỉ đoạn cơ sở trong CS tạo thành địa chỉ vật lí 20-bit mà sẽ được gửi đi bởi BIU. Thanh ghi CS chỉ đến vị trí bắt đầu của đoạn mã hiện thời. IP chứa độ lệch từ địa chỉ cơ sở tới byte lệnh tiếp theo sẽ được nhận và giải mã. Như vậy để tạo nên một địa chỉ vật lí 20 bit cần truy cập, BIU dịch trái địa chỉ cơ sở đoạn trong CS đi 4 bit bằng cách chèn 4 số 0 vào sau bit có ít ý nghĩa nhất rồi cộng với địa chỉ offset 16 bit trong IP hoặc trong thanh ghi đa năng. Thí dụ, nếu thanh ghi CS chứa 348Ah thì địa chỉ cơ sở của đoạn mã là 348Ah. Khi BIU cộng thêm độ lệch 4214h trong IP vào địa chỉ cơ sở thì kết quả là địa chỉ vật lí 20-bit 38AB4h như sau: 348A0h + 04214h = 38AB4h. Một cách biểu diễn cho địa chỉ vật lí 20-bit là *CS:IP*. Thí dụ, địa chỉ vật lí trong trường hợp trên là 348A : 4214. Tóm lại, thanh ghi CS chứa 16 bit cao của địa chỉ cơ sở đoạn, thanh ghi con trỏ lệnh IP chứa địa chỉ offset 16-bit để xác định vị trí byte lệnh tiếp theo sẽ được nhận và giải mã. Cộng giá trị trong IP với địa chỉ đoạn cơ sở sẽ tạo ra địa chỉ vật lí thực sự. Bất cứ khi nào 8086 truy cập bộ nhớ, BIU cộng địa chỉ offset vào địa chỉ đoạn cơ sở. Ta xem xét cách 8086 sử dụng thanh ghi ngăn xếp và thanh ghi con trỏ ngăn xếp để định địa chỉ vật lí.

h, Thanh ghi đoạn ngăn xếp (SS) và thanh ghi con trỏ ngăn xếp (SP): Ngăn xếp là một phần bộ nhớ đặt riêng để lưu trữ địa chỉ dữ liệu khi đang thực hiện chương trình con. 8086 cho phép đặt riêng một đoạn ngăn xếp 64 kbyte. 16 bit cao của địa chỉ đoạn này được lưu giữ trong thanh ghi đoạn ngăn xếp SS. Thanh ghi con trỏ ngăn xếp SP trong đơn vị thực hiện lệnh chứa địa chỉ offset 16-bit. Byte nhớ dùng để lưu trữ từ cuối cùng gọi là đỉnh của ngăn xếp.

Địa chỉ vật lí dùng cho đọc và viết vào ngăn xếp được tính bằng cách cộng nội dung thanh ghi con trỏ lệnh vào địa chỉ đoạn cơ sở trong SS. Thí dụ, SS chứa địa chỉ cơ sở 5000h như hình 2.3, khi cộng thêm FFE0h trong SP vào thì được kết quả là địa chỉ vật lí của đỉnh ngăn xếp 5FFE0h. Địa chỉ vật lí có thể biểu diễn dưới dạng một số 5FFE0h hay dạng SS:SP (5000:FFE0).



SS		5	0	0	0	0
SP	+		F	F	E	0
Địa chỉ vật lí của đỉnh ngăn xếp		5	F	F	E	0

Hình 2.3 Tính địa chỉ vật lí của đỉnh ngăn xếp từ địa chỉ lôgic SS:SP.

i, Thanh ghi con trỏ và thanh ghi chỉ số trong đơn vị thi hành lệnh EU: Ngoài thanh ghi con trỏ ngăn xếp, đơn vị thi hành lệnh còn có thanh ghi con trỏ cơ sở BP 16-bit và thanh ghi chỉ số nguồn DI 16-bit. Các thanh ghi này dùng cho lưu trữ số liệu tạm thời giống như các thanh ghi đa năng. Tuy nhiên, chúng được dùng chính là để lưu trữ địa chỉ offset 16-bit của từ số liệu trong một đoạn, ví dụ SI lưu trữ địa chỉ offset của từ số liệu trong đoạn dữ liệu. Địa chỉ vật lí của dữ liệu trong bộ nhớ trong trường hợp này tính bằng cách cộng nội dung thanh ghi trong SI với địa chỉ cơ sở đoạn trong thanh ghi DS.

2.1.2. Truy cập dữ liệu trực tiếp và truy cập qua thanh ghi

Trong thảo luận trước về BIU của 8086, ta đã miêu tả cách 8086 truy cập một byte dữ liệu khi sử dụng nội dung thanh ghi CS và IP và 8086 truy cập ngăn xếp sử dụng nội dung thanh ghi SS và SP. Có một vài cách khác nhau mà 8086 có thể truy cập dữ liệu mà nó cần sử dụng. Các cách khác nhau mà bộ xử lí có thể truy cập dữ liệu gọi là các *chế độ (mode) địa chỉ*. Trong câu lệnh hợp ngữ, chế độ định địa chỉ được chỉ ra ngay trong câu lệnh. Ta sẽ sử dụng lệnh MOV của 8086 để minh hoạ vài chế độ định địa chỉ của 8086. Lệnh MOV có dạng:

MOV <đích>,<nguồn>

Trong đó <đích> là toán hạng đích và <nguồn> là toán hạng nguồn. Khi được thực hiện, lệnh này sẽ copy một từ hay một byte từ vị trí nguồn tới vị trí đích. Nguồn có thể là một số được viết trực tiếp trong câu lệnh, là một thanh ghi xác định, hay một ô nhớ được chỉ ra bằng một trong 24 cách khác nhau. Nguồn và đích không thể cùng là ô nhớ trong một câu lệnh.

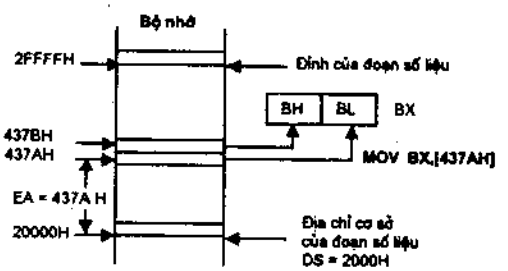
Chế độ định địa chỉ trực tiếp: Giả sử trong chương trình cần đặt số 437Bh vào thanh ghi CX. Lệnh MOV CX,437Bh sẽ thực hiện thao tác này. Khi thực hiện, lệnh sẽ đặt số Hex 437Bh trực tiếp lên thanh ghi 16-bit CX. Điều này được gọi là truy cập bộ nhớ trực tiếp vì số được nạp vào thanh ghi CX sẽ được đặt vào hai ô nhớ ngay sau mã lệnh MOV. Quá trình này tương tự như cách địa chỉ cổng được đặt vào bộ nhớ ngay sau mã lệnh nhập dữ liệu.

Chế độ định địa chỉ thanh ghi: Trong chế độ này, thanh ghi là toán hạng nguồn của câu lệnh. Ví dụ, lệnh MOV CX,AX sẽ copy nội dung thanh ghi AX vào thanh ghi CX. Nhớ rằng vị trí đích là trước dấu phẩy trong câu lệnh và vị trí nguồn là sau dấu phẩy. Và nội dung của AX chỉ copy vào CX chứ không phải di chuyển thực sự. Nói cách khác, nội dung trước của thanh ghi CX bị viết đè lên nhưng nội dung AX không đổi. Ví dụ, CX chứa 2A84h và AX chứa 4971h, sau khi lệnh MOV CX,AX thực hiện CX chứa 4791h và AX vẫn chứa 4971h. Có thể MOV bất cứ thanh ghi 16 bit nào vào thanh ghi 16 bit khác và thanh ghi 8 bit vào thanh ghi 8 bit bất kì. Tuy nhiên không thể dùng lệnh MOV CX,AL bởi vì đây là cố gắng copy thanh ghi nguồn 8 bit vào thanh ghi đích 16 bit.

2.1.3. Truy cập dữ liệu trong bộ nhớ

Các chế độ định địa chỉ bộ nhớ: Để truy cập dữ liệu trong bộ nhớ, 8086 phải tạo ra địa chỉ vật lí 20-bit. 8086 làm điều này bằng cách cộng giá trị địa chỉ hiệu dụng 16 bit vào địa chỉ cơ sở đoạn là số 16 bit một trong bốn thanh ghi đoạn. ***Địa chỉ hiệu dụng*** (EA: effective address) là độ lệch (displacement hay offset) của toán hạng mong muốn tính từ địa chỉ cơ sở đoạn. Trong các trường hợp, bất cứ địa chỉ cơ sở đoạn nào cũng dùng được, nhưng thông dụng nhất là đoạn dữ liệu. Hình 2.4 chỉ ra dạng đồ hoạ mà địa chỉ hiệu dụng EA được cộng vào địa chỉ cơ sở đoạn dữ liệu để chỉ ra toán hạng trong bộ nhớ và cách tính 20 bit địa chỉ vật lí bởi BIU. Địa chỉ vật lí bắt đầu của đoạn dữ liệu trong hình 2.4 là 20000h. BIU cộng thêm địa chỉ hiệu dụng 437Ah

vào địa chỉ cơ sở đoạn dữ liệu 20000h để tạo ra địa chỉ vật lí gửi tới bộ nhớ. BIU gửi địa chỉ vật lí 20-bit 2437Ah tới bộ nhớ. Địa chỉ vật lí hoặc viết dưới dạng số 2437Ah hay dạng segment base:offset (2000: 437A).



DS		2 0 0 0 0
EA	+	4 3 7 A
Địa chỉ vật lí		2 4 3 7 A

Hình 2.4 Tính địa chỉ vật lí từ nội dung thanh ghi đoạn và địa chỉ hiệu dụng EA.

Đơn vị thi hành lệnh EU tính toán địa chỉ hiệu dụng cho toán hạng dựa vào thông tin ghi trong câu lệnh. Có thể chỉ cho EU sử dụng con số trong lệnh như là địa chỉ hiệu dụng, sử dụng nội dung thanh ghi xác định như địa chỉ hiệu dụng, hay tính toán địa chỉ hiệu dụng bằng cách cộng số trong lệnh với nội dung của một hay hai thanh ghi xác định. Trong các giáo trình về hợp ngữ có chỉ ra những cách xác định địa chỉ hiệu dụng.

Chế độ định địa chỉ bộ nhớ trực tiếp: Với chế độ định địa chỉ bộ nhớ đơn giản nhất, địa chỉ hiệu dụng chỉ là số 16 bit được viết trực tiếp trong câu lệnh. Lệnh MOV BL, [437Ah] là một ví dụ. Ngoặc vuông xung quanh số 437Ah là viết tắt cho câu "nội dung ô nhớ tại độ lệch tính từ địa chỉ cơ sở là". Khi được thực hiện, lệnh này sẽ copy "nội dung ô nhớ tại độ lệch tính từ địa chỉ cơ sở đoạn dữ liệu là " 437Ah vào thanh ghi BL. BIU tính toán địa chỉ ô nhớ vật lí 20-bit bằng cách cộng thêm địa chỉ hiệu dụng 437Ah vào địa chỉ cơ sở đoạn dữ liệu. Chế độ định địa chỉ này được gọi là trực tiếp vì độ dịch chuyển của toán hạng được chỉ ra trực tiếp trong câu lệnh. Độ lệch trong câu lệnh sẽ được cộng vào cơ sở đoạn dữ liệu trừ khi yêu cầu BIU cộng nó với cơ sở đoạn khác.

Một thí dụ khác về chế độ định địa chỉ trực tiếp là lệnh MOV BX, [437Ah]. Khi thực hiện, lệnh này copy từ 16 bit từ bộ nhớ vào thanh ghi BX. Bởi vì mỗi địa chỉ của 8086 đại diện cho một byte vậy từ phải lấy từ hai ô nhớ. Byte tại độ lệch 437Ah so với cơ sở đoạn dữ liệu sẽ được copy vào BL, nội dung của địa chỉ cao hơn tiếp theo, độ lệch 437Bh, sẽ được copy vào thanh ghi BH như hình 2.4 chỉ ra. Dựa vào mã lệnh, 8086 sẽ tự động xác định số byte mà nó phải truy cập trong bộ nhớ. Một điểm quan trọng ở đây là 8086 luôn luôn lưu trữ byte thấp của từ trong địa chỉ thấp và byte cao của từ trong địa chỉ cao.

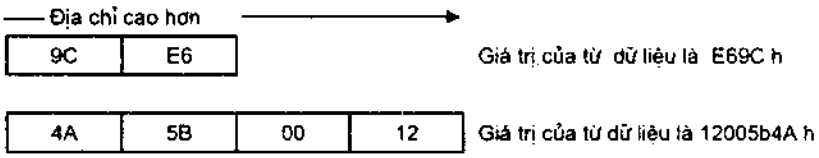
Hai thí dụ trên đây chỉ ra cách mà chế độ truy cập địa chỉ trực tiếp có thể được dùng để xác định nguồn của trường toán hạng. Truy cập địa chỉ trực tiếp cũng được dùng để xác định toán hạng đích trong bộ nhớ. Ví dụ lệnh MOV [437Ah],BX sẽ copy nội dung của thanh ghi BX vào hai ô nhớ trong đoạn dữ liệu. Nội dung của BL sẽ copy vào ô nhớ có độ lệch là 437Ah, nội dung thanh ghi BH sẽ copy vào ô nhớ có độ lệch 437Bh.

Hãng Intel có lí do khi thiết kế vi xử lí 8086 truy cập bộ nhớ sử dụng các địa chỉ lôgic *sement: offset* chứ không phải truy cập bộ nhớ trực tiếp với địa chỉ 20-bit. Điều này cho phép 8086 chỉ phải thao tác và lưu trữ giá trị 16 bit thay cho giá trị 20 bit, khiến cho việc ghép nối dễ dàng hơn với bản mạch nhớ 16 bit hay 8 bit và với thanh ghi 16 bit trong 8086. Lí do nữa là kiểu làm việc với các đoạn nhớ như vậy sẽ rất hữu ích trong một hệ máy vi tính có hoạt động

kiểu chia xẻ thời gian (time sharing). Xét một hệ thống đa nhiệm, được điều khiển bởi một vi xử lí. Thí dụ, nếu mỗi nhiệm vụ cần tới 20 mili giây thì sau khi làm việc 20 mili giây cho mỗi nhiệm vụ, CPU phải quay trở lại chương trình của nhiệm vụ tiếp theo. Mỗi lần CPU nối thông cho chương trình một nhiệm vụ, nó phải truy cập vào vùng mã lệnh mới và vùng dữ liệu mới. Đoạn làm cho sự chuyển mạch này trở nên dễ dàng và an toàn hơn. Mỗi chương trình người dùng có thể được phân cho một đoạn nhớ riêng biệt dành cho mã lệnh và dữ liệu. Chương trình người dùng chứa offset hay độ lệch so với những địa chỉ cơ sở các đoạn này. Để chuyển chương trình nhiệm vụ này sang chương trình nhiệm vụ khác, CPU chỉ phải tải lại bốn thanh ghi đoạn với địa chỉ cơ sở đoạn được phân cho nhiệm vụ thứ hai. Nói cách khác, phân đoạn giúp dễ dàng giữ chương trình và dữ liệu nhiệm vụ này tách biệt khỏi nhiệm vụ khác và phân đoạn làm dễ dàng chuyển từ chương trình này sang chương trình khác.

2.1.4. Tập lệnh máy của họ vi xử lí 80×86 và mã hoá các lệnh

Họ 80×86 dùng 6 loại định dạng số liệu với độ dài từ 8, 16 hoặc 32 bit tuỳ loại vi xử lí. Dữ liệu kí tự như chữ, các con số và các kí hiệu khác được lưu trữ dưới dạng mã chuẩn ASCII. Một từ 16 hoặc 32 bit được lưu trữ tại các ô nhớ liên tiếp nhau theo cách: byte (hoặc từ) ít ý nghĩa nhất LSB được lưu ở vùng ô nhớ có địa chỉ thấp hơn và byte (hoặc từ) nhiều ý nghĩa nhất ở ô nhớ có địa chỉ cao hơn. Thí dụ như sau:



Lệnh của bộ vi xử lí được mã hóa dưới dạng các con số 0 và 1 gọi là mã máy. Một lệnh dài 1 đến một vài byte, gồm 2 thành phần chính là mã lệnh (mã toán) và các toán hạng. Đôi khi còn có cả các tiền tố nữa. Việc mã hoá này là phức tạp và phụ thuộc vào nhiều yếu tố. Trước khi liệt kê tóm tắt tập lệnh mã máy của vi xử lí 80×86 với số nhịp đồng hồ cần cho mỗi lệnh ta hãy xét các thí dụ:

- Lệnh *truyền số liệu tức thời tới thanh ghi* (immediate to register). Lệnh này chạy trong 4 nhịp đồng hồ. Lệnh gồm một byte mã lệnh và 1 byte số liệu (khi truyền 1 byte) hoặc 2 byte số liệu (khi truyền một từ). Định dạng của lệnh là:

1	0	1	1	w	Reg	byte số liệu thấp	byte số liệu cao

- Lệnh *truyền dữ liệu giữa thanh ghi và ô nhớ*:

1	0	0	0	1	0	d	w	mod	Reg	R/M	Độ dịch (disp.) thấp	Độ dịch (disp.) cao
										hoặc	Địa chỉ trực tiếp thấp	Địa chỉ trực tiếp cao

Các chữ trong định dạng lệnh đại diện cho các giá trị số trong từng trường hợp cụ thể phụ thuộc vào các toán hạng và chế độ định địa chỉ:

- Các bit **w** và **Reg** quy định cho các thanh ghi như sau:

w = 1	w = 0	Reg
AX	AL	000
BX	BL	011

CX	CL	001
DX	DL	010
SP	AH	100
DI	BH	111
BP	DH	101
SI	CH	110
CS		01
DS		11
ES		00
SS		10

- Bit **d** chỉ hướng đi của dữ liệu, d = 1 dữ liệu đi đến thanh ghi cho bởi 3 bit Reg; d = 0 đến từ thanh ghi đó.
- 2 bit **mod** cùng với 3 bit **R/M** quy định chế độ định địa chỉ cho các toán hạng của lệnh theo như bảng sau:

mod / R/M	00	01	10	11	
	Chế độ địa chỉ bộ nhớ			Chế độ thanh ghi	
				w = 0	w = 1
000	[BX]+[SI]	[BX]+[SI]+disp.8	[BX]+[SI]+disp.16	AL	AX
001	[BX]+[DI]	[BX]+[DI]+disp.8	[BX]+[DI]+disp.16	CL	CX
010	[BP]+[SI]	[BP]+[SI]+disp.8	[BP]+[SI]+disp.16	DL	DX
011	[BP]+[DI]	[BP]+[DI]+disp.8	[BP]+[DI]+disp.16	BL	BX
100	[SI]	[SI]+disp.8	[SI]+disp.16	AH	SP
101	[DI]	[DI]+disp.8	[DI]+disp.16	CH	BP
110	d16 (địa chỉ trực tiếp)	[BP]+disp.8	[BP]+disp.16	DH	SI
111	[BX]	[BX]+disp.8	[BX]+disp.16	BH	DI

- disp.8 và disp.16 là viết tắt của độ dịch (displacement) 8 và 16 bit.
- Các giá trị trong chế độ địa chỉ bộ nhớ là các địa chỉ hiệu dụng EA sẽ được cộng với DS để tạo ra địa chỉ vật lí (riêng BP phải được cộng với SP).

- Số chu kì nhịp đồng hồ dùng cho các kiểu địa chỉ hiệu dụng EA như sau với lưu ý rằng cần phải cộng thêm 4 nhịp đồng hồ khi truy xuất từ dữ liệu ở các địa chỉ lẻ.

Địa chỉ hiệu dụng EA	Số nhịp đồng hồ (nhịp EA)
offset tức thời	6
BASE (BX,BP, SI,DI)	5
BASE + Độ dịch	9
BASE + chỉ số (BP+DI, BX+SI)	7
BASE + chỉ số (BP+DI, BX+SI)	8
BASE + chỉ số (BP+DI, BX+SI) +Độ dịch	11
BASE + chỉ số (BP+SI, BX+DI) +Độ dịch	12

Do đó, mã máy cho lệnh MOV AL,89 sẽ là:

1011 0000 1000 1001 = B089 h

mã máy cho lệnh MOV CL,[BX] sẽ là :

1000 1010 0000 1111 = 8A0F h

Trong cách định địa chỉ kể trên có trường hợp các thanh ghi địa chỉ đoạn và thanh ghi offset được ngầm định đi kèm với nhau từng cặp để địa chỉ hoá các toán hạng trong các vùng khác nhau của bộ nhớ.

Thí dụ, lệnh MOV CX, [BX] +30 ; là chuyển nội dung của 2 ô nhớ liên tiếp có địa chỉ lôgic là DS : (BX+30) và DS : (BX+31) vào CX.

Ở đây thanh ghi đoạn mã DS là thanh ghi đoạn ngầm định, không được viết ra trong mã lệnh. Đôi khi không muốn dùng các thanh ghi đoạn ngầm định đó thì có thể viết rõ thanh ghi đoạn cần thiết và thêm dấu hai chấm (:) trước thanh ghi offset.

Thí dụ, MOV AL, ES:[BX] ; cụm (ES:) chỉ rõ loại bỏ thanh ghi đoạn ngầm định DS và thay bằng thanh ghi đoạn ES.

Có thể tóm tắt các chế độ định địa chỉ như bảng sau.

Chế độ định vị	Toán hạng	Thanh ghi đoạn ngầm định
Thanh ghi	Reg	
Tức thời	Data	
Trực tiếp	[offset]	DS
Gián tiếp thanh ghi	[BX]	DS
	[SI]	DS
	[DI]	DS
Cơ sở	[BX] + Độ dịch	DS
	[BP] + Độ dịch	SS
Chỉ số	[DI] + Độ dịch	DS
	[SI] + Độ dịch	DS
Chỉ số cơ sở	[BX] + [DI] + Độ dịch	DS
	[BX] + [SI] + Độ dịch	DS
	[BP] + [DI] + Độ dịch	SS
	[BP] + [SI] + Độ dịch	SS

Các thanh ghi đoạn và thanh ghi offset ngầm định đi kèm với nhau như sau:

Thanh ghi đoạn	CS	DS	ES	SS
Thanh ghi offset	IP	SI, DI, BX	DI	SP, BP

Phần phụ lục 1 có liệt kê tỉ mỉ mã máy cùng số xung nhịp đồng hồ của các lệnh máy 8086. Các số liệu này cần thiết khi muốn tính toán các khoảng thời gian thật chính xác trong PC.

Các ngôn ngữ bậc cao đều cung cấp các phương tiện cho phép thao tác trực tiếp với bộ nhớ hoặc với các chip bổ trợ. Nhưng để tăng tốc độ giao tiếp cũng như tăng độ linh động thì nhiều người vẫn thích dùng ngôn ngữ máy cho các mục đích này. Dưới đây là tập các lệnh máy được viết dưới dạng các từ gợi nhớ (mnemonic) cho các vi xử lí từ 8086 đến 80486.

Từ gợi nhớ	Chức năng	Từ gợi nhớ	Chức năng
Các lệnh của họ 80x86			
AAA	Chỉnh sau phép cộng 2 số dạng ASCII	CMP	So sánh toán hạng đích và gốc
AAD	Chỉnh 2 số mã ASCII trước phép chia	CMPS	So sánh 2 chuỗi byte hay từ

AAM	Chỉnh sau phép nhân 2 số mã ASCII	CMPSB	So sánh xâu (byte)
AAS	Chỉnh sau phép trừ 2 số mã ASCII	CMPSW	So sánh xâu (từ)
ADC	Cộng có cờ nhớ	CWD	Biến đổi từ thành từ kép
ADD	Cộng 2 toán hạng	DAA	Hiệu chỉnh thập phân sau phép cộng
AND	Và từng bit tương ứng của 2 toán hạng	DAS	Hiệu chỉnh thập phân sau phép trừ
CALL	Gọi chương trình con	DEC	Giảm toán hạng đích đi 1
CBW	Chuyển byte thành từ	DIV	Chia không dấu
CLC	Xoá cờ nhớ	ESC	Thoát
CLD	Xoá cờ hướng	HLT	Treo
CLI	Xoá cờ ngắt	IDIV	Chia số nguyên
CMC	Lấy bù cờ nhớ	IMUL	Nhân số nguyên
IN	Đọc cổng vào/ra	JS	Nhảy nếu có cờ dấu
INC	Tăng toán hạng đích lên 1	JZ	Nhảy nếu bằng 0
INT	Gọi ngắt	LAHF	Nạp 8 bit thấp của cờ vào AH
INTO	Ngắt nếu bị tràn	LDS	Nạp ô nhớ từ kép vào thanh ghi đoạn dữ liệu
IRET	Trở về chỗ bị ngắt	LEA	Nạp địa chỉ hiệu dụng
JA	Nhảy nếu ở trên	LES	Nạp con trỏ khi dùng ES
JAE	Nhảy nếu ở trên hoặc bằng	LOCK	Khoá bus
JB	Nhảy nếu thấp hơn	LODS	Nạp xâu
JBE	Nhảy nếu thấp hơn hoặc bằng	LODSB	nạp xâu (byte)
JC	Nhảy nếu có cờ nhớ	LODSW	Nạp xâu (từ)
JCXZ	Nhảy nếu CX=0	LOOP	Vòng lặp
JE	Nhảy nếu bằng	LOOPE	Lặp lại trong khi bằng
JG	Nhảy nếu lớn hơn	LOOPNE	Lặp lại khi không bằng
JGE	Nhảy nếu lớn hơn hoặc bằng	LOOPNZ	Lặp lại khi không bằng 0
JL	Nhảy nếu nhỏ hơn	LOOPZ	Lặp lại khi bằng 0
JLE	Nhảy nếu nhỏ hơn hoặc bằng	MOV	Chuyển nguồn tới đích
JMP	Nhảy không điều kiện	MOVS	Chuyển xâu
JNA	Nhảy nếu không ở trên	MOVSB	Chuyển xâu (byte)
JNAE	Nhảy nếu không ở trên hoặc bằng	MOVSW	Chuyển xâu (từ)
JNB	Nhảy nếu không ở dưới	MUL	Phép nhân
JNBE	Nhảy nếu không ở dưới hoặc bằng	NEG	Đảo dấu hay lấy bù 2
JNC	Nhảy nếu không có cờ nhớ	NOP	Không hành động
JNE	Nhảy nếu không bằng	NOT	Đảo dấu (lấy bù 1)
JNG	Nhảy nếu không lớn hơn	OR	Hoặc các bit tương ứng của 2 toán hạng
JNGE	Nhảy nếu không lớn hơn hoặc bằng	OUT	Viết cổng vào/ra
JNL	Nhảy nếu không nhỏ hơn	POP	Hồi phục nội dung (các thanh ghi, ...)
JNLE	Nhảy nếu không nhỏ hơn hoặc bằng	POPF	Hồi phục nội dung các cờ
JNO	Nhảy nếu không tràn	PUSH	Đẩy nội dung (thanh ghi,...) vào ngăn xếp
JNP	Nhảy nếu không có cờ chẵn lẻ	PUSHF	Đẩy nội dung cờ vào ngăn xếp
JNS	Nhảy nếu không có cờ dấu	RCL	Quay trái qua cờ nhớ
JNZ	Nhảy nếu không bằng 0	RCR	Quay phải qua cờ nhớ
JO	Nhảy nếu có cờ tràn	REP	Lặp lại
JP	Nhảy nếu có cờ chẵn lẻ	REPE	Lặp lại khi bằng
JPE	Nhảy nếu chẵn lẻ chẵn	REPNE	Lặp lại khi không bằng

JPO	Nhảy nếu chẵn lẻ lẻ	REPNZ	Lặp lại khi không bằng 0
REPZ	Lặp lại trong khi bằng 0	STC	Đặt cờ nhớ
RET	Trở về	STD	Đặt cờ hướng
ROL	Quay trái	STI	Đặt cờ ngắt
ROR	Quay phải	STOS	Lưu trữ xâu
SAHF	Lưu trữ AH vào byte thấp của cờ	STOSB	Lưu trữ xâu (byte)
SAL	Dịch trái số học	STOSW	Lưu trữ xâu (từ)
SAR	Dịch phải số học	SUB	Phép Trừ
SBB	Trừ có mượn	TEST	Kiểm tra (nhân lôgic đích với gốc)
SCAS	Quét xâu	WAIT	Đợi
SCASB	Quét xâu (byte)	XCHG	Trao đổi
SCASW	Quét xâu (từ)	XLAT	Chuyển đổi bảng
SHL	Dịch trái	XOR	Hoặc tuyệt đối từng bit tương ứng 2 số
SHR	Dịch phải		
Các lệnh chỉ có trong 80286, 386 và 486			
ARPL	Chỉnh trường RPL của bộ chọn	LSL	Nạp độ dài đoạn nhớ
BOUND	Kiểm tra biên của trường	LTR	Nạp thanh ghi nhiệm vụ
CLTS	Xoá cờ chuyển nhiệm vụ	OUTS	Xuất xâu ra cổng vào/ra
ENTER	Tạo khối các thông số để vào CTC	POPA	Hồi phục tất cả các thanh ghi đa năng
INS	Nhập xâu từ cổng vào/ra	PUSHA	Đẩy vào ngăn xếp các thanh ghi đa năng
LAR	Nạp quyền thâm nhập	SGDT	Lưu trữ thanh ghi bảng bộ mô tả toàn cục
LEAVE	Ra khỏi CTC (chương trình con)	SIDT	Lưu trữ thanh ghi bảng bộ mô tả ngắt
LGDT	Nạp thanh ghi bảng bộ mô tả toàn cục	SLDT	Lưu trữ thanh ghi bảng bộ mô tả cục bộ
LIDT	Nạp thanh ghi bảng bộ mô tả ngắt	SMSW	Lưu trữ từ trạng thái máy
LLDT	Nạp thanh ghi bảng bộ mô tả cục bộ	STR	Lưu trữ thanh ghi nhiệm vụ
LMSW	Nạp từ trạng thái máy	VERR	Kiểm tra một bộ chọn đoạn để đọc
		VERW	Kiểm tra một bộ chọn đoạn để viết
Các lệnh chỉ có trong 80386 và 486			
BSF	Quét bit về phía trước	SETL	Đặt byte nếu nhỏ hơn
BSR	Quét bit về phía sau	SETLE	Đặt byte nếu nhỏ hơn hoặc bằng
BT	Kiểm tra bit	SETNA	Đặt byte nếu không ở trên
BTC	Kiểm tra và đảo bit	SETNAE	Đặt byte nếu không ở trên hoặc bằng
BTR	Kiểm tra và xoá bit	SETNB	Đặt byte nếu không ở dưới
BTS	Kiểm tra và đặt bit	SETNBE	Đặt byte nếu không ở dưới hoặc bằng
CDQ	Biến đổi từ kép thành từ kép bốn	SETNC	Đặt byte nếu không nhớ
CMPSD	So sánh xâu (từ kép)	SETNE	Đặt byte nếu không bằng
CWDE	Biến đổi từ thành từ kép trong EAX	SETNG	Đặt byte nếu không lớn hơn
JECXZ	Nhảy nếu ECX bằng 0	SETNGE	Đặt byte nếu không lớn hơn hoặc bằng
LFS	Nạp con trỏ khi dùng FS	SETNL	Đặt byte nếu không nhỏ hơn
LGS	Nạp con trỏ khi dùng GS	SETNLE	Đặt byte nếu không nhỏ hơn hoặc bằng
LSS	Nạp con trỏ khi dùng SS	SETNO	Đặt byte nếu không tràn
LODSD	Nạp xâu (từ kép)	SETNP	Đặt byte nếu không có chẵn lẻ
MOVSD	Chuyển xâu (từ kép)	SETNS	Đặt byte nếu không dấu
MOVSX	Chuyển với Sign-eXtend	SETNZ	Đặt byte nếu không bằng 0
MOVZX	Chuyển với Zero-eXtend	SETO	Đặt byte nếu tràn

SCASD	Quét xâu (từ kép)	SETP	Đặt byte nếu có chẵn lẻ
SETA	Đặt byte nếu ở trên	SETPE	Đặt byte nếu chẵn lẻ chẵn
SETAE	Đặt byte nếu ở trên hoặc bằng	SETPO	Đặt byte nếu có chẵn lẻ lẻ
SETB	Đặt byte nếu ở dưới	SETS	Đặt byte nếu có dấu
SETBE	Đặt byte nếu ở dưới hoặc bằng	SETZ	Đặt byte nếu bằng 0
SETC	Đặt byte nếu có cờ nhớ	SHLD	Dịch trái (từ kép)
SETE	Đặt byte nếu bằng	SHRD	Dịch phải (từ kép)
SETG	Đặt byte nếu lớn hơn	STOSD	Lưu trữ xâu (từ kép)
SETGE	Đặt byte nếu lớn hơn hoặc bằng		
Các lệnh chỉ có trong 80486			
BSWAP	Hoán chuyển byte	INVLPG	Vo hiệu hoá TLB (cho chế độ trang)
CMPXCHG	So sánh và trao đổi	WBINVD	Ghi trở lại bộ nhớ chính vào nhớ ngầm
INVD	Vô hiệu hoá bộ nhớ ngầm	XADD	Hoán chuyển và cộng

2.2. THỰC HIỆN LỆNH MÁY BÊN TRONG VI XỬ LÍ

2.2.1. Các quá trình thực hiện lệnh

Ta sẽ khảo sát chi tiết hơn cách thức làm việc của các đơn vị bên trong vi xử lí. Nhắc lại rằng CPU thực hiện một lệnh theo một chuỗi các bước như sau:

Tìm và nạp lệnh từ bộ nhớ vào thanh ghi lệnh IR.

Chuyển thanh ghi con trỏ lệnh IP trỏ tới lệnh tiếp theo.

Xác định loại lệnh vừa nạp.

Nếu lệnh sử dụng dữ liệu trong bộ nhớ thì xác định nơi chứa dữ liệu.

Tìm và nạp dữ liệu vào các thanh ghi bên trong CPU (nếu có).

Thực thi lệnh.

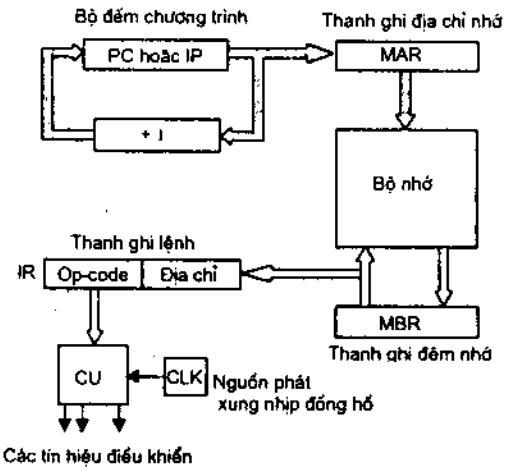
Lưu các kết quả vào các vùng thích hợp.

Trở lại bước 1 để bắt đầu thực hiện lệnh kế tiếp.

Xét một mô hình đơn giản của vi xử lí như hình 2.5. Đó là một CPU thực hiện chương trình theo từng bước lệnh một như sơ đồ khối đọc lệnh từ bộ nhớ (bỏ qua các đường số liệu). Trong sơ đồ này chỉ có các đường địa chỉ và các đường điều khiển. Ta biết có 2 loại thông tin được "chảy" trong hệ máy tính: thông tin về địa chỉ và thông tin về số liệu trong đó thông tin về số liệu bao gồm các lệnh, các hằng số và biến số.

Có thể quy ước một cách viết để biểu diễn quá trình truyền thông tin đó; gọi là *ngôn ngữ truyền thanh ghi (Register Transmision Langue).*

Thí dụ: [MAR] ← [PC] có nghĩa là truyền (sao chép, copy) nội dung của bộ đếm chương trình vào thanh ghi địa chỉ nhớ MAR, lưu ý rằng trong quá trình này



Hình 2.5 Các đường địa chỉ của một CPU đơn giản.

nội dung của PC không thay đổi.

[PC] ← [PC] + 1 có nghĩa là : tăng nội dung của PC lên 1

[MS(20)] ← [PC] có nghĩa là: sao chép nội dung của PC vào vùng nhớ 20 trong bộ nhớ chính.

a, Quá trình đọc lệnh

Thanh ghi IP trong 8086 thực chất là thanh ghi đếm chương trình PC như cách mà các vi xử lý đời trước nó thường được gọi. Ta biết nội dung thanh ghi đếm chương trình này chính là địa chỉ của lệnh kế tiếp. Để chạy một lệnh, đầu tiên nội dung của PC được chuyển vào thanh ghi địa chỉ nhớ MAR (tức là [MAR] ←[PC]). Khi truyền xong, nội dung của PC tự động tăng lên 1 ([PC] ← [PC]+1). Thanh ghi địa chỉ nhớ MAR chứa địa chỉ của một vùng trong bộ nhớ chính MS, ở đó số liệu được đọc/viết. Tiếp đó khi thực hiện chu trình đọc, lệnh cần thực hiện sẽ được đọc từ bộ nhớ và truyền tới thanh ghi đệm nhớ MBR ([MBR]←[MS[MAR]]). Thanh ghi MBR là nơi lưu giữ tạm thời các lệnh dữ liệu nhận được từ bộ nhớ trong chu kì đọc hoặc số liệu được truyền tới CPU trong chu kì viết.

Tiếp theo, lệnh được chuyển từ MBR tới thanh ghi lệnh IR. ở đây nó được chia làm 2 trường. Mỗi trường là một từ mà ở đó các bit được nhóm với nhau thành một thực thể lôgic. Một trường trong IR chứa *mã toán* (op-code) để báo cho CPU biết mã toán nào đang được thực hiện. Trường khác được gọi là *trường toán hạng*, chứa địa chỉ của số liệu được dùng trong khi thực hiện lệnh. Đôi khi trường toán hạng được bỏ đi vì không phải tất cả các mã toán cần một vùng nào đó trong bộ nhớ. Các lệnh đang được mô tả này thuộc lớp các lệnh một địa chỉ.

Đơn vị điều khiển CU lấy mã toán từ IR cùng với một dòng các xung nhịp đồng hồ và phát ra các tín hiệu điều khiển tất cả các phần của CPU. Đơn vị điều khiển có trách nhiệm chuyển các nội dung của bộ đếm chương trình vào MAR, thực hiện một chu kì đọc và chuyển các nội dung của MBR tới IR.

Chuỗi các hoạt động kể trên, trong đó lệnh kế tiếp được chuyển từ bộ nhớ tới thanh ghi lệnh được gọi là *giai đoạn tìm-nạp lệnh* (fetch phase). Trong giai đoạn này, lệnh được đọc từ bộ nhớ và được giải mã bởi đơn vị điều khiển. Giai đoạn tìm-nạp lệnh được tiếp nối bởi giai đoạn thực thi lệnh trong đó đơn vị điều khiển CU phát ra tất cả các tín hiệu cần thiết để chạy lệnh. Đơn vị điều khiển CU là phần phức tạp nhất của CPU. Dưới đây liệt kê chuỗi các lệnh xảy ra trong giai đoạn tìm-nạp lệnh, trong đó FETCH là nhãn, IR(op-code) có nghĩa là trường mã toán của thanh ghi lệnh.

Fetch	[MAR] ← [PC]	copy PC tới MAR
	[PC] ← [PC] +1	tăng PC lên 1
	[MBR] ← [MS([MAR])]	đọc lệnh
	[IR] ← [MBR]	chuyển nó tới IR
	CU ← [IR(op-code)]	

b, Đường số liệu trong CPU

Hình 2.6 là sơ đồ CPU kể trên được thêm vào các đường số liệu để xét trong giai đoạn thực thi lệnh. Trong đó ta thêm vào các thanh ghi số liệu D0 (Data Reg.) và đơn vị tính số học lôgic ALU. Thanh ghi số liệu giữ các kết quả trung gian và tạm thời trong quá trình tính toán. Thanh ghi số liệu cần thiết trong các máy tính loại 1 địa chỉ vì các thuật toán (thí dụ, +, -, *, /) xảy ra trên một toán hạng được phân biệt bởi lệnh được đặt trước trong thanh ghi số liệu hay một trong các toán hạng ban đầu. Thanh ghi số liệu này gọi là thanh ghi tích luỹ.

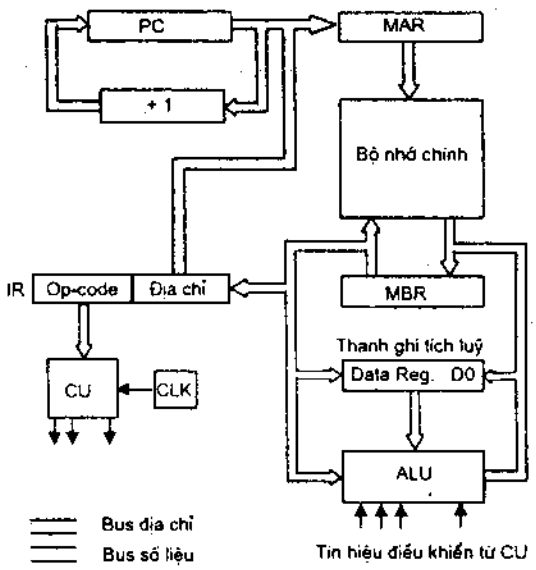
Thí dụ, nếu ta cộng các nội dung của vùng nhớ 1234 tới thanh ghi số liệu, số liệu trong thanh ghi sẽ được thay thế bởi các nội dung trước của thanh ghi số liệu cộng với nội dung của vùng 1234 được cho bởi lệnh.

ALU thực hiện tất cả các phép tính lôgic và số học. Các toán tử lôgic và số học được áp lên nội dung của thanh ghi số liệu hoặc của MBR. Lối ra của ALU được quay ngược lại tới các thanh ghi này.

Sự khác nhau cơ bản giữa các toán tử lôgic và số học ở chỗ, trong các toán tử lôgic cờ sẽ không được phát ra khi bit A và B của các từ A và B được hoạt động. Bảng sau là các thí dụ về các toán tử lôgic và số học điển hình trong 8086.

Giả sử ta cần chạy một chương trình đơn giản: P = Q + R. Dấu "+" có nghĩa là phép toán cộng số học. Nhớ rằng P, Q và R là kí hiệu các tên được gán cho các vùng chứa các biến trong bộ nhớ.

```
MOV D0, Q   ;  nạp thanh ghi D0 nội dung
               của vùng nhớ Q
ADD  D0,R   ;  cộng nội dung của vùng nhớ
               R với thanh ghi D0
MOV P,D0    ;  cất nội dung thanh ghi D0
               vào vùng P
```



Hình 2.6 Các đường địa chỉ và số liệu trong CPU.

Có một chuỗi các hoạt động được thực hiện trong các giai đoạn tìm-nạp và thực hiện lệnh của lệnh ADD D0,R. Các hoạt động này nói cho ta biết những cái gì đang thực sự xảy ra trong CPU. Hai toán tử cùng trong 1 dòng được chạy đồng thời. Các lệnh trong dạng như [PC] ←[MAR] hoặc [D0] ←[D1]+[MAR] thường được gán cho tên là *các vi lệnh* (microinstructions). Một lệnh ở *mức hợp ngữ* (thí dụ, MOV, ADD...) được thực hiện bằng một chuỗi vi lệnh. Nói chung, *vi lệnh* và *vi chương trình* là thuộc lĩnh vực quan tâm của các nhà thiết kế máy tính, chúng là những hợp phần bên trong tạo nên CPU. Sau đây là hoạt động của đơn vị điều khiển nhìn theo các vi lệnh.

Fetch	[MAR] ← [PC]	chuyển nội dung PC tới MAR
	[PC] ← [PC] +1	tăng nội dung PC lên 1
	[MBR] ← [MS([MAR])]	đọc từ bộ nhớ chính
	[IR] ← [MBR]	chuyển nội dung MBR tới IR
	CU ← [IR(op-code)]	chuyển op-code từ IR tới CU
ADD	[MAR] ← [IR(adrress)]	chuyển địa chỉ toán hạng tới MAR
	[MBR] ← [MS([MAR])]	đọc số liệu
	ALU ← [MBR], ALU ← [D0]	thực hiện phép cộng
	[D0] ←ALU	chuyển lối ra ALU tới thanh ghi số liệu.

c, *Các lệnh nhảy có điều kiện*

Để có cơ chế chọn lựa hoặc lặp lại một nhóm lệnh, CPU phải có khả năng chạy chương trình có lệnh nhảy (rẽ nhánh) tuyệt đối hoặc lệnh nhảy có điều kiện. Lệnh nhảy là một lệnh

buộc CPU chạy một lệnh ngoài chuỗi lệnh bình thường. Nhảy có điều kiện cho phép tạo các cấu trúc IF THEN ELSE. Sơ đồ khối hình 2.7 chỉ ra các đường địa chỉ và số liệu mới được thêm vào CPU để thực hiện các lệnh nhảy. Ở đây có thêm 3 phần tử: mạch lật nhớ với bit nhớ C (carry Flip Flop), thanh ghi trạng thái bộ xử lí PSR (processor status register) và một đường giữa trường địa chỉ của thanh ghi lệnh và bộ đếm chương trình.

Hoạt động của mạch lật nhớ như sau: bất cứ khi nào, thí dụ phép cộng hoặc trừ được thực hiện, kết quả sẽ đi tới thanh ghi số liệu và bit nhớ (hoặc mượn) được lưu trữ trong mạch lật nhớ.

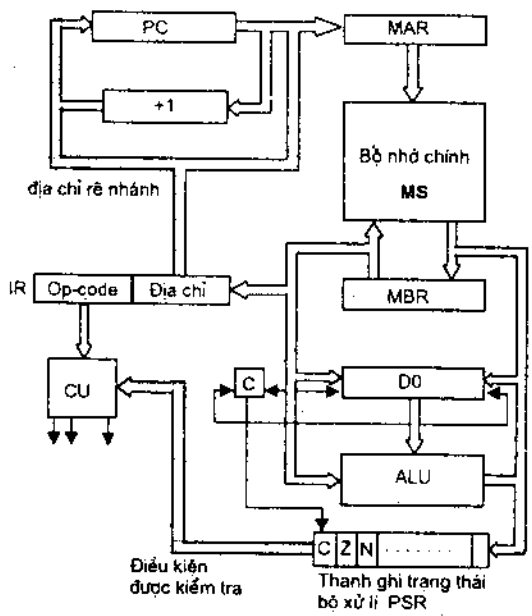
Thí dụ, cộng 00000100 + 00000001 cho tổng 00000101 và bit nhớ C = 0 trong khi tổng 1000110 + 100001110 cho kết quả 00010100 và bit nhớ C = 1. Tương tự, nếu nội dung của thanh ghi số liệu được dịch trái (hoặc phải) một bit, bit đó sẽ "tràn" ra ngoài và được truyền tới mạch lật nhớ. Vì lí do như vậy, bit nhớ có thể được coi như là một bit mở rộng của thanh ghi số liệu. Sau khi CPU thực hiện một thuật toán lôgic và số học, nó cập nhật các bit của thanh ghi trạng thái PSR để phản ánh bản chất của kết quả. Các bit của PSR là:

C: bit nhớ, được đặt nếu nhớ được sinh ra trong phép toán cuối cùng. Dĩ nhiên bit C cũng giống như bit nhớ trong mạch lật nhớ.

Z: bit zê-rô, được đặt nếu phép toán cuối cùng có kết quả bằng 0.

N: bit dấu âm, được đặt nếu kết quả của phép toán cuối cùng mang dấu âm trong mã bù hai (tức là MSB = 1).

V: bit tràn, được đặt nếu phép toán cuối cùng tràn số. Đó là khi phép toán trên một hoặc hai số bù hai cho kết quả ngoài dải cho phép.



Hình 2.7 Các đường thông tin trong CPU và các lệnh có điều kiện. Lưu ý rằng bit C là một phần trong thanh ghi PSR và một phần như bit mở rộng của thanh ghi số liệu.

Bảng sau cho ví dụ về tác dụng của phép cộng hai số tới nội dung của PSR.

Toán hạng 1 + toán hạng 2 = kết quả	Các bit trạng thái của PSR
00000011 + 00000110 = 00000111	C=0, Z=0, N=0, V=0
11111111 + 00000001 = 00000000	C=1, Z=1, N=0, V=0
01100110 + 00110010 = 10011000	C=0, Z=0, N=1, V=1
11001001 + 10100000 = 01101001	C=1, Z=0, N=0, V=1

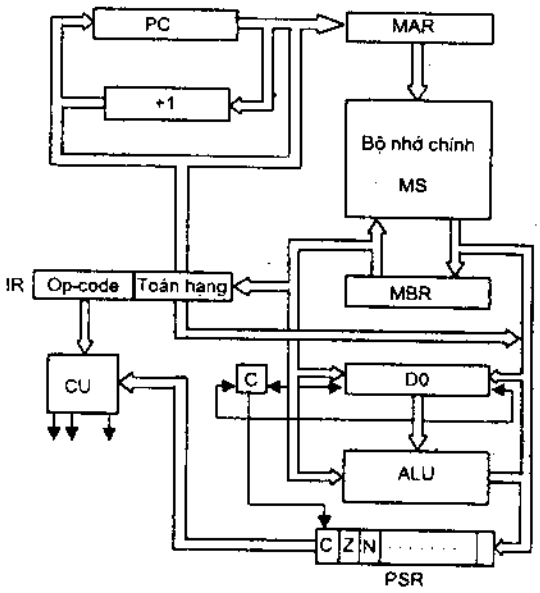
PSR được nối tới đơn vị điều khiển CU, cho phép một số loại lệnh nhất định thăm dò thanh ghi này. Thí dụ các lệnh kiểm tra liệu phép toán cuối cùng được thực hiện bởi bộ xử lí sinh ra kết quả dương hay không, hoặc có xảy ra tràn số hay bit nhớ có được đặt hay không. Ở đây cần

một cơ cấu cho phép làm một việc gì đó nếu kết quả của phép thử là đúng (true) và làm việc khác nếu kết quả là sai (false). Sự cải tiến cuối cùng trong hình 2.7 so với hình 2.6 là việc thêm vào các đường giữa trường địa chỉ của thanh ghi lệnh và bộ đếm chương trình. Điều này cho phép máy tính đáp ứng lại được với kết quả thăm dò của PSR.

Một lệnh có thể kiểm tra một hoặc nhiều bit của PSR và nếu bit được xoá thì lệnh kế tiếp sẽ được nhận từ bộ nhớ theo cách thông thường. Nếu bit được đặt, lệnh kế tiếp sẽ nhận được từ vùng mà địa chỉ của nó chứa trong thanh ghi lệnh. Loại lệnh này gọi là lệnh nhảy có điều kiện.

d, Toán hạng tức thời

Đôi khi cần có các phép toán mà mà toán hạng là các giá trị số nào đó. Thí dụ lệnh cộng số 12h vào nội dung thanh ghi D0, thí dụ ADD D0,12h. Lúc này toán hạng là một con số thực sự được sử dụng bởi mã toán ADD. Hình 2.8 sẽ chỉ ra một đường số liệu bổ sung giữa trường toán hạng của thanh ghi IR và ALU dành cho mục đích này.



Hình 2.8 Đường số liệu của toán hạng tức thời.

e, Kiến trúc của một vi xử lí thực tế

Phải nói rằng những năm gần đây kiến trúc máy tính đã được cải tiến bởi các tiến bộ về mặt công nghệ nhiều hơn là về khoa học máy tính và lập trình. Sự phát triển nhanh chóng của công nghệ vật liệu bán dẫn đã tạo ra các CPU có độ rộng từ dữ liệu ngày càng lớn tới 32 và 64 bit và có thêm nhiều thanh ghi trong đó. Các CPU điển hình với nhiều bus và một dãy các thanh ghi được xuất hiện. Nó cũng có các thanh ghi điều khiển giống như CPU giả định kể trên (PC, MAR, MBR, IR) nhưng sự thay đổi chính là sự hiện diện của nhiều thanh ghi số liệu và thanh ghi địa chỉ. Với các thanh ghi có độ rộng từ dữ liệu 16 bit sẽ cho phép một định dạng lệnh khác với định dạng một địa chỉ được mô tả trước đây. Để thâm nhập các thanh ghi này ta cần có định dạng lệnh hai địa chỉ. Do đó một lệnh lúc này gồm có 3 trường: mã toán, địa chỉ toán hạng và địa chỉ thanh ghi.

Thí dụ:

Lệnh MOV AX, [1234]

Mã toán	Thanh ghi toán hạng đích	Địa chỉ toán hạng nguồn

Lệnh MOV [1234], DX

Mã toán	Địa chỉ toán hạng đích	Thanh ghi toán hạng nguồn

Thay vì chỉ có một thanh ghi tích luỹ như ở CPU giả định xét trên, việc xuất hiện nhiều thanh ghi số liệu cho phép CPU thực tế sẽ có hiệu suất tính toán mạnh hơn. Khi có nhiều thanh ghi số liệu, người lập trình có thể lưu trữ những giá trị số liệu thường dùng trong các thanh ghi này trên chip xử lí thì sẽ cho phép tính toán nhanh hơn khi lưu trữ chung trong bộ

nhớ chính. Thí dụ lệnh cộng nội dung 2 thanh ghi sẽ không cần đòi hỏi một chu kì quy chiếu bộ nhớ nào cả. Lúc này chỉ có khó với người lập trình là phải chọn thanh ghi nào giữ biến nào. May mắn là trong hầu hết các chương trình được viết trong ngôn ngữ bậc cao, điều này đã được các trình biên dịch sắp đặt tự động cả. Sự xuất hiện thêm nhiều thanh ghi địa chỉ cho phép lưu giữ địa chỉ của các toán hạng được thâm nhập trong bộ nhớ. Nhiều vi xử lí lưu trữ lệnh trong hơn một từ nhớ, mã toán chứa trong một từ và địa chỉ của các toán hạng trong các từ tiếp sau. Sự sắp xếp này cho phép sử dụng bộ nhớ một cách hiệu quả nhưng chương trình sẽ chạy hơi chậm vì có nhiều hơn một từ nhớ cần được thâm nhập trong một chu kì đọc lệnh. Hoạt động này sẽ có thể được tăng tốc bằng cách trước tiên nạp toán hạng vào thanh ghi địa chỉ rồi sử dụng các lệnh phân biệt vùng của toán hạng trong nội dung thanh ghi địa chỉ. Một khi đã nạp địa chỉ của toán hạng vào thanh ghi địa chỉ rồi thì không phải quan tâm đến nó nữa. Lúc đó có thể thâm nhập toán hạng qua thanh ghi địa chỉ (có nghĩa rằng ta có thể dùng lệnh một từ để thâm nhập toán hạng như với lệnh hai từ). Thí dụ, lệnh ADD D0, [A1] có nghĩa là cộng nội dung vùng nhớ được chỉ bởi thanh ghi địa chỉ A1với nội dung của thanh ghi số liệu D0. Trong ngôn ngữ thanh ghi RTL có thể viết:

$$[D0] \leftarrow [D0] + [MS ([A1])]$$

Lưu ý rằng ta viết [A1] chứ không phải là A1 để ngụ ý rằng toán hạng không phải là nội dung của A1 mà là nội dung của vùng nhớ có địa chỉ là nội dung của A1.

Một ứng dụng khác của thanh ghi địa chỉ là trong việc tính toán động các địa chỉ của máy tính. Vì có thể áp một phép toán số vào nội dung của thanh ghi địa chỉ nên có thể làm việc với các địa chỉ có thể biến đổi. Phép tính các địa chỉ trong thời gian chạy cho phép người lập trình có thể hoạt động trên các cấu trúc bảng và dữ liệu kiểu khác.

2.2.2. Đơn vị điều khiển CU

Đơn vị điều khiển CU (Control Unit) là phần phức tạp nhất trong CPU. Hãy khảo sát sơ lược hoạt động của đơn vị này và cách nó cho phép CPU thực hiện lệnh hiện tại trong các thanh ghi lệnh. Hình 2.9 là mô hình của một CPU đơn giản được dùng để minh hoạ cho hoạt động của đơn vị điều khiển CU. Có hai cách tiếp cận khác nhau để thiết kế một CU. Cách thứ nhất là tạo nên một CU tự nó là một máy tính trong máy tính và biến đổi mỗi lệnh máy thành một chuỗi các lệnh sơ khai gọi là *vi lệnh*. Cách tiếp cận khác là tìm ra chuỗi các phép toán lôgic và số học cần thiết cho việc thực hiện một lệnh rồi thiết kế mạch lôgic thích hợp để thực hiện các chuỗi này.

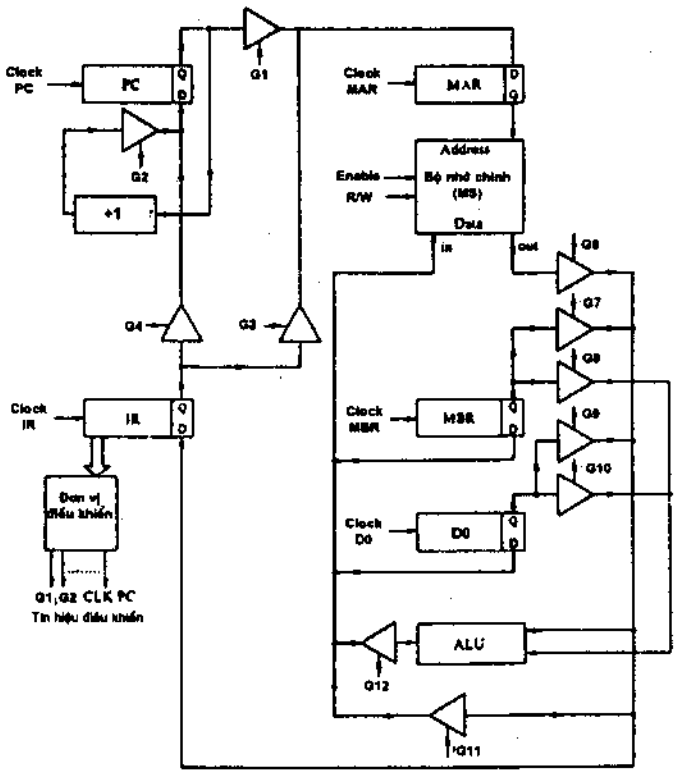
a, Đơn vị điều khiển được vi chương trình hoá

Ngôn ngữ tự nhiên của máy tính là các lệnh máy thường được biểu diễn bằng các từ gợi nhớ (mnemonic) cho ta các lệnh hợp ngữ. Các lệnh hợp ngữ này được gọi là các *lệnh vĩ mô* (macroinstruction). Mỗi lệnh vĩ mô được dịch bởi một số các hoạt động rất thô sơ (primitive) được gọi là *các lệnh vi mô* hay *vi lệnh* (microinstruction). Nói cách khác, có một ngôn ngữ thậm chí còn thô sơ hơn lệnh máy. Một vi lệnh là một sự kiện nhỏ nhất xảy ra bên trong máy tính và có thể bao gồm việc lật trạng thái một mạch lật hoặc chuyển số liệu từ một thanh ghi này tới thanh ghi khác.

Quá trình xử lí một lệnh vĩ mô bằng cách thực hiện một chuỗi các vi lệnh được gọi là *dịch lệnh*. Cấu trúc bên trong của CPU hình 2.9 kể trên có khác với các cấu trúc trên các hình trước khi mô tả chu trình lệnh. Cấu trúc lúc này bao gồm một cơ chế cho phép các luồng thông tin được chuyển dịch trong CPU. Mỗi một thanh ghi (bộ đếm chương trình, MAR, các thanh ghi số

liệu v.v...) được tạo bởi các flip-flop D. Sau mỗi nhịp đồng hồ, số liệu từ lối vào sẽ được chốt tới lối ra của các flip-flop này và giữ lại ở đó đến khi xuất hiện xung nhịp sau. Một bus dữ liệu cao tốc m-bit nối các thanh ghi với nhau. Giả sử ta muốn máy tính thực hiện một chu kì lệnh với mã toán cộng nội dung vùng nhớ có địa chỉ xác định với nội dung thanh ghi số liệu D0, rồi đặt kết quả vào D0: ADD D0, <địa chỉ>

Ta có thể viết một chuỗi các hoạt động - gọi là chuỗi các vi lệnh- xảy ra trong suốt quá trình thực hiện lệnh này không chỉ theo ngôn ngữ thanh ghi RTL mà theo các thuật ngữ cho phép (enable) các cửa (gates) và lật các flip-flop. Bảng 2.1 minh hoạ chuỗi các vi lệnh đó. Cần nhấn mạnh rằng pha tìm-nạp lệnh (fetch) của tất cả các lệnh thì như nhau và chỉ có pha thực thi lệnh (execute) biến đổi phù hợp với mã toán được đọc trong quá trình nhận lệnh.



Hình 2.9 Điều khiển luồng dữ liệu bên trong CPU.

Bảng 2.1 **Một chu trình lệnh được dịch ra thuật ngữ RTL**

Bước	Ngôn ngữ truyền thanh ghi RTL	Các hoạt động yêu cầu
1	[MAR] ← [PC]	cho phép G1, cấp nhịp cho MAR
1a	INC ← [PC]	
2	PC ← INC	cho phép G2, cấp nhịp cho PC
3	[MBR] ← [MS ([MAR])]	cho phép MS, R/$\overline{W}$ =1, cho phép G6 cho phép G11, cấp nhịp cho MBR
4	[IR] ← [MBR]	cho phép G7, cấp nhịp cho MBR
4a	CU ← [IR (op-code)]	
5	[MAR] ← [IR (địa chỉ)]	cho phép G3, cấp nhịp cho MAR
6	[MBR] ← [MS ([MAR])]	cho phép MS, R/$\overline{W}$ = 1, cho phép G6 cho phép G11, cấp nhịp cho MBR
7	ALU ← [MBR]	cho phép G7
7a	ALU ← [D0]	cho phép G10
8	[D0] ← ALU	cho phép G12, cấp nhịp cho thanh ghi số liệu

• **Vi chương trình**

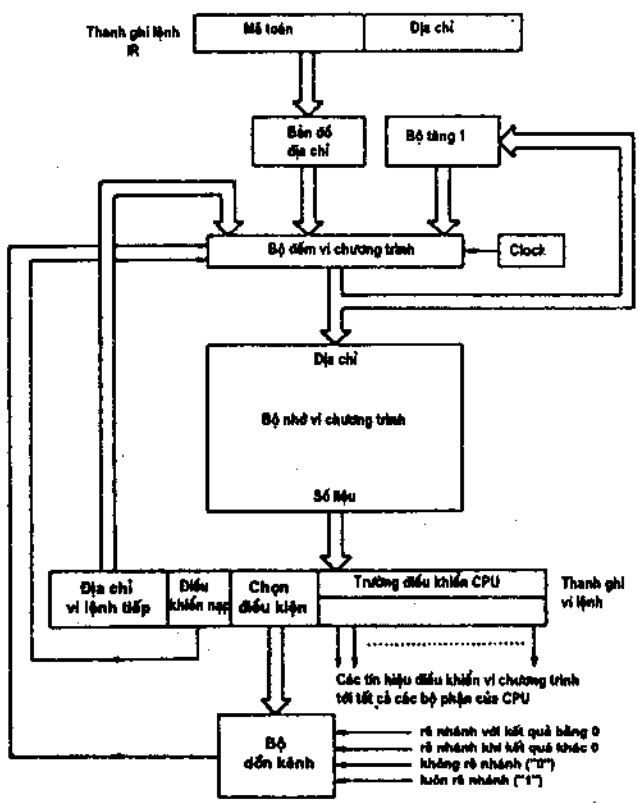
Tưởng tượng rằng lối ra của đơn vị điều khiển CU gồm 12 tín hiệu các cửa G_1 -G_{12}, 2 tín hiệu điều khiển bộ nhớ chính và 5 tín hiệu xung nhịp để kích hoạt các lối vào của các thanh ghi PC, MAR, MBR, IR và D0. Bảng 2.2 liệt kê 19 tín hiệu lối ra của CU theo một dãy các giá trị nhị phân được phát ra trong suốt quá trình nhận và chạy lệnh cộng ADD. Nếu cấp 19 tín hiệu trong bảng 2.2 tới các thành phần khác nhau của CPU trong hình 2.9 thì một chu trình lệnh sẽ được thực hiện. Trong máy tính thực thường có từ 64 đến 200 tín hiệu điều khiển như vậy.

Bảng 2.2 **Các tín hiệu điều khiển được phát trong các pha nhận và thực hiện lệnh cộng ADD**

Số bước	Các tín hiệu điều khiển cửa												Điều khiển bộ nhớ		Các xung nhịp kích hoạt thanh ghi				
	G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	Enable	R/$\overline{W}$	PC	MAR	MBR	D0	IR
1	1	0	0	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	x	1	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
4	0	0	0	0	0	0	1	0	0	0	0	0	0	x	0	0	0	0	1
5	0	0	1	0	0	0	0	0	0	0	0	0	0	x	0	1	0	0	0
6	0	0	0	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0
7	0	0	0	0	0	0	1	0	0	1	0	0	0	x	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	1	0	x	0	0	0	1	0

Một trong những sự khác nhau quan trọng giữa lệnh vi mô và lệnh vĩ mô là ở chỗ lệnh vi mô chứa nhiều trường hơn và có thể cung cấp một vài chứ không phải 1 hoặc 2 toán hạng như lệnh vĩ mô. Tám bước trong bảng 2.1 là một vi chương trình dịch một giai đoạn nhận lệnh (fetch) được theo sau bởi một lệnh cộng ADD.

Như vậy các lệnh vĩ mô được dịch ra thành một chuỗi các lệnh vi mô và có một vi chương trình cho mỗi mã toán (op-code) của CPU. Bây giờ ta xem xét chính vi chương trình và phần cứng đòi hỏi để chạy nó. Vi chương trình được chạy bởi một cơ chế tương tự như việc chạy lệnh vĩ mô. Thí dụ, thay vì cho bộ đếm chương trình thì ở đây, *đơn vị điều khiển được vi chương trình hoá* có một *bộ đếm vi chương trình*. Cấu trúc cơ bản của



Hình 2.10 Cấu trúc của một CPU được vi chương trình hoá.

một CU được vi chương trình hoá chỉ ra trên hình 2.10. Nó có một *bộ đếm vi chương trình*, một *bộ nhớ vi chương trình* và *thanh ghi vi lệnh*. Địa chỉ vi lệnh từ bộ đếm vi chương trình được đặt tới lối vào địa chỉ của bộ nhớ vi chương trình và số liệu ra của bộ nhớ được cấp tới thanh ghi vi lệnh.Thông tin trong thanh ghi vi lệnh được chia làm 4 trường: trường địa chỉ vi lệnh kế tiếp, trường điều khiển nạp bộ đếm vi chương trình, trường chọn lọc điều kiện và trường điều khiển CPU. Hầu hết các bit trong thanh ghi vi lệnh thuộc về trường điều khiển CPU và điều khiển luồng thông tin bên trong CPU bằng việc mở các cổng 3 trạng thái và cấp nhịp lật cho các thanh ghi như đã mô tả trên.

- **Điều khiển chuỗi vi lệnh**

Nếu bộ đếm vi chương trình được đếm các bước 0, 1, 2, ... thì một dòng các vi lệnh kế tiếp nhau sẽ xuất hiện trong thanh ghi lệnh như cách được mô tả trong bảng 2.1. Các bit điều khiển CPU của mỗi vi lệnh sẽ xác định luồng thông tin bên trong CPU. Tuy nhiên, như trong trường hợp của đơn vị điều khiển vĩ mô, thường cần thay đổi chuỗi các vi lệnh được chạy. Thí dụ, chúng ta có thể muốn lặp lại một nhóm các vi lệnh n lần; hoặc muốn nhảy từ pha tìm-nạp lệnh tới pha thực thi lệnh hoặc muốn gọi một chương trình phụ nào đó.

Ba trường thanh ghi sẽ điều khiển chuỗi vi lệnh, cho phép bộ đếm vi chương trình thực hiện cả hoạt động nhảy rẽ nhánh có và không có điều kiện tới vùng bên trong bộ nhớ vi chương trình. Hoạt động này là cần thiết để chạy các lệnh vĩ mô như BRA, BCC, BSC, BEQ v.v... Trong hoạt động bình thường, bộ đếm vi chương trình sẽ đếm qua các lệnh một cách liên tiếp và địa chỉ của vi chương trình kế tiếp là địa chỉ hiện tại cộng với 1. Bằng việc nạp các nội dung của trường địa chỉ vi lệnh kế tiếp của trường vi lệnh hiện tại vào bộ đếm vi chương trình, một hoạt động rẽ nhánh có thể tới được một điểm bất kì trong bộ nhớ vi chương trình. Nói cách khác, mỗi vi lệnh xác định liệu vi lệnh kế tiếp được đưa vào trong chuỗi hoặc đưa ra từ trường địa chỉ kế tiếp của vi lệnh hiện tại. Vậy thì cái gì xác định liệu bộ đếm vi chương trình liên tục theo một chuỗi hoặc được nạp từ *trường địa chỉ vi lệnh kế tiếp* của *vi lệnh hiện tại* ?

Trường điều khiển nạp bộ đếm vi chương trình (microprogram counter load control field) của thanh ghi vi lệnh báo cho bộ đếm vi chương trình hoặc là lấy địa chỉ lệnh kế tiếp của nó bằng việc tăng 1 trong một chuỗi liên tục hoặc là từ việc *ánh xạ địa chỉ* (address mapper) hoặc là dùng địa chỉ trong trường địa chỉ kế tiếp của thanh ghi vi lệnh.

Trường chọn điều kiện (codition select field) của thanh ghi vi lệnh thực hiện nhảy có điều kiện ở mức lệnh vĩ mô bằng việc nhảy có điều kiện ở mức vi lệnh. Trong sơ đồ hình 2.10, một trong bốn điều kiện có thể được chọn. Các điều kiện này nhận được từ ALU và thường là các bit cờ của thanh ghi trạng thái bộ xử lí PSR (hay còn gọi là thanh ghi mã điều kiện CCR -condition code register), thí dụ Z, N, C, V. Trường chọn điều kiện xác định những bit cờ nào được cấp tới lối ra của *bộ dồn kênh chọn điều kiện*. Nếu lối ra của bộ dồn kênh là *true* thì một hoạt động nhảy được thực hiện tới địa chỉ được xác định bởi nội dung của trường địa chỉ vi lệnh kế tiếp, trong trường hợp ngược lại vi chương trình sẽ chạy liên tục theo chuỗi bình thường. Trong hình 2.10 hai trong số các điều kiện nhận được từ CCR và 2 bit luôn là *true* và *false*. Điều kiện *false* ngụ ý rằng "không bao giờ rẽ nhánh" và *true* là "luôn luôn rẽ nhánh" (tức là lệnh *goto*). Một cú nhảy có điều kiện ở mức lệnh vĩ mô (thí dụ, BEQ) được dịch bởi các vi lệnh theo cách sau. Trường chọn điều kiện của vi lệnh chọn bit trạng thái thích hợp của CCR để kiểm tra, thí dụ nếu lệnh vĩ mô là BEQ thì bit Z được chọn. Trường điều khiển nạp bộ đếm vi chương trình bao gồm phép toán (operation) "rẽ nhánh tới địa chỉ trong thanh ghi vi lệnh trong điều kiện được chọn là *true*" . Như vậy, nếu điều kiện được chọn là *true* (tức là Z = 1), một hoạt động nhảy được

thực hiện tới một điểm trong vi chương trình cho phép một bước nhảy tương ứng trong chương trình vĩ mô. Nếu điều kiện được chọn là *false* (tức là Z = 0), chuỗi hiện tại của các vi lệnh bị kết thúc bằng việc khởi phát một chu trình lệnh (nhận và thực hiện lệnh) mới.

- **Thực hiện chu trình lệnh**

Phần đầu tiên của mỗi vi chương trình được thực hiện bởi đơn vị điều khiển phù hợp với pha tìm-nạp lệnh vĩ mô và kết thúc với mã toán lệnh vĩ mô (op-code) được đặt vào các thanh ghi lệnh. Mã toán được nạp tới bộ ánh xạ địa chỉ (address mapper), bộ này tìm một bảng đang chứa địa chỉ khởi phát của vi chương trình cho mỗi mã toán có thể. Như vậy bộ ánh xạ địa chỉ đã dịch một nhóm bit nào đó của mã toán thành vùng vi chương trình tương ứng mà vùng đó sẽ thực hiện mã toán. Sau khi vi chương trình này được chạy, một hoạt động nhảy không điều kiện tới điểm khởi phát của vi chương trình được thực hiện nhằm dịch pha tìm-nạp lệnh vĩ mô và rồi xử lí lại được tiếp tục.

Dĩ nhiên việc mô tả đơn vị điều khiển được vi chương trình kể trên là hết sức đơn giản. Trong thực tế vi chương trình thường bao gồm các phương tiện liên quan đến các ngắt, bộ nhớ chính và các thiết bị vào/ra. Một trong những ưu điểm lớn của đơn vị điều khiển được vi chương trình là khả năng biến đổi nội dung của bộ nhớ vi chương trình (đôi khi được gọi là bộ nhớ điều khiển) và do đó cho phép có thể thiết kế được các lệnh mức máy của riêng người sử dụng.

Thực tế rất có khả năng chọn một tập các vi chương trình nhằm sẽ chạy mã máy của một máy tính khác hoàn toàn. Trong trường hợp này máy tính được coi là mô phỏng máy tính khác. Việc này là hữu ích nếu chúng ta cần thay đổi máy tính cũ của mình thành máy mới mà mã máy riêng của nó không tương thích với các chương trình cũ. Việc mô phỏng áp dụng tới chương trình tồn tại trong mã nhị phân (object) ở đĩa từ. Bằng việc viết các vi chương trình (trên máy mới) để dịch mã máy của máy cũ cho phép có khả năng tận dụng được ưu điểm của máy mới mà vẫn sử dụng được phần mềm cũ.

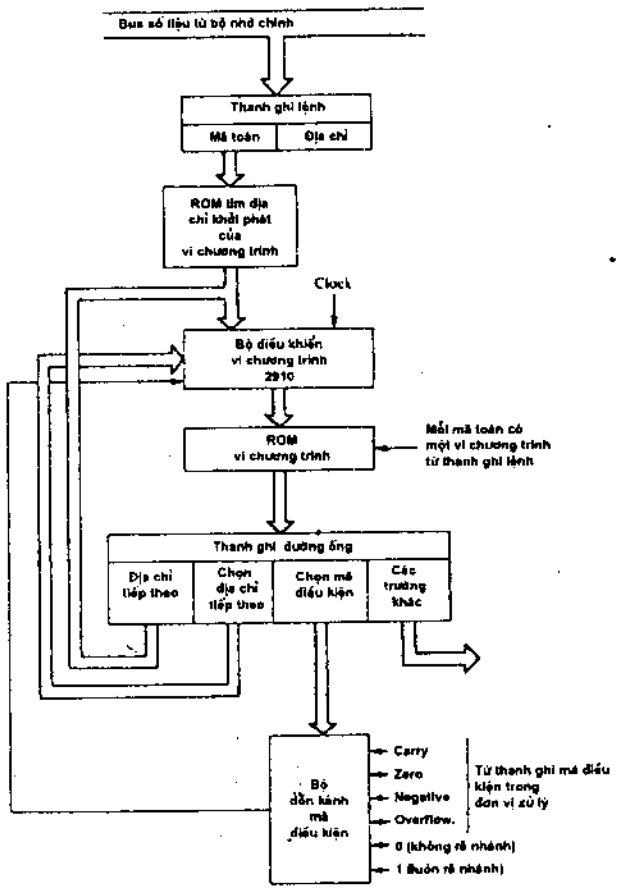
Một vi xử lí lắp trong máy vi tính là một CPU được chế tạo cứng trên một chip (CPU on a chip). Trong khi đó người dùng có thể thiết kế các máy tính số cho mục đích riêng đủ mạnh theo ý mình nhờ vào các chip xử lý bit-slice. Như trên đã nói, chip bit-slice có những chức năng như một phần nào đó của bộ vi xử lí và có thể ghép nối chúng với nhau theo một cách nào đó để tạo nên một CPU tuỳ ý. Các chip bit-slice thường thuộc về một trong hai loại tương ứng với các vùng chức năng bên trong vi xử lí (điều khiển vi chương trình và ALU). Các linh kiện bit-slice điển hình như vi mạch Am2910 thực tế là một đơn vị điều khiển CU và Am2910 là một dạng ALU 4 bit. Các linh kiện *bit-slice* cũng là một cầu nối giữa máy vi tính và máy tính lớn.

Để hiểu cách thức đơn vị điều khiển xác định việc thực hiện một lệnh vĩ mô ta hãy xét cấu trúc cuả một đơn vị điều khiển được vi chương trình tương tự hình 2.10 là hình 2.11. Nếu ta giả thiết rằng một mã toán đang ở trong thanh ghi lệnh của CPU, chuỗi sự kiện sau sẽ xảy ra:

1. Mã toán được cấp tới lối vào của ROM để tra cứu địa chỉ khởi phát của vi lệnh có liên quan trong bảng các địa chỉ. Địa chỉ thứ nhất của vi chương trình được đưa vào bộ đếm vi chương trình của Am2910 và cũng xuất hiện ở lối ra của nó, ở đó nó chọn vi lệnh thứ nhất từ bộ nhớ vi chương trình.
2. Vi lệnh được giữ trong một *thanh ghi đường ống*, gọi như vậy vì nó giữ vi lệnh hiện tại trong khi đơn vị điều khiển đang bận nhận vi lệnh kế tiếp.
3. Vi lệnh được dùng để điều khiển hoạt động của CPU. Vi lệnh điển hình rộng 64 đến 128 bit và được chia thành một số trường. Dưới đây là minh hoạ một vi lệnh giả định.

Địa chỉ rẽ nhánh	Chọn vi lệnh kế tiếp	Chọn mã điều kiện	Điều khiển dịch ALU	Chọn nguồn ALU	Điều khiển dịch ALU	chức năng ALU	Các địa chỉ thanh ghi A và B	Hằng dữ liệu trực tiếp	Điều khiển bus hệ thống
16 bit	4 bit	4 bit	3 bit	3 bit	3 bit	3 bit	8 bit	16 bit	4 bit

Định dạng vi lệnh 64 bit



Hình 2.11 Cấu trúc của một đơn vị điều khiển vi chương trình.

Dưới đây là các thí dụ về các trường vi chương trình

Chọn nguồn

Mã	R	S
0	A	Q
1	A	B
2	0	Q
3	0	B
4	0	A
5	D	A
6	D	Q
7	D	0

Chức năng ALU

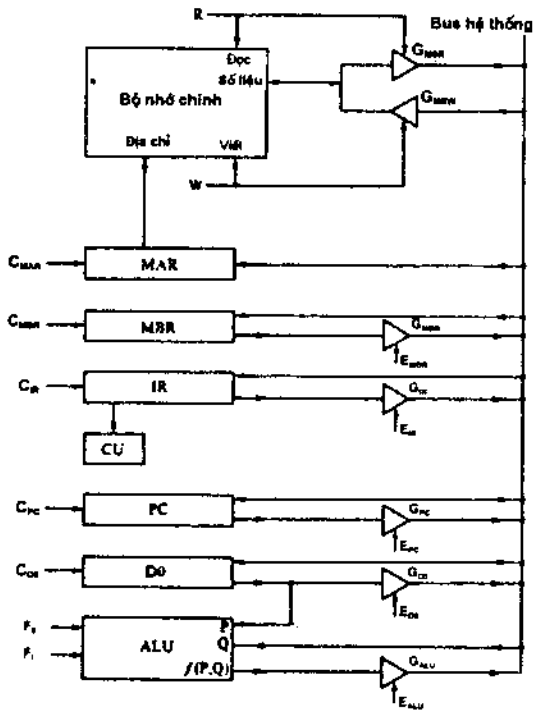
Mã	Chức năng
0	R+S
1	S-R
2	R-S
3	R OR S
4	R AND S
5	R NAND S
6	R EX OR S
7	R EXNOR S

b, Đơn vị điều khiển lôgic ngẫu nhiên

Loại đơn vị điều khiển kể trên thực hiện việc dịch một lệnh máy bằng một vi chương trình được lưu trữ trong bộ nhớ chỉ đọc ROM. Độ phức tạp của đơn vị điều khiển được vi chương trình không liên quan trực tiếp tới độ phức tạp của các lệnh máy mà nó phải dịch cũng giống như trường hợp độ phức tạp của máy tính mức lệnh máy không liên quan gì tới độ phức tạp của các chương trình ngôn ngữ bậc cao đang chạy trên nó. Nhưng đôi khi các nhà thiết kế máy tính lại muốn cung cấp các lệnh mức máy gần gũi với các tiện ích phức tạp được cho bởi các ngôn

ngữ bậc cao. Trong những trường hợp như vậy, phát sinh một mối quan hệ thực sự giữa độ phức tạp của ngôn ngữ bậc cao với độ phức tạp của đơn vị điều khiển. Như phần trên đã nói, cách tiếp cận để xây dựng một đơn vị điều khiển khác là tìm ra chuỗi các phép toán lôgic và số học cần thiết cho việc thực hiện một lệnh rồi thiết kế mạch lôgic thích hợp để thực hiện các chuỗi này. Lúc đó ta có đơn vị điều khiển được xây dựng bằng các mạch lôgic ngẫu nhiên hay *đơn vị điều khiển lôgic ngẫu nhiên* (random logic control unit). Từ ngẫu nhiên ở đây chỉ hàm ý là "không theo một cách thiết kế nhất định nào".

Đơn vị điều khiển được vi chương trình có thể dễ dàng thích ứng với các máy tính khác nhau chỉ với một ít sửa đổi nhưng *đơn vị điều khiển lôgic ngẫu nhiên* thì chỉ được dùng cho một CPU nhất định và không dễ sửa đổi được. Ta sẽ khảo sát một CPU đơn giản với đơn vị điều khiển lôgic ngẫu nhiên như hình 2.12. Một bus đơn nối tất cả các thanh ghi cho phép truyền chỉ một từ số liệu một lần từ nguồn tới đích. Bộ nhớ chính nhận các địa chỉ của vùng nhớ cần thâm nhập một cách trực tiếp từ thanh ghi MAR mà lối ra của nó được nối tới lối vào địa chỉ của bộ nhớ chính MS. Điều kiện này làm đơn giản đi các mạch điều khiển bus. Bus số liệu 2 hướng được nối tới bus hệ thống bằng các cổng 3 trạng thái G_{MSR} và G_{MSW}. Trong chu kì viết bộ nhớ (W là *true*), số liệu được truyền từ bus hệ thống tới bộ nhớ qua cổng G_{MSW}. Trong chu kì đọc (R là true), số liệu được truyền từ bộ nhớ tới bus hệ thống qua cổng G_{MSR}.



Hình 2.12 Cấu trúc của CPU đơn giản dùng để khảo sát cấu tạo của đơn vị điều khiển lôgic ngẫu nhiên

Các thanh ghi MBR, thanh ghi số liệu, bộ đếm chương trình và thanh ghi lệnh được sắp đặt với bus nội bộ theo cùng một cách. Khi một trong những thanh ghi này muốn đưa số liệu lên bus, cổng 3 trạng thái của nó được mở. Ngược lại, số liệu từ bus được nạp vào một thanh ghi bởi một xung nhịp đưa tới thanh ghi. Lưu ý rằng thanh ghi lệnh IR được sắp đặt nhằm cho nó có thể nhận số liệu trực tiếp từ bộ nhớ chính mà không phải qua MBR như CPU đã giả định trước đây. ALU nhận số liệu từ 2 nguồn, bus nội bộ và thanh ghi số liệu, rồi đưa các tín hiệu ra của nó tới bus hệ thống. Điều này nảy sinh vấn đề: ALU làm sao đặt số liệu lên cùng một bus trong cùng một thời gian nếu nó nhận số liệu từ bus hệ thống ? Có 2 cách giải quyết. Cách thứ nhất là phải đặt một thanh ghi trong ALU để giữ số liệu vào không đổi trong khi bus được nối tới lối ra của ALU. Nói cách khác ALU có một thanh ghi MBR của riêng nó. Cách thứ hai dựa vào hiệu ứng thời gian trễ của các tín hiệu khi đi qua một mạng điện lôgic. Số liệu truyền từ lối vào của ALU tới lối ra mất chừng 50ns.

Giả thiết rằng các nội dung của MBR được cấp tới ALU bằng việc mở cổng G_{MBR} và thời gian đủ đã trôi qua cho tín hiệu lối ra của ALU ổn định. Nếu bây giờ cổng G_{MBR} bị cấm, tín hiệu lối vào ALU sẽ không còn ổn định. Tuy vậy, tín hiệu ra không thể thay đổi ngay lập tức vì có thời gian trễ bên trong kể trên. Kết quả là nếu cổng G_{ALU} được mở và thanh ghi số liệu được cấp xung nhịp, bus hệ thống có thể được sử dụng cho việc truyền số liệu trước khi lối ra của ALU thay đổi. ALU được điều khiển bởi mã 2 bit, F0 và F1 có chức năng như bảng 2.3. Để cho thiết kế của đơn vị điều khiển lôgic ngẫu nhiên đơn giản ta sẽ thiết kế một mã toán 3 bit cho tổng số 8 lệnh như bảng 2.4. Bảng 2.5 liệt kê cả các tín hiệu nhịp đồng hồ, được kí hiệu là C_R và tất cả các tín hiệu điều khiển cổng 3 trạng thái, kí hiệu là E_D, cần để chạy mỗi lệnh. Kí hiệu Z là bit cờ zero từ CCR được giả thiết là một phần của ALU.

Bảng 2.3 **Mã điều khiển ALU**

F_1	F_0	Chức năng
0	0	Cộng P với Q
0	1	Trừ P đi Q
1	0	Tăng Q lên 1
1	1	Giảm Q đi 1

Bảng 2.4 **Tập lệnh đơn giản cho CPU ở hình 2.12**

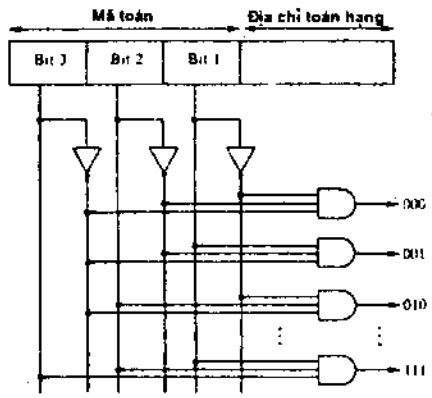
Op-code	Từ gợi nhớ	Phép toán
000	LOAD M	[D0] ← [MS(M)]
001	STORE M	[MS(M)] ← [D0]
010	ADD M	[D0] ← [D0] + [MS(M)]
011	SUB M	[D0] ← [D0] - [MS(M)]
100	INC M	[MS(M)] ← [MS(M)] + 1
101	DEC M	[MS(M)] ← [MS(M)] - 1
110	BRA M	[PC] ← M
111	BEQ M	IF Z=1 THEN [PC] ← M

Bảng 2.5 **Dịch tập lệnh ở bảng 2.4 trong ngôn ngữ RTL và các vi lệnh**

Lệnh	Op-code	Các hoạt động (RTL)	Các hoạt động điều khiển
Fetch	-	[MAR] ← [PC]	E_{PC}=1 C_{MAR}
		[IR] ← [MS([MAR])]	R=1 C_{IR}
		ALU ← [PC]	E_{PC}=1 F_1, F_0 =1,0
		[PC] ← ALU	E_{ALU}=1 C_{PC}
LOAD	000	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		[D0] ← [MS([MAR])]	R=1 C_{D0}
STORE	001	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		MS([MAR])) ← [D0]	E_{D0}=1 W=1
ADD	010	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		[MBR] ← [MS([MAR])]	R=1 C_{MBR}
		ALU ← [MBR]	E_{MBR}=1 F_1, F_0 =0,0
		D0 ← ALU	E_{ALU}=1 C_{D0}
SUB	011	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		MBR] ← [MS([MAR])]	R=1 C_{MBR}
		ALU ← [MBR]	E_{MBR}=1 F_1, F_0 =0,1
		D0 ← ALU	E_{ALU}=1 C_{D0}

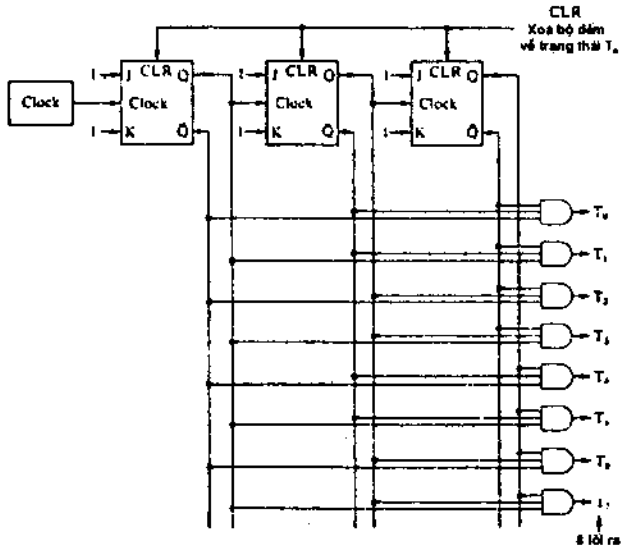
INC	100	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		MBR] ← [MS([MAR])]	R=1 C_{MBR}
		ALU ← [MBR]	E_{MBR}=1 F_1, F_0 =1,0
		[MBR] ← ALU	E_{ALU}=1 C_{MBR}
		[MS([MAR])] ← [MBR]	E_{MBR}=1 W=1
DEC	101	[MAR] ← [IR]	E_{IR}=1 C_{MAR}
		MBR] ← [MS([MAR])]	R=1 C_{MBR}
		ALU ← [MBR]	E_{MBR}=1 F_1, F_0 =1,1
		[MBR] ← ALU	E_{ALU}=1 C_{MBR}
		[MS([MAR])] ← [MBR]	E_{MBR}=1 W=1
BRA	110	[PC] ← [IR]	E_{IR}=1 C_{PC}
BEQ	111	IF Z=1 THEN [PC] ← [IR]	IF Z=1 THEN E_{IR}=1, C_{PC}

Để chạy một lệnh phải thực hiện 2 việc: giải mã toán 3 bit thành 8 hoạt động kế tiếp nhau và thực hiện các hoạt động đó. Hình 2.13 là cấu trúc của một bộ giải mã 3 thành 8. Với một giá trị nhị phân 3 bit đầu vào chỉ có một trong 8 đầu ra được đặt ở mức tích cực cao, các đầu ra còn lại ở mức thấp. Thí dụ, nếu mã tương ứng với lệnh cộng ADD (tức là 010) được nạp vào thanh ghi IR trong pha nhận lệnh thì lối ra 2 từ dãy cổng AND sẽ ở mức lôgic cao. Để thực hiện chuỗi các vi lệnh cần phải có nguồn tạo ra các xung khởi phát (trigơ) các vi lệnh. Mạch điện tạo ra một chuỗi các tín hiệu trigơ đó gọi là bộ tạo xung tuần tự (sequencer).



Hình 2.13 Bộ giải mã lệnh (3 thành 8).

Hình 2.14 là sơ đồ một bộ tạo xung tuần tự 8 bước. Các lối ra của 3 trigơ J-K được thiết kế thành một bộ đếm tiến 3 bit và được nối tới 8 cổng AND 3 đầu vào để cung cấp các tín hiệu nhịp T_0-T_7. Hình 2.15 là giản đồ thời gian của các xung điều khiển. Lưu ý rằng bộ giải mã thời gian ở đây cũng giống như bộ giải mã lệnh trên hình 2.13. Vì không phải tất cả các lệnh vĩ mô đều đòi hỏi một số vi lệnh như nhau, do vậy có thêm một lối vào Reset cho phép đặt lại bộ tạo xung tuần tự trở về trạng thái T_0 ở bất kì thời điểm nào. Bước tiếp theo là kết hợp các tín hiệu từ bộ giải mã lệnh với các tín hiệu điều chỉnh (timing signals) từ

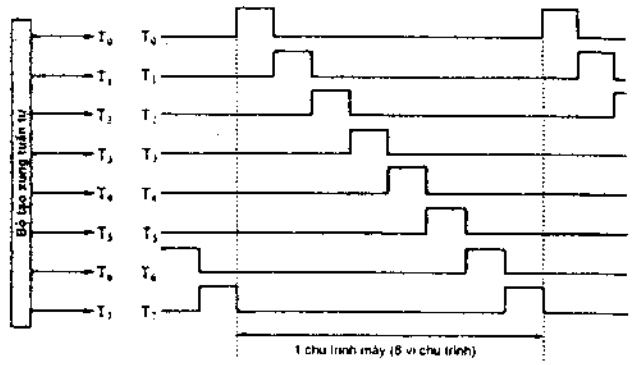


Hình 2.14 Bộ tạo xung tuần tự 8 bước.

sequencer để phát ra các tín hiệu điều khiển thực sự.

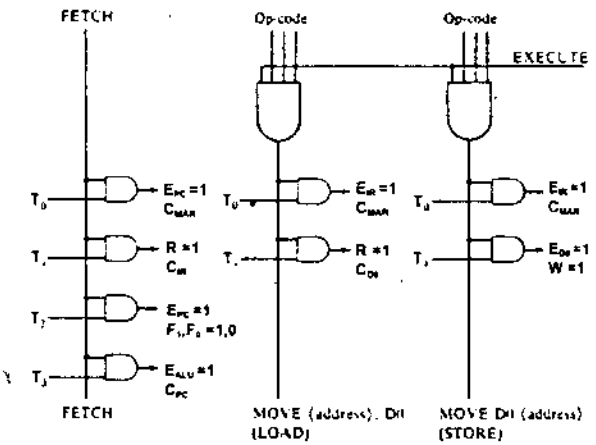
Hình 2.16 chỉ ra một trong những cách giải quyết việc này. Đối với mỗi một trong 8 lệnh mức máy, một trong các đường dây dọc từ bộ giải mã lệnh sẽ ở mức lôgic 1 cho phép mở cổng AND mà nó nối tới. Khi các tín hiệu nhịp thời gian T_0 đến T_7 được phát ra, các lối ra của các cổng AND sẽ cho phép đơn vị điều khiển lôgic ngẫu nhiên hoạt động.

Lối ra của mỗi cổng AND, tương ứng với một vi lệnh nhất định (thí dụ C_{MAR}), được nối tới một cổng OR nhiều lối vào.



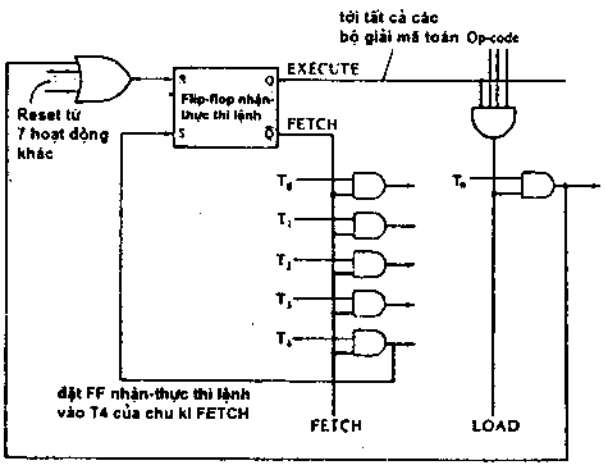
Hình 2.15 Giản đồ thời gian của các xung điều khiển.

Lối ra của các cổng OR này sẽ khởi phát các vi lệnh tương ứng. Trong hình này, các cổng AND với lối vào EXECUTE là phần giải mã trong sơ đồ hình 2.13. Đối với một vi lệnh, một nhóm các tín hiệu điều khiển có thể được kích hoạt tại một thời gian xác định. Như vậy mỗi tín hiệu điều khiển có thể được kích hoạt bởi một vài lệnh vĩ mô (có khả năng diễn ra trong các chu kì nhịp đồng hồ khác nhau) và được phát bởi một máy phát vòng. Đơn vị điều khiển luôn ở một trong 2 trạng thái, là pha nhận lệnh (fetch) hoặc pha thực thi lệnh (execute).



Hình 2.16 Đơn vị điều khiển lôgic ngẫu nhiên.

Hình 2.17 là một phần mở rộng sơ đồ hình 2.16, trong đó có một mạch lật RS được dùng để chuyển giữa 2 trạng thái này. Khi Q = 0, hoạt động hiện tại là pha nhận lệnh và khi Q = 1 là pha thực thi lệnh. Ở cuối của mỗi pha nhận lệnh, một xung Clock từ máy phát nhịp đặt mạch lật này lên 1 cho phép lệnh hiện tại được giải mã và chạy. Bộ tạo xung tuần tự cũng được reset ở cuối pha nhận lệnh. Cuối



Hình 2.17 Mạch lật nhận-thực thi lệnh.

mỗi pha thực thi lệnh, mạch lật được xoá về 0 và Lộ tạo xung tuần tự sẽ được reset, cho phép pha nhận lệnh kế tiếp được bắt đầu. Bảng 2.6 chỉ ra các lệnh mức máy có thể được trình bày theo cả 2 loại thuật ngữ: tín hiệu phụ thuộc thời gian và vi lệnh. Các vi lệnh là: cho phép mở bộ điều khiển bus, nhịp cho các thanh ghi, các bit chọn chức năng của ALU, các tín hiệu điều khiển đọc/viết bộ nhớ chính và đặt/xoá mạch lật nhận-thực thi lệnh. Với mỗi vi lệnh ta có thể viết một biểu thức Boolean theo dạng lệnh mức máy và một chuỗi các xung theo thời gian.

Bảng 2.6 **Diễn giải các lệnh máy**

Lệnh	Thời gian	ENABLE		Clock						ALU				MS	FF		
		MBR	IR	PC	D0	ALU	MAR	MBR	IR	PC	D0	F1	F0	R	W	R	S
FETCH	T0	0	0	1	0	0	1	0	0	0	0	x	x	0	0	0	0
	T1	0	0	0	0	0	0	0	1	0	0	x	x	1	0	0	0
	T2	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
	T3	0	0	0	0	1	0	0	0	1	0	x	x	0	0	0	0
	T4	0	1	0	0	0	1	0	0	0	0	x	x	0	0	0	1
LOAD	T0	0	0	0	0	0	0	0	0	0	1	x	x	1	0	1	0
STORE	T0	0	0	0	1	0	0	0	0	0	0	x	x	0	1	1	0
ADD	T0	0	0	0	0	0	0	0	0	0	0	x	x	1	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	T2	0	0	0	0	1	0	0	0	0	1	x	x	0	0	1	0
SUB	T0	0	0	0	0	0	0	1	0	0	0	x	x	1	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
	T2	0	0	0	0	1	0	0	0	0	1	x	x	0	0	1	0
INC	T0	0	0	0	0	0	0	1	0	0	0	x	x	1	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
	T2	0	0	0	0	1	0	1	0	0	0	x	x	0	0	0	0
	T3	1	0	0	0	0	0	0	0	0	0	x	x	0	1	1	0
DEC	T0	0	0	0	0	0	0	1	0	0	0	x	x	1	0	0	0
	T1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
	T2	0	0	0	0	1	0	1	0	0	0	x	x	0	0	0	0
	T3	1	0	0	0	0	0	0	0	0	0	x	x	0	1	1	0
BRA	T0	0	1	0	0	0	0	0	0	1	0	x	x	0	0	1	0
BEQ	T0	0	1	0	0	0	0	0	0	z	0	x	x	0	0	1	0

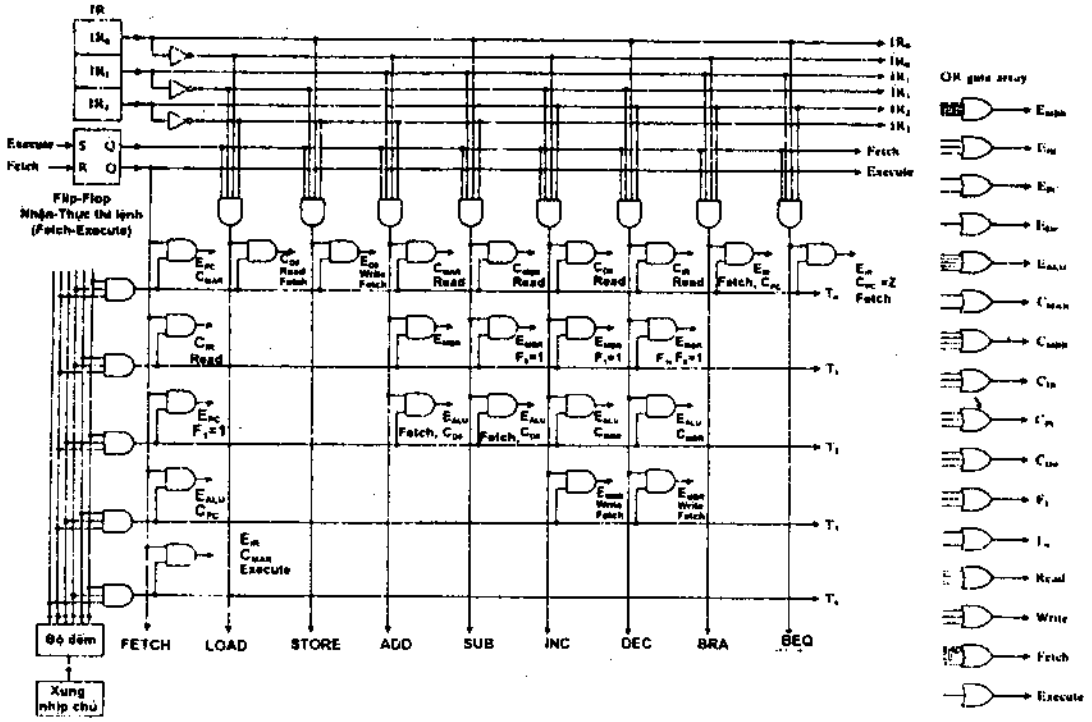
Thí dụ, xem biểu thức cho E_{MBR}, E_{IR}, C_{MAR}.

$$E_{MBR} = ADD\ T_1 + SUB\ T_1 + INC\ T_1 + INC\ T_3 + DEC\ T_1 + DEC\ T_1 + DEC\ T_3$$

$$E_{IR} = Fetch\ T_4 + BRA\ T_0 + BEQ\ T_0$$

C_{MAR} = Fetch T_0 + Fetch T_1

Hình 2.18 là sơ đồ chi tiết của một đơn vị điều khiển logic ngẫu nhiên.



Hình 2.18 Sơ đồ chi tiết của một đơn vị điều khiển lôgic ngẫu nhiên.

So sánh giữa 2 thiết kế đơn vị điều khiển theo kiểu vi chương trình hoá và lôgic ngẫu nhiên ta thấy:

- Đơn vị điều khiển lôgic ngẫu nhiên có tốc độ hoạt động nhanh hơn vì nó được tối ưu hoá cho một ứng dụng cụ thể. Hơn nữa loại vi chương trình hoá thì chậm hơn vì nó cần phải mất thời gian đọc vi lệnh từ bộ nhớ vi chương trình.
- Vi chương trình thường cho một thiết kế mềm dẻo hơn. Vì nó tồn tại trong bộ nhớ ROM nên có thể dễ dàng sửa đổi nội dung trong giai đoạn thiết kế hoặc sản xuất.
- Vi chương trình có ứng dụng cao trong những năm 70 khi mà tốc độ của bộ nhớ chính còn chậm (1-2μs) trong khi thời gian thâm nhập bộ nhớ điều khiển chỉ chừng 50-100ns. Khi đó các lệnh mã máy phức tạp sẽ được chạy rất nhanh nhờ vi mã. Nhưng đến hiện nay, do thời gian thâm nhập bộ nhớ chính thường thấp hơn 100ns nên các đơn vị điều khiển lôgic ngẫu nhiên cứng (hard) có thể được thiết kế để chạy các lệnh nhanh hơn nhiều các đơn vị điều khiển vi mã. Nhiều vi xử lí mạnh hiện nay được vi chương trình hoá bên trong nhưng người dùng không thể sửa đổi được vi chương trình. Các thế hệ xử lí có kiến trúc RISC thì không phải là vi chương trình hoá.

2.3. CÁC BỘ XỬ LÍ RISC

Các vi xử lí hoạt động theo nguyên tắc kể trên có số lượng lệnh vĩ mô lớn (thường hơn 300 lệnh), có các khả năng định địa chỉ phức tạp, các lệnh này được thực hiện bằng một chuỗi vi lệnh, vi chương trình. Do vậy các vi xử lí này được gọi là loại có *tập lệnh phức*

hợp CISC (Complex Instruction Set Computer). Vi chương trình được phát triển mạnh trong những năm 70 khi mà bộ nhớ chính có tốc độ truy nhập tương đối chậm. Lúc đó các nhà thiết kế máy tính sử dụng bộ nhớ chính tốc độ chậm để lưu trữ các lệnh phức tạp tạo nên chương trình ở mức máy. Các lệnh đó sẽ được dịch bằng vi mã trong bộ nhớ điều khiển vi chương trình có tốc độ nhanh hơn. Ngày nay, các bộ nhớ chính có thời gian truy cập ngắn hơn 80 ns làm cho các ưu điểm kể trên của vi chương trình không những biến mất mà còn gây cản trở nữa cho hệ thống. Thay vào đó là một kiểu kiến trúc vi xử lí mới có tên gọi là RISC cho phép thực hiện một mã toán chỉ trong một chu kì máy.

Trong hàng thập kỉ, các nhà khoa học máy tính đã nghiên cứu cách thức mà các máy tính hiện nay chạy c' ương trình. Điều tra thống kê đã chỉ ra những kết luận s au:

• Trong nhiều chương trình điển hình được xử lí bằng bộ vi xử lí CISC như nói trên thì đến 80% thời gian chạy chương trình được thực hiện chỉ bởi 20% số lệnh trong tập lệnh.

Thí dụ, tác giả Fairclough đã chia các lệnh ra thành 8 nhóm và đã thống kê được các kết quả trung bình tần suất sử dụng (xét về cả chương trình và kiến trúc máy tính) của chúng như sau:

Nhóm	1	2	3	4	5	6	7	8
Tần suất %	45,28	28,73	10,75	5,92	3,91	2,93	2,05	0,44

Trong đó:

1 là nhóm các lệnh chuyển dữ liệu
2 là nhóm các lệnh thay đổi chương trình (lệnh rẽ nhánh -branch, gọi-call, trả về-return)
3 là nhóm các lệnh tính số học
4 là nhóm các lệnh so sánh
5 là nhóm các lệnh lôgic
6 là nhóm các lệnh dịch
7 là nhóm các lệnh xử lí bit
8 là nhóm các lệnh vào/ra.

Rõ ràng trong tổng số 8 nhóm lệnh thì 2 nhóm lệnh chuyển dữ liệu và thay đổi chương trình có tần suất sử dụng chiếm đến 74% .

• Có những trường hợp, để đạt được cùng một kết quả thì việc chạy một chuỗi lệnh đơn giản sẽ nhanh hơn là chạy một lệnh phức tạp.

Những kết luận này là tiền đề để một loại vi xử lí khác ra đời; nó hoạt động theo phương pháp sử dụng *tập lệnh ngắn gọn* RISC (Reduced Instruction Set Computer). Tập lệnh ở đây được hạn chế vào các lệnh chính yếu cần thiết trong số 20% lệnh nói trên và phần cứng được thiết kế sao cho các phần tử RISC này hiểu ngay được lệnh máy do chương trình cung cấp. Do đó, các lệnh loại này không cần chuyển cho bộ giải mã vi lệnh mà có thể thực hiện được ngay.

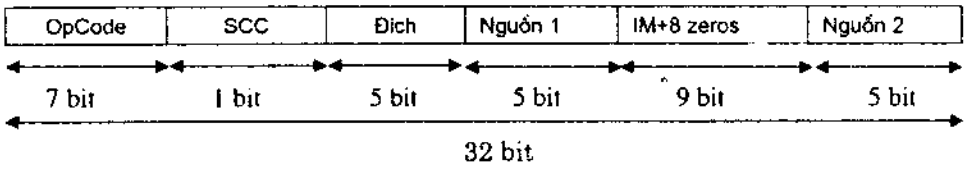
2.3.1. Các đặc điểm của kiến trúc RISC

Kiến trúc RISC cần có những đặc điểm sau:

1. Cần có đủ số thanh ghi (thực chất là các bộ nhớ ngay trong chip, có thời gian truy cập nhanh hơn so với bộ nhớ ngoài) để khắc phục được hiệu ứng nẹt cổ chai (bottleneck).

2. Vì các lệnh trong nhóm 2 (call, return,...) thường xảy ra nên một kiến trúc hiệu dụng cần tạo sự dự phòng cho việc chuyển hiệu quả các tham số giữa các thủ tục.
3. Các bộ xử lí RISC không nên thực hiện thường xuyên các lệnh đã được sử dụng. Các lệnh phức tạp làm tăng thời gian thiết kế, sản xuất và thử nghiệm.
4. Nếu một bộ xử lí RISC đủ nhanh thì nó phải chạy được một lệnh trong một chu kì máy. Điều này cũng có nghĩa là sự phức tạp của các lệnh phải có giới hạn của chúng.
5. Kết quả tất yếu suy ra từ điều trên là một kiến trúc có hiệu quả không nên được vi chương trình hoá vì nó phải dịch một lệnh bằng việc chạy các vi lệnh. Một bộ xử lí RISC gần với một kiến trúc vi chương trình trong đó sự phân biệt giữa chu trình máy và chu trình vi lệnh bị biến mất.
6. Bộ xử lí hiệu quả phải có một định dạng lệnh đơn. Khi đó việc giải mã lệnh có thể được thực hiện bằng một mạch lôgic giải mã tối thiểu.
7. Để giảm thời gian chạy lệnh, các lệnh dài có liên quan tới các vi xử lí thế hệ 1 và 2 cần phải bỏ đi.

Đến đây ta sẽ xét 3 khía cạnh cơ bản của kiến trúc RISC: dùng một số lớn các thanh ghi được chế tạo trên chip (on-chip), dùng các cửa sổ thanh ghi gối lên nhau để giảm một cách đáng kể sự truyền tham số giữa các thủ tục và cơ chế đường ống (pipelining) trong đó cho phép sự gối lên nhau trong việc thực hiện lệnh. Trong kiến trúc RISC do nhóm Berkeley đề xướng sử dụng định dạng lệnh 3 thanh ghi như dưới đây. Mỗi trường toán hạng cho phép một trong số 32 thanh ghi bên trong được truy cập.

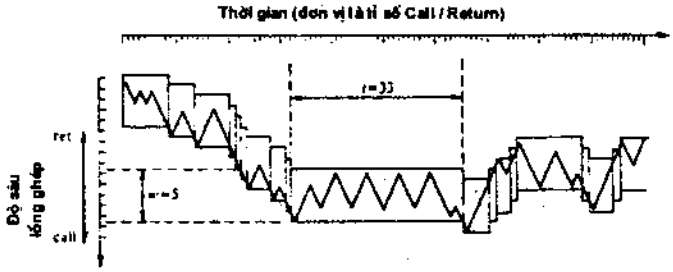


Hai trường SCC và IM+8 zeros liên quan đến việc cập nhật thanh ghi mã điều kiện và toán hạng tức thời. Định dạng RISC cho phép các lệnh dạng ADD X,Y,Z trong đó X, Y, Z là các thanh ghi bên trong. Vì rằng mỗi trường toán hạng dài 5 bit nên RISC có $2^5 = 32$ thanh ghi bên trong. Nhưng trong thực tế, RISC có đến 138 thanh ghi đa năng bên trong dùng cho người sử dụng. Nguyên nhân khác nhau này là do một cơ chế gọi là cửa sổ, cơ chế đó cho phép người lập trình chỉ nhìn thấy một tập nhỏ trong số toàn bộ các thanh ghi trong một thời điểm nào đó.

2.3.2. Các cửa sổ thanh ghi

Một đặc điểm thiết kế quan trọng trong kiến trúc RISC mà nhóm Berkeley nêu ra là cách thức nó có thể định vị không gian thanh ghi tới mỗi thủ tục mới (perocedure). Mặc dù chỉ có 12 thanh ghi được đòi hỏi bởi mỗi lời gọi thủ tục; việc lồng liên tiếp các thủ tục sẽ làm tăng nhanh chóng số các thanh ghi trên chip được gán cho các thủ tục (nếu mỗi thủ tục đều có các biến cục bộ của riêng chúng). Có thể nghĩ rằng ý định dành một tập các thanh ghi cho mỗi thủ tục mới là không thực tế vì rằng gọi một chuỗi các thủ tục lồng lên nhau sẽ đòi hỏi một lượng bộ nhớ quá giới hạn. May thay, mặc dù về nguyên tắc các thủ tục có thể lồng lên nhau với mức độ tuỳ ý nhưng các công trình nghiên cứu đã chỉ ra rằng hầu hết các thủ tục không lồng lên nhau với mức độ lớn bất kì. Trên hình 2.19 cho thấy mỗi lệnh CAll được biểu diễn bằng một đường đi xuống về phía phải và mỗi lệnh RETURN cũng vậy. Trong hình có vẽ một cửa sổ có độ sâu lồng ghép là 5. Ta thấy rằng chỉ có một dãy lệnh CALL và RETURN có độ sâu lồng ghép

thủ tục là 6. Vậy chương trình vẫn có thể tiếp tục được thực hiện trong cửa sổ đó một giai đoạn dài.



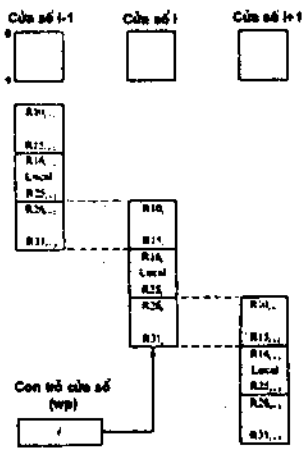
Hình 2.19 Các thủ tục lồng ghép là hàm của thời gian.

Do vậy, việc chọn một số hạn chế các thanh ghi cục bộ cho một chuỗi các thủ tục lồng nhau là hoàn toàn khả thi. Một cơ chế thực hiện không gian làm việc biến cục bộ cho các thủ tục được các nhà thiết kế RISC Berkeley đưa ra là quản lí tới 8 thủ tục lồng nhau bằng việc cung cấp không gian làm việc trên chip cho mỗi thủ tục. Nếu mức độ lồng nhau lớn hơn sẽ buộc CPU phải sử dụng không gian bộ nhớ ngoài cho việc lưu trữ các biến cục bộ. Không gian bộ nhớ được dùng bởi các thủ tục có thể được chia làm 4 loại:

1. *Không gian toàn cầu* là một vùng không gian địa chỉ lôgic có thể được thâm nhập bởi tất cả các thủ tục. Nó được dùng để giữ các hằng số và số liệu có thể được đòi hỏi từ bất kì điểm nào trong chương trình. Hầu hết các vi xử lí chỉ có các thanh ghi toàn cục bên trong.
2. *Không gian cục bộ* là vùng không gian địa chỉ lôgic dành riêng cho thủ tục. Nó được dùng như một không gian làm việc cho thủ tục hiện tại.
3. *Không gian tham số nhập liệu* giữ các tham số được nhập bởi thủ tục hiện tại từ thủ tục mẹ của nó gọi là các thanh ghi "cao".
4. *Không gian tham số xuất liệu* giữ các tham số được xuất bởi thủ tục hiện tại tới các thủ tục con của nó gọi là các thanh ghi "thấp".

2.3.3. Các cửa sổ và việc truyền tham số

Một trong những lí do làm cho tần suất sử dụng các phép toán chuyển số liệu trở nên lớn là vì việc cần thiết phải truyền các tham số tới các thủ tục và nhận chúng từ các thủ tục. Một hệ thống giảm được việc truyền số liệu này sẽ cho phép tăng đáng kể tốc độ chạy chương trình. Kiến trúc RISC Berkeley giải quyết vấn đề truyền tham số bằng phương pháp đa cửa sổ lồng nhau. Một cửa sổ là một tập các thanh ghi khả kiến với một thủ tục hiện tại. Hình 2.20 minh hoạ cấu trúc của các cửa sổ lồng nhau của RISC. Giả thiết rằng bộ xử lí hiện đang dùng tập cửa sổ thứ i. Một thanh ghi đặc biệt gọi là con trỏ cửa sổ WP (window pointer) chỉ thị cửa sổ được kích hoạt hiện tại.



Hình 2.20 Cấu trúc các cửa sổ lồng nhau trong RISC.

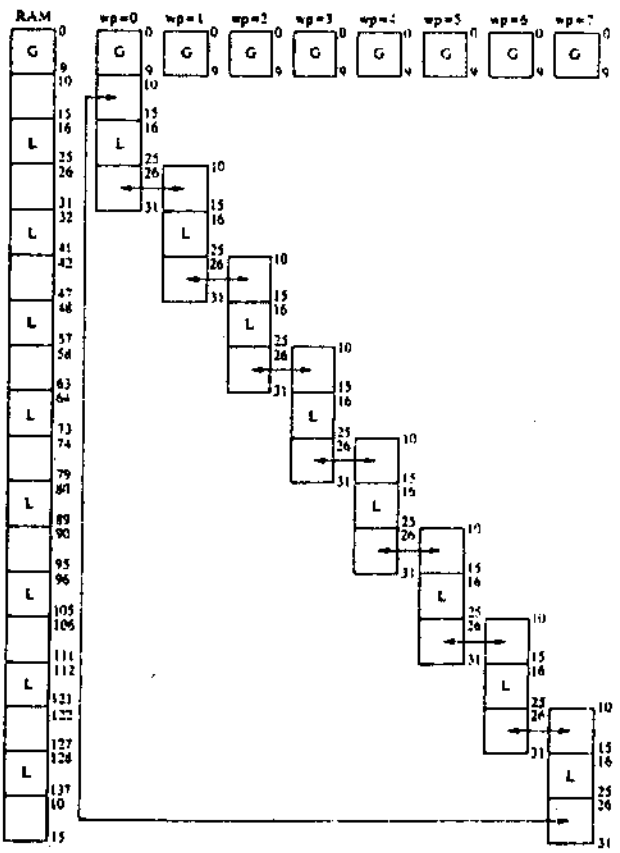
Trong trường hợp này, WP chứa giá trị i. Như mô tả ở trên, mỗi cửa sổ được chia thành 4 phần: các thanh ghi R0-R9 là tập các thanh ghi toàn cục, các thanh ghi

R10-R15 được dùng bởi thủ tục để nhận các tham số từ mẹ của nó và truyền các tham số quay trở lại mẹ của nó, các thanh ghi R16-R25 là mười thanh ghi không thể truy cập được trực tiếp bởi bất kì thủ tục khác, các thanh ghi R26-R31 được dùng bởi thủ tục để truyền và nhận các tham số từ con của nó (tức là thủ tục được gọi bởi chính nó).

Tất cả các cửa sổ bao gồm 32 thanh ghi, R0-R31, và mỗi một trong những thanh ghi này được địa chỉ hoá bởi một lệnh RISC gồm 5 bit địa chỉ trên trường toán hạng. Trong phần này ta sẽ sử dụng từ gợi nhớ để mô tả hoạt động của RISC. Một lệnh có dạng ADD R3, R12. R25 là [R25] ← [R3]+[R12], ở đây R3 bên trong không gian địa chỉ toàn cục, R12 được nhập (hoặc xuất) từ không gian thủ tục mẹ và R25 trong không gian địa chỉ cục bộ. Lưu ý rằng các lệnh số học và lôgic RISC luôn có giá trị 32 bit.

Bất cứ khi nào một thủ tục được gọi bởi một lệnh dạng CALLR Rd,<địa chỉ>, các nội dung của con trỏ cửa sổ được tăng lên 1 và giá trị hiện tại của bộ đếm chương trình được cất trong thanh ghi Rd của cửa sổ mới. Lưu ý rằng kiến trúc RISC Berkeley không dùng ngăn xếp thông thường trong bộ nhớ chính để cất các địa chỉ trở về của chương trình con.

Khi một cửa sổ mới được gọi ra (trong hình 2.20 là cửa sổ i), thủ tục mới sẽ nhìn một tập các thanh ghi khác với cửa sổ trước đó. Các thanh ghi toàn cục R0-R9 là một ngoại lệ vì chúng là chung cho mọi cửa sổ. Cửa sổ R10 của thủ tục con giống với cửa sổ R26 của thủ tục gọi (tức là thủ tục mẹ). Cấu trúc toàn bộ của một hệ cửa sổ RISC được cho trên hình 2.21. Bên trái hình là dãy thanh ghi thực tế gồm tất cả các thanh ghi đa năng trên chip. Tám cột liên quan tới các cửa sổ từ 0-7 chỉ ra cách thức mỗi cửa sổ được định bản đồ trong dãy nhớ vật lí và cách thức tổ chức các vùng lồng lên nhau. Lưu ý rằng các cửa sổ được tổ chức theo kiểu vòng tròn để cho cửa sổ 0 theo sau cửa sổ 7 và cửa sổ 7 đi trước cửa sổ 0. Tổng số các thanh ghi được đòi hỏi là: 10 toàn cục + 8x10 cục bộ + 8x6 truyền tham số = 138 thanh ghi.



Hình 2.21 Cấu trúc chi tiết của các cửa sổ RISC.

2.3.4. Tràn cửa sổ

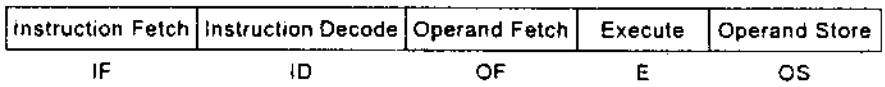
Tuy nhiên là các tài nguyên của bất kì bộ xử lí nào cũng đều có giới hạn; như trong trường hợp kiến trúc RISC của Berkeley các thanh ghi được hạn chế tới 8 cửa sổ. Nếu các thủ tục lồng lên nhau với mức độ lớn hơn hoặc bằng 8 thì sẽ xảy ra việc tràn cửa sổ vì rằng không còn một cửa sổ mới nào cho sự gọi một thủ tục tiếp theo. Khi xảy ra tràn cửa sổ, điều duy nhất làm được

là dùng bộ nhớ chính để lưu giữ các số liệu tràn. Trong thực tế thường cửa sổ già nhất được cất thay vì cửa sổ mới vừa được tạo nên bởi thủ tục vừa được gọi. Nếu số thủ tục được trả về (return) trừ đi số thủ tục được gọi (call) vượt quá 8 sẽ xảy ra tràn cửa sổ. Cửa sổ không tràn bằng nghịch đảo của tràn cửa sổ và cửa sổ trẻ nhất được cất trong bộ nhớ chính phải được trả về một cửa sổ.

2.3.5. Kiến trúc RISC và xử lí đường ống

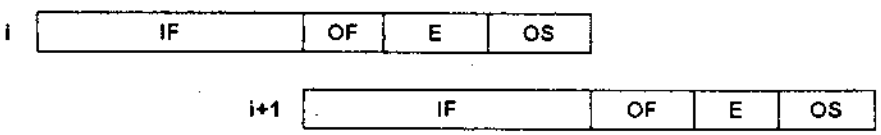
Hình 2.22 minh hoạ một chu trình máy của một vi xử lí giả định đơn giản thực hiện một lệnh dạng ADD P (tức là [A] ← [A] + [M(P)]), ở đây A là một thanh ghi tích luỹ trên chip và P là một vùng nhớ nào đó. Lệnh này được thực hiện qua 5 giai đoạn:

1. *Nhận lệnh* (instruction fetch): đọc lệnh từ bộ nhớ hệ thống và tăng bộ đếm chương trình lên 1.
2. *Giải mã lệnh* (instruction decode): giải mã lệnh được đọc từ bộ nhớ trong giai đoạn trước. Bản chất của giai đoạn này rất phụ thuộc vào tính phức tạp của việc lập mã lệnh. Một lệnh thông thường có thể được giải mã trong một vài nanô giây trong khi một lệnh phức tạp có thể đòi hỏi phải qua một bảng tra cứu nạp sẵn trong ROM để giải mã.
3. *Nhận toán hạng* (operand fetch): toán hạng lệnh được đọc từ bộ nhớ hệ thống hoặc từ một thanh ghi trên chip rồi được nạp vào CPU.
4. *Thực hiện lệnh* (Execute): phép toán lệnh được thực hiện.
5. *Lưu trữ toán hạng* (Operand store): kết quả của giai đoạn thực hiện lệnh được viết vào đích toán hạng. Đó có thể là một thanh ghi trên chip hoặc một vùng nhớ ngoài.



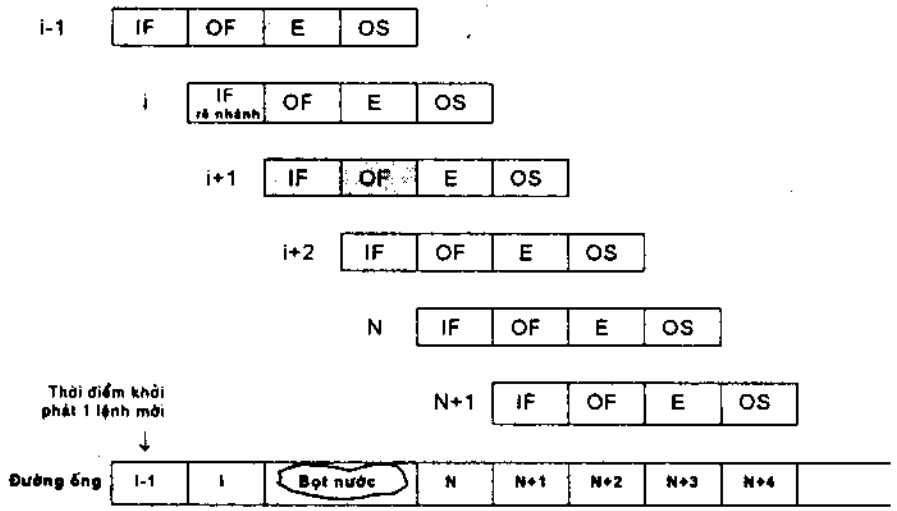
Hình 2.22 Cấu trúc của một chu trình máy.

Mỗi giai đoạn kể trên được thực hiện trong một khoảng thời gian nhất định (thường là một bội số nguyên lần của chu kì nhịp đồng hồ hệ thống). Một vài lệnh có thể đòi hỏi ít hơn 5 giai đoạn trên; thí dụ lệnh CMP R1,R2 so sánh R1 và R2 bằng cách trừ R1 cho R2 để đặt các mã điều kiện nên không cần giai đoạn lưu trữ toán hạng. Cách sắp xếp kể trên rõ ràng không hiệu quả. Hãy xem xét giai đoạn thực hiện lệnh. Giai đoạn này được diễn ra trong *đơn vị thực hiện lệnh*, nó chỉ chiếm mất một phần năm chu trình lệnh; nói cách khác *đơn vị thực hiện lệnh* trong CPU sẽ không làm gì cả trong 80% thời gian còn lại của chu trình lệnh !. Vì lí do này, một kĩ thuật gọi là *kĩ thuật đường ống* (pipelining) được áp dụng nhằm nâng cao tốc độ xử lí hiệu dụng lên bằng cách gối lên nhau về mặt thời gian các giai đoạn khác nhau trong việc thực hiện lệnh. Theo cách nói đơn giản, một xử lí đường ống có thể cho phép thực hiện lệnh thứ i trong khi nhận lệnh thứ i+1 trong cùng một thời gian. Hình 2.23 mô tả cách mà một bộ xử lí RISC thực hiện xử lí đường ống. RISC gối giai đoạn nhận lệnh từ bộ nhớ ngoài với 3 giai đoạn xảy ra bên trong chip: nhận toán hạng, thực hiện lệnh và lưu trữ toán hạng. Lưu ý rằng 3 giai đoạn này chỉ chiếm mất một thời gian xấp xỉ như giai đoạn nhận lệnh vì các hoạt động xảy ra ngay bên trong CPU và các toán hạng được nhận hoặc lưu trữ vào ngay các thanh ghi của CPU nên có tốc độ xử lí rất nhanh. Hơn nữa vì RISC có một định dạng lệnh thông thường rất đơn giản nên giai đoạn giải mã lệnh là không cần thiết.



Hình 2.23 Đường ống lệnh.

Tuy nhiên kĩ thuật đường ống gặp phải một trở ngại mà các nhà thiết kế hệ thống phải tính đến, đó là các "bọt nước" xuất hiện trên đường ống do các lệnh rẽ nhánh sinh ra và tính phụ thuộc của các dữ liệu. Một đường ống là một cấu trúc trật tự được phát triển trên các quy tắc. Ở một giai đoạn bất kì trong việc thực hiện một chương trình, một đường ống chứa 2 hoặc nhiều hơn các lệnh với các giai đoạn khác nhau. Hãy xem hình 2.24 trong đó một chuỗi lệnh đang được thực hiện trong xử lí đường ống. Khi bộ xử lí gặp một lệnh rẽ nhánh, nó buộc phải nạp lại nội dung *bộ đếm chương trình* tới một giá trị mới. Điều này có nghĩa là tất cả các công việc hữu ích đã được thực hiện bởi đường ống đến đây sẽ bị bỏ đi, vì lệnh tiếp ngay sau lệnh rẽ nhánh sẽ không được thực hiện. Khi dữ liệu trong đường ống bị đẩy ra ngoài hoặc đường ống bị treo vào trạng thái chạy không, chúng ta nói rằng một bọt nước đã xuất hiện trong đường ống và làm chậm lại quá trình xử lí. Dĩ nhiên, nếu đường ống càng dài thì càng có nhiều lệnh phải bị đẩy ra khỏi nó khi xuất hiện lệnh rẽ nhánh. Cấu trúc RISC của Berkeley khắc phục vấn đề bọt nước bằng cách không đẩy ngay các lệnh sau lệnh rẽ nhánh ra ngoài đường ống.

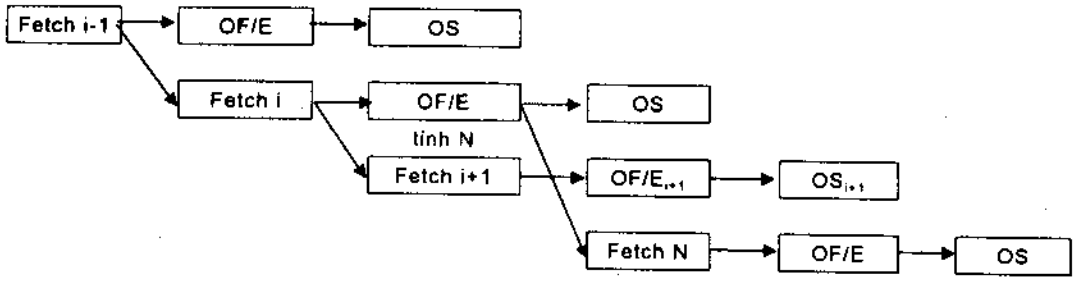


Hình 2.24 Tác động của lệnh rẽ nhánh trong đường ống.

Lệnh theo sau lệnh rẽ nhánh thì luôn được thực hiện Xét tác động của chuỗi lệnh sau:

ADD R1, R2, R3	[R3] ← [R1]+[R2]
JMPX N	[PC] ← [N] tức là nhảy tới địa chỉ N
ADD R3, R4, R5	[R5] ← [R3] +[R4]
ADD R7, R8, R9	Không được chạy vì có lệnh rẽ nhánh

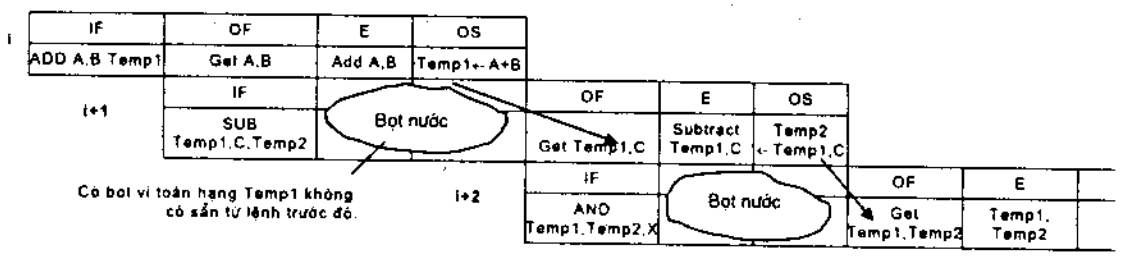
Bộ xử lí sẽ tính R5 = R1+R2+R4 trước khi chạy lệnh rẽ nhánh. Đây là một cách xử lí lạ với những người lập trình ngôn ngữ máy thông thường vì thường không bao giờ một lệnh sau lệnh rẽ nhánh lại được chạy cả. Cách xử lí này cũng thường không phải lúc nào cũng thực hiện được. Vì khi đó bộ biên dịch phải đưa vào một lệnh chạy không (NOP) sau lệnh rẽ nhánh và chấp nhận một điều không tránh khỏi là có một bọt nước. Các bộ xử lí RISC nói chung coi cơ chế này như một *bước nhảy trễ* (delayed jump) hoặc gọi là *kĩ thuật rẽ nhánh và thực hiện* (branch-and-execute technique). Hình 2.25 là minh hoạ cách mà bộ xử lí RISC thực hiện kĩ thuật nhảy trễ. Lưu ý ở đây địa chỉ đích của nhánh được tính trong giai đoạn thực hiện lệnh của chu trình máy.



Hình 2.25 Nhảy trễ.

Ảnh hưởng do tính phụ thuộc dữ liệu trong đường ống là: các chuỗi lệnh xác định chạy trong đường ống sẽ gặp rắc rối vì phép toán hiện tại đòi hỏi một kết quả từ phép toán trước đó và phép toán trước này lại đang chưa rời khỏi đường ống. Hình 2.26 minh hoạ sự phụ thuộc dữ liệu xảy ra như thế nào. Giả sử người lập trình muốn thực hiện phép tính X:=(A+B)AND(A+B-C). Gia thiết rằng A,B,C,X và 2 giá trị tạm thời Temp1 và Temp2 nằm ở các thanh ghi trong cửa sổ hiện tại. Chúng ta có thể viết:

```
ADD  A,B,Temp1          [Temp1] ← [A]+[B]
SUB  Temp1,C,Temp2      [Temp2] ← [Temp1]-[C]
AND  Temp1,Temp2,X          [X] ← [Temp1].[Temp2]
```



Hình 2.26 Ảnh hưởng của sự phụ thuộc dữ liệu.

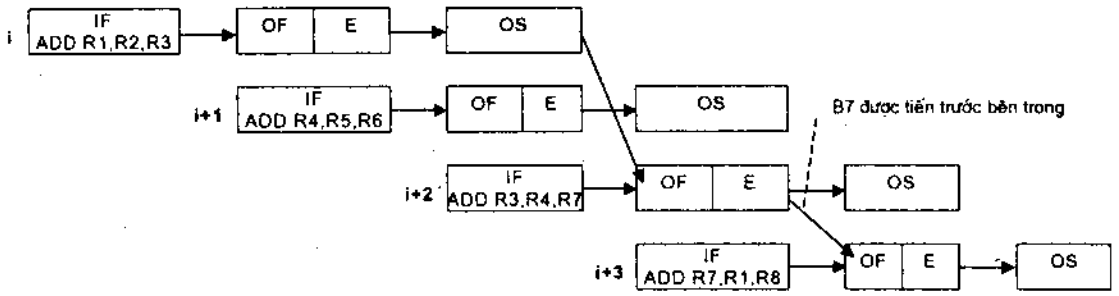
Lệnh thứ i+1 trong hình 2.26 bắt đầu chạy trong thời gian xảy ra giai đoạn nhận toán hạng của lệnh trước đó. Tuy nhiên, lệnh i+1 không có thể chạy tới giai đoạn tiếp sau - là pha nhận toán hạng - vì toán hạng thực sự mà nó đòi hỏi không nhận viết ngược trở lại tới file thanh ghi cho 2 chu kì đồng hồ khác. Hậu quả là một bọt nước cần phải được đưa vào đường ống trong lúc lệnh n+1 đợi số liệu cho nó. Tương tự, phép toán AND cũng bị đưa vào một bọt nước vì nó cũng đòi hỏi kết quả của phép toán trước đó mà phép toán đó lại đang còn ở trong đường ống.

Hình 2.27 mô tả một kĩ thuật gọi là *internal forwarding* được thiết kế để khắc phục ảnh hưởng của tính phụ thuộc dữ liệu. Thí dụ phù hợp với một đường ống 3 công đoạn như trong RISC. Chuỗi các hoạt động sau được thực hiện:

```
1.   ADD  R1,R2,R3      [R3] ← [R1]+[R2]
2.   ADD  R4,R5,R6      [R6] ← [R4]+[R5]
3.   ADD  R3,R4,R7      [R7] ← [R3]+[R4]
4.   ADD  R7,R1,R8      [R8] ← [R1]+[R7]
```

Trong thí dụ này, lệnh 3 sử dụng toán hạng được phát ra bởi lệnh 1 (tức là nội dung của thanh ghi R3). Tuy vậy, vì có sự xen vào lệnh 2, toán hạng đích được phát bởi lệnh 1 có thời gian được viết vào file thanh ghi trước khi nó được đọc như là toán hạng nguồn bởi lệnh 3. Lệnh 3 phát một toán hạng đích R7, toán hạng này được đòi hỏi như là toán hạng nguồn bởi

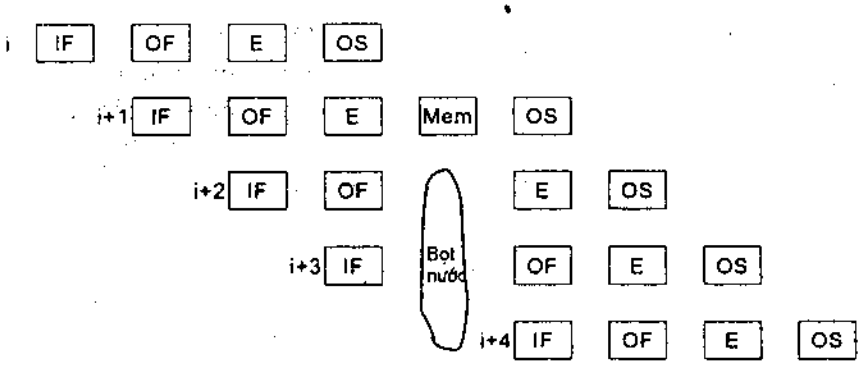
lệnh tiếp theo. Nếu bộ xử lí đọc toán hạng nguồn được yêu cầu bởi lệnh 4 từ file thanh ghi, nó sẽ tìm giá trị cũ của R7. Bằng kĩ thuật này, bộ xử lí sẽ truyền trực tiếp R7 từ đơn vị thực hiện lệnh 3 tới đơn vị thực hiện lệnh 4.



Hình 2.27 Kĩ thuật internal forwarding.

2.3.6. Thâm nhập bộ nhớ bên ngoài vi xử lí

Ta biết bộ xử lí CISC thông thường có nhiều cách định địa chỉ được dùng chung với nhiều lệnh quy chiếu bộ nhớ. Thí dụ lệnh ADD D0,-(A5) thực hiện việc cộng nội dung của D0 với đỉnh ngăn xếp được trỏ bởi A5 rồi đẩy kết quả vào ngăn xếp này. Kiến trúc RISC Berkeley chỉ cho phép 2 loại lệnh quy chiếu bộ nhớ bên ngoài là lệnh nạp (load) và lưu trữ (store). Kết quả là tất cả các phép toán số học và lôgic được thực hiện bởi RISC chỉ tới các toán hạng nguồn và đích trong các thanh ghi. Tương tự, kiến trúc này chỉ cung cấp một số hạn chế các chế độ định địa chỉ để thâm nhập toán hạng trong bộ nhớ chính. ta thử thảo luận một vài nguyên tắc chung của các lệnh nạp và lưu trữ trong môi trường RISC. Phép toán thanh ghi nạp dạng LOAD (Rx)S2,Rd có tác động [Rd] ← [M([Rx]+S2)]. Hình 2.28 mô tả chuỗi các hoạt động thực hiện lệnh này.



Hình 2.28 Chạy một lệnh nạp LOAD. Việc quy chiếu bộ nhớ ngoài trong quá trình chạy lệnh đã gây ra một bọt nước trong đường ống.

Trong giai đoạn nhận lệnh nguồn, thanh ghi Rx, được sử dụng để tính địa chỉ hiệu dụng, sẽ được đọc từ file thanh ghi và được dùng để tính địa chỉ hiệu dụng của toán hạng trong giai đoạn thực hiện lệnh. Tuy vậy, khi tiến hành giai đoạn thực hiện lệnh ta không thể chuyển tới giai đoạn lưu trữ toán hạng vì toán hạng chưa được đọc từ bộ nhớ chính. Do đó bộ nhớ chính phải được thâm nhập để đọc toán hạng và giai đoạn lưu trữ toán hạng phải được thực hiện để nạp toán hạng vào thanh ghi đích Rd.

CÂU HỎI VÀ BÀI TẬP

2. 1 Hãy tra tập mã lệnh 8086 ở phần phụ lục để viết ra các mã của các lệnh vi mô sau, mỗi lệnh chiếm bao nhiêu nhịp đồng hồ ? Thời gian để thực hiện chúng ?

MOV AL,56 ; MOV DX,0300; OUT DX,AL
IN 80; POP; PUSH AX
AND AX,3000; TEST AX, 2000; RET;

2. 2 Chương trình lớn nhất có thể được địa chỉ bởi các bộ xử lí với bề rộng bit địa chỉ như sau là bao nhiêu ? Chương trình lớn nhất cũng cùng nghĩa với không gian địa chỉ và được đo bằng byte.

a) 12 bit b) 16 bit
c) 24 bit d) 32 bite e) 48 bit

2. 3 Có một hệ thống vi xử lí của hãng Intel 8086 dùng cho xử lí văn bản. Giả thiết rằng hệ điều hành, các trình xử lí văn bản (word processor) và không gian đệm cho các kí tự. nằm trong vùng 1MB. Phần nhớ còn lại là dùng cho việc đọc/viết văn bản. Nếu trung bình một từ tiếng Anh gồm 5 kí tự ASCII và một trang văn bản gồm 35 dòng, mỗi dòng 12 từ thì trong bộ nhớ mở rộng 8 MB có thể lưu trữ được bao nhiêu trang văn bản ?.

2.4 Giả sử bộ vi xử lí được dùng để xử lí một hình ảnh số (thí dụ trong máy in Laser). Hình ảnh đó gồm n x n điểm ảnh (pixel). Mỗi điểm ảnh có thể nhận 1 trong 16 mức xám (từ đen đến trắng). Nếu kích thước bức ảnh là 20 x 20 cm, hãy tính độ phân giải của bức ảnh (số pixel / cm).

2.5 Ta biết máy tính Von Neumann lưu trữ chương trình và số liệu trên cùng một bộ nhớ. Theo bạn một hệ thống khác lưu trữ chương trình và số liệu ở 2 bộ nhớ tách biệt thì có những ưu, nhược điểm gì ?

2.6 Giải thích ý nghĩa của các hoạt động theo ngôn ngữ RTL sau:

a) [D3] ← 4
b) [D3] ← [MS(4)]
c) [MS(4)] ← [MS(2)]
d) [MS(5)] ← [MS([MS(2)])]
e) [D4] ← [MS([D2])]
f) [D1] ← [MS(4 + [D2])]

2.7 Giải thích chức năng của các thanh ghi sau trong CPU:

a) PC hay IP
b) MAR
c) MBR
d) IR

2.8 Giải thích ý nghĩa của chu trình nhận lệnh ("fetch cycle") và mô tả hoạt động của nó theo ngôn ngữ RTL.

2.9 Một vài vi xử lí có 1 hay 2 thanh ghi tích luỹ. Bạn có nghĩ rằng có thể một máy tính phải xác định được số lí tưởng các thanh ghi đa năng cho nó không? Tại sao ?

2.10 Vi lệnh là gì ?

2.11 Kiến trúc RISC có những gì để phân biệt nó với kiến trúc CISC ? Xử lí đường ống cho phép người thiết kế máy tính tăng tốc độ hiệu dụng của bộ xử lí bằng cách gối chồng các hoạt động lên nhau. Những vấn đề nào nảy sinh ra khi thực hiện cơ chế này ?

CHƯƠNG 3. HỆ THỐNG MÁY VI TÍNH VỚI HỌ VI XỬ LÍ 80×86, HOẠT ĐỘNG ĐA NHIỆM

Dưới con mắt của người lập trình, các máy vi tính tương thích IBM/PC gồm một vi xử lí, các chip nhớ và một số chip bổ trợ. Trong hầu hết các máy vi tính, tất cả linh kiện, mạch điện chính được lắp đặt trên bản mạch chính; các phần quan trọng khác được lắp đặt trên các bản mạch ghép nối cắm trên các khe cắm mở rộng.

3.1. HỆ THỐNG MÁY VI TÍNH 8086 CƠ BẢN

3.1.1. Giới thiệu

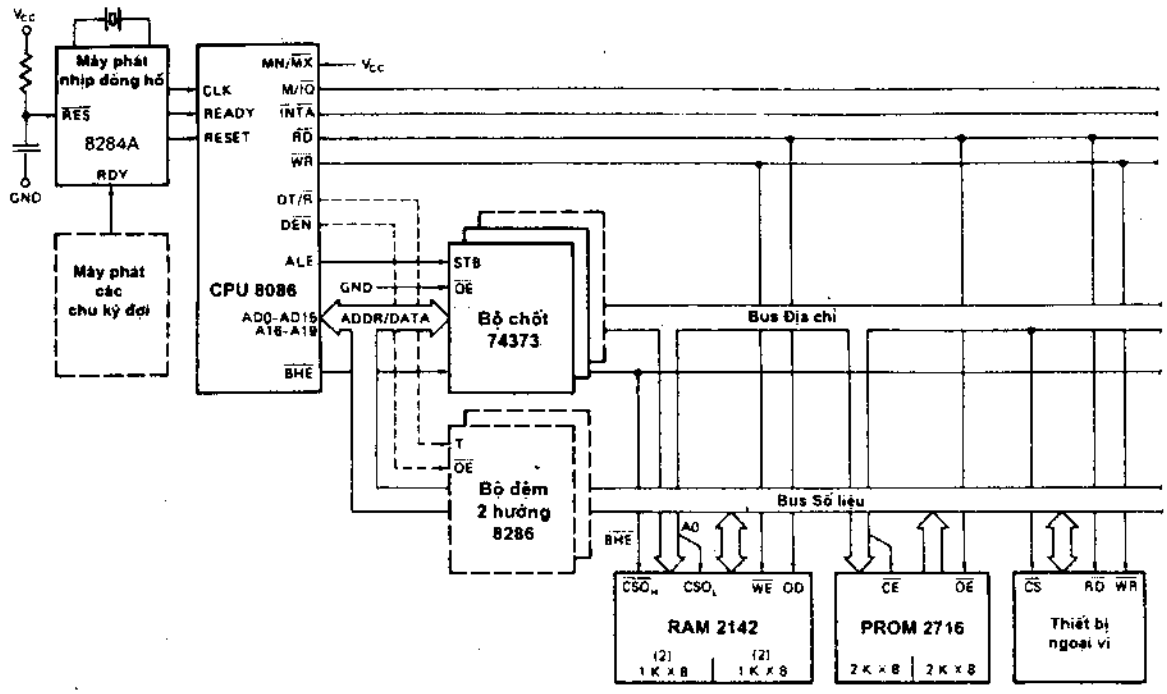
Hình 3.1 là một trong các sơ đồ khối đơn giản của máy vi tính IBM/PC được phát triển trên cơ sở chip vi xử lí 8086. Trước tiên, ta thấy có CPU 8086, các chip nhớ ROM và RAM. Sau đó là các thiết bị ngoại vi được nối với các cổng vào/ra. Ở phần sau, có rất nhiều thiết bị được nối với các cổng, thí dụ như các chip bổ trợ cho truyền thông tin song song, chip UART-8250 cho truyền nối tiếp, đặc biệt là các thiết bị video ghép nối với màn hình, các chip điều khiển bàn phím, ổ đĩa v.v...

Trong hình còn thấy bus dữ liệu, bus điều khiển, bus địa chỉ. Bus điều khiển cơ bản gồm các dây tín hiệu có tên *M/* $\overline{IO}$, $\overline{RD}$ và $\overline{WR}$. Nếu 8086 đọc từ bộ nhớ hay cổng, tín hiệu $\overline{RD}$ sẽ được kích hoạt. Nếu 8086 viết vào bộ nhớ hay cổng thì $\overline{WR}$ sẽ được kích hoạt. Trong khi đang đọc/viết bộ nhớ thì tín hiệu M/ $\overline{IO}$ ở mức cao và trong khi đọc/viết với cổng thì M/ $\overline{IO}$ sẽ ở mức thấp. Các tín hiệu M/ $\overline{IO}$, $\overline{RD}$, $\overline{WR}$ dùng để kích hoạt các thiết bị được định địa chỉ. 8086 có một bus đơn có tên *ADDR/DATA* - bus địa chỉ/dữ liệu được nối với các bộ chốt và các bộ đệm ở một phía. Phía bên kia, bus này được tách ra thành hai bus địa chỉ và bus dữ liệu riêng rẽ. Lí do là vì để tiết kiệm chân, 16 bit thấp của địa chỉ được hợp kênh bên trong chip và đi nhờ trên bus dữ liệu.

Bước đầu tiên trong bất cứ thao tác nào, 8086 sẽ phát ra 16 bit thấp của tín hiệu định địa chỉ xác định đơn vị mà 8086 muốn truy xuất lên bus địa chỉ/dữ liệu. Bộ chốt ngoài, như bộ chốt tám bit 74LS373 trên hình, sẽ cất giữ địa chỉ này trong suốt quá trình hoạt động còn lại. Để báo chính xác thời điểm cho thanh chốt này hoạt động, 8086 gửi đi tín hiệu chốt địa chỉ ALE (Address Latch Enable). Bởi vì địa chỉ được lưu giữ trong đầu ra của thanh chốt nên sau đó 8086 sẽ xoá địa chỉ khỏi bus địa chỉ/dữ liệu và dùng bus cho việc đọc và viết dữ liệu. Khối tiếp theo là các bộ đệm ba trạng thái hai chiều. Với những hệ thống nhỏ thì bộ đệm không cần nhưng khi có thêm nhiều thiết bị nối vào hệ thống thì bộ đệm lại rất cần thiết.

Nguyên do là vì các điện dung lối vào của các thiết bị (cỡ vài pF) được nối tới bus lối ra của CPU. Khi số thiết bị lớn thì điện dung tổng sẽ lớn dẫn tới việc mạch lối ra của 8086 sẽ không cung cấp đủ dòng điện để điều khiển việc nạp và phóng tụ điện đủ nhanh và như vậy có nghĩa rằng tốc độ thay đổi trạng thái của bus không đủ nhanh. Chính các bộ đệm thêm vào sẽ cho phép cung cấp dòng đủ lớn để khắc phục điều đó.

Các bộ đệm trên bus dữ liệu phải là loại hai chiều bởi vì 8086 vừa phát, vừa thu dữ liệu. Tín hiệu thu/phát dữ liệu DT/$\overline{R}$ từ 8086 sẽ quy định hướng mà dữ liệu sẽ truyền qua bộ đệm. Khi tín hiệu DT/$\overline{R}$ được đặt ở mức cao thì bộ đệm sẽ khởi động để truyền dữ liệu từ 8086 đến ROM, RAM, hay cổng vào/ra.

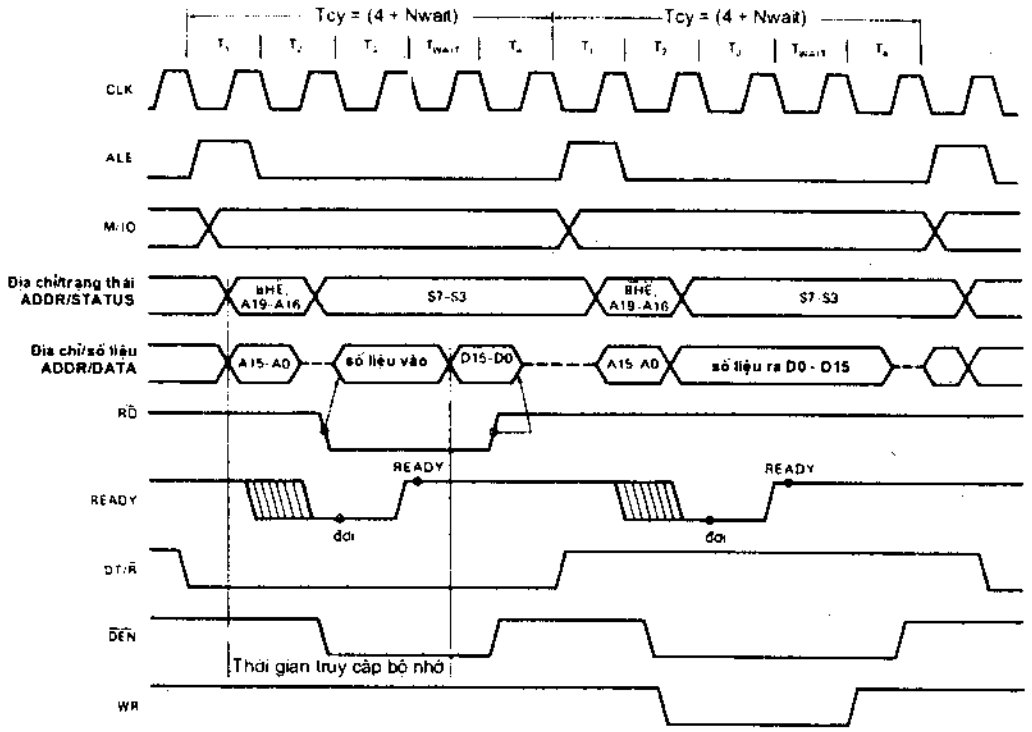


Hình 3.1 Một hệ thống máy vi tính dùng 8086.

Khi tín hiệu DT/$\overline{R}$ được đặt ở mức thấp thì bộ đệm khởi động để cho phép truyền dữ liệu từ ROM, RAM hay cổng vào CPU-8086. Bộ đệm sử dụng trên bus dữ liệu thì phải có lối ra ba trạng thái, vì vậy lối ra có thể thả nổi (treo) khi bus đang được dùng cho các hoạt động khác. Ví dụ, nếu không muốn đầu ra bộ đệm tác động vào bus dữ liệu khi mà 8086 đang truyền đi 16 bit thấp của địa chỉ trên bus này thì 8086 sẽ đặt dây tín hiệu $\overline{DEN}$ xuống mức thấp để kích hoạt lối ra ba trạng thái trên bộ đệm tại thời điểm chính xác của bất cứ một hoạt động nào. Phần cuối cùng trong hình là bộ phát xung đồng hồ 8284A nằm ở góc trái trên cùng. Thiết bị này sử dụng một tinh thể để tạo ra xung đồng hồ có tần số ổn định mà xung đồng hồ này sẽ từng nhịp kích hoạt 8086 thực hiện các câu lệnh một cách tuần tự. 8284A cũng dùng để đồng bộ tín hiệu RESET và tín hiệu READY với xung đồng hồ. Khi đầu vào RESET được kích hoạt, 8086 sẽ nhảy đến địa chỉ FFFF0h để nhận lệnh tiếp theo. Lệnh đầu tiên của chương trình khởi động hệ thống thường được đặt ở tại địa chỉ này, vì vậy việc kích hoạt tín hiệu này là một cách để khởi động hệ thống. Ta sẽ thảo luận cách sử dụng tín hiệu READY trong phần sau. Bây giờ, ta sẽ xem xét các tín hiệu có trên bus khi 8086 đọc dữ liệu từ cổng hay ô nhớ.

3.1.2. Các hoạt động của bus 8086 trong một chu kì đọc

Hình 3.2 là giản đồ xung chỉ ra hoạt động của các tín hiệu trên bus của máy vi tính 8086 trong một quá trình đọc và viết đơn giản. Dạng sóng đầu tiên trong hình là dạng xung đồng hồ. Một chu kỳ của xung đồng hồ này được gọi là một *trạng thái*. Một trạng thái được tính từ sườn âm của một xung đồng hồ đến sườn âm của xung tiếp theo (ví dụ khoảng thời gian T_1 trong hình). Các phiên bản khác nhau của 8086 có tần số đồng hồ tối đa từ 5 đến 10 MHz, vì vậy thời gian nhỏ nhất của một trạng thái là từ 100 đến 200 ns. Một quá trình cơ bản của bộ vi xử lí thực hiện đọc một byte từ bộ nhớ hay viết một byte vào cổng được gọi là một *chu kì máy*, hay chu kỳ bus (ví dụ khoảng thời gian T_{cycle} trong hình). Một chu kì máy gồm nhiều trạng thái.



Hình 3.2 Giản đồ xung của các chu kì đọc và viết.

Thời gian mà vi xử lí cần để nhận lệnh và thi hành một lệnh gọi là một *chu kì lệnh*. Một chu kì lệnh gồm một hay nhiều chu kì máy.

Tóm lại, các trạng thái tạo nên một chu kì máy và các chu kì máy tạo nên một chu kì lệnh. Xét các hoạt động xuất hiện trên bus 8086 trong một chu kì đọc của máy tính.

Trong trạng thái T_1 của chu kì đọc, 8086 trước tiên thiết lập trạng thái cho tín hiệu $M/\overline{IO}$. Nếu 8086 đọc từ bộ nhớ thì $M/\overline{IO}$ ở mức cao và nếu 8086 đọc từ cổng vào/ra trong chu kì này thì $M/\overline{IO}$ sẽ ở mức thấp. Trong hình 3.2 vẽ hai dạng sóng của tín hiệu $M/\overline{IO}$ được cắt chéo nhau bởi vì tín hiệu này có thể ở mức thấp hay có thể ở mức cao trong một chu kì đọc. Điểm mà hai sóng giao nhau chỉ ra thời gian tại đó tín hiệu trở nên ổn định trong chu kì đọc này. Cũng như thế, trong phần còn lại của sơ đồ, các đường cắt nhau thể hiện cho thời điểm mà thông tin trên một đường hay một nhóm đường bị thay đổi.

Sau khi thiết lập tín hiệu $M/\overline{IO}$, 8086 gửi đi tín hiệu ALE ở mức cao. Tín hiệu này nối với đầu vào STB của bộ chốt tám bit 74LS373 (hình 3.1) để kích hoạt bộ chốt này. Như trên hình 3.1, đầu vào dữ liệu của bộ chốt được nối với AD0-AD15, AD16 - AD19 của 8086 và tín hiệu $\overline{BHE}$ (Bus High Enable). Sau khi 8086 kích hoạt ALE ở mức cao, nó gửi địa chỉ của ô nhớ mà 8086 muốn đọc trên bus ADR/DATA. Vì rằng ALE ở mức cao nên bộ chốt thông và thông tin về địa chỉ đi qua bộ chốt tới đầu ra của nó. Sau đó 8086 hạ tín hiệu ALE xuống mức thấp và cấm bộ chốt. Địa chỉ từ lối ra bộ chốt sẽ theo bus địa chỉ tới bộ nhớ và các thiết bị vào/ra.

Điểm đầu tiên mà hai sóng cắt nhau chính là thời điểm 8086 vừa đặt tín hiệu địa chỉ ổn định lên bus ADDR/DATA. Sau khi ALE xuống thấp, thông tin về địa chỉ được lưu trong bộ chốt vì vậy 8086 không cần phải phát địa chỉ đi nữa. Do đó, 8086 thả nổi các đường AD0-AD15 và sử dụng chúng cho truyền dữ liệu đọc được từ bộ nhớ hay cổng (trên hình 3.1 là các đường

đứt nét trên bus ADDR/DATA). Cũng vào thời gian đó 8086 xoá tín hiệu $\overline{BHE}$, xoá thông tin trên A16-A19 và gửi đi thông tin về trạng thái trên các đường này.

8086 bây giờ đã sẵn sàng cho đọc dữ liệu từ ô nhớ và cổng, nó thiết lập tín hiệu $\overline{RD}$ ở mức thấp tại sát cuối trạng thái T2. Trên hình 3.1 tín hiệu $\overline{RD}$ dùng để cho phép bộ nhớ hoặc thiết bị vào/ra. Khi các thiết bị này được kích hoạt thì chúng sẽ đặt một byte hay một từ lên bus dữ liệu. Mối quan hệ nhân quả được chỉ ra trên giản đồ xung hình 3.2 bằng mũi tên từ sườn âm của tín hiệu $\overline{RD}$ tới thời khoảng mà bus được sử dụng cho việc đọc dữ liệu vào CPU (reserved data in) của dạng xung trên bus ADDR/DATA. Gốc mũi tên luôn đặt ở vị trí hay ở sườn tín hiệu mà tại đó bắt đầu xảy ra một hoạt động nào đó. Đầu mũi tên trong trường hợp này chỉ dùng để thể hiện cho kết quả tác động của một tín hiệu từ thiết bị này lên thiết bị khác. Chúng không dùng để chỉ thị tín hiệu nguyên nhân và kết quả trong cùng một thiết bị.

Trên hình 3.2 còn có phần thời gian truy xuất bộ nhớ (memory asscess time) nằm cuối sơ đồ. Thời gian truy cập là khoảng thời gian mà bộ nhớ cần để xuất ra dữ liệu hợp lệ (ổn định) sau khi nhận được tín hiệu định địa chỉ và tín hiệu $\overline{RD}$. Nếu thời gian truy xuất bộ nhớ quá dài, lối ra của bộ nhớ sẽ không xuất được đủ nhanh dữ liệu hợp lệ trong một chu kì máy để 8086 nhận một cách chính xác. Trong trường hợp đó, 8086 sẽ xử lí tất cả dữ liệu xuất hiện trên bus rồi tiếp tục chạy chu kì máy tiếp theo. Các dữ liệu này sẽ gồm cả dữ liệu không hợp lệ và sẽ làm cho toàn bộ chương trình bị đổ vỡ. Vậy các bộ nhớ ROM, RAM và thiết bị vào/ra cần có thời gian truy xuất đủ nhỏ để làm việc chính xác trong hệ thống 8086. Trong trường hợp bộ nhớ và các thiết bị vào/ra có thời gian truy xuất lớn thì CPU phải xen vào một vài *trạng thái đợi* (wait state) để hệ thống làm việc ổn định. Xem lại sơ đồ khối hình 3.1, ta thấy có một đầu vào của CPU 8086 tên là READY. Khi chân này ở mức cao, 8086 hoạt động một cách bình thường như nói trên. Nếu READY được đặt xuống mức thấp tại một thời điểm trong một chu kì máy, 8086 sẽ chèn thêm một hay vài trạng thái đợi. Thí dụ như giữa T3 và T4 (trong hình 3.2). Một thiết bị phần cứng bên ngoài sẽ đặt đầu vào READY xuống thấp trước sườn dương của xung đồng hồ T2, sau khi 8086 hoàn thành chu kì máy T3, 8086 sẽ chèn vào một trạng thái đợi. Trong trạng thái đợi, tín hiệu trên bus vẫn giữ nguyên như khi bắt đầu trạng thái đợi. Địa chỉ của ô nhớ được lưu giữ trên đầu ra của bộ chốt, tín hiệu trên bus điều khiển, $M/\overline{IO}$ và $\overline{RD}$ cũng không thay đổi trong trạng thái đợi T_{WAIT} (như trên giản đồ hình 3.2). Bộ nhớ hay thiết bị vào/ra có ít nhất một chu kì đồng hồ để xuất dữ liệu ra. Nếu lối vào READY ở mức cao trong chu kì T3 hoặc trong trạng thái đợi thì sau một trạng thái đợi 8086 sẽ tiếp tục với chu kì T4 như bình thường.

Nếu tín hiệu READY của 8086 vẫn ở mức thấp tại cuối trạng thái đợi thì 8086 sẽ chèn tiếp trạng thái đợi khác cho đến khi tín hiệu READY được đặt ở mức cao.

Tóm lại, việc chèn các trạng thái đợi sẽ làm ổn định hoạt động trên bus. Việc chèn này trao thêm cho thiết bị được gọi một số chu kì đồng hồ để có thể xuất ra dữ liệu hợp lệ. Như vậy ta có thể sử dụng ROM chạy chậm hơn trong hệ thống bằng cách thêm một mạch đơn giản có thể tạo ra xung ở lối vào READY thấp mỗi khi ROM được định địa chỉ. Tất nhiên, sẽ không có trạng thái đợi chèn vào chu kì đọc khi đọc dữ liệu từ các thiết bị chạy nhanh hơn.

Chú ý trong hình 3.1, tín hiệu READY được truyền từ vi mạch phát xung đồng hồ 8284A vào để tín hiệu READY thực sự gắn vào 8086 và đồng bộ với đồng hồ hệ thống.

Xem lại hình 3.2 để tìm hiểu chức năng của tín hiệu $\overline{DEN}$ và $DT/\overline{R}$ trong một chu kì đọc. Trong chu kì máy T1, 8086 xác lập $DT/\overline{R}$ xuống mức thấp để đặt bộ đệm dữ liệu trong chế độ thu. Sau đó, khi 8086 đã kết thúc việc sử dụng bus dữ liệu cho việc gửi 16 bit địa chỉ thấp, 8086

sẽ xác lập tín hiệu $\overline{DEN}$ xuống thấp để kích hoạt bộ đệm bus dữ liệu. Cổng hay bộ nhớ đặt dữ liệu lên bus dữ liệu, dữ liệu này qua bộ đệm vào 8086.

Các hoạt động trên bus 8086 trong một chu kì đọc có thể tóm tắt như sau: CPU đặt tín hiệu $M/\overline{IO}$ lên mức cao nếu đọc từ bộ nhớ và đặt $M/\overline{IO}$ xuống mức thấp nếu đọc từ cổng. Cùng lúc đó, 8086 đặt ALE lên mức cao để kích hoạt bộ chốt địa chỉ. Sau đó nó gửi đi tín hiệu $\overline{BHE}$ và địa chỉ mong muốn trên các đường dây AD0-A19. Tiếp đó 8086 đặt ALE xuống thấp. thông tin về địa chỉ được chốt ở đầu ra của bộ chốt. Sau khi 8086 sử dụng các đường dây AD0-AD15 cho việc truyền địa chỉ thì nó sẽ xoá các địa chỉ trên các đường này và đặt các đường này trong chế độ nhận dữ liệu (thả nổi chúng). 8086 tiếp theo sẽ đặt tín hiệu $\overline{RD}$ xuống thấp. Tín hiệu $\overline{RD}$ xuống thấp sẽ kích hoạt cổng hay bộ nhớ gửi ra dữ liệu mong muốn trên bus dữ liệu. Để kết thúc chu kì, 8086 lại đặt $\overline{RD}$ lên cao. Điều này làm cho bộ nhớ hay thiết bị vào/ra đặt các đầu ra của nó ở trạng thái treo trên bus dữ liệu. Nếu READY xuống thấp trước hay trong chu kì máy T2, thì 8086 sẽ liên tục chèn thêm các trạng thái đợi chừng nào mà tín hiệu READY còn ở mức thấp. Khi tín hiệu READY lên mức cao, 8086 sẽ tiếp tục với chu kì T4. Các trạng thái đợi được dùng để thêm thời gian cho các thiết bị chạy chậm có thể xuất ra dữ liệu đúng. Nếu hệ thống lớn cần thêm bộ đệm bus dữ liệu, tín hiệu $DT/\overline{R}$ nối với bộ đệm này sẽ kích hoạt chúng cho việc nhập dữ liệu trong quá trình đọc hay kích hoạt chúng cho xuất dữ liệu trong quá trình viết. Tín hiệu $\overline{DEN}$ sẽ kích hoạt bộ đệm tại các khoảng thời gian chính xác trong chu kì máy.

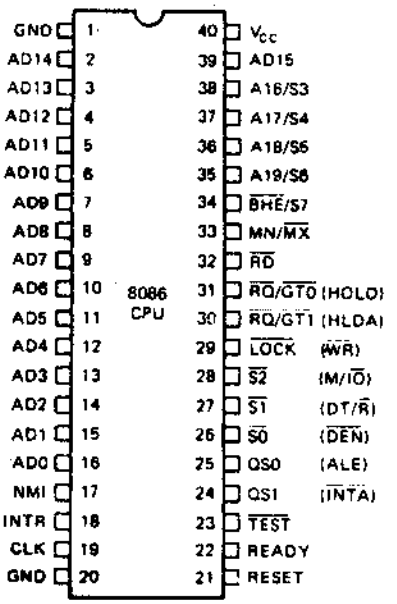
3.1.3. Các hoạt động của bus 8086 trong một chu kì viết

Bây giờ ta xem xét giản đồ của một chu kì viết trong nửa phải của hình 3.2. Giản đồ này gần giống như ở chu kì đọc. Trong chu kì viết T1, 8086 đặt tín hiệu $M/\overline{IO}$ xuống thấp nếu viết vào cổng và đặt $M/\overline{IO}$ lên cao nếu viết vào bộ nhớ. Cùng lúc đó, 8086 đặt ALE lên mức cao để kích hoạt bộ chốt địa chỉ. Sau đó nó gửi đi tín hiệu $\overline{BHE}$ và địa chỉ cần được viết trên các đường dây AD0-A19. Khi viết vào cổng, các đường dây A16-A19 sẽ luôn ở mức thấp bởi vì 8086 chỉ gửi đi địa chỉ cổng 16-bit. Sau khi địa chỉ đã truyền qua bộ chốt, 8086 sẽ đặt ALE xuống thấp để chốt địa chỉ trên đầu ra. Bên cạnh việc giữ địa chỉ, các thanh chốt cũng có chức năng như bộ đệm cho các đường địa chỉ. Khi thông tin về địa chỉ được chốt lại, 8086 xoá thông tin địa chỉ này trên các đường dây AD0 ÷ AD15 và gửi dữ liệu cần viết lên bus dữ liệu. 8086 lại đặt $\overline{WR}$ xuống thấp. Tín hiệu $\overline{WR}$ dùng để kích hoạt bộ nhớ hay cổng mà dữ liệu sẽ viết vào. Sau khi bộ nhớ hay cổng nhận dữ liệu từ bus, 8086 sẽ lại đặt tín hiệu lên cao và thả nổi bus dữ liệu.

Nếu bộ nhớ hay thiết bị vào/ra không nhận kịp một từ dữ liệu trong một chu kì máy thì phần cứng bên ngoài có thể được dùng để tạo ra một xung READY mức thấp mỗi khi mà bộ nhớ hay hay thiết bị vào/ra được định địa chỉ. Nếu lối vào READY được đặt ở mức thấp trước hay trong chu kì máy T2, 8086 sẽ chèn các trạng thái đợi sau trạng thái T3. Chú ý là trong trạng thái đợi thì tín hiệu trên bus dữ liệu, bus địa chỉ, bus điều khiển đều không đổi, vì thế bộ nhớ hay thiết bị vào/ra có thêm một hay nhiều chu kì xung nhịp để nhận dữ liệu từ bus dữ liệu. Nếu lối vào READY được đặt lên mức cao trước khi kết thúc chu kì đợi thì 8086 sẽ tiếp tục với chu kì T4 ngay khi nó kết thúc trạng thái đợi. Nếu lối vào READY vẫn ở mức thấp trước khi kết thúc trạng thái đợi, 8086 sẽ chèn tiếp các trạng thái đợi khác cho đến khi lối vào READY lên cao. 8086 có thể chèn bao nhiêu chu kì đợi cần thiết cho thiết bị nhận dữ liệu cũng được.

Nếu hệ thống cần bộ đệm trên bus dữ liệu, tín hiệu $DT/\overline{R}$ sẽ được nối với chiều vào của bộ đệm. Trong một chu kì viết, 8086 nâng $DT/\overline{R}$ lên mức cao để đặt bộ đệm trong chế độ phát. Khi 8086 đặt $\overline{DEN}$ xuống mức thấp để kích hoạt bộ đệm thì dữ liệu từ 8086 sẽ qua bộ đệm tới cổng hay ô nhớ.

Hình 3.3 là sơ đồ các chân tín hiệu của vi xử lí 8086. Dưới đây là một số mô tả về các chân để ta có thể hình dung rõ hơn về chức năng của máy vi tính 8086.



Hình 3.3 Chip vi xử lí 8086.

Chân 40 là V_{cc} được cấp nguồn +5V, chân 1 và chân 20 nối đất. Lối vào xung đồng hồ tại chân 19 có tên là CLK. Như đã chỉ ra ở phần trước, 8086 cần một tín hiệu định thời từ máy phát xung đồng hồ bên ngoài để đồng bộ hoạt động bên trong bộ xử lí. Bây giờ ta xem xét các đường bus địa chỉ/dữ liệu AD0-AD15. Nhớ lại trong phần trước, 8086 có bus địa chỉ 20 bit và bus dữ liệu 16 bit, 8086 dùng 16 đường địa chỉ thấp chung với 16 đường của bus dữ liệu để giảm bớt số chân. 8086 gửi đi tín hiệu ALE (Address Latch Enable) trên chân 25 để kích hoạt bộ chốt địa chỉ. Bốn bit cao của địa chỉ được gửi đi trên các đường dây có tên A16/S3 đến A19/S6. Bit địa chỉ A16-A19 được gửi đi trên các đường dây S3-S6 trong phần đầu của chu kì máy. Thông tin về trạng thái giúp nhận dạng loại thao tác nào sẽ phải thực hiện trong chu kì sẽ được gửi đi trên các đường dây S3-S6 này trong phần sau của chu kì. Xét các chân tín hiệu của bus điều khiển, chân 32 trên hình có tên $\overline{RD}$. Tín hiệu $\overline{RD}$ này sẽ được đặt xuống thấp khi 8086 đang đọc dữ liệu từ bộ nhớ hay từ cổng. Chân 29 có tên $\overline{WR}$ và tên $\overline{CLOCK}$ bên cạnh bởi vì chân này có hai chức năng. Chức năng của chân 29 và các chân khác (từ 24 đến 31) phụ thuộc vào chế độ mà 8086 đang chạy.

Chế độ hoạt động của 8086 được xác định bởi mức lôgic tại chân 33 MN/ $\overline{MX}$. Nếu chân 33 là cao thì 8086 sẽ có chạy trong *chế độ Min* (minimum) và các chân từ 24 đến 31 có chức năng như viết trong ngoặc cạnh các chân trong hình. Thí dụ trong chế độ Min, chân 29 sẽ có chức năng là $\overline{WR}$ mà sẽ ở mức thấp bất cứ khi nào 8086 viết vào cổng hay ô nhớ. Các tín hiệu $\overline{RD}$, $\overline{WR}$ và $M/\overline{IO}$ là trung tâm của bus điều khiển trong chế độ Min. Nếu chân $MN/\overline{MX}$ đặt xuống mức thấp thì 8086 sẽ chạy trong chế độ Max.(maximum). Trong chế độ này, chân 24 đến 31 sẽ có chức năng như các tên bên cạnh chân không ở trong ngoặc. Tín hiệu bus điều khiển ($\overline{S0}$, $\overline{S1}$, $\overline{S2}$) được gửi dưới dạng mã hoá trên chân 26, 27 và 28. Một thiết bị điều khiển bus bên ngoài sẽ giải mã các tín hiệu mã hoá này để tạo ra tín hiệu điều khiển theo quy định cho hệ thống có hai hay nhiều vi xử lí cùng dùng chung bus. Một chân quan trọng khác là chân 21-RESET. Nếu lối vào chân này được kích hoạt và sau đó thả ra thì 8086 không kể là đang làm gì sẽ nhận lệnh tiếp theo từ địa chỉ vật lí FFFF0h. Tại địa chỉ FFFF0h này sẽ để lệnh đầu tiên muốn vi xử lí thực hiện sau khi reset hay ngay sau khi bật nguồn.

Cuối cùng, 8086 còn có hai lối vào ngắt là NMI (nonmaskable interrupt) chân 17 và INTR (Interrupt) chân 18. Tín hiệu có thể được đưa vào một trong các chân này để yêu cầu 8086 ngừng thực hiện chương trình hiện hành để thực hiện một thủ tục ngắt, thủ tục ngắt có quan tâm đến trạng thái gây ra ngắt. Thí dụ, có thể nối sensor nhiệt độ từ nồi hơi nóng vào một ngắt của 8086. Nếu nồi hơi quá nóng, sensor nhiệt độ sẽ kích lối vào ngắt làm cho 8086 ngừng thực hiện chương trình hiện hành và thực hiện thủ tục của dịch vụ ngắt, trong trường hợp này là tắt nguồn nhiên liệu. Một lệnh trở về (return) tại cuối thủ tục của dịch vụ ngắt sẽ trao lại quyền thực hiện cho chương trình bị ngắt.

3.2. CÁC VI XỬ LÍ HỌ 80×86 DÙNG TRONG HỆ ĐIỀU HÀNH ĐA NHIỆM

3.2.1. Các vấn đề chung về hệ điều hành đa nhiệm

Nguyên lí hoạt động cơ bản của một hệ thống phân chia thời gian (timeshare system) là CPU thực hiện một nhiệm vụ trong một vài mili giây rồi thực hiện nhiệm vụ tiếp theo trong một vài mili giây nữa và cứ như vậy qua tất cả các nhiệm vụ rồi quay lại từ nhiệm vụ đầu. Vì tốc độ xử lí của CPU là nhanh nên mỗi người sử dụng một nhiệm vụ tưởng rằng CPU chỉ có phục vụ riêng cho họ. Một hệ điều hành tổ chức hoạt động phân chia thời gian như vậy được gọi là hệ *điều hành đa người dùng* (multiuser operating system). Chương trình hoặc phần chương trình cho mỗi người dùng được coi như là một *nhiệm vụ* (task) và do vậy hệ điều hành này cũng được gọi là *hệ điều hành đa nhiệm* (multitasking operating system). Trong mục này ta sẽ thảo luận một số vấn đề chính liên quan đến việc xây dựng một hệ điều hành đa nhiệm và các vi xử lí cho phép thực hiện được việc đó là 80286, 80386, 80486 và Pentium.

Trong một hệ như vậy, phần của hệ điều hành xác định thời điểm chuyển giữa các nhiệm vụ được gọi là *bộ giám sát* (supervisor). Phương pháp lên lịch trình chuyển nhiệm vụ thông thường là phương pháp phân chia thời gian (time-slice); trong đó, CPU chạy một nhiệm vụ trong chừng vài chục mili giây rồi chuyển tới một nhiệm vụ khác. Phương pháp lên lịch trình khác là dùng mức ưu tiên giữa các nhiệm vụ. Trong đó một nhiệm vụ đang chạy có mức ưu tiên thấp hơn sẽ bị ngắt bởi một nhiệm vụ khác có mức ưu tiên cao hơn. Khi nhiệm vụ có mức ưu tiên cao kết thúc, hệ sẽ quay về chạy nhiệm vụ có mức ưu tiên thấp kể trên. Cách này thích hợp cho một vài ứng dụng điều khiển vì nó cho phép nhiệm vụ quan trọng nhất sẽ được thực hiện đầu tiên. Chip điều khiển ngắt ưu tiên PIC 8259A thường được dùng để khởi tạo và quản lí những yêu cầu dịch vụ kiểu như vậy.

Các thanh ghi, số liệu, các con trỏ v.v... được sử dụng bởi nhiệm vụ đang chạy được gọi là *môi trường hay trạng thái* của nhiệm vụ đó. Khi xảy ra một sự chuyển sang nhiệm vụ khác, môi trường hay trạng thái của nhiệm vụ bị ngắt này phải được cất giữ nhằm cho nhiệm vụ đó có thể được khởi động lại thích hợp trong khoảng thời gian sau. Cách thông thường là giữ các biến môi trường và trạng thái trong một đoạn nhớ đặc biệt hoặc trong ngăn xếp. Một vài hệ điều hành dùng mỗi ngăn xếp tách biệt cho mỗi nhiệm vụ. Khi xảy ra chuyển nhiệm vụ, hệ điều hành cất biến môi trường của nhiệm vụ bị ngắt và con trỏ tới môi trường được cất giữ. Đến thời gian quay lại nhiệm vụ đó, hệ điều hành sử dụng con trỏ để thâm nhập vào biến môi trường đã được cất giữ. Xử lí này thường được gọi là chuyển cảnh (context switching).

Một vấn đề khác liên quan đến hệ điều hành đa nhiệm là các nhiệm vụ cần sử dụng một cách có trật tự các *tài nguyên* như máy in, ổ đĩa v.v... Thí dụ, một người dùng ở một thiết bị đầu cuối nào đó cần đọc một file từ ổ đĩa cứng và in nó trên máy in của hệ thống. Hiển nhiên là các nhiệm vụ đó không thể thực hiện được chỉ trong vài chục mili giây. Do đó phải tăng thời gian truy nhập các tài nguyên để công việc được thực hiện hoàn hảo. Một bit cờ trong bộ nhớ sẽ được sử dụng để chỉ thị liệu ổ đĩa có đang được sử dụng bởi một nhiệm vụ nào đó hay không. Cũng như vậy, một bit cờ khác dùng cho máy in. Nếu nhiệm vụ không thể thâm nhập vào tài nguyên do nó bị bận thì nói rằng nhiệm vụ đó bị chặn. Hầu hết các hệ điều hành dùng cách sắp hàng các nhiệm vụ đợi cho các tài nguyên. Khi một nhiệm vụ kết thúc công việc với một tài nguyên, nó sẽ đặt lại cờ cho tài nguyên đó. Nhiệm vụ tiếp theo trong hàng có thể đặt cờ để chỉ thị rằng tài nguyên bị bận ...

Sự cần thiết phải bảo vệ: Một vấn đề cần quan tâm trong các hệ điều hành đa nhiệm là khi hai hoặc nhiều người dùng có ý định đọc và thay đổi các nội dung của một vùng nhớ trong cùng một thời gian. Lúc này phần chương trình trong đó giá trị của một biến đang được kiểm tra hoặc thay đổi phải được bảo vệ khỏi sự xâm nhập của các nhiệm vụ khác cho đến khi nhiệm vụ này hoàn tất. Phần mã phải được bảo vệ được gọi là *vùng tới hạn* (critical region). Một kỹ thuật được gọi là *loại trừ tương hỗ* (mutual exclusion) được dùng để ngăn không cho hai nhiệm vụ cùng xâm nhập đồng thời vào vùng tới hạn. Trong một hệ phân mảnh thời gian, một cờ được dùng để cung cấp chip kĩ thuật loại trừ tương hỗ. Đoạn chương trình hợp ngữ sau chỉ ra cách mà một cờ được dùng để tạo kĩ thuật loại trừ tương hỗ trong một vùng mã tới hạn.

```
; các lệnh dùng để thâm nhập vùng mã tới hạn được bảo vệ bởi cờ - USER 1
            MOV  AL,01          ; nạp cờ báo vùng bận
HOLD:       XCHG  AL, FLAG      ; hoán chuyển và đặt cờ
            CMP  AL, 01         ; kiểm tra xem vùng có bận ?
            JE  HOLD            ; vòng lặp đến khi vùng rỗi
; các lệnh thâm nhập vùng tới hạn được sen vào ở đây
            MOV  FLAG, 00 ; đặt lại cờ cho vùng tới hạn dùng cho nhiệm vụ khác

; các lệnh dùng để thâm nhập vùng mã tới hạn được bảo vệ bởi cờ -  USER 2
            MOV  AL,01          ; nạp cờ báo vùng bận
HOLD:       XCHG  AL, FLAG      ; hoán chuyển và đặt cờ
            CMP  AL, 01         ; kiểm tra xem vùng có bận ?
            JE  HOLD            ; vòng lặp đến khi vùng rỗi
; các lệnh thâm nhập vùng tới hạn được xen vào ở đây
            MOV  FLAG, 00   ; đặt lại cờ cho vùng tới hạn dùng cho nhiệm vụ khác
```

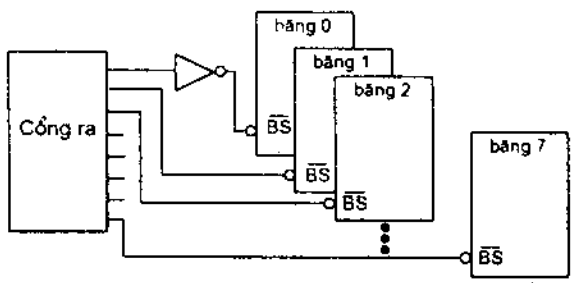
Chuỗi lệnh là như nhau cho các nhiệm vụ. Nếu nhiệm vụ 1 cần nhập vào một phần mã đặc biệt, nó sẽ nạp giá trị cờ báo vùng tới hạn bận vào AL. Một lệnh đơn XCHG AL, FLAG sẽ hoán đổi byte trong AL với byte trong vùng nhớ với tên là FLAG. Việc này phải thực hiện trong một lệnh vì cơ chế phân mảnh thời gian không thể chuyển tới nhiệm vụ khác lúc giữa chừng. Sau khi cờ được đọc nó được so sánh với giá trị bận. Nếu vùng tới hạn bận, chương trình sẽ duy trì vòng lặp đợi trong nhiều mảnh thời gian cho đến khi vùng này rỗi. Nếu cờ bằng 0, chương trình sẽ thâm nhập vùng tới hạn. Lệnh XCHG đã đặt cờ để chỉ thị vùng tới hạn bận. Sau khi thao tác tại vùng tới hạn kết thúc, lệnh MOV FLAG, 00 đặt lại cờ để báo rằng vùng tới hạn không bận nữa. Nhiệm vụ 2 khi đó có thể hoán đổi cờ và nhảy vào vùng tới hạn nếu cần thiết. Một vùng khác cần được bảo vệ là mã hệ điều hành.

3.2.2. Quản lí bộ nhớ

Có hai lí do cần phải quản lí bộ nhớ một cách đặc biệt trong hệ điều hành đa nhiệm. Lí do thứ nhất là bộ nhớ vật lí thường không đủ lớn để chứa hệ điều hành và tất cả các chương trình ứng dụng đang được chạy bởi các người dùng khác nhau. Lí do thứ hai là phải đảm bảo rằng các nhiệm vụ chạy không xâm nhập vào các vùng nhớ được bảo vệ. Một vài cách quản lí bộ nhớ có thể được làm bởi phần mềm hệ điều hành, nhưng thường phải có thêm hỗ trợ của phần cứng gọi là *đơn vị quản lí bộ nhớ* MMU (memory management unit). Có thể điểm qua về những phương pháp khác để giải quyết vấn đề bộ nhớ có dung lượng hạn chế như sau:

Một trong những phương pháp là sử dụng là viết trình hợp dịch ra thành các mô-đun và dùng *sơ đồ nhớ phủ* (overlay). Khi trình hợp dịch assembler chạy, mô-đun đang chạy của trình hợp dịch được nạp vào bộ nhớ và một khối nhớ gọi là vùng phủ được dự trữ dùng cho trình hợp dịch. Mô-đun thứ nhất của trình hợp dịch được nạp vào vùng phủ này. Khi trình hợp dịch chạy tới điểm mà ở đó nó cần tới mô-đun tiếp theo, nó sẽ đọc mô-đun đó (được coi như một vùng phủ) từ đĩa vào vùng phủ dự trữ trong bộ nhớ. Khi trình hợp dịch đạt tới điểm mà ở đó nó cần vùng phủ khác, nó sẽ đọc vùng phủ khác từ đĩa và nạp vào cùng một vùng phủ trong bộ nhớ. Các thức này thường được dùng cho các trình hợp dịch, các trình biên dịch, các bộ xử lí từ, các chương trình bảng tính.

Một cách khác để mở rộng không gian nhớ trong máy vi tính là *chuyển băng nhớ*. Các vi xử lí chỉ có 16 đường địa chỉ thì chỉ quản lí được tới 64 kbyte nhớ vật lí. Bằng việc sử dụng và chuyển các băng nhớ có thể mở rộng bộ nhớ này như hình 3.4. Phần cứng được cấu hình sao cho khi lần đầu đóng nguồn, băng nhớ 16 kbyte có nhãn là 0 được cho phép. Giả thử băng này chiếm không gian địa chỉ hệ thống từ 4000h đến 7FFFh và các đường địa chỉ hệ thống A0 đến A13 được dùng để định địa chỉ các byte nhớ này. Để chuyển tới băng 1, một byte làm tắt băng 0 và bật băng 1 sẽ được xuất tới cổng được chọn. Bây giờ băng 1 lại chiếm không gian địa chỉ từ 4000h tới 7FFFh và các đường địa chỉ từ A0 đến A13 lại được dùng để địa chỉ hoá các byte nhớ trong băng này. Như thấy trên hình, sơ đồ chuyển băng cho phép bộ xử lí thâm nhập 8 băng, mỗi băng 16 kbyte nhớ tức là tổng cộng có 128 kbyte nhớ qua một cửa sổ 16 kbyte trongkhông gian địa chỉ của bộ xử lí. Tiêu chuẩn LIM/EMS 3.2 (Lotus-Intel-Microsoft Expanded Memory Standard) là một phương tiện chuyển băng nhớ trong các máy tính IBP PC và PS/2 để mở rộng bộ nhớ thành bộ nhớ dãn rộng (expanded memory). Phần cứng cho nó thường là một bản mạch cắm vào khe cắm mở rộng chứa đến 8 Mbyte trang nhớ, mỗi trang 16 kbyte là các băng nhớ và các thanh ghi chuyển băng. Các thanh ghi này được dùng để điều khiển trang nhớ nào được chọn.



Hình 3.4 Sơ đồ khối cho thấy bộ nhớ có thể được mở rộng bằng chuyển băng nhớ.

Bộ nhớ ảo về cơ bản là phần mở rộng của sơ đồ bộ nhớ cache. Bộ nhớ cache, sẽ được thảo luận ở chương sau, là bộ nhớ có dung lượng nhỏ nhưng tốc độ truy cập rất nhanh. Sơ đồ bộ nhớ ảo đơn giản chỉ là thêm vào một ổ đĩa cứng tới cấp bậc bộ nhớ. Ổ đĩa cứng trở thành bộ nhớ chính chứa số liệu và chương trình. DRAM hoạt động như là một cache trung gian và bộ nhớ cache SRAM hoạt động như là một cache tốc độ cao cho DRAM. Trong một hệ thống bộ nhớ ảo, các đoạn mã và số liệu hiện đang được dùng cho chương trình sẽ được nạp từ đĩa vào DRAM và khi cần sẽ được truy nhập bởi bộ điều khiển cache. Nếu một chương trình đang chạy cần một đoạn nhớ mà lại không có trong DRAM thì đoạn đòi hỏi này ddược đọc từ đĩa tới DRAM. Nếu DRAM đầy thì một trong các đoạn trong đó sẽ được hoán đổi tới đĩa để tạo chỗ trống cho đoạn đòi hỏi được nạp vào DRAM.

Có ba cách khác nhau để khởi tạo các khối mã và số liệu được hoán đổi vào và ra DRAM. Một sơ đồ là hoán đổi các đoạn nhớ. Sơ đồ này có ưu điểm là các đoạn phù hợp với các cấu trúc mã và số liệu trong chương trình. Nhược điểm là ở chỗ nếu kích thước đoạn lớn thì thời gian hoán đổi sẽ lâu làm chậm chương trình. Sơ đồ thứ hai là dùng các trang nhớ có kích thước cố

định (điển hình là 4 kbyte). Những trang nhỏ như vậycó thể được hoán đổi một cách nhanh chóng nhưng chúng không phù hợp với cấu trúc lôgic của chương trình. Cách thứ ba là cho phép người lập trình viết một chương trình sử dụng các đoạn lôgic và chia các đoạn thành các trang 4 kbyte dùng cho hoán đổi vào và ra bộ nhớ vật lí.

Thuật ngữ ảo ở đây được gán cho không gian nhớ xuất hiện theo con mắt của người lập trình mà không phải là một thực thể vật lí hiện diện trong DRAM ở bộ nhớ chính. Nói cách khác, nếu ta đang viết một chương trình cho một hệ thống với bộ nhớ ảo, ta có thể tạo ra các đoạn nhớ như thể ta có , thí dụ, đến hàng gigabyte không gian nhớ, mặc dù hệ thống thực sự chỉ có 4 Mbyte nhớ vật lí. Không gian nhớ ảo có thể lớn hơn bộ nhớ vật lí vì tất cả các đoạn nhớ lôgic không hiện diện trong bộ nhớ vật lí cùng một lần. Như với sơ đồ cache, một hệ thống nhớ ảo làm việc vì hầu hết các chương trình chỉ cần một phần nhỏ mã lệnh và số liệu trong một thời gian nhất định.

Bộ nhớ ảo có thể được quản lí toàn bộ bởi hệ điều hành nhưng hầu hết các máy vi tính đều sử dụng một thiết bị phần cứng gọi là đơn vị quản lý bộ nhớ MMU để trợ giúp cho bộ xử lí. Các vi xử lí Intel có các MMU được tích hợp ngay trên cùng một chip với CPU. Khi ta viết một chương trình hợp ngữ, thường gán cho các địa chỉ một tên. Các địa chỉ làm việc trong chương trình được gọi là địa chỉ lôgic bởi vì chúng chỉ thị các vị trí lôgic của mã lệnh và số liệu. Một thí dụ là lệnh JNZ NEXT. Nhãn NEXT đại diện cho địa chỉ lôgic mà chương trình sẽ nhảy đến đó khi cờ zero không được đặt. Khi chương trình 8086 được hợp dịch, mỗi địa chỉ lôgic được đại diện bởi 16 bit độ lệch (offset) và 16 bit địa chỉ cơ sở đoạn nhớ. Khi đó đơn vị BIU trong 8086 sẽ phát ra một địa chỉ vật lí thực sự bằng việc kết hợp 2 phần này với nhau theo cách nhân 16 bit địa chỉ cơ sở với 16 và cộng với 16 bit offset.

Các bộ chọn của 80286, 80386 và 80486 có 14 bit địa chỉ, 2 bit mức đặc quyền. Với 14 bit địa chỉ có thể chọn 1 trong số 2^{14} = 16 384 bộ mô tả đoạn trong bảng các bộ mô tả. Vì rằng mỗi bộ mô tả đại diện cho một đoạn nhớ nên chương trình có thể xâm nhập tới 16 384 đoạn nhớ. Trong 80286 phần offset của địa chỉ ảo là 16 bit nên mỗi đoạn có thể chứa tới 64 Kbyte nhớ. Không gian địa chỉ lôgic hoặc địa chỉ ảo sẽ là 16 384 đoạn × 65 536 byte/đoạn hay bằng 1 Gbyte.

Bộ nhớ vật lí là lượng RAM và ROM thực sự có trong hệ thống. Khi MMU nhận được một địa chỉ lôgic từ CPU, nó sẽ kiểm tra xem liệu đoạn chứa địa chỉ đó có tồn tại trong không gian nhớ vật lí hay không. Nếu có tồn tại, MMU sẽ cộng thành phần offset (độ lệch) của địa chỉ với thành phần địa chỉ cơ sở của đoạn chứa trong bộ mô tả đoạn để hình thành nên một địa chỉ vật lí rồi xuất ra bus địa chỉ. Các từ mã hoặc số liệu được địa chỉ sẽ được trả về CPU trên bus số liệu. Nếu MMU phát hiện rằng đoạn được chọn không hiện diện trong không gian nhớ vật lí nó sẽ gửi một yêu cầu ngắt tới CPU. Hệ điều hành lúc này sẽ tiến hành một thủ tục ngắt đọc đoạn mã lệnh và số liệu mong muốn từ đĩa và nạp nó vào bộ nhớ vật lí. Tiếp đó MMU sẽ tính toán và xuất ra địa chỉ vật lí như nói trước đây. Hoạt động này là bán tự động nên người dùng không nhận thấy rằng đoạn đã được nạp. Trong một hệ thống có một lượng lớn bộ nhớ vật lí tỉ lệ "trúng đích" này có thể đạt tới 90 – 95%.

Khi CPU hoặc MMU muốn nạp một đoạn nhớ từ đĩa vào bộ nhớ vật lí, trước tiên nó phải tạo một vùng trống trong đó. Tuỳ vào hệ thống, nó có thể làm điều này bằng việc nén các đoạn đã tồn tại và thay đổi các bộ mô tả đoạn để chỉ tới một vùng nhớ vật lí mới hoặc bằng việc hoán chuyển đoạn đang muốn đưa vào với đoạn hiện đang ở trong bộ nhớ vật lí. Để quyết định đoạn nào được hoán chuyển trở lại bộ nhớ, nhiều hệ thống sử dụng 1 bit thâm nhập (accessed bit)

trong bộ mô tả để giám sát thời gian đoạn được sử dụng. Một đoạn được sử dụng ở mức thấp thường được hoán chuyển ra đĩa. Một vài hệ thống nhớ ảo cũng có một bit gọi là *dirty bit* trong mỗi bộ mô tả đoạn. Bit này sẽ được đặt nếu các nội dung của đoạn được thay đổi, một đoạn sẽ được viết trở lại đĩa khi cần không gian nhớ. Nếu bit này được xoá, tức là khi đoạn không trong trạng thái biến đổi và nội dung của nó đã được copy trên đĩa thì nội dung của đoạn có thể được viết đè lên bởi một đoạn mới. Bằng việc kiểm tra như vậy sẽ tiết kiệm được thời gian đòi hỏi để viết một đoạn lên đĩa.

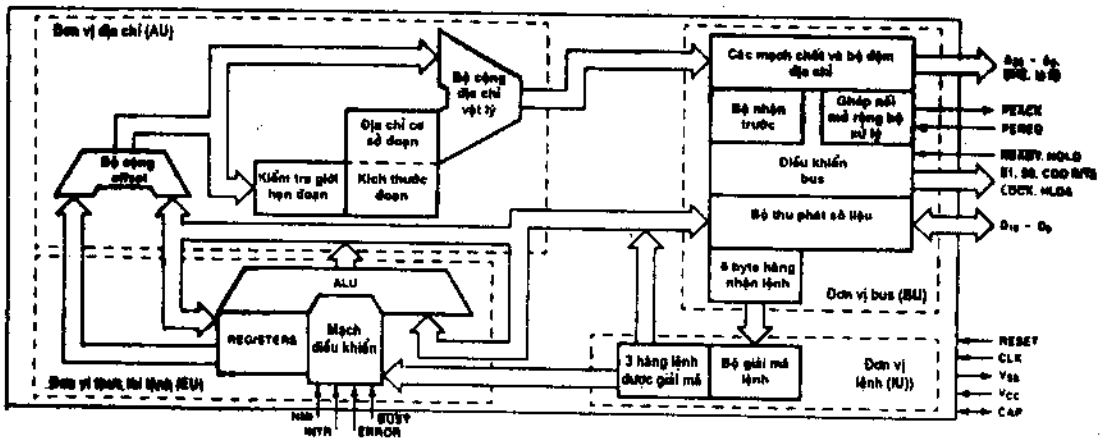
Ngoài việc cho khả năng tạo bộ nhớ ảo, việc sử dụng bảng bộ mô tả để dịch một địa chỉ lôgic ra địa chỉ vật lí còn có một ưu điểm khác nữa. Thành phần bộ chọn của mỗi địa chỉ lôgic chứa 2 bit đại diện cho mức đặc quyền của phần chương trình đang muốn thâm nhập vào đoạn nhớ. Bộ mô tả cho mỗi đoạn chứa 2 bit đại diện cho mức đặc quyền của đoạn đó. Khi một chương trình đang chạy muốn thâm nhập vào đoạn, MMU so sánh mức đặc quyền trong bộ chọn với mức đặc quyền trong bộ mô tả. Nếu bộ chọn đoạn có mức đặc quyền bằng hoặc lớn hơn thì MMU sẽ cho phép đoạn được thâm nhập. Nếu mức đặc quyền của bộ chọn thấp hơn mức đặc quyền trong bộ mô tả thì MMU sẽ từ chối thâm nhập và gửi một tín hiệu ngắt tới CPU để chỉ thị là đã có sự vi phạm mức đặc quyền. Như vậy, các bit đặc quyền và phương pháp phát ra địa chỉ vật lí một cách gián tiếp này cung cấp một cơ chế bảo vệ cho các đoạn nhớ, thí dụ ngăn không cho các chương trình ứng dụng có mức đặc quyền thấp thâm nhập vào phần lõi của hệ điều hành có mức đặc quyền cao nhất.

3.2.3. Vi xử lý 80286

Vi xử lí 80286 được dùng đầu tiên trong các máy IBM PC/AT và ở một số loại IBM PS/2 và máy nhái. Với 134.000 transistor, 80286 có tốc độ truyền dữ liệu lên tới 2 MIPS (ở tần số xung nhịp hệ thống 8 MHz). Nó tương thích hoàn toàn với 8086, điều đó có nghĩa rằng bất kì một chương trình nào được viết trong 8086 sẽ chạy tốt trong 80286. Thêm nữa, 80286 chạy chương trình nhanh hơn và linh hoạt hơn 8086. Việc tăng tốc độ do tốc độ nhịp đồng hồ nhanh hơn (6MHz so với 4,77MHz ở 8088). Có lẽ sự cải tiến quan trọng nhất đối với 80286 là nó hỗ trợ cho hoạt động đa nhiệm. 80286 có thể chạy ở một trong hai chế độ: thực và bảo vệ . Trong chế độ thực, 80286 được lập trình giống như 8086 và có thể thâm nhập một dải địa chỉ lên tới 1 Mbyte giống 8086. Trong chế độ bảo vệ, 80286 có thể địa chỉ hoá tới 16 Mbyte nhớ và có thể dự trữ một lượng bộ nhớ định trước cho mỗi chương trình khả thi nhằm đề phòng vùng nhớ này có thể được dùng bởi bất kì một chương trình nào khác. Điều này có nghĩa rằng một vài chương trình có thể chạy đồng thời (đó là đa nhiệm vụ) mà không xảy ra sự rủi ro về một chương trình tình cờ thay đổi các nội dung của các vùng nhớ của các chương trình khác. Một hệ điều hành khi sử dụng chế độ bảo vệ 80286 có thể định vị bộ nhớ trong số một vài nhiệm vụ thì hiệu quả hơn hệ điều hành dùng 8086. Hình 3.5 là sơ đồ khối bên trong của 80286.

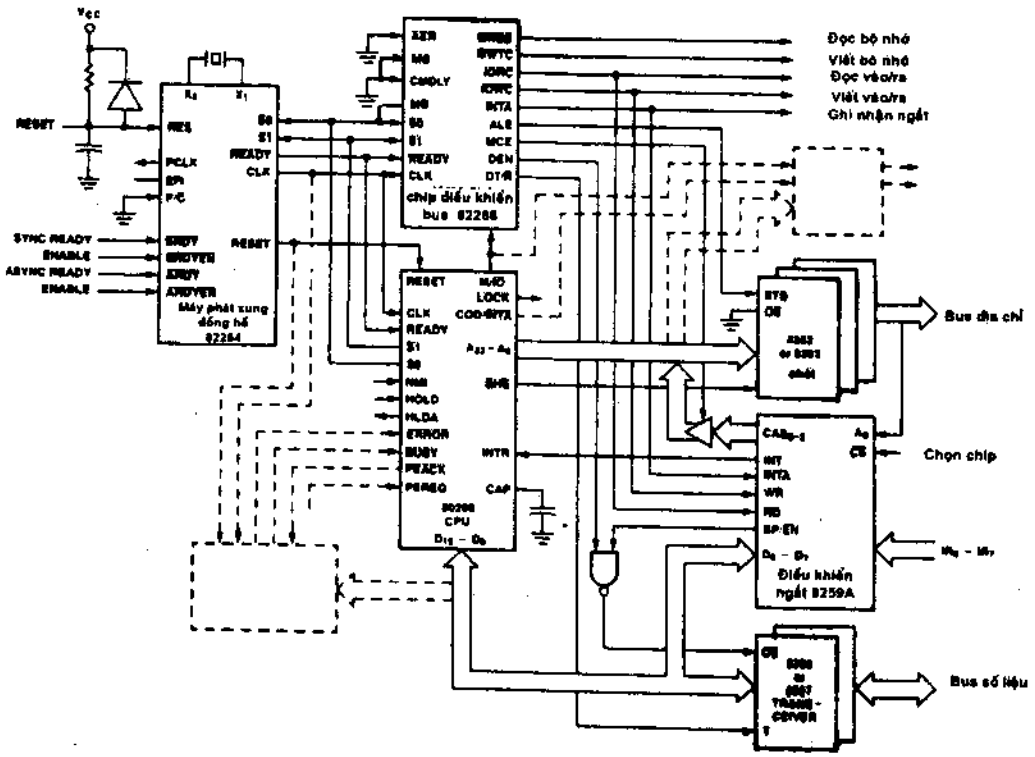
80286 có bus số liệu rộng 16 bit và bus địa chỉ không dồn kênh rộng 24 bit cho phép bộ xử lý thâm nhập tới 16 Mbyte nhớ vật lý. Phần cứng bộ nhớ được khởi tạo với một băng nhớ lẻ và một băng chẵn giống như 8086. Băng chẵn sẽ được phép khi A0 ở mức thấp và băng lẻ được phép khi $\overline{BHE}$ ở mức thấp. Để truy xuất một từ thì cả hai chân này phải cùng ở mức thấp. Các bộ đệm bên ngoài được dùng cho cả bus địa chỉ và số liệu. Về khía cạnh điều khiển thì 80286 hoạt động tương tự như 8086 trong chế độ MAX. Các tín hiệu trạng thái $\overline{S0}$, $\overline{S1}$ và $M/\overline{IO}$ được giải mã bởi chip điều khiển bus 82288 để tạo ra các tín hiệu điều khiển bus, đọc, viết và ghi nhận ngắt. Các chân *HOLD*, *HLDA*, *INTR*, $\overline{INTA}$, *NMI*, $\overline{READY}$ và $\overline{LOCK}$ cũng như $\overline{RESET}$ về cơ bản có chức năng hoạt động giống như trong 8086. Chip 82284 được dùng để

phát ra các xung đồng hồ cho 80286 và đồng bộ các tín hiệu *RESET* và $\overline{READY}$. Bốn tín hiệu PEREQ, $\overline{PEACK}$, $\overline{BUSY}$ và $\overline{TEST1}$ được dùng để ghép nối 80286 với các phần mở rộng của vi xử lý như bộ đồng xử lí toán 80287. Dạng sóng trong chu kì máy của 80286 cũng giống trong 8086.



Hình 3.5 Sơ đồ khối của 80286.

Hình 3.6 là một sơ đồ đơn giản nối mạch giữa 80286 và các chip khác.



Hình 3.6 Mạch điện với 80286.

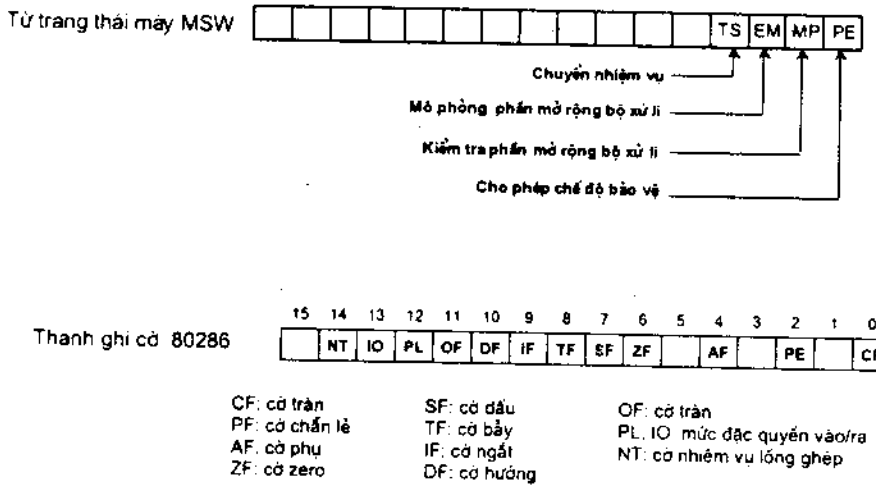
1. Chế độ hoạt động địa chỉ thực

Sau khi 80286 được reset nó sẽ khởi phát chạy trong chế độ thực giống ở 8086. Trong chế độ này 80286 có thể định địa chỉ tới 1 Mbyte nhớ vật lí. Do được tăng cường cải tiến về các kiểu

định địa chỉ đường ống và các phần cứng khác nên nó sẽ chạy nhanh hơn 8086 ở cùng một tốc độ xung nhịp. Lúc này bảng vectơ ngắt được đặt ở 1 kbyte nhớ đầu tiên của bộ nhớ. 80286 có thêm một vài loại ngắt bổ xung so với 8086.

2. *Chế độ hoạt động bảo vệ*

Bước đầu tiên để chuyển 80286 sang chế độ bảo vệ là đặt bit cho phép bảo vệ trong thanh ghi từ trạng thái máy MSW (machine status word) lên mức lôgic cao. Hình 3.7 chỉ ra định dạng của từ này cùng thanh ghi cờ trong 80286.



Hình 3.7 Định dạng từ trạng thái máy và thanh ghi cờ.

Để thay đổi các bit trong MSW người dùng nạp từ mong muốn vào một thanh ghi hoặc vùng nhớ và chạy lệnh nạp từ trạng thái máy LMSW. Bước cuối cùng là chạy một lệnh nhảy bên trong đoạn để khởi phát chương trình chính của hệ thống. Bước nhảy này là cần thiết để sắp hàng các byte lệnh vì trong chế độ bảo vệ, hàng nhận lệnh có chức năng khác với trong chế độ thực. Chuyển 80286 sang chế độ bảo vệ cho phép mạch MMU cung cấp một bộ nhớ ảo và việc bảo vệ các đoạn nhớ. Như đã nói ở trên, MMU dùng 14 bit bộ chọn để thâm nhập bộ mô tả đoạn trong bảng các bộ mô tả. Bộ mô tả chứa 24 bit địa chỉ cơ sở vật lý, các mức đặc quyền và một vài bit điều khiển của đoạn cần thâm nhập. Nếu mức đặc quyền trong bộ chọn bằng hoặc lớn hơn mức đặc quyền trong bộ mô tả thì sự thâm nhập này được phép. MMU cũng kiểm tra bit "P" trong bộ mô tả để xác định xem liệu đoạn nhớ cần truy xuất có trong bộ nhớ vật lí hay không. Nếu không, một ngoại lệ sẽ được phát ra. Lúc này thủ tục dịch vụ sẽ nạp đoạn đó vào bộ nhớ và trả về chương trình ngắt. Nếu cả hai điều kiện kể trên thoả mãn, MMU sẽ cộng 16 bit địa chỉ offset trong địa chỉ lôgic với 24 bit địa chỉ cơ sở trong bộ mô tả để tạo ra 24 bit địa chỉ vật lí cần tham nhập trong bộ nhớ. Một khi 80286 chuyển sang chế độ bảo vệ rồi thì chỉ có cách reset lại hệ thống mới đưa nó quay trở lại chế độ thực.

So với 8086, vi xử lý 80286 có một số lệnh mới trong chế độ bảo vệ như sau:
CTS – Xoá cờ chuyển nhiệm vụ trong từ trạng thái máy.
LGDT – Nạp thanh ghi bảng bộ mô tả toàn cục từ bộ nhớ.
SGDT – Lưu trữ nội dung thanh ghi bảng bộ mô tả toàn cục vào bộ nhớ.
LIDT – Nạp thanh ghi bảng bộ mô tả ngắt từ bộ nhớ.
LLDT – Nạp bộ chọn và bộ mô tả liên quan vào LDTR.

SLDT – Lưu trữ bộ chọn từ LDTR vào một thanh ghi hoặc bộ nhớ.
LTR – Nạp thanh ghi nhiệm vụ với bộ chọn và bộ mô tả cho TSS.
STR – Lưu trữ bộ chọn từ thanh ghi nhiệm vụ vào thanh ghi hoặc bộ nhớ.
LMSW – Nạp thanh ghi từ trạng thái máy từ thanh ghi hoặc bộ nhớ.
SMSW – Lưu trữ thanh ghi từ trạng thái máy vào thanh ghi hoặc bộ nhớ.
LAR – Nạp các byte quyền thâm nhập của bộ mô tả vào thanh ghi hoặc bộ nhớ.
LSL – Nạp độ dài đoạn từ bộ mô tả vào thanh ghi hoặc bộ nhớ.
ARPL - Điều chỉnh mức đặc quyền yêu cầu của bộ chọn (chỉ điều chỉnh xuống).
VERR – Xác định liệu đoạn nhớ được chỉ định bởi bộ chọn có thể đọc được không.
VERW – Xác định liệu đoạn nhớ được chỉ định bởi bộ chọn có thể viết được không.

3.2.4. Vi xử lí 80386

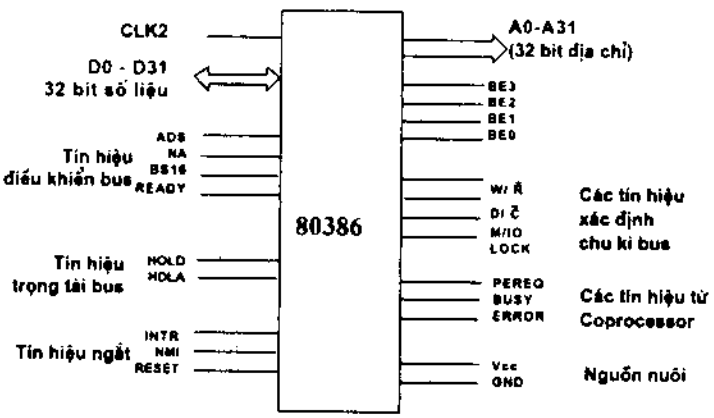
Vi xử lí 80386 được dùng trong các máy IBM PS/2 80 và nhiều máy nhái khác. Nó hỗ trợ các chức năng cơ bản như 80286 và đảm bảo việc quản lí bộ nhớ ở chế độ bảo vệ như trong 80286. Tốc độ xung nhịp ở đây từ 16 đến 33 MHz và cao hơn. Với 275.000 transistor, 80386 có tốc độ 11 MIPS (tại 33 MHz).

Vì 80386 là loại có độ rộng bus số liệu và địa chỉ đều là 32 bit và có mạch tính số học-lôgic ALU 32 bit nên nó có thể làm việc trực tiếp với từ dữ liệu dài 32 bit. Các đoạn nhớ của 80386 có thể dài tới 4 Gbyte và một chương trình có thể có tới 16 384 đoạn. Không gian địa chỉ ảo lúc đó là 16 384 × 4Gbyte = 64 Tbyte nhớ. Với 32 bit địa chỉ, 80386 cho phép địa chỉ hoá tới 4 Gbyte nhớ vật lí. Nó có một chế độ hoạt động kiểu 8086-ảo cho phép dễ dàng chuyển qua lại giữa các nhiệm vụ trong chế độ bảo vệ và chế độ thực.

Sơ đồ các nhóm chân tín hiệu của nó như hình 3.8.

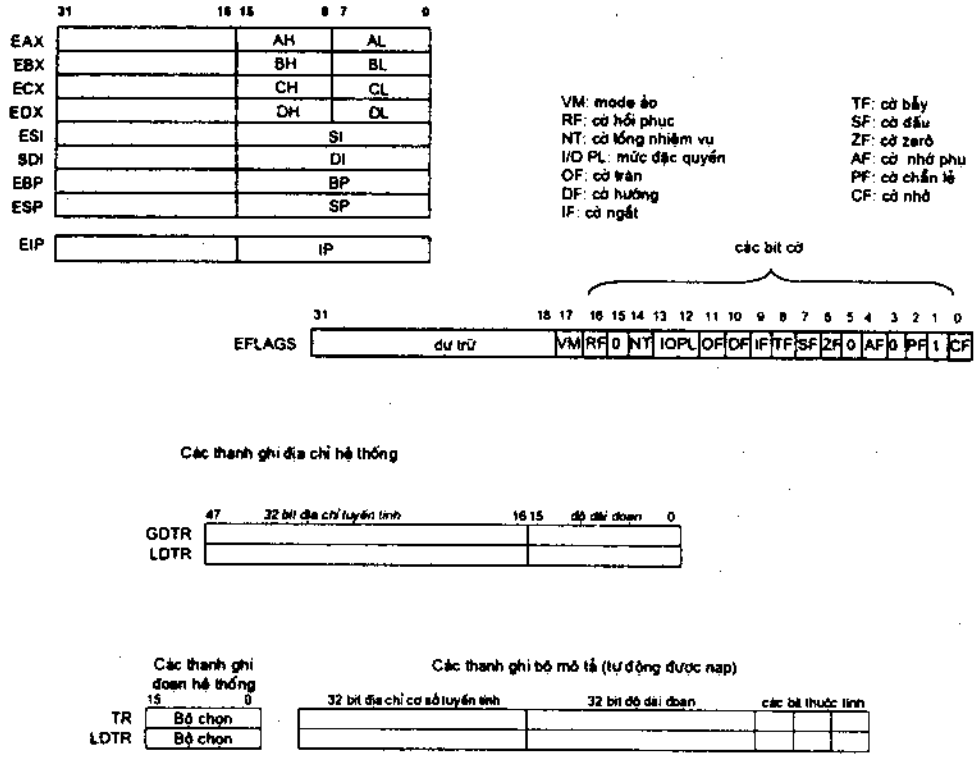
1. Chế độ hoạt động thực: Sau khi reset, 80386 hoạt động trong chế độ thực cơ bản cũng giống như ở 8086 và 80286. Các thanh ghi trong 80386 vì dài 32 bit nên được kí hiệu thêm chữ E (extended) ở đầu, thí dụ EAX, EBX v.v...

Một đặc điểm mới là nó có các thanh ghi gỡ rối. Một bộ gỡ rối mềm có thể nạp các địa chỉ điểm dừng trong chúng để hỗ trợ cho việc gỡ rối. 80386 có thể được ra lệnh "dừng" khi đơn vị địa chỉ trong bộ xử lí tính toán một địa chỉ tuyến tính trùng hợp với một trong các địa chỉ ở các thanh ghi gỡ rối. Phương pháp cũ để đặt điểm dừng là dùng lệnh điểm dừng, thí dụ INT 3 không thể dùng để gỡ rối mã trong ROM trong khi phương pháp thanh ghi điểm dừng có thể vì nó không phụ thuộc vào các byte mã đang được biến đổi. Thanh ghi cờ EFLAG 32 bit trong 386 là thanh ghi cờ trong 8086 và 286 được mở rộng. Trong chế độ thực còn có các thanh ghi điều khiển từ CR0 đến CR3. 16 bit thấp của CR0 là từ trạng thái máy MSW của 80286. Vi xử lí 80386 cũng chuyển sang chế độ bảo vệ bằng việc đặt bit ít ý



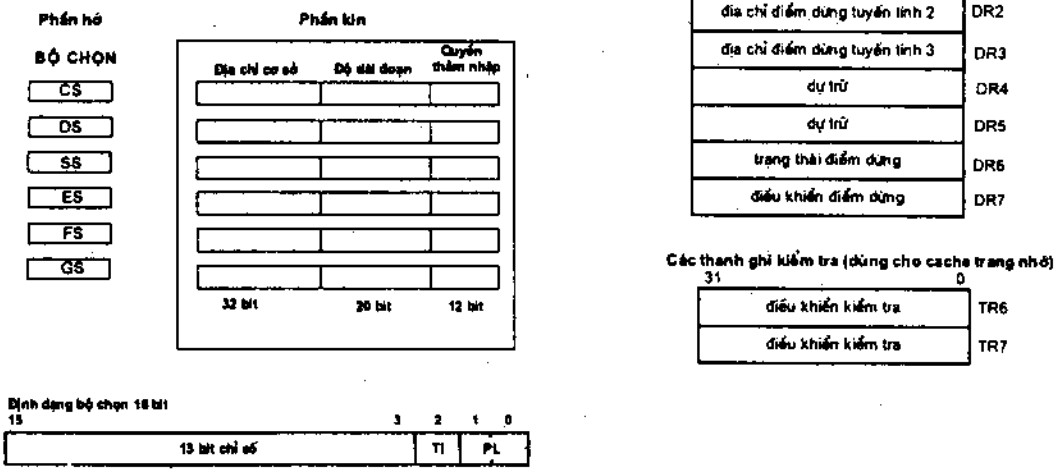
Hình 3.8 Các nhóm chân tín hiệu của 80386.

nghĩa nhất LSB của thanh ghi này tới 1. Thanh ghi CR1 được dự trữ và thanh ghi CR2 và CR3 được dùng cho chế độ trang sẽ được thảo luận về sau. Hình 3.9a và b là các thanh ghi của 80386.



Hình 3.9a Các thanh ghi đa năng và các thanh ghi hệ thống.

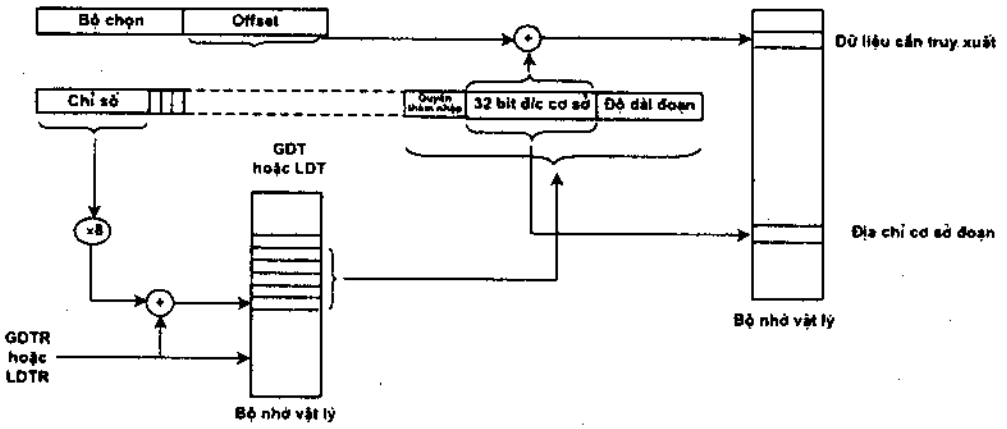
Các thanh ghi đoạn gồm 2 phần: *phần hở* gọi là *bộ chọn* được nạp bởi người lập trình và *phần kín* do CPU nạp



Hình 3.9b Các thanh ghi đoạn với định dạng bộ chọn và các thanh ghi gỡ rối, kiểm tra.

2. Chế độ hoạt động bảo vệ: Sức mạnh của 80386 là ở khả năng hoạt động trong chế độ bảo vệ và chế độ 8086 ảo. Như nói trên đây, 386 chuyển từ chế độ thực sang chế độ bảo vệ bằng việc đặt bit LSB của thanh ghi CR0 lên 1. Sơ đồ địa chỉ bộ nhớ ảo của 386 trong chế độ bảo vệ rất giống ở 286 ngoại trừ rằng các đoạn của nó lớn hơn và một cơ chế tuỳ chọn là bộ nhớ trang cho phép các đoạn được chia thành các trang 4 kbyte để có thể hoán chuyển vào ra bộ nhớ vật lí nhanh hơn.

Trong chế độ bảo vệ, mỗi một địa chỉ của 386 gồm một *bộ chọn* đoạn 16 bit và offset 32 bit. Bộ chọn đoạn chỉ tới *bộ mô tả* đoạn trong *bảng các bộ mô tả* và offset xác định vùng mã lệnh hoặc số liệu mong muốn trong đoạn. Sử dụng giá trị offset 32 bit có nghĩa là độ dài các đoạn có thể là bất kì từ 1 tới 2^{32} = 4 Gbyte nhớ. Hình 3.10 chỉ ra dịnh dạng của bộ chọn và cách dùng nó để thâm nhập vào bảng bộ mô tả.



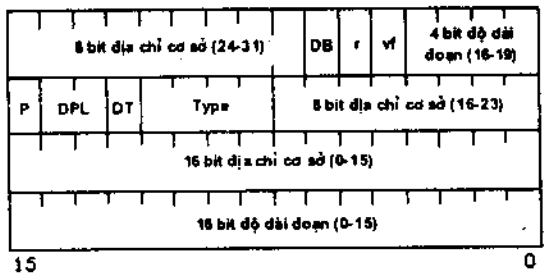
Hình 3.10 Cách 386 dùng bộ chọn để thâm nhập một bộ mô tả trong bảng các bộ mô tả và cách tính địa chỉ tuyến tính.

13 bit chỉ số của bộ chọn được nhân với 8 và được dùng như một con trỏ tới bộ mô tả đoạn trong một bảng các bộ mô tả. Giá trị chỉ số được nhân với 8 vì mỗi bộ mô tả dài 8 byte trong bảng các bộ mô tả. Bộ mô tả chứa địa chỉ cơ sở vật lý cho đoạn nhớ. MMU cộng địa chỉ này với địa chỉ hiệu dụng hoặc phần offset của địa chỉ lôgic từ câu lệnh để phát ra một địa chỉ vật lí.

Có hai loại bảng bộ mô tả trong một hệ thống 386 là toàn cục và cục bộ. Một hệ thống chỉ có một bảng bộ mô tả toàn cục GDT (Global descriptor table). Bảng này chứa các bộ mô tả đoạn cho các đoạn nhớ của hệ điều hành và bộ mô tả cho các đoạn cần đuợc thâm nhập bởi tất cả các nhiệm vụ. Bảng bộ mô tả cục bộ LDT được khởi tạo trong hệ thống dùng cho mỗi một nhiệm vụ hoặc một nhóm các nhiệm vụ có quan hệ chặt chẽ với nhau. Nếu bit chỉ thị TI (bit 2) của bộ chọn bằng 0 thì 13 bit chỉ số sẽ chỉ tới bộ mô tả nằm trong bảng bộ mô tả toàn cục. Nếu TI bằng 1 thì sẽ chỉ tới bảng mô tả cục bộ. Hai bit ít ý nghĩa của bộ chọn gọi là các bit mức đặc quyền yêu cầu RPL.

Xét một bộ mô tả đoạn như hình trên thấy rằng địa chỉ cơ sở được biểu diễn bằng 32 bit trong khí chỉ có 20 bit được chọn để biểu diện cho độ dài của đoạn. Khi bit G bằng 0 thì giá trị 20 bit này sẽ đại diện cho độ dài đoạn tính ra byte và đoạn có thể có độ dài lên tới 1 Mbyte. Nếu bit G bằng 1, thì giá trị 20 bit này sẽ đại diện cho độ dài của đoạn trong các khối 4 kbyte. Giá trị cực đại của 1048576 khối × 4 kbyte/khối cho ta độ dài cực đại của đoạn lên tới 4 Gbyte. Nếu có ý định thâm nhập vùng ngoài giới hạn trên thì sẽ xảy ra ngoại lệ 5. Byte 5 của bộ mô tả (byte thâm nhập) chứa thông tin về mức đặc quyền, về quyền thâm nhập và loại đoạn nhớ.

G: chỉ độ dài đoạn tính bằng byte hay trang.
G = 0: độ dài lớn nhất của đoạn là 2^{20} = 1Mbyte;
G = 1: độ dài lớn nhất của đoạn là 2^{20} trang = 4 Gbyte.
DB = 0: địa chỉ / toán hạng 16 bit;
DB = 1: địa chỉ / toán hạng 32 bit.
r : dự trữ. vf : cho người dùng hoặc hệ điều hành.
P = 1: đoạn có trong bộ nhớ vật lí;
P = 0: đoạn không có trong bộ nhớ.
DPL : mức đặc quyền của đoạn (0-3).
DT : loại bộ mô tả DT = 1: đoạn ứng dụng,
DT = 0 :đoạn hệ thống.



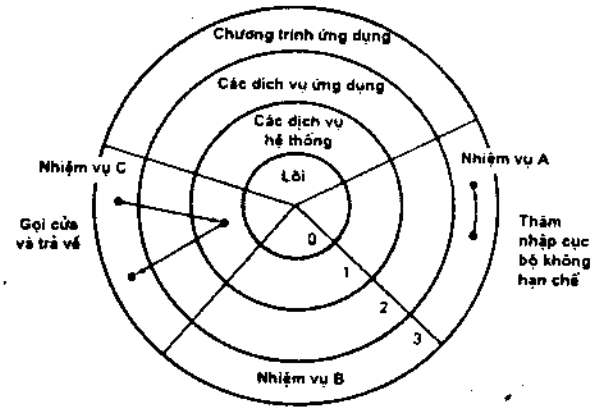
Các giá trị thường là: base 24-31 = 0000 0000; G = 0; DB = 0; vf = 0; limit 16-19 = 0000.

Khi một chương trình có ý định thâm nhập vào một đoạn, bộ chọn cho đoạn đó được nạp vào phần hở của thanh ghi đoạn. Thí dụ, để thâm nhập vào đoạn số liệu bộ chọn phải được nạp vào phần hở của thanh ghi DS, ES, FS hoặc GS. Bộ mô tả đoạn lúc này được tự động nạp vào phần kín của thanh ghi đoạn (còn gọi là cache của bộ mô tả đoạn). Nếu mức đặc quyền của bộ chọn và mức đặc quyền của đoạn mã lệnh hiện tại cao hơn mức đặc quyền của bộ mô tả thì một ngoại lệ sẽ xảy ra và cấm sự thâm nhập đoạn. Ngược lại bit P trong bộ mô tả đoạn sẽ được kiểm tra để xem liệu đoạn có hiện diện trong bộ nhớ vật lí hay không. Nếu không thì một ngoại lệ (số 11) sẽ xảy ra và chương trình ngoại lệ sẽ đọc đoạn từ đĩa vào bộ nhớ vật lí. Khi đoạn đã ở trong bộ nhớ vật lí, đơn vị địa chỉ sẽ tính các địa chỉ vật lí để thâm nhập các từ số liệu trong đoạn. Như trên hình cho thấy, giá trị offset từ địa chỉ lôgic được cộng với địa chỉ cơ sở của đoạn từ bộ mô tả để hình thành một địa chỉ tuyến tính. Khi 386 hoạt động chỉ trong chế độ đoạn, địa chỉ tuyến tính này là địa chỉ vật lí được xuất ra trên các đường địa chỉ và BE tới bộ nhớ.

80386 lưu trữ các địa chỉ cơ sở và độ dài đoạn cho các bảng bộ mô tả GDT và LDT hiện đang sử dụng trong các thanh ghi bên trong. Thanh ghi bảng các bộ mô tả toàn cục GDTR được dùng để giữ 32 bit địa chỉ cơ sở và độ dài đoạn cho bảng bộ mô tả toàn cục. Thanh ghi này được khởi động với lệnh nạp thanh ghi bảng bộ mô tả toàn cục LGDT khi khởi động hệ thống. Thanh ghi bảng bộ mô tả cục bộ được dùng để lưu giữ các địa chỉ cơ sở và độ dài của bảng bộ mô tả cục bộ cho nhiệm vụ đang chạy. Lệnh LLDT được dùng để nạp thanh ghi này. Lệnh này có thể được chạy chỉ bởi các chương trình đang chạy ở mức đặc quyền cao nhất. Do đó trừ khi một nhiệm vụ đang hoạt động ở mức đặc quyền cao nhất, còn thì nó không thể thâm nhập bảng bộ mô tả cục bộ của nhiệm vụ khác được. Sự chuyển nhiệm vụ thường được điều khiển bởi phần lõi của hệ điều hành, phần này hoạt động ở mức ưu tiên cao nhất.

- ***Các mức đặc quyền***: 80386 khởi tạo 4 mức đặc quyền cho các nhiệm vụ được đánh số với thứ tự ưu tiên giảm dần từ 0 đến 3. Phần lõi hệ điều hành có mức đặc quyền cao nhất là bằng 0. Các dịch vụ hệ thống như các thủ tục trong BIOS ở mức 1 và các trình quản lí thiết bị hoạt động ở mức 2. Các chương trình ứng dụng và các nhiệm vụ của người dùng hoạt động ở mức đặc quyền thấp nhất là mức 3. Có thể hình dung việc gán cho các nhiệm vụ các mức đặc quyền như hình 3.11 sau. Mức đặc quyền của một đoạn nhớ được đại diện bởi các bit 5 và 6 của byte quyền thâm nhập trong bộ mô tả. Hai bit này được gọi là *mức đặc quyền bộ mô tả* DPL (descriptor previlege level). Mức này được thiết lập khi xây dựng chương trình. Mức đặc quyền của một nhiệm vụ đang thực thi được đại diện bởi các bit DPL trong byte quyền thâm nhập của bộ mô tả hiện trong cache bộ mô tả của thanh ghi đoạn mã CS. Nó được gọi là *mức đặc quyền hiện tại* CPL (curent privilede level). Khi một chương trình cần thâm nhập một đoạn số liệu nó

sẽ nạp bộ chọn đoạn vào phần hở của thanh ghi đoạn, thí dụ là DS. Mức đặc quyền được giải mã trong bít ít ý nghĩa nhất của bộ chọn này được gọi là mức đặc quyền yêu cầu RPL (requesting privilege level). Để thâm nhập thành công một đoạn, cả hai mức RPL và CPL phải nhỏ hơn hoặc bằng DPL của đoạn. Nói cách khác, mức đặc quyền của nhiệm vụ đang thực thi và mức đặc quyền của bộ chọn yêu cầu phải lớn hơn hoặc bằng mức đặc quyền của đoạn muốn truy nhập. Nếu điều kiện này không thoả mãn, một ngoại lệ sẽ xảy ra.



Hình 3.11 Bốn mức đặc quyền của các nhiệm vụ.

• ***Các cửa gọi:*** Thường thì một nhiệm vụ không thể thâm nhập một đoạn có mức đặc quyền cao hơn, thí dụ chương trình của người dùng không thể thâm nhập lõi của hệ điều hành, BIOS hoặc các thủ tục tiện ích khác. Nhưng có một cách gọi gián tiếp các thủ tục đó qua một cấu trúc đặc biệt gọi là *cửa* (gate). Có 4 loại cửa: cửa gọi, cửa bẫy, cửa ngắt và nhiệm vụ. Cửa đơn giản là một loại bộ mô tả đoạn đặc biệt. Bộ mô tả cửa được đặt trong GDT hoặc trong LDT. Khi một chương trình gọi một thủ tục trong một đoạn khác, bộ chọn cho cửa gọi đoạn đó được nạp vào thanh ghi CS và bộ mô tả cửa gọi được nạp vào phần kín của thanh ghi này. Bộ mô tả cửa gọi chứa một bộ chọn, bộ chọn này trỏ tới bộ mô tả đoạn nhớ của thủ tục. Bộ mô tả cửa gọi cũng chứa offset của thủ tục được gọi trong đoạn nhớ của nó. Nếu cửa được xác định là hợp lệ thì bộ chọn từ cửa gọi và bộ mô tả đoạn tương ứng sẽ được nạp vào thanh ghi CS. Bộ xử lí khi ấy sẽ sử dụng địa chỉ cơ sở từ bộ mô tả đoạn và giá trị offset từ bộ mô tả cửa gọi để tính địa chỉ vật lí của thủ tục cần gọi. Do vậy, cú gọi được thực hiện một cách gián tiếp qua bộ mô tả cửa gọi chứ không phải trực tiếp qua một bộ mô tả đoạn. Việc thâm nhập gián tiếp như vậy có 2 ưu điểm. Thứ nhất, nó cho phép kiểm tra mức đặc quyền khác trước khi việc thâm nhập tới một thủ tục trong đoạn nhớ có mức đặc quyền cao hơn được phép. Mức đặc quyền của chương trình gọi được so sánh với mức đặc quyền được định rõ trong cửa gọi. Nếu mức đặc quyền của chương trình gọi thấp hơn mức đặc quyền trong cửa gọi thì việc thâm nhập bị cấm. Thí dụ, nếu DPL trong bộ mô tả cửa là 2 thì một chương trình mức 2 có thể sử dụng cửa gọi để gọi một thủ tục có mức đặc quyền là 1 nhưng một chương trình ở mức 3 thì không thể. Một ưu điểm khác là chương trình sử dụng không thể tình cờ nhảy vào các đoạn có mức đặc quyền cao hơn. Nếu muốn chúng phải vào các offset nhất định được chứa trong các bộ mô tả cửa gọi. điều này tương tự như kiểu bảo vệ bằng cách sử dụng các ngắt mềm để gọi các dịch vụ của BIOS và DOS thay vì gọi chúng trực tiếp.

• ***Các mức đặc quyền vào/ra:*** Có hai cơ chế để bảo vệ các cổng vào/ra trong 80386. Cơ chế thứ nhất liên quan đến các bit mức đặc quyền vào/ra trong thanh ghi cờ của 386. Chỉ có hệ điều hành hoặc thủ tục hoạt động ở mức đặc quyền 0 mới có thể đặt các bit IOPL này. Để chạy các lệnh IN, INS, OUT, OUTS, CLI và STI, mức đặc quyền CPL của thủ tục hoặc nhiệm vụ phải bằng hoặc thấp hơn IOPL được đại diện bởi các bit này. Nếu không một ngoại lệ mức đặc quyền sẽ xảy ra. Cơ chế thứ hai là một ảnh bit (bit map) cho phép vào.ra tùy chọn. Bản đồ bit này cho phép các cổng chỉ được liên quan tới những nhiệm vụ xác định. Nếu đặc điểm này được sử dụng, một bản đồ bit sẽ được khởi tạo cho một nhiệm vụ. Mỗi bit trong bản đồ đại diện cho một địa chỉ cổng 1 byte, như vậy các cổng 16 bit sẽ dùng 2 bit và cổng 32 bit dùng 4 bit trong bản đồ bit. Số 0 trong bản đồ có nghĩa là cổng có sẵn cho nhiệm vụ. Khi một nhiệm vụ muốn thâm nhập một cổng, 386 sẽ so sánh CPL của nhiệm vụ với IOPL. Nếu kiểm tra cho qua và bản đồ bit vào/ra có hiệu lực thì 386 sẽ kiểm tra các bit tương ứng với địa chỉ cổng. Nếu bản đồ có giá trị 0 trong các bit thì việc thâm nhập là được phép. Ngược lại thì một ngoại lệ sẽ xảy ra. Tình cờ nếu 386 đang hoạt động trong chế độ thực thì không thể bảo vệ các cổng được.

• ***Quản lý các ngắt và ngoại lệ:*** Trong chế độ bảo vệ, các bộ mô tả cửa cho các thủ tục ngắt và ngoại lệ được giữ trong một bảng các bộ mô tả đặc biệt gọi là *bảng bộ mô tả ngắt* IDT (interrupt descriptor table). Bảng này có thể được nạp ở bất kì vùng nhớ nào có sẵn trong bộ nhớ. Trong quá trình khởi phát địa chỉ cơ sở và độ dài đoạn cho bảng bộ mô tả ngắt được nạp vào thanh ghi bảng bộ mô tả ngắt IDTR với lệnh nạp LIDT. Khi xảy ra một ngắt hoặc ngoại lệ, loại của nó (số ngắt hoặc số ngoại lệ) được nhân với 8 và cộng với địa chỉ cơ sở IDT trong thanh ghi IDT. Kết quả là một con trỏ tới bộ mô tả cửa trong bảng bộ mô tả ngắt. Cửa ở đây có thể là một cửa ngắt, cửa bẫy hoặc cửa nhiệm vụ. Thí dụ, cửa ngắt chứa một bộ chọn cho đoạn mà ở đó thủ tục ngắt được cất giữ chứ không phải địa chỉ cơ sở của đoạn. Lí do là ở chỗ, mức đặc quyền có thể được kiểm tra trước khi thâm nhập tới thủ tục ngắt được đảm bảo. Nếu CPL đủ cao thì bộ chọn từ cửa sẽ được nạp vào thanh ghi CS và được dùng để thâm nhập bộ mô tả cho đoạn chứa thủ tục ngắt. Bộ mô tả đoạn có thể trong LDT hoặc GDT. 32 bit offset từ cửa sẽ được cộng với địa chỉ cơ sở từ bộ mô tả để tạo ra địa chỉ tuyến tính cho thủ tục ngắt thực sự. Điều này cũng giống như cơ chế với cửa gọi chỉ có thêm vào là ở cuối thủ tục thì một lệnh IRET được dùng thay cho lệnh RET.

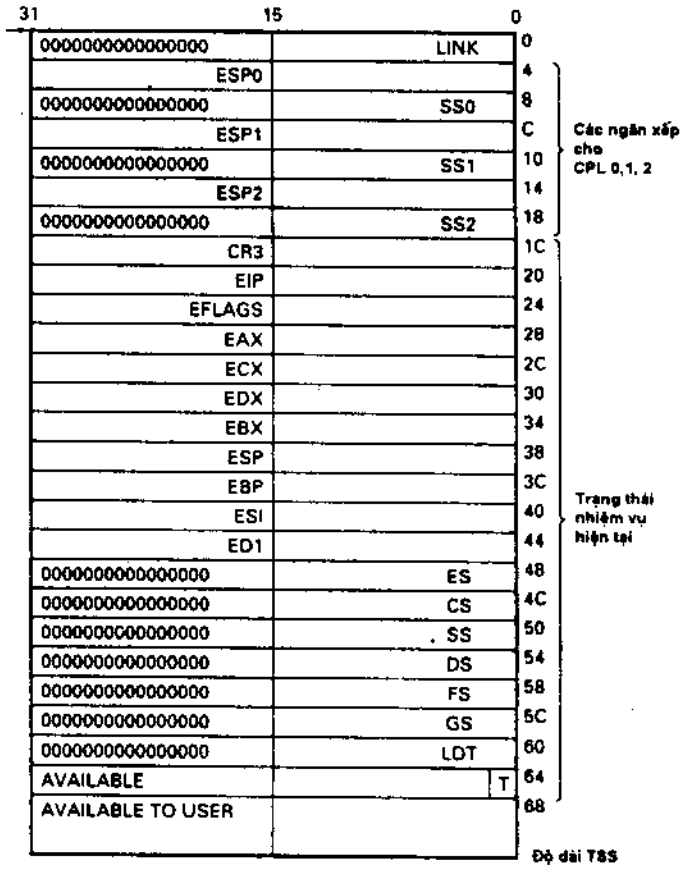
• ***Chuyển nhiệm vụ:*** Trong một hệ điều hành đa nhiệm, mỗi chương trình của người dùng có thể được khởi tạo như một nhiệm vụ tách biệt. Khi khoảng thời gian của người dùng tới, hệ điều hành sẽ chuyển việc thực thi từ nhiệm vụ đang hiện tại tới nhiệm vụ tiếp theo. Mỗi nhiệm vụ trong hệ thống 386 hoạt động ở chế độ bảo vệ được gán cho một *đoạn trạng thái nhiệm vụ* TSS (task state segment). Định dạng đoạn trạng thái này được chỉ ra trên hình 3.12. TSS giữ các bản copy của tất cả các thanh ghi và cờ, bộ chọn cho bảng mô tả cục bộ nhiệm vụ và một kết nối tới đoạn trạng thái nhiệm vụ của nhiệm vụ chạy trước đó. Bộ mô tả cho đoạn trạng thái nhiệm vụ được giữ trong bảng bộ mô tả toàn cục GDT cho phép hệ điều hành thâm nhập được trong suốt việc chuyển nhiệm vụ. Thanh ghi nhiệm vụ TR giữ bộ chọn và bộ mô tả cho đoạn trạng thái nhiệm vụ của nhiệm vụ hiện đang thực thi. Lệnh nạp thanh ghi nhiệm vụ LTR có thể được dùng để nạp thanh ghi nhiệm vụ với bộ chọn và bộ mô tả đoạn cho một nhiệm vụ xác định nhưng trong suốt quá trình chuyển nhiệm vụ, thanh ghi nhiệm vụ được nạp một cách tự động với bộ chọn và bộ mô tả cho nhiệm vụ mới.

Một sự chuyển nhiệm vụ có thể được thực hiện theo bất kì một trong bốn cách sau:

Một lệnh nhảy xa hoặc lệnh gọi chứa một bộ chọn trỏ tới một bộ mô tả đoạn trạng thái nhiệm vụ. Lệnh gọi được dùng nếu muốn trả về nhiệm vụ trước. Lệnh nhảy được dùng nếu không cần điều đó. đây là phương pháp đơn giản nhất và có thể được thực hiện dễ dàng bởi phần lõi hệ điều hành ở cuối của khoảng thời gian dành cho các nhiệm vụ.

1. Bộ chọn trong lệnh nhảy xa hoặc lệnh gọi sẽ trỏ tới một cửa nhiệm vụ. Trong trường hợp này, bộ chọn cho đích TSS ở trong cửa nhiệm vụ. Cơ chế gián tiếp ở đây tương tự như được mô tả ở trên với cửa gọi.

2. Một ngắt xảy ra và bộ chọn ngắt sẽ trỏ tới một cửa nhiệm vụ trong bảng bộ mô tả ngắt. Cửa nhiệm vụ chứa bộ chọn cho đoạn trạng thái nhiệm vụ mới. Nếu tất cả các kiểm tra mức đặc quyền là đạt thì bộ chọn và bộ mô tả cho nhiệm vụ ngắt sẽ được nạp vào thanh ghi nhiệm vụ. Bit nhiệm vụ lồng ghép (NT) trong thanh ghi cờ sẽ được đặt.



Hình 3.12 Định dạng đoạn trạng thái nhiệm vụ

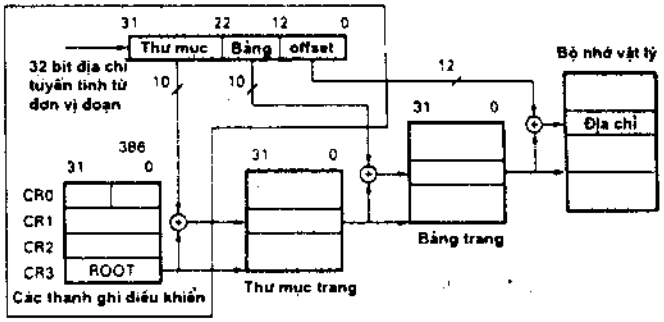
3. Lệnh IRET được thực hiện với bit NT trong thanh ghi cờ được đặt. Các thủ tục ngắt phức tạp thường được viết và quản lí thành các nhiệm vụ tách biệt nhau. Lệnh IRET sử dụng bộ chọn liên kết ngược trong đoạn trạng thái nhiệm vụ để trả về nhiệm vụ bị ngắt. Điều này tương tự như cách làm việc trong chế độ thực của lệnh IRET.

• ***Chế độ quản lí theo trang:*** Các nhà thiết kế 80386 nhận thấy rằng thời gian đòi hỏi để hoán chuyển các đoạn rất lớn vào và ra bộ nhớ vật lí trở nên rất lâu. Do vậy trong 386 cho phép tuỳ chọn một cơ chế quản lí bộ nhớ theo trang cho phép các đoạn được chia thành các trang chỉ dài 4 kbyte nhằm làm cho việc hoán chuyển nhanh hơn.

80386 chuển sang chế độ trang bằng việc đặt bit MSB của thanh ghi CR0 với lệnh đơn giản MOV CR0, EAX. Trong chế độ này, đơn vị quản lí trang sẽ dùng địa chỉ tuyến tính, đã được tính bởi đơn vị quản lí đoạn, để định ra địa chỉ vật lí. Hình 3.13 chỉ ra cách làm việc của sơ đồ quản lí theo trang.

Mỗi trang trong bộ nhớ vật lí được gọi là các khung trang (page frame). 12 bit ít ý nghĩa nhất của địa chỉ tuyến tính đại diện cho offset của từ số liệu cần truy nhập tróng số 4 kbyte của một khung trang. 10 bit địa chỉ từ A12 đến A21 được dùng để chon điểm vào tới một trong các bảng trang. Như vậy sẽ có 1024 bảng các trang. Các bảng này lại được giữ trong các bảng khác gọi là thư mục trang. 10 bit cao từ A22 đến A31 của địa chỉ tuyến tính được dùng để chọn

điểm vào mong muốn trong thư mục trang. Địa chỉ cơ sở 32 bit cho thư mục trang được giữ trong thanh ghi điều khiển CR3 của 80386.



Hình 3.13 Tính địa chỉ vật lí trong chế độ trang.

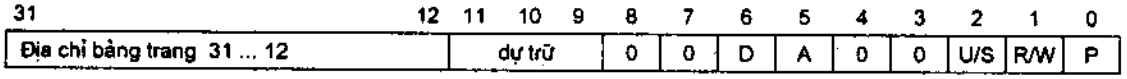
CR3 trỏ tới địa chỉ cơ sở của thư mục trang và các bit địa chỉ tuyến tính A22 đến A31 trỏ tới một trong 1024 điểm vào trong thư mục . Điểm vào được chọn trong thư mục trỏ tới địa chỉ cơ sở của một trong 1024 bảng trang và các bit địa chỉ tuyến tính A12 đến A21 trỏ tới một trong các điểm vào của bảng trang được chọn. Điểm vào được chọn đó chứa địa chỉ cơ sở 32 bit của khung trang mong muốn. Các bit địa chỉ tuyến tính A0 đến A11 được dùng để thâm nhập từ mã lệnh hoặc số liệu trong khung trang được chọn. Các bit này được cộng với địa chỉ cơ sở từ điểm vào bảng trang để tạo ra địa chỉ vật lí cần truy xuất ở bộ nhớ. Dung lượng cực đại của bộ nhớ trong cấu trúc này là:

1024 bảng trang × 1024 trang/ một bảng trang × 4096 byte / trang = 4 Gbyte.

Một hệ thống có thể được khởi tạo với một thư mục trang nhưng thông thường hay trao cho mỗi nhiệm vụ một trang riêng và do vậy nó cũng có một tập các bảng trang riêng. Do thư mục và bảng trang đều định xứ trong bộ nhớ nên để tránh phải đọc các điểm vào thư mục trang và các điểm vào bảng trang từ các bảng nhớ khi thâm nhập bộ nhớ, 386 duy trì một vùng nhớ cache đặc biệt gọi là bộ đệm chuyển hoá riêng TLB (translation lookaside buffer). Đó là một loại cache liên hợp 4 đường giữ các điểm vào bảng trang cho 32 trang vừa được dùng mới đây nhất. Khi 386 phát ra một địa chỉ tuyến tính, 20 bit trên được so sánh với các thẻ cho 32 điểm vào trong TLB. Nếu trùng nhau thì có nghĩa là điểm vào bảng trang cần truy xuất nằm trong TLB. Địa chỉ cơ sở từ điểm vào này được dùng để tính địa chỉ vật lí. Nếu không trùng nhau, 386 sẽ đọc điểm vào bảng trang từ bộ nhớ và tự nó vào TLB. Nếu bit P trong điểm vào bảng trang là 1, chỉ thị rằng trang có tồn tại trong bộ nhớ vật lí, thì địa chỉ vật lí sẽ được tính toán và từ mong muốn trong trang sẽ được truy cập. Nếu bit P trong điểm vào bảng trang bằng 0, chỉ thị rằng trang không hiện diện trong bộ nhớ vật lí, bộ xử lí sẽ phát ra một ngoại lệ lỗi trang (số 14). Sau khi chương trình ngoại lệ này hoán chuyển trang vào bộ nhớ vật lí, đơn vị quản lí trang sẽ tính và xuất ra địa chỉ vật lí cho từ mong muốn.

Khi chế độ trang được phép, các bit U/S và R/W trong các điểm vào thư mục trang và bảng trang có thể được dùng như một cơ chế bảo vệ đoạn. Bit U/S trong điểm vào thư mục hoặc bảng trang được dùng để phân biệt một trong hai mức đặc quyền là người dùng hoặc người giám quản. Nếu U/S bằng 0 là mức đặc quyền người dùng tương đương với mức 3 trong chế độ đoạn. Ngược lại khi U/S bằng 1 là mức đặc quyền giám quản tương ứng với các mức 0, 1 và 2.

Bit R/W có thể được dùng để thiết lập các quyền thâm nhập đọc/viết cho các trang hoặc các bảng trang. Hình 3.14 sau chỉ ra các mức đặc quyền được tạo bởi sự biến đổi các tổ hợp U/S và R/W. Lưu ý rằng 11 biểu diễn đặc quyền nhất và 00 biểu diễn ít đặc quyền nhất. Nếu các quyền thâm nhập được cho trong điểm vào thư mục trang khác với các quyền thâm nhập được cho trong điểm vào bảng trang , mức đặc quyền ít nhất sẽ quyết định các quyền thâm nhập.



U/$\overline{S}$	R/$\overline{W}$	Mức được phép 3	Các mức thâm nhập được phép 0, 1 hoặc 2
0	0	không	đọc/viết
0	1	không	đọc/viết
1	0	chỉ đọc	đọc/viết
1	1	đọc/viết	đọc/viết

Hình 3.14 Định dạng cho các điểm vào thư mục và bảng trang (a) và các quyền thâm nhập được tạo ra bởi tổ hợp các bit U/S và R/W trong các điểm vào bảng trang.

3. Chế độ hoạt động 8086 ảo: 80386 dễ dàng chuyển tới chế độ 8086 ảo để chạy trong một khoảng thời gian dành cho chương trình loại 8086 và rồi dễ dàng quay về chế độ bảo vệ để chạy trong khoảng thời gian dành cho nhiệm vụ hoạt động trong chế độ bảo vệ Điều này có nghĩa rằng một vài người dùng trong hệ thống đa người dùng có thể chạy các chương trình trong chế độ bảo vệ UNIX và các người dùng khác có thể đang chạy các chương trình DOS trong chế độ thực. Khi 386 đang chạy trong chế độ bảo vệ thực hiện việc chuyển nhiệm vụ, nó kiểm tra bit VM trong thanh ghi cờ. Nếu bit này được đặt, 386 sẽ nhảy vào chế độ 8086 ảo để thực thi nhiệm vụ mới. Nếu bit VM không được đặt, 386 sẽ thực thi nhiệm vụ mới như một nhiệm vụ trong chế độ bảo vệ bình thường.

Trong chế độ 8086 ảo, vi xử lí 386 sẽ tính địa chỉ vật lí bằng việc sử dụng cơ chế *segment : offset* như trong 8086. Do đó dải địa chỉ là 1 Mbyte nhớ. Với một nhiệm vụ 8086 ảo, dải địa chỉ này nằm ở dải 1 Mbyte thấp nhất trong không gian địa chỉ của bộ xử lí. Nếu hệ thống cần chạy một vài nhiệm vụ 8086 ảo khác nhau thì 386 hoạt động trong chế độ trang nhằm cho mỗi nhiệm vụ 8086 được trao cho một bảng trang và một tập các trang khác nhau trong bộ nhớ vật lí. Lợi ích của phương pháp này là các bit U/S và R/W cung cấp sự bảo vệ mà trong chế độ thực thông thường không có được.

Để chạy các nhiệm vụ 8086 chế độ ảo hệ điều hành phải có một phần mã mức đặc quyền 0 được gọi là trình giám sát máy ảo (virtual machine monitor). Mục đích chính của nó là chặn các ngắt, các ngoại lệ và các lệnh INT xảy ra khi đang chạy nhiệm vụ 8086. Biết rằng hầu hết các chương trình hệ thống 8086 sử dụng các ngắt mềm INT để thâm nhập các thủ tục vào/ra BIOS và DOS. Trong chế độ 8086 ảo, lệnh INT chỉ được chạy ở mức đặc quyền 0 là mức cao nhất. Vì rằng nhiệm vụ 8086 luôn hoạt động ở mức 3, mức thấp nhất, nên 386 sẽ phát ra một ngoại lệ bất cứ khi nào chương trình 8086 định chạy lệnh INT. Chương trình điều khiển ngoại lệ này được đặt trong trình giám sát máy ảo, như vậy sự kiểm tra sẽ có hiệu quả ở điểm này. Trong quá trình chuyển nhiệm vụ tới trình giám sát, trạng thái của nhiệm vụ 8086 được cất trong thanh ghi TSS của nó. Bit VM trong thanh ghi cờ được xoá nhằm cho trình giám sát có thể hoạt động trong chế độ bảo vệ bình thường. Nếu một lời gọi một dịch vụ chẳng hạn như lệnh

"mở file" trong DOS, trình giám sát sẽ gọi thủ tục tương đương trong hệ điều hành 80386 ở chế độ bảo vệ để mở file. Cơ chế này duy trì tất cả sự bảo vệ được xây dựng bên trong hệ điều hành chính của 386. Khi file đã được mở, việc chạy chương trình được trả về hệ điều hành DOS. Lệnh IRET được dùng để trả về chương trình DOS 8086 ảo khôi phục lại trạng thái nhiệm vụ 8086. Bit VM trong thanh ghi cờ được khôi phục về 1 nhằm cho chương trình 8086 được khởi phát lại trong chế độ 8086 ảo. Đối với các lời gọi dịch vụ DOS không liên quan tới hoạt động vào/ra, trình giám sát có thể trả việc chạy chương trình cho DOS để thực hiện. Sau khi dịch vụ được thực hiện, DOS sẽ hoàn trả việc chạy chương trình cho chương trình 8086.

Trong chế độ 8086 ảo, các ngắt cũng bị chặn bởi trình giám sát. Trong hầu hết trường hợp, trình giám sát sẽ chuyển việc chạy chương trình tới hệ điều hành 386 hoạt động ở chế độ bảo vệ để phục vụ ngắt. Hệ điều hành 386 sẽ dùng bảng mô tả ngắt và sơ đồ cửa nhằm duy trì sự bảo vệ. Nếu không yêu cầu bảo vệ, trình giám sát có thể trả việc chạy chương trình tới DOS hoặc tới chương trình 8086 để phục vụ ngắt.

Khi một ngắt đồng hồ xảy ra báo kết thúc khoảng thời gian, chương trình sẽ chuyển từ nhiệm vụ 8086 tới nhiệm vụ trình giám sát. Thông qua cửa IDT, trình giám sát sẽ chuyển tới lịch biểu hệ điều hành 80386. Lịch biểu lúc đó sẽ chuyển tới nhiệm vụ người dùng tiếp theo. Bit VM trong ảnh thanh ghi cờ của TSS cho nhiệm vụ mới sẽ xác định liệu nhiệm vụ được chạy trong chế độ 8086 ảo hoặc chế độ 386 bảo vệ. ở đây 386 cung cấp một cơ chế tương đối đơn giản để thay đổi giữa các chương trình loại 8086 và các chương trình loại 386 chế độ bảo vệ.

3.2.5. Vi xử lí 80486

Chip 486 có mật độ tích hợp rất cao bao gồm bên trong một vi xử lí 386 đã cải tiến, một bộ đồng xử lí toán 80387, một bộ điều khiển cache và một bộ nhớ cache tốc độ cao có dung lượng 8 Kbyte. Giống như 386, bus số liệu và bus địa chỉ ở đây cũng rộng 32 bit, tốc độ truyền tải dữ liệu có thể lên tới 160 Mbyte/s. Đây là thiết kế dựa trên kết quả của việc tích hợp hai loại phần tử xử lí CISC và RISC trong cùng một chip CPU. Trong 486, các lệnh mã máy thường xuyên được sử dụng sẽ được cài đặt sẵn và có thể được thực hiện ngay, trong khi các lệnh phức tạp nhất và ít được sử dụng sẽ thông qua bộ giải mã vị lệnh. Theo phương thức này, hầu như tất cả các lệnh có thể được thực hiện gọn trong một chu kì xung nhịp đồng hồ.

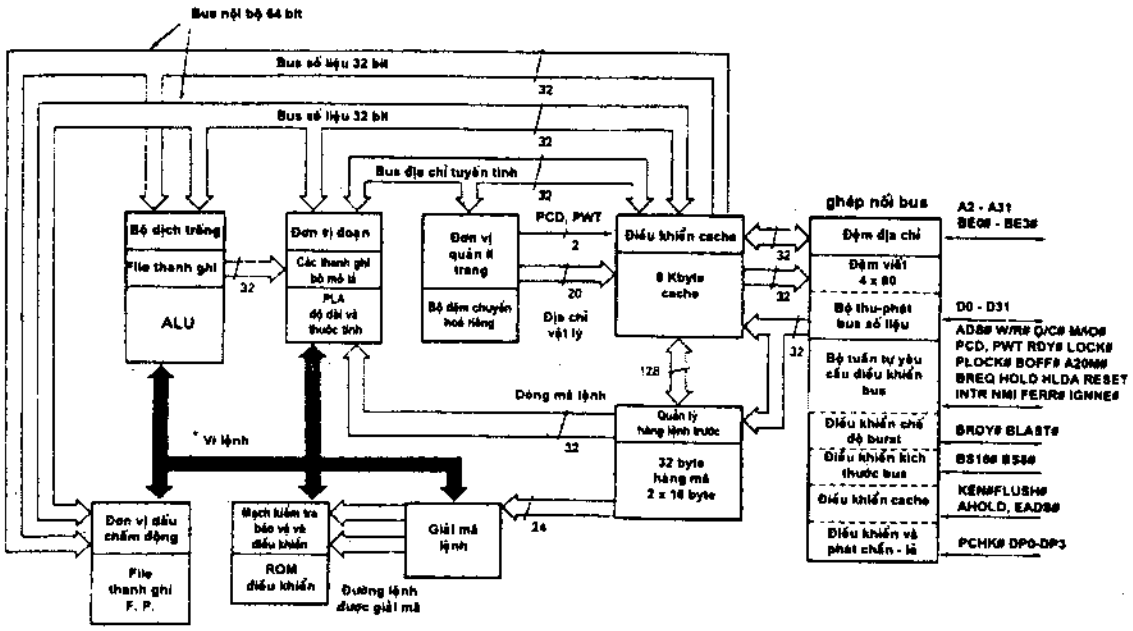
Vi xử lí i486 hoàn toàn tương thích với i386 về tập lệnh và các loại số liệu. Nó cũng có thể chạy trong các chế độ thực, bảo vệ cũng như phân đoạn và phân trang. Sơ đồ khối bên trong của 80486 theo hãng Intel được chỉ ra như hình 3.15.

Một trong những đặc điểm quan trọng là trong 486 được tích hợp luôn bộ đồng xử lí toán. Về nguyên tắc nó cũng giống như chip 80387 cho vi xử lí 386 nhưng vì được tích hợp ngay trên cùng một chip nên cho phép chạy các lệnh toán nhanh hơn trong một hệ gồm 386 và 387 tách rời bình thường chừng 3 lần.

Mạch cache liên hợp 4 đường trong 486 làm việc giống như cache ngoại nhưng đường cache ở đây là 16 byte chứ không phải 4 byte.

Sơ đồ còn cho thấy ở đây có đường ống lệnh 5 tầng cho phép chạy các lệnh nhanh hơn ở 386 nhiều. Sơ đồ này thường dùng trong vi xử lí RICS cho phép một vài lệnh có thể "chảy" trong đường ống cùng một lúc. 486 sẽ nhận một vài lệnh trước và trong khi đang chạy một lệnh nó sẽ giải mã và khởi phát chạy lệnh tiếp theo sớm đến mức có thể được. Thực sự 486 có thể thực thi các phần của một vài lệnh trong cùng một thời gian. Thí dụ, giả thử 486 phải thực hiện chuỗi lệnh có ý nghĩa như sau:

MOV AX, nội dung vùng nhớ
ADD CX, BX
SHR AX,1
MOV nội dung vùng nhớ , CX



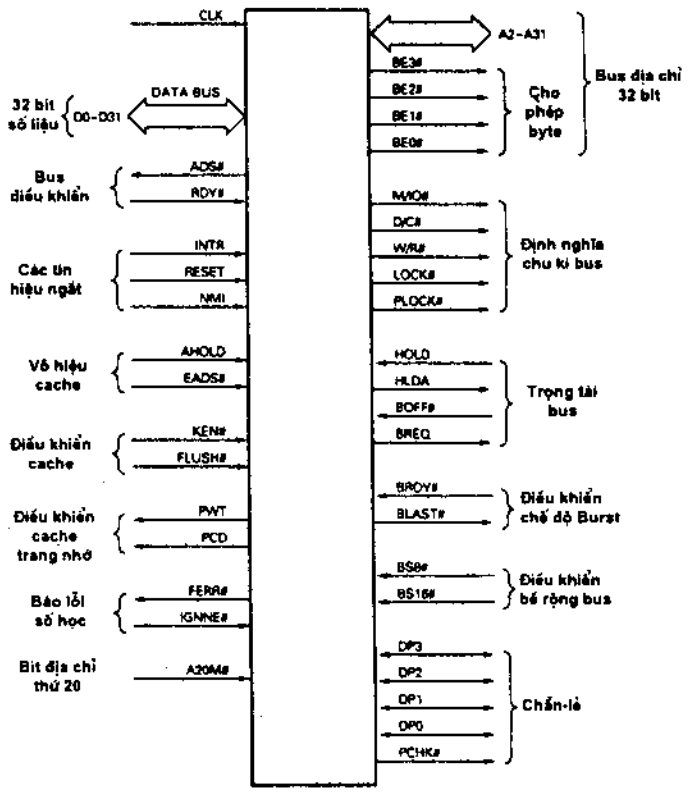
Hình 3.15 Sơ đồ khối của vi xử lí 80486.

Một vài nhịp đồng hồ trước khi thực hiện lệnh thứ nhất, 486 đã lấy tất cả các lệnh đơn giản và khởi phát giải mã chúng. Khi lệnh đầu tiên được thực hiện thì việc giải mã lệnh thứ hai đã hoàn thành. Vì lệnh ADD không dùng các bus hoặc số liệu đọc từ bộ nhớ bởi lệnh MOV nên nó có thể được thực hiện trước khi lệnh MOV hoàn thành. Giống như vậy, sự giải mã lệnh SHR sẽ được hoàn thành trong khi lệnh ADD CX, BX đang chạy; và trong chu kỳ nhịp tiếp theo lệnh SHR sẽ được thực hiện. Trong chu kỳ nhịp mà lệnh SHR được thực hiện, sự giải mã lệnh cuối cùng được hoàn thành và địa chỉ của vùng nhớ sẽ được đưa ra bus địa chỉ. Trong chu kì đồng hồ tiếp theo, từ trong thanh ghi AX sẽ được đưa ra trên bus số liệu. Việc tăng cường sự xen phủ như vậy cho phép 486 chạy nhiều lệnh trong một chu kì đồng hồ một cách hiệu dụng. Việc nhận lệnh, giải mã lệnh và thực hiện mỗi lệnh thực tế xảy ra vài nhịp đồng hồ nhưng vì các hoạt động này xen phủ lên nhau với các quá trình giải mã và thực hiện lệnh nên thời gian thực cho mỗi lệnh chỉ mất có một nhịp đồng hồ.

Thí dụ, một hoạt động viết bộ nhớ 16 bit mất chừng 22 nhịp đồng hồ trong vi xử lí 8088 và 4 nhịp trong 386 thì chỉ diễn ra 1 nhịp trong 80486. Các lệnh nhảy có điều kiện cũng gặp thuận lợi trong cơ chế đường ống của 486. Khi 486 giải mã một lệnh nhảy có điều kiện, nó tự động nhận một hoặc nhiều lệnh từ địa chỉ đích ngay trong trường hợp cú nhảy xảy ra. Nếu sự rẽ nhánh được thực hiện thì 486 không phải đợi qua một chu kỳ bus cho lệnh đầu tiên ở địa chỉ rẽ nhánh. Một lệnh nhảy có điều kiện cần chừng 16/4 chu kì nhịp ở 8088 và 8/3 chu kì nhịp ở 386 thì chỉ mất chừng 3/1 chu kì nhịp trong 486.

Hầu hết những cải tiến trong 486 liên quan đến các tín hiệu và ghép nối phần cứng. Hình 3.16 là các nhóm chân tín hiệu của 486.

Một nhóm chân tín hiệu mới trong 486 là nhóm chẵn lẻ CP0 đến DP3 và PCHK#. Các tín hiệu này cho phép 486 thực hiện việc phát và kiểm tra chẵn lẻ cho việc đọc/viết bộ nhớ. Nhóm tín hiệu mới khác là tín hiệu sẵn sàng truyền khối (burst mode) BRDY#, và tín hiệu kết thúc khối BLAST#. Các tín hiệu này được dùng để điều khiển chế độ đọc/viết một khối dữ liệu từ/ đến bộ nhớ. Trong chế độ bình thường, 1 chu kì bus không có trạng thái đợi chiếm mất 2 chu kì đồng hồ giống như trong 80386. Trong chế độ burst, thời gian cần thiết cho việc chuyển tải dữ liệu như vậy giảm xuống còn 1 chu kì nhịp đồng hồ. Chế độ burst được bắt đầu bằng việc thâm nhập bình thường bộ nhớ và kết thúc trong 2 nhịp đồng hồ. Trong chu kì nhịp thứ hai, 80486 nâng tín hiệu trong chân *BLAST* lên mức cao để chỉ thị rằng nó muốn chạy chế độ truyền dữ liệu kiểu burst. Nếu hệ thống được định địa chỉ có khả năng thực hiện một chu kì burst thì nó hạ dây *BRDY* xuống mức tích cực thấp. Từ đó tất cả các chu kì bus chỉ mất một nhịp đồng hồ. Để kết thúc chu kì burst, 486 hạ tín hiệu *BLAST* xuống mức tích cực thấp để báo rằng giá trị cuối cùng của chu trình burst hiện tại đã được truyền với tín hiệu *BRDY* hoặc *RDY* tiếp theo. Chú ý rằng các số liệu được truyền trong một chu trình burst phải nằm trong một vùng 16 byte tức là địa chỉ của chúng phải nằm trong dải xxxxxxx0h tới xxxxxxxfh (xxxxxxx = 0000000h tới fffffffh). Điều này phù hợp với một hàng cache trong bộ nhớ cache nội của vi xử lí.



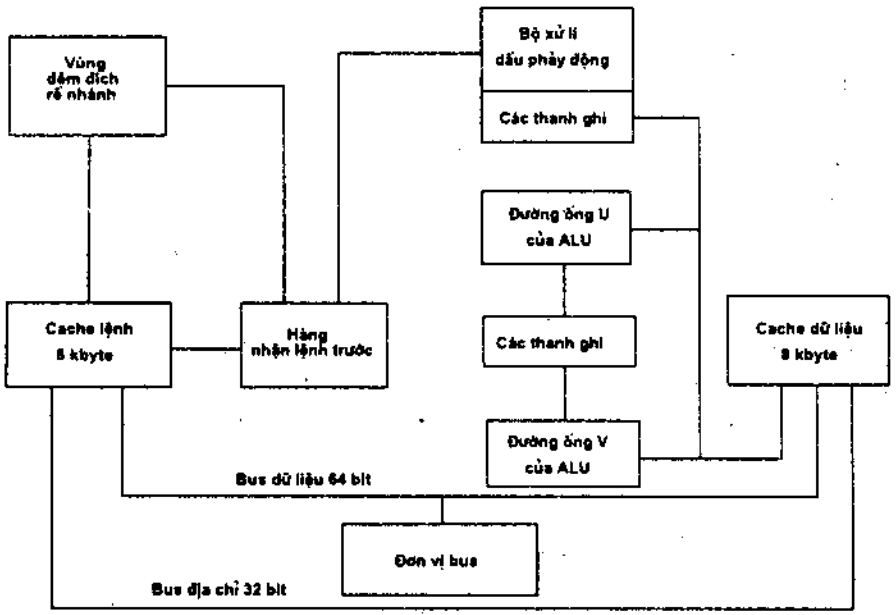
Hình 3.16 Các nhóm chân tín hiệu của 80486.

Với vi xử lí 486 DX2 hoặc DX4 , *tần số nhịp được cấp cho CPU sẽ được nhân 2 hoặc 3 ở bên trong chip* cho phép có thể tăng tốc độ của vi xử lí mà không đòi hỏi bản mạch chính cũng phải có cùng tốc độ đó. Điều này làm giảm nhẹ việc thiết kế chế tạo các bản mạch chính phải chạy ở tần số cao, do đó giá thành sản phẩm cũng được giảm theo. Thí dụ, vi xử lí i486 DX2/50MHz chỉ cần chạy với bản mạch chính 25MHz, DX2/66MHz hay DX4/100MHz với bản mạch chính 33MHz. Theo phương án này, nhìn chung máy vi tính phải chịu thiệt về hiệu suất, vì CPU tiến hành xử lí số liệu nhanh gấp hai hoặc ba lần so với các bộ phận trên bản mạch chính, do đó có thể phải xen vào một vài chu kì đợi (wait cycle). Để giải quyết vấn đề này, một bộ nhớ cache ngoài đủ rộng để giữ tạm các lệnh và dữ liệu mà bộ vi xử lí phải đợi. Nếu được thiết kế hợp lí, hệ thống như vậy có thể đạt tới hiệu suất 80% so với các hệ thống có tốc độ xung nhịp trên bản mạch chính bằng tốc độ xung nhịp ở trong CPU.

3.2. 6. Vi xử lí Pentium

Vi xử lí Pentium không chỉ nhanh hơn 80486 mà còn có một số đặc điểm mới được thiết kế đặc biệt cho các ứng dụng phục vụ file. Đó là bus dữ liệu rộng 64 bit tuy vẫn chỉ có các thanh ghi 32 bit bên trong. Chip có khả năng thực hiện 2 lệnh song song theo công nghệ siêu vô hướng (superscalar) với 2 đường ống U và V. Công nghệ này có khả năng làm việc với các cấu trúc RISC. Chip có khả năng định địa chỉ bổ xung, chỉ tiêu được cải thiện với các phép tính dấu chấm động và các bộ nhớ cache nội lớn và mềm dẻo hơn. Bên trong chip có 2 bộ nhớ cache nội 8 Kbyte, một cho lệnh và một cho dữ liệu.

Sơ đồ khối của vi xử lí Pentium có thể được trình bày như hình 3.17 gồm:



Hình 3.17 Sơ đồ khối của vi xử lí Pentium.

- Hai bộ nhớ cache 8 kbyte riêng biệt, một dành cho lệnh và một dành cho số liệu.
- Hai hàng nhận lệnh trước 32 bit và hai bộ tính số học và lôgic ALU được cung cấp lệnh và dữ liệu trên hai đường ống song song U và V (superscalar) cho phép thi hành hai lệnh máy trong một chu kì.
- Bộ xử lí vẫn làm việc với các thanh ghi 32 bit và bus địa chỉ 32 bit nhưng có thể nối với bus số liệu ngoài là 64 bit.
- Bus nội bộ là 64 bit và 128 bit, bus từ bộ nhớ cache nội nối với các vùng đệm của các ALU rộng 256 bit cho phép tăng tốc độ trao đổi dữ liệu trong nội bộ vi xử lí lên rất cao.
- Có một vùng nhớ gọi là vùng đệm đích rẽ nhánh BTB (branch target buffer) của lệnh đối với 256 lệnh rẽ nhánh mới đây nhất.
- Có một bộ đồng xử lí với hiệu suất cao hơn nhiều lần các thế hệ trước nhờ các giải thuật nhanh hơn, cách sắp xếp bố trí lệnh cũng như có thể thực hiện đồng thời hai lệnh.

Một vài đặc trưng khác làm cho Pentium phân biệt được với các vi xử lí trước là:

Kích thước trang của bộ nhớ trong chế độ bảo vệ và ảo không chỉ hạn chế ở 4 kbyte mà có thể lên đến 2Mbyte hay 4Mbyte. Khả năng tự tìm kiếm các lỗi phức tạp và gỡ rối (debugging) bằng các mạch phần cứng lắp thêm. Có thể tự đánh giá được hiệu suất thi hành chương trình. Mặc dù là một vi xử lí được xây dựng trên nguyên tắc xử lí CISC nhưng Pentium đã ứng dụng

nhiều công nghệ mới giống như các loại vi xử lí RISC tốc độ cao, thí dụ như việc sử dụng các cấu trúc đường ống, cấu trúc superscalar và dự đoán rẽ nhánh. Điều này làm cho nó có hiệu suất cao như các vi xử lí RISC thực sự. Các lệnh chạy trong Pentium được thực hiện trên một cấu trúc superscalar với hai đường ống song song U và V, các cache nội bộ và vùng đệm đích rẽ nhánh BTB. Xét qua các cấu trúc này sẽ cho phép giải thích tại sao Pentium lại có hiệu suất cao hơn so với các vi xử lí trước đó:

Cấu trúc superscalar với hai đường ống song song: Việc thi hành lệnh trong các vi xử lí 80x86 và Pentium được diễn ra qua 5 công đoạn:

1. Nhận lệnh trước (*Prefetch*): lệnh máy được lấy từ bộ nhớ chuyển vào CPU.
2. Giải mã lệnh 1 (*Decode 1*): giải mã, phân tích để xác định loại lệnh.
3. Giải mã lệnh 2 (*Decode 2*): xác định địa chỉ ô nhớ cần truy nhập.
4. Thực hiện lệnh (*Execute*).
5. Viết các kết quả dữ liệu vào các thanh ghi và bộ nhớ (*Writeback*).

Nếu như mỗi công đoạn đòi hỏi tối thiểu một chu kì nhịp đồng hồ thì khi thi hành tuần tự các công đoạn này cho mỗi lệnh, thông thường phải mất tổng cộng năm chu kì nhịp. Nhưng với một cấu trúc đường ống có thể thực hiện đồng thời song song những công đoạn trên, Pentium tạo ra một phương thức làm việc như sau: khi lệnh thứ (n) vừa chuyển sang công đoạn thứ hai thì một lệnh mới thứ (n+1) được đưa ngay vào công đoạn thứ nhất v.v... Nhờ đó cả thảy có 5 lệnh sẽ cùng được xử lí đồng thời trên một đường ống và có thể coi mỗi lệnh chỉ được xử lí trong một chu kì nhịp đồng hồ. Hơn thế nữa, Pentium lại có đến hai đường ống gọi tên là U và V được chế tạo hoạt động song song với nhau. Về mặt lí thuyết, nó sẽ có khả năng thi hành cùng một lúc hai lệnh. Tuy nhiên, thực tế không phải lúc nào cũng thực hiện được điều đó vì hai lệnh kế tiếp nhau thường phụ thuộc vào nhau và lúc này chỉ có thể xử lí chúng một cách tuần tự. Một mạch đặc biệt sẽ kiểm tra tính phụ thuộc dữ liệu đó xem chúng có yêu cầu phải liên kết trong quá trình xử lí hay không. Nếu không phụ thuộc, hai lệnh sẽ được gửi đến hai đường ống theo đúng thứ tự. Quá trình có thể dẫn giải như sau: trong công đoạn một, lệnh hiện tại sẽ được đưa vào đơn vị giải mã của đường ống U, lệnh kế tiếp vào đơn vị giải mã của đường ống V. Nếu trong công đoạn hai, mạch đặc biệt nói trên xác định có thể thực hiện đồng thời được hai lệnh thì mỗi lệnh sẽ trải qua các công đoạn còn lại trong mỗi đường ống của nó, ngược lại thì lệnh trong ống U sẽ trải qua công đoạn kế tiếp như bình thường còn lệnh kế tiếp trong ống V sẽ được chuyển qua thi hành trong ống U như là lệnh theo sau lệnh hiện tại.

Bộ nhớ cache nội bộ: Pentium có hai bộ nhớ cache riêng biệt, một dành cho số liệu và một dành cho lệnh. Do có tốc độ nhanh lại được được tích hợp ngay trong chip vi xử lí nên các lệnh và dữ liệu ở đây có thể được truy cập rất nhanh. Mỗi cache có dung lượng 8kbyte gồm hai tuyến, 128 hàng cache, mỗi hàng dài 32 byte. Cả hai cache này có thể được truy xuất đồng thời từ hai đường ống U và V nhờ vào cấu trúc đan xen gấp tám (trang).

Pentium có thể hoạt động nhịp nhàng với các vi xử lí khác trong chế độ đa xử lí (mỗi vi xử lí có một cache nội riêng) nhờ được cấp một giao thức đặc biệt gọi là MESI (Modified, Exclusive, Shared, Invalid). Nó cho phép hỗ trợ việc đồng bộ hoá giữa các cache trong một hệ thống đa xử lí. Giao thức có chứa thủ tục *bus snooping* cho phép một bộ xử lí thăm dò và xử lí các thông tin được trữ trong các bộ nhớ cache của các bộ xử lí khác trong hệ thống.

Vùng đệm đích rẽ nhánh BTB: Xét trường hợp khi CPU thực hiện một lệnh nhảy (JUMP). Lúc này, thay vì tiếp tục với lệnh kế tiếp trong bộ nhớ thì việc thi hành chương trình lại được tiếp tục với một lệnh hoàn toàn khác tại địa chỉ đích của lệnh nhảy. Điều đó dẫn tới việc phải "tống" lệnh kế tiếp trên, vốn đã nằm sẵn trong đường ống, ra ngoài và phải nạp một lệnh mới

vào đường ống. Kết quả là hiệu suất của bộ xử lí sẽ bị giảm xuống vì phải mất đi một vài chu kì nhịp đồng hồ cho quá trình trên. Để khắc phục tình trạng này, Pentium có một vùng nhớ đệm gọi là vùng đệm đích rẽ nhánh BTB (nghĩa là vùng đệm dành cho các lệnh nhảy - IF/THEN). Nó được sử dụng trong công đoạn giải mã lệnh 1 (Decode 1) của tất cả các lệnh nhảy gần (NEAR JUMP). Nếu lệnh nhảy được thi hành, CPU sẽ cất cả địa chỉ lệnh nhảy và địa chỉ đích của lệnh vào BTB. Khi thực hiện lệnh nhảy, thay vì nạp lệnh kế tiếp theo sau lệnh nhảy vào đường ống, CPU sẽ nạp lệnh ở địa chỉ đích của lệnh nhảy. Còn nếu lệnh nhảy chưa được thi hành, tức là các địa chỉ lệnh nhảy và địa chỉ đích của nó chưa được đăng kí trong BTB thì lệnh kế tiếp theo sau lệnh nhảy sẽ được nạp vào đường ống như bình thường.

Phần mềm của Pentium tương thích với những phần mềm của 80386 và 80486 nhưng có thời gian thực hiện lệnh nhanh hơn nhiều. Với khoảng 3.1 triệu transistor ở phiên bản đầu thế hệ 1, Pentium có thể chạy với các tần số 60 MHz và tốc độ xử lí là 100 MIPS với tốc độ bus trên bản mạch chính là 66 MHz. Trong khi thế hệ Pentium đầu tiên được gọi là P5 đang được sử dụng thì Intel đã đưa ra thế hệ tiếp sau, Intel P6, với tên gọi là *Pentium Pro*. Các Pentium thế hệ 2 có các phiên bản 75/100/120/133/150/166/200 MHz. Trên cơ sở công nghệ 0,4micro với 250 MIPS, nó có thiết kế đổi mới và tốc độ xử lí nhanh hơn nhưng lại giữ được tính tương thích hoàn toàn với tập lệnh của 386 và 486. Điều đó cho phép vẫn sử dụng được một khối lượng đồ sộ các phần mềm DOS và Microsoft Windows trên thế giới. Chip P6 thuộc loại superscalar và superpipelining. Thay vì cho 5 công đoạn là 7 công đoạn mới trong đường ống và nó có khả năng xử lí 3 chứ không phải là 2 lệnh đồng thời như P5. Khác với Pentium có thiết kế CISC, P6 được chế tạo theo công nghệ RISC nhưng sử dụng các mạch thông dịch gắn trên bản mạch chính để chuyển đổi các lệnh của 486 thành các lệnh RISC. Pentium Pro dùng phương pháp thực hiện theo suy đoán (speculative execution) để tối ưu hoá quá trình xử lí, đó là phương pháp lưu trữ và phân tích trên 30 lệnh trước khi chúng được thực hiện. Các lệnh này đều được dự đoán là chúng sắp đi qua bộ xử lí nên được hướng dẫn và sắp xếp thứ tự thích hợp để tối thiểu hoá thời gian xử lí. Đồng thời cũng nhờ phương pháp suy đoán này mà P6 ít gặp trường hợp phải "nhốt" lệnh trong đường ống khi có hai lệnh yêu cầu phải được hoàn thành trong cùng một lúc, như P5 đã mất rất nhiều thì giờ vì nó. Nhờ suy đoán, P6 đã nâng cao hiệu quả lên 100% so với P5. Bộ xử lí P6 còn có một số tính năng tiên tiến khác: dùng phương pháp đặt tên lại thanh ghi (register renaming) cho phép tăng số lượng thanh ghi lôgic lên nhiều để tránh trường hợp tranh chấp thanh ghi và sử dụng một ghép nối trực tiếp tốc độ cao với cache cấp hai nên không bị chậm vì bus số liệu khi truy cập cache. Với 5,5 triệu transistor, đầu tiên chạy với tốc độ 133MHz có phiên bản P6 đã chạy lên tới trên 200MHz. Các chip vi xử lí *Pentium II* với công nghệ *MMX* (Multimedia extensions) và bộ nhớ cache nội lớn tới 32Kbyte lại là một phiên bản cải tiến mới. Các chip này sử dụng ít lệnh hơn nhiều so với một CPU chuẩn thông thường. Với bộ nhớ cache nội lớn, nó có thể thực hiện các phép xử lí nhanh hơn Pentium tương đương từ 7÷10%. Chip có các tốc độ xung nhịp 166/200/233 MHz với trên 4,5 triệu transistor cho phép hỗ trợ cho các nhu cầu truyền thông đa phương tiện với các ứng dụng modem, đồ hoạ, video, audio cần tốc độ truyền dữ liệu nhanh. Pentium II ra đời tháng 5 năm 1997 có các tốc độ 233/266/300 MHz với độ tích hợp tới 7,5 triệu transistor cho một khả năng xử lí mạnh hơn. Do sử dụng những quy trình sản xuất mới cho phép giảm khoảng cách giữa các transistor cấu thành chip tới 0,25 micron, vì vậy có thể tăng tốc độ xung nhịp lên cao tới trên 400MHz. Với bus địa chỉ rộng 36 bit, chip có khả năng địa chỉ hoá bộ nhớ tới 64 Gbyte. Chip có bus cache tích hợp rộng 64 bit; bus dữ liệu ngoài rộng 64 bit và bên trong rộng tới 300 bit. Được giới thiệu vào mùa xuân năm 1999 với tốc độ xung nhịp 500MHz, kiến trúc của Pentium III tương tự như

Pentium II ngoại trừ việc thêm 70 lệnh được tối ưu cho truyền thông đa phương tiện. Tốc độ xung nhịp của chip có thể vượt quá 1GHz. Đến nay các Pentium IV cũng đã ra đời ...

3.3. HIỆU NĂNG VÀ ĐỘ TIN CẬY CỦA MỘT MÁY VI TÍNH

3.3.1. Hiệu năng của một vi xử lí

Hiệu năng của một vi xử lí được xác định bởi các thông số sau:

- *Tần số nhịp đồng hồ*, thông số này quyết định đến tốc độ thực hiện các tác vụ và tốc độ chuyển giao dữ liệu.

- *Độ rộng của bus dữ liệu*, bus số liệu càng rộng thì lượng thông tin được truyền tải trong một chu kì bus càng nhiều.

- *Độ rộng bus địa chỉ*, thông số này xác định số ô nhớ mà vi xử lí có thể quản lí được. Số ô nhớ đó bằng 2^n, trong đó n là độ rộng bus địa chỉ. Thường một ô nhớ chứa 1 byte = 8 bit thông tin.

Ngoài các thông số trên, bề rộng các thanh ghi bên trong vi xử lí (16 hay 32 bit) và cơ chế hoạt động (như cách kiến trúc: đường ống, siêu hướng, các cahe nội, ngoại cũng như các công nghệ xử lí CSIC, RISC, v.v...) cũng quyết định đến hiệu năng của vi xử lí trong từng trường hợp ứng dụng cụ thể.

Với kiểu cấu trúc của các máy vi tính hiện nay có thể đánh giá hiệu năng của vi xử lí theo đơn vị số triệu lệnh được xử lí trong một giây MIPS (Million Instructions per Second):

$$MIPS = \frac{f \times N}{M + T}$$

Trong đó: f là tần số làm việc của vi xử lí tính ra MHz;
N là số lượng các bộ tính số học và lôgic độc lập trong vi xử lí;
M là số vi lệnh trung bình trong một lệnh của vi xử lí;
T là hệ số thời gian truy cập bộ nhớ phụ thuộc vào số chu kì đợi.

3.3.2. Hiệu năng của một máy vi tính

Hiệu năng của máy vi tính (performance) là số đo sức mạnh tính toán của nó. Giá trị này không những phụ thuộc vào chính hiệu năng của vi xử lí mà còn phụ thuộc vào các bộ phận khác (như tốc độ thâm nhập của bộ nhớ, tốc độ giao tiếp của các thiết bị vào/ra v.v...). Do đó hiệu năng của một máy tính còn phụ thuộc vào cách mà vi xử lí tương tác với các bộ phận khác. Vi xử lí tương tác với các bộ phận khác trong hệ thống theo 3 cách : qua việc thâm nhập bộ nhớ trực tiếp hoặc gián tiếp, qua các cổng vào/ra và với các tín hiệu ngắt.

Bộ nhớ có thể được thâm nhập theo một trong hai cách: qua phương thức thâm nhập trực tiếp DMA hoặc qua các thanh ghi bên trong CPU. Bộ điều khiển ổ đĩa và các cổng thông tin nối tiếp có thể thâm nhập bộ nhớ trực tiếp qua bộ điều khiển DMA được. Tất cả thiết bị khác sẽ truyền số liệu với bộ nhớ qua các thanh ghi của vi xử lí. Các cổng vào/ra là các phương tiện thông tin sơ cấp của vi xử lí với các mạch điện trên máy tính ngoài bộ nhớ.

Thường xác định *hiệu năng* bằng cách cho máy tính chạy một tập các lệnh nào đó trong một số lần quy định rồi ghi lại thời gian thực hiện.

3.3.3. Độ tin cậy là một trong những nhân tố chính cần quan tâm khi chọn lựa các linh kiện và tính toán thiết kế chúng để tạo nên máy tính. Một thiết bị điện tử nói chung có thể bị

hỏng hóc do một trong rất nhiều nguyên nhân, nên về nguyên tắc thì không thể nói trước được khi nào thiết bị đã cho bị hỏng. Tuy nhiên, do các thiết bị chuyên dụng như máy tính chẳng hạn thường được hoạt động liên tục trong thời gian dài nên người ta thường quan tâm đến thời gian trung bình giữa các lần xảy ra hỏng hóc MTBF (Mean Time between Failure) và thời gian trung bình để sửa chữa MTTR (Mean Time To Repair) của một linh kiện thành phần. Phép lấy trung bình theo thời gian như vậy có thể được thay bằng phép lấy trung bình trên một tập hợp các linh kiện giống nhau. Như vậy, độ tin cậy của một linh kiện đã cho được định nghĩa bằng $1 - p$; ở đây p là xác suất xảy ra hỏng hóc trong một thời gian đã cho. Thí dụ, nếu 1 trong 10 linh kiện bị hỏng trong 1 năm (tức là p = 0,1) thì có thể nói độ tin cậy của thiết bị đó là 90%. Xét sau thời gian t, một số rất lớn các linh kiện N vẫn còn làm việc thì sự biến đổi của N theo thời gian dN/dt là một số âm (vì N giảm). Tỉ số của tốc độ giảm N trên tổng số N được gọi là *tốc độ hỏng hóc* L:

$$L \equiv -\frac{dN}{dt} / N$$

Tốc độ này thường được biểu diễn theo đơn vị phần trăm hỏng hóc trong 1000 giờ. Thí dụ, nếu một linh kiện được nói là có một tốc độ hỏng hóc là 0,003% trong 1000 giờ thì có nghĩa là sẽ trông đợi một con số là 30 hỏng hóc xảy ra trong 1000 giờ khi xét trong một tập hợp 1 triệu linh kiện như vậy.

Giả thiết rằng tốc độ hỏng hóc L là không đổi, từ biểu thức trên có thể viết:

$\log N = -Lt + C$ với C là hằng số tích phân

hay:

$$N = e^{-Lt+C} = e^{-Lt}.e^{C} = Ke^{-Lt} \text{ với } K = e^{C}$$

K đại diện cho số linh kiện ban đầu tại $t=0$ và thường được viết là N_0. Vậy:

$$N = N_0 e^{-Lt}$$

Nếu ta định nghĩa độ tin cậy r(t) của một tập hợp N_0 linh kiện là:

$$r(t) \equiv \frac{N(t)}{N_0} = e^{-Lt}$$

thì thời gian trung bình giữa các lần hỏng hóc MTBF (chính là thời gian sống mong đợi của một linh kiện) là:

$$\frac{\sum_{n=1}^{N_0} \textit{thời gian sống của linh kiện thứ } i}{N_0} = \int_0^{\infty} r(t)dt$$

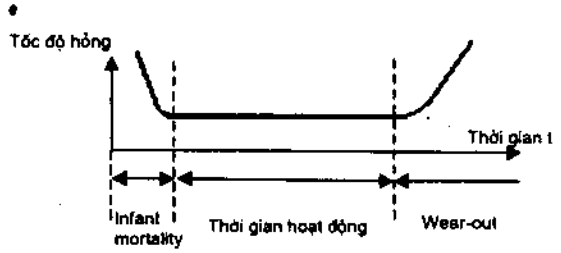
Khi tích phân biểu thức này $MTBF = m = 1/L$.

Như vậy, $r(t) = e^{t/m}$. Cần lưu ý rằng thời gian trung bình giữa các lần hỏng hóc là một đại lượng nhận được bằng các phép đo đạc. Vậy tốc độ hỏng hóc của một tập hợp các linh kiện phải được đo bằng việc quan sát hoạt động của chúng. Dĩ nhiên linh kiện chọn lọc có thể không được nằm trong số đo đạc thống kê và có thể xảy ra hỏng hóc sớm hơn.

Một mặt ta mong muốn có các linh kiện với MTBF lớn, mặt khác lại mong muốn có các linh kiện hay hệ thống có thể được sửa chữa nhanh chóng khi chúng bị hỏng. Tham số hữu ích khác là MTTR (cũng được gọi là MTRF - mean time to repair a fault). Hai tham số này có thể được kết hợp để cho một tham số gọi là giá trị của một hệ thống:

$$Giá\ trị\ của\ hệ\ thống = \frac{MTBF}{MTBF + MTTR}$$

Trong thực tế, tốc độ hỏng hóc L, và do đó cả m, phụ thuộc theo thời gian như hình 3.18 chỉ ra. Phần bên trái của đường cong là sự hỏng trước của một linh kiện trong quá trình sản xuất. Phần nằm ngang là thời gian sử dụng của linh kiện và phần bên phải tương ứng với sự già hoá của linh kiện do làm việc lâu ngày (wear-out).Vì vậy, nhiều công ti sản xuất linh kiện và thiết bị đã xây dựng một quá trình xử lí *burn-in* (đốt nóng) linh kiện trước khi bán ra thị trường. Trong quá trình này, các linh kiện được cho hoạt động ở nhiệt độ cao hơn mức bình thường nhằm tăng tuổi làm việc của linh kiện trong một thời gian ngắn. Nói cách khác là để rút ngắn thời gian hỏng trước (*infant mortility*).



Hình 3.18 Sự phụ thuộc tốc độ hỏng vào thời gian.

Các linh kiện bị hỏng trong quá trình xử lí này sẽ được loại bỏ và do đó người mua sẽ nhận được các linh kiện còn lại có độ tin cậy cao hơn. Dĩ nhiên, các linh kiện đã được burn-in sẽ có giá thành cao hơn. Sự hỏng hóc của các linh kiện là một hàm của nhiệt độ. Khi nhiệt độ làm việc tăng thì độ tin cậy giảm. Ta sẽ xét sự hỏng hóc của một hệ thống gồm nhiều linh kiện tạo nên. Giả thiết rằng một hệ thống gồm 2 linh kiện R_1 và R_2 có độ tin cậy tương ứng là r_1 và r_2. Giả thiết rằng hệ sẽ hỏng khi một trong hai linh kiện bị hỏng. Vậy độ tin cậy của hệ thống này $r = r_1 . r_2$. Thí dụ, nếu R_1 có *99%* độ tin cậy và R_2 có *95%* độ tin cậy thì độ tin cậy toàn bộ là $0,99 \times 0,95 = 0,94$ hay *94%*. Độ tin cậy toàn hệ thống thực ra phụ thuộc vào độ tin cậy thành phần thấp nhất. Khi sử dụng (mắc mạch, chạy ...) một cách nối tiếp các linh kiện thì độ tin cậy toàn hệ thống sẽ chính xác bằng tích của các độ tin cậy thành phần của các linh kiện.

Có khả năng thiết kế một hệ thống chỉ hỏng khi có hơn một linh kiện hỏng khi sử dụng một cách song song các linh kiện với nhau. Một hệ thống như vậy được gọi là hệ thống có *dư thừa* (redundancy) hay hệ thống *có dự phòng* (back-up). Thí dụ khi cho chạy song song 2 ổ đĩa trong một máy vi tính thì hệ thống sẽ chỉ hỏng khi cả 2 ổ đĩa bị hỏng.

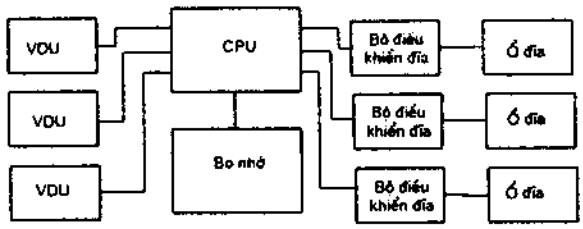
Nếu độ tin cậy của thành phần thứ i là r_i thì xác xuất xảy ra hỏng hóc của nó là $1 - r_i$. Vậy nếu có n linh kiện chạy song song thì xác suất xảy ra hỏng hóc toàn bộ là:

$(1 - r_1)(1 - r_2) \ldots (1 - r_n)$.

Do đó độ tin cậy của toàn hệ thống là:

1 - độ bất tin cậy = $1 - (1 - r_1)(1 - r_2) \ldots (1 - r_n)$

Tất nhiên số linh kiện chạy song song càng nhiều thì giá thành sản phẩm cũng như năng lượng tiêu tốn cho hệ thống càng cao. Thí dụ, xét một hệ thống như hình 3.19 gồm có một CPU, một môđun nhớ, ba VDU, 3 bộ điều khiển ổ đĩa và 3 ổ đĩa.

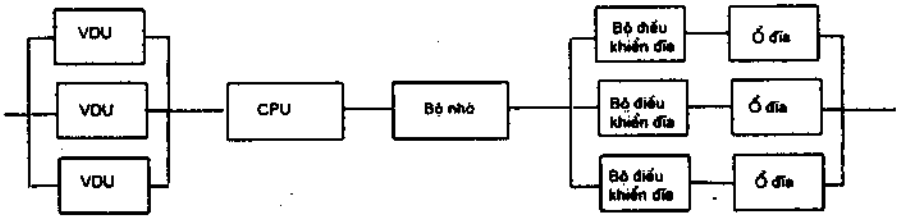


Hình 3.19 Hệ thống máy vi tính với 3 VDU và 3 ổ đĩa.

Độ tin cậy của các linh kiện thành phần như sau:

Linh kiện	Độ tin cậy
VDU	0,98
CPU	0,999
Bộ nhớ	0,8
Bộ đ/k đĩa	0,995
Ổ đĩa	0,8

Nếu muốn tăng độ tin cậy lên trong đó các VDU và các bộ điều khiển và ổ đĩa được chạy song song với nhau thì có thể thiết kế hệ như hình 3.20 sau. .



Hình 3.20 Thiết kế sơ đồ tăng độ tin cậy của máy tính ở hình 3.19.

Lúc này hệ thống chỉ gồm 4 nhóm được sử dụng một cách nối tiếp là: các VDU, CPU, bộ nhớ và các bộ điều khiển và các ổ đĩa. Độ tin cậy của các thành phần hệ thống sẽ là:

Các VDU: 1-(1-0,98)(1-0,98)(1-0,98) = 0,999992
CPU : 0,999
Bộ nhớ : 0,8

Các bộ điều khiển và ổ đĩa:

$$1 - (1 - 0,995 \times 0,8)(1 - 0,995 \times 0,8)(1 - 0,995 \times 0,8) = 0,9915$$

Vậy độ tin cậy của toàn hệ thống là tích 4 thành phần này là:

$$0,999992 \times 0,999 \times 0,8 \times 0,9915 = 0,7924 \text{ hoặc } 79\%$$

Cần lưu ý rằng các kết quả tính toán này hầu như phụ thuộc vào độ tin cậy thấp của bộ nhớ. Nếu thiết kế 2 môđun nhớ hoạt động song song với nhau thì độ tin cậy của chúng sẽ là 1 - (1 - 0,8)(1 - 0,8) = 0,96 và độ tin cậy của hệ thống sẽ tăng lên tới là *0,9509* hay *95%* !

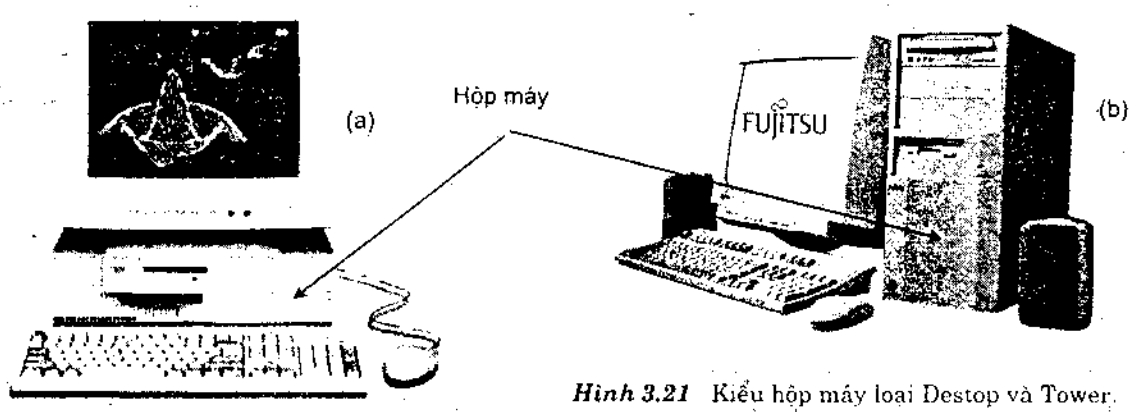
Rõ ràng cách tính độ tin cậy phụ thuộc vào 2 giả thiết: một là độ tin cậy của các linh kiện là không đổi trong suốt thời gian sống của nó; hai là độ tin cậy của toàn hệ thống được tính theo sơ đồ nối tiếp hoặc song song. Điều này chỉ đúng cho các hệ đơn giản còn với các hệ phức tạp phải được tính trong các hoàn cảnh cụ thể thậm chí phụ thuộc cả vào các yếu tố có liên quan đến con người như điều kiện bảo dưỡng thiết bị.

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

3.1 Thực hành khảo sát một hệ máy vi tính IBM PC.

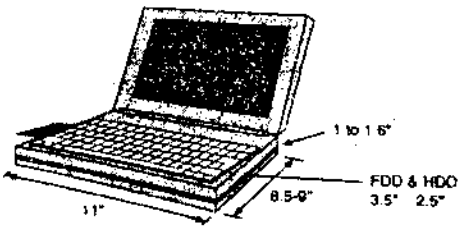
Một máy vi tính chuẩn hiện nay ít nhiều có các phần sau:

- *Một bản mạch chính*. Trên bản mạch này chứa các bộ phận:
  - Một chip vi xử lí cùng bộ đồng xử lí toán,
  - Bộ nhớ chính có tốc độ truy cập cao bao gồm bộ nhớ quy ước, bộ nhớ mở rộng, bộ nhớ cache và ROM BIOS hoặc flash BIOS,
  - Các chip bổ trợ giúp vi xử lí trong những việc riêng biệt,
  - Một tập hợp các bus và các khe cắm mở rộng dùng để kết nối vi xử lí với bộ nhớ và các bản mạch ghép nối,
  - Một tập hợp các bản mạch ghép nối cho phép vi xử lí liên lạc và điều khiển các thiết bị vào/ra qua các cổng là các đầu nối vật lí. Các bản mạch này có thể được cắm trên các khe cắm mở rộng nhưng cũng có thể được tích hợp ngay trên bản mạch chính (on board).
- *Các thiết bị nhớ ngoài* (nhớ khối) có dung lượng lớn nhưng tốc độ truy cập thấp hơn như các ổ đĩa từ cứng và mềm, các ổ đĩa quang CD-ROM v.v...
- *Các thiết bị vào/ra* để nhập xuất dữ liệu cho vi xử lí như bàn phím, chuột, màn hình và máy in v.v...
- *Bộ nguồn nuôi* từ mạng điện thành phố hoặc ắc quy.



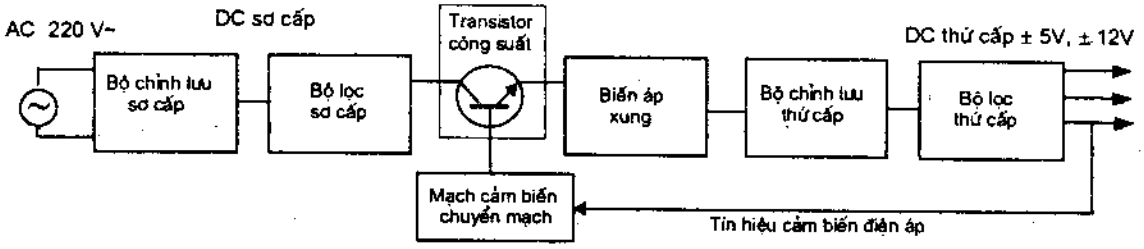
Hình 3.21 Kiểu hộp máy loại Destop và Tower.

Trừ các thiết bị vào/ra như bàn phím, màn hình, chuột và máy in nằm riêng rẽ ở bên ngoài; các bộ phận còn lại thường được lắp đặt bên trong *hộp máy PC. Kiểu hộp máy* quy định hình dạng, kích thước và cách bố trí các bộ phận bên trong hộp máy PC. Các kiểu hộp máy này được thiết kế tuỳ theo nhu cầu về không gian làm việc, khả năng mở rộng thiết bị và sự thuận tiện trong sử dụng. Hiện có 3 loại phổ biến cho các máy tính để bàn: loại Desktop có hộp CPU nằm ngang như hình 3.21a; loại Tower có hộp CPU kiểu đứng và Minitower (hộp đứng loại nhỏ) như hình 3.21b. Ngoài ra còn các kiểu *máy sổ tay* (notebook, laptop) cho phép mang xách được, ở đó màn hình, bàn phím và cả chuột được lắp đặt ngay trên một khối được chỉ ra trên hình 3.22.



Hình 3.22 Kiểu dáng máy Laptop (notebook).

Sơ đồ khối bộ nguồn nuôi kiểu đóng ngắt trong máy vi tính như hình 3.23, trong đó transistor (hoặc vi mạch) công suất trong mạch tạo dao động cùng sơ cấp biến áp xung được nối trực tiếp với mạng điện thành phố 220VAC. Vì vậy khi cần đo đạc các dạng xung điện ở bên sơ cấp bằng các thiết bị điện tử được nuôi bằng mạng điện thành phố cần hết sức cẩn thận. Mạch thứ cấp của biến áp xung dao động tạo các nguồn ra một chiều DC ±5V, ±12V nuôi các bộ phận trong máy vi tính được phân cách về điện với mạch sơ cấp (là mạch nối trực tiếp với mạng điện thành phố 220VAC) vì biến áp này là loại biến áp phân cách giữa sơ và thứ cấp. Do vậy, các dây nối đất (GND) là an toàn trong trường hợp hệ hoạt động bình thường.

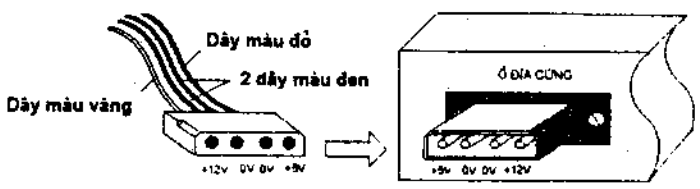


Hình 3.23 Sơ đồ khối của nguồn nuôi kiểu đóng cắt.

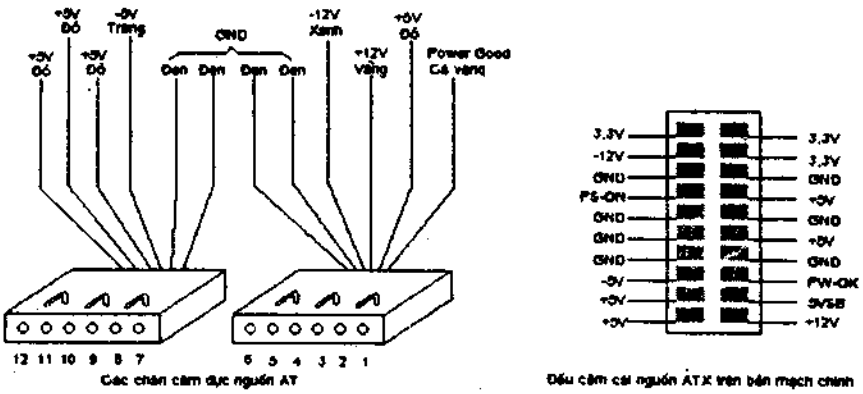
Phần phụ lục 2 cho sơ đồ nguyên lí chi tiết của một loại nguồn nuôi máy IBM-PC/486.

Các lối ra của nguồn nuôi ±5V và ±12V trên các nguồn AT và ATX là các đầu cắm đực và cái chuẩn tương thích với các đầu cắm cái và đực trên bộ phận được nuôi như bản mạch chính và các thiết bị ngoại vi. Hình 3.24 là các đầu cắm đực loại to cung cấp các điện áp nuôi cho ngoại vi và ổ cắm cái trên ổ đĩa từ cứng. Các đầu cắm đều có các khấc định vị nên không thể cắm nhầm với nhau được, vì vậy để tránh hỏng hóc cần hết sức tránh tháo rời các dây ra khỏi các ổ nhựa chuẩn khi bảo dưỡng máy.

Hình 3.24 Đầu cắm đực (từ nguồn AT) và ổ cái trên ổ đĩa cứng.



Hình 3.25 là các đầu cắm nuôi bản mạch chính của nguồn AT và ATX.



Hình 3.25 Các đầu cắm nguồn nuôi AT và ATX.

3.2 Trả lời các câu hỏi sau:

- Trạng thái của 8086 được bắt đầu đo từ điểm nào trên xung đồng hồ ?
- Tại sao các bus AD0 - AD15 lại cần các bộ chốt ?
- Mục đích của tín hiệu ALE ?
- Mô tả các sự kiện tuần tự trên các đường dữ liệu/địa chỉ, ALE, $M/\overline{IO}$, $\overline{RD}$ khi 8086 nhận và thực hiện một từ lệnh.
- Khi 8086 thực hiện viết vào một ô nhớ, $\overline{RD}$, $\overline{WR}$ và $M/\overline{IO}$ có mức logic gì ? Khi đọc từ cổng thì thế nào ?
- Nêu điểm khác nhau cơ bản giữa các chế độ hoạt động Min và Max của 8086.
- Mô tả đáp ứng của 8086 khi chân Reset được đặt lên mức cao.
- Tại sao trong máy tính cần có các bộ đệm trên bus địa chỉ, dữ liệu, điều khiển.
- 8086 chuyển sang trạng thái đợi thế nào.
- 8086 chuyển sang trạng thái đợi ở thời điểm nào trong một chu kì máy?
- Trong trạng thái đợi, thông tin trên các bus như thế nào?
- Trạng thái đợi kéo dài bao lâu?
- Có bao nhiêu trạng thái đợi có thể được chèn vào một chu kì máy?
- Tại sao 8086 cần phải chèn các trạng thái đợi ?
- Chức năng của các tín hiệu $\overline{DEN}$ và $DT/\overline{R}$ là gì?
- Mũi tên chỉ từ điểm này đến điểm khác trên dạng xung tín hiệu cho biết điều gì ?
- Vẽ một sơ đồ khối của một bộ phân tích logic đơn giản và mô tả ngắn gọn về hoạt động của nó (bao gồm cả hoạt động của xung đồng hồ và bộ kích hoạt).
- Cần sử dụng cái gì để tạo xung nhịp cho bộ đo khi muốn đo thời gian?
- Tín hiệu nào và sườn xung nào ta dùng định thời cho bộ đo và dùng word nào để đặt tại đầu vào bộ nhận dạng từ để kích hoạt (trigger) bộ đo dùng cho việc quan sát những thông số sau:
  a. Các địa chỉ gửi ra sau lệnh RESET.

b. Các lệnh đọc được sau lệnh RESET (giả sử từ lệnh đầu tiên là 9CFAh)

c. Cả địa chỉ gửi ra và từ đọc được.

d. Nối đầu vào clock quantifer với tín hiệu nào để tạo ra chỉ dữ liệu đọc được từ cổng.

- Bộ phân tích logic có thể hiển thị dữ liệu có trước khi bị kích như thế nào ?
- Trình bày 2 mục đích của các bộ giải mã địa chỉ trong hệ thống máy tính.

3.3 Tin chắc rằng máy vi tính ở trạng thái ngắt điện; quan sát, phân định và ghi nhận xét về các bộ phận đã được nêu ở phần tóm tắt kể trên.

3.4 Bật máy tính, theo dõi quá trình khởi động. Quan sát các hoạt động của máy: các môtơ làm mát nguồn nuôi, CPU; các đèn báo nguồn, báo hoạt động của ổ đĩa cứng, đĩa mềm; các báo hiệu của loa PC; phân định các tiếng động của ổ đĩa; xem các thông báo trên màn hình trong quá trình khởi động. Ghi các nhận xét để có thể mô tả lại quá trình khởi động bên ngoài của PC cho tới khi DOS hiện ra màn hình dấu nháy con trỏ sẵn sàng hoặc Windows hiện cửa sổ làm việc.

3.5 Khảo sát máy vi tính bằng lệnh MSD.EXE trong thư mục DOS.

3.6 Cho chạy chạy lệnh MEM.EXE trong DOS với các tuỳ chọn /C; /D; /M; /F. Ghi nhận xét về các thông tin trên màn hình.

3.7 Khảo sát máy vi tính bằng chương trình SYSINFO.EXE trong thư mục NU.

3.8 Khảo sát máy vi tính bằng chương trình Debug theo các bước sau:

- Tại dấu nháy của DOS, tạo một file có tên TEST.bat như sau;

  copy con TEST.bat <Enter>

  Dai hoc Quoc gia Ha noi, Khoa Cong nghe, Hello ! Welcome !

  ^z <Enter.

- Cho chạy chương trình Debug.exe (xem chương 19) như sau để nạp file văn bản vừa tạo ra vào đoạn nhớ hiện tại và có địa chỉ offset là 100h:

  Debug TEST.bat <Enter>

  Chạy lệnh d 100 <Enter> .

  Quan sát và ghi lại nhận xét các chỉ thị trên màn hình.

- Chạy lệnh R để hiện nội dung các thanh ghi. Giải thích các giá trị thanh ghi và các bit cờ.
- Điền các ô nhớ từ địa chỉ offset = 105h các giá trị 41h (tương ứng với kí tự ASCII "A") bằng lệnh F rồi quan sát lại vùng nhớ từ 100h bằng lệnh d 100. Nhận xét ? Điền chữ U ?
- Cổng máy in LPT có địa chỉ 378h của máy vi tính trong phòng thí nghiệm đã được ghép sẵn các chỉ thị lôgic ở 8 bit ra (D0-D70. Bằng lệnh Debug (OUT) viết các byte có giá trị 1, 2, 4, 8, 16, 32, 64, 128 ra cổng. Quan sát các đèn chỉ thị LED.
- Viết đoạn chương trình ở địa chỉ 100h xuất ra cổng LPT dãy số từ 0 đến 255 bằng lệnh A và cho chạy bằng lệnh G 100. Quan sát các đèn chỉ thị và nhận xét.

3.9 Dùng các công cụ vôn mét và dao động kí đo và ghi lại các dạng sóng tại các điểm sau:

- Vôn mét đo giá trị thế tại các đầu ra bộ nguồn nuôi.
- Dao động kí đặt ở chế độ ghép tín hiệu vào xoay chiều (AC); quan sát và ghi lại thế mấp mô tại các lối ra nguồn nuôi.

- Dùng dao động kí quan sát dạng sóng tại chân CLK (B20) trên một khe cắm ISA còn để trống. Ghi và đo tần số của dạng sóng. Nếu đo trên một máy vi tính có tốc độ khác thì tần số này có thay đổi không ? Tại sao ?

3.10 Dùng dao động kí thử quan sát dạng sóng tại các dây bất kì trên bus địa chỉ và bus số liệu. Kết quả ? Cho nhận xét và giải thích nguyên nhân.

3.11 Khảo sát phiên bản ROM-BIOS:

- Có thể dùng lệnh nào trong chương trình Debug để kiểm tra phiên bản của ROM BIOS ? Phiên bản này được lưu trữ tại địa chỉ F000 : FFFE với các máy IBM. Mã nhận dạng như sau:

Mã	Ý nghĩa
FCh	AT
Feh	PC hoặc XT
FFh	PC

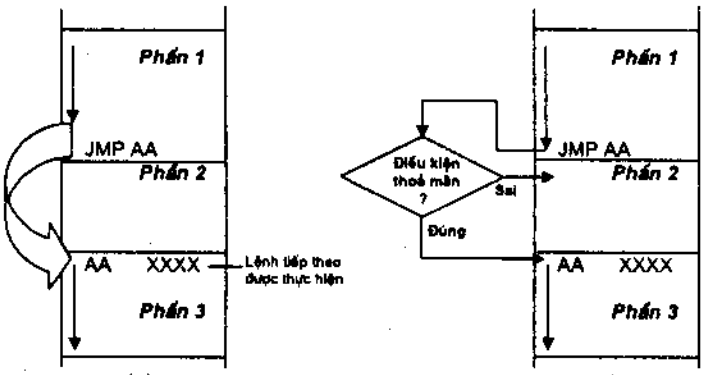
- Ta biết rằng sau khi khởi động, chương trình nhảy đến địa chỉ F000:FFF0 gần đỉnh bộ nhớ 1MB của 8086. Tại địa chỉ này có một lệnh nhảy dài 5 byte tới địa chỉ chương trình khởi động, còn 11 byte nữa để ghi thông tin về thế hệ và ngày xuất xưởng của BIOS. Hãy dùng một lệnh của Debug khảo sát thông tin này, chú ý rằng định dạng được ghi ở đây là *định dạng Intel* tức là byte thấp trước byte cao sau.

3.12 Nhập và cho chạy chương trình cho phép đo tốc độ hoạt động của CPU theo đơn vị MHz bằng cách đo chính xác một vòng lặp ngắn. Vì các vi lệnh khác nhau giữa các hãng Cyrix, IBM và INTEL, AMD nên việc định thời lượng của các lệnh (instruction timing) cũng khác nhau. Để hiệu chỉnh trước tiên phải nhận diện được hãng cung cấp, sau đó chương trình sẽ chọn một hệ số tỉ lệ đặc trưng cho hãng để bù lại những sai khác này. Chương trình thực hiện việc đo đạc độc lập với các thiết kế cache ngoại. Chương trình sẽ báo tốc độ chạy bên trong CPU. Lệnh xoá cờ ngắt CLI được dùng để cấm các ngắt trong quá trình chạy chương trình. Việc đo thời gian ở đây sử dụng bộ đếm 2 của chip PIT-8253 trong PC. Tín hiệu nhịp vào đầu vào CLK2 luôn giữ ở 1,19318 MHz bất kể loại CPU nào đang được dùng trên bản mạch chính.

3.13 *Nghiên cứu các lệnh nhảy và việc tạo các thời gian trễ chính xác.*

Sức mạnh thực sự của máy tính xuất phát ở chỗ nó có khả năng chọn giữa hai hoặc một chuỗi các hoạt động dựa trên một vài điều kiện nào đó, lặp lại một chuỗi các lệnh cùng với một vài điều kiện tồn tại hoặc lặp lại một chuỗi các lệnh cho đến khi điều kiện nào đó xảy ra. Đó là các lệnh nhảy không điều kiện và có điều kiện. Các lệnh nhảy này được sử dụng để chỉ cho CPU các địa chỉ nhận lệnh tiếp theo (không nhất thiết là địa chỉ kế tiếp sau lệnh vừa thực hiện). Khi thực hiện các lệnh nhảy có điều kiện, các cờ trạng thái chỉ thị liệu khi nào các điều kiện này tồn tại hay không. Trong lệnh nhảy không điều kiện, không có yêu cầu về trạng thái cờ nào áp đặt lệnh nhảy xảy ra, nghĩa là lệnh nhảy luôn luôn luôn xảy ra bất kể trạng thái của các cờ hiệu. Nói cách khác, khi lệnh được thi hành, một cú nhảy luôn xảy ra để thay đổi trình tự thi hành. Hình 3.26a mô tả trình tự chương trình có lệnh nhảy không điều kiện. Khi lệnh nhảy không điều kiện JMP AA trong phần 1 được thực hiện, việc điều khiển chương trình được chuyển sang một điểm trong phần 3 được nhận diện bởi nhãn AA.

Việc thi hành các lệnh được tiếp tục với lệnh tương ứng ở vị trí AA. Như vậy, các lệnh trong phần 2 của chương trình đã bị bỏ qua. Trái lại, đối với một lệnh nhảy có điều kiện, các điều kiện tình trạng hiện có tại thời điểm lệnh nhảy được thi hành sẽ quyết định liệu có nhảy hay không. Nếu (các) điều kiện này được thoả mãn thì việc nhảy sẽ xảy ra; còn ngược lại chương trình sẽ thực hiện lệnh kế tiếp sau lệnh nhảy. Các điều kiện được tham khảo bởi một lệnh nhảy có điều kiện là các cờ trạng thái như cờ mang (CF), cờ chẵn lẻ (PF) và cờ tràn (OF). Như hình 3.26b đã chỉ ra, việc thi hành lệnh nhảy có điều khiển trong phần 1 dẫn đến việc thực hiện một phép kiểm tra. Nếu các điều kiện của phép kiểm tra không thoả mãn việc thi hành chương trình sẽ theo đường "sai" và tiếp tục thực hiện lệnh kế tiếp. Ngược lại, nếu kết quả phép kiểm tra là "đúng", một cú nhảy sẽ được tiến hành để chuyển đến phần chương trình được nhận diện là AA ở phần 3 và các lệnh trong phần 2 bị bỏ qua.



Hình 3.26 Trình tự của chương trình có lệnh nhảy không điều kiện (a) và có điều kiện (b).

Chú ý tới các lệnh nhảy và thời gian thực hiện một lệnh ta có thể tạo được các *khoảng thời gian trễ* chính xác trong máy tính. Tốc độ chạy các lệnh của vi xử lí được quyết định bởi đồng hồ hệ thống. Mỗi một lệnh xảy ra trong một khoảng thời gian nào đó. Nói cách khác nếu lấy chu kì nhịp đồng hồ hệ thống làm đơn vị thì có thể nói mỗi một lệnh được chạy trong khoảng thời gian tồn tại một số chu kì xung nhịp nhất định (hay nói vắn tắt là lệnh đó chiếm một số xung đồng hồ nào đó). Lệnh MOV chuyển nội dung giữa 2 thanh ghi chẳng hạn đòi hỏi 2 chu kì nhịp đồng hồ để chạy; lệnh DAA đòi hỏi 4 chu kì đồng hồ v.v... Lệnh JNZ đòi hỏi 16 chu kì nhịp nếu điều kiện nhảy thoả mãn nhưng chỉ mất có 4 nhịp đồng hồ nếu không nhảy. Bảng trong phụ lục sẽ chỉ rõ số chu kì nhịp đồng hồ đòi hỏi cho mỗi lệnh. Khi sử dụng số liệu trong bảng này ta có thể tính thời gian chính xác để chạy một lệnh hoặc một chuỗi lệnh. Thí dụ, nếu tốc độ đồng hồ 8086 là 5 MHz thì chu kì nhịp là $1/50.10^6 = 0{,}2$ µs; như vậy một lệnh cần 4 nhịp đồng hồ sẽ được thực hiện trong 0,2 µs x 4 = 0,8 µs. Trong thực tế cần tạo ra các khoảng thời gian trễ chính xác giữa 2 lệnh. Thí dụ, ta muốn đọc giá trị số liệu từ một cổng vào/ra với quãng cách chính xác từng 1 mili giây một. Ta có thể dùng một vòng lặp để thực hiện khoảng trễ 1 ms đó :

```
        MOV  CX, N          ;  4 chu kì = C0
Trễ:    NOP                 ;  3
        NOP                 ;  3          = CL
        LOOP Trễ            ; 17 hoặc 5
```

Lệnh MOV CX,N nạp cho thanh ghi CX số thời gian muốn lặp lại vòng trễ. Lệnh LOOP Trễ sẽ giảm nội dung thanh ghi CX đi 1 và nếu nội dung này chưa bằng 0 nó sẽ cho chương trình nhảy tới địa chỉ lệnh có nhãn "Trễ". Chương trình khi đó sẽ bắt vi xử lí chạy 2 lệnh NOP và lệnh LOOP cho đến khi nội dung của CX bằng 0.

Đầu tiên ta tính số chu kì đồng hồ C_T cần để tạo ra khoảng trễ mong muốn, thí dụ cho 8086 chạy ở tần số 5 MHz:

C_T = 1000 μs /0,2 μs = 5000 chu kì.

Bước tiếp theo là viết số chu kì đồng hồ được đòi hỏi cho mỗi lệnh như trên. Số chu kì nhịp đồng hồ cho các lệnh chỉ chạy một lần trong vòng lặp được gọi là *overhead* C_0. Lệnh đó là MOV CX,N chiếm 4 chu kì đồng hồ.

Tiếp theo ta xác định số chu kì nhịp đồng hồ xảy ra trong vòng lặp. 2 lệnh NOP chiếm 6 chu kì, Lệnh LOOP chiếm 17 chu kì nếu nó không nhảy lại nhãn "trễ" và chỉ chiếm 5 chu kì nếu còn trong vòng lặp. Cú nhảy thực ra dài hơn vì hàng nhận lệnh phải nạp lại từ đầu địa chỉ mới. Trừ thời gian rất gần khoảng kết thúc vòng lặp, toàn bộ đòi hỏi hết 17 chu kì. Do đó ta có thể dùng 17 như là số chu kì cho lệnh LOOP và bù trừ về sau cho sự kiện là thời gian kết thúc chiếm ít hơn 12 chu kì. Trong trường hợp này, số chu kì trong một vòng lặp là: $C_L = 6 + 17 = 23$.

Tổng số chu kì nhịp đồng hồ được trễ bởi vòng lặp bằng số thời gian vòng lặp chạy nhân với thời gian cho một vòng. Để chính xác phải trừ đi 12 chu kì không dùng đến khi chạy lệnh LOOP cuối cùng. Tổng số chu kì đòi hỏi là:

$C_T = C_0 + N(C_L) - 12.$

Vậy, $N = \frac{C_T - C_0}{C_L} + 12 = \frac{5000 - 4 + 12}{23} = 218 = 0DAh$

Có thể nhận được một thời gian trễ đủ dài bằng việc đếm lùi một thanh ghi hoặc một ô nhớ nào đó khi lồng các vòng trễ với nhau, thí dụ:

```
                              ;     số trạng thái
            MOV BX,COUNT 1    ;     4
CNTDN1: MOV CX,COUNT 2        ;     4(COUNT1)
CNTDN2: LOOP CNTDN 2          ;     ((17 x COUNT2) - 12)COUNT1
            DEX BX            ;     2(COUNT1)
            JNZ CNTDN1        ;     16(COUNT1) - 12
```

Nguyên tắc ở đây là nạp CX số COUNT2 và đếm lùi thanh ghi CX một số lần bằng COUNT1. Để xác định số trạng thái mà chương trình sẽ chạy, quan sát thấy lệnh LOOP sẽ chạy COUNT2 lần cho mỗi lần CX được nạp với COUNT1. Tổng số trạng thái là COUNT1 lần số các trạng thái tới 4 lệnh cuối cùng cộng với 4 cho lệnh MOV BX, COUNT1. Cách tốt nhất để nhận được các giá trị chưa biết COUNT1 và COUNT2 là chọn một giá trị như FFFFh cho COUNT2 và tìm giá trị COUNT1.

Khi sử dụng vòng lặp trễ để tạo các khoảng thời gian chính xác cần lưu ý:

- Các đơn vị BIU và EU trong vi xử lí hoạt động không đồng bộ với nhau, do vậy một số chu kì nhịp bổ sung cần phải thêm vào cho một số chuỗi lệnh. Với một chuỗi lệnh đã cho,

các chu kì thêm vào luôn như nhau nhưng hiển nhiên là các chu kì này không bao gồm các số đã cho trong phụ lục.

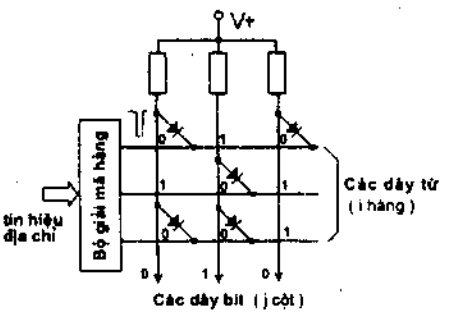
- Số chu kì nhịp đòi hỏi để đọc một từ dữ liệu từ bộ nhớ hoặc viết một từ tới bộ nhớ phụ thuộc vào byte đầu tiên của từ được đặt ở địa chỉ chẵn hay lẻ. Vi xử lí 8086 sẽ đòi hỏi thêm 4 chu kì nhịp để đọc hoặc viết một từ được đặt ở các địa chỉ lẻ.
- Số chu kì nhịp đồng hồ đòi hỏi đọc/viết từ hoặc đến bộ nhớ phụ thuộc vào các chế độ địa chỉ dùng để thâm nhập byte đó.
- Nếu một hệ máy vi tính được thiết kế để xen vào các trạng thái đợi trong quá trình thâm nhập bộ nhớ thì sẽ làm tăng số chu kì đòi hỏi cho thâm nhập bộ nhớ.

3.14 Mục đích của các tụ điện nhỏ được nối giữa Vcc và đất trong các sơ đồ mạch in của máy vi tính là để làm gì ?

CHƯƠNG 4. VI XỬ LÍ VÀ BỘ NHỚ CHÍNH

Các vi mạch nhớ bán dẫn có tốc độ truy cập cao, năng lượng tiêu thụ thấp và nhờ tính tương thích về kích thước và mức lôgic trong các máy vi tính, nên được thiết kế làm *bộ nhớ chính* (hay bộ nhớ trong) nằm trên *bản mạch chính* cùng với CPU. Bộ nhớ chính được tạo thành từ các mô-đun nhớ, các mô-đun này được thiết kế lắp ráp trên cơ sở các chip nhớ có dung lượng và độ dài từ hạn chế.

4.1. CÁC CHIP NHỚ BÁN DẪN



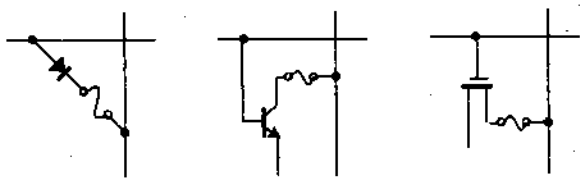
Hình 4.1 Bộ nhớ ROM diode.

4.1.1. Các loại vi mạch nhớ

Các vi mạch nhớ này gồm 2 loại chính là ROM và RAM. Riêng với RAM lại có 2 loại là SRAM (RAM tĩnh) và DRAM (RAM động).

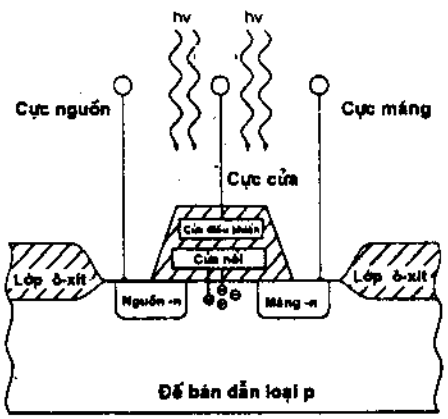
Bộ nhớ chỉ có thể đọc ROM (Read-Only Memory) là loại có nội dung được viết sẵn một lần khi chế tạo và được giữ mãi cố định (non volatile). ROM lập trình kiểu mặt nạ được chế tạo trên một phiến si-líc nhằm tạo ra những tiếp giáp bán dẫn điện theo một chiều (thí dụ, transistor, diode) tại các điểm vắt chéo nhau trên một ma trận các dây dẫn hàng (từ số liệu) và cột (bit số liệu) như thí dụ trong hình 4.1. Tại đó, các điểm vắt chéo chứa diode sẽ mang thông tin là 0, các điểm còn lại mang thông tin là 1. Khi lối ra bộ giải mã địa chỉ ở mức thấp chọn một hàng thì thế lối ra của các dây bit phản ánh các giá trị được lưu trữ trong chip nhớ. Trong trường hợp này chip lưu trữ 3 từ nhớ, mỗi từ 3 bit là 010, 101 và 001.

Các chip PROM (Programmable ROM) cho phép người dùng có thể ghi thông tin được 1 lần. Đó là một loại ROM mà khi sản xuất, tất cả các điểm vắt chéo đều được đặt các diode hoặc transistor nối tiếp với 1 cầu chì như hình 4.2. Khi cần ghi thông tin với mức lôgic "1" ở điểm vắt chéo nào thì chỉ việc cho dòng đủ lớn đi qua và làm cháy đứt cầu chì tương ứng ở điểm đó. Rõ ràng loại PROM này chỉ ghi thông tin được một lần và không xoá được.



Hình 4.2 Cầu chì trong các điểm vắt chéo với diode, transistor lưỡng cực và transistor trường.

Các chip nhớ EPROM (Erasable PROM) cho phép xoá được bằng tia cực tím và *EEPROM* (Electrically EPROM) cho phép xoá được bằng điện. Trong các chip nhớ này, thay vì cho diode hoặc transistor, ở các điểm vắt chéo dùng các transistor MOS có *cửa nổi* được chế tạo theo công nghệ FAMOST (Floating gate avalanche injection MOS transistor) như hình 4.3. Số liệu có thể được viết vào bằng điện và được lưu giữ theo kiểu cố định. Trong ô nhớ dùng transistor này,



Hình 4.3 Một ô nhớ FAMOST.

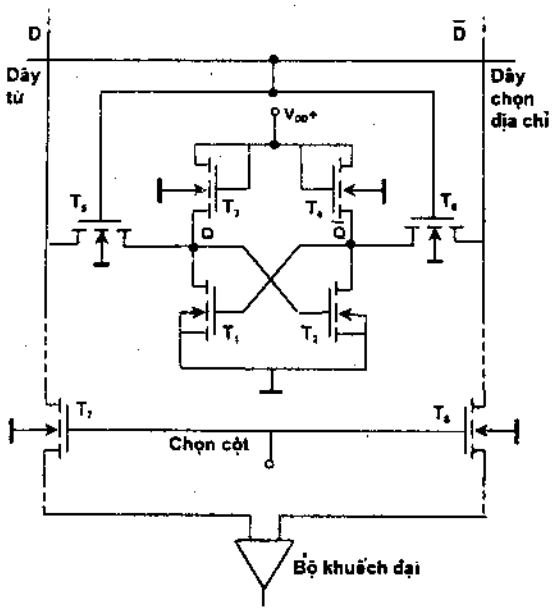
cực cửa được nối với đường từ, cực máng nối với đường bit và cực nguồn nối với nguồn chuẩn được coi là nguồn cho mức lôgic 1. *Cửa nổi* (floating gate) là một vùng vật liệu được thêm vào giữa lớp cách điện cao tại cửa transistor. Nếu cửa nổi không có điện tích thì nó không ảnh hưởng gì tới cực cửa điều khiển và transistor hoạt động như bình thường. Tức là khi dây từ được kích hoạt (cực cửa có điện thế dương) thì transistor thông, cực máng và nguồn được nối với nhau qua kênh dẫn và dây bit có mức lôgic 1. Nếu cửa nổi có các điện tử trong đó với điện tích âm, chúng sẽ ngăn trường điện điều khiển của cực cửa điều khiển và dù dây từ có được kích hoạt thì cũng không thể phát ra trường đủ mạnh với cực cửa điều khiển để làm thông transistor. Lúc này đường bit không được nối với nguồn chuẩn và ô nhớ coi như được giữ giá trị lôgic 0. Việc nạp các điện tử vào vùng cửa nổi, tức là tạo ra các ô nhớ có giá trị lôgic 0, được thực hiện bởi các xung điện có độ dài cỡ 50ms và độ lớn +20V khi đặt vào giữa cực cửa và máng. Lúc đó những điện tích mang có năng lượng lớn sẽ đi qua lớp cách điện giữa đế (substrate) và cửa nổi. Chúng tích tụ trong vùng cửa nổi và được giữ ở đây sau khi xung chương trình tắt. Đó là do cửa nổi được cách điện cao với xung quanh và các điện tử không còn đủ năng lượng sau khi lạnh đi, để có thể vượt ra ngoài lớp cách điện đó nữa. Chúng sẽ được giữ ở đây trong một thời gian rất dài (ít nhất là 10 năm). Để xoá các thông tin, tức là làm mất các điện tích điện tử trong vùng cửa nổi, phải chiếu ánh sáng tử ngoại UV vào chip nhớ. Những điện tử ở đây lúc này hấp thụ năng lượng nhảy lên các mức năng lượng cao, chúng sẽ rời cửa nổi như cách thâm nhập vào đó. Trong chip EPROM có một *cửa sổ làm bằng thuỷ tinh thạch anh* chỉ để cho ánh sáng tử ngoại đi qua khi cần xoá số liệu trong bộ nhớ. Với các EEPROM, việc nạp các điện tử cho cửa nổi được thực hiện như cách ở EPROM. Bằng một xung điện thế tương đối dài, các điện tích mang năng lượng cao được phát ra trong đế sẽ thấm qua lớp cửa ôxit và tích tụ trong cửa nổi. Để xoá EEPROM, một lớp kênh màng mỏng ôxit giữa vùng cửa nổi trải xuống dưới đế và cực máng giữ vai trò quan trọng. Các lớp cách điện không thể là lí tưởng được, các điện tích mang có thể thấm qua lớp phân cách với một xác suất thấp. Xác suất này tăng lên khi bề dày của lớp giảm đi và điện thế giữa hai điện cực ở hai mặt lớp cách điện tăng lên. Muốn phóng các điện tích trong vùng cửa nổi, một điện thế (-20V) được đặt vào cực cửa điều khiển và cực máng. Lúc này các điện tử âm trong cửa nổi được chảy về cực máng qua kênh màng mỏng ô-xit và số liệu lưu giữ được xoá đi.

Các chip ROM hiện nay có thời gian thâm nhập cỡ từ 120 ns đến 150 ns.

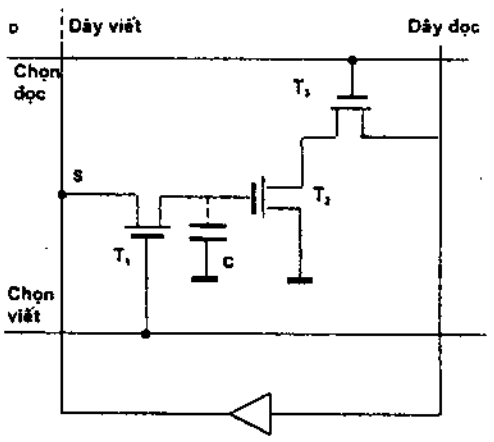
Các chip RAM (Random Access Memory) là loại bộ nhớ có thể ghi/đọc được. Đây là loại *bộ nhớ không cố định* (volatile) tức là thông tin trữ trong nó sẽ bị mất đi khi cắt nguồn điện nuôi vi mạch. RAM bán dẫn có 2 loại: *RAM tĩnh*, được viết tắt là SRAM (static RAM) và *RAM động*, được viết tắt là DRAM (dinamic RAM).

Một ô nhớ SRAM lưu trữ thông tin bởi trạng thái của một mạch lật (flip-flop) với lối ra của nó ở một trong hai mức lôgic cao hoặc thấp. Thuật ngữ "tĩnh" chỉ ra rằng khi nguồn nuôi chưa bị cắt thì nội dung của ô nhớ vẫn được giữ nguyên. Trong khi đó, một ô nhớ của DRAM lưu trữ thông tin bằng mức điện áp trên một tụ điện có điện dung C (thí dụ, mức điện áp cao trên tụ được coi là giá trị thông tin 1, mức thấp là 0). Mức điện áp U_C này được quyết định bởi lượng điện tích Q được nạp trên tụ ($U_C = Q/C$). Nhưng lượng điện tích đó lại bị biến đổi theo thời gian do sự phóng hoặc nạp điện qua các điện trở khoá (có giá trị không thể bằng vô cùng). Vì vậy phải định kì nạp lại giá trị ban đầu cho tụ điện, nếu không điện áp trên tụ sẽ bị biến đổi và làm đổi mức lôgic lưu trữ tại đó. Thuật ngữ "động" hàm ý rằng mặc dù chip nhớ chưa bị cắt điện nuôi nhưng vẫn phải thường xuyên hồi phục thông tin trong nó. Quá trình hồi phục đó gọi là "làm tươi" ô nhớ. Tất nhiên việc viết hay đọc một ô nhớ DRAM cũng chính là việc làm tươi ô nhớ đó. Hình 4.4 cho sơ đồ cấu trúc của một phần tử nhớ SRAM dùng mạch lật với các transistor MOS. Việc ghi và đọc thông tin vào ô nhớ này được thực hiện bằng các dây tín hiệu địa chỉ mở thông 2 transistor T_5 và T_6. Lúc này thông tin từ trong ô nhớ có thể được đọc từ dây

bit hoặc được ghi vào qua đó. Hình 4.5 là cấu trúc của một phần tử nhớ DRAM. Bình thường cả 2 tín hiệu chọn T_1 và T_3 đều không có. Tụ C sẽ giữ điện tích mà nó có. Điện tích này giảm (hoặc tăng) dần qua các điện trở của các transistor khoá. Khi truyền số liệu đến dây ghi tín hiệu cần ghi và tín hiệu chọn T_1 thì tụ C sẽ được nạp (hoặc phóng) qua T_1. Ta đã ghi được thông tin vào ô nhớ. Để đọc thông tin ra, truyền tín hiệu chọn mở T_3, trên dây đọc sẽ có điện thế tương ứng với điện thế trên tụ C. Nếu truyền tín hiệu chọn đồng thời đến T_1 và T_3 thì điện thế đọc được đưa về ghi lại vào C sau khi đã được khuếch đại lên. Đây chính là việc làm tươi ô nhớ.



Hình 4.4 Phần tử nhớ SRAM.



Hình 4.5 Phần tử nhớ DRAM.

4.1.2. Tổ chức của một chip nhớ

Các chip nhớ được chế tạo gồm các bộ phận sau:

Một ma trận nhớ gồm các ô nhớ, mỗi ô lưu trữ 1 bit thông tin

Các mạch lôgic giải mã địa chỉ ô nhớ.

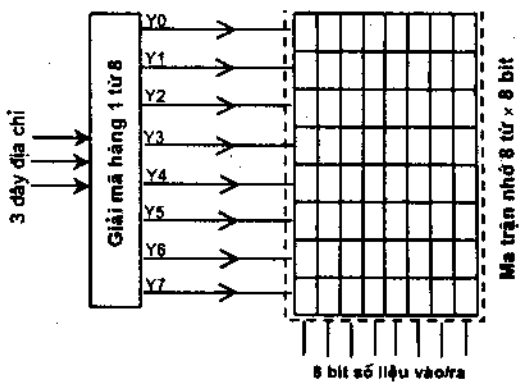
Các mạch lôgic cho phép đọc hoặc viết ô nhớ.

Các mạch đệm vào/ra, mở rộng địa chỉ v.v...

Ma-trận nhớ được tổ chức theo một trong hai cách:

Tổ chức theo *từ với sự lựa chọn tuyến tính*. Một ma trận như vậy có số hàng W bằng số lượng từ nhớ, số cột B bằng số bit trong một từ. Vậy cần 1 bộ giải mã hàng chọ phép chọn một trong W từ. Hình 4.6 là thí dụ với ma trận gồm 8 từ (W = 8), mỗi từ dài 8 bit (B = 8). Ở đây chỉ cần 1 bộ giải mã 3 đường địa chỉ để chọn 1 trong 8 hàng. Phương pháp này ít được dùng vì khi tổng số từ W lớn thì cần một bộ giải mã quá lớn.

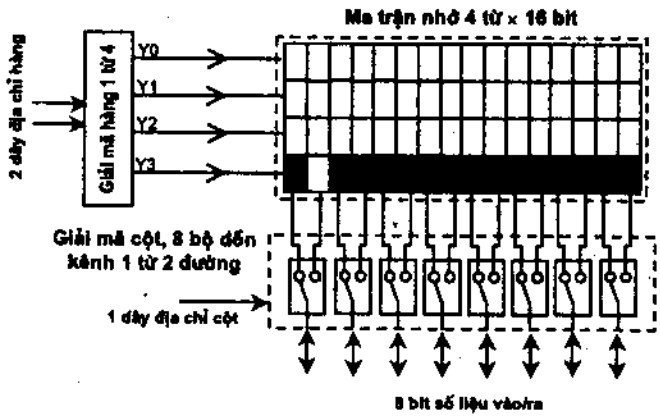
Tổ chức theo kiểu *giải mã 2 bước*. Ở đây, tổng số bit trong một hàng của ma trận tạo nên một từ vật lí. Một từ vật lí này bao gồm S từ



Hình 4.6 Tổ chức ma trận nhớ theo từ với sự chọn tuyến tính.

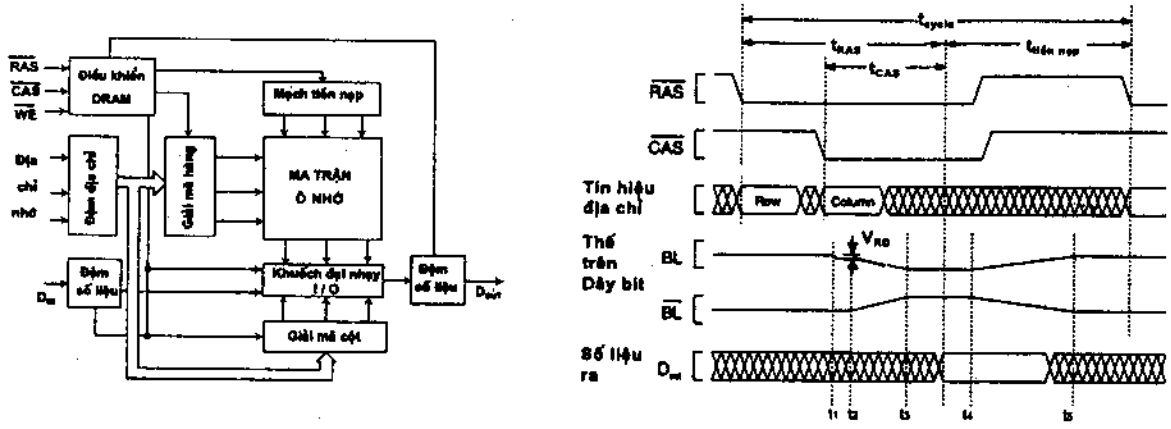
lôgic cần truy nhập đọc/viết, mỗi từ lôgic như trên đã nói gồm B bit. Như vậy cần hai bộ giải mã: bộ giải mã hàng chọn một trong W từ vật lí và bộ giải mã cột gồm B bộ dồn kênh 1 từ S đường.

Hình 4.7 là thí dụ với ma trận 8 × 8 trong hình 3.6 được tổ chức lại thành ma trận 4 từ × 16 bit. So với cách tổ chức trên, ở đây thay vì cho bộ giải mã 1 từ 8 với 3 dây địa chỉ thì nay là bộ giải mã hàng 1 từ 4 chỉ cần 2 dây địa chỉ. Bộ giải mã cột ở đây là 8 bộ dồn kênh 1 từ 2 đường , mỗi đường dẫn đến 1 bit trong từ lôgic. Trên hình ta thấy 8 ô nhớ tô đậm tạo thành 1 từ lôgic, 8 ô nhớ có đường gạch chéo tạo thành từ lôgic thứ 2 trong từ vật lí thứ 3 gồm 16 bit nhớ.



Hình 4.7 Tổ chức ma trận nhớ theo kiểu giải mã 2 bước.

Hình 4.8 là sơ đồ khối tổ chức của một chip nhớ trong đó các chân tín hiệu thâm nhập hàng $\overline{RAS}$ (row access strobe) và thâm nhập cột $\overline{CAS}$ (column access strobe) khi ở mức tích cực thấp thì DRAM nhận địa chỉ được đặt vào và coi chúng lần lượt là địa chỉ hàng hoặc địa chỉ cột. Chân $\overline{WE}$ khi ở mức tích cực thấp cho phép viết vào chip nhớ.



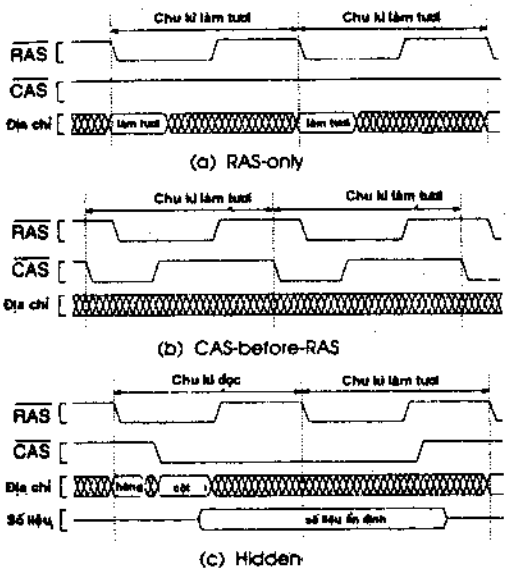
Hình 4.8 Sơ đồ cấu tạo và giản đồ thời gian của các tín hiệu trong một chip nhớ.

Hình 4.8 bên phải là giản đồ thời gian của các tín hiệu trong quá trình thâm nhập ô nhớ. Để tăng độ tin cậy của hệ thống, trong chip có mạch *tiền nạp* (điện) có nhiệm vụ nạp cho thế trên các đường dây bit luôn bằng 1/2 V_{CC} trước khi cho phép thâm nhập ô nhớ. Thời gian thâm nhập chip nhớ T_{cycle} bằng tổng các thời gian thâm nhập hàng T_{RAS}, thời gian thâm nhập cột T_{CAS} và thời gian tiền nạp $T_{tiền\ nạp}$.

Do giá thành rẻ và tiêu thụ năng lượng thấp nên DRAM chiếm một dung lượng lớn trong bộ nhớ chính. Tuy nhiên tốc độ truy cập của DRAM lại chậm hơn SRAM. Vì vậy có 2 vấn đề phải được quan tâm tới các chip nhớ này là các chế độ hoạt động nhanh và các chế độ làm tươi.

Hai kĩ thuật thông dụng nhất để giảm các trạng thái đợi do tốc độ truy cập chậm của DRAM so với tốc độ của vi xử lí là *chế độ trang* và *chế độ cột tĩnh*.

Vì số liệu ổn định sẽ chỉ có ở lối ra sau khi tín hiệu $\overline{RAS}$ xuống mức thấp với thời gian cỡ nào đó (thí dụ 100 ns) nên trước khi một hàng khác được xâm nhập thì tín hiệu này phải được nâng lên mức cao và mất một khoảng thời gian giữ tại đó. Thời gian này chính là thời gian tiền nạp (thí dụ chừng 80 ns). Điều này làm cho tổng thời gian thâm nhập chip sẽ tăng lên.Vậy suy ra rằng nếu như các từ số liệu được đọc/viết tới các ô nhớ trên cùng một trang (hàng) thì không cần thời gian tiền nạp và lúc đó địa chỉ hàng của chúng là như nhau và không mất thời gian gửi tín hiệu $\overline{RAS}$ nữa. Bằng một số chip điều khiển DRAM thích hợp có thể cho phép đọc/viết số liệu theo trang không có trạng thái đợi. Ở chế độ cột tĩnh, trong thời gian thâm nhập một hàng, chip điều khiển tiến hành một chuỗi các tín hiệu định địa chỉ hàng $\overline{RAS}$ và cột $\overline{CAS}$ bình thường. Nếu địa chỉ tiếp theo mà chip điều khiển gửi ra có cùng hàng thì một bộ so sánh bên ngoài sẽ báo cho chip điều khiển. Lúc này chip sẽ giữ các tín hiệu $\overline{RAS}$ và $\overline{CAS}$ ở mức thấp và gửi ra địa chỉ cột tới các lối vào địa chỉ của DRAM. Ngay khi vi xử lí tiếp tục thâm nhập vùng nhớ trong cùng một trang (hàng) chip điều khiển sẽ giữ các dây này ở mức thấp và gửi ra chỉ các tín hiệu địa chỉ cột tới DRAM. Chế độ cột tĩnh thực hiện khó hơn chế độ trang nhưng chạy nhanh hơn vì không đòi hỏi xung strobe $\overline{CAS}$. Hiện thông dụng có 3 phương pháp làm tươi DRAM là: chỉ kích hoạt tín hiệu giải mã hàng (RAS-only), kích hoạt tín hiệu giải mã cột trước hàng (CAS-before-RAS) và làm tươi ẩn (hidden). Giản đồ xung của các phương pháp làm tươi như hình 4.9. Phương pháp đầu thực hiện những chu trình đọc giả để làm tươi nhờ các mạch điện bên ngoài. Hai phương pháp sau cho phép hoạt động nhanh hơn và mạch làm tươi được tích hợp ngay trong chip nhớ.



Hình 4.9 Giản đồ thời gian của 3 phương pháp làm tươi DRAM.

4.1.3. Tổ chức bộ nhớ vật lí trong máy tính

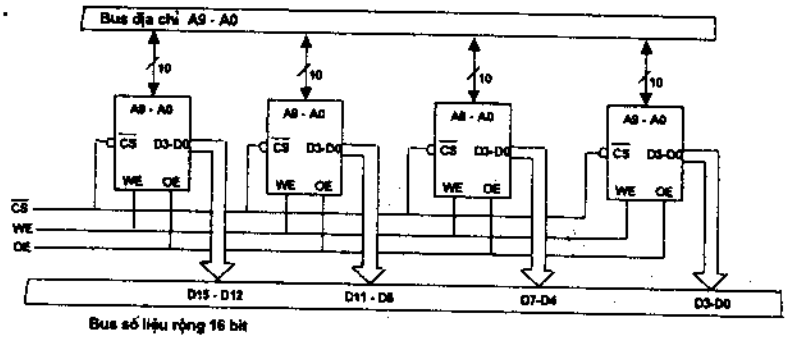
Độ rộng bus địa chỉ của CPU và kiến trúc của bản mạch chính quyết định dung lượng bộ nhớ vật lí của máy vi tính. Thí dụ, 8086/8088 có bus địa chỉ rộng 20 bit nên có thể sử dụng tối đa 2^{20} = 1 Mbyte nhớ vật lí; 80266 có 24 đường địa chỉ nên quản lí được tối đa 16 Mbyte nhớ trong khi các vi xử lí từ 80386 có tới 32 đường địa chỉ nên có thể quản lí được tới 4 Gbyte nhớ v.v...

Bộ nhớ vật lí trong máy tính được thiết kế trên cơ sở ghép các chip nhớ với nhau. Mỗi chip nhớ có số từ và độ dài từ nhớ hữu hạn. Một chip nhớ được tổ chức theo *n* từ nhớ, mỗi từ gồm *m* bit có dung lượng là *m.n* bit. Các chip thường được tổ chức theo bit (m = 1), theo nibble (m = 4) và theo byte (m = 8). Thí dụ, chip 256k x 1 có 256k từ nhớ, mỗi từ 1 bit; chip 32k x 8 có 32k từ nhớ, mỗi từ 8 bit. Nếu độ dài từ *m* bằng đúng bề rộng ô nhớ (thí dụ 8 bit = 1 byte trong máy IBM/PC) và số từ nhớ *n* của chip nhớ bằng đúng không gian địa chỉ mà vi xử lí quản lí được thì

vấn đề nối giữa vi xử lý và chip nhớ rất đơn giản. Các đường dây địa chỉ lối ra của CPU sẽ được nối với các đường dây địa chỉ tương ứng lối vào của chip. Trong thực tế có khác, các chip nhớ thường có số từ n cũng như độ dài từ m khác nhau nên phải ghép nhiều chip với nhau để đủ dung lượng nhớ cần thiết. Lúc đó phải tính đến các phương pháp ghép nối các chip để tạo thành bộ nhớ có số từ cũng như độ dài từ cho đúng yêu cầu của hệ xử lí.

a, Tăng độ dài từ dữ liệu

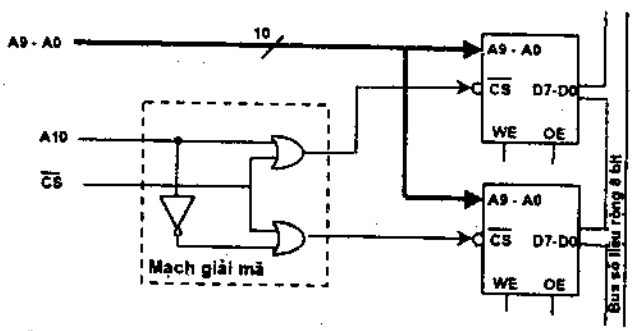
Nếu độ dài từ của một chip là m bit thì với bus số liệu rộng $k.m$ bit ta chỉ việc ghép k chip nhớ trên là được, với điều kiện các tín hiệu chọn chip $\overline{CS}$ trên k chip đó phải được kích hoạt đồng thời. Hình 4.10 là một thí dụ ghép 4 chip 1k x 4 để có bộ nhớ 1k x 16 cho vi xử lí có bus dữ liệu rộng 16 bit.



Hình 4.10 Bộ nhớ 1k x 16 bit.

b, Tăng số ô nhớ

Nếu dùng 2^k chip nhớ, mỗi chip có dung lượng n.m bit, ta có thể tăng số từ nhớ (mỗi từ dài m bit) lên 2^k lần bằng bộ giải mã địa chỉ có k đường địa chỉ vào chọn 1 trong số 2^k đường ra. Hình 4.11 là sơ đồ thí dụ đơn giản của bộ nhớ có dung lượng 2k x 8 được thiết kế từ 2 chip nhớ (= 2^1 = 2, k = 1) có dung lượng 1k x 8. Bộ giải mã 1 → 2 được thực hiện qua cổng NOT, tức là chỉ có 1 trong 2 chip nhớ luân phiên được chọn ở từng thời điểm. 1024 ô nhớ của chip phía trên (có địa chỉ từ 000h ÷ 3FFh) được chọn khi A10 = 0 và 1024 ô nhớ của chip bên dưới (có địa chỉ 400h ÷ 7FFh) được chọn khi A10 = 1.



Hình 4.11 Tăng dung lượng bộ nhớ lên gấp hai.

c, Các mô-đun nhớ

Bộ nhớ DRAM thường được thiết kế thành các mô-đun nhớ cài đặt ngay trên bản mạch chính. Đó là các "thanh" nhớ 1 hàng chân SIMM và 2 hàng chân DIMM RAM v.v... Những máy tính tương thích IBM có 2 kiểu SIMM, 30 chân (9 bit dữ liệu) và 72 chân (36 bit dữ liệu). Những máy dùng Pentium lại phổ biến các DIMM với 169 chân (64 bit dữ liệu không kiểm tra chẵn lẻ) và 72 chân (có kiểm tra chẵn lẻ).

Tốc độ truy cập của các mô-đun được biểu thị bằng đơn vị thời gian na-nô giây. Hiện trên thị trường có sẵn các loại có tốc độ truy cập từ 120 đến 60 ns.

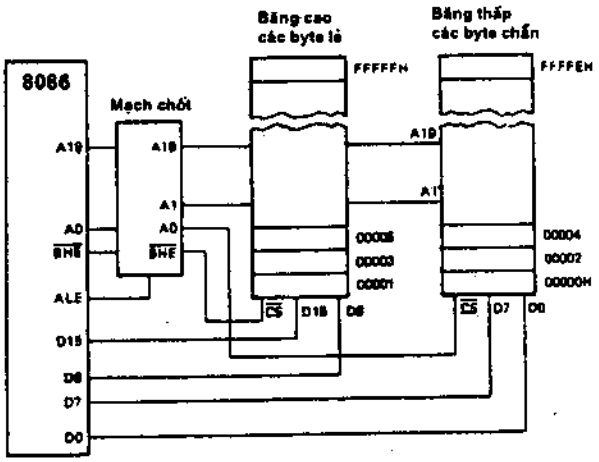
Các chip DRAM trong bộ nhớ chính hầu hết là sử dụng loại FPM RAM (Fast Page-Mode RAM) có tốc độ làm việc trên bus không quá 30MHz. Chúng không thể làm việc tốt với các CPU hiện nay có tốc độ xử lí nhanh hơn 66MHz. Vì vậy, thời gian gần đây đã ra đời loại RAM

ghi trước dữ liệu EDO RAM (Extended-Data-Out RAM). Đây là những SIMM và DIMM đặc biệt cho phép định giờ trùng lặp giữa những truy xuất liên tiếp. Thiết kế của loại này dựa trên cơ sở thống kê thực tế sau: một khi CPU cần truy xuất dữ liệu ở một ô nhớ nào đó thì có nhiều khả năng nó cũng sẽ cần truy xuất tại các ô nhớ có địa chỉ lân cận. Để có được các địa chỉ ô nhớ mới này nhanh hơn, EDO RAM được cấu tạo sao cho có thể giữ được thông tin về địa chỉ của các ô nhớ trước. EDO RAM cũng là một loại FPM RAM nhưng nó có thể lưu trữ dữ liệu tại đầu ra, giữ cho chúng đủ lâu để tín hiệu có thể tới được CPU một cách đảm bảo. EDO RAM cho phép hiệu suất của hệ thống được tăng từ 15% đến 40%. Một nhược điểm của EDO RAM là không có tính kiểm tra chẵn lẻ và không hỗ trợ các bản mạch chính có tốc độ bus cao hơn 66 MHz. Gần đây đã xuất hiện loại mô-đun SDRAM tương tự như EDO RAM nhưng cho phép truyền dữ liệu hàng loạt ở tốc độ rất cao bằng cách sử dụng một ghép nối nhịp đồng hồ tốc độ cao. Những SDRAM này cho phép chạy với tốc độ bus lên tới trên 100 MHz.

4.1.4. Tổ chức băng nhớ

Tuỳ thuộc vào độ rộng bus dữ liệu và yêu cầu cụ thể của việc truy xuất bộ nhớ mà bộ nhớ cũng được tổ chức theo băng nhớ.

Thí dụ, về cách tổ chức băng nhớ trong bộ nhớ máy tính 8086. Vì 8086 là vi xử lí có bus dữ liệu 16 bit (rộng 2 byte) nên về nguyên tắc nó truy xuất các từ (word) có địa chỉ chẵn chỉ trong 1 chu kì bus trong khi truy xuất các từ có địa chỉ lẻ phải mất 2 chu kì bus. Để đọc/viết 1 từ chỉ trong 1 chu kì máy, bộ nhớ 1Mbyte = 1 048 576 byte được chia thành hai băng, mỗi băng 524 288 byte như hình 4.12.



Địa chỉ	Loại số liệu	$\overline{BHE}$	A0	Số chu kì bus	Đường số liệu được dùng
0000	BYTE	1	0	1	D0-D7
0000	WORD	0	0	1	D0-D15
0001	BYTE	0	1	1	D7-D15
0001	WORD	0	1	thứ nhất	D0-D7
		1	0	thứ hai	D7-D15

Hình 4.12 Chia bộ nhớ thành 2 băng để truy xuất từ.

Các đường số liệu của băng chẵn được nối tới 8 đường thấp D7-D0 của bus dữ liệu 8086. Các đường số liệu của băng lẻ được nối tới 8 đường cao D15-D8. Đường địa chỉ A0 được dùng để cho phép (enable) băng thấp khi nó ở mức lôgic 0. Các đường địa chỉ còn lại từ A1 đến A19 được dùng để chọn địa chỉ cần truy nhập trong băng. Tín hiệu $\overline{BHE}$ được đưa ra từ 8086 cùng một thời gian với tín hiệu địa chỉ qua mạch chốt nhờ xung chốt địa chỉ ALE. Bảng trên hình cũng chỉ ra các mức lôgic của A0 và $\overline{BHE}$ với các loại thâm nhập bộ nhớ khác nhau. Nếu đọc/viết 1 byte từ địa chỉ chẵn (thí dụ lệnh được thực hiện là MOV AH:DS:BYTE PTR[0000]) A0 sẽ bằng 0 và $\overline{BHE}$ =1; băng thấp sẽ được phép và băng lẻ cao sẽ bị cấm và vi xử lí sẽ tự động truyền byte số liệu từ các đường số liệu thấp tới AH trong thanh ghi AX. Nếu truy nhập 1 từ (thí dụ,

lệnh MOV AX,DS:WORD PTR[0000]), cả hai A0 và $\overline{BHE}$ đều bằng 0 và cả 2 băng sẽ được phép, byte thấp sẽ truyền vào AL, byte cao vào AH. Trường hợp khác, khi truy nhập vào các byte lẻ (thí dụ lệnh MOV AL,DS:BYTE PTR[0001]), thì A0=1 và $\overline{BHE}$ =0. Do đó băng thấp sẽ bị cấm và byte dữ liệu sẽ truyền vào băng cao (lẻ) trên các đường từ D15-D8 vào thanh ghi AL. Cuối cùng, khi truy nhập từ lẻ (thí dụ, lệnh MOV AX,DS:WORD PTR[0001]), cần phải mất 2 chu kì bus. Trong chu kì thứ nhất, 8086 sẽ ra địa chỉ 0001h, A0 = 1 và $\overline{BHE}$ =0; một byte từ địa chỉ 00001h sẽ được đọc vào 8086 qua các đường dây D15-D8 và copy vào AL. Trong chu kì bus thứ hai, 8086 sẽ gửi ra địa chỉ 00002h. Vì đây là địa chỉ chẵn nên A0 sẽ bằng 0. Tuy nhiên vì chỉ thâm nhập byte lúc này nên $\overline{BHE}$ sẽ bằng 1. Byte thứ hai sẽ được đọc vào 8086 qua các đường dữ liệu D7-D0 và copy vào AH.

Lí do chính mà A0 và $\overline{BHE}$ được thiết kế hoạt động như vậy để tránh viết những byte không mong muốn vào vùng nhớ xác định khi 8086 viết một byte. Để hiểu điều này ta thử nghĩ điều gì sẽ xảy ra khi cả 2 băng nhớ đều được phép khi viết và CPU viết một byte tới địa chỉ 00002h với lệnh MOV DS:BYTE PTR[0002],AL. Số liệu từ AL sẽ được viết vào địa chỉ 00002h như bình thường. Nhưng vì băng cao cũng được phép nên các số liệu ngẫu nhiên trên D15-D8 cũng sẽ được viết vào địa chỉ 00003h. Khi 8086 được thiết kế sao cho $\overline{BHE}$ =1 trong suốt quá trình viết byte này thì băng cao bị cấm và điều vừa kể trên sẽ không xảy ra.

4.1.5. Bộ nhớ cache

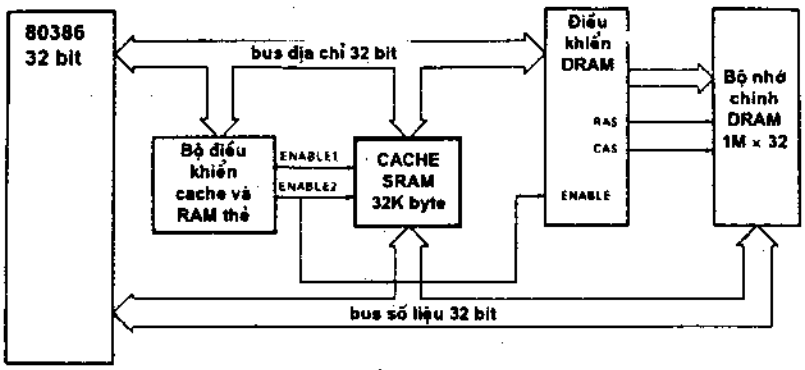
Ta biết SRAM được chế tạo từ các mạch nhớ bán dẫn là các mạch lật, có tốc độ truy cập nhanh nhưng tiêu tốn năng lượng và giá thành cao. Vì vậy DRAM được chế tạo trên nguyên tắc nhớ các điện tích nạp trên tụ điện, tiêu thụ công suất thấp và giá rẻ đã được sử dụng làm bộ nhớ chính. Tuy vậy, DRAM có tốc độ truy cập lại chậm hơn so với SRAM. Để tăng hiệu suất sử dụng DRAM có nhiều phương pháp nhằm làm giảm thời gian làm tươi (refresh) như các phương pháp xâm nhập cột tĩnh, phương pháp đan xen với các trang v.v... đã được áp dụng. Phương pháp sử dụng bộ nhớ cache là một cách tăng hiệu suất bộ nhớ rất phổ biến hiện nay.

Cache là một lượng SRAM nhỏ (có tốc độ truy cập nhanh) cùng bộ điều khiển được đặt giữa CPU và bộ nhớ chính gồm các DRAM (có tốc độ truy cập chậm hơn) nhằm làm tăng hiệu suất của hệ thống nhớ. Cache chứa các thông tin mới vừa được CPU sử dụng gần đây nhất. Khi CPU đọc/viết dữ liệu nó sẽ đưa ra địa chỉ của chương trình tới bộ điều khiển cache. Nếu xác định được rằng dữ liệu có địa chỉ đó hiện đã được sao vào cache - *gọi là trúng cache* (cache hit) - thì CPU sẽ thực hiện một chu kì bus đọc/viết dữ liệu không có chu kì đợi trên cache (vì SRAM có thời gian xâm nhập nhanh). Ngược lại, khi thấy rằng không có địa chỉ của dữ liệu cần truy nhập trong cache - *gọi là trật cache* (cache miss)- CPU sẽ truy cập bộ nhớ chính với một vài chu kì đợi (vì DRAM có thời gian thâm nhập chậm hơn). Hiệu suất của cache phụ thuộc vào tỉ số *cache hit* trên *cache miss*. Với các hệ thống cache hiện nay, tỉ số này thường đạt tới hơn 90%. Cache được tổ chức thành các *dòng cache* (cache line), mỗi dòng có thể nhận thông tin từ bộ nhớ chính được lưu trữ (đệm) lại trong một hoạt động đọc/viết. Kích thước của một dòng tuỳ thuộc vào dung lượng dữ liệu trong CPU hoặc dung lượng cache cấp 1 trong CPU (là cache được tích hợp ngay trong chip vi xử lí). Trong 80386, mỗi dòng cache rộng 32 bit (4byte), với 80486 là 128 bit (16 byte) và Pentium là 256 bit (32 byte). Mỗi dòng cache luôn được nạp đầy dữ liệu trong 1 lần CPU truy cập bộ nhớ, không phụ thuộc vào số byte đọc/viết của CPU trong lần truy cập bộ nhớ. Kích thước bộ nhớ cache càng lớn càng làm tăng hiệu suất cache do lượng thông tin được lưu trữ trong mỗi lần xảy ra cache miss càng lớn hơn. Vì các lệnh thường gồm các địa chỉ liên tục với nhau nên kích thước dòng càng lớn càng có thể có một tỉ lệ cache hit cao.

Có 3 phương pháp thiết kế cache:

- *Write-Through*: Trong đó, khi xảy ra cahe hit, bộ điều khiển cache cập nhật nội dung của bộ nhớ cache, đồng thời cũng chuyển hoạt động ghi của CPU tới bộ nhớ chính để đảm bảo tính nhất quán giữa cache và bộ nhớ chính. Thiết kế này đơn giản nhưng hiệu suất thấp vì phải truy cập bộ nhớ chính có tốc độ chậm.
- *Posted-Write-Through*: Hầu hết các hệ thống cache sử dụng phương pháp này. Nếu xảy ra cache hit, *bộ điều khiển cache* sẽ viết từ mới tới cache với *chu kì bus không có trạng thái đợi* và báo ngay cho 80386 biết rằng việc viết bộ nhớ đã xong. Nếu xảy ra cache miss, nó cũng báo như vậy cho CPU. Toàn bộ tác vụ được lưu trong một vùng đệm để rồi sau đó bộ điều khiển cache sẽ viết các từ dữ liệu đó vào bộ nhớ chính. Quá trình viết này là trong suốt (transparent) đối với vi xử lí trừ khi bộ nhớ chính vẫn còn tham gia vào hoạt động viết trước đó. Như vậy, các bộ phận khác sẽ không được sử dụng bus hệ thống cho đến khi quá trình write-through được hoàn thành.
- *Write-back*: Bộ nhớ sẽ chỉ được cập nhật khi cần thiết. Bộ nhớ chính sẽ được ghi khi một truy xuất đọc được thực hiện bởi một đơn vị làm chủ bus khác đối với một địa chỉ bộ nhớ mà hiện chứa dữ liệu là cũ (so với dữ liệu tương ứng trong bộ nhớ cache), hay khi một dòng cache chứa thông tin được cập nhật sắp sửa bị ghi đè lên vì thông tin này là cũ.

Mỗi dòng cache được nhận diện bằng một *thẻ bài* (tag), đó là nơi cất giữ địa chỉ của dữ liệu cũng như thông tin tình trạng bổ sung. Trong cache cấp 2 (là cache được lắp trên bản mạch chính), các thẻ này nằm trong một bộ nhớ riêng biệt tại *bộ điều khiển cache* (có tốc độ truy cập thậm chí còn nhanh hơn bộ nhớ cache). Trong khi truy tìm một địa chỉ trong bộ nhớ cache, địa chỉ này không chỉ được đọc ra từ thẻ mà còn được so sánh với địa chỉ của truy xuất cụ thể bằng cách sử dụng một bộ so sánh. Tất nhiên việc này mất thêm một khoảng thời gian nhưng được bù lại bởi tốc độ truy cập nhanh của cache. *Bộ điều khiển cache* nằm ngay trong CPU (với cache cấp 1) hoặc là một vi điều khiển được lắp trên bản mạch chính (với cache cấp 2). Bộ này có chức năng điều khiển việc liên lạc với CPU cũng như các thông tin đến và đi được đệm trữ trong các dòng cache. Hình 4.13 là một thí dụ sơ đồ tổ chức bộ nhớ chính trong 80386 khi sử dụng cache.

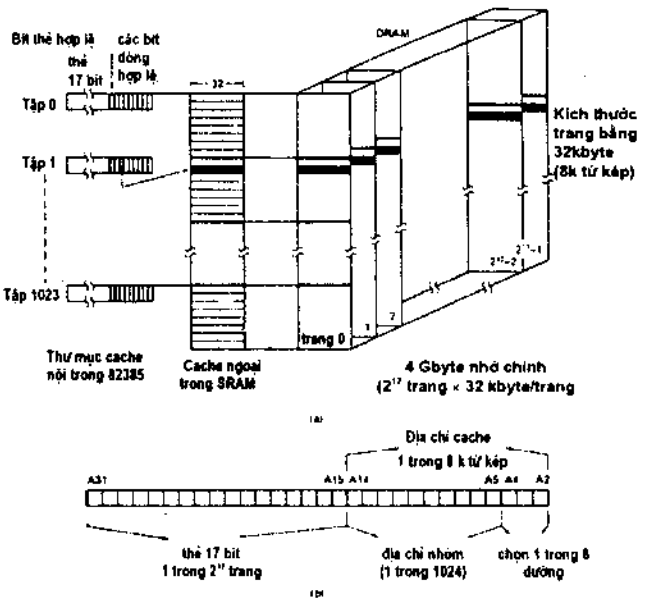


Hình 4.13 Bộ nhớ chính sử dụng RAM cache.

Để theo dõi vùng nhớ chính nào đang hiện diện trong RAM cache, bộ điều khiển cache sử dụng *thư mục cache* là một bộ nhớ lắp trong vi mạch điều khiển cache. Mỗi vùng trong cache được đại diện bởi một điểm vào trong thư mục. Định dạng chính xác cho điểm vào thư mục phụ thuộc vào các sơ đồ cụ thể. Có 3 sơ đồ cơ bản cho cách thức tổ chức cache như sau.

a, Cache ánh xạ trực tiếp (direct mapped cache)

Có thể dùng sơ đồ hình 4.14 làm một thí dụ cho bộ nhớ cache 32 kbyte lắp cho vi xử lí 80386. Đây là vi xử lí 32 bit địa chỉ nên nó có thể quản lí được tới 2^{32} = 4Gbyte nhớ. Nó cũng có bus dữ liệu rộng 32 bit nên có thể cho phép đọc 4 byte một lần từ bộ nhớ. Như vậy một nhóm 4 byte song song thì được gán cho 1 dòng cache. Bộ nhớ cache 32 kbyte này được phân thành 8 k dòng, mỗi dòng 4 byte. Các dòng này lại được xắp thành 1024 tập hợp (set), mỗi tập gồm 8 dòng. Bộ điều khiển cache coi 4 Gbyte nhớ như 2^{17} = 131 072 trang nhớ, mỗi trang bằng 32 kbyte chính bằng kích thước của bộ nhớ cache. Việc ánh xạ trực tiếp ở đây có nghĩa là một dòng được đánh số trong một trang nhớ ở bộ nhớ chính sẽ luôn được copy tới cùng một dòng tương ứng trong cache. Thư mục bên trái hình được dùng để theo dõi dòng nào trong bộ nhớ chính hiện đang được copy vào bộ nhớ cache. Thư mục chứa một điểm vào 26 bit cho mỗi tập 8 dòng trong cache. 17 bit cao của điểm vào thư mục là thẻ bài (tag). Thẻ dùng để nhận dạng trang nhớ trong bộ nhớ chính mà một dòng hoặc một tập các dòng trong cache được copy vào. Mỗi điểm vào thư mục cũng có một bit *thẻ hợp lệ* (tag valid bit) và tám bit *dòng hợp lệ* (line valid bit), mỗi bit cho một dòng trong tập. Khi 80386 gửi ra một địa chỉ 32 bit để đọc một từ từ bộ nhớ, các bit địa chỉ A31-A15 đại diện cho một trang nhớ chính, các bit từ A14-A5 đại diện cho tập chứa một dòng mong muốn và các bit từ A4-A2 nhận dạng số dòng trong tập chứa *từ* dữ liệu cần truy nhập. Bộ điều khiển cache trước tiên dùng các bit địa chỉ từ A14-A5 để chọn điểm vào thư mục cho tập chứa dòng được định vị. Sau đó nó so sánh với 17 bit địa chỉ từ 80386 với thẻ 17 bit được lưu trữ trong điểm vào thư mục. Nếu tương đương, nó tiếp tục kiểm tra bit *thẻ hợp lệ*. Nếu bit này được đặt, nó sẽ kiểm tra bit *dòng hợp lệ* cho dòng được định vị bởi các bit từ A4-A2. Nếu tất cả thẻ và dòng hợp lệ thì dòng dữ liệu đó đã ở trong cache và có nghĩa là đã xảy ra cache hit. Trong trường hợp này, bộ điều khiển sẽ đưa các bit địa chỉ từ A14-A2 đến bộ nhớ cache và cho phép bộ nhớ cache xuất từ cần truy nhập lên bus số liệu. Nếu 17 bit cao của địa chỉ từ 80386 không giống như thẻ trong thư mục (bit thẻ không hợp lệ) hoặc bit dòng cho dòng được định vị không hợp lệ thì hoạt động đọc là *cache miss*. Trong trường hợp này, bộ điều khiển cache sẽ gửi một địa chỉ toàn bộ từ 80386 tới bộ điều khiển DRAM của bộ nhớ chính. Bộ điều khiển DRAM sẽ điều khiển bộ nhớ chính xuất ra dữ liệu trong dòng được định vị lên bus dữ liệu. Khi dòng dữ liệu này xuất hiện, bộ điều khiển cache sẽ kích hoạt (enable) bộ nhớ cache, nhằm làm cho dòng này cũng có thể được viết vào cache cũng như vào 80386. Nó cũng đồng thời cập nhật thư mục cache để chỉ thị rằng dòng này bây giờ đã ở trong cache. Nếu dòng này hoặc một phần bất kì của nó được cần đến lần nữa, nó có thể được đọc trực tiếp từ cache.

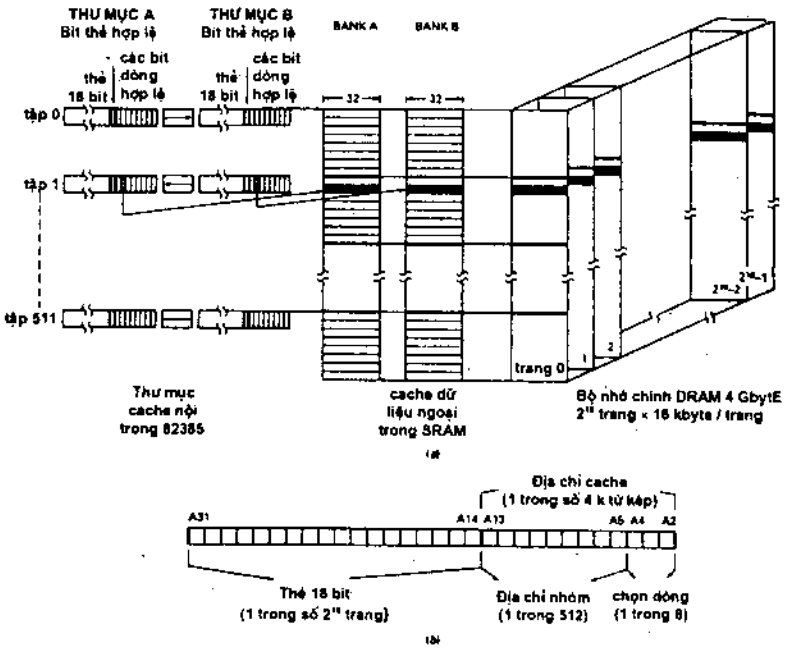


Hình 4.14 Tổ chức cache ánh xạ trực tiếp 32 kbyte.

Khi 80386 viết một từ tới bộ nhớ, bộ điều khiển cache sẽ thu tóm địa chỉ và từ dữ liệu rồi báo ngay cho 80386 rằng việc truyền đã hoàn tất. Sau đó nó sẽ kích hoạt bộ nhớ chính nhằm cho từ dữ liệu được viết tới một địa chỉ chính xác trong bộ nhớ chính. Nếu từ dữ liệu hiện diện trong cache, nó cũng vẫn được viết vào cache. Xử lí posted-write không đòi hỏi bất kì trạng thái đợi nào trừ khi bộ nhớ vẫn bận với việc viết trước đó.

b, Cache liên hợp tập hai hàng

Một khó khăn với tổ chức kể trên là khi chương trình sử dụng cùng dòng được đánh số như nhau từ 2 trang nhớ trong cùng một lúc, nó sẽ phải tráo đổi 2 dòng giữa bộ nhớ chính và cache. Để tránh điều này phải dùng sơ đồ *tổ chức cache liên hợp 2 hàng* (2-way set associative cache system) như hình 4.15. Ở đây dùng 2 bộ nhớ cache cùng với 2 thư mục cache tách biệt nhằm cho cùng các dòng từ các trang khác nhau có thể được ghi vào cùng một lúc.



Hình 4.15 Tổ chức cache liên hợp 2 dòng.

Mỗi bộ nhớ cache có kích thước bằng một nửa so với cache ánh xạ trực tiếp. Như vậy, bộ điều khiển cache coi bộ nhớ gồm 262 144 trang, mỗi trang gồm 4096 dòng. Để nhận diện một trong số các trang này, thẻ trong mỗi điểm vào thư mục cache chứa 18 bit. Mỗi điểm vào thư mục trong hệ thống này cũng có bit thẻ hợp lệ, 8 bit dòng hợp lệ và bit LRU (least recently used bit). Xét quá trình đọc dữ liệu. Khi 80386 xuất ra một địa chỉ, bộ điều khiển cache dùng các bit địa chỉ từ A13 đến A15 để chọn điểm vào thích hợp trong mỗi thư mục cache. Sau đó nó sẽ so sánh 18 bit cao của địa chỉ từ 80386 với thẻ trong mỗi điểm vào thư mục được chọn. Nếu một trong những thẻ này phù hợp, bộ điều khiển sẽ kiểm tra bit thẻ hợp lệ trong điểm vào thư mục đó. Nó cũng kiểm tra bit dòng hợp lệ cho dòng được hiện diện bởi các bit địa chỉ từ A4 đến A2. Nếu các bit này được đặt, bộ điều khiển cache sẽ xuất ra các bit địa chỉ từ A13 đến A2 tới cache liên quan với thư mục đó và cho phép cache xuất ra từ dữ liệu mong muốn trên bus dữ liệu. Nếu từ dữ liệu được tìm thấy trong cache A thì bit LRU trong điểm vào thư mục được đặt để chỉ thị rằng cache A vừa mới được sử dụng. Nếu từ dữ liệu tìm được trong cache B thì bit LRU

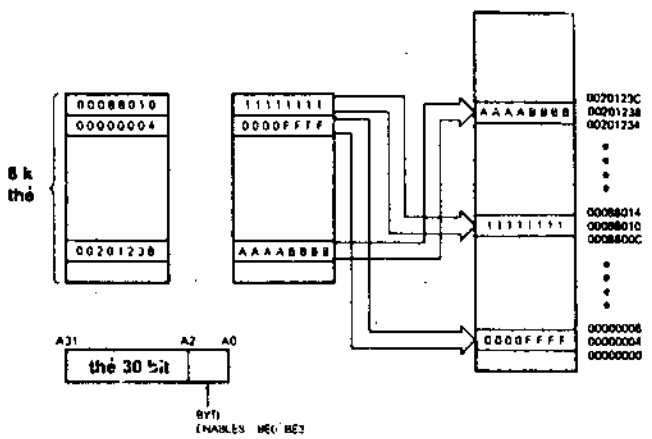
được đặt để chỉ thị rằng cache B vừa mới được sử dụng. Cơ chế này được dùng để xác định cache nào sẽ được sử dụng để giữ một dòng mới được đọc vào từ bộ nhớ chính. Khi một hoạt động đọc xảy ra cache miss, bộ điều khiển cache sẽ gửi địa chỉ và các tín hiệu điều khiển mong muốn tới bộ nhớ chính để đọc một dòng đang chứa từ dữ liệu mong muốn.Khi dòng này xuất hiện trên bus dữ liệu, bộ điều khiển cache sẽ viết nó tới cache được sử dụng lâu nhất và cập nhật điểm vào thư mục tương ứng. Nếu bộ điều khiển cache phát hiện rằng thẻ cho quá trình đọc chính xác nhưng một bit dòng hợp lệ không hợp lệ, nó sẽ đọc dòng từ bộ nhớ chính và viết nó vào bộ nhớ cache mà thư mục của nó chứa thẻ. Điều này đảm bảo rằng các dòng liền kề từ một trang trong bộ nhớ chính được kết thúc trong cùng một cache.

Đối với hoạt động viết, tổ chức cache liên hợp 2 hàng này sử dụng thiết kế kiểu posted-write-trough nói trên. Bộ điều khiển cache luôn viết từ dữ liệu ra tới bộ nhớ chính và nếu từ này hiện diện trong cache thì nó cũng sẽ cập nhật từ trong cache.

Tổ chức cahe liên hợp 4 hàng chỉ đơn thuần là sự mở rộng của tổ chức liên hợp 2 hàng. Bộ nhớ cache được chia thành 4 khối có kích thước bằng nhau và bộ nhớ chính được xem như được chia thành các trang có kích thước bằng với kích thước của mỗi khối. Tổ chức này cho phép một khả năng linh động hơn loại cache liên hợp 2 hàng.

c, Cache liên hợp hoàn toàn (Fully associative cache system)

Hình 4.16 chỉ ra sơ đồ khối của hệ thống này. Ở đây, một khối 4 byte hoặc một dòng từ bộ nhớ có thể được viết vào bất kì vùng nào trong cache.



Hình 4.16 Tổ chức cache liên hợp hoàn toàn.

Hệ thống có một bus địa chỉ 32 bit như vậy có thể định vị được 4Gbyte nhớ hay 1Giga các dòng 4 byte. Vì 1Gbyte bằng 2^{30} byte nên cần một thẻ dài 30 bit để nhận diện mỗi khối hoặc dòng được lưu trữ trong cache. Mỗi điểm vào trong thư mục như vậy phải có 30 bit cho thẻ và các bit dùng để theo dõi dòng được dùng gần đây như thế nào. Cache liên hợp hoàn toàn có ưu điểm là nó có thể giữ các dòng được đánh số như nhau trong một vài trang khác nhau cùng một thời gian. Nhược điểm của nó là 30 bit cao của mỗi địa chỉ nhớ được gửi ra bởi CPU phải được so sánh với tất cả các thẻ trong thư mục để xem liệu dòng dữ liệu đó có hiện diện trong cache hay không. Điều này làm mất thêm nhiều thời gian. Cũng vậy, khi cache loại này chứa đầy dữ liệu thì một vài thuật giải phải được sử dụng để xác định xem dòng nào sẽ bị viết đè lên khi một dòng mới phải cần được viết vào cache từ bộ nhớ chính. Thuật giải thông dụng nhất là thay thế dòng già nhất với dòng mới.

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

4.1 Hãy thiết kế mạch giải mã tạo xung chọn chip nhớ dùng các cổng NOT và cổng NAND 2 lối vào cho địa chỉ 300h.

4.2 Hãy thiết kế tổ chức bộ nhớ theo kiểu giải mã tuyến tính và giải mã 2 bước cho bộ nhớ 16 từ mỗi từ 4 bit. So sánh ?

4.3 Hãy thiết kế tổ chức bộ nhớ theo kiểu giải mã 2 bước cho bộ nhớ 1024 từ mỗi từ 8 bit. So sánh với kiểu giải mã tuyến tính ?

4.4 Giải thích tại sao một băng nhớ hoạt động theo mode trang lại có thể cho phép CPU thâm nhập nó không có trạng thái đợi ?

4.5 Phân biệt sự khác nhau giữa mode trang và mode cột tĩnh ?

4.6 Bộ điều khiển cache lưu trữ các dòng từ bộ nhớ vào cache như thế nào ?

4.7 Với một hệ thống cache ánh xạ trực tiếp, mỗi điểm vào trong thẻ RAM đại diện cho cái gì ?

4.8 Trong hệ thống cache ánh xạ trực tiếp, chỉ có một khối nhớ với một số đặc biệt có thể hiện diện trong cache mỗi lần. Trong hệ thống cache liên hợp 2 hàng người ta khắc phục vấn đề này như thế nào ?

4.9 Xem lại chương 1 để mô tả cách dùng bit chẵn lẻ để kiểm tra lỗi trong RAM của máy vi tính ?

4.10 Khi sử dụng mã Hamming để kiểm tra lỗi ở DRAM thì phải cần bao nhiêu bit cho lập mã khi muốn phát hiện và sửa một lỗi bit đơn trong từ dữ liệu dài 64 bit ?

PHẦN HAI

GHÉP NỐI MÁY VI TÍNH VỚI CÁC THIẾT BỊ NGOẠI VI

CHƯƠNG 5. CẤU TRÚC BUS

5.1. CẤU TRÚC CÁC BUS

Như đã biết, chu kì bus (hay là chu kì máy) là một quá trình xảy ra trên bus để truyền tải dữ liệu. Lượng dữ liệu tối đa được chuyển qua bus trong một khoảng thời gian nào đó gọi là tốc độ truyền tải tối đa. Tốc độ này là một nhân tố quan trọng ảnh hưởng đến hiệu năng của máy vi tính, nó được tính theo số byte được truyền tải trong 1 giây:

$$\textit{Tốc độ truyền tải tối đa} = \frac{\textit{Tốc độ bus (MHz)} \times \textit{số byte trong một lần truyền tải}}{\textit{Số chu kỳ đồng hồ cho mỗi lần truyền tải}}$$

Xem xét các mạch điện của bản mạch chính thấy rằng cả 3 thông số bên vế phải đều phụ thuộc vào cấu trúc của bus trên bản mạch. Nói cách khác, hiệu năng chung của hệ máy vi tính không những phụ thuộc vào kiểu và tốc độ của vi xử lí mà còn phụ thuộc vào cấu trúc của các bus trên bản mạch chính.

Một máy vi tính có thể có nhiều loại bus như sau:

- *Bus bộ xử lí* là các đường truyền giữa CPU và các mạch đệm trung gian. Nó thường được dùng để truyền dữ liệu giữa CPU và bus hệ thống hoặc giữa CPU và bộ nhớ cache ngoại. Bus này hoạt động với tốc độ nhanh nhất so với các loại bus khác và không bị tắc nghẽn. Nó cũng bao gồm các bus dữ liệu, địa chỉ và điều khiển. Thí dụ trong hệ thống Pentium, bus bộ xử lí có 64 đường dữ liệu, 32 đường địa chỉ và các đường điều khiển. Tốc độ bus bộ xử lí bằng tốc độ đồng hồ ngoài CPU (tốc độ đồng hồ bên trong CPU thường bằng tốc độ ngoại nhân với một hệ số nào đó). Nếu chip này chạy ở tốc độ bản mạch chính 66 MHz và có thể truyền 1 bit dữ liệu cho mỗi chu kì đồng hồ ta sẽ có tốc độ truyền dữ liệu tối đa là:

66 MHz × 64 bit = 4.224 Mbit/giây = 4.224 Mbit/giây : 8 = 528 Mbyte/giây.

- *Bus bộ nhớ* được sử dụng để truyền thông tin giữa CPU và bộ nhớ chính. Bus này hoặc là thành phần của chính bus bộ xử lí hoặc trong nhiều trường hợp được phân cách với bus bộ xử lí bằng các mạch đệm là các chip chuyên dụng. Với các hệ thống chạy ở tốc độ đồng hồ bản mạch chính cao hơn 16 MHz sẽ có một chip điều khiển bộ nhớ cho phép điều khiển sự ghép nối giữa bus bộ xử lí có tốc độ nhanh hơn và bộ nhớ chính có tốc độ truy xuất chậm hơn. Vậy thông tin truyền trên bus bộ nhớ được truyền với tốc độ chậm hơn nhiều so với thông tin trên bus bộ xử lí. Tuy nhiên độ rộng bus dữ liệu ở đây luôn được thiết kế bằng độ rộng bus dữ liệu của bus bộ xử lí.

- *Bus vào/ra* còn gọi là bus mở rộng cho phép CPU thông tin được với các thiết bị ngoại vi. Nó cho phép bổ sung vào hệ thống máy tính các thiết bị để mở rộng tính năng của PC. Các khe cắm mở rộng được nối vào bus mở rộng. Các bản mạch ghép nối được cắm vào các khe cắm này. Do đó khi nói về tiêu chuẩn cho một loại bus mở rộng nào đó cũng có nghĩa là cho các khe cắm mở rộng và card dùng với nó.

Cần lưu ý rằng trong các máy PC hiện nay, nhiều thiết bị ngoại vi được tích hợp ngay trên bản mạch chính. Thí dụ chúng có ít nhất hai bộ điều khiển chuẩn ghép nối ổ đĩa IDE (sơ và thứ cấp), bộ điều khiển đĩa mềm, hai cổng nối tiếp và một cổng song song thường nằm trong một chip riêng gọi là Super I/O chip. Nhiều hệ thống bản mạch chính còn được bổ sung thêm nhiều thành phần khác như mạch âm thanh, mạch ghép nối mạng, v.v...Trong một hệ thống như vậy, có thể các khe cắm mở rộng không cần thiết nhưng những bộ điều khiển cài sẵn và các cổng

vẫn sử dụng bus vào/ra để thông tin với CPU như bình thường giống như chúng được lắp trên các bản mạch cắm vào các khe cắm mở rộng như trước đây.

Cần phân biệt *bus vào/ra* có tốc độ xung nhịp định chuẩn 8,33MHz (tương thích với loại bus gọi là bus ISA) với *bus cục bộ* (local bus) là các hệ bus nối giữa CPU và các bộ đệm hoạt động không bị hạn chế bởi giới hạn 8,33MHz này.

Bus cục bộ, thí dụ như bus nối giữa vi xử lí với các bộ đệm số liệu và địa chỉ cùng máy phát xung nhịp đồng hồ hệ thống. Tín hiệu truyền trên bus lúc này có tần số bằng tần số hoạt động cao ở lối ra vi xử lí và không bị hạn chế bởi giới hạn 8,33 MHz được chuẩn hoá như ở bus vào/ra truyền thống. Các bus vào/ra hệ thống kiểu như ISA nối từ các bộ đệm ra tới các khe cắm mở rộng thế hệ trước (như khe cắm loại ISA) nhằm liên kết vi xử lí với các bản mạch ghép nối cắm trên chúng với điều kiện bản mạch này hoạt động ở tốc độ xung nhịp không quá 8,33 MHz. Nghĩa là nếu tần số đồng hồ hệ thống của bản mạch chính cao hơn nhiều tần số chuẩn của bus (thí dụ, tần số bản mạch chính sử dụng vi xử lí Pentium 66MHz so với tần số chuẩn của bus ISA là 8,33MHz) thì nhà thiết kế phải sử dụng các bộ chia tần có hệ số chia lớn hơn để đảm bảo có xung nhịp đúng tần số chuẩn đó trên bus vào/ra truyền thống.

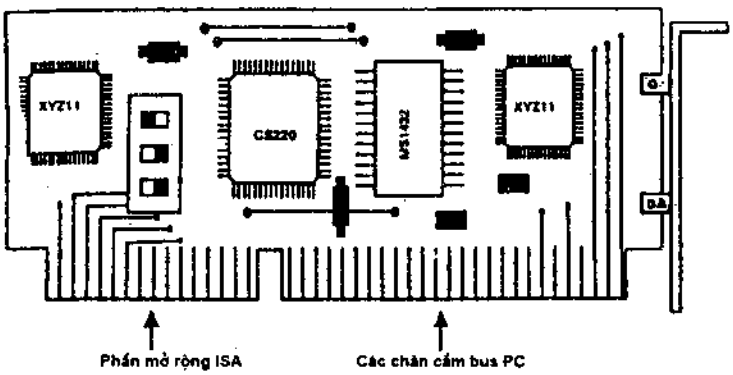
Hiện thông dụng có 6 loại ***cấu trúc bus mở rộng*** phổ biến dùng cho các mục đích ghép nối với các thiết bị ngoại vi. Tương ứng với chúng là 6 loại khe cắm mở rộng (expansion slots) và các card ghép nối cho các ứng dụng cắm trên chúng. Ngoài các chuẩn về hình dạng cơ khí, các thông số chính cho phép đánh giá chúng là độ rộng dữ liệu, số địa chỉ quản lí và tốc độ truyền tải dữ liệu. Các kênh yêu cầu ngắt và DMA cũng là những chỉ tiêu thường được xem xét.

5.1.1. Bus PC (Personal Computer bus)

Bus này ra đời phục vụ cho kiến trúc của vi xử lí Intel-8088 trước đây chạy ở 4,77MHz. Tần số xung nhịp trên bus được chuẩn hoá là 8,33 MHz. Nó có bus số liệu rộng 1byte = 8 bit, do vậy tốc độ truyền tải dữ liệu tối đa là 8,33 Mbyte/giây. Bus địa chỉ rộng 20 bit, do vậy có thể quản lí được bộ nhớ tới 1Mbyte địa chỉ. Khe cắm mở rộng nối với nó là loại 2 hàng chân, 62 tiếp điểm như đã được trình bày trong giáo trình cấu trúc máy vi tính. Các tiếp điểm được chế tạo từ bản mạch in 2 mặt có mạ vàng hoặc thiếc. Kích thước tiêu chuẩn của một card PC là: cao 106,68 mm (4,2 inch), dài 333,5 mm (13,13 inch) và rộng 12,7 mm (0,5 inch).

5.1.2. Bus ISA (Industry Standard Architecture)

Hãng IBM đã phát triển chuẩn bus ISA cho các máy tính AT dùng vi xử lí 80286. Ưu điểm của nó so với bus kể trên là có độ rộng bus số liệu lên tới 2 byte = 16 bit do đó có thể truyền được 2 byte này trong một nhịp đồng hồ hay tính ra tốc độ truyền tải tối đa có thể đạt được tới 16,66 Mbyte/giây. Bus địa chỉ cũng được mở rộng lên tới 24 bit do vậy có thể quản lí được tới 16 Mbyte địa chỉ bộ nhớ. Khe cắm ISA là một khe cắm PC cộng thêm một khe cắm 2 hàng chân nữa gồm 32 tiếp điểm, trong đó được thêm vào 8 đường số liệu và 4 đường địa chỉ cao. Do vậy, một card PC bất kì hoàn toàn có thể cắm và sử dụng được trên khe cắm ISA. Các card ISA hiện nay vẫn đang được dùng rất phổ biến trong các loại máy vi tính vì chúng có chất lượng tốt cho nhiều ứng dụng tốc độ không cần cao như: card âm thanh, card fax-modem, card I/O v.v.. giá thành các linh kiện điện tử trong các bản mạch ISA đều rất rẻ và công nghệ chế tạo đã ổn định qua nhiều năm. Hình 5.1 là bản vẽ một card ISA với 2 nhóm tiếp điểm cho khe cắm PC và khe cắm mở rộng cho chuẩn bus IAS. Bên phải là khe cắm với các chân tín hiệu.



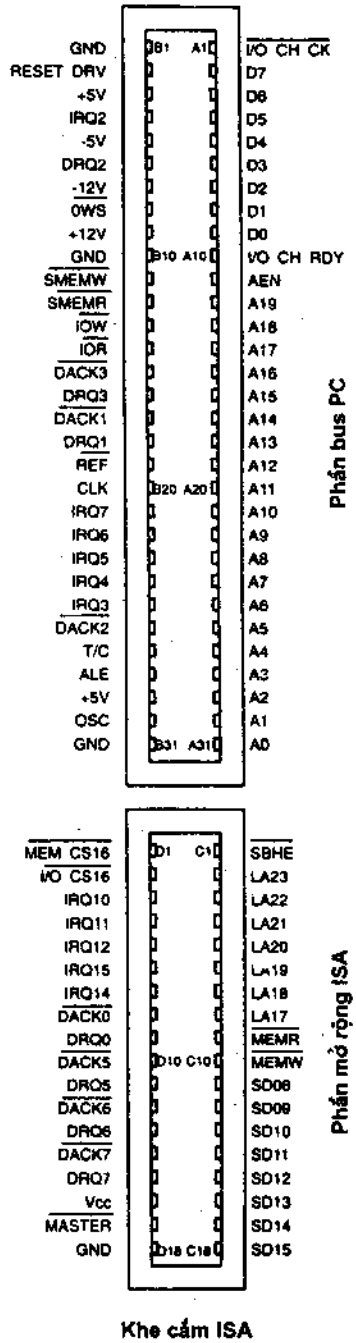
Hình 5.1 Card và khe cắm ISA.

5.1.3. Bus MCA (Microchannel Interface Architecture)

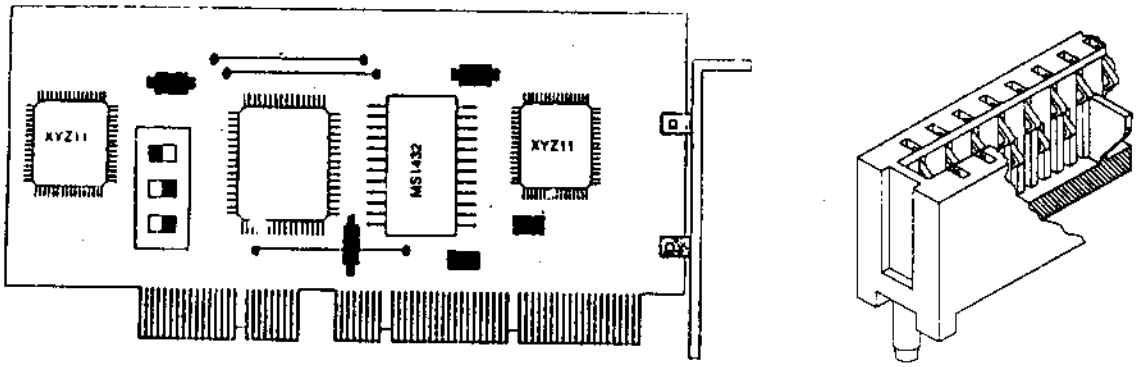
Hãng IBM đã phát triển cấu trúc bus vi kênh này cho các máy tính PS/2 của họ. Cấu trúc này thêm vào một số đặc điểm như: việc làm chủ bus được hỗ trợ tốt cho phép một bản mạch ghép nối được nắm quyền điều khiển bus từ bộ xử lí và thông tin dữ liệu với bộ nhớ chính một cách độc lập với bộ xử lí; hỗ trợ cho việc tự định cấu hình của các bản mạch ghép nối (tiền đề cho chức năng cắm là chạy Plug and play), v.v... Bus này hoàn toàn không tương thích với bus ISA và có thể hoạt động với bề rộng bus dữ liệu 16 hoặc 32 bit. Sự khác nhau chính về mặt kĩ thuật là trong khi các bus kể trên là đồng bộ thì bus MCA là một loại bus dị bộ. Nghĩa là nó được làm việc ở một tần số nhịp đồng hồ cố định không phụ thuộc vào đồng hồ hệ thống của PC. Một bus MCA có thể cho một tốc độ truyền tải tối đa lên tới 160 Mbyte/giây. Chuẩn bus MCA cho 4 kích thước card khác nhau với 4 kiểu khe cắm 16 bit, 16 bit với những phần mở rộng video, 16 bit với những phần mở rộng bộ nhớ và 32 bit.

5.1.4. Bus EISA (Extended ISA)

Chuẩn bus này được ra đời do sự cạnh tranh với bus MCA. Khe cắm EISA có 2 nấc. Nấc trên cho các tiếp điểm hoàn toàn tương thích với chuẩn ISA, do vậy một card ISA có thể cắm vào đó. Có một cái chốt đặc biệt cho phép các card EISA có thể được cắm sâu hơn nữa vào khe cắm để có thể tiếp xúc với nấc dưới có các tiếp điểm được nối với các bus số liệu và địa chỉ rộng 32 bit. Chuẩn EISA cũng có tốc độ nhịp trên bus là 8,33MHz nhưng do độ rộng bus dữ liệu là 32 bit nên tốc độ truyền tải tối đa ở đây lên tới trên 32 Mbyte/giây và bus có thể địa chỉ hoá được bộ nhớ tới 4Gbyte. Các chân tín hiệu tương ứng với ISA và EISA được chỉ ra như hình bên. Hình 5.2 bên trái là bản vẽ card EISA với 2 nấc cắm. Hình 5.2 bên phải là hình vẽ một mẫu khe cắm EISA với 2 tầng tiếp điểm khác nhau. Kích thước tiêu chuẩn của một card EISA là: cao 127 mm (5 inch), dài 333,5 mm (13,13 inch) và rộng 12,7 mm (0,5 inch).



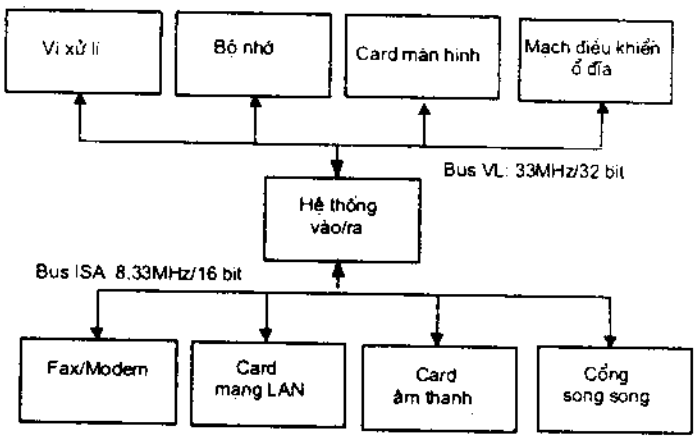
Khe cắm ISA



Hình 5.2 Bản mạch và một mẩu khe cắm EISA gồm 2 nấc.

5.1.5. Bus cục bộ VL (VESA Local)

Nhược điểm chính của các bus PC, ISA và EISA là tốc độ truyền tải tối đa bị hạn chế bởi tốc độ xung nhịp đã được chuẩn hoá trên bus. Đó là hiện tượng "nghẹt cổ chai" (bottleneck) ở tần số 8,33MHz làm lãng phí thời gian xử lí của các vi xử lí có tốc độ cao hơn và nói chung làm giảm hiệu suất của toàn hệ thống. Do vậy hiệp hội VESA (Video Electronics Standards Association) đã phát triển chuẩn bus cục bộ này, đầu tiên cho phép các card ghép nối từ vi xử lí tới bản mạch video tốc độ nhanh có thể hoạt động được. Nó sử dụng một khe cắm chuẩn ISA cùng với một khe cắm mở rộng nối vào bus cục bộ. Cấu trúc của bus được trình bày trên hình 5.3, các chân tín hiệu được cho trong phần phụ lục ở cuối sách.

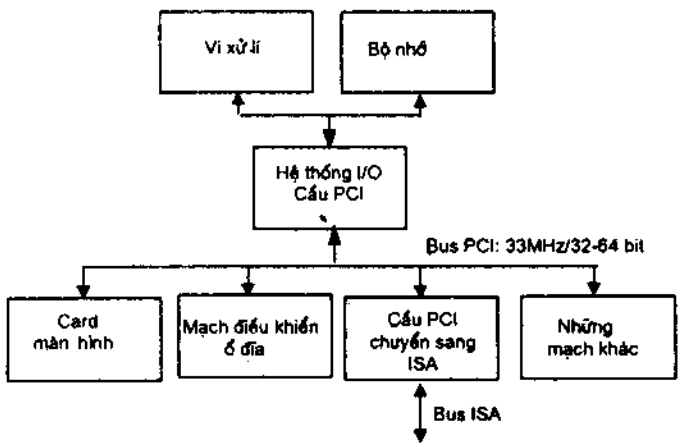


Hình 5.3 Cấu trúc bus VL.

Các mạch ghép nối cho các ứng dụng đòi hỏi tốc độ chuyển tải nhanh như bộ vi xử lí, bộ nhớ, bản mạch ghép nối màn hình, bộ điều khiển ổ đĩa v.v... được nối với bus cục bộ có tần số hoạt động phụ thuộc vào đồng hồ hệ thống và bề rộng các bus dữ liệu và địa chỉ là 32 bit. Như vậy tốc độ truyền dữ liệu trên bus này phụ thuộc vào tần số đồng hồ hệ thống nhưng chuẩn VL quy định tốc độ này không quá 33MHz. Thời gian gần đây ra đời các bản mạch chính đã chạy với tốc độ 66MHz hoặc tới 100MHz thì quy định này cũng trở thành một hạn chế của chuẩn bus VL. Còn các bản mạch cho các ứng dụng thông thường như modem, card mạng, card âm thanh v.v...vẫn sử dụng bus ISA như bình thường.

5.1.6. Bus PCI (Peripheral Component Interconnection).

Song song với hiệp hội VESA, hãng Intel cũng đã phát triển chuẩn ghép nối bus cục bộ riêng biệt này cho các vi xử lí Pentium của mình và hiện nay đã trở nên rất phổ biến gọi là chuẩn liên kết các thành phần ngoại vi PCI. Bus này có thể kết nối với bus bộ xử lí thông qua một chip cầu nối đặc biệt. Vì vậy chỉ cần thay đổi chip cầu nối này khi thay đổi vi xử lí còn các thiết bị ngoại vi vẫn làm việc được với tất cả các loại vi xử lí. Công nghệ này cho phép truy cập tốt các bộ nhớ nhanh, các ổ đĩa và card màn hình tốc độ cao. Một tập các vi mạch ghép nối như 82430PCI được sản xuất sẵn cho mục đích đó. Giống như bus VL, bus PCI cũng sử dụng đồng hồ hệ thống cho việc truyền dữ liệu nhưng có ưu điểm hơn là nó có thể hoạt động trên các đường dẫn quá 32 hoặc 64 bit. Do tốc độ truyền tải dữ liệu cao nên số ghép nối bus được hạn chế tới 2 hoặc 3 đơn vị (thường là chỉ gồm card màn hình và bộ điều khiển ổ đĩa cứng). Trong trường hợp tốc độ xung nhịp là 33MHz với độ rộng bus dữ liệu là 64 bit (8 byte), bus này cho một tốc độ truyền tải dữ liệu lên tới 528 Mbyte/giây. Hình 5.4 chỉ ra cấu trúc của một bus PCI, trong đó một *cầu I/O PCI* cho phép xâm nhập tới các bản mạch ISA, EISA hoặc MCA. Do phải đáp ứng với yêu cầu truyền tốc độ cao và giảm kích thước các bản mạch ghép nối PCI nên các khe cắm PCI không thể tương thích với các khe cắm ISA hoặc EISA.



Hình 5.4 Cấu trúc bus PCI.

5.1.7. Bus nối tiếp chung USB (univeral serial bus).

Bus USB là một công nghệ mới và sẽ được phát triển nhanh chóng trong những năm tới. Cáp USB cho phép nối móc xích tới 127 chân. Mặc dù tốc độ truyền dữ liệu chỉ cỡ 12Mb/giây bus này cũng đáp ứng thoả đáng cho hầu hết các thiết bị ngoại vi. Một lợi thế của chuẩn này là các thiết bị ngoại vi tự nhận dạng tương thích với hệ thống "cắm và chạy ngay" PnP (plug and play) và cung cấp tiêu chuẩn công nghệ cho các kết nối tương lai. Hơn nữa những thiết bị USB có thể cắm hoặc thôi cắm "nóng", nghĩa là không phải tắt máy vi tính khi muốn cắm vào hoặc rút ra một thiết bị USB vào hệ máy tính. Thời gian gần đây phần lớn các bản mạch chính đã được thiết kế hỗ trợ chuẩn bus này.

5.2. NĂNG SUẤT TRUYỀN DỮ LIỆU

So sánh giữa các cấu trúc bus ta thấy *năng suất truyền dữ liệu* (throughput) phụ thuộc vào số các byte đang được thông tin cho mỗi quá trình truyền dữ liệu và tốc độ truyền. Với các bus PC, ISA và EISA tốc độ này bị hạn chế ở 8,33MHz trong khi các bus VL và PCI vượt qua được giới hạn này và phụ thuộc vào tốc độ của vi xử lí. Trong nhiều ứng dụng, bus ISA đảm bảo

một công nghệ tốt vì nó đã được phát triển trong một thời gian dài hơn, có năng suất truyền khá tốt và cái chính là rẻ và có độ tin cậy cao. Loại cấu trúc bus hay công nghệ ghép nối được sử dụng phụ thuộc vào yêu cầu năng suất truyền dữ liệu. Bảng 5.1 chỉ ra một vài tốc độ truyền tải dữ liệu điển hình trong đó cao nhất là việc sử dụng thông tin giữa vi xử lí và bộ nhớ cũng như card ghép nối màn hình. Tốc độ này tuỳ thuộc vào các chương trình ứng dụng và hệ điều hành, thí dụ các giao diện *Người dùng - Đồ hoạ GUI* (Graphics User Interface) đòi hỏi một năng suất truyền cao hơn nhiều các chương trình chạy trong chế độ văn bản trong khi các bản mạch âm thanh chất lượng cao (16 bit, tốc độ lấy mẫu 44,1 kHz) cũng chỉ đòi hỏi một tốc độ truyền thấp cỡ 88 kbyte/giây.

Bảng 5.1 **Một vài tốc độ truyền tải điển hình**

Thiết bị	Tốc độ truyền tải	Ứng dụng
Đĩa cứng	5 Mbyte / giây	Truyền dữ liệu điển hình
Card âm thanh	88 kbyte / giây	16 bit, tốc độ lấy mẫu 44,1 kHz
Card mạng LAN	10 Mbit / giây	Mạng Ethernet 10Mbit/giây
RAM	66 Mbyte / giây	Truyền từ vi xử lí tới RAM
Thông tin nối tiếp	1kbyte / giây	9600 bit / giây
Card SVGA	15 Mbyte / giây	Độ phân giải 1024x768 với 256 màu

CÂU HỎI, BÀI TẬP

Trả lời các câu hỏi trắc nghiệm sau, chọn lựa 1 trong 4 đáp án từ a đến d:

- Các bản mạch ghép nối vào/ra sử dụng trong PC là:
  PC, ISA, EISA, MCA, VL, PCI
  PC, ISA, ANSI, MCA, VL, PCI
  PC, ESA, EISA, MCA, VL, PCI
  PC, ISA, EISA, MCA, VL, PCM
- Có bao nhiêu byte có thể thông tin với PC trong một lần:
  a) 1 b) 2 c) 4 d) 8
- Tốc độ đồng hồ của card ISA là:
  a) 4,77MHz b) 8,33MHz
  c) 16MHz d) bằng tốc độ đồng hồ hệ thống
- Tốc độ truyền dữ liệu cực đại của bus ISA là:
  a) 8,33Mbyte/sec b) 16,66Mbyte/sec
  c) 32Mbyte/sec d) 64Mbyte/sec
- Kĩ thuật MCA khác so với ISA/EISA là:
  a) MCA dùng hệ thống cache
  b) MCA dùng một địa chỉ và bus dữ liệu chung
  c) MCA dùng ghép nối nối tiếp
  d) Kĩ thuật MCA sử dụng bus không đồng bộ
- Quá trình xử lí sau đây có tốc độ truyền dữ liệu cao nhất:
  a) xử lí truyền dữ liệu tới bộ nhớ b) card âm thanh
  c) card mạng cục bộ LAN d) truyền dữ liệu trong MODEM
- Ngoại vi nào nên nối với bus cục bộ:
  a, card âm thanh b, card video c, card điều khiển ổ đĩa cứng
  d, card Fax e, card mạng LAN f, card Modem

CHƯƠNG 6. CÁC PHƯƠNG PHÁP THÔNG TIN CƠ BẢN GIỮA CPU VÀ THIẾT BỊ NGOẠI VI

6.1. ĐỊNH ĐỊA CHỈ THIẾT BỊ NGOẠI VI

Trong hệ thống ghép nối máy tính, có thể định địa chỉ cho các thiết bị ngoại vi theo một trong hai cách sau:

1. Ngoại vi được định địa chỉ trong cùng một không gian nhớ với bộ nhớ chính. Lúc này các thiết bị ngoại vi sẽ chiếm một vùng nào đó trong không gian bộ nhớ, phần còn lại là dành cho các vi mạch nhớ. Trong trường hợp này quá trình trao đổi thông tin giữa CPU và ngoại vi gọi là quá trình ***vào/ra ánh xạ bộ nhớ*** (memory-mapped I/O). Tín hiệu điều khiển $IO/\overline{M}$ luôn ở mức lôgic 0 và lệnh trao đổi thông tin dữ liệu cho cả bộ nhớ và thiết bị ngoại vi là lệnh MOV. Thí dụ, các bản mạch ghép nối màn hình thường được định địa chỉ trong không gian bộ nhớ như các card EGA, VGA có địa chỉ trong dải từ A0000h đến C7FFFh ...

2. Ngoại vi được định địa chỉ trong không gian 64kbyte cho các cổng vào/ra. Bộ nhớ chính được hoàn toàn giữ nguyên (thí dụ, 1Mbyte trong 8086) với các lệnh trao đổi thông tin MOV; còn các lệnh trao đổi thông tin với các cổng vào/ra lúc này là các lệnh IN/OUT (trong asembly), PORT [địa chỉ cổng] (trong Pascal) v.v... Ta gọi đây là quá trình ***vào/ra tách biệt*** (isolated I/O). Đây là phương pháp phổ biến cho hầu hết các ngoại vi, trong hệ điều hành WINDOWS 95 ta có thể nhận được các thông tin khá tỉ mỉ về các ngoại vi nối với PC và địa chỉ của chúng theo cách truy nhập sau:

Control Panel → System → Device Manager → Properties → Input/Output

6.2. QUÁ TRÌNH MÓC NỐI THÔNG TIN

Dữ liệu được máy tính nhập vào hoặc xuất ra theo chương trình. Khi máy tính muốn gửi dữ liệu ra ngoại vi, một lệnh trong chương trình sẽ viết dữ liệu tới *cổng ra* (out port) nhằm thông tin với thiết bị ngoại vi. Khi máy tính muốn nhận dữ liệu từ ngoại vi, một lệnh đọc dữ liệu từ *cổng vào* (in port), thí dụ cổng này được nối với bàn phím, sẽ được thực hiện. Thuật ngữ cổng hoặc cảng (port) biểu thị một giao diện (interface) giữa máy tính và thiết bị ngoại vi. Các cổng vào/ra thường là các mạch điện phức tạp. Các cổng này thường có *các thanh ghi* dùng để chứa các số liệu (các thanh ghi đệm số liệu), các thanh ghi chứa các từ dữ liệu ra điều khiển và các thanh ghi lưu trữ từ chỉ thị trạng thái hiện tại của cổng.

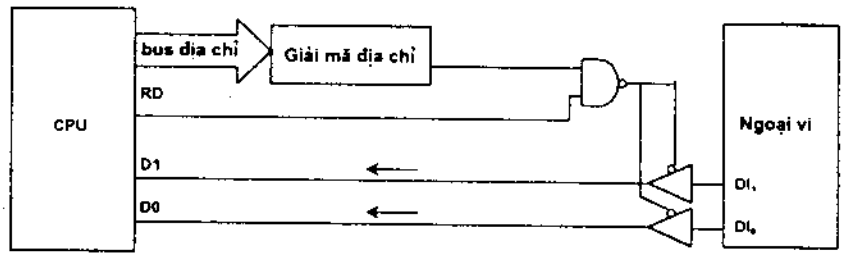
Quá trình phát và thu các *tín hiệu điều khiển* để khởi phát việc trao đổi dữ liệu được gọi là quá trình *thông tin có móc nối* (handshake). Khi xảy ra quá trình này, mạch ghép nối sẽ phát hoặc thu các tín hiệu điều khiển đó - gọi là các *tín hiệu móc nối* - với vi xử lí. Do vậy, việc trao đổi thông tin giữa các đối tác thông tin (CPU và các thiết bị ngoại vi) được thực hiện theo một trong hai cách dưới đây.

6.2.1. Thông tin không có móc nối (no-handshaking)

Các bên thông tin với nhau đọc vào hoặc viết ra dữ liệu một cách trực tiếp mà không cần biết đối tác kia có sẵn sàng hoặc chấp nhận hay không. Thí dụ, trong hệ đo nhiệt độ ghép nối máy vi tính: điện áp ra trên cặp nhiệt điện được khuếch đại lên, tiếp đó được biến đổi thành tín hiệu số rồi được đọc vào máy vi tính. Rõ ràng tín hiệu này lúc nào cũng sẵn sàng cho CPU đọc

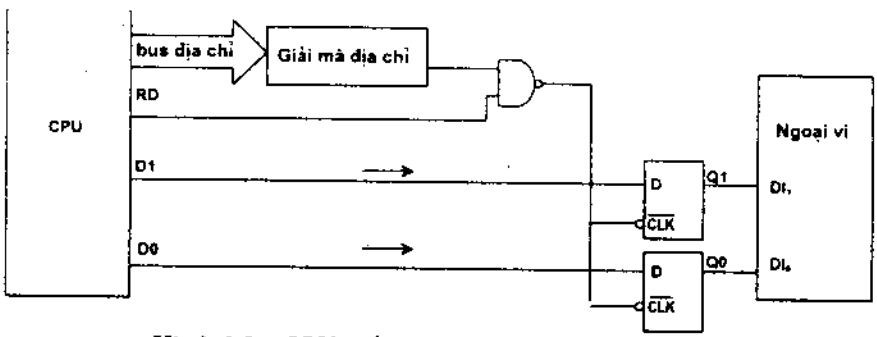
vì nhiệt độ là một đại lượng biến đổi rất chậm so với tốc độ đọc dữ liệu của máy vi tính. Cũng vậy, nếu dùng máy vi tính điều khiển bộ hiển thị số của các giá trị nhiệt độ kể trên thì bất cứ lúc nào CPU cũng có thể đưa tín hiệu hiển thị ra được. Đây gọi là các quá trình vào/ra không có móc nối.

Hình 6.1 là thí dụ về một sơ đồ mạch điện cho phép CPU đọc dữ liệu không có móc nối từ ngoại vi.



Hình 6.1 CPU đọc dữ liệu không có móc nối.

Mạch điện rất đơn giản, chỉ gồm có bộ giải mã địa chỉ và các cổng ra 3 trạng thái. Tín hiệu từ các ngoại vi (thí dụ các mức thế lôgic từ các cảm biến được đưa ra 2 đầu DI_0 và DI_1) được đưa qua 2 cổng 3 trạng thái vào bus số liệu của CPU. Khi đọc vào, CPU không cần biết đến trạng thái của thiết bị ngoại vi mà đưa ngay tín hiệu địa chỉ đến bộ giải mã cùng tín hiệu RD để mở các cổng 3 trạng thái cho phép đọc từ số liệu trên DI_0 và DI_1. Hình 6.2 là sơ đồ cho phép CPU viết dữ liệu không có móc nối ra ngoại vi, ở đây các trigơ D được dùng làm thanh ghi chốt dữ liệu từ D0-D1 vào DI_0 và DI_1.



Hình 6.2 CPU viết dữ liệu không có móc nối.

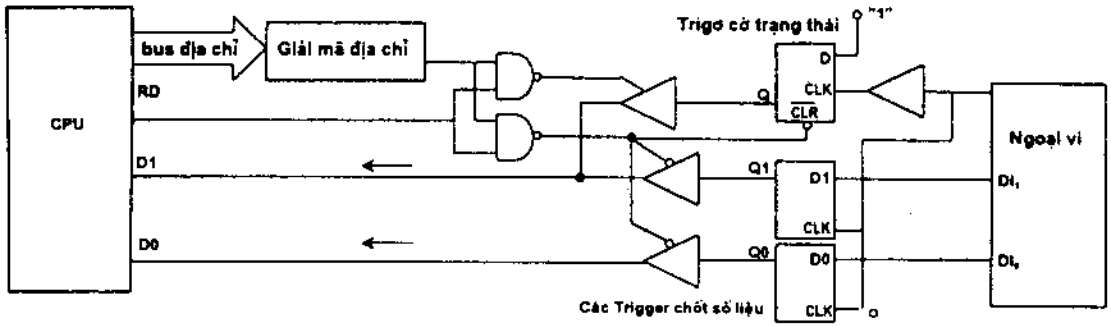
6.2.2. Thông tin có móc nối (handshaking)

Các bên thông tin chỉ trao đổi dữ liệu khi có tín hiệu móc nối *sẵn sàng* READY hoặc tín hiệu *ghi nhận* ACK từ bên kia. Phương thức này dùng khi tốc độ xử lí dữ liệu của các đối tác thông tin không bằng nhau (thí dụ, bên phát dữ liệu có tốc độ phát nhanh hơn tốc độ xử lí thông tin bên thu). Hình 6.3 là một thí dụ sơ đồ cho phép CPU đọc có móc nối dữ liệu từ ngoại vi khi có yêu cầu bằng tín hiệu READY.

Dây dữ liệu D1 cũng đồng thời được dùng để đọc trạng thái ngoại vi, nếu mức lôgic trên dây này bằng 0 là ngoại vi chưa sẵn sàng, bằng 1 là đã sẵn sàng.

Ngoại vi phát tín hiệu STB (strobe) có 2 tác dụng: một là tạo xung nhịp CLK làm lối ra của trigơ cờ trạng thái bằng 1 (chốt số liệu "1" từ D sang Q và cho ra cổng 3 trạng thái); hai là tạo

xung nhịp CLK cho phép chốt các số liệu từ DI_0 và DI_1 tới các lối ra Q0 và Q1 trên các trigơ chốt số liệu.

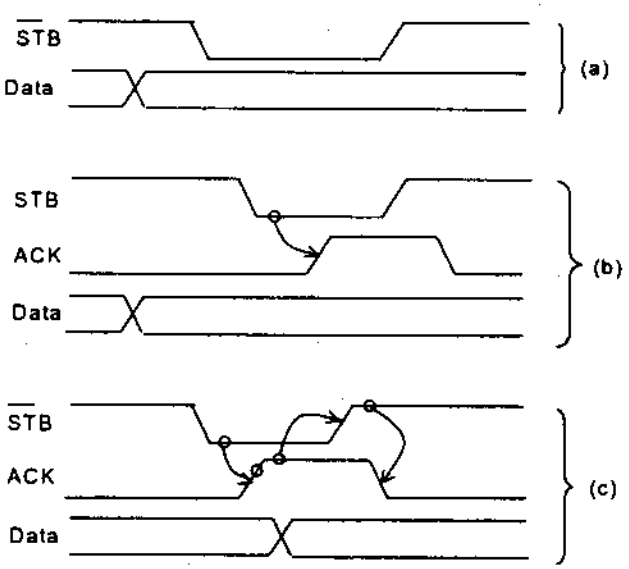


Hình 6.3 CPU đọc tín hiệu có móc nối từ ngoại vi.

CPU sẽ liên tục đọc cờ trạng thái qua dây D1, nếu cờ được xác lập CPU sẽ xuất tín hiệu địa chỉ tới bộ giải mã địa chỉ và làm tích cực dây RD. Kết quả là làm mở các cổng 3 trạng thái cho phép đọc số liệu vào từ các dây dữ liệu D0 và D1. Thao tác này cũng đồng thời tạo xung xoá $\overline{CLR}$ đưa lối ra Q tức là cờ trạng thái về "0" kết thúc một chu trình đọc từ dữ liệu có móc nối.

Có thể phân biệt 3 trường hợp truyền thông tin móc nối tuỳ thuộc vào yêu cầu cụ thể, dạng các xung móc nối được cho như hình 6.4 sau:

1. *Vào/ra với xung Strobe STB đơn giản*, trong nhiều ứng dụng, số liệu chỉ hợp lệ (ổn định) trên bus trong một thời gian nhất định và do vậy bên thu chỉ có thể đọc nó khi bên phát gửi ra xung $\overline{STB}$. Một thí dụ là trường hợp đọc mã ASCII từ bàn phím vào CPU. Đường $\overline{STB}$ được nối tới một lối vào của cổng vào và CPU sẽ thăm dò nó để xác định khi nào có thể đọc dữ liệu từ bàn phím. Hình 6.4a cho thấy sau khi chip điều khiển bàn phím xuất ra dữ liệu song song trên bus, nó sẽ hạ dây móc nối $\overline{STB}$ xuống thấp để báo số liệu đã sẵn sàng.
2. *Vào/ra móc nối đơn*, hình 6.4b chỉ ra dạng sóng móc nối trong trường hợp này. Mạch điện lúc này gồm có dây tín hiệu $\overline{STB}$ từ ngoại vi tới máy tính và dây ACK từ máy tính đến ngoại vi. Khi ngoại vi muốn xuất ra dữ liệu, nó gửi tín hiệu móc nối $\overline{STB}$ tới CPU. Vi xử lí phát hiện tín hiệu này nhờ thăm dò hoặc ngắt và đọc byte số liệu vào máy tính. Sau đó nó gửi ra tín hiệu ghi nhận *ACK* để báo cho ngoại vi biết rằng số liệu đã được đọc và cho phép ngoại vi gửi byte số liệu kế tiếp.
3. *Vào/ra móc nối kép*, phương



Hình 6.4 Ba trường hợp sử dụng tín hiệu STB và ACK.

pháp này thậm chí đòi hỏi có sự hợp tác chặt chẽ hơn nữa giữa các bên thông tin tuy mạch điện vẫn như phương pháp trên. Thiết bị phát trong trường hợp này là ngoại vi hạ đường $\overline{STB}$ xuống mức thấp để hỏi bên thu là CPU đã sẵn sàng chưa ? Bên thu nâng dây ACK lên mức cao để báo đã sẵn sàng. Bên phát đưa byte số liệu lên bus và nâng lại dây $\overline{STB}$ lên mức cao để báo rằng đã có số liệu ổn định trên bus. Sau khi đọc xong số liệu, bên thu là CPU sẽ hạ dây ACK về mức thấp để chỉ thị rằng đã đọc xong số liệu và báo sẵn sàng nhận byte số liệu tiếp.

6.3. CÁC QUÁ TRÌNH TRAO ĐỔI DỮ LIỆU - QUÁ TRÌNH VÀO/RA

Quá trình trao đổi dữ liệu giữa CPU và các ngoại vi gọi tắt là *quá trình vào/ra*. Quá trình này bao gồm các bước sau:

1. Xác định xem liệu có một thiết bị nào đó sẵn sàng trao đổi thông tin với CPU hay không.
2. Nếu có thì xác định xem là thiết bị nào.
3. Giải quyết tranh chấp nếu có nhiều thiết bị đều cùng muốn trao đổi thông tin với CPU.
4. Phát ra các tín hiệu điều khiển và đồng bộ để trao đổi dữ liệu.

Chương trình thực hiện việc trao đổi dữ liệu gọi là chương trình phục vụ (service routine). Có thể thực hiện tất cả các bước trên bằng điều khiển mềm (thí dụ như vào/ra theo chương trình) hoặc dùng phần cứng cho bước đầu tiên và phần mềm cho các bước còn lại (như vào/ra bằng ngắt) v.v... Xét cụ thể các phương pháp này như dưới đây.

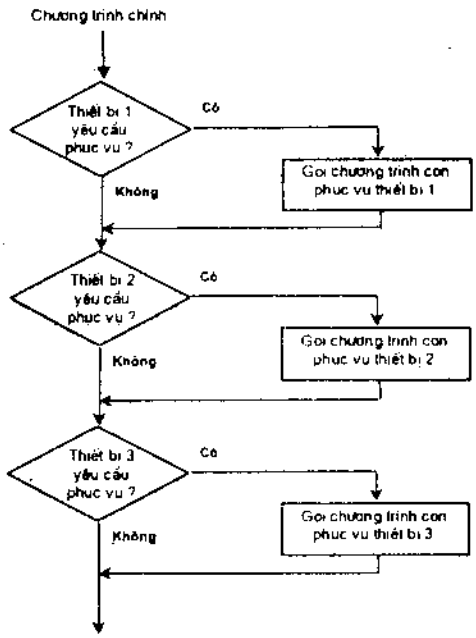
6.3.1. Vào/ra điều khiển bằng chương trình

Đây là phương pháp thông tin được khởi phát từ phía CPU. Dạng đơn giản nhất của việc thực hiện phương pháp này là một thiết bị ngoại vi nào đó sẵn sàng trao đổi dữ liệu với CPU và một lệnh bên trong chương trình sẽ khởi phát quá trình trao đổi. Nếu như chương trình được viết sao cho đảm bảo thời gian để thiết bị hồi phục trạng thái sẵn sàng giữa mỗi lệnh trao đổi dữ liệu thì không cần có tín hiệu móc nối nữa. Thí dụ như trường hợp dùng bộ vi xử lí để kiểm tra một vài đại lượng biến đổi chậm như nhiệt độ phòng chẳng hạn. Tóm lại, trong trường hợp này giữa CPU, bộ nhớ và các ngoại vi trao đổi thông tin với nhau bằng các chương trình qua các thanh ghi và bộ nhớ từng lệnh (byte, từ) một. Mạch ghép nối có nhiệm vụ làm tương thích về lôgic và công nghệ giữa CPU và ngoại vi.

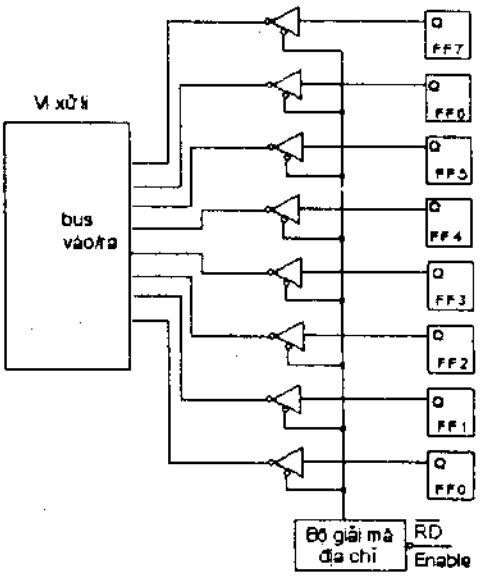
Có hai phương pháp điều khiển quá trình thông tin loại này:

a, Phương pháp hỏi vòng (polling): Trong phương pháp này, chương trình liên tục kiểm tra trạng thái của các ngoại vi để xem có yêu cầu trao đổi dữ liệu (yêu cầu phục vụ) hay không. Đến khi một ngoại vi nào đó có yêu cầu phục vụ (tức là có tín hiệu READY) thì chương trình sẽ nhảy vào một chương trình con phục vụ rồi lại tiếp tục kiểm tra ngoại vi kế tiếp. Quá trình này cứ được thực hiện cho tới ngoại vi cuối rồi được lặp lại từ đầu. Phương pháp này tỏ ra đơn giản nhưng gặp phải một nhược điểm là khi số lượng ngoại vi tăng lên sẽ làm lãng phí thời gian và khả năng đáp ứng tức thời với yêu cầu phục vụ là rất thấp vì CPU phải kiểm tra tất cả các trạng thái của các ngoại vi trước khi quay về ngoại vi có yêu cầu phục vụ. Hình 6.5 là lưu đồ của chương trình vào/ra theo chương trình sử dụng hỏi vòng.

Nếu thông tin được phát hoặc nhận từ một thiết bị ngoại vi thì thiết bị đó sẽ thiết lập một cờ trạng thái thích hợp bằng một mạch lật. Bộ vi xử lí sẽ kiểm tra cờ và nhảy vào chương trình phụ phục vụ.

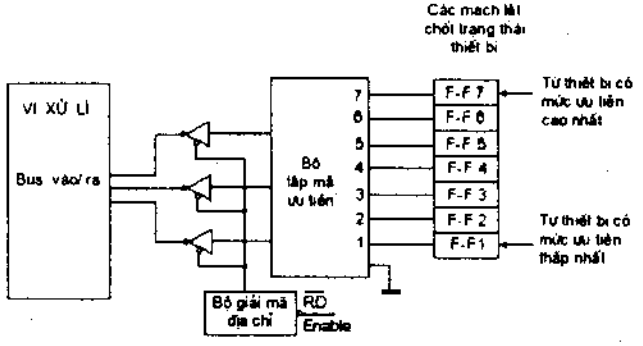


Hình 6.5 Lưu đồ hỏi vòng.



Hình 6.6 Đọc các cờ trạng thái trên 8 thiết bị.

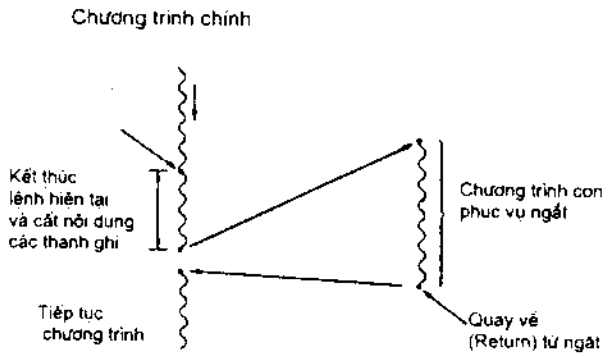
Hình 6.6 là một sơ đồ cho phép đọc các cờ trạng thái của 8 thiết bị trên bus vào/ra. Thông tin này có thể được đọc vào thanh ghi tích luỹ và được kiểm tra mỗi bit để xác định thiết bị nào yêu cầu phục vụ. Trong phương pháp này, mức độ ưu tiên của các thiết bị ngoại vi được thiết lập bằng việc kiểm tra thông tin trạng thái của các thiết bị có mức ưu tiên cao hơn trước. Bit 7 của bus vào/ra sẽ tương ứng với thiết bị có mức ưu tiên cao nhất và bit 6 sẽ được nối với mạch lật của thiết bị có mức ưu tiên cao tiếp theo. Số các đường vào/ra có thể được giảm xuống bằng việc xen vào một bộ lập mã ưu tiên giữa các cờ trạng thái của thiết bị và bus vi xử lí như hình 6.7. Một bộ lập mã ưu tiên có thể có một vài lối vào ở mức tích cực trong cùng một thời gian nhưng nó sẽ phát ra một mã nhị phân chỉ tương ứng với đường vào được đánh số cao nhất trong số các đường tích cực. Như vậy, tất cả các chốt trạng thái thiết bị hoặc các chốt ngắt có thể được gán cho một mức ưu tiên bằng phần cứng. Thiết bị có mức ưu tiên cao nhất được nối tới đường lối vào được đánh số cao nhất. Một bộ lập mã ưu tiên 8 đường chấp nhận các lối vào từ cờ trạng thái 1 tới 7. Lối ra của bộ lập mã là một mã nhị phân 3 bit đại diện cho đường lối vào tích cực có mức ưu tiên cao nhất trong các đường tích cực. Thông tin này được đưa vào CPU cho phép nó xác định chương trình phụ tương ứng dựa trên mã 3 bit này. Nếu không có thiết bị nào muốn được phục vụ CPU sẽ đọc được mã 000 từ bộ lập mã ưu tiên. Sơ đồ này cho phép chạy nhanh và hiệu quả hơn sơ đồ dùng phần mềm như ở trên.



Hình 6.7 Đọc cờ trạng thái bằng lập mã ưu tiên.

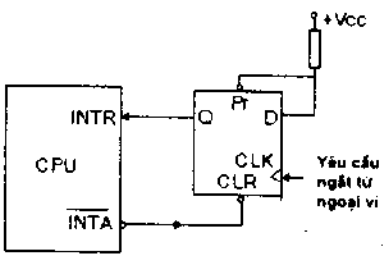
b, Phương pháp ngắt

Phương pháp này như sau, trong khi CPU đang thực hiện chuỗi lệnh của chương trình chính, nếu một ngoại vi nào đó cần trao đổi thông tin với CPU, nó sẽ gửi một tín hiệu yêu cầu trao đổi thông tin gọi là *yêu cầu ngắt* (thí dụ $\overline{INTR}$) tới CPU. CPU sẽ thực hiện nốt lệnh hiện tại và trả lại bằng tín hiệu *chấp nhận ngắt* (thí dụ, INTA). Chương trình chính lúc này bị ngắt và CPU chuyển sang *chương trình con phục vụ ngắt* tức là chương trình con trao đổi thông tin với ngoại vi yêu cầu. Sau khi xong việc, nhờ lệnh RET và các lệnh cất, hồi phục ngăn xếp (PUSH, POP) CPU sẽ quay về đúng chỗ bị ngắt và tiếp tục thực hiện chương trình chính như hình 6.8 chỉ ra.

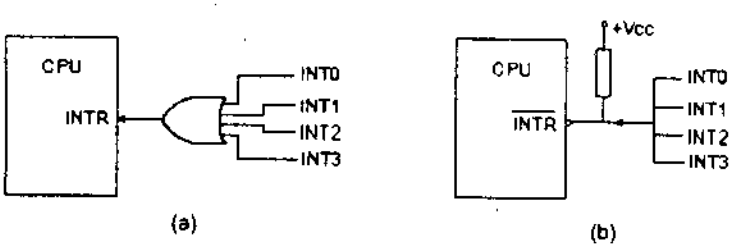


Hình 6.8 Chuỗi các sự kiện xảy ra trong quá trình ngắt.

Các tín hiệu yêu cầu ngắt từ các cổng vào/ra của thiết bị ngoại vi bất kì đều được gửi tới chân INTR của CPU nhờ các mạch lôgic điều khiển ngắt. Tín hiệu INTR ở chân lối vào yêu cầu ngắt này được kiểm tra ở cuối mỗi chu kì lệnh. Tức là khi mỗi lệnh được thực hiện xong, mức điện thế ở đây đều được kiểm tra. Thí dụ, nếu xuất hiện mức cao có nghĩa là một yêu cầu ngắt đã xảy ra; một lệnh rẽ nhánh tới một chương trình phụ sẽ được nạp vào thanh ghi lệnh. Chương trình này sẽ tiến hành *hỏi vòng ngắt* (interrupt polling) các thiết bị (bắt đầu từ thiết bị có mức ưu tiên cao nhất) như cách thức của phương pháp vào/ra bằng hỏi vòng để xác định *nguồn gây ngắt* và phục vụ ngắt. Khi phục vụ ngắt xong, cờ trạng thái lối vào của nó được xoá và đầu ngắt của CPU lại được kiểm tra như trên. Một chu kì lệnh có thể khá dài nên tín hiệu yêu cầu ngắt thường được chốt lại bằng mạch lật cho đến khi bộ xử lí trả về tín hiệu ghi nhận $\overline{INTA}$ để xoá như sơ đồ hình 6.9. Nhiều nguồn ngắt có thể đưa tín hiệu yêu cầu ngắt của chúng tới một đầu vào yêu cầu ngắt INTR của bộ vi xử lí bằng một cổng OR nhiều lối vào, nếu mức tích cực của đầu vào yêu cầu ngắt của CPU là mức 1 (hình 6.10a) hoặc dùng các đầu ra yêu cầu ngắt của các thiết bị ngoại vi là các cổng collector hở với 1 điện trở gánh chung, nếu mức tích cực của đầu vào yêu cầu ngắt $\overline{INTR}$ của CPU là mức 0 (hình 6.10b).



Hình 6.9 Dùng mạch lật để chốt tín hiệu yêu cầu ngắt INTR đến CPU.



Hình 6.10 Nhiều nguồn ngắt cùng đưa đến một đầu vào yêu cầu ngắt của CPU.

Phương pháp hiệu quả nhất là dùng các *sơ đồ ngắt vectơ*, trong đó nguồn ngắt phát ra tín hiệu yêu cầu ngắt cùng với số nhận dạng thiết bị và chương trình sẽ rẽ nhánh tới chương trình con ngắt tương ứng nhờ vào số nhận dạng này. Khi cần tốc độ nhanh hơn, thay vì cho số nhận dạng là địa chỉ của chương trình con phục vụ ngắt ứng với thiết bị ngắt đó được nạp sẵn trong một chip ROM sẽ được đưa vào bus dữ liệu ngay cho CPU.

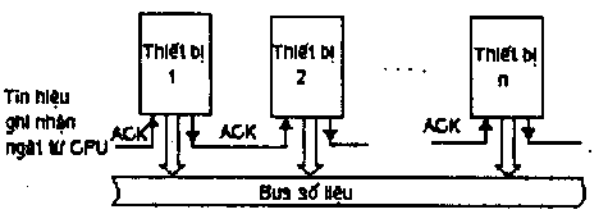
Hầu hết các hệ thống ngắt vectơ đều cho phép ngắt và cấm một ngắt bất kì bằng phần mềm do người lập trình hoặc được thực hiện tự động bởi bộ vi xử lí. Thí dụ, trong một số khoảng thời gian khi chương trình chính chạy, có thể có một số thiết bị ngoại vi đặc biệt không được phép gây ngắt tới CPU. CPU thường tự động cấm ngắt trong những trường hợp như: khi khởi động hệ thống vì lúc này hệ cần nạp các thông số cần thiết trước khi ghi nhận bất cứ yêu cầu ngắt nào từ ngoại vi; hoặc trong trường hợp sau khi ngắt được ghi nhận vì cấm ngắt sẽ cho phép xác định nguồn ngắt. Ngắt bị cấm bằng lệnh xoá cờ ngắt CLI và được phép bằng lệnh lập cờ ngắt STI. Trước khi chạy một chuỗi lệnh có các ngắt bị cấm, chương trình sẽ truyền một từ (word) tới *thanh ghi mặt nạ ngắt* (mask register) để ngăn sự kích hoạt các nguồn gây ngắt đó. Các vi xử lí thường có đầu vào ngắt không che (cấm) được NMI (non-maskable). Lệnh cấm ngắt không có tác dụng với đầu ngắt này và nó được dành để phục vụ các yêu cầu ưu tiên tuyệt đối như ngắt do lỗi nguồn nuôi v.v...

Có hai phương pháp để xác định nguồn ngắt có *mức ưu tiên* cao nhất:

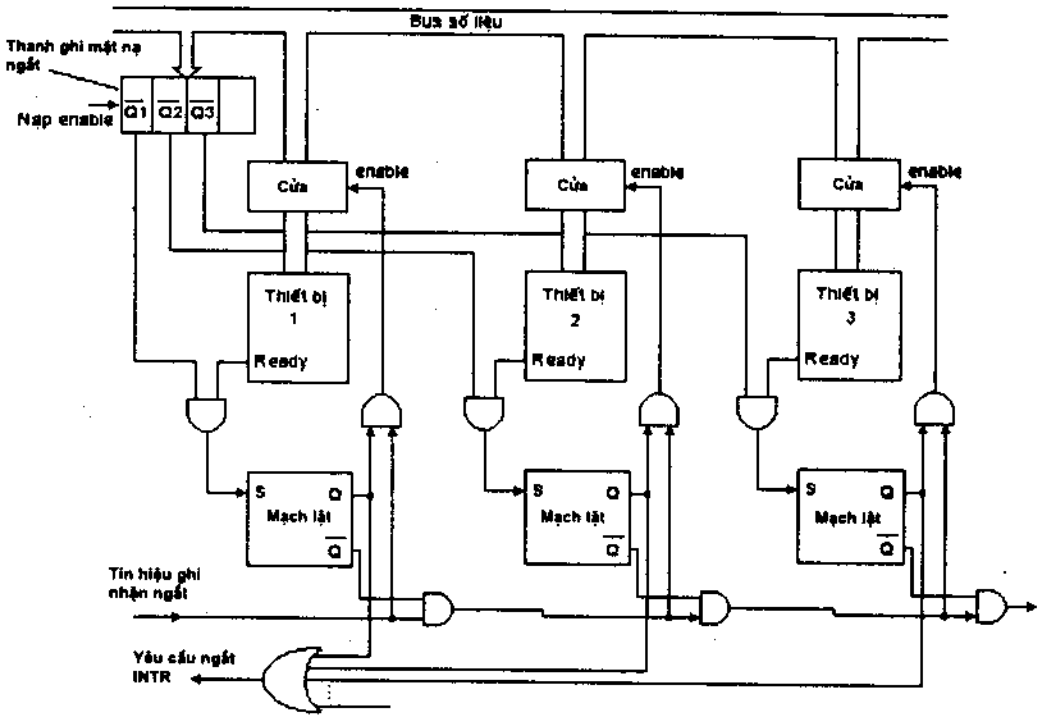
- *Dùng sơ đồ có cấu hình móc xích* (daisy chain):

Phương pháp này cho phép giảm thời gian tìm nguồn ngắt có mức ưu tiên cao nhất bằng cách thay thế chương trình hỏi vòng bằng phần cứng như sơ đồ khối hình 6.11. Khi phát hiện trong chuỗi các dây *ghi nhận ngắt* của các thiết bị tín hiệu yêu cầu ngắt INTR, vi xử lí sẽ truyền *tín hiệu ghi nhận* ACK tới thiết bị ngoại vi 1 được nối theo kiểu móc xích. Nếu là nguồn gây ngắt, thiết bị 1 sẽ chặn tín hiệu ACK tại đó và đưa vectơ ngắt của nó vào bus số liệu cho bộ xử lí đọc. Nếu không phải, thiết bị 1 sẽ truyền tín hiệu ACK sang cho thiết bị 2. Thiết bị 2 sẽ thực hiện thủ tục giống như vậy và cứ tiếp tục cho đến thiết bị ở cuối chuỗi mắt xích. Một thí dụ về sơ đồ nguyên lí của phương pháp này được cho trong hình 6.12.

Bằng thanh ghi mặt nạ ngắt, có thể che tín hiệu sẵn sàng (ready) của một thiết bị ngoại vi bằng cổng AND tương ứng. Vi xử lí truyền một từ che ngắt vào thanh ghi mặt nạ. Mỗi thiết bị cần che tương ứng với một bit trong thanh ghi. Lối ra của các bit bị che này cấm mạch lật (Q = 0) và do đó thiết bị không thể gửi tín hiệu yêu cầu ngắt tới vi xử lí. Khi một thiết bị không bị che và có yêu cầu ngắt (Ready), mạch lật tương ứng được đặt (Q = 1), dẫn tới đường ngắt được kích hoạt đồng thời lối ra tương ứng $\overline{Q} = 0$ sẽ đóng cổng AND không cho tín hiệu ghi nhận ACK đi tiếp sang mắt xích phía sau nữa. Khi tín hiệu truyền tới thiết bị có mạch lật được đặt, một tập các cửa được cho phép mở (enable) để truyền một số nhận dạng các thiết bị hoặc địa chỉ chương trình phục vụ từ thiết bị vào bus số liệu. Tín hiệu ghi nhận ngắt không lan truyền qua các thiết bị này cho tới khi vi xử lí phục vụ xong thiết bị và reset lại các mạch lật. Lúc này, tín hiệu ghi nhận ngắt ACK có thể tiếp tục lan truyền tới thiết bị kế tiếp có phát ra tín hiệu yêu cầu ngắt.Vi xử lí sẽ lặp lại cho tới khi đường ngắt bị xoá.

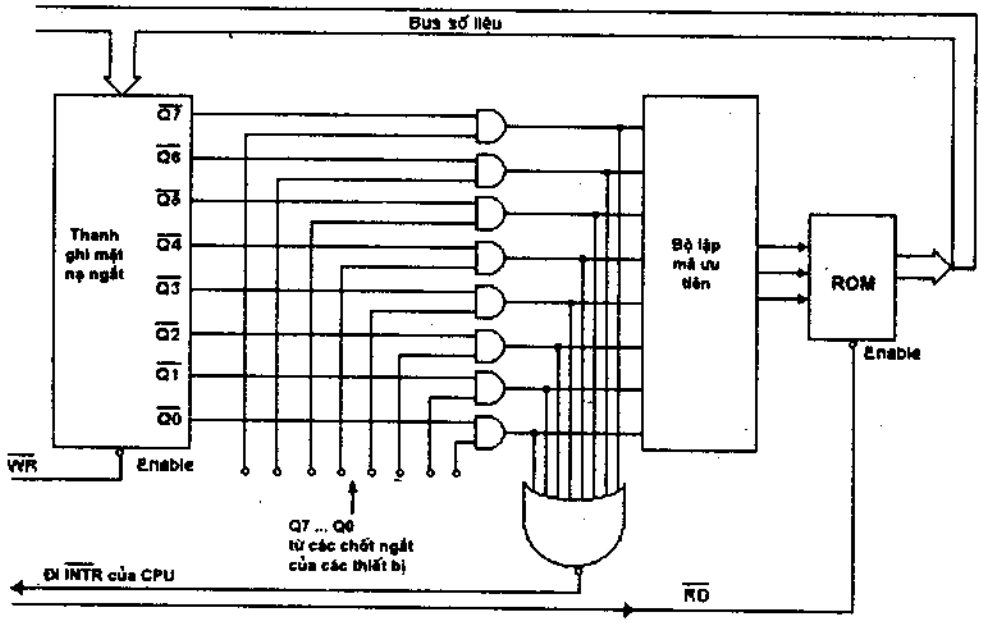


Hình 6.11 Sơ đồ khối của ngắt theo móc xích.



Hình 6.12 Một thí dụ về sơ đồ móc xích.

• Phương pháp xác định nguồn ngắt nhanh nhất là sử dụng *mạch điều khiển ngắt ưu tiên* PIC (Priority-Interrupt Controller). Các mạch PIC thường được tích hợp ngay trong một chip (thí dụ chip PIC-8259A, sẽ được trình bày rõ trong chương sau). Sơ đồ khối cấu hình đơn giản của một mạch PIC như hình 6.13.



Hình 6.13 Sơ đồ điều khiển ngắt ưu tiên.

Trong sơ đồ này, một chốt ngắt không bị che phát ra tín hiệu yêu cầu ngắt tới vi xử lí và tín hiệu này cũng được đưa vào bộ lập mã ưu tiên. Lối ra của bộ lập mã là một tín hiệu 3 bit đại diện cho thiết bị có mức ưu tiên cao nhất trong số thiết bị yêu cầu ngắt. Tín hiệu này hoặc được đưa trực tiếp qua bus số liệu vào CPU để nhận dạng thiết bị cần được phục vụ hoặc khi cần tốc độ thông tin nhanh hơn thì nó được đưa vào ROM để địa chỉ của chương trình con phục vụ ngắt từ lối ra của nó được truyền trực tiếp tới CPU qua bus số liệu.

Phương pháp *vào/ra theo hỏi vòng* là một cơ chế đồng bộ vì các ngoại vi được phục vụ theo một trình tự nhất định. Trong khi đó *vào/ra theo ngắt* mang tính dị bộ vì các ngoại vi được phục vụ theo yêu cầu của chúng (yêu cầu ngắt) không theo một trật tự định trước. Nhờ đó ngắt là một xử lí rất linh hoạt và được dùng trong nhiều trường hợp đặc biệt như: dùng trong các hệ thống báo động với các cảm biến, ngắn mạch; báo mất nguồn nuôi trong PC; dùng trong các hệ thống console (bàn phím, màn hình) có cách thức truy cập ngẫu nhiên theo không gian và thời gian; dùng cho các báo lỗi nghiêm trọng về phần cứng v.v.. Tuy nhiên ngắt cũng có những hạn chế của nó như việc phát hiện lỗi trong phần mềm gọi ngắt phức tạp hơn cơ chế thăm dò, phải mất một thời gian nhất định trước khi phục vụ ngắt để cất giữ các thanh ghi và cờ trạng thái bằng các lệnh PUSH ...

c, Ngắt trong máy vi tính 80×86

Các ngắt trong họ vi xử lí 80×86 có thể được phát ra từ một trong ba nguồn sau:

Nguồn thứ nhất là tín hiệu bên ngoài được đưa vào chân *ngắt không thể che được* NMI hoặc được đưa vào chân *yêu cầu ngắt* INTR. Ngắt xảy ra do tín hiệu đưa vào một trong các chân nói trên được gọi là *ngắt cứng* (hardware interrupt). Nguồn ngắt thứ hai là các câu lệnh gọi ngắt INT. Các ngắt loại này được gọi là *ngắt mềm* (software interrupt). Nguồn ngắt thứ ba là các lỗi phát sinh trong quá trình thực hiện câu lệnh (thí dụ ngắt chia một số cho 0), CPU sẽ tự động ngắt chương trình đang chạy bằng ngắt số 0. Các ngắt xảy ra ngay trong CPU loại này được gọi là *ngoại lệ* (exception).

Vào cuối mỗi chu trình lệnh, 8086 sẽ kiểm tra xem có ngắt nào được yêu cầu không. Nếu có yêu cầu ngắt, 8086 sẽ trả lời theo các bước sau:

1. Giảm con trỏ ngăn xếp đi 2 và cất thanh ghi cờ vào ngăn xếp.
2. Không cho phép ngắt cứng từ chân INTR bằng cách xoá cờ ngắt IF trong thanh ghi cờ.
3. Xoá cờ bước TF trong thanh ghi cờ.
4. Giảm con trỏ ngăn xếp đi 2 và cất nội dung thanh ghi đoạn mã vào ngăn xếp.
5. Giảm con trỏ ngăn xếp đi 2 một lần nữa và cất nội dung thanh ghi con trỏ lệnh hiện thời vào ngăn xếp.
6. Thực hiện một lệnh nhảy gián tiếp *far jump* tới phần đầu của chương trình con phục vụ ngắt do người dùng viết.

Vi xử lí 8086 có thể phục vụ được tới 256 ngắt khác nhau, được đánh số từ 0 đến 255. Mỗi ngắt ứng với một chương trình con phục vụ ngắt và sẽ được thực hiện khi được gọi. Địa chỉ lôgic ô nhớ bắt đầu của mỗi chương trình này gọi là một *vectơ ngắt* dài 4 byte gồm địa chỉ đoạn và địa chỉ offset 16 bit , địa chỉ offset được đặt trước địa chỉ đoạn. 256 địa chỉ này được lưu trữ lần lượt trong vùng nhớ thấp nhất của bộ nhớ gọi là *bảng các vectơ ngắt* có độ dài là 4*256 = 1024 byte từ địa chỉ 0000:0000 đến 0000:03FF. Như vậy, địa chỉ của chương trình xử lí ngắt số 0 nằm ở 4 ô nhớ từ 0000:0000 đến 0000:0003, địa chỉ của chương trình ngắt số 1 từ 0000:0004 đến 0000:0007 v.v... Do đó để xác định ô nhớ chứa địa chỉ bắt đầu cua chương trình con phục vụ ngắt ta chỉ việc nhân số ngắt với 4.

Có thể thay đổi vectơ ngắt làm cho nó trỏ đến một địa chỉ chương trình con do người sử dụng máy vi tính tự viết và chương trình này sẽ được chạy khi ngắt có số thứ tự tương ứng được gọi. Đó là vì nội dung của ROM BIOS, trong đó có cả bảng vectơ ngắt, được nạp sang RAM (quá trình làm RAM bóng - Shadow RAM) trong quá trình khởi động hệ thống và do vậy có thể dễ dàng thay đổi nội dung của bảng vectơ ngắt trong vùng RAM này.

Các ngắt cứng được tạo ra bằng việc đưa tín hiệu vào các chân INT và NMI trong khi các ngắt mềm được tạo ra bằng lệnh INT. Ta sẽ thảo luận qua về một số loại ngắt trước khi đi vào chi tiết các ngắt phần cứng trong PC.

Để gọi một ngắt mềm bằng chương trình, trước tiên phải nạp các giá trị *hàm ngắt* vào các thanh ghi chỉ định nào đó gọi là *các thanh ghi vào*. Sau khi thực hiện xong ngắt, các kết quả hoặc trạng thái của các bộ phận liên quan cũng sẽ được nạp vào một số thanh ghi chỉ định gọi là *các thanh ghi ra*. Thí dụ, đoạn chương trình sau trong Pascal gọi hàm 00h của ngắt 16h trong BIOS để đọc mã bàn phím và kí tự ASCII của phím được nhấn:

```
regs. AH := $00;          { nạp hàm 00h vào thanh ghi vào AH }
intr ($16, regs);         { gọi ngắt bàn phím 16h }
x:= regs.AH;              { gán nội dung thanh ghi ra chứa mã quét cho biến x }
y:= regs. AL;             { gán nội dung thanh ghi ra chứa kí tự ASCII cho biến y}
```

Cách thức gọi ngắt trong các ngôn ngữ bậc thấp và cao được tóm tắt trong phần 3 của sách. Bảng sau cho mối quan hệ giữa các số ngắt, địa chỉ chương trình con phục vụ ngắt và đầu vào yêu cầu ngắt IRQ trong 3 loại ngắt của các máy vi tính từ AT trở đi.

IRQ	Số ngắt	Địa chỉ CT con ngắt	Chức năng
	00	0EE5 : 0158	Chia cho không (ngoại lệ)
	01	0070 : 06F4	Chạy từng bước chương trình (ngoại lệ)
	02	0707 : 0016	Ngắt không che (ngắt cứng)
	03	0070 : 06F4	Điểm dừng (ngoại lệ)
	04		Tràn (ngoại lệ)
	05		In ra màn hình
	06		Dự trữ
	07		Dự trữ
IRQ0	08	0955 : 0000	PIT-8253 đạt tới điểm kết thúc đếm TC
IRQ1	09	CA51 : 1923	Bàn phím
IRQ2	0A		Yêu cầu ngắt từ PIC tớ
IRQ3	0B		COM2
IRQ4	0C		COM1
IRQ5	0D		LPT2
IRQ6	0E		Điều khiển ổ đĩa mềm
IRQ7	0F		LPT1
IRQ8	70		Đồng hồ thời gian thực trong RAM CMOS
IRQ9	71		Định hướng lại chương trình về ngắt 0Ah
IRQ10	72		Dành cho người sử dụng
IRQ11	73		Dành cho người sử dụng
IRQ12	74		Chuột PS/2
IRQ13	75		Đồng xử lí toán
IRQ14	76		Điều khiển ổ đĩa cứng

IRQ15	77		Dành cho người sử dụng
	10		BIOS - Chế độ đồ hoạ Video
	11		BIOS - Kiểm tra thiết bị
	12		BIOS - Kích thước bộ nhớ
	13		BIOS - Điều khiển đĩa
	14		BIOS - Thông tin nối tiếp
	16		BIOS - Bàn phím
	17		BIOS - Máy in
	19		BIOS - Khởi động lại
	1A		BIOS - Thời gian của ngày
	21		DOS - Dịch vụ DOS
	27		DOS - Chương trình thường trú TSR

Chương trình viết trong C sau đây xác định địa chỉ của các chương trình con ngắt.

Chương trình 6.1 Xác định địa chỉ chương trình con ngắt;

```
#i nclude    <stdio.h>
int   main (void)
{
int      far  *ptr;
int      seg, off, intr;
     ptr =(int  far  *) 0x00;
    puts (" Dia_chi_Ngat ");
    for  (intr=0; intr<16; intr ++)
    {
      off=*ptr;
       seg=*ptr;
       ptr ++;
       printf ( "&02d &04x:&04x\n", intr, seg, off );
  }
 return (0);
}
```

Dưới đây mô tả qua một vài chương trình con phục vụ ngắt trong 8086.

- ***Các ngắt mềm với số ngắt trong dải từ 0 đến 255***

Lệnh INT của 8086 có thể yêu cầu 8086 thực hiện một trong số tối đa 256 ngắt. Loại ngắt mong muốn được xác định bởi phần sau của câu lệnh. Thí dụ, câu lệnh INT 32 sẽ yêu cầu 8086 thực hiện đáp ứng đối với một ngắt số 32. Vi xử lí sẽ cất thanh ghi cờ vào ngăn xếp, xoá các cờ TF và IF, đồng thời cất địa chỉ của dòng lệnh tiếp theo vào ngăn xếp. Sau đó, nó sẽ lấy địa chỉ bắt đầu của thủ tục phục vụ ngắt cho vào các thanh ghi CS và IP từ bảng con trỏ ngắt trong bộ nhớ. Giá trị dành cho thanh ghi IP luôn nằm tại ô nhớ có địa chỉ được xác định bằng cách lấy kiểu ngắt nhân với 4, còn giá trị dành cho thanh ghi CS nằm tại ô nhớ có địa chỉ cao hơn 2 đơn vị. Với ngắt 32, giá trị dành cho thanh ghi IP được giữ tại địa chỉ 4 × 32 = 128 hay 80h, còn giá trị dành cho thanh ghi CS được giữ tại địa chỉ 82h trong bảng vector ngắt. Các ngắt mềm được phát ra bởi lệnh INT có rất nhiều ứng dụng. Một ứng dụng khác của các ngắt mềm là dùng để

kiểm tra các thủ tục phục vụ ngắt. Ví dụ, có thể dùng lệnh INT 0 để chạy thủ tục phục vụ ngắt chia một số cho 0 mà không cần phải chạy một chương trình chia thực sự. Một thí dụ khác, có thể dùng một lệnh INT 2 để chạy thủ tục phục vụ ngắt NMI. Điều đó cho phép kiểm tra thủ tục NMI mà không cần đưa một tín hiệu từ bên ngoài vào chân NMI của 8086.

♦ ***Các ngắt cứng INTR với số ngắt trong dải từ 0 đến 255***

Lối vào INTR của 8086 cho phép các tín hiệu bên ngoài ngắt một chương trình đang chạy. Tuy nhiên, khác với lối vào NMI, ngắt INTR có thể bị che (cấm) khiến cho yêu cầu ngắt không được thực hiện. Nếu cờ ngắt (IF) bị xoá thì lối vào INTR sẽ bị cấm. Cờ IF có thể bị xoá tại bất kì thời điểm nào bằng lệnh CLI (*Clear Interrupt*). Lối vào INTR sẽ được cho phép trong trường hợp cờ IF được lập. Cờ IF có thể được lập tại bất kì thời điểm nào bằng lệnh STI (*Set Interrupt*).

Khi 8086 được reset, cờ ngắt sẽ tự động bị xoá. Do đó, trước khi 8086 có thể đáp ứng tín hiệu yêu cầu ngắt trên lối vào INTR, ta phải lập cờ IF bằng một lệnh STI. 8086 được thiết kế như vậy là để các cổng, các bộ định thời, các thanh ghi, v.v... có thể được khởi tạo trước khi lối vào INTR được cho phép. Nói cách khác, cơ chế này cho phép có được một bộ vi xử lí 8086 trong trạng thái đã sẵn sàng trước khi bị ngắt.

Nhớ rằng cờ IF cũng phải bị xoá một cách tự động trong bất kì một thủ tục đáp ứng ngắt nào của 8086. Có hai lí do để làm điều đó. Thứ nhất, nó sẽ ngăn không cho các tín hiệu trên lối vào INTR ngắt một thủ tục phục vụ ngắt có quyền ưu tiên cao hơn đang trong quá trình thực thi. Tuy nhiên, nếu ta vẫn muốn tín hiệu trên lối vào INTR có thể ngắt một thủ tục ngắt khác đang được thực thi, thì có thể cho phép lại lối vào INTR bằng một lệnh STI tại bất kì thời điểm nào.

Lí do thứ hai để cấm lối vào INTR ở đầu một thủ tục phục vụ ngắt là để đảm bảo một tín hiệu trên lối vào INTR không làm cho 8086 liên tục tự ngắt chính mình. Lối vào INTR được kích hoạt bởi một điện áp ở mức cao. Nói cách khác, mỗi khi lối vào INTR ở mức cao và ngắt INTR được cho phép thì 8086 sẽ bị ngắt. Nếu ngắt INTR không bị cấm trong lần đáp ứng yêu cầu ngắt đầu tiên, 8086 có thể sẽ liên tục bị ngắt và không bao giờ có thể thực hiện được thủ tục đáp ứng nguyên nhân gây ra ngắt thực sự.

Lệnh IRET ở cuối mỗi thủ tục phục vụ ngắt sẽ khôi phục các cờ về lại trạng thái của chúng trước khi thủ tục ngắt được thực hiện bằng cách lấy thanh ghi cờ ra khỏi ngăn xếp. Điều này sẽ cho phép lại lối vào INTR. Nếu một tín hiệu ở mức cao vẫn còn hiện diện trên lối vào INTR, nó sẽ khiến cho 8086 lại bị ngắt một lần nữa. Nếu không muốn 8086 lại bị ngắt bởi cùng một tín hiệu, phải sử dụng một thiết bị phần cứng đảm bảo tín hiệu ấy được hạ xuống mức thấp trước khi cho phép lại ngắt INTR bằng lệnh STI hoặc trước lệnh IRET trong thủ tục phục vụ ngắt INTR. Đáp ứng của 8086 đối với một tín hiệu yêu cầu ngắt INTR có đôi chút khác biệt so với đáp ứng các ngắt loại khác. Sự khác biệt lớn nhất là đối với ngắt mềm là loại ngắt INTR được gửi tới 8086 thông qua một mạch phần cứng khác như *chip điều khiển ngắt ưu tiên*.

♦ ***Ngắt số 0: phép chia cho 0***

8086 sẽ tự động thực hiện ngắt số 0 nếu kết quả của phép toán DIV hoặc phép toán IDIV quá lớn so với thanh ghi đích. Với ngắt số 0, 8086 sẽ lưu thanh ghi cờ vào ngăn xếp, xoá các cờ IF và TF cũng như lưu địa chỉ trở về (CS và IP) vào ngăn xếp. Sau đó, nó lấy địa chỉ bắt đầu của thủ tục phục vụ ngắt cho thanh ghi CS và IP lần lượt tại các địa chỉ 00002h và 00000h trong bảng con trỏ ngắt. Vì đáp ứng cho ngắt số 0 của 8086 là hoàn toàn tự động và không thể cấm bằng bất kì hình thức nào nên phải làm chủ được nó trong các chương trình có sử dụng các lệnh DIV và IDIV. Chỉ có một cách là bằng cách nào đó đảm bảo kết quả sẽ không quá lớn so với thanh ghi kết quả. Đó là việc phải chắc chắn số chia khác 0 rồi sau đó mới thực hiện phép

chia theo từng bước, do đó mà kết quả của phép chia không bao giờ quá lớn. Một cách khác để làm chủ đáp ứng ngắt 0 của 8086 chỉ đơn giản là viết một thủ tục phục vụ ngắt nhằm thực hiện những nhiệm vụ mà ta mong muốn khi xảy ra một phép chia không hợp lệ. Ưu điểm của phương pháp này là không cần phải có một chương trình con thực hiện phép chia phức tạp trong chương trình chính của mình, 8086 sẽ tự động kiểm tra và chỉ thực hiện thủ tục ngắt khi có vấn đề xảy ra.

♦ ***Ngắt số 1: ngắt bước đơn*** (single-step)

Các chức năng bước đơn có trong một số chương trình giám sát và gỡ rối. Khi ra lệnh cho một hệ thống chạy bước một, nó sẽ chỉ thực hiện một câu lệnh rồi dừng lại. Nhờ đó, có thể kiểm tra được nội dung các thanh ghi và các ô nhớ. Nếu không có gì sai sót, có thể bảo hệ thống chạy câu lệnh tiếp theo. Nói cách khác, trong chế độ bước đơn, hệ thống sẽ dừng lại sau khi thực hiện một câu lệnh và chờ chỉ thị tiếp theo của người lập trình. Cờ bước (cờ bẫy - trap flag) và đáp ứng ngắt số 1 của 8086 cho phép thực hiện chức năng bước đơn một cách khá dễ dàng trong các hệ thống sử dụng vi xử lí 8086.

Nếu cờ bước được lập, 8086 sẽ tự động thực hiện ngắt 1 sau khi chạy xong một lệnh. Khi 8086 thực hiện ngắt 1, nó đẩy thanh ghi cờ vào ngăn xếp, xoá các cờ TF và IF, lưu địa chỉ của dòng lệnh tiếp theo vào ngăn xếp. Sau đó, nó sẽ lấy địa chỉ bắt đầu của thủ tục phục vụ ngắt 1 cho các thanh ghi CS và IP tại địa chỉ 00006h và 00004h.

Nếu muốn chạy trong chế độ bước đơn, cần phải thực hiện những nhiệm vụ sau: lập cờ bước, viết một thủ tục phục vụ ngắt có nhiệm vụ lưu tất cả các thanh ghi vào ngăn xếp để sau này có thể kiểm tra hoặc hiển thị chúng lên màn hình, và nạp địa chỉ bắt đầu của thủ tục phục vụ ngắt 1 vào các địa chỉ 00004h và 00006h. Một thủ tục bước đơn trên thực tế phụ thuộc rất nhiều vào hệ thống mà nó tác động tới. Sau đây là trình bày cách lập và xoá cờ bước. 8086 không có lệnh nào có thể lập hoặc xoá cờ bước một cách trực tiếp. Những thao tác cần phải thực hiện ở đây là: đẩy thanh ghi cờ vào ngăn xếp, đặt bit cờ bước về giá trị mong muốn, sau đó lấy lại thanh ghi cờ ra khỏi ngăn xếp. Sau đây là đoạn lệnh lập cờ bước:

```
PUSHF                          ; Cất các cờ vào ngăn xếp
MOV BP, SP                     ; Sao nội dung SP sang BP để dùng làm chỉ số
OR WORD PTR[BP+0], 0100H       ; Đặt bit TF
POPF                           ; Lấy lại thanh ghi cờ
```

Để xoá cờ bước, chỉ cần thay lệnh OR trong đoạn lệnh trên bằng dòng lệnh AND WORD PTR[BP+0], 0FEFFh.

Chú ý: phải dùng [BP+0] vì BP không thể được dùng như một con trỏ nếu không đi kèm với một độ dịch (displacement). Cờ bước phải được xoá khi 8086 thực hiện ngắt 1, do vậy mà mode bước đơn bị cấm trong khi 8086 đang thực hiện thủ tục phục vụ ngắt.

♦ ***Ngắt số 2: ngắt cứng không thể che được.*** 8086 sẽ tự động thực hiện một ngắt số 2 khi tác động một tín hiệu yêu cầu ngắt là một sườn xung lên vào chân NMI của nó. Khi thực hiện ngắt 2, 8086 sẽ cất các cờ vào ngăn xếp, xoá cờ IF và TF, đẩy địa chỉ của dòng lệnh tiếp theo vào ngăn xếp. Sau đó, nó sẽ lấy địa chỉ bắt đầu của thủ tục ngắt 2 cho CS và IP tại các địa chỉ 0000Ah và 00008h. Tên gọi ***không thể che được*** có nghĩa là đáp ứng cho ngắt 2 không thể bị cấm (bị che) bởi bất kì câu lệnh nào. Cũng vì chân tín hiệu này không thể bị cấm dù vô tình hay cố ý, cho nên người ta dùng nó để báo cho 8086 biết cần phải lưu ý đến một số điều kiện nào đấy xảy ra ở bên ngoài. Thí dụ, ta có thể nối một sensor áp suất từ một bình đun hơi nước

với chân NMI. Khi áp suất vượt quá một giá trị do ta đặt trước, sensor sẽ gửi một tín hiệu yêu cầu ngắt tới 8086. Thủ tục phục vụ ngắt 2 trong trường hợp này có thể khoá đường dẫn nhiên liệu tới bình đun, mở van xả áp suất và kéo còi báo động.

Một ứng dụng thường gặp khác của ngắt kiểu 2 là dùng để lưu trữ dữ liệu trong trường hợp hệ thống gặp sự cố về điện. Người ta dùng một số mạch điện tử để phát hiện thời điểm nguồn nuôi xoay chiều gặp sự cố và gửi một tín hiệu yêu cầu ngắt tới chân NMI. Do hầu hết các nguồn nuôi đều có các tụ lọc có điện dung lớn, nguồn điện áp một chiều vẫn sẽ được duy trì sau khi nguồn xoay chiều bị cắt trong vòng 50 ms. Khoảng thời gian này thừa đủ để thủ tục phục vụ ngắt 2 lưu lại các dữ liệu của chương trình vào trong những bộ nhớ RAM có nguồn nuôi dự phòng. Khi nguồn xoay chiều được cấp trở lại, dữ liệu của chương trình sẽ được khôi phục lại từ các bộ nhớ RAM có nguồn nuôi dự phòng này, do vậy, chương trình có thể chạy tiếp từ thời điểm bị dừng lại trước đó.

- ***Ngắt số 3: ngắt điểm dừng***

Ngắt số 3 được thực hiện bởi câu lệnh INT 3. Công dụng chính của ngắt 3 là cho phép chức năng tạo điểm ngắt (breakpoint) trong một hệ thống. Khi chèn vào chương trình một điểm ngắt, hệ thống sẽ thực hiện các câu lệnh trước điểm ngắt rồi sau đó nhảy tới thủ tục phục vụ điểm ngắt. Không như chức năng bước đơn là dừng chương trình sau khi thực hiện xong một lệnh, chức năng điểm ngắt chạy tất cả các lệnh phía trước cho tới khi gặp điểm ngắt và dừng chương trình. Khi gặp yêu cầu chèn một số điểm ngắt vào trong chương trình, phần lớn các hệ 8086 sẽ tạm thời thay byte lệnh tại địa chỉ cần chèn điểm ngắt bằng CCH - mã 8086 của câu lệnh INT 3. Khi 8086 thực hiện câu lệnh INT 3 này, nó cất thanh ghi cờ vào ngăn xếp, xoá các cờ TF và IF, đồng thời đẩy các giá trị của CS và IP lúc này đang trỏ tới dòng lệnh tiếp theo trong chương trình chính vào ngăn xếp. Sau đó, 8086 lấy địa chỉ bắt đầu của thủ tục phục vụ ngắt 3 cho vào các thanh ghi CS và IP tại các địa chỉ 0000Eh và 0000Ch. Thủ tục phục vụ ngắt 3 thông thường sẽ cất nội dung của tất cả các thanh ghi vào ngăn xếp. Tuỳ theo từng hệ thống mà sau đó nó sẽ hiển thị nội dung các thanh ghi lên màn hình và chờ lệnh tiếp theo từ người sử dụng; hoặc trong một số hệ thống đơn giản, chỉ trao quyền điều khiển cho người sử dụng mà thôi. Trong trường hợp này, có thể dùng lệnh Examine Register để kiểm tra xem nội dung các thanh ghi tại điểm ngắt đó có đúng không.

- ***Ngắt số 4: ngắt tràn***

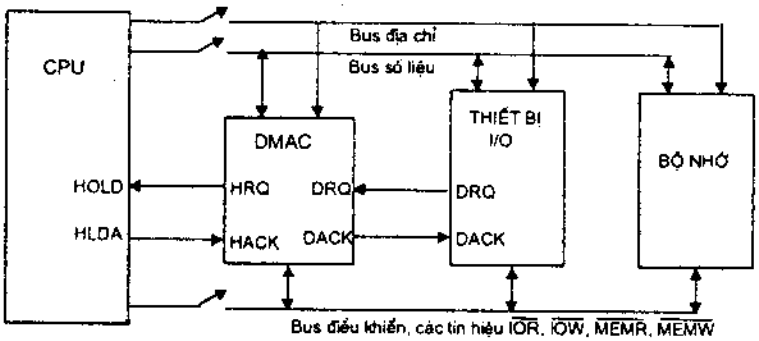
Cờ tràn (OF) của 8086 sẽ được lập nếu kết quả của một phép tính số học trên các số có dấu quá lớn so với thanh ghi đích hay một ô nhớ. Ví dụ, nếu cộng số 01101100_2 (bằng 108 trong hệ thập phân) với số 01010001_2 (bằng 81 trong hệ thập phân), kết quả sẽ là 10111101 (189 trong hệ thập phân). Đây là một kết quả đúng nếu chúng ta cộng hai số nhị phân không dấu. Nhưng kết quả này không còn đúng nữa đối với các số có dấu. Với các phép tính có dấu, số 1 tại vị trí bit nhiều ý nghĩa nhất cho biết kết quả là một số âm và được biểu diễn theo mã bù 2. Do đó, kết quả 10111101 trên thực tế biểu diễn giá trị -67 tính theo hệ thập phân, rõ ràng không phải là một kết quả đúng của phép cộng hai số +108 và +89.

Có hai cách chính để phát hiện và trả lời lỗi tràn trong chương trình. Cách thứ nhất là đặt một câu lệnh nhảy nếu có lỗi tràn JO (Jump if Overflow) ngay sau một lệnh tính toán số học. Nếu cờ tràn được lập do kết quả của phép tính số học đó, chương trình sẽ nhảy đến địa chỉ được xác định trong lệnh JO. Tại địa chỉ này, có thể đặt một chương trình con để đáp lại lỗi tràn theo cách mà mình muốn. Cách thứ hai để phát hiện và đáp lại lỗi tràn là đặt lệnh INTO *(Interrupt on Overflow)* ngay sau lệnh tính toán số học trong chương trình. Nếu cờ tràn không

được lập khi 8086 thực hiện lệnh INTO, lệnh này chỉ có chức năng đơn giản như một lệnh NOP. Tuy nhiên, nếu cờ tràn được lập biểu thị một lỗi tràn kết quả, 8086 sẽ thực hiện ngắt 4 sau khi kết thúc lệnh INTO. Khi 8086 thực hiện ngắt 4, nó sẽ cất thanh ghi cờ vào ngăn xếp, xoá các cờ TF và IF, đồng thời cất địa chỉ của dòng lệnh tiếp theo vào ngăn xếp. Sau đó, nó lấy địa chỉ bắt đầu thủ tục phục vụ ngắt cho vào các thanh ghi CS và IP tại địa chỉ 00012h và 00010h. Các câu lệnh trong thủ tục phục vụ ngắt sẽ thực hiện đáp ứng mong muốn đối với lỗi tràn kết quả. Ưu điểm của phương pháp dùng lệnh INTO và ngắt 4 là chương trình con đáp ứng lỗi tràn có thể được truy cập một cách dễ dàng từ bất kì chương trình nào.

6.3.2. Vào/ra bằng thâm nhập bộ nhớ trực tiếp DMA

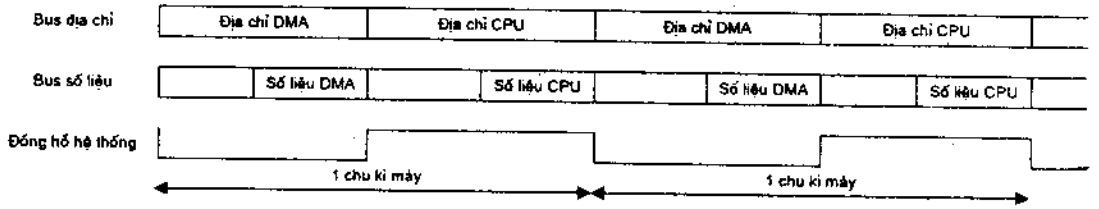
Trong phương pháp *vào/ra bằng chương trình* kể trên, dữ liệu thường phải được chuyển qua lại từ bộ nhớ qua CPU rồi tới thiết bị ngoại vi hoặc ngược lại bằng việc thực hiện từng lệnh một qua sự điều khiển của CPU với sự tham gia của các thanh ghi. Tốc độ trao đổi dữ liệu do đó không thể nhanh được. Trong một số trường hợp, thí dụ khi cần chuyển cả một mảng số liệu có tính chất giống hệt nhau giữa bộ nhớ và các ngoại vi nhanh như màn hình có thể dùng một phương pháp trao đổi thông tin trực tiếp giữa chúng mà không cần chuyển qua CPU: đó là phương pháp DMA (Direct Memory Access). Trong trường hợp này, CPU giao quyền điều khiển bus dữ liệu cho một chip điện tử gọi là chip DMAC (DMA Controller). CPU không cần phải đọc, giải mã và thực hiện các lệnh chuyển số liệu cũng như các lệnh điều chỉnh địa chỉ, kiểm tra độ dài của mảng dữ liệu nữa mà nó trao toàn quyền cho chip DMAC. Chip này sẽ tự tạo địa chỉ, tạo các tín hiệu đọc/viết bộ nhớ, đọc/viết ngoại vi, tính số từ đã chuyển trực tiếp giữa ngoại vi và bộ nhớ và thông báo cho CPU biết khi chuyển xong mảng số liệu quy định (kết thúc DMA). Các quá trình này được thực hiện hoàn toàn bằng các mạch điện tử (phần cứng) và lại được thực hiện trực tiếp giữa bộ nhớ và ngoại vi nên tốc độ trao đổi thông tin rất nhanh. Nói cách khác đây là phương pháp dùng phần cứng hoàn toàn. DMAC 8237 là một chip bổ trợ dùng để điều khiển quá trình DMA đã được mô tả kĩ trong giáo trình cấu trúc máy vi tính. Sơ đồ khối nối CPU với DMAC cùng bộ nhớ và thiết bị ngoại vi như hình 6.14.



Hình 6.14 Các tín hiệu điều khiển DMA.

Khi có yêu cầu trao đổi dữ liệu bằng DMA với bộ nhớ, thiết bị ngoại vi sẽ gửi tín hiệu yêu cầu DRQ tới chip DMAC. Chip này sẽ đưa tín hiệu yêu cầu treo HRQ đến chân HOLD của CPU. Nếu cho phép phục vụ DMA, vi xử lí sẽ tự treo (tự đưa bus của nó vào trạng thái trở kháng cao) và trả lại bằng xung ghi nhận HLDA đến chân HACK của chip DMAC để thông báo cho phép DMAC sử dụng bus. Lúc đó chip này sẽ truyền tín hiệu DACK tới thiết bị ngoại vi để cho phép thiết bị này trao đổi dữ liệu trực tiếp với bộ nhớ. Khi kết thúc quá trình DMA, thì chip DMAC sẽ đưa tín hiệu yêu cầu treo HRQ trở về mức không tích cực (HRQ = 0).

Có hai phương pháp thực hiện DMA: cách thứ nhất là CPU chuyển nhượng bus cho DMAC rồi tự treo; cách thứ hai là DMAC lấy lén chu kì, trong đó DMAC tận dụng những khoảng thời gian trong một chu kì bus mà CPU không truy cập bộ nhớ (khi CPU bận phát ra tín hiệu địa chỉ cho các chu kì đọc hoặc viết bộ nhớ) để thực hiện phép DMA. Phương pháp thứ hai đòi hỏi việc đồng bộ thời gian rất nghiêm ngặt. Hình 6.15 là giản đồ thời gian của phép DMA bằng phương pháp lấy lén chu kì.



Hình 6.15 DMA lấy lén chu kì, các vùng gạch thẳm chỉ thị số liệu chưa ổn định.

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

6.1 Một bản mạch ghép nối được cắm trên khe cắm mở rộng ISA, địa chỉ các thanh ghi bên trong bản mạch đó thuộc loại địa chỉ ánh xạ bộ nhớ hay địa chỉ tách biệt ? Tại sao ?

6.2 Trong các trường hợp sau, trường hợp nào bắt buộc phải cần các tín hiệu móc nối:

- CPU truyền các byte kí tự ASCII ra máy in;
- CPU đọc ô nhớ trong bộ nhớ chính;
- CPU điều khiển 2 bit B0 và B1 của chip bổ trợ PPI-8255 khi phát âm ra loa PC;
- Máy vi tính viết các byte dữ liệu ra một cổng vào/ra để điều khiển bật/tắt một bộ đèn điều khiển giao thông.

6.3 So sánh hai phương pháp hỏi vòng và ngắt.

6.4 Hiểu thế nào về ngắt véctơ ? Tại sao dùng ROM địa chỉ lại cho phép thông tin nhanh hơn không dùng ?

6.5 Liệt kê và mô tả khái quát các bước 8086 sẽ thực hiện để đáp ứng một ngắt.

6.6 Mô tả mục đích của bảng véctơ ngắt trong một hệ 8086

6.7 Các địa chỉ nào của bảng véctơ ngắt được dùng cho ngắt 2

6.8 Địa chỉ của chương trình con phục vụ ngắt 4 là 0010:0082. Hãy xác định thứ tự và địa điểm địa chỉ này được cất trong bảng véctơ ngắt

6.9 Địa chỉ 00080h trong bảng véctơ ngắt chứa giá trị 4A24h còn địa chỉ 00082h chứa giá trị 0040h. Các ô nhớ này chứa địa chỉ của ngắt nào?

6.10 Địa chỉ bắt đầu của chương trình con phục vụ ngắt là bao nhiêu ?

6.11 Mô tả sơ lược các điều kiện để 8086 thực hiện một trong các ngắt sau: ngắt 0, ngắt 1, ngắt 2, ngắt 3 và ngắt 4. Vì sao cần phải PUSH tất cả các thanh ghi sẽ được sử dụng tại đầu mỗi thủ tục ngắt và POP chúng ở cuối thủ tục?

6.12 Vì sao phải dùng lệnh IRET thay cho lệnh RET thông thường ở cuối thủ tục phục vụ ngắt?

6.13 Ưu nhược điểm của hai phương pháp vào/ra bằng chương trình và vào/ ra bằng DMA ? Trong trường hợp nào thì vào/ra bằng DMA không có ý nghĩa ?

6.14 Tại sao việc truyền số liệu bằng DMA lại nhanh hơn bằng chương trình ?

CHƯƠNG 7. GHÉP NỐI VI XỬ LÍ VỚI CÁC CHIP BỔ TRỢ

Các chip bổ trợ là các mạch điện thường được định địa chỉ qua các cổng vào/ra nên có thể nhìn nhận chúng như là các ngoại vi ghép nối với vi xử lí để tạo nên một hệ máy vi tính chuẩn. Các địa chỉ này là cố định trong các máy vi tính dùng vi xử lí từ 8086/88 cho tới các vi xử lí như Pentium hiện nay. Các chip bổ trợ đã được mô tả trong giáo trình cấu trúc máy tính, ở đây ta chỉ khảo sát chúng theo chức năng ghép nối. Đó là việc tạo ra các bộ định thời chính xác trong máy tính, điều khiển ngắt, thực hiện phương pháp truyền số liệu bằng DMA và lưu trữ cấu hình của máy vi tính v.v... Những chip bổ trợ cho các mục đích phức tạp khác được đề cập riêng trong các chương sau.

7.1. CÁC BỘ ĐỊNH THỜI TRONG MÁY VI TÍNH

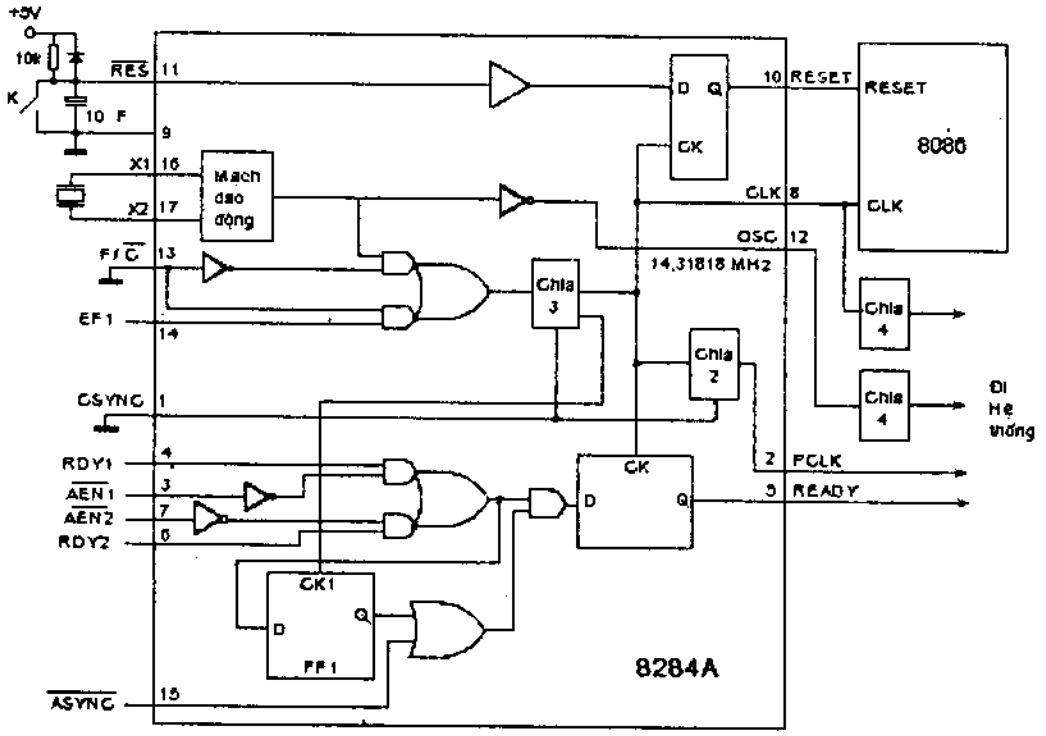
Các máy phát nhịp đồng hồ (clock) và các bộ định thời (timer) tạo nên nhịp tim của PC. Các chức năng chính của máy tính là tính toán và truyền dữ liệu xảy ra từng bước một với các xung được phát ra bởi một đồng hồ điện tử.

7.1.1. Xung nhịp đồng hồ cho hệ thống máy vi tính

Có lẽ các sự kiện thời gian cơ bản nhất trong PC là sự hoạt động theo từng nhịp một của CPU. Tốc độ của các nhịp này được xác định bởi tần số của một mạch dao động đặc biệt phát ra các xung nhịp tần số cao. Tần số này gọi là tốc độ đồng hồ của CPU và nó quyết định việc CPU thực hiện các nhiệm vụ của mình nhanh đến đâu. Với mỗi nhịp đồng hồ, CPU sẽ thực hiện được một phần lệnh máy. Tất cả các lệnh của CPU đòi hỏi 1 hoặc nhiều hơn 1 chu kì nhịp đồng hồ.

Xung nhịp đồng hồ dùng cho CPU và hệ thống đòi hỏi phải chính xác và ổn định. Vì vậy mạch tạo xung thường được xây dựng trên cơ sở các máy phát sóng thạch anh có độ ổn định tần số rất cao (độ bất ổn định tương đối cỡ 10^{-6} khi không có ổn nhiệt và lên tới 10^{-8}-10^{-9} khi có ổn nhiệt). Hình 7.1 là một sơ đồ dùng vi mạch 8284A tạo xung nhịp đồng hồ CLK cho vi xử lí 8086, nó còn cung cấp xung nhịp cho toàn bộ hệ thống máy vi tính và dùng trong quá trình khởi động lại của CPU. Thạch anh có tần số cộng hưởng f_t = 14,31818 MHz (do đã thường được dùng trong các mạch vô tuyến truyền hình) là một linh kiện nối với mạch dao động bên trong chip 8284 qua 2 chân X1 và X2. Xung nhịp phát ra sẽ được chia 3 để được các xung đồng hồ cấp cho CPU là 4,77 MHz. Chia 4 sẽ cho các xung đồng hồ 3,57955 MHz cấp cho bản mạch ghép nối màn hình CGA (tần số này chính bằng tần số của tín hiệu sóng mang phụ màu trong TV) và chia 12 sẽ cho xung 1,19318 MHz dùng cho các *mạch định thời hệ thống* trong PC. Nhấn công tắc K sẽ tạo ra xung RESET để khởi động toàn bộ hệ thống. Các đầu vào còn lại là các tín hiệu bổ sung cho 8284A như EF1 là lối vào xung nhịp ngoại; *F /C* là chọn loại nguồn nhịp (đầu này ở mức cao khi chọn nhịp ngoại, mức thấp khi chọn nguồn phát từ thạch anh); $\overline{AEN1}$ và $\overline{AEN2}$ để kiểm tra các tín hiệu vào READY; RDY1 và RDY2 là đầu vào tín hiệu READY từ ngoại vi; CSYNC và $\overline{ASYNC}$ dùng để tạo tín hiệu READY đồng bộ v.v...

Thời gian thực hiện các lệnh máy - lệnh vĩ mô - trong các chế độ địa chỉ khác nhau đòi hỏi số chu kì nhịp đồng hồ khác nhau. Khi viết các chương trình cần các thao tác chính xác về thời gian phải nắm rõ vấn đề này. Tất nhiên sai số về thời gian phải chịu cho mỗi chu kì lệnh sẽ bằng tích của sai số nhịp đồng hồ (như trên cỡ 10^{-6}) với số nhịp trong một chu kì lệnh.



Hình 7.1 Một mạch tạo xung đồng hồ cho PC.

7.1.2. Ba bộ định thời hệ thống trong máy PC

Kể từ máy PC/XT đến nay, các máy vi tính IBM PC đều dùng chip định thời khả trình PIT-8253 với 3 bộ đếm lùi 16 bit, có thể chương trình hoá được, tham gia vào việc thực hiện 3 chức năng quan trọng được thảo luận dưới đây. Lối vào CLK của cả 3 bộ đếm đều được cấp xung đồng hồ có tần số cố định là 1,193180MHz được phát ra từ bộ dao động thạch anh của máy vi tính. Do vậy với những máy PC hiện nay có tần số xung đồng hồ cho vi xử lí cao hơn thì phải dùng các bộ chia tần có hệ số chia lớn hơn 12 để vẫn đảm bảo có xung đồng hồ với tần số này. Dưới đây là tóm tắt hoạt động của 3 bộ định thời đó.

- ***Bộ đếm 0: cập nhật đồng hồ hệ thống***

Khi ROM BIOS khởi động hệ thống, nó đặt bộ đếm 0 hoạt động ở chế độ đếm 2, đếm nhị phân và nạp số đếm ban đầu (chính là hệ số chia tần) là số 0. Lối vào $GATE_0$ được đặt cố định ở mức lôgic 1 để luôn cho phép bộ đếm hoạt động. Như vậy lối ra của bộ đếm 0 sẽ có các xung đồng hồ có tần số 1.193.180 / 2^{16} = 18,206 Hz. Sườn lên của các xung này chính là các tín hiệu yêu cầu ngắt được đưa tới lối vào IRQ0 của chip điều khiển ngắt cứng PIC-8259. Chip sẽ gửi yêu cầu đó tới CPU để thực hiện ngắt số 08h trong một giây là 18,206 lần. BIOS chứa một chương trình con phục vụ ngắt 08h, ngắt này làm những việc như sau:

- Tăng lên 1 giá trị của một bộ đếm 4 byte đặt tại vùng địa chỉ dữ liệu của BIOS là 0040:006C đến 0040:006F. Thời gian trôi qua kể từ khi bật PC sẽ được tính bằng giá trị hiện tại của bộ đếm này nhân với khoảng thời gian giữa 2 lần gọi ngắt kề nhau là 1/18,206 giây. Muốn tính ra thời gian thực thì phải cộng với giá trị thời gian thực ở thời điểm bật máy PC. Khi trôi qua 24 giờ, bộ đếm sẽ được đặt lại bằng 0 và sau đó lại được tiếp tục cập nhật. Có thể đọc và viết giá trị của 4 byte này thông qua ngắt 1A của BIOS.

- Giảm đi 1 giá trị của một bộ đếm 1 byte ở địa chỉ 0040:0040. Khi bắt đầu thao tác với một ổ đĩa mềm, BIOS đặt giá trị ban đầu của byte đó là 37h. Nếu giá trị byte đó đạt tới 0 thì chương trình ngắt sẽ phát ra một lệnh ngắt điện, dừng quay môtơ của ổ đĩa mềm nếu nó đang quay. Như vậy mỗi môtơ chỉ quay được liên tục khoảng 2 giây.

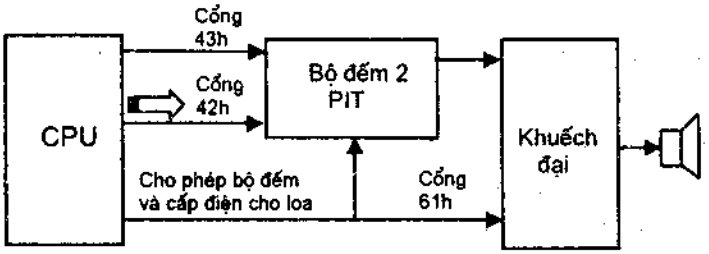
- Gọi ngắt mềm 1Ch (cho chạy lệnh INT 1Ch). Thường vectơ ngắt của ngắt này chỉ tới một lệnh đơn giản là lệnh kết thúc ngắt mà không làm gì cả IRET. Nếu người sử dụng muốn chạy một chương trình con của riêng mình 18,206 lần trong một giây thì có thể thay đổi nội dung của véctơ ngắt để chỉ tới một địa chỉ chương trình đó. Thường điều này được áp dụng cho việc kích hoạt các chương trình thường trú TSR như để kiểm tra trạng thái của ổ đĩa cứng hoặc mềm cho việc đọc dữ liệu, hiện thời gian thực hoặc phát âm ra loa độc lập với CPU v.v...

- ***Bộ đếm 1: làm tươi bộ nhớ***

Bộ đếm được hoạt động ở chế độ đếm 3 với giá trị đếm ban đầu thường là 18 (12h). Sườn lên của các xung đồng hồ lối ra có tần số 1.193.180 / 18 = 66.288 Hz được đưa tới kênh 0 của bộ điều khiển DMA-8237 cho phép tạo ra các *chu kì đọc giả* để làm tươi bộ nhớ DRAM. Do các cách làm tươi trong các máy tính rất khác nhau nên giá trị này có thể khác. Do vậy bằng các lệnh đọc lại (read-back) hoặc chốt (latch) bộ đếm 1 ta có thể xác định được tốc độ làm tươi bộ nhớ thực sự của PC.

- ***Bộ đếm 2: tham gia vào phát sóng âm ra loa***

Có thể trình bày đơn giản sơ đồ hệ phát sóng âm ra loa PC như hình 7.2. Bộ đếm 2 hoạt động ở chế độ 3, xung đồng hồ 1,19318 MHz cũng được đưa tới lối vào xung nhịp CLK2 nhưng lối vào GATE2 và công tắc điện tử cấp điện cho bộ khuếch đại lại được điều khiển bởi bit 0 và 1 của cổng B chip PPI-8255 có địa chỉ tại 061h. Chỉ khi 2 bit này cùng đồng thời nhận giá trị 1 hệ mới phát được sóng điện có tần số nào đó ra loa. Dải tần số phát ra thấp nhất từ 1.193.180 / 65535 = 18,206 Hz đến cao nhất bằng 1.193.180 / 1 = 1,19318 MHz. Tất nhiên dải âm thanh chỉ từ 16 đến 20.000 Hz. Nhớ rằng 6 bit còn lại của cổng B chip PPI-8255 dùng cho mục đích khác nên phải dùng phép xử lí bit với các toán tử bitwise và từ bitmask để không được làm ảnh hưởng đến chúng khi làm việc với loa PC.

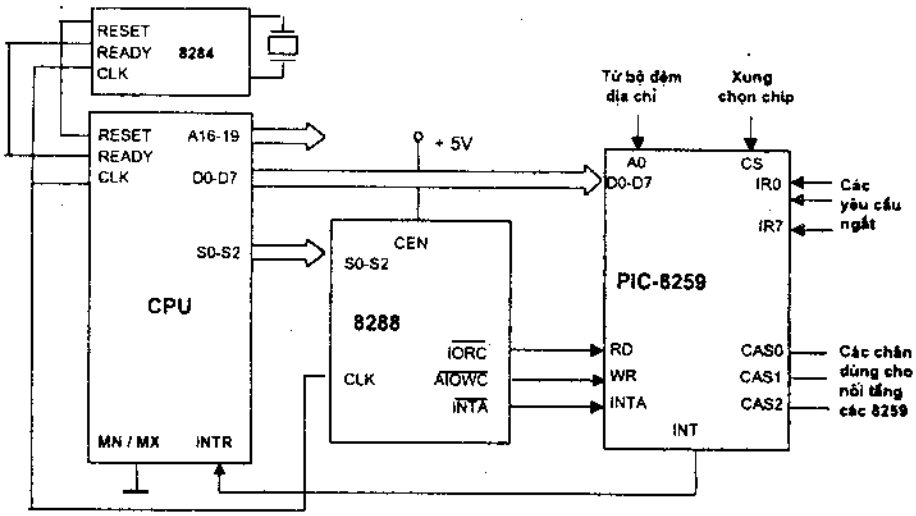


Hình 7.2 Hệ phát sóng âm ra loa.

7.2. NGẮT CỨNG VỚI CHIP ĐIỀU KHIỂN ƯU TIÊN NGẮT PIC-8259A

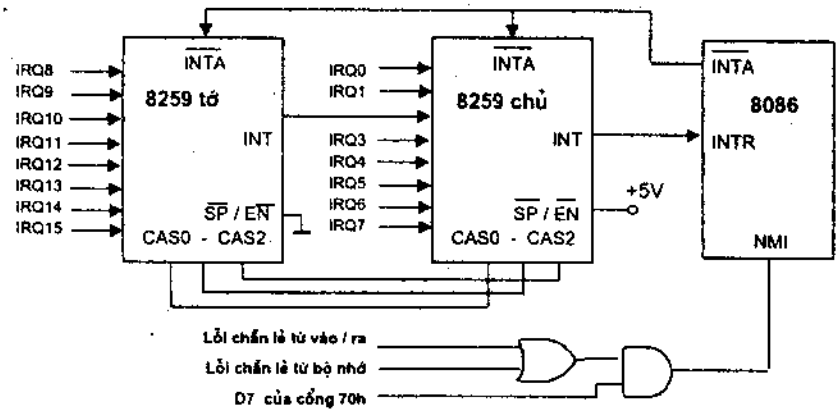
Với ngắt cứng, *tín hiệu yêu cầu phục vụ ngắt* hay *tín hiệu sẵn sàng* (ready) từ phía ngoại vi được nối với một lối vào ngắt của máy tính. Theo đó máy tính có thể thực hiện các công việc khác cho tới khi nó bị ngắt bởi tín hiệu này từ một thiết bị ngoại vi. Chương trình con phục vụ ngắt sẽ nhập hoặc xuất dữ liệu trong vòng một vài micrô giây rồi quay trở về thực hiện tiếp chương trình bị ngắt. Như vậy các thao tác nhập xuất dữ liệu chỉ chiếm một phần rất nhỏ thời gian của bộ vi xử lí. Có nhiều ứng dụng được sử dụng ngắt cứng như việc đọc vào các kí tự

ASCII từ bàn phím theo cơ chế ngắt; đếm số ngắt từ một bộ định thời để tạo thành một đồng hồ thời gian thực theo giờ, phút, giây; phát hiện các tình huống khẩn cấp hay kiểm tra kết quả công việc dựa trên cơ chế ngắt v.v... Về nguyên tắc, mỗi một ứng dụng này yêu cầu một lối vào ngắt khác nhau nhưng với bộ vi xử lí 8086 chỉ có duy nhất hai lối vào ngắt NMI và INTR. Nếu ta dành lối vào NMI cho ngắt khi mất nguồn nuôi thì sẽ chỉ còn lại duy nhất một lối vào ngắt INTR cho tất cả các ứng dụng còn lại. Do đó với các ứng dụng mà các tín hiệu yêu cầu ngắt đến từ nhiều nguồn khác nhau, phải sử dụng một mạch bổ trợ gọi là bộ ***điều khiển ngắt ưu tiên PIC*** (Priority Interrupt Controller) để "dồn" các tín hiệu ngắt cứng vào một lối vào ngắt duy nhất trên bộ vi xử lí. Máy vi tính IBM/PC sử dụng chip bổ trợ PIC-8259A cho việc phục vụ ngắt cứng. Chip này giữ một vai trò quan trọng với các ngoại vi như bàn phím, cổng nối tiếp, ổ đĩa hay các bản mạch ghép nối mạng. Chip cho phép xử lí 8 ngắt đồng thời trong máy PC/XT. Sơ đồ ghép chip PIC với CPU trong chế độ MAX như hình 7.3.



Hình 7.3 Ghép nối PIC-8259 với CPU trong chế độ MAX.

Từ máy AT trở đi vì có 2 chip PIC được ghép tầng với nhau nên có thể cho phép xử lí đồng thời 15 yêu cầu ngắt. Công nghệ chế tạo chip là NMOS và yêu cầu cấp nguồn +5V. Hình 7.4 là sơ đồ nối hai PIC-8259A chủ và tớ với vi xử lí 8086.



Hình 7.4 Sơ đồ ghép nối 2 PIC-8259A với 8086.

Để trình bày cách thức hoạt động của 8259A trong một hệ 8086, trước hết chúng ta cần xem cách thức làm việc của lối vào INTR. Nếu cờ ngắt của 8086 được lập và lối vào INTR nhận được một tín hiệu ở mức cao, 8086 sẽ:

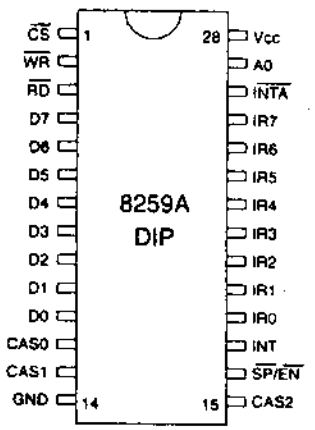
1. Phát ra hai xung chấp nhận ngắt từ chân $\overline{INTA}$ của nó đến chân $\overline{INTA}$ của 8259A. Các xung $\overline{INTA}$ báo cho 8259A gửi kiểu ngắt về cho 8086 theo đường bus số liệu.
2. Nhân kiểu ngắt nhận được từ 8259A với 4 để lấy địa chỉ của con trỏ ngắt trong bảng véctơ ngắt.
3. Cất các cờ vào ngăn xếp.
4. Xoá các cờ IF và TF.
5. Cất địa chỉ trở về vào ngăn xếp.
6. Lấy địa chỉ bắt đầu của thủ tục phục vụ ngắt từ bảng véctơ ngắt và nạp địa chỉ này vào các thanh ghi CS và IP.
7. Thực hiện thủ tục phục vụ ngắt.

Sơ đồ các chân tín hiệu của PIC 8259A như hình 7.5. Các yêu cầu ngắt được xử lí bởi 3 thanh ghi 8 bit, mỗi bit tương ứng với 8 đường ngắt từ IR0 đến IR7:

Thanh ghi che ngắt IMR (Interrupt Mask Register) được dùng để cấm (che) hoặc cho phép (không che) từng lối vào ngắt riêng biệt. Mỗi bit của nó tương ứng với một lối vào ngắt có cùng chỉ số. Để cho phép một lối vào ngắt, phải gửi đi một từ lệnh có bit tương ứng với lối vào ấy bằng 0.

Thanh ghi yêu cầu ngắt IRR (Interrupt Request Register) có nhiệm vụ theo dõi các lối vào đang yêu cầu phục vụ. Tín hiệu ngắt xuất hiện trên lối vào nào, bit tương ứng với lối vào ấy của IRR sẽ được lập. Tín hiệu yêu cầu ngắt phải tồn tại trên lối vào IR cho tới khi xuất hiện sườn xuống của xung $\overline{INTA}$ thứ nhất.

Thanh ghi ngắt đang được phục vụ ISR (In-Service Register) có nhiệm vụ theo dõi xem các lối vào ngắt nào đang được phục vụ. Với mỗi lối vào đang được phục vụ, bit tương ứng với nó trong ISR sẽ được lập.



Hình 7.5 Sơ đồ chân tín hiệu của PIC 8259A.

Mạch lôgic giải quyết ưu tiên PR (Priority Resolver) trên cơ sở nội dung của ba thanh ghi này quyết định yêu cầu ngắt INT hay không gửi đến CPU thông qua logic điều khiển (Control Logic) của 8259.

Chế độ làm việc của 8259 được xác lập bởi các *từ điều khiển* được nạp viết bởi chương trình. Có hai loại từ điều khiển: *từ điều khiển khởi động* ICW (Intialization Control Word) và *từ điều khiển hoạt động* OCW (Operation Control Word). Các thanh ghi của PIC-8259 trong PC/XT cũng như PIC-chủ trong máy AT được thâm nhập qua các cảng 020h và 021h và PIC-tớ trong AT được thâm nhập qua cảng 0A0h và 0A1h. Các địa chỉ và số liệu đọc và viết cho các thanh ghi này như sau:

Địa chỉ I/O IRQ0-IRQ7 (1)	Địa chỉ I/O IRQ3-IRQ15 (2)	Số liệu đọc	Số liệu viết
020h	0A0h	IRR	ICW1
		ISR	OCW2
		vectơ ngắt	OCW3
021h	0A1h	IMR	ICW2
			ICW3
			OCW1

(1) cho PC/XT và PIC-chủ trong AT; (2) cho PIC-tớ trong AT.

Bây giờ hãy khảo sát cụ thể 8259A hoạt động như thế nào trong quá trình này. Bus dữ liệu của máy tính cho phép 8086 gửi các từ điều khiển đến 8259A và đọc từ trạng thái của 8259A. Các lối vào điều khiển $\overline{RD}$ và $\overline{WR}$ sẽ điều khiển các quá trình truyền nhận này khi vi mạch được chọn bằng cách đặt lối vào $\overline{CS}$ của nó xuống mức thấp. Ngoài ra bus dữ liệu 8 bit còn cho phép 8259A gửi số ngắt cho 8086.

Nếu 8259A được phép (enable), tín hiệu ngắt đưa đến bất kì lối vào nào trong số 8 lối vào ngắt kí hiệu từ *IR0* đến *IR7* cũng khiến cho 8259A đưa lối ra INT của nó lên mức cao. Nếu chân này được nối với chân INTR của 8086 và cờ ngắt của 8086 được lập thì tín hiệu mức cao này sẽ gây ra đáp ứng ngắt INTR mà ta đã mô tả trước đây.

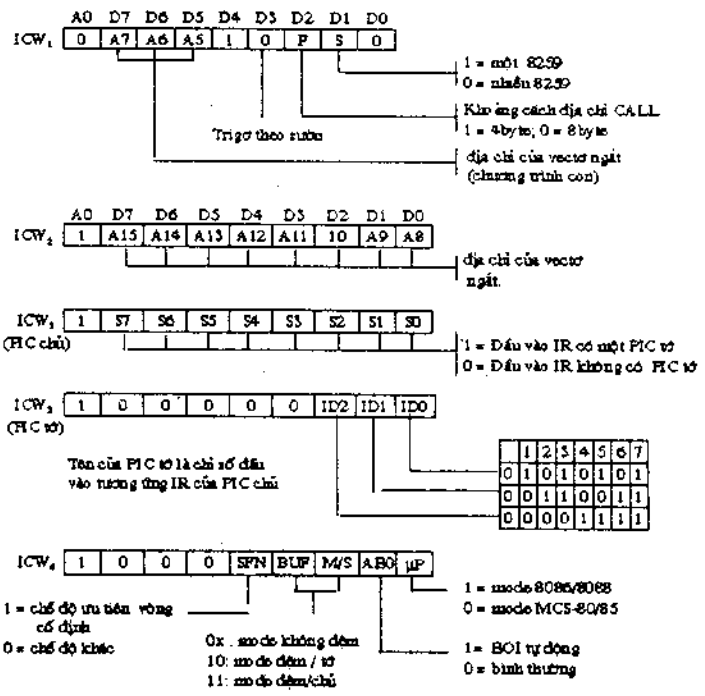
Lối vào $\overline{INTA}$ của 8259A được nối với lối ra $\overline{INTA}$ của 8086. Vi mạch 8259A sử dụng xung $\overline{INTA}$ đầu tiên phát ra từ 8086 để thực hiện một số thao tác, tuỳ thuộc vào mode hoạt động mà nó được lập trình. Khi nhận được xung $\overline{INTA}$ thứ hai từ 8086, 8259A sẽ xuất số ngắt lên bus dữ liệu 8 bit. Số ngắt mà nó gửi cho 8086 được xác định bởi lối vào IR nhận được tín hiệu yêu cầu ngắt và số ngắt mà người lập trình viết cho 8259A khi khởi động nó. Điều quan trọng ở đây là 8259A sẽ "dồn" các tín hiệu ngắt từ 8 nguồn khác nhau vào lối vào INTR của 8086 và gửi cho 8086 số ngắt tương ứng của một trong 8 lối vào ngắt ấy.

Bộ lôgic giải quyết ưu tiên có nhiệm vụ xác định xem yêu cầu ngắt trên một trong các lối vào IR của nó có quyền được phục vụ hay không và khi nào sẽ được phục vụ.

Ta hãy xét một thí dụ cụ thể. Giả sử IR2 và IR4 đang được cho phép và có một tín hiệu yêu cầu ngắt xuất hiện trên lối vào IR4. Yêu cầu ngắt trên IR4 sẽ lập bit 4 của thanh ghi IRR lên 1. Bộ lôgic giải quyết ưu tiên sẽ phát hiện ra bit này được lập và đi kiểm tra các bit trong thanh ghi ISR xem có lối vào ưu tiên cao hơn nào đang được phục vụ hay không. Nếu một lối vào có quyền ưu tiên cao hơn đang được phục vụ, biểu hiện bởi bit tương ứng với lối vào đó trong ISR đang được lập, bộ lôgic giải quyết ưu tiên sẽ không làm gì cả. Nếu không có ngắt nào có quyền ưu tiên cao hơn đang được phục vụ, bộ lôgic giải quyết ưu tiên sẽ kích hoạt mạch điện chức năng gửi một tín hiệu yêu cầu ngắt đến 8086. Khi 8086 trả lời bằng các xung $\overline{INTA}$, 8259A sẽ gửi đi số ngắt được gán cho IR4 khi khởi động 8259A. Như đã nói ở trên, 8086 sẽ dùng số ngắt nhận được từ 8259A để tìm và thực hiện thủ tục phục vụ ngắt được viết cho lối vào ngắt IR4.

Bây giờ giả sử trong khi 8086 đang thực hiện thủ tục phục vụ ngắt IR4, một tín hiệu ngắt lại xuất hiện trên IR2 của 8259A. Tín hiệu này sẽ lập bit 2 của IRR. Do ta giả thiết IR2 không bị che, bộ lôgic giải quyết ưu tiên sẽ phát hiện ra bit này trong IRR được lập và quyết định xem nó có gửi một tín hiệu yêu cầu ngắt khác tới 8086 không. Để làm điều đó, bộ lôgic giải quyết ưu tiên sẽ tìm kiếm trong thanh ghi ISR. Nếu một bit có quyền ưu tiên cao hơn trong ISR đang được lập tức là một ngắt có quyền ưu tiên cao hơn đang được phục vụ. Bộ lôgic giải quyết ưu tiên sẽ đợi cho tới khi bit có quyền ưu tiên cao hơn ấy bị xoá đi rồi mới gửi tín hiệu ngắt cho lối vào ngắt mới tới 8086. Nếu bộ lôgic giải quyết ưu tiên phát hiện ngắt mới có quyền ưu tiên cao hơn ngắt có quyền ưu tiên cao nhất đang được phục vụ, nó sẽ đặt bit tương ứng trong ISR lên 1 và kích hoạt mạch điện chức năng gửi một tín hiệu INT mới cho 8086. Với thí dụ ở đây, IR2 có quyền ưu tiên cao hơn IR4 nên bộ lôgic giải quyết ưu tiên sẽ đặt bit 2 của ISR lên 1 và kích hoạt mạch điện chức năng gửi đi một tín hiệu INT mới cho 8086. Nếu lối vào INTR của 8086 trước đó đã được cho phép lại bằng một lệnh STI ở đầu thủ tục phục vụ ngắt IR4 thì tín hiệu INT mới sẽ lại ngắt 8086 một lần nữa. Khi 8086 phát đi tín hiệu $\overline{INTA}$ thứ hai để trả lời ngắt này, 8259A sẽ gửi cho nó kiểu ngắt được gán cho IR2. 8086 sẽ sử dụng kiểu ngắt vừa nhận được để tìm và thực hiện thủ tục phục vụ ngắt IR2.

Cuối thủ tục phục vụ ngắt 2, ta phải gửi cho 8259A một từ lệnh có nhiệm vụ xoá bit 2 của thanh ghi ISR để các ngắt có quyền ưu tiên thấp hơn có thể được phục vụ. Sau đó, lệnh IRET sẽ trả quyền điều hành cho thủ tục phục vụ ngắt IR4. Cuối thủ tục phục vụ ngắt IR4, ta gửi cho 8259A một từ lệnh xoá bit 4 của ISR để các ngắt có quyền ưu tiên thấp hơn có thể được phục vụ. Lệnh IRET cuối thủ tục ngắt IR4 sẽ trả quyền điều khiển lại cho chương trình chính. Tất cả nghe có vẻ rất phức tạp nhưng đó chỉ là một trường hợp đặc biệt của các thủ tục lồng nhau. Nếu thủ tục phục vụ ngắt IR4 không cho phép lại lối vào INTR của 8086 bằng lệnh STI, 8086 sẽ không thể đáp ứng tín hiệu yêu cầu ngắt IR2 cho đến khi nó hoàn thành thủ tục IR4. Ta biết rằng PIC 8259 là một vi mạch khả trình nên việc lập trình cho nó phải qua 2 giai đoạn: khởi động bằng việc viết từ điều khiển khởi động rồi sau mới lập trình cho hoạt động trao đổi thông tin. Sau khi khởi động hệ thống, CPU ghi các từ điều khiển khởi động vào các thanh ghi ICW và các từ điều khiển hoạt động vào các thanh ghi OCW của PIC. Định dạng của các từ điều khiển khởi động ICW như hình 7.6.



Hình 7.6 Định dạng các từ điều khiển khởi động 8259A.

Bước đầu tiên trong quá trình khởi động bất kì vi mạch nào cũng là tìm ra địa chỉ cơ sở hệ thống của vi mạch ấy từ sơ đồ hoặc từ bản đồ bộ nhớ của hệ thống. Bước tiếp theo là tìm ra các địa chỉ nội bộ trong vi mạch. Đối với vi mạch 8259A, hai địa chỉ nội bộ của nó được chọn lựa bởi mức cao hay thấp trên chân A0; thí dụ nếu chân A0 được nối với đường địa chỉ A1 của hệ thống thì các địa chỉ nội bộ tương ứng là 0 và 2. Tiếp theo, lấy địa chỉ nội bộ cộng với địa chỉ cơ sở của vi mạch để thu được địa chỉ hệ thống cho các thành phần khác nhau bên trong vi mạch. Tiếp theo, dựa vào định dạng từ điều khiển nêu trên để viết vào từ khởi động vào 8259A. Thường thì các từ lệnh ICW1 và ICW2 phải được ghi vào vi mạch 8259A của hệ thống. Nếu hệ thống có chứa các vi mạch 8259A tớ (chế độ nối tầng) thì từ lệnh ICW3 phải được truyền vào vi mạch chủ và một từ lệnh ICW3 khác phải được ghi vào vi mạch tớ. Nếu hệ thống là một hệ 8086 hoặc nếu muốn xác lập một số điều kiện đặc biệt nhất định thì sẽ phải ghi một từ lệnh ICW4 vào vi mạch chủ và vào mỗi vi mạch tớ. Điểm đáng lưu ý đầu tiên về định dạng của các ICW là các bit được kí hiệu là A0 ở đầu trái của các từ lệnh này không phải là một phần của từ lệnh thực sự. Bit này cho biết địa chỉ nội bộ nơi phải ghi từ lệnh đó vào. Theo lược đồ này, các từ lệnh ICW1 và ICW2 phải được ghi vào vi mạch 8259A của hệ thống. Nếu hệ thống có chứa các vi mạch 8259A tớ (chế độ nối tầng) thì từ lệnh ICW3 phải được truyền vào vi mạch chủ và

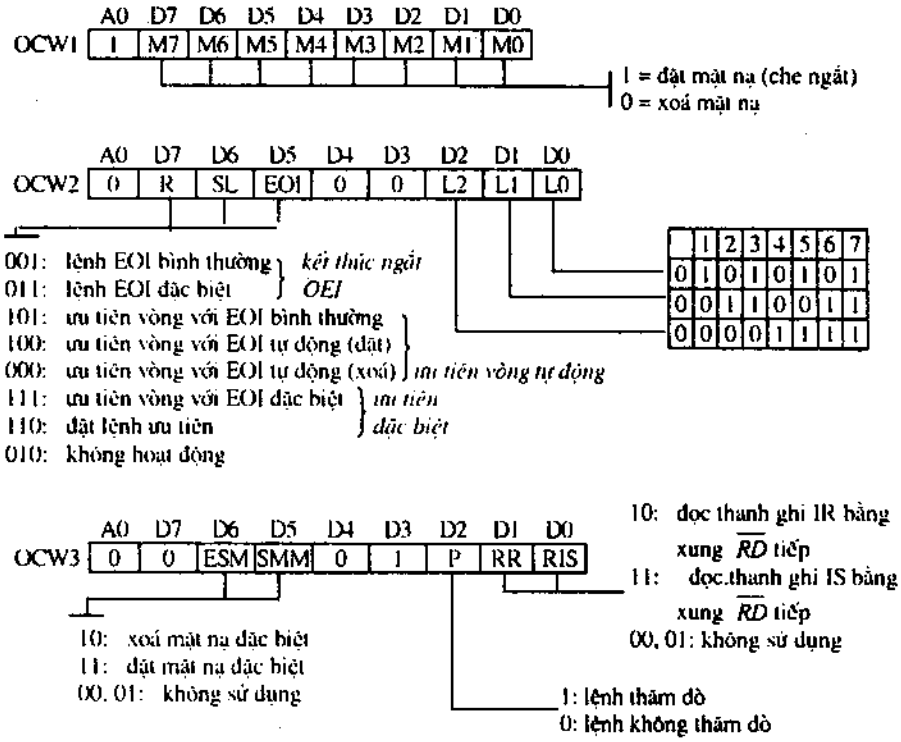
một từ lệnh ICW3 khác phải được ghi vào vi mạch tớ. Nếu hệ thống là một hệ 8086 hoặc nếu bạn muốn xác lập một số điều kiện đặc biệt nhất định thì bạn sẽ phải ghi một từ lệnh ICW4 vào vi mạch chủ và vào mỗi vi mạch tớ. Bây giờ hãy để ý đến định dạng (format) của các từ lệnh ICW này. Điểm đáng lưu ý đầu tiên về format của các ICW trên hình là các bit được kí hiệu A0 ở đầu trái của các từ lệnh này không phải là một phần của từ lệnh thực sự. Bit này cho biết địa chỉ nội bộ phải ghi từ lệnh đó vào. Ví dụ, A0 = 0 bên cạnh ICW1 cho biết ICW1 phải được ghi vào địa chỉ nội bộ 0, thí dụ vi mạch 8259A sẽ tương ứng với địa chỉ hệ thống FF00h.

Bước tiếp theo là xây dựng từ điều khiển. Bit LSB của ICW1 sẽ cho 8259A biết nó có cần chờ đợi ICW4 hay không. Thí dụ, nếu dùng 8259A trong một hệ 8086 thì cần phải ghi ICW4 vào vi mạch. Trong trường hợp này, đặt bit D0 bằng 1. Nếu chỉ sử dụng 1 vi mạch 8259A, đặt bit D1 bằng 1. Khi được sử dụng với 8086, bit D2 có thể nhận giá trị tuỳ ý, do đó đặt D2 bằng 0. Bit D3 được dùng để xác lập chế độ trigger theo mức hay trigger theo sườn xung. Trong chế độ trigger theo mức, chương trình ngắt được yêu cầu mỗi khi xuất hiện mức điện áp cao trên lối vào IR. Trong chế độ trigger theo sườn xung, tín hiệu trên lối vào IR phải chuyển từ mức thấp lên mức cao và duy trì ở mức cao cho đến khi ngắt được thực hiện. Khi hoạt động trong một hệ thống 8086 thì không cần quan tâm đến các bit D5, D6 và D7, do đó đặt chúng bằng 0 cho đơn giản. Như vậy, ICW1 đối với thí dụ này là 00010011. Trong một hệ 8086, ICW2 được dùng để báo cho 8259A biết số ngắt phải gửi đi để đáp ứng tín hiệu ngắt trên lối vào IR0. Để đáp ứng tín hiệu ngắt trên các lối vào IR khác, 8259A sẽ tự động cộng chỉ số lối vào với giá trị cơ sở này và gửi kết quả cho 8086 dùng làm số ngắt của lối vào IR đó. Do các số ngắt từ 0 đến 31 hoặc là các ngắt chuyên dụng hoặc là để dự trữ nên số 32 (thập phân) là số ngắt thấp nhất cho các ứng dụng. Nếu cho ICW2 bằng 00100000 hay 32 trong hệ thập phân, 8259A sẽ gửi giá trị này cho 8086 làm số ngắt để đáp ứng ngắt IR0. Với yêu cầu ngắt trên lối vào IR1, 8259A sẽ gán cho số ngắt 00100001 nhị phân hay 33 thập phân, yêu cầu ngắt trên lối vào IR2 sẽ được gán cho số ngắt 00100010 nhị phân hay 34 thập phân. Tương tự như vậy đối với các yêu cầu ngắt trên các lối vào IR còn lại. Trong bất kì ICW2 nào nạp vào 8259A, ba bit thấp nhất luôn luôn bằng 0 vì 8259A sẽ tự động cấp phát các bit này sao cho tương ứng với chỉ số của lối vào IR.

Nếu hệ thống có sử dụng vi mạch tớ, cần phải nạp ICW3 vào vi mạch chủ để báo cho cho nó biết các lối vào IR nào có nối với vi mạch tớ. Vi mạch chủ cần phải được thông báo điều này để nó biết với các lối vào IR nào thì phải xuất ID của vi mạch tớ lên các đường CAS0, CAS1 và CAS2. Bên cạnh đó, phải nạp ICW3 vào vi mạch 8259A tớ để cấp nhận dạng ID cho nó. Số ID cấp cho vi mạch tớ đúng bằng chỉ số của lối vào IR trên vi mạch chủ nối với vi mạch tớ ấy. Khi vi mạch chủ xuất số ID lên các đường CAS, vi mạch tớ sẽ nhận ra ID của nó và gửi số ngắt cần thiết cho 8086 khi nó nhận được một xung $\overline{INTA}$.

Trong thí dụ này, lí do duy nhất khiến ta phải ghi ICW4 là để 8259A biết nó đang hoạt động trong một hệ thống 8086. Việc này được thực hiện bằng cách đặt bit D0 của ICW4 bằng 1. Một bit đáng quan tâm khác trong từ lệnh này là D1, bit tự động kết thúc ngắt EOI (automatic end-of-interrupt bit). Nếu bit này được đặt, 8259A sẽ tự động xoá bit tương ứng với lối vào ngắt đang được phục vụ trong thanh ghi ISR khi nó nhận được xung $\overline{INTA}$ thứ hai. Mục đích của nó là để 8259A sau đó có thể đáp ứng tín hiệu ngắt trên các lối vào IR có quyền ưu tiên thấp hơn. Nói cách khác, lúc này một chương trình phục vụ ngắt có quyền ưu tiên thấp hơn cũng có thể ngắt chương trình có quyền ưu tiên cao hơn. Do ta không muốn tự động kết thúc ngắt nên ICW4 trong thí dụ là 00000001. Ngoài các thanh ghi từ điều khiển khởi động ICW, PIC 8259A còn có các thanh ghi từ điều khiển hoạt động OCW.

Định dạng của các từ này được chỉ ra trên hình 7.7.



Hình 7.7 Định dạng các thanh ghi từ điều khiển hoạt động OCW.

Từ lệnh OCW1 phải được nạp vào 8259A để cho phép các lối vào IR muốn đáp ứng. Thí dụ, giả sử chỉ muốn dùng hai lối vào IR2 và IR3. Để cho phép lối vào IR nào, bit có vị trí tương ứng với lối vào ấy trong OCW1 phải được đặt bằng 0, do đó ta cần đặt 2 bit D2 và D3 của OCW1 bằng 0, các bit còn lại được đặt bằng 1. Từ OCW1 của ta do vậy sẽ là 11110011. OCW2 chủ yếu được dùng để xoá một bit trong thanh ghi ISR. Tuy ta có thể thực hiện thao tác này tại bất kì thời điểm nào nhưng nó thường được thực hiện ở cuối mỗi thủ tục ngắt. ý nghĩa của việc làm này là khi bit ấy bị xoá đi, 8259A có thể đáp ứng các tín hiệu ngắt có quyền ưu tiên thấp hơn. Trong các hệ thống nhỏ, thường sử dụng từ lệnh kết thúc ngắt không xác định. Từ OCW2 trong trường hợp này là 00100000. Khi 8259A nhận được OCW này, nó sẽ tự động xoá bit tương ứng với lối vào IR đang được phục vụ trong thanh ghi ISR. Nếu muốn xoá một bit xác định trong ISR, đặt các bit D7, D6 và D5 bằng 011 và đặt vị trí của bit muốn xoá trong 3 bit thấp của OCW2. Cũng có thể dùng OCW2 để yêu cầu 8259A quay vòng đặc quyền của các lối vào IR sao cho sau khi được phục vụ, quyền ưu tiên của một lối vào sẽ bị hạ xuống mức thấp nhất. Nếu quan tâm, có thể tìm hiểu về OCW3 qua các tài liệu kĩ thuật của Intel.

Có 4 chế độ ngắt được phân biệt tóm tắt như sau:

Chế độ	Chức năng, ý nghĩa
Ưu tiên cố định (full nested).	Mỗi đầu ngắt được gắn mức ưu tiên cố định. Mức 0 là cao nhất, mức 7 là thấp nhất.
Ưu tiên vòng tự động (auto rotating).	Ưu tiên đồng đều, mỗi mức sau khi được phục vụ trở thành mức thấp nhất cho đến khi có ngắt tiếp xảy ra.
Ưu tiên vòng theo quy định (specific rotating).	Phần mềm của hệ quy định mức ưu tiên thấp nhất. Thứ tự ưu tiên được lập trên cơ sở quy định này.
Thăm dò ngắt (polled).	Phần mềm của hệ đọc thanh ghi trạng thái của 8259, từ đó xác định nguồn ngắt và quyết định phục vụ ngắt.

Trong máy PC, các chip PIC-8259A được xác lập chế độ hoạt động ngay khi khởi động hệ thống (booting) nên trừ khi cần thay đổi còn thì người sử dụng PC không cần khởi động lại chúng. Đoạn chương trình sau cho phép khởi động PIC với các xung yêu cầu ngắt được trigơ bằng sườn lên, cho phép các đường IRQ8 đến IRQ15.

```
# define  ICR   0x20          /* cổng điều khiển ngắt          */
# define  IMR   0x21          /* cổng thanh ghi che ngắt       */
  outportb (ICR, 0x13);       /* 0001 0011  sườn xung, 1 PIC,  đòi hỏi ICW4    */
  outportb (IMR, 0);          /* 0000 0000  dùng vectơ ngắt 08-0F cho IRQ0-IRQ7*/
  outportb (IMR, 9);          /* 0000 1001  ICW4: mốt đệm, EOI bình thường, 8088 */
```

Trong thanh ghi che ngắt IMR, một đường yêu cầu ngắt được cho phép bằng việc xoá bit tương ứng ở thanh ghi này về 0. Khi ngắt xảy ra, tất cả các đường yêu cầu ngắt khác bị cấm và chúng chỉ được cho phép lại bằng việc đặt bit EOI trên cổng điều khiển ngắt, thí dụ bằng lệnh outportb (ICR, 0x20);

Dưới đây là một chương trình hợp ngữ cho phép khởi động 8259A trong một bản mạch ghép nối SDK-86 của hãng Intel và có thể dùng nó làm mẫu để viết các chương trình phục vụ một hay nhiều ngắt. Chương trình này khởi động hệ thống để phát thời gian thực giờ, phút, giây từ một tín hiệu ngắt 1 kHz và đọc các mã ASCII từ bàn phím theo cơ chế ngắt. Lúc này tín hiệu PCLK 2.4576 MHz được coi là đã được nối với lối vào CLK của bộ đếm 0 của 8254, chân GATE của bộ đếm 0 cũng đã được nối lên mức cao và chân OUT của bộ đếm 0 được nối với lối vào IR0 của 8259A. Ngoài ra, giả sử tín hiệu báo phím được nhấn (key-pressed strobe) từ bàn phím được đưa vào chân IR2 của 8259A.

Chương trình 7.1

```
1                                ; CHƯƠNG TRÌNH 7.1
2                                ; Đoạn chương trình này trình bày cách khởi tạo bảng véctơ ngắt, bộ điều
3                                ; khiển ngắt ưu tiên 8259A và bộ đếm/định thời khả trình 8254
4
5   0000                           AINT_TABLE SEGMENT WORD PUBLIC
6   0000  02*(0000)                TYPE_64 DW 2 DUP (0)    ; Không gian dành cho địa chỉ của thủ tục clock
7   0004  02*(0000)                TYPE_65 DW 2 DUP (0)    ; Không sử dụng trong chương trình này
8   0008  02*(0000)                TYPE_66 DW 2 DUP (0)    ; Không gian chứa địa chỉ của thủ tục keyboard
9   000C                         AINT_TABLE ENDS
10
11  0000                         DATA SEGMENT WORD PUBLIC
12  0000  00                       SECONDS DB 0
13  0001  00                       MINUTES DB 0
14  0002  00                       HOURS DB 0
15  0003  03 E8                    INT_COUNT DW 03E8H              ; Số đếm ngắt 1 kHz
16  0005  64*(00)                  KEY_BUF DB 100 DUP(0)           ; Bộ đệm cho 100 kí tự ASCII
17  0069                         DATA ENDS
18
19  0000                         STACK_SEG SEGMENT               ; Không sử dụng từ khoá STACK vì sẽ
20  0000  64*(0000)                DW 100 DUP(0)                 ; chạy chương trình EXE2BIN
21                                 TOP_STACK LABEL WORD
22  00C8                         STACK_SEG ENDS
```

```
23
24  0000                    CODE SEGMENT PUBLIC
25                            ASSUME CS:CODE, DS:AINT_TABLE, SS: STACK_SEG
26  0000  B8 0000s              MOV AX, STACK_SEG                    ; Khởi tạo thanh ghi đoạn
27  0003  8E D0                 MOV SS, AX                           ; ngăn xếp
28  0005  BC 00C8r              MOV SP, OFFSET TOP_STACK  ;Khởi tạo thanh ghi con trỏ ngăn xếp
29  0008  B8 0000s              MOV AX, AINT_TABLE                   ; Khởi tạo thanh ghi đoạn
30  000B  8E D8                 MOV DS, AX                           ; dữ liệu
31                          ; Xác định địa chỉ cho các thủ tục phục vụ ngắt
32  000D  C7 06 0002r 0000s     MOV TYPE_64+2, SEG CLOCK             ; Nhập vào địa chỉ thủ tục clock
33  0013  C7 06 0000r 004Er     MOV TYPE_64, OFFSET CLOCK
34  0019  C7 06 000Ar 0000s     MOV TYPE_66+2, SEG KEYBOARD          ; Nhập vào địa chỉ của thủ tục
35  001F  C7 06 0008r 0055r     MOV TYPE_66, OFFSET KEYBOARD         ; keyboard
36                          ; Khởi tạo thanh ghi đoạn dữ liệu
37                            ASSUME DS:DATA
38  0025  B8 0000s              MOV AX, DATA
39  0028  8E D8                 MOV DS, AX
40                          ; Khởi tạo bộ điều khiển ngắt ưu tiên 8259A
41  002A  B0 13                 MOV AL, 00010011B          ; Trigger theo sườn xung, có 1 PIC, ICW4
42  002C  BA FF00               MOV DX, 0FF00H             ; Trỏ tới thanh ghi điều khiển 8259A
43  002F  EE                    OUT DX, AL                 ; Nạp ICW1
44  0030  B0 40                 MOV AL, 01000000B          ; Kiểu ngắt đầu tiên của 8259A là 64
45  0032  BA FF02               MOV DX, 0FF02H             ; Trỏ tới địa chỉ nạp ICW2
46  0035  EE                    OUT DX, AL                 ; Nạp ICW2
47  0036  B0 01                 MOV AL, 00000001B          ; ICW4, mode 8086
48  0038  EE                    OUT DX, AL                 ; Nạp ICW4
49  0039  BO FA                 MOV AL, 11111010B          ; OCW1 để cho phép IR0 và IR2
50  003B  EE                    OUT DX, AL                 ; Nạp ICW1
51                          ; Khởi tạo bộ đếm 0 của 8254
52  003C  B0 37                 MOV AL, 00110111B  ; Xung vuông 1 kHz, LSB trước MSB sau, BCD
53  003E  BA FF07               MOV DX, 0FF07H             ; Trỏ tới thanh ghi điều khiển của 8254
54  0041  EE                    OUT DX, AL                 ; Nạp từ điều khiển bộ đếm 0
55  0042  B0 58                 MOV AL, 58H                ; Byte thấp của số đếm
56  0044  BA FF01               MOV DX, 0FF01H             ; Trỏ tới thanh ghi số liệu của bộ đếm 0
57  0047  EE                    OUT DX, AL                 ; Nạp byte thấp
58  0048  BO 24                 MOV AL, 24H                ; Byte cao của số đếm
59  004A  EE                    OUT DX, AL                 ; Nạp byte cao
60                          ; Cho phép lối vào ngắt của 8086
61  004B  FB                    STI
62  004C  EB FE             HERE: JMP HERE                 ; Đợi ngắt
63
64  004E                    CLOCK PROC FAR
65                          ;                              ; Đoạn lệnh của thủ tục clock
66  004E  B0 20                 MOV AL, 00100000B          ; OCW2 ứng với chế độ EOI không xác định
67  0050  BA FF00               MOV DX, 0FF00H             ; Địa chỉ nạp OCW2
68  0053  EE                    OUT DX, AL                 ; Nạp OCW2 để kết thúc ngắt
69  0054  CF                    IRET
```

```
0055                    CLOCK ENDP
                        
0055                    KEYBOARD PROC FAR
                        ;                       ; Đoạn lệnh của thủ tục keyboard
0055  B0 20                  MOV AL, 00100000B   ; OCW2 ứng với chế độ EOI không xác định
0057  BA FF00                MOV DX, 0FF00H      ; Địa chỉ nạp OCW2
005A  EE                    OUT DX, AL          ; Nạp OCW2 để kết thúc ngắt
005B  CF                    IRET
005C                    KEYBOARD ENDP
005C                    CODE ENDS
                            END
```

Trong chương trình này, đầu tiên khai báo một đoạn có tên là AINT_TABLE để dành không gian cho các véctơ trỏ tới các thủ tục phục vụ ngắt. Ví dụ lệnh TYPE_64 DW 2 DUP(0) sẽ dành ra một từ trong bộ nhớ cho địa chỉ offset của thủ tục ngắt 64 và một từ cho địa chỉ đoạn cơ sở của thủ tục ấy. Lệnh TYPE_65 DW 2 DUP(0) dành ra một từ trong bộ nhớ cho địa chỉ offset và một từ cho địa chỉ đoạn cơ sở của thủ tục ngắt 65, v.v... Ngay sau đây sẽ thấy dùng các lệnh trong chương trình chính để nạp địa chỉ thực sự của các thủ tục ngắt vào các ô nhớ này. Chú ý, do cách hoạt động của chương trình EXE2BIN, đoạn AINT_TABLE phải nằm ở ngay đầu chương trình để được nạp vào địa chỉ tuyệt đối 0000:0100h để chương trình có thể chạy đúng khi nạp vào bản mạch SDK-86.

Bước tiếp theo trong chương trình là khai báo một đoạn dữ liệu và dành ra một số ô nhớ cho các biến đếm giờ, phút, giây và 100 kí tự đọc từ bàn phím. Sau đoạn dữ liệu, ta khởi tạo đoạn ngăn xếp.

Đầu chương trình chính, khởi tạo các thanh ghi đoạn ngăn xếp và thanh ghi con trỏ ngăn xếp. Sau đó khởi tạo thanh ghi DS cho trỏ tới bảng véctơ ngắt vừa thiết lập khi bắt đầu chương trình.

Bốn câu lệnh tiếp theo nạp địa chỉ của các thủ tục ngắt clock và keyboard vào các vị trí dành cho kiểu 64 và 66 trong bảng véctơ ngắt.

Sau khi nạp xong bảng véctơ ngắt, ta ASSUME DS:DATA và khởi tạo DS cho trỏ tới đoạn dữ liệu, nơi chứa số liệu của các thủ tục clock và keyboard.

Bước tiếp theo là khởi tạo 8259A như đã mô tả trong phần trước. Bit A0 bên cạnh ICW1 trên hình 7.6 bằng 0 nên ICW1 được ghi vào địa chỉ thấp trong số 2 địa chỉ của 8259A, tức địa chỉ FF00h. Trong thí dụ này, gán cho ngắt IR0 kiểu 64, do đó ICW2 phải là 01000000. Bit A0 bên cạnh ICW2 trên hình 7.6 bằng 1 nên ICW2 sẽ được ghi vào địa chỉ cao trong số 2 địa chỉ của 8259A, tức địa chỉ FF02H. Tương tự như vậy, ICW4 và OCW1 cũng được ghi vào địa chỉ hệ thống FF02h.

Phần tiếp theo trong chương trình chính khởi tạo bộ đếm 0 của 8254 ở mode 3, đếm theo mã BCD, và đọc/ghi byte thấp trước byte cao sau. Để tạo ra tín hiệu có tần số 1 kHz từ tín hiệu PCLK 2.4576 MHz, phải nạp vào bộ đếm 0 giá trị 2458. Đây không phải là tín hiệu có tần số chính xác bằng 1 kHz nhưng là tần số gần nhất ta có thể thu được. Tần số PCLK của bản mạch được chọn để có thể chia ra tần số nhịp chính xác điều khiển tốc độ baud chứ không phải để phát ra tần số 1 kHz cho đồng hồ thời gian thực.

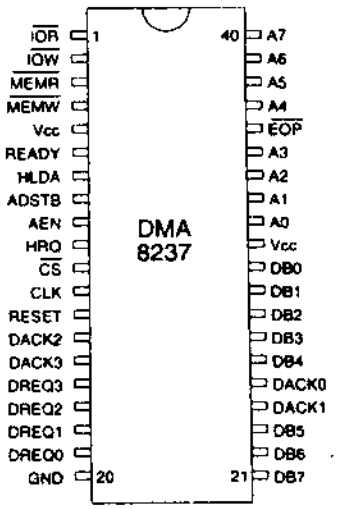
Cuối cùng, sau khi bộ định thời đã được khởi tạo, cho phép lối vào ngắt INTR của 8086 bằng lệnh STI để 8086 có thể đáp ứng các tín hiệu INT từ 8259A và đợi cho đến khi có một ngắt xảy ra bằng lệnh HERE: JMP HERE.

Với hai thủ tục phục vụ ngắt, ở đây chỉ trình bày phần khung và các lệnh EOI mà thôi. Các thủ tục thật sự được coi như bài tập. Cần lưu ý các thủ tục phục vụ ngắt phải được khai báo xa để trình biên dịch nạp cả hai giá trị cho CS và IP từ bảng véctơ ngắt. Ngoài ra cần lưu ý kết thúc ngắt ở cuối mỗi thủ tục.

Hãy nhớ lại trong phần trước, khi 8259A đáp ứng một tín hiệu IR, nó sẽ lập bit tương ứng trong ISR. Bit này cần được xoá đi trong thân hoặc ở cuối mỗi thủ tục ngắt để bộ giải quyết ưu tiên có thể đáp ứng các ngắt sẽ xảy ra có cùng mức ưu tiên hay có mức ưu tiên thấp hơn. Cuối mỗi chương trình con ngắt, điều này được thực hiện bằng cách nạp OCW2 vào 8259A. Giá trị OCW2 bằng 00100000 mà ta chọn sẽ báo cho 8259A xoá bit tương ứng với lối vào IR đang được phục vụ. Đây là một lệnh kết thúc ngắt (EOI) không xác định.

7.3. TRUYỀN SỐ LIỆU DMA BẰNG CHIP 8237A

Vi mạch 8237A của hãng Intel là thí dụ điển hình của một chip điều khiển thâm nhập bộ nhớ trực tiếp DMAC trong máy vi tính IBM/PC. Sơ đồ các chân tín hiệu của chip DMAC 8237A được chỉ ra trên hình 7.8. Theo sơ đồ khối hình 6.14 chương trước chỉ ra, ta thấy rằng CPU và DMAC chia xẻ thời gian sử dụng các bus địa chỉ, số liệu và điều khiển. Các chuyển mạch điện tử trong sơ đồ này chịu trách nhiệm đó. Khi hoạt động DMA, một ngoại vi (thí dụ, bộ điều khiển ổ đĩa) muốn gửi byte số liệu bằng phương pháp này thì trước hết nó phải gửi tín hiệu yêu cầu DMA là DREQ tới một lối vào (gọi là một kênh) của chip DMAC. Nếu kênh này không bị che, chip DMAC sẽ đáp lại bằng tín hiệu ghi nhận yêu cầu HRQ. Bộ xử lí CPU sẽ đáp ứng bằng cách thả nổi bus và gửi ra tín hiệu ghi nhận HLDA tới DMAC. Khi DMAC nhận được tín hiệu này, nó sẽ gửi một tín hiệu điều khiển các chuyển mạch kể trên tới vị trí DMA của chúng. Việc này làm ngắt CPU khỏi bus và nối DMAC với bus. Lúc này DMAC nhận quyền điều khiển bus và gửi ra địa chỉ nhớ mà số liệu từ ngoại vi cần viết vào. Tiếp theo nó gửi tín hiệu ghi nhận DMA là DACK0 tới ngoại vi để báo cho ngoại vi hãy sẵn sàng gửi tiếp số liệu. Cuối cùng, chip DMAC hạ cả hai đường $\overline{IOR}$ và $\overline{MEMW}$ trên bus điều khiển xuống mức lôgic thấp để cho phép ngoại vi xuất byte số liệu ra và viết vào bộ nhớ. Khi việc truyền số liệu hoàn tất, chip DMAC lấy đi tín hiệu ghi nhận yêu cầu HRQ và thả nổi bus. Các chuyển mạch quay về vị trí ban đầu cho phép CPU nắm lại quyền điều khiển bus cho đến khi có yêu cầu DMA tiếp theo. Việc truyền số liệu bằng DMA từ bộ nhớ tới ngoại vi cũng được thực hiện theo cách tương tự chỉ có điều chip DMAC lúc này sẽ làm tích cực đường điều khiển đọc bộ nhớ $\overline{MEMR}$ và đường điều khiển viết số liệu ra $\overline{IOW}$.



Hình 7.8 Sơ đồ chân tín hiệu của DMAC-8237A.

Các thanh ghi trong chip được chia thành hai nhóm: nhóm các thanh ghi điều khiển, trạng thái và nhóm các thanh ghi làm việc. Bảng sau liệt kê các thanh ghi đó. Khi không có yêu cầu DMA, 8237 liên tục kiểm tra các đường yêu cầu số liệu DRQ vào tất cả các chu kì đồng hồ, đồng thời kiểm tra chân chọn mạch $\overline{CS}$ để xem CPU có ý định gửi lệnh đến hoặc đọc/ghi thông số, trạng thái, chế độ DMA từ (hoặc vào) chip DMAC hay không. Nếu $\overline{CS}$ = 0 và HRQ = 0, 8237 tuân theo sự điều khiển bằng chương trình của CPU. A0-A3 là

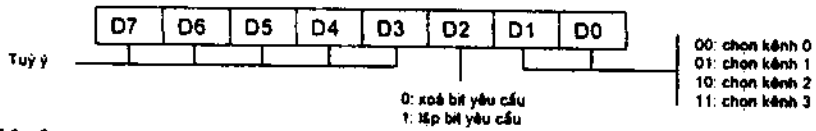
những đầu vào địa chỉ để chọn các thanh ghi trong của 8237. Những thanh ghi này được ghi vào hoặc đọc ra tuỳ thuộc vào tín hiệu điều khiển đọc/viết *IORD* và *IOWD* của CPU. Khi vào chế độ hoạt động, chip DMAC phát yêu cầu treo HRQ cho CPU. Số liệu được chuyển bằng DMA trong những chu kì này theo một trong những chế độ sau:

Chế độ chuyển từng từ (Single transfer Mode).
Chế độ chuyển mảng (Block transfer).
Chế độ chuyển theo nhu cầu (Demand Transfer).
Chế độ ghép tầng (Cascade mode).

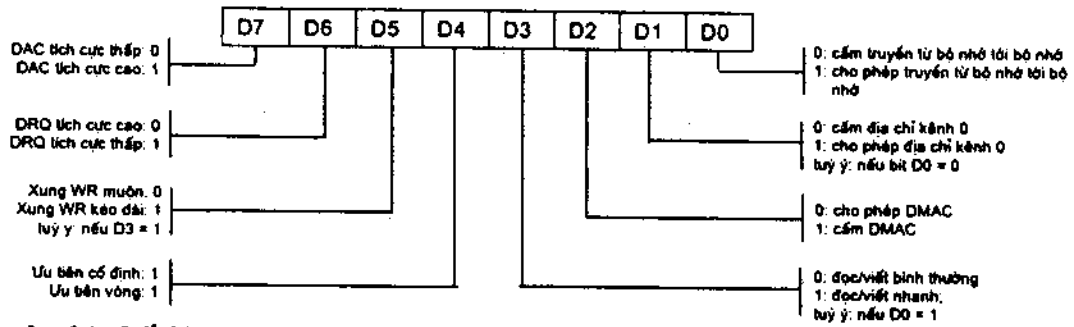
Tên thanh ghi	Độ dài (bit)	Số lượng
Địa chỉ gốc	16	4
Số từ gốc	16	4
Địa chỉ tức thời	16	4
Số từ tức thời	16	4
Trạng thái	8	1
Lệnh	8	1
Tạm thời	8	1
Chế độ	6	4

8237 có thể làm việc ở một trong hai chế độ ưu tiên khác nhau theo quy định phần mềm từ CPU là: chế độ ưu tiên cố định: kênh 0 có mức ưu tiên cao nhất, kênh 3 có mức thấp nhất; và chế độ ưu tiên vòng: kênh vừa được phục vụ sẽ có mức ưu tiên thấp nhất, lúc mới lập chế độ kênh 0 ở mức cao nhất. Trước khi để DMAC làm việc, chương trình cần phải lập cho nó trật tự ưu tiên, mặt nạ (che) cho từng kênh, v.v... và quy định cho từng kênh địa chỉ DMA, độ dài mảng (số đếm từ) bằng cách ghi vào các thanh ghi trong của 8237 theo quy định Định dạng của chúng như hình sau:

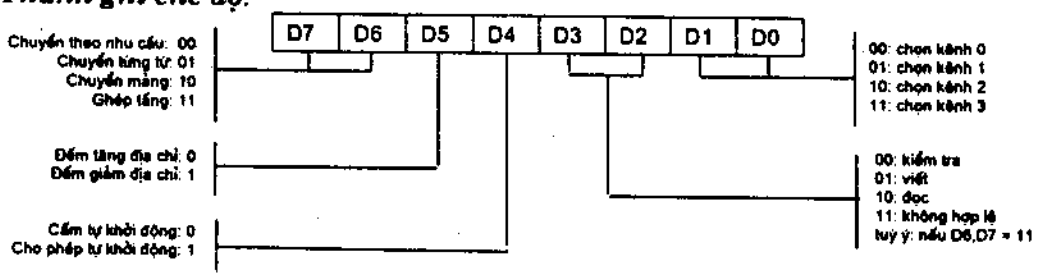
Thanh ghi yêu cầu DMA:



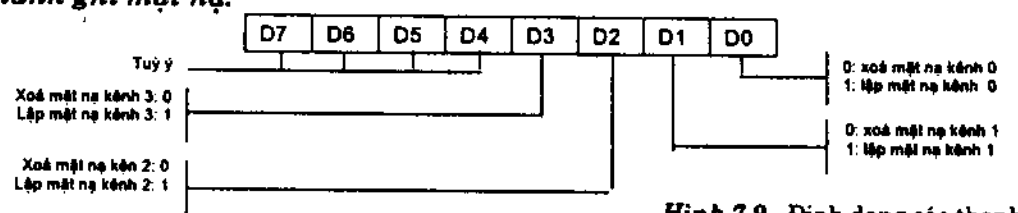
Thanh ghi lệnh:



Thanh ghi chế độ:



Thanh ghi mặt nạ:



Hình 7.9 Định dạng các thanh ghi.

Trong quá trình hoạt động, 8237 luôn cập nhật trạng thái của mình vào thanh ghi trạng thái để CPU có thể đọc ở bất cứ thời điểm nào. Thông tin trạng thái cho biết kênh DMA nào đã đạt đến số đếm kết thúc TC (terminal count) tức là đã chuyển xong mảng số liệu có độ dài quy định ở thanh ghi số đếm từ gốc, hoặc bị bắt buộc kết thúc chuyển do tác động của tín hiệu EOP đang chờ phục vụ ở bất cứ kênh nào nếu có.

8237 còn có thể nhận và thực hiện hai lệnh phần mềm từ CPU:

Lệnh xóa mạch flip-flop trong nhằm để đưa về trạng thái ban đầu trước khi đọc hoặc viết địa chỉ hoặc số đếm từ mới vào 8237.

Lệnh xóa toàn bộ (master clear) có chức năng như RESET lạnh: tất cả các thanh ghi lệnh, trạng thái, yêu cầu, mạch lật trong đều bị xóa, thanh ghi mặt nạ được lập và 8237 chuyển sang chu kì nghỉ.

Mã thâm nhập các thanh ghi địa chỉ và số từ của 8237 được trình bày trong bảng 7.1

***Bảng 7.1* Mã thâm nhập các thanh ghi địa chỉ và số từ của 8237**

Kênh	Thanh ghi	Đọc / viết	$\overline{CS}$		$\overline{IOR}$		$\overline{IOW}$	A3	A2	A1	A0	Mạch lật trong	Bus số liệu DB0- DB7
	Địa chỉ gốc và tức thời	viết	0	1	0	0	0	0	0			0	A0-A7
			0	1	0	0	0	0	0			1	A8-A15
	Địa chỉ tức thời	đọc	0	0	1	0	0	0	0			0	A0-A7
			0	0	1	0	0	0	0			1	A8-A15
0	Số từ gốc và tức thời	viết	0	1	0	0	0	0	1			0	W0-W7
			0	1	0	0	0	0	1			1	W8-W15
	Số từ tức thời	đọc	0	0	1	0	0	0	1			0	W0-W7
			0	0	1	0	0	0	1			1	W8-W15
	Địa chỉ gốc và tức thời	viết	0	1	0	0	0	1	0			0	A0-A7
			0	1	0	0	0	1	0			1	A8-A15
	Địa chỉ tức thời	đọc	0	0	1	0	0	1	0			0	A0-A7
			0	0	1	0	0	1	0			1	A8-A15
1	Số từ gốc và tức thời	viết	0	1	0	0	0	1	1			0	W0-W7
			0	1	0	0	0	1	1			1	W8-W15
	Số từ tức thời	đọc	0	0	1	0	0	1	1			0	W0-W7
			0	0	1	0	0	1	1			1	W8-W15
	Địa chỉ gốc và tức thời	viết	0	1	0	0	1	0	0			0	A0-A7
			0	1	0	0	1	0	0			1	A8-A15
	Địa chỉ tức thời	đọc	0	0	1	0	1	0	0			0	A0-A7
			0	0	1	0	1	0	0			1	A8-A15
2	Số từ gốc và tức thời	viết	0	1	0	0	1	0	1			0	W0-W7
			0	1	0	0	1	0	1			1	W8-W15
	Số từ tức thời	đọc	0	0	1	0	1	0	1			0	W0-W7
			0	0	1	0	1	0	1			1	W8-W15
	Địa chỉ gốc và tức thời	viết	0	1	0	0	1	1	0			0	A0-A7
			0	1	0	0	1	1	0			1	A8-A15
	Địa chỉ tức thời	đọc	0	0	1	0	1	1	0			0	A0-A7
			0	0	1	0	1	1	0			1	A8-A15
3	Số từ gốc và tức thời	viết	0	1	0	0	1	1	1			0	W0-W7
			0	1	0	0	1	1	1			1	W8-W15
	Số từ tức thời	đọc	0	0	1	0	1	1	1			0	W0-W7
			0	0	1	0	1	1	1			1	W8-W15

Bảng 7.2 Dưới đây liệt kê các lệnh của chip 8237.

Bảng 7.2 **Các lệnh của chip DMA-8237**

Tín hiệu						Lệnh
A3	A2	A1	A0	$\overline{IO\ R}$	$\overline{IO\ W}$	
1	0	0	0	0	1	Đọc thanh ghi trạng thái
1	0	0	0	1	1	Nạp thanh ghi lệnh
1	0	0	1	0	1	! Không hợp lệ
1	0	0	1	1	0	Nạp thanh ghi yêu cầu DMA
1	0	1	0	0	1	! Không hợp lệ
1	0	1	0	1	0	Ghi từng bit cho thanh ghi mặt nạ
1	0	1	1	0	1	! Không hợp lệ
1	0	1	1	1	0	Nạp thanh ghi chế độ
1	1	0	0	0	1	! Không hợp lệ
1	1	0	0	1	0	Xóa mạch lật trong
1	1	0	1	0	1	Đọc thanh ghi tạm thời
1	1	0	1	1	0	RESET DMA
1	1	1	0	0	1	! Không hợp lệ
1	1	1	0	1	1	! Không hợp lệ
1	1	1	1	0	1	! Không hợp lệ
1	1	1	1	1	1	Ghi toàn bộ thanh ghi mặt nạ

Trong PC/XT một chip DMAC chỉ có khả năng phục vụ các ngoại vi 8 bit (các thiết bị chỉ có các thanh ghi 8 bit). Trong máy PC/AT (16 bit) có thêm một chip DMAC nữa gọi là chủ (master) được nối với CPU. Các chân HRQ và HLDA của chip DMAC tớ được nối với kênh 0 của chip chủ để nhằm cho các kênh 0-3 có mức ưu tiên cao hơn 3 kênh của chip chủ. Các kênh 0-3 của chủ được quy ước là kênh DMA từ 4-7 của máy AT. Bốn kênh của chip tớ phục vụ cho ngoại vi 8 bit, trong khi đó 3 kênh từ 5 đến 7 không sử dụng được cấu hình cho các thiết bị 16 bit.

Bảng sau liệt kê địa chỉ vào/ra thanh ghi điều khiển và trạng thái của DMA-8237

DMA (1)	DMA (2)	Đọc/viết (R/W)	Thanh ghi
08h	D0h	R	trạng thái
08h	D0h	W	lệnh
09h	D2h	W	yêu cầu DMA
0Ah	D4h	W	che kênh
0Bh	D6h	R	chế độ DMA
0Fh	DEh	W	mặt nạ

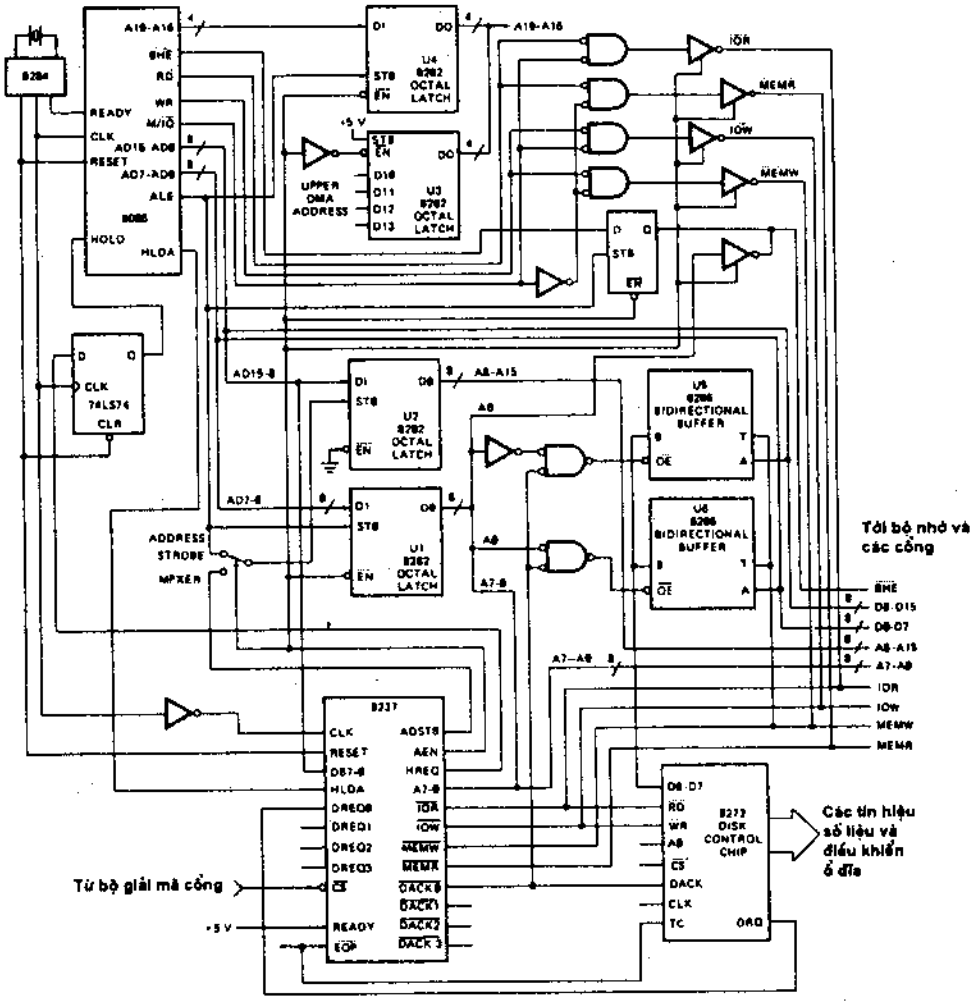
(1): chủ trong PC/XT, tớ trong AT; (2): chủ trong AT.

Các địa chỉ vào/ra của thanh ghi địa chỉ và đếm số từ được liệt kê dưới đây:

DMA (1)	DMA (2)	Thanh ghi
00h	C0h	địa chỉ kênh 0/4
01h	C1h	đếm kênh 0/4
02h	C2h	địa chỉ kênh 1/5
03h	C3h	đếm kênh 1/5
04h	C4h	địa chỉ kênh 2/6
05h	C5h	đếm kênh 2/6
06h	C6h	địa chỉ kênh 3/7
07h	C7h	đếm kênh 3/7

(1): chủ trong PC/XT, tớ trong AT, (2): chủ trong AT.

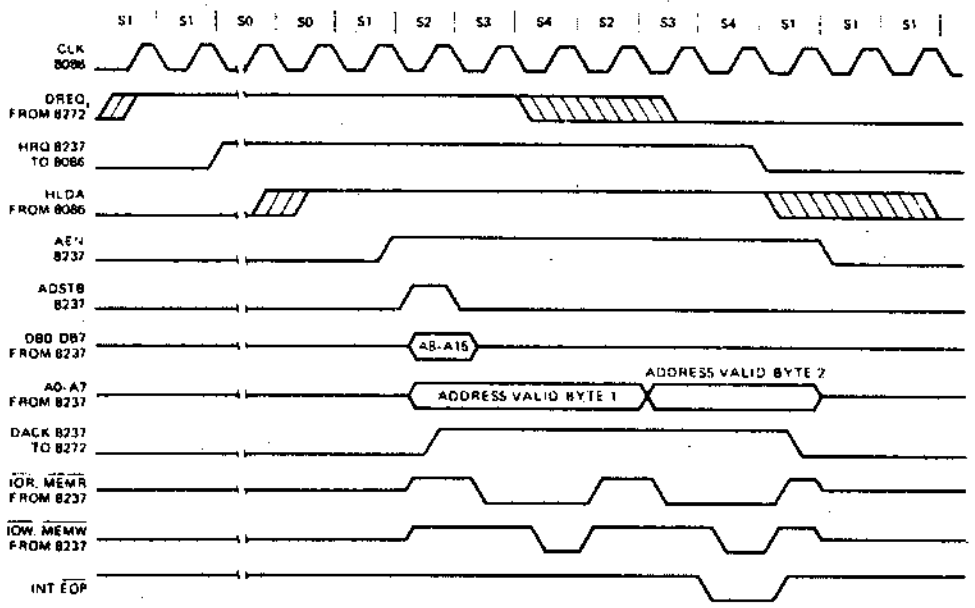
Hình 7.10 là một thí dụ sơ đồ chi tiết của mạch điều khiển 8237 và bộ điều khiển ổ đĩa mềm 8272 trong một máy vi tính 8086.



Hình 7.10 Mạch điều khiển DMA 8237 và bộ điều khiển ổ đĩa mềm 8272.

Do tốc độ xử lí lệnh của các vi xử lí thế hệ mới rất nhanh nên đến nay các máy vi tính không cần sử dụng phương pháp DMA cho nhiệm vụ này nữa nhưng đây vẫn là một thí dụ tốt để tìm hiểu cụ thể vấn đề. Qua đó ta có thể áp dụng để nắm và hiểu về nguyên tắc hoạt động của các mạch ghép nối ngoại vi được cắm trên các khe cắm mở rộng khi trao đổi số liệu với bộ nhớ bằng phương pháp DMA.

Hình 7.11 là giản đồ thời gian của việc truyền số liệu DMA trên các đường dây của hệ thống. Ta sẽ bắt đầu xem xét từ điểm trên hình 7.10 nơi mà 8237 làm dây AEN ở mức cao và bắt đầu nắm quyền điều khiển bus. Lúc đó nó sẽ gửi ra 8 bit địa chỉ thấp trên các chân từ A0 - A7 và 8 bit cao trên các chân từ DB0 - DB7. Tiếp đó nó sẽ phát ra xung chốt địa chỉ ADSTB ở mức cao để chốt các bit địa chỉ này vào vi mạch 8282 để giải phóng bus số liệu. Cùng thời gian đó, nó sẽ gửi tín hiệu DACK tới bộ điều khiển đĩa để báo rằng nó đã sẵn sàng cho việc truyền số liệu. Sau đó DMAC 8237 sẽ làm tích cực các tín hiệu điều khiển bus để thực hiện việc truyền số liệu. Với việc truyền từ bộ nhớ tới ổ đĩa là các tín hiệu $\overline{MEMR}$ và $\overline{IOW}$.



Hình 7.11 Giản đồ xung của quá trình DMA trong mạch hình 7.10.

Với việc truyền ngược lại là các tín hiệu $\overline{IOR}$ và $\overline{MEMW}$. Lưu ý rằng 8237 không phải xuất ra một địa chỉ vào/ra để cho phép bộ điều khiển ổ đĩa làm việc. Khi được chương trình trong chế độ DMA, bộ điều khiển ổ đĩa chỉ cần có các tín hiệu cho phép $\overline{IOR}$ và $\overline{IOW}$ cho việc truyền số liệu. Cũng vậy, 8237 không phải xuất ra một địa chỉ mới trên các dây A8 - A15 khi việc truyền số liệu tiếp theo được thực hiện trừ khi các bit này phải được thay đổi. Điều này tiết kiệm thời gian trong việc truyền nhiều byte. Khi số byte cần thiết đã được truyền xong, chip DMAC sẽ phát xung kết thúc quá trình $\overline{EOP}$ ở mức thấp tới 8086 và cũng đặt tín hiệu AEN xuống mức thấp. Điều này trả quyền điều khiển bus về cho CPU.

Như trong giáo trình cấu trúc máy tính đã trình bày, việc chương trình cho hoạt động DMA phải qua bước khởi động chế độ làm việc. Vì chip DMAC được nối tới các cổng vào/ra nên ta phải viết từ điều khiển khởi động cho nó như cho một thiết bị ngoại vi bất kì. Nếu có nhiều DMAC được ghép nối tầng thì cũng phải làm như vậy cho tất cả DMAC. Mỗi chip DMA 8237 có 4 lối vào yêu cầu DMA gọi là các kênh DMA. Mỗi kênh có thể được chương trình hoá để truyền một byte đơn lẻ cho mỗi yêu cầu, một khối các byte cho mỗi yêu cầu DMA hoặc giữ quá trình truyền các byte cho tới khi nó nhận được một tín hiệu đợi (wait signal) trên chân vào/ra $\overline{EOP}$.

Trong máy tính IBM PC, các đường tín hiệu phục vụ DMA được đưa ra các chân trên các khe cắm mở rộng (thí dụ như ở bus ISA như hình 5.1). Đó là 4 chân từ DRQ1 đến DRQ3 cho phép các card ngoại vi có thể yêu cầu sử dụng bus cho việc truyền số liệu DMA.

Khi chip DMAC nắm quyền điều khiển bus, nó sẽ báo cho ngoại vi biết bằng tín hiệu ghi nhận trên các chân từ DACK0 đến DACK3. Tín hiệu AEN trên các chân tín hiệu này được sử dụng để mở cổng cho địa chỉ DMA trên bus. Khi số byte truyền xong, đầu T/C nhảy lên mức cao để báo cho ngoại vi biết rằng đã kết thúc quá trình DMA. Vì máy AT dùng 2 chip 8259A ghép tầng nên có 15 đầu yêu cầu ngắt lối vào. 11 đầu được nối tới các chân IRQ trên bus mở rộng. Nó cũng dùng 2 chip DMAC 8237A để có 7 kênh DMA. Tín hiệu lối vào DREQ cho mỗi kênh và tín hiệu DACK được thiết kế trên bus. Còn tín hiệu REFRESH được dùng để chỉ thị hoạt động làm tươi bộ nhớ DRAM đang diễn ra v.v...Việc làm tươi bộ nhớ DRAM trong máy IBM PC ban đầu

được thực hiện bởi các chip định thời PIT-8253 và chip DMAC 8237. Như trên đã nói, bộ đếm 2 trong chip định thời 8253 được chương trình hoá sẽ sản ra các xung 15 μs một lần. Xung này được đưa tới kênh 0 của chip DMAC 8237A. Chip này được chương trình hoá để tiến hành một chu kì đọc giả (dummy) bộ nhớ DRAM và viết vào một cảng không tồn tại nào đó. Khi 8237A nhận được xung này, nó sẽ gửi tín hiệu ghi nhận yêu cầu DMA tới vi xử lí. 8086 sẽ đáp lại bằng tín hiệu HLDA cho phép DMAC nắm quyền điều khiển bus, gửi ra một địa chỉ nhớ, một tín hiệu đọc bộ nhớ và một tín hiệu ghi nhận DMA là DACK0. 8 bit thấp của địa chỉ nhớ này sẽ đi tới lối vào của tất cả các ô nhớ DRAM. Tín hiệu DACK0 từ DMAC sẽ kích hoạt chân $\overline{RAS}$ xuống mức thấp. Sau mỗi hoạt động DMA, thanh ghi địa chỉ hiện tại trong 8237A sẽ tự động tăng lên 1 (hoặc giảm đi 1) tuỳ thuộc vào chương trình. Hoạt động DMA tiếp theo sẽ làm tươi hàng tiếp theo trong bộ nhớ DRAM. Thí dụ, nếu 8237A được lập trình để truyền 64 kbyte theo phương pháp DMA từ địa chỉ 0 thì chuỗi địa chỉ gửi ra sẽ làm tươi tất cả 256 hàng một. Mỗi hàng trong mỗi băng nhớ sẽ được làm tươi 15 μs một lần. Với tần số đồng hồ 4,77 MHz trong máy IBM PC, như vậy một chu kì làm tươi bằng DMA mất chừng 820 ns tức là chừng 5% thời gian xử lí. Trong các máy AT, chip DMAC 8237A không được sử dụng nữa mà thay bằng một chip đặc biệt chuyên dụng 82C08, cho phép thời gian này còn được giảm hơn nữa.

Chương trình sau là một thí dụ về việc truyền số liệu từ khối nhớ có địa chỉ từ 10000h đến 13FFFh tới khối nhớ có dải địa chỉ từ 14000h tới 17FFFh.

Chương trình 7.2

```
LATCHB          EQU     10H             ; chốt (B)
CLEAR_FL        EQU     7CH             ; địa chỉ F/L của mạch lật
CH0_ADD         EQU     70H             ; địa chỉ kênh 0
CH1_ADD         EQU     72H             ; địa chỉ kênh 1
CH1_CNT         EQU     73H             ; bộ đếm kênh 1
MODE            EQU     7BH             ; địa chỉ thanh ghi chế độ
CMMD            EQU     78H             ; địa chỉ thanh ghi lệnh
MASKS           EQU     7FH             ; địa chỉ thanh ghi mặt nạ
REQ             EQU     79H             ; địa chỉ thanh ghi yêu cầu DMA
STATUS          EQU     78H             ; địa chỉ thanh ghi trạng thái

; thanh ghi vào     ES:SI           địa chỉ khối dữ liệu cần truyền
;                   ES:DI           địa chỉ nơi đến
;                   CX              số byte cần chuyển
; thanh ghi ra      không có
;
TRANSFER        PROG    FAR
                PUSH    AX
; lập trình cho LatchB;
                MOV     AX,ES
                MOV     AL,AH
                SHR     AL,1
                SHR     AL,1
                SHR     AL,1
                SHR     AL,1
```

```
                    OUT     LatchB, AL
; xoá mạch lật;
                    OUT     CLEAR_FL, AL
; đặt địa chỉ nguồn vào kênh 0;
                    MOV     AX,SI
                    OUT     CH0_ADD,AL
                    MOV     AL,AH
                    OUT     CH0_ADD, AL
; đặt địa chỉ đích vào kênh 1;
                    MOV     AX,DI
                    OUT     CH1_ADD, AL
                    MOV     AL,AH
                    OUT     CH1_ADD,AL
; chỉ định số byte cần chuyển;
                    MOV     AX,CX
                    DEC     AX
                    OUT     CH1_CNT, AL
                    MOV     AL,AH
                    OUT     CH1_CNT, AL
; đặt chế độ;
                    MOV     AL,88H
                    OUT     MODE, AL
                    MOV     AL,85H
                    OUT     MODE, AL
; cho phép việc truyền;
                    MOV     AL,1
                    OUT     CMMD, AL
; bỏ mặt nạ che kênh 1:
                    MOV     AL,0EH
                    OUT     <ASKS, AL
; yêu cầu DMA;
                    MOV     AL, 04
                    OUT     REQ, AL
; AGAIN             IN      AL, STATUS]
                    TEST    AL, 1
                    JZ      AGAIN           ; đợi kết thúc DMA
; kết thúc thủ tục truyền;
                    POP     AX
                    RET
TRANSFER            ENDP
```

7.4. THÂM NHẬP RAM CMOS

Tất cả PC với cấu hình 80286 và cao hơn đều có ít nhất 64 byte nhớ đặc biệt trong RAM CMOS được nuôi bằng nguồn ắc-quy để lưu trữ các thông tin về cấu hình và thời gian thực kể cả khi ngắt điện máy tính. Các byte nhớ này không được định bản đồ nhớ

trong không gian địa chỉ của CPU mà chúng được thâm nhập qua 2 cổng vaò/ra có địa chỉ ở 70h và 71h. Một chương trình sẽ viết địa chỉ cấu hình (từ 00h đến 3Fh) tới cổng 70h rồi viết một giá trị byte mới hoặc đọc giá trị byte hiện tại ở địa chỉ 71h. Nội dung của các byte nhớ đó và địa chỉ đã được nêu tỉ mỉ trong giáo trình cấu trúc máy vi tính. Việc ghi và đọc nội dung của RAM CMOS được thực hiện qua chương trình SETUP khi khởi động máy tính hoặc chạy một chương trình SETUP.EXE thương phẩm sẵn có. Người lập trình cũng có thể tự mình viết các chương trình thâm nhập bằng các lệnh viết đọc 2 cổng 70h và 71h như kể trên. Dưới đây là chương trình nguồn của tác giả Michael Tisher viết chạy trong TURBO PASCAL cung cấp các hàm cho phép ghi và đọc các byte của RAM CMOS cho đồng hồ thời gian thực

Chương trình 7.3

```
{ Chuc nang : cung cap hai ham cho phep ghi va doc cac byte  *)
{*                    cua RAM CMOS dong ho thoi gian thuc     *)
{*-------------------------------------------------------------*)

Progam RTC;
 Uses Crt;
 type Jours = array[1..7] of string[10] ;          { Bang ten thu }
const
      RTCAdrPort = $70 ;                           { Thanh ghi dia chi cua RTC }
      RTCDtaPort = $71;                            { Thanh ghi du lieu cua RTC }
      SECONDES          = 0;                       { Dia chi mot so o nho cua RTC }
      MINUTES           = 2;
      HEURES            = 4;
      JOURSEMAINE       = 6;
      JOUR              = 7;
      MOIS              = 8;
      ANNEE             = 9;
      ETATA             = 10;
      ETATB             = 11;
      ETATC             = 12;
      ETATD             = 13;
      DIAGNOSTIC        = 14;
      SIECLE            = 50;
      JoursSemaine : Jours = ( 'Chu Nhat', 'Thu Hai', 'Thu Ba', 'Thu Tu', 'Thu Nam','Thu Sau', 'Thu Bay' );
{*************************************************************}
{ RCTRead: Đọc nội dung một ô nhớ cũ }
{ Vao       : Địa chỉ của ô nhớ cần đọc  }
{* Ra        : Nội dung ô nhớ             }
{* Chu thich : Nếu địa chỉ không nằm trong giới hạn cho phép (0 đến 63) thì trả về -1 }
{*************************************************************}
Function RTCRead(Adresse : integer) : integer;
        Begin
        if (Adresse < 0)  or  (Adresse > 63)        { Dia chi hop le ? }
        then RTCRead := -1  else                      {Khong ! }
```

```
                begin
                Port[RTCAdrPort]:= Adresse;          { Truyen dia chi toi RTC }
                RTCRead:= Port[RTCDtaPort]           { Doc noi dung cua no }
                end
        End;
{*****************************************************************}
{*RTCDT    : Đọc nội dung ô nhớ chứa ngày và giờ của RTC rồi chuyển kết quả thành số nhị phân nếu kết
quả là số BCD }
{* Vao     :    Địa chỉ ô nhớ cần đọc              *}
{* Ra      :    Nội dung ô nhớ dưới dạng số nhị phân  *}
{* Nếu địa chỉ nằm ngoài giơi hạn cho phép (0 đến 63) thì trả về -1 *}
{*****************************************************************}
Function RTCDT(Adresse : integer) : integer;
var Valeur : integer;              { Bien chua gia tri doc duoc }
        Begin
        if  (RTCRead(ETATB) and 2 = 0)  {So BCD hay nhi phan ? }
        then RTCDT := RTCRead(Adresse)   {So nhi phan }
        else                        {So BCD}
                begin
                Valeur   := RTCRead(Adresse);   { Doc noi dung o nho }
                RTCDT := (Valeur shr 4 ) * 10 + valeur and 15;          {Chuyen so BCD thanh nhi phan }
                end
        End;
{*****************************************************************}
{*  RTCWrite: Ghi một byte nhớ của RTC          *}
{*  Vao       : Xem dưới                        *}
{*  Ra        : Không                           *}
{*  Chu thich : Địa chỉ phải nằm trong giới hạn từ 0 đến 63   *}
{*****************************************************************}
Procedure RTCWrite(Adresse  :  integer;  { Dia chi o nho can ghi }  Contenu :  byte);    { Gia tri moi cua no }
        Begin
         Port[RTCAdrPort]  :=Adresse;        { Truyền địa chỉ tới  RTC }
         Port[RTCDtaPort]  :=Contenu                  {Ghi }
        End;
{*****************************************************************}
{*      Chuong trinh chinh                  *}
{*****************************************************************}
BEGIN
        ClrScr;                              { Xoá màn hình }
        writeln ('RTC  (c) 1987 by Michael Tisher ',#13#10);
        writeln ('Cac thong tin doc tu RAM CMOS cua RTC');
        writeln ('=============================='#13#10);
        if RTCRead(Diagnostic)and 128 = 0 then    { Pin con tot ? }
                Begin                          { Pin con tot }
          writeln('-Dong ho dang o che do',(RTCRead (ETATB) and 2)*6+12,'gio');
          writeln('-Bay gio la  :  ',  RTCDT(HEURES),  ' : ',RTCDT(MINUTES) :2,':', RTCDT (SECONDES): 2);
```

```
        write('-Ngay : ', JoursSemaine[ RTCDT (JOURSEMAINE)]);
        writeln('  ',RTCDT(JOUR),'  ',RTCDT(MOIS),'  ',TCDT(SIECLE,TCDT(ANNEE));
              End else                              { Het pin }
        write('          Chu y ! Het pin cho RTC ');
END.
```

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

7.1 Ta biết rằng người thiết kế hệ thống PC luôn cẩn thận trong mọi trường hợp. Do đó thường thời gian làm tươi bộ nhớ thường được thiết kế nhiều hơn mức cần thiết . Tỉ số giữa thời gian này trên toàn bộ thời gian chạy gọi là *refresh overhead* thường tới 10% hay hơn nữa. Về nguyên tắc, có thể giảm tốc độ làm tươi trên PC nhằm làm tăng năng suất truyền dữ liệu (data throughput) đến 5%. Khi số liệu bị mất do tốc độ làm tươi quá chậm sẽ xuất hiện sai số chẵn lẻ khi đọc bộ nhớ và lúc đó PC sẽ hiện một thông báo lên màn hình. Hãy chạy thử một chương trình thay đổi tốc độ làm tươi bộ nhớ của PC và kiểm tra tác động của nó đến chỉ tiêu của PC khi sử dụng các chương trình *benchmark*. Hãy nhập và cho chạy chương trình sau cho phép đặt lại chế độ làm tươi bộ nhớ của PC. Giá trị đếm ban đầu trong bộ đếm 2 có thể thay đổi được từ 2 đến 255.

⊖ Chương trình 7.4

```
main (argc, argv) /* chương trình gọi refresh.c  */
int  argc;
char  *argv[];
{  int count;
   if (argc > 2) {
          printf("\n\nArgument is missing !\nFormat:   refresh <count>\n\n");
          exit (2);
}
count = atoi (argv[1]);
   if (count <0x02 || count >0xff)  {
          printf("\n\nInvalid value %d !!\n\n", count);
          exit (1);
}
outportb(0x43,0x56);                    /* 0101 0110 */
outportb(0x41,count);
          printf("\n\nNew refresh value %d set.\n\n", count);
          exit (0);
}
```

7.2 Viết lại chương trình trên cho phép đặt tốc độ làm tươi chậm hơn nữa. Giá trị nạp có thể cho lên đến trên 1000. Xác định với tốc độ làm tươi chậm xuống bao nhiêu thì xảy ra lỗi chẵn lẻ ?

7.3 Tại sao các chế độ hoạt động của PIT-8253 trong PC lại ở mode 2 hoặc 3 ? Hãy thử thay đổi chế độ của chúng sang các mode còn lại, quan sát hiện tượng phản ứng của PC và giải thích. Chú ý rằng sau khi được khởi động nóng (nhấn Ctrl-Alt-Del), BIOS sẽ lại khởi tạo cho PIT hoạt động ở các mode bình thường.

7.4 Viết và cho chạy chương trình phát 8 âm thanh ở bát độ thứ năm: Đồ, Rê, Mi, Fa, Sol, La, Si, Đố ra loa PC bằng việc điều khiển bộ đếm 2 trên PIT. Tần số các âm tính ra Hz cho như sau:

Đồ 261,63 - Đô# 277,18 - Re 293,66 - Re# 311,13

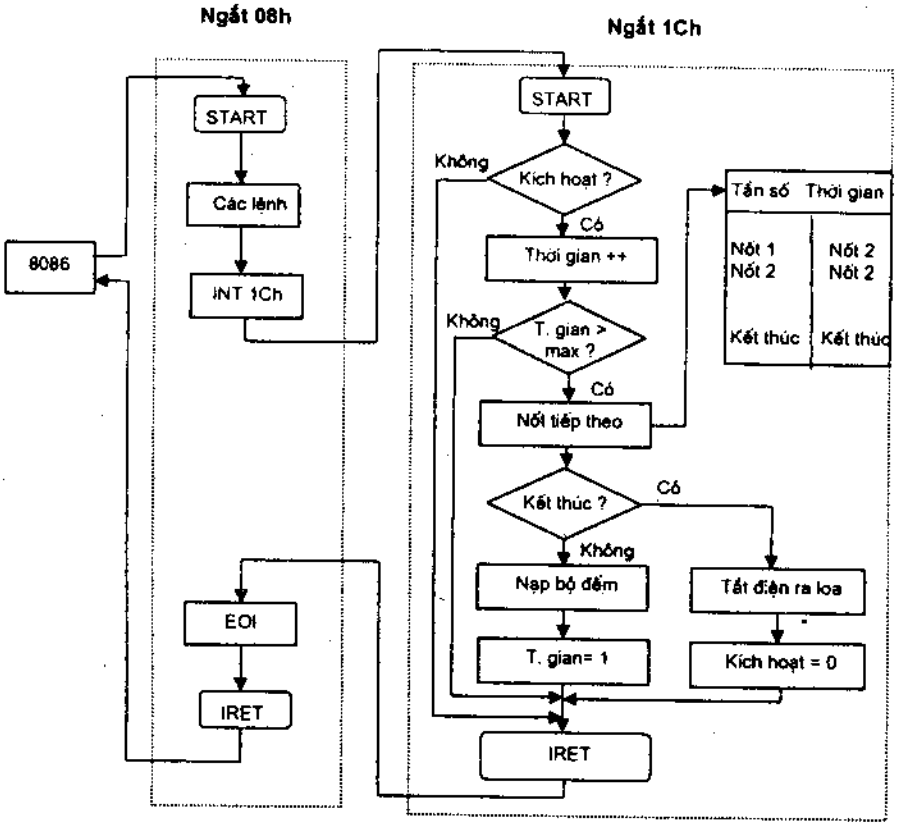
Mi 329,63 - Fa 349,23 - Fa# 369,99 - Son 392,00

Son# 415,30 - La 440,00 - La# 466,16 - Si 498,88 - Đố 523,25

Không được dùng các lệnh Sound và Nosound, mỗi âm phát trong 2 giây.

7.5 Viết chương trình cho phép bộ đếm 2 trong PIT và ngắt 08h cùng hoạt động để tạo ra chuỗi âm thanh nền độc lập với CPU. Tần số tính ra Hz trên 8 bát độ (octave) của các nốt nhạc được cho trong bảng dưới đây và lưu đồ gợi ý của chương trình như sau.

Âm	Bát độ 1	Bát độ 2	Bát độ 3	Bát độ 4	Bát độ 5	Bát độ 6	Bát độ 7	Bát độ 8
C	16,4	32,7	65,4	130,8	261,6	523,3	1046,5	2093,0
C#	17,3	34,7	69,3	138,6	272,2	554,4	1108,8	2217,5
D	18,4	36,7	73,4	146,8	293,7	587,3	1174,7	2349,3
D#	19,5	38,9	77,8	155,6	311,1	622,3	1244,5	2489
E	20,6	41,2	82,4	164,8	329,6	659,3	1328,5	2637
F	21,8	43,6	87,3	174,6	349,2	698,5	1396,9	2793,8
F#	23,1	46,3	92,5	185	370	740	1480	2960
G	24,5	49,0	98,0	196	392	784	1568	3136
G#	26	51,9	103,8	207,7	415,3	830,6	1661,2	3322,4
A	27,5	55,0	110	220	440	880	1760	3520
A#	29,1	58,3	116,5	233,1	466,2	923,3	1864,7	3729,3
B	30,9	61,7	123,5	247	493,9	987,8	1975,5	3951,1



7.6 Ta có thể phát một âm ra loa với việc luân phiên đóng và mở 2 công tắc điện tử trong hệ thống phát âm ra loa PC mà không cần chú ý tới bộ đếm trong PIT . Việc này được thực hiện bằng một vòng lặp đặt và xoá các bit B0 và B1 của cổng B chip PPI-8255. Tốc độ chạy chương trình càng nhanh, âm phát ra có tần số càng cao. Tuy nhiên phương pháp này có những nhược điểm: CPU luôn phải hoạt động cho mục đích kể trên, tần số âm phụ thuộc vào tốc độ của CPU, nhiều can nhiễu do đóng mở công tắc điện tử. Hãy sử dụng phương pháp này phát ra các âm gần với âm La = 440 Hz và âm Đồ 261 Hz. Thời gian phát luân phiên là 1 giây phát, 1 giây ngừng.

7.7 Trong một hệ thống có bộ nhớ RAM được nuôi bằng pin dự phòng dùng để lưu lại dữ liệu khi mất điện, ngăn xếp thường được đặt trong bộ nhớ RAM này. Điều này làm cho việc lưu lại các thanh ghi và dữ liệu khẩn cấp được dễ dàng. Giả sử bộ nhớ RAM được nuôi bằng pin dự phòng nằm trong dải địa chỉ từ 08000h đến 08FFFh. Hãy viết một thủ tục phục vụ ngắt khi mất nguồn để:

- Lập một mạch lật được nối với bit 0 của cổng 28H để báo nguồn nuôi đã bị cắt.
- Lưu tất cả các thanh ghi vào ngăn xếp.
- Lưu giá trị của con trỏ ngăn xếp trỏ tới biến cuối cùng vào địa chỉ 8000h.
- Lưu nội dung các ô nhớ từ địa chỉ 00100h tới địa chỉ 003FFH vào sau giá trị của con trỏ ngăn xếp ở đầu bộ nhớ RAM nuôi bằng pin dự phòng (một câu lệnh thao tác chuỗi có lẽ sẽ hữu dụng trong trường hợp này)
- Dừng.

Khi điện được cấp trở lại, thủ tục khởi động sẽ kiểm tra mạch lật. Nếu mạch lật được lập, thủ tục khởi động sẽ sao dữ liệu được lưu trở lại vị trí hoạt động của chúng, khởi tạo thanh ghi đoạn ngăn xếp và sau đó lấy giá trị của thanh ghi SP ra khỏi địa chỉ 8000h. Từ giá trị này, nó có thể khôi phục lại các thanh ghi đã cất vào ngăn xếp và quay lại thực hiện công việc kể từ thời điểm bị mất điện. Đó là quá trình khởi động nóng. Nếu không muốn khởi động nóng, ta có thể reset mạch lật bằng một công tắc RESET bên ngoài để hệ thống bắt đầu lại từ đầu, hay khởi động lạnh.

7.8 Tại sao lối vào INTR của 8086 lại tự động bị cấm khi 8086 bị RESET? Lối vào INTR của 8086 được cho phép như thế nào để đáp ứng các ngắt? Câu lệnh nào được dùng để cấm lối vào INTR?

7.9 Tại sao trong đáp ứng ngắt INTR lại phải dành ra một phần để tự động cấm lối vào INTR?

7.10 Lối vào INTR được tự động cho phép lại như thế nào ở cuối thủ tục phục vụ ngắt INTR?

7.11 Mô tả phản ứng của 8086 nếu nó nhận được tín hiệu ngắt NMI trong khi đang thực hiện một phép chia gây ra ngắt chia cho zero.

7.12 Các lối ra số liệu của một bộ ADC 8 bit được nối với các bit D0-D7 của cổng FFF9h, và tín hiệu EOC từ ADC được đưa tới lối vào NMI của 8086. Hãy viết một chương trình chính và một thủ tục phục vụ ngắt đơn giản cho phép đọc từng byte số liệu từ ADC. Nếu bit MSB của số liệu bằng 0, biểu hiện số liệu vẫn nằm trong thang, hãy cộng byte ấy với tổng chạy (running total) được lưu trong 2 ô nhớ liên tiếp. Nếu bit MSB của số liệu bằng 1, tức số liệu tràn thang, số liệu vào sẽ bị bỏ qua. Sau khi 100 mẫu đã được lấy tổng, hãy chia cho 100 để lấy giá trị trung bình, lưu giá trị trung bình này vào một ô nhớ khác rồi xoá tổng trở về 0.

7.13 Viết thuật toán và chương trình cho một thủ tục phục vụ ngắt có nhiệm vụ bật một đèn LED nối với bit D0 của cổng FFFAh trong 25 giây và tắt nó đi trong 25 giây. Thủ tục này cũng bật một đèn LED khác nối với bit D1 của cổng FFFAh trong 1 phút và tắt nó đi trong 1 phút. Giả thiết rằng có một tín hiệu ngắt 1 Hz được nối với chân NMI của 8086 và mức cao xuất ra cổng sẽ bật sáng đèn

7.14 Mô tả chuỗi các thao tác của một vi mạch 8259A và của 8086 nối với nhau như trên hình 7.3 khi 8259A nhận được tín hiệu ngắt tại lối vào IR2 của nó. Giả sử chỉ có duy nhất chân IR2 của 8259A là không bị che và lối vào INTR của 8086 đã được cho phép lại bằng lệnh STI.

7.15 Mô tả chức năng của các đường CAS0, CAS1 và CAS2 trong một hệ thống nối tầng 8259A.

7.16 Mô tả phản ứng của một vi mạch 8259A nếu nó đồng thời nhận được tín hiệu ngắt trên các chân IR3 và IR5 của nó. Giả sử chế độ đặc quyền cố định cho các lối vào IR. 8259A sẽ phản ứng như thế nào khi nhận được tín hiệu ngắt IR3 trong khi đang phục vụ ngắt IR5?

7.17 Tại sao cần phải gửi một lệnh EOI cho 8259A trong thủ tục phục vụ ngắt?

7.18 Trình bày chuỗi các từ lệnh và đoạn lệnh dùng để khởi tạo một vi mạch 8259A có địa chỉ cơ sở là FF10h theo các điều kiện sau: trigger theo sườn xung; chỉ có một vi mạch 8259A; hệ thống 8086; kiểu ngắt 40 cho IR0; EOI bình thường; không bộ đệm (nonbuffered mode); không lồng nhau hoàn toàn (not special fully nested mode); chỉ cho phép IR1 và IR3.

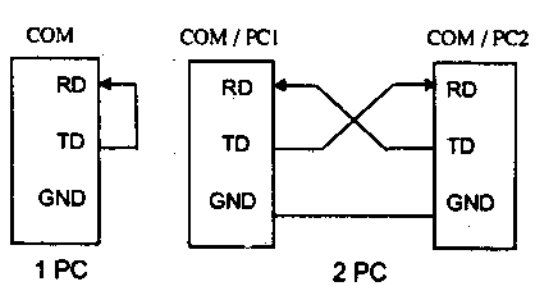
7.19 Ưu điểm chính của việc gọi các thủ tục của BIOS bằng các ngắt mềm so với việc gọi chúng bằng các địa chỉ tuyệt đối?

7.20 PIC-8259A được nối với 8086 theo sơ đồ hình 7.3 có địa chỉ F0h và F1h, làm việc ở chế độ đơn lẻ trong hệ có đệm bus, chế độ ưu tiên cố định và với EOI thường, IR được trigơ theo mức, tín hiệu IR0 được gán, số ngắt là 50h. Đoạn chương trình khởi động cho nó là:

```
MOV     AL, 1Bh         ; ICW1
OUT     0F0h, AL        ; ICW1 đưa ra cổng F0h
MOV     AL,50h          ; ICW2
OUT     0F1h, AL        ; ICW2 đưa ra cổng F1h
MOV     AL, 0Dh         ; ICW4
OUT     0F1h, AL        ; ICW4 đưa ra cổng F1h
```

7.21 Hãy giải thích ý nghĩa các dữ liệu được xuất ra các cổng trên cơ sở định dạng các thanh ghi ICW.

7.22 Nối bắt chéo các đầu phát TD và đầu thu RD của 2 cổng COM (nếu có 2 máy tính) hoặc nối đầu TD với RD của một cổng COM (nếu chỉ có một máy tính) như hình bên. Viết và cho chạy thực hành chương trình điều khiển cổng RS-232 bằng phương pháp ngắt sau:



⊖ Chương trình 7.5

```
# include <dos.h>
# include <conio.h>
```

```
# include <stdio.h>
# include <bios.h>
# include <serial.h>

void  interrupt  rs_interrupt (void);
void  set_upserial (void);
void  send_character (int ch);
int    get_character (int ch);
int    get_buffer (void);
void   set_vectors (void);
void   reset_vectors (void);
void   enable_interrupt (void);
void   disable_interrupt (void);

void  interrupt (*oldvect ) ();

char   buffer [RSBUFSIZE];
unsigned  int    starbuf = 0, endbuf = 0;
int   main (void)
{
int   ch, done  = FALSE;
        setup_serial ();
        set_vectors ();                    /* đặt véctơ ngắt và lưu trữ cái cũ */
        enable_interrupt ();
        printf (" nhấn ESC để kết thúc chương trình \ n");
        do
        {
                if (kbhit ())
                {
                        ch = getcha ();
                        if (ch=ESC) braeak;
                        send_character (ch);
                }                          /* làm rỗng bộ đệm của RS-232  */
                do
                {       if ((ch = get_buffer ())  !=  -1 ) putch (ch);
                } while (ch != -1);
        } while (! done);
        disable_interrupts ();
        reset_vectors ();
        return () ;
}
void      interrupts   rs_interrupt (void);
{
        disable ();
        if  ((inportb (IIR)  & RX_MASK ) == RX_TD )
        {
```

```
                buffer (endbuf)  = inportb (RXR);
                endbuf++
                if (endbuf==RSBUFSIZE) endbuf = 0;
        }       /* đặt kết thúc cờ ngắt */
        outportb(ICR, EOI);
        enable();
}
void    setup_serial (void)
{
        int     RS232_setting;
        RS232_setting = BAUD1200 | STOPBIT1 | NOPARTY | DATABIT7;
        bioscom (0, RS232_setting, COM1);
}
void    send_character (int ch)
{
char    status;
        do
        {
                status = inportb(LSR) & 0x40;
        } while  (status != 0x40);   /* lặp lại cho tới khi bộ đệm Tx rỗng, bit 6 được đặt */
        outportb(TXDATA, (char) ch);
        }
int     get_character (void)
{
int     status;
        do
        {
                status = inportb(LSR) & 0x01;
        } while (status != 0x01);    /* lặp lại cho tới khi bit 1 được đặt */
        return ( (int) inportb(TXDATA));
}
int     get_buffer (void)
{
int     ch;
        if (startbuf == endbuf)  return (-1);
        ch = (int) buffer (startbuf);
        startbuf++
        if (startbuf == RSBUFSIZE ) startbuf =0;
        return (ch);
}
void    set_vectors(void)
{
        oldvect = setvect(0x0C);
        setvect(0x0c, rs_interrupt);
}       /* bỏ các véctơ ngắt trước khi thoát chương trình */
void    reset_vectors(void)
```

```
{
        setvect(0x0C, oldvect);
}
void    disable_interrupts(void)
{
int     ch;
        disable();
        ch = inportb(IMR) | -IRQ4;          /* cấm ngắt IRQ4 */
        outportb(IMR, ch);
        outportb(IER, 0);
        enable();
}
void    enable_interrupts(void)
{
int     ch;
        disable();          /*khởi động cổng RS232 */
        ch = inportb(MCR) | MC_INT;
        outportb(MCR, ch);          /* cho phép các ngắt tới IRQ4 */
        outportb(IER, 0x01);
        ch = inportb(IMR) & IRQ4;
        outportb(IMR, ch);
        enable ();
}
```

7.23 Viết và chạy chương trình xác định các ngắt được phép (enable) và bị cấm (disable) trong máy vi tính đang dùng bằng cách đọc thanh ghi che ngắt IMR như sau:

Θ Chương trình 7.6

```
#include <stdio.h>
#include <dos.h>
enum ints {IRQ0=0x01, IRQ1=0x02, IRQ2=0x04, IRQ3=0x08,
           IRQ4=0x10, IRQ5=0x20, IRQ6=0x40, IRQ7=0x80};
# define  IMR  0x21;
int  main (void)
{
int  ch;
        ch = inportb (IMR);
        if (ch & IRQ0 ) printf (" IRQ0 bị cấm  \ n");
        else  printf ("IRQ0 được_phép \ n");
                if (ch & IRQ1 ) printf (" IRQ1 bị cấm  \ n");
                else  printf ("IRQ1 được_phép \ n");
                  . . . . . .
        return (0)
}
```

Thay đổi chương trình trên sử dụng một vòng for { } để kiểm tra các bit trạng thái ngắt.

```
#include <stdio.h>
#include <dos.h>
# define   IMR   0x21;
int  main (void)
{
int  ch, bitmask ;
        for  (bitmask=0x01; bitmask<=0x80; bitmask<<=1)
        {
          ch = inportb (IMR) ;
                if ( ch & bitmask )
                . . . . . .
        }
  return
}
```

7.24 Chứng minh rằng chương trình trên là một hệ thống đa nhiệm thực sự khi xen vào một vòng trễ trong vòng chính (main loop) như chỉ ra dưới đây. Chương trình sẽ có khả năng đệm tất cả các kí tự nhận được và hiện chúng tới màn hình khi thời gian trễ sleep kết thúc.

Θ Chương trình 7.7

```
do
{
        sleep (10) ;        /* nhảy tới sleep trong 10 giây, hệ thời gian thực  */
                            /* sẽ đệm tất cả các kí tự   */
        if  (kbhit () )
        {
        ch = getcha () ;
        if ( ch = ESC )  break ;
        send_character (ch);
        }        /*  làm rỗng bộ đệm RS-232          */
        do
        {
                if ( ( ch = get_buffer () ) != -1 ) putch (ch) ;
        } while (ch != -1) ;
}       while  (!done) ;
```

7.25 Thay đổi chương trình trên nhằm cho các kí tự được phát sẽ hiện trên nửa phần trên màn hình và các kí tự được thu sẽ hiện trên nửa dưới.

7.26 Viết chương trình truyền 1000 từ số liệu 16 bit từ một vùng nhớ này sang vùng nhớ khác bằng phương pháp DMA.

7.27 Tìm hiểu và nhập chương trình sau cho phép hiện thời gian của đồng hồ hệ thống bên trong PC. Nếu nhấn một phím trên bàn phím, tốc độ hiện sẽ tăng nhanh lên 8 lần; nhấn lần nữa trở lại bình thường.

⊖ Chương trình 7.8

```
main ()
{
        displaytime ();
/* tăng tốc lên 8 lần */
outportb(0x43,0x34);                    /* từ điều khiển 0011 0100 */
outportb(0x40,0x00);                    /*nạp số đếm byte thấp */
outportb(0x40,0x20);                    /*nạp byte cao*/
displaytime();
/* tốc độ hiện bình thường*/
outportb(0x43,0x34);                    /* từ điều khiển 0011 0100 */
outportb(0x40,0x00);                    /*nạp số đếm byte thấp */
outportb(0x40,0x00);                    /*nạp byte cao*/
displaytime();
exit();
}
void displaytime(void)
{ struct dostime_t time;
        for (;;) {
                dosgettime (&time);                     /* đọc thời gian */
                /* hiện thời gian theo định dạng giờ: phút: giây  phần trăm giây */
                _settextposition (1,1), printf("%i:%i:%i,%i                ",
                        time.hour,time.minnute,time.second,time.hsecond);
                if (kbhit()) { getch ();
                                return;
                }
        }
}
```

7.28 Viết chương trình đọc nội dung đồng hồ thời gian thực (ngày, tháng, năm) của RAM CMOS qua việc viết địa chỉ vào thanh ghi địa chỉ 70h và đọc nội dung ở thanh ghi dữ liệu 71h rồi so sánh với kết quả nhận được bằng lệnh *getdate (year,month,day)*

Gợi ý:

- Phạm vi đọc trong 64 byte đầu tiên (xem giáo trình cấu trúc máy tính).
- Các số được ghi dưới dạng mã BCD và được chuyển đổi ra mã nhị phân theo công thức sau: GTNP = (BCD shr 4)×10 + BCD and 15
- trong đó GTNP là giá trị theo mã nhị phân; BCD là giá trị theo mã BCD

CHƯƠNG 8. CÁC CỔNG SONG SONG VÀ NỐI TIẾP - GHÉP NỐI MÁY IN VÀ MODEM

8.1. CÁC CỔNG SONG SONG VÀ NỐI TIẾP TRONG MÁY VI TÍNH

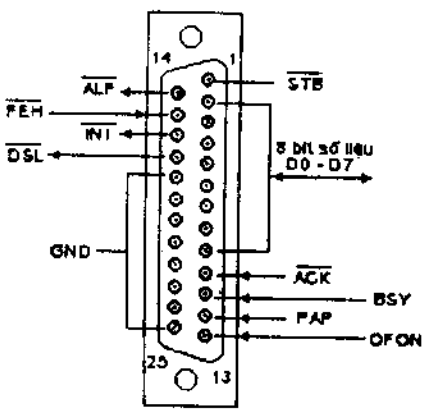
Máy tính IBM/PC được trang bị 2 loại cổng giao tiếp với thế giới bên ngoài là cổng song song và nối tiếp. Các cổng này là các ghép nối vật lí cho phép nối hệ thống vi tính với các thiết bị ngoại vi như máy in, modem v.v...

Cổng được gọi là song song vì nó cho phép truyền số liệu đồng thời các bit của một từ dữ liệu trong một nhịp truyền. Như vậy nếu từ dữ liệu dài 8 bit thì ngoài dây chung (dây đất) ra, cổng sẽ phải có 8 dây dữ liệu cho 8 bit thông tin. Trong khi đó, cổng nối tiếp chỉ cho phép truyền lần lượt mỗi nhịp một bit trong từ dữ liệu và như vậy cổng chỉ cần 1 dây truyền dữ liệu nhưng lại phải cần ít nhất là 8 nhịp truyền như thí dụ vừa nêu.

8.2. CỔNG SONG SONG GHÉP NỐI VỚI MÁY IN

8.2.1. Cổng song song LPT

Cổng song song LPT trong máy vi tính có đầu nối loại D-25 theo chuẩn Centronics (hình 8.1). Nó cho phép dùng cho cả phát và nhận số liệu, do đó có thể thiết kế các thiết bị ngoài như ổ đĩa cứng mang xách được ghép nối với PC qua cổng này. Tuy vậy cổng song song được dùng chủ yếu cho việc ghép nối với máy in và do vậy nó cũng có tên là LPT (Line PrinTer). Hiện nay thường có hai cổng LPT được lắp trên một bản mạch cắm vào khe cắm mở rộng hoặc được lắp đặt ngay trên bản mạch chính. Ba nhóm địa chỉ vào/ra được dùng cho các cổng song song tại các máy IBM và PS/2 có địa chỉ như sau (một cổng trước đây được lắp đặt trên các card màn hình CGA và HGC có nhóm địa chỉ vào/ra ở 3BCh - 3BFh đến nay hầu như không có).



Hình 8.1 Đầu nối D-25 cho cổng LPT

Trong quá trình khởi động, ROM BIOS sẽ kiểm tra 3 nhóm cổng vào/ra lần lượt từ 3BCh đến 378h và 278h để xem liệu có phần cứng cổng song song được lắp ở đó không. BIOS lưu trữ các địa chỉ vào/ra cơ sở này trong vùng dữ liệu tại đoạn 0040h và nếu phát hiện ra sự tồn tại của cổng sẽ coi chúng là những giao diện có số từ 0 đến 3 và DOS cũng đặt tên các cổng như bảng sau:

Nhóm địa chỉ I/O	Cổng song song
3BCh - 3BFh	Cổng trên card VGA hay HGC
378h - 37Fh	Cổng song song thứ 2 (378h - 37Ah)
278h - 27Fh	Cổng song song thứ 3 (278h - 27Ah)

Mặc dù ROM BIOS chỉ tìm 3 cổng song song, nhưng vùng dữ liệu của nó có thể dành để lưu trữ tới 4 địa chỉ vào/ra. Nếu ta cài đặt phần cứng cho một cổng song song thứ tư, ta có thể thâm nhập nó như giao diện 3 bằng các hàm của BIOS. Nếu đặt địa chỉ vào/ra cơ sở của nó ở địa chỉ 0040:000Eh trong vùng dữ liệu của BIOS thì MS-DOS không thể thâm nhập đến cổng thứ 4 này được.

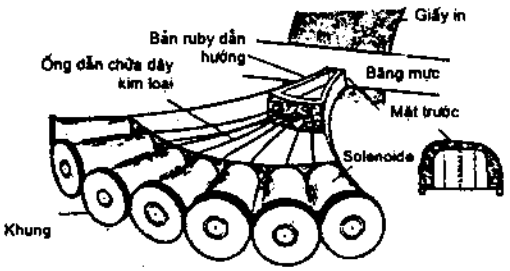
Vùng lưu giữ địa chỉ cơ sở của cổng	Giao diện (BIOS)	Tên thiết bị (MS-DOS)
0040:0008h	0	LPT1
0040:000Ah	1	LPT2
0040:000Ch	2	LPT3
0040:000Eh	3	

8.2.2. Ghép nối máy in qua cổng LPT

Có rất nhiều loại máy in nhưng ta sẽ chỉ xét 3 loại thông dụng nhất cho máy vi tính hiện nay là máy in ma trận điểm (ở Việt Nam thường gọi là máy in kim), máy in phun mực và máy in Laser.

a, Nguyên tắc hoạt động của máy in kim

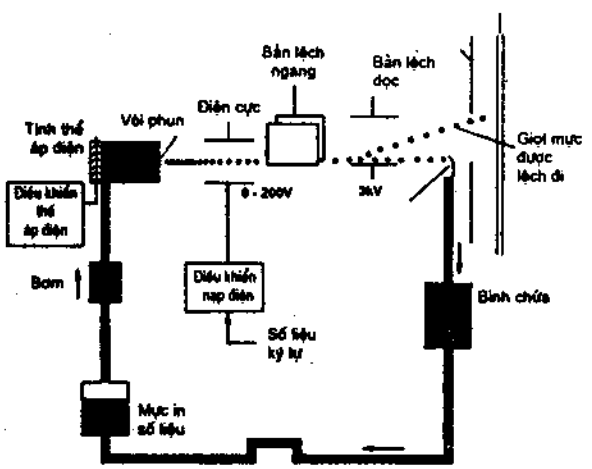
Hình 8.2 mô tả cơ chế hoạt động của một đầu in ma trận điểm ảnh của máy in kim. Các cuộn nam châm điện solenoid ở đây sẽ điều khiển 7 sợi dây sắt đặt trong lòng mỗi ống dây. Vì các dây được làm bằng chất sắt từ nên chúng có thể chuyển động vào ra lõi các cuộn solenoid khi có dòng điện chảy qua cuộn dây. Các dây được chụm lại bởi một bản dẫn hướng ruby để tạo thành một cột gồm 7 dây. Bằng việc đưa tín hiệu cấp năng lượng vào một nhóm các cuộn solenoid, một nhóm dây tương ứng sẽ được đẩy đập vào băng mực để tạo ra các hình mẫu điểm ảnh trên giấy. Sau mỗi cột điểm ảnh được in, đầu in được chuyển dịch đi một điểm để in cột khác. Sau 5 hoặc 7 cột được in tức là đã in xong một kí tự. Vì kí tự được tạo bởi các điểm ảnh nhỏ nên có thể chương trình cho bất kì một tập kí tự nào và tất nhiên một số máy in cũng cho phép lập trình cho các hình ảnh đồ hoạ điểm ảnh. Hiện nay đầu in 24 kim có thể cho một chất lượng tương đối cao.



Hình 8.2 Đầu in ma trận điểm.

b, Máy in phun mực

Máy in phun mực có đặc điểm cơ bản được minh hoạ trên hình 8.3. Một tia mực rất nhỏ được phát ra từ một vòi cho một dòng các giọt mực rất nhỏ có tốc độ cao. Vòi được rung với tần số siêu âm nhằm làm cho dòng mực được cắt ra thành những giọt tách biệt. Khi mỗi một giọt

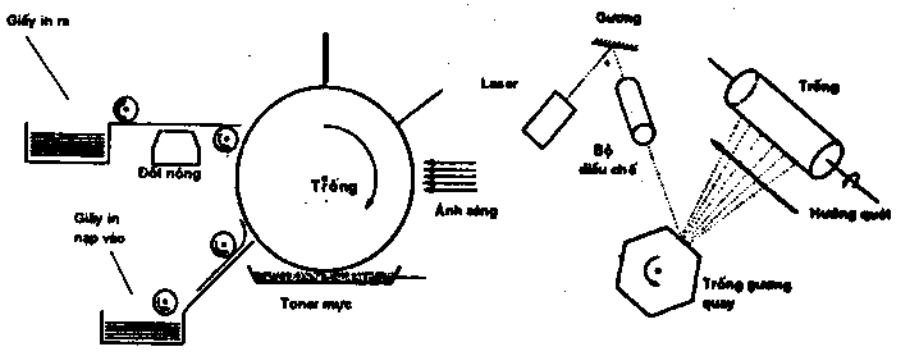


Hình 8.3 Máy in phun mực.

rời khỏi vòi, nó được tích điện để nhằm cho dòng các giọt mực có thể được lệch đi dưới tác dụng của một trường tĩnh điện khi đi qua 2 bản cực điện. Như vậy khi đưa tín hiệu số liệu vào 2 bản cực điện, có thể điều khiển được độ lệch chuyển động của tia và do đó có thể tạo được các ảnh mong muốn khi tia này đập lên giấy. Giấy in thường được thiết kế đặt ngoài tâm trục của tia mực để cho khi không có tín hiệu (trường điện bằng 0), dòng mực không đập vào giấy mà rơi vào một bình chứa chỏ có thể sử dụng lại. Máy in phun có độ phân giải cao (hơn 300 dot / inch), tốc độ in nhanh và hầu như không gây ra tiếng ồn.

c, Máy in laser

Nguyên tắc máy in laser hoạt động giống như một máy photocopy. Nguyên lí hoạt động của nó được chỉ ra trên hình 8.4. Phần quan trọng nhất của máy là một cái trống kim loại được chế tạo hết sức chính xác có phủ lớp nhạy quang, thí dụ như selenium. Kích thước của trống bằng kích thước của giấy in. Selenium khi được chiếu sáng sẽ từ một chất cách điện chuyển sang dẫn điện. Trong các máy in laser hiện đại, selenium được thay bằng các hợp chất hữu cơ khác. Đầu tiên, trống được nạp các tích điện lên cao cỡ hàng ngàn vôn bằng trường tĩnh điện. Dưới tác dụng của nguồn ánh sáng laser mạnh có cường độ phụ thuộc vào các điểm ảnh cần in, hình ảnh được copy vào trống khi cho nó quay. Sau một vòng quay, trống đã lưu giữ một hình ảnh về điện không nhìn thấy của bản gốc. Đó là vì các vùng có ánh sáng chiếu đến mạnh hơn sẽ dẫn điện và làm mất đi các điện tích nhiều hơn những vùng có ánh sáng chiếu tới yếu. Nói cách khác sự phân bố mật độ điện tích tại các điểm trên trống sẽ phản ánh đúng sự tồn tại các điểm ảnh của bản gốc. Bước tiếp theo là cho trống quay gần một toner mực gồm các hạt bột mực rất nhỏ. Các hạt mực này sẽ được hút vào mặt trống ít hay nhiều tuỳ thuộc vào mật độ điện tích tại các điểm. Vùng có ánh sáng chiếu đến mạnh (tương ứng với vùng trắng hơn ở ảnh gốc) sẽ lưu giữ ít điện tích và sẽ hút ít mực hơn. Bước tiếp theo là cho trống quay tiếp xúc với mặt giấy với điều kiện giấy đã được tích điện thậm chí còn cao hơn trống. Kết quả là các hạt mực trên trống sẽ được bám vào giấy tại những điểm mà chúng tiếp xúc với giấy. Khâu cuối cùng là đốt nóng bề mặt giấy nhằm làm cho mực bám chặt mãi vào giấy. Độ phân giải của hình ảnh phụ thuộc vào hệ thống quang học và kích thước của tia laser cho ta tới 300 điểm trên một inch. Độ phân giải này đủ thích hợp cho những văn bản đòi hỏi chất lượng cao. Với những hình ảnh chất lượng cao hơn nữa độ phân giải có thể lên tới 800 điểm ảnh trên 1 inch



Hình 8.4 Nguyên tắc hoạt động của máy in laser.

d, Ghép nối máy in qua cổng LPT

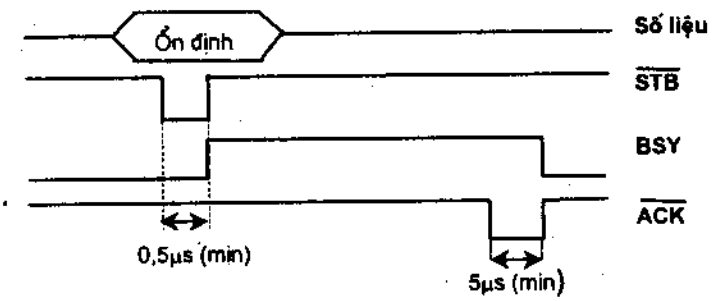
Mỗi cổng LPT có 3 thanh ghi có thể thâm nhập được qua 3 địa chỉ cổng vào/ra gán cho chúng. Định dạng của 2 thanh ghi trạng thái và điều khiển như sau:

Bit								Ý nghĩa thanh ghi trạng thái
7	6	5	4	3	2	1	0	
x	.	.	.	.	.	.	.	1 = máy in không bận
.	x	.	.	.	.	.	.	0 = máy in đã sẵn sàng nhận kí tự tiếp theo
.	.	x	.	.	.	.	.	1 = máy in không có giấy
.	.	.	x	.	.	.	.	1 = máy in ở trạng thái on-line
.	.	.	.	x	.	.	.	0 = có lỗi
.	.	.	.	.	x	.	.	không dùng
.	.	.	.	.	.	x	.	không dùng
.	.	.	.	.	.	.	x	không dùng

Bit								Ý nghĩa thanh ghi điều khiển
7	6	5	4	3	2	1	0	
x	.	.	.	.	.	.	.	không dùng
.	x	.	.	.	.	.	.	không dùng
.	.	x	.	.	.	.	.	không dùng
.	.	.	x	.	.	.	.	1 = chạy ngắt khi ACK = 0
.	.	.	.	x	.	.	.	1 = chọn máy in (on line)
.	.	.	.	.	x	.	.	0 = khởi động máy in
.	.	.	.	.	.	x	.	1 = tự động xuống dòng
.	.	.	.	.	.	.	x	1 = truyền số liệu tới máy in

Các thanh ghi này được dùng để gửi số liệu tới máy in, xác định trạng thái máy in và điều khiển hoạt động của máy in. Thanh ghi số liệu được thâm nhập ở địa chỉ vào/ra cơ sở (thí dụ 378h). Những PC hiện nay đều cho phép thanh ghi này có thể được sử dụng cả viết và đọc số liệu (2 hướng). Thanh ghi trạng thái có địa chỉ bằng địa chỉ cơ sở cộng 1 (thí dụ 379h). Đây là thanh ghi chỉ đọc. Có thể đọc trực tiếp nó bằng các lệnh vào/ra hoặc dùng ngắt 17h, hàm 02h trong BIOS. Thanh ghi điều khiển được thâm nhập ở địa chỉ cơ sở cộng 2 (thí dụ 37Ah) và có thể dùng cho cả đọc lẫn viết.

Vì tốc độ truyền kí tự của cổng song song nhanh hơn tốc độ in của máy in nên thông tin ở đây phải có móc nối (hình 8.5). Khởi đầu, PC đặt các số liệu lên bus sau đó kích hoạt đường $\overline{STB}$ xuống mức thấp để thông tin cho máy in biết rằng số liệu đã ổn định trên bus. Khi máy in xử lí xong dữ liệu, nó sẽ trả lại tín hiệu $\overline{ACK}$ xuống mức thấp để ghi nhận. PC đợi cho đến khi đường BSY từ máy in xuống thấp (máy in không bận) thì sẽ đưa tiếp số liệu lên bus.



Hình 8.5 Móc nối thông tin với máy in.

Lập trình cho máy in có thể được thực hiện bằng ngôn ngữ bậc cao, bằng các dịch vụ trong MS-DOS qua ngắt 21h và hàm 40h với giá trị handle 4 cho LPT1. Để dùng hàm này gửi số liệu

tới một cổng máy in khác ta phải hàm 3Dh để mở cổng mong muốn (LPT2 hoặc LPT3) như một thiết bị rồi cho qua handle của cổng khi gọi ngắt 21h hàm 40h. Có thể in một kí tự đơn tới LPT1 bằng ngắt 21h hàm 05h.

Ngắt 17h trong BIOS với 3 hàm có thể làm việc với bất kì cổng song song nào. Chức năng cùng các thanh ghi vào (cần nạp trước khi gọi ngắt) và các thanh ghi ra (chứa các byte trạng thái sau khi ngắt) như bảng sau.

Chức năng	Các thanh ghi vào	Các thanh ghi ra
Khởi động cổng máy in	AH = 01h (hàm 01h) DX = số máy in (00h-02h)	AH: chứa trạng thái máy in
Viết một kí tự tới máy in	AH = 00h (hàm 00h) AL = kí tự cần in DX = số máy in	AH: chứa trạng thái máy in
Nhận trạng thái máy in	AH = 02h (hàm 02h) DX = số máy in	AH: chứa trạng thái máy in

Trạng thái của máy in được biết qua việc đọc thanh ghi trạng thái cổng. Mức thấp nhất của việc lập trình với cổng song song được thực hiện bằng việc viết hoặc đọc trực tiếp các cổng vào/ra liên quan đến cổng. Cách này thường được dùng khi cần truyền số liệu hai hướng giữa hai máy tính qua các cổng song song.

8.3. CỔNG THÔNG TIN NỐI TIẾP VÀ GHÉP NỐI MODEM

Thông tin nối tiếp được dùng trong rất nhiều ứng dụng ghép nối với máy vi tính như chuột, modem, máy vẽ, máy in, máy quét v.v... Vì dữ liệu được truyền song song trên các bus máy tính nên để truyền nối tiếp được, bên phát dùng các *thanh ghi dịch song song-nối tiếp* để biến đổi các byte song song thành một chuỗi bit nối tiếp nhau cho ra cổng và ở bên thu phải dùng các *thanh ghi dịch nối tiếp - song song* để biến đổi ngược lại.

Thông tin nối tiếp được phân biệt thành hai loại: truyền nối tiếp đồng bộ và không đồng bộ. Khi *truyền nối tiếp đồng bộ*, ngoài đường dây truyền số liệu giữa hai trạm phát và thu cần có thêm một đường dây điều khiển để truyền tín hiệu nhịp nhằm để bên thu xác định được các thời điểm tại đó số liệu trên đường truyền đã ổn định. Trong khi đó, *truyền nối tiếp không đồng bộ* chỉ cần 1 dây truyền trong đó các thông tin đồng bộ được truyền ngay cùng với các từ dữ liệu. Đó là các bít *start* (bit bắt đầu) chỉ thị sự bắt đầu của khối số liệu được truyền và bít *stop* (bit kết thúc) báo kết thúc khối số liệu đó cùng một số *bit phát hiện và sửa lỗi* được ghép cùng các bit số liệu để tạo thành một *khung truyền* (frame) hay một *SDU* (serial data unit).

Giữa hai cổng thông tin nối tiếp có thể có các phương thức trao đổi thông tin như sau:

Nối đơn công (Simplex Connection): số liệu chỉ được truyền theo một hướng.

Bán song công (Half-Duplex): số liệu truyền theo hai hướng, nhưng mỗi thời điểm chỉ được truyền theo một hướng.

Song công (Full-Duplex): số liệu được truyền đồng thời theo hai hướng.

8.3.1. Chuẩn thông tin nối tiếp

Do hạn chế về dải tần của đường truyền nên hầu hết các thiết bị đầu cuối số liệu DTE (Data Terminal Equipment) muốn thông tin với nhau không thể nối trực tiếp với môi trường truyền dẫn analog được mà phải thông qua các thiết bị thông tin số liệu DCE (Data

Communication Equipment). Thí dụ, việc thông tin giữa hai máy tính hoặc máy fax là các DTE qua đường điện thoại công cộng phải được nối qua hai thiết bị DCE là các Modem như hình 8.6.

Các tiêu chuẩn chính cho thông tin số liệu nối tiếp hiện nay được xây dựng bởi các tổ chức ITU (International Telecommunications Union), EIA (Electronics Industry Association) và ISO (International Standards Organisation). Các chuẩn ITU về thông tin nối tiếp thuộc về nhóm V và các chuẩn EIA về thông tin nối tiếp thuộc về nhóm RS. Chuẩn RS-232C quy định các ghép nối nối tiếp giữa một DTE và DCE với khoảng cách cực đại là 17 đến 20m và tốc độ truyền số liệu cực đại lên tới 20 kbps.

Hình 8.6 Các DTE và DCE.

Tín hiệu điện theo chuẩn RS-232C là lưỡng cực: mức lôgic 1 có điện thế dương so với đất (0V), mức 0 có điện thế âm so với đất.

Với tín hiệu ra: mức lôgic cao có điện thế trong dải từ +5V đến +15V
mức lôgic thấp từ -5V đến -15V.

Với tín hiệu vào: mức lôgic cao từ +3V đến +15V
mức lôgic thấp từ -3V đến -15V.

Lôgic trên đường truyền số liệu TD và RD được đổi thành lôgic âm, nghĩa là mức lôgic cao có điện thế âm so với đất (0V) và mức lôgic thấp có điện thế dương so với đất. Dữ liệu trên đường truyền chỉ ở một trong hai trạng thái MARK hoặc SPACE, lần lượt tương ứng với trạng thái điện thế âm hoặc dương tức là tương ứng với mức lôgic 1 hoặc 0. Với các máy PC, các mức điện thế điển hình là ±12V. Dữ liệu được truyền lần lượt theo từng nhóm bít. Mỗi nhóm gọi là một SDU hay một khung truyền. Một khung truyền bao gồm:

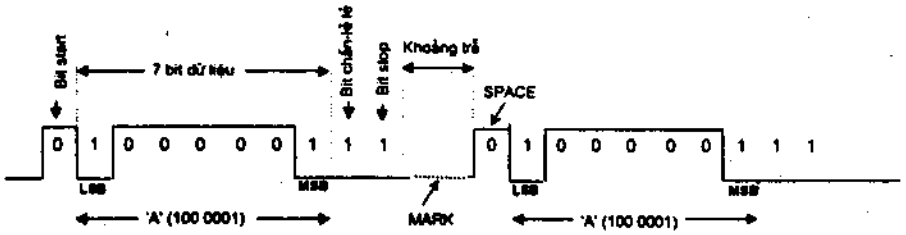
1 *bit start* luôn ở mức lôgic thấp, điện thế dương.

1 hoặc 1,5 hoặc 2 *bit stop* luôn ở mức lôgic cao, điện thế âm.

1 hoặc không có 1 *bit kiểm tra chẵn lẻ*,

5, 6 hoặc 7 *bit số liệu*.

Thí dụ, khi các khung truyền đại diện cho các kí tự (với mã ASCII là 7 bit) được truyền trên đường dây lần lượt với một khoảng thời gian trễ giữa chúng. Trong khoảng thời gian này đường truyền ở vào trạng thái MARK (mức lôgic cao). Hình 8.7 là một thí dụ về tín hiệu nhận được trên đường truyền TxD hoặc RxD khi truyền các bit thông tin mã hoá ASCII cho 2 chữ 'A' (100 0001 = 41h) với bit chẵn-lẻ lẻ.



Hình 8.7 Tín hiệu trên đường truyền của các bit biểu diễn cho 2 kí tự 'A'.

Tốc độ truyền số liệu nối tiếp được đo bằng số bit truyền trong một giây bps. Trong các hệ thống truyền số liệu với mã nhị phân tốc độ này trùng với tốc độ baud là số lần thay đổi trạng thái của một kênh truyền tín hiệu trong một giây. Tất cả các máy PC đều có cổng ghép nối thông tin nối tiếp. Cổng sơ cấp gọi là COM1 (hoặc COM3) và thứ cấp là COM2 (hoặc COM4). Có 2 loại đầu cắm tín hiệu cho các cổng này là D-25 (25 chân) và D-9 (9 chân) thường được gắn ở phía sau hộp máy (hình 8.8). Sử dụng loại nào cho COM1 hoặc COM2 là tuỳ ở người lắp ráp máy tính, thường thì hay dùng đầu cắm D-9 cho COM1. Các đầu cắm cho các cổng nối tiếp gắn trên hộp máy PC bao giờ cũng là loại đầu cắm đực (male), đầu cắm ở cáp nối ra các thiết bị ngoại vi là đầu cắm cái (female). Việc này để tránh nhầm lẫn với đầu cắm D-25 dùng cho cổng song song LPT gắn trên hộp máy PC luôn là loại đầu cắm cái (female).

Hình 8.8 Các đầu cắm D-25 và D-9 của các cổng COM ở hộp máy PC.

Các chân tín hiệu trên các đầu cắm được nối ra các đường dây để các thiết bị DTE và DCE thông tin với nhau. Ngoài dây đất GND có điện thế 0V, có thể phân thành hai nhóm đường dây gồm nhóm các đường truyền dữ liệu TxD, RxD và nhóm các đường tín hiệu điều khiển (gọi là các tín hiệu móc nối thông tin) gồm các đường còn lại. Bảng sau mô tả tên và chức năng của các đường tín hiệu.

Chân số		Tên	Kí hiệu	Chức năng
D-25	D-9			
1	-	Frame Ground	FG	Thường được nối với vỏ bọc kim của cáp dẫn hoặc đất.
2	3	Transmit Data	TxD	Số liệu được phát từ DTE (thí dụ PC hoặc thiết bị đầu cuối) tới DCE qua đường TxD.
3	2	Receive Data	RxD	Số liệu được thu từ DCE vào DTE.
4	7	Request To Send	RTS	DTE đặt đường này ở mức tích cực khi nó sẵn sàng phát số liệu.
5	8	Clear To Send	CTS	DCE đặt đường này ở mức tích cực để thông tin cho DTE rằng nó sẵn sàng nhận số liệu.
6	6	Data Set Ready	DSR	Chức năng tương tự như CTS nhưng được kích hoạt bởi DTE khi nó sẵn sàng nhận số liệu.
20	4	Data Terminal Ready	DTR	Chức năng tương tự như RTS nhưng được kích hoạt bởi DCE khi nó muốn phát số liệu.
8	1	Data Carier Detect	DCD	DCE đặt đường này ở mức tích cực để báo cho DTE biết là đã thiết lập được liên kết với DCE từ xa (nhận được sóng mang từ bên DCE đối tác)
22	9	Ring Indicator	RI	DCE (loại lắp ngoài) báo với DTE có một cuộc gọi từ xa vừa gọi đến.
7	5	Signal Ground	SG	GND

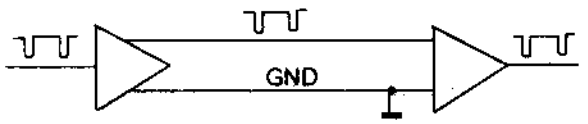
Các chuẩn RS-422A (có đường truyền cân bằng) và RS-423A (có đường truyền không cân bằng) cho phép tăng tốc độ truyền số liệu và khoảng cách thông tin lên lớn hơn nữa.

Chuẩn RS-485 là một chuẩn tăng cường của RS-422. Nó quy định cho các ghép nối thông tin đa điểm và song công, do vậy rất thích hợp cho các ứng dụng trên mạng máy tính. Tốc độ truyền cực đại ở đây không hạn chế và được đặt bởi thời gian tăng của sườn xung, thường cỡ 10Mbps. Với độ dài cáp 1,2 km mạng máy tính dùng chuẩn RS-485 cho phép ghép nối tới 32 cặp máy thu/phát.

Đường dây dẫn tín hiệu ảnh hưởng đến các xung số liệu ở 3 trường hợp sau:

- Làm suy giảm biên độ xung: do điện trở của đường dây.
- Làm méo dạng xung: do điện dung kí sinh giữa các cặp đường dây, điện trở và điện cảm của đường dây gây nên méo dạng pha và giảm dải truyền.
- Gây ồn: do ồn nội (ồn nhiệt của đường dây) và ồn ngoại là các tác nhân bên ngoài tác động lên đường truyền.

Ngoài đường dây chung (dây đất), các đường truyền không cân bằng (như theo chuẩn RS-423) sử dụng một đường dây cho tín hiệu số trong khi các đường truyền cân bằng (như theo chuẩn RS-422) sử dụng hai đường dây cho mỗi tín hiệu như các hình 8.9 và 8.10 chỉ ra. Dòng điện trong 2 dây dẫn của đường truyền cân bằng ngược pha với nhau. Do vậy, đường truyền này có ưu điểm là ít chịu ảnh hưởng của ồn ngoại vì các nguồn này đều cùng gây nên trên hai dây dẫn cường độ ồn bằng nhau về độ lớn nhưng có pha ngược nhau. Khi hoạt động ở tốc độ



Hình 8.9 Đường truyền ghép nối không cân bằng (RS-423).

Hình 8.10 Đường truyền ghép nối cân bằng (RS-422).

truyền cao, các đầu cuối của dây dẫn thường được nối với các *terminator* có trở kháng bằng trở kháng đặc tính (trở sóng) của đường dây (thí dụ, 50 hay 70 Ω). Các mạch ghép nối theo chuẩn RS-422 có thể tải được tới 10 máy thu. Chúng không có đường dây nối đất do vậy được dùng rất tốt khi cần phải cách li giữa 2 nút trong mạng thông tin. Nếu sử dụng tốc độ truyền thấp thì có thể tăng được khoảng cách thông tin lên. Thí dụ, trong một vài trường hợp khi chất lượng cáp dẫn tốt và môi trường ít nhiễu thì có thể sử dụng các thiết bị với chuẩn RS-232 thông tin với nhau ở khoảng cách lên tới 1 km với tốc độ 1200 bps.

Bảng sau cho các thông số của các chuẩn ghép nối thông tin nối tiếp cơ bản.

EIA	RS-232C	RS-423A	RS-422A	RS-485
ITU	V.28	V.10/X26	V.11/X27	
Tốc độ truyền số liệu	20kbps	300kbps	10Mbps	10Mbps
Khoảng cách cực đại	17-20m	1200m	1200m	1200m
Loại nối tải	không cân bằng	không cân bằng vi sai	cân bằng vi sai	cân bằng vi sai
Số cặp thu-phát	1	1	1	32
Điện thế điều khiển	±3V + ±15V	±2V + ±6V	±2V + ±5V	±2V + ±5V
Số đường dây cho một tín hiệu	1	2	2	2

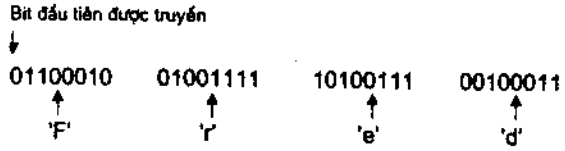
8.3.2. Truyền nối tiếp các mã ký tự ASCII

Các dữ liệu gồm các con số, kí tự v.v... được truyền từ nguồn tới đích thường được mã hoá. Ba tập mã kí tự thông dụng hiện nay là mã Baudot, mã ASCII (American Standards Code for Information Interchange) và mã EBCDIC (Extended Binary-Coded Decimal Interchange Code). Việc truyền mã ASCII trong thông tin nối tiếp ở máy vi tính là rất thông dụng. Đó là việc truyền các kí tự được mã hoá bằng các số nhị phân 7 bit với 96 ký tự in ra được và 32 ký tự không in được dùng cho điều khiển như được trình bày trong bảng sau.

Bảng 8.1 **Bảng mã ASCII**

Số hex	Kí tự	Số hex	Kí tự	Số hex	Kí tự	Số hex	Kí tự
00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	NL	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DEL	30	0	50	P	70	p
11	DC_1	31	1	51	Q	71	q
12	DC_2	32	2	52	R	72	r
13	DC_3	33	3	53	S	73	s
14	DC_4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	EBT	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

Tại các thiết bị phát dùng chuẩn RS-232C, bit ít ý nghĩa nhất LSB của kí tự ASCII được truyền đầu tiên, bít nhiều ý nghĩa nhất MSB truyền sau cùng. Thí dụ sau là các nhóm bit đại diện cho thông báo "Fred" được truyền nối tiếp với bit chẵn-lẻ lẻ.



Các kí tự in được thuộc về các chữ cái thường và chữ hoa ('a' - 'z', 'A' - 'Z') các chữ số ('0' - '9') và các kí tự khác như kí tự phân cách (SPACE), '!', '#' v.v... Lưu ý rằng các mã chữ thường và hoa chỉ khác nhau ở bit thứ 5 (hơn kém nhau giá trị 2^5= 32). Dưới đây giải thích một vài kí tự điều khiển.

BS (Backspace): Nếu được gửi tới màn hình nó sẽ xoá kí tự được gửi ra màn hình trước đó. Khi nhấn phím cách (*backspace)* trên bàn phím nó sẽ phát ra một kí tự ^H (Cntrl-H).

HT (Horizontal Tab): Chuyển con trỏ màn hình một khoảng cách (tab) về bên phải vị trí hiện tại. Thường khoảng đó được đặt bởi phần mềm. Khi nhấn phím TAB trên bàn phím nó sẽ trả về kí tự ^I hoặc khi nhấn ^I sẽ thực hiện được chức năng kể trên.

LF (Line feed) và **CR** (Carriage Return): Kí tự CR đặt con trỏ tới vị trí đầu dòng. Kí tự LF đặt con trỏ xuống dưới một dòng.

FF (From Feed): Nếu được gửi tới màn hình sẽ dịch vị trí con trỏ đi một vị trí về bên phải (chức năng ngược với BS) hoặc được gửi tới máy in

VT (Vertical Tab): Nếu được gửi tới màn hình nó sẽ chuyển vị trí con trỏ lên một đường phía trên. Khi nhấn phím ^K trên bàn phím cũng cho phép thực hiện chức năng này.

NUL (Null): Thường được dùng để pad sự bắt đầu của các kí tự cần truyền và để phân ranh giới một xâu kí tự.

BEL (Bell): Khi truyền kí tự này tới một thiết bị phát âm nó sẽ phát ra một âm thanh. Chức năng tương tự khi nhấn ^G trên bàn phím.

DC1-DC4 (Device Control): Điều khiển hoạt động của các màn hình và các máy in. Một máy thu sử dụng kí tự DC1 (^Q) để báo cho máy phát ngừng phát. Kí tự DC3 (^S) báo cho máy phát lại tiếp tục phát. Loại thông tin này thuộc loại móc nối mềm (software handshaking).

Esc (Escape): Một số phần mềm dùng kí tự này cho việc thoát khỏi các menu hoặc tuỳ chọn.

Del (Delete): Xoá kí tự ở vị trí hiện tại của con trỏ.

EXT (End of Text): Dùng để ngắt một xử lí (^C).

EOT (End of Transmission): Hệ điều hành UNIX dùng kí tự này để báo rằng người sử dụng đã kết thúc việc nhập dữ liệu. Giống kí tự end-of-file (^D).

SUB (Substitute): Dùng để xác định một kí tự kết thúc tệp (end-of-file EOF).

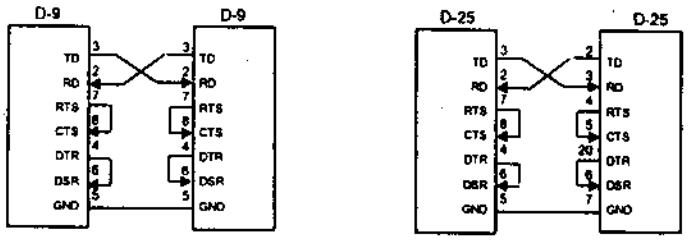
8.3.3. Điều khiển thông tin nối tiếp giữa hai thiết bị

Tuỳ thuộc vào tương quan giữa các tốc độ truyền và thu số liệu mà có thể cần phải có sự điều khiển thông tin hay không. Nếu máy thu có khả năng đọc các kí tự thu trước khi máy phát truyền kí tự tiếp theo thì không cần có sự điều khiển này. Đó là thông tin không có móc nối (no-handshake). Máy thu nhận các kí tự và lưu trữ vào một vùng nhớ đặc biệt trước khi đọc nó. Vùng nhớ này gọi là *bộ đệm thu* (receive buffer). Điển hình vùng này chỉ lưu giữ một ký tự. Nếu các kí tự đã được đưa đến đây không được đọc kịp trước khi nhận kí tự khác (ta nói bộ đệm chưa được rỗng) thì có thể xảy ra hiện tượng kí tự hiện tại sẽ bị viết đè lên bởi các kí tự khác đến sau. Khi xảy ra điều này thì cần phải thực hiện việc điều khiển thông tin giữa bên phát và bên thu. Kĩ thuật điều khiển thông tin giữa hai thiết bị để tránh điều trên xảy ra gọi là *kĩ thuật móc nối* (handshaking). Đó là việc buộc máy phát ngừng truyền kí tự cho đến khi máy thu đọc xong các kí tự đang nằm trong bộ đệm thu, tức là cho đến khi bộ đệm thu rỗng. Có hai loại móc nối:

Móc nối cứng (hardware handshaking) dùng các đường tín hiệu điều khiển móc nối như CTS, RTS, DTR và DSR.

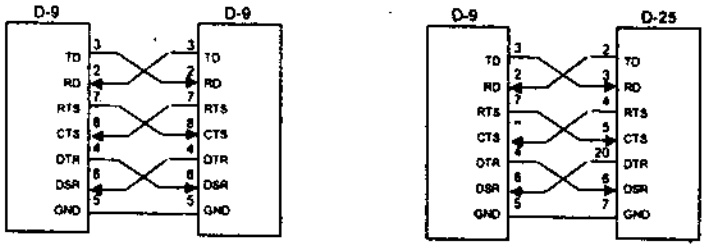
Móc nối mềm (software handshaking) liên quan tới việc gửi các kí tự điều khiển đặc biệt như các kí tự từ D1 đến D4 trong bảng mã ASCII để truyền thông tin móc nối giữa bên phát và bên thu.

Hình 8.11 là sơ đồ nối các đường dây giữa hai trạm thu/phát trong thông tin không có móc nối. Trong trường hợp này máy thu có thể đọc số liệu nhận được từ bộ đệm thu trước khi kí tự khác được nhận. Số liệu được truyền từ đầu TxD của máy phát và được nhận ở đầu RxD ở bên thu. Khi một DTE (thí dụ, máy PC) cần nối với một DTE khác thì đường TxD ở một bên được nối với đường RxD ở bên kia và ngược lại. Các đầu tín hiệu móc nối ở mỗi bên được nối cặp với nhau (RTS nối trực tiếp với CTS, DTR với DSR) nhằm làm cho các máy luôn ở trạng thái sẵn sàng thu và sẵn sàng truyền.



Hình 8.11 Nối dây thông tin giữa hai cổng RS-232C theo kiểu không móc nối.

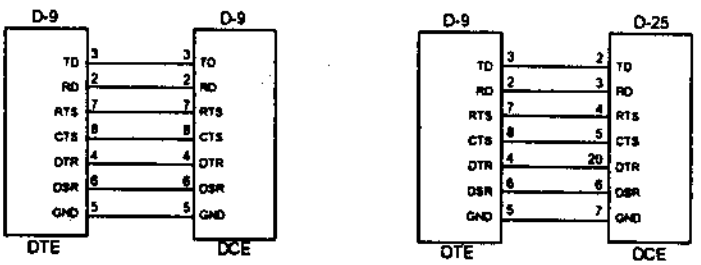
Hình 8.12 là sơ đồ nối dây trong thông tin với móc nối cứng giữa 2 DTE với nhau.



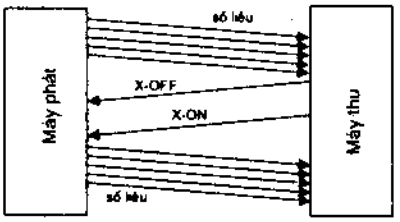
Hình 8.12 Thông tin RS-232C có móc nối (móc nối cứng).

Các đường điều khiển được dùng ở mức tích cực cao. Khi một máy muốn phát số liệu nó nâng dây RTS của mình lên mức cao rồi kiểm tra trạng thái dây CTS cho đến khi dây này ở mức tích cực cao. Chừng nào dây này còn ở mức thấp thì có nghĩa là máy thu vẫn còn bận và chưa thể nhận được số liệu. Khi máy thu đọc xong số liệu từ bộ đệm thu của nó, đường RTS nối với CTS của máy phát sẽ tự động nhảy lên mức cao để chỉ thị cho máy phát rằng máy thu bây giờ đã sẵn sàng nhận số liệu tiếp theo. Quá trình nhận số liệu cũng giống như quá trình truyền. Khi DCE muốn phát tới DTE, đầu vào DSR bên máy thu sẽ trở nên tích cực. Nếu máy thu không thể nhận số liệu nó sẽ đặt đường DTR ở mức không tích cực. Khi nó ở trạng thái xoá nhận, nó sẽ đặt đường DTR ở mức tích cực và máy bên kia sẽ được phép phát số liệu. Đường DTR sẽ được đặt ở mức không tích cực cho tới khi kí tự được xử lí xong. Khi nối DTE với DCE thì các đường móc nối được nối trực tiếp với nhau như hình 8.13.

Hình 8.13 Nối DTE với DCE.



Hình 8.14 là giản đồ mô tả quá trình thông tin với móc nối mềm. Có hai kí tự ASCII được dùng để khởi phát và dừng truyền số liệu.



Hình 8.14 Móc nối mềm dùng hai kí tự X-OFF và X-ON.

Đó là kí tự DC1 trong bảng mã ASCII có giá trị là 11h gọi là X-ON và kí tự DC3 có giá trị 13h gọi là X-OFF. Khi máy phát nhận được kí tự X-OFF nó sẽ ngừng phát số liệu cho đến khi nhận được kí tự X-ON. Loại móc nối này thường được dùng trong trường hợp khi máy phát và máy thu có thể xử lí số liệu tương đối nhanh. Thường máy thu có một bộ đệm số liệu lớn chứa các kí tự nhận được. Khi bộ đệm này đầy, nó sẽ truyền sang bên phát kí tự X-OFF, sau khi đọc xong hết nội dung bộ đệm nó sẽ truyền kí tự X-ON.

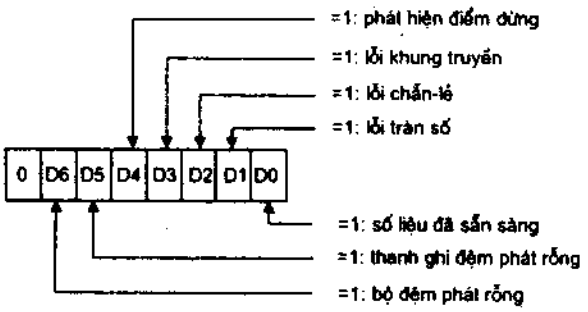
8.3.4. Lập trình thông tin nối tiếp

a, Lập trình trực tiếp với các cổng nối tiếp

Chip UART-8250 được dùng trong máy vi tính PC/XT phục vụ cho thông tin nối tiếp với tốc độ truyền số liệu cực đại là 9600 bps. Từ máy AT trở đi dùng chip 16450 hay 82450 cho phép một tốc độ truyền cao hơn đến 19200 bps. Các chip này được lắp ngay trên bản mạch chính hoặc được lắp trên bản mạch ghép nối vào/ra cắm trên khe cắm mở rộng của máy tính. Địa chỉ của một số thanh ghi như sau:

Địa chỉ cơ sở →		
COM1 = 3F8h	Đệm số liệu phát và thu (TD/RD)	Địa chỉ cơ sở
COM2 = 2F8h	Cho phép ngắt	Địa chỉ cơ sở + 1
	Nhận dạng ngắt	Địa chỉ cơ sở + 2
	Định dạng số liệu (LCR)	Địa chỉ cơ sở + 3
	Điều khiển Modem	Địa chỉ cơ sở + 4
	Trạng thái đường truyền (LSR)	Địa chỉ cơ sở + 5
	Trạng thái Modem	Địa chỉ cơ sở + 6
	Lưu trữ đọc/viết tạm thời	Địa chỉ cơ sở + 7

- Các thanh ghi đệm số liệu phát và thu có cùng một địa chỉ bằng địa chỉ cơ sở.
- Thanh ghi trạng thái đường truyền xác định trạng thái của các bộ đệm thu và phát. Đây là thanh ghi chỉ đọc (tất cả các bit được ghi tự động bởi phần cứng). Định dạng của từ trạng thái như sau:



Khi truyền dữ liệu, một kí tự mới có thể được viết tới bộ đệm trước khi kí tự trước đó được gửi. Để tránh điều này, bít số sáu D6 được kiểm tra để xác định liệu vẫn còn một kí tự trong bộ đệm phát không. Nếu bit này được đặt lên 1 tức là bộ đệm phát đã rỗng. Vậy để gửi một kí tự phải viết một vòng lặp kiểm tra sau:

Kiểm tra bit D6 cho tới khi nó được đặt =1
Gửi kí tự

Thí dụ, trong ngôn ngữ Pascal sẽ là:

```
repeat
    trang_thai := port [dia chiLSR] and $40;
until (trang_thai = $40);
```

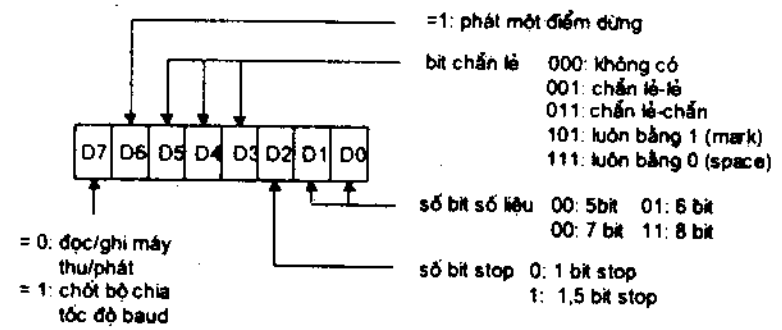
Khi thu số liệu, bit D0 được kiểm tra để xác định liệu đã có số liệu trong bộ đệm thu chưa theo vòng lặp sau:

Kiểm tra bit D0 cho đến khi nó được đặt = 1;
Đọc kí tự

Thí dụ, trong ngôn ngữ Pascal sẽ là:

```
repeat
   trang_thai := port [dia chi LSR] and $01;
until (trang_thai = $01);
```

• Thanh ghi định dạng dữ liệu (LCR) khởi tạo các thông số thông tin như số bit cho một kí tự, số bit chẵn lẻ, số bit stop. Nó có thể được viết hoặc đọc. Định dạng các bit điều khiển trong thanh ghi như sau:



Để thâm nhập thanh ghi đệm thu/phát bit D7 phải xoá bằng 0. Còn khi muốn khởi tạo bộ chia tốc độ baud, bit được đặt lên 1. Tốc độ baud được đặt bằng việc nạp một hệ số chia N, 16 bit vào địa chỉ của bộ đệm số liệu thu/phát và địa chỉ tiếp theo. Giá trị được nạp phụ thuộc vào tần số máy phát sóng thạch anh nối với UART. Thường tần số này bằng 1,8432 MHz. Do đó:

$$Tốc\ độ\ baud = \frac{tần\ số\ nhịp\ đồng\ hồ}{16 \times N}$$

Bảng sau tính sẵn một số hệ số chia N (được nạp vào thanh ghi đệm số liệu TD/RD) ứng với các tốc độ baud tương ứng.

Tốc độ Baud	Hệ số chia	Tốc độ Baud	Hệ số chia
110	0417h	2400	0030h
300	0180h	4800	0018h
600	00C0h	9600	000Ch
1200	0060h	19200	0006h
1800	0040h		

Để nạp hệ số chia tốc độ baud, trước hết bit 7 của thanh ghi LCR được đặt lên 1. Sau đó byte thấp của N được nạp vào địa chỉ thanh ghi đệm số liệu rồi byte cao được nạp tiếp vào địa chỉ lớn hơn 1. Cuối cùng bit 7 được xoá về 0. Thí dụ, để có tốc độ 9600 baud với tần số nhịp là 1,8432 MHz ở cổng COM1; trước hết số 0Ch được nạp vào địa chỉ 3F8h, sau đó số 00h được nạp vào địa chỉ 3F8h +01h = 3F9h. Khi bit 7 được xoá về 0, một thao tác đọc từ địa chỉ cơ sở sẽ đọc từ bộ đệm RD và một thao tác viết sẽ viết tới bộ đệm TD.

b, Lập trình qua DOS và BIOS

Một máy PC chuẩn có thể hỗ trợ tới 4 cổng COM. Những địa chỉ cơ sở của chúng được đặt trong bộ nhớ BIOS và địa chỉ của mỗi cổng được lưu trữ ở vùng địa chỉ

0040:0000 (COM1); 0040:0002 (COM2); 0040:0004 (COM3) và 0040:0008 (COM4).

- Lệnh MODE COM [:] tốc độ baud [, chẵn-lẻ [, bề rộng từ [, số bit stop [, p]]]]

được dùng để đặt các tham số của cổng nối tiếp. Thí dụ, lệnh khởi tạo cổng COM2 với tốc độ 2400 baud, một bit chẵn lẻ-chẵn, 8 bit số liệu và một 1 bit stop như sau:

```
mode com2:2400,e,8,1
```

- Lệnh prt = (int far *) 0x0400000 ; khởi động một con trỏ xa để khởi phát sự thông tin của BIOS với các địa chỉ cổng. Mỗi địa chỉ là 16 bit do vậy con trỏ sẽ trỏ tới một giá trị nguyên. Một con trỏ xa được dùng như vậy có thể thâm nhập được tới 1 MB nhớ trong khi một con trỏ gần chỉ có thể thâm nhập tới cực đại là 64 kB. Chương trình sau là một ví dụ áp dụng:

Chương trình 8.1

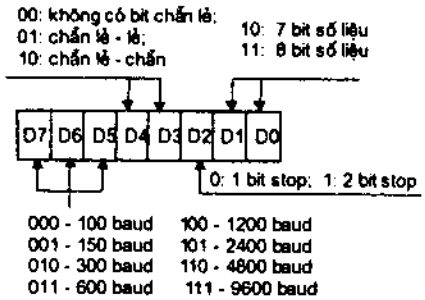
```
# include <stdio.h>
  # include <conio.h>
  int main (void)
 {
  int far * ptr;                          / * con trỏ 20 bit */
          ptr = (int far ) 0x400000;      /* 0040x0000 */          clrscr ( );
          printf ("COM1: %04x\n",*ptr);
          printf ("COM2: %04x\n",*(ptr + 1));
          printf ("COM1: %04x\n",*(ptr + 2));
          printf ("COM1: %04x\n",*(ptr + 3));
  return ( );
 }
```

Việc kiểm tra và điều khiển các cổng COM được lập trình qua BIOS. Hàm sử dụng trong C là: int bioscom (int cmd, char abyte, in port)

Trong đó port = 0 cho cổng COM1; port = 1 cho cổng COM2;

cmd: 0 - đặt các thông số thông tin tới giá trị được cho bởi abyte
1 - gửi kí tự
2 - nhận kí tự
3 - trả về trạng thái của thông tin. Khi cmd được đặt tới 0, thiết bị được chương trình hoá

Trong chế độ này, định nghĩa các bit trong byte được cho như sau:



Khi cmd = 3, giá trị trả về là một số nguyên 16 bit không dấu.

Các bit từ 8 đến 15 được định nghĩa như sau:

Bit 15 - Time out	Bit 11 - Lỗi khung truyền
Bit 14 - Thanh ghi dịch phát rỗng	Bit 10 - Lỗi chẵn lẻ
Bit 13 - Thanh ghi giữ kí tự phát rỗng	Bit 9 - Lỗi chạy quá chạy chương trình
Bit 12 - Phát hiện điểm dừng	Bit 8 - Số liệu sẵn sàng

Chương trình sau dùng combios ( ) để tạo nên một chương trình phát/thu. Để kiểm tra, hoặc là nối TxD với RxD của một máy tính với nhau hoặc nối giữa hai máy tính với nhau. Các đường RTS và CTS tại hai máy tính được nối với nhau như hình 8.9.

Chương trình 8.2 Phát / thu qua cổng COM

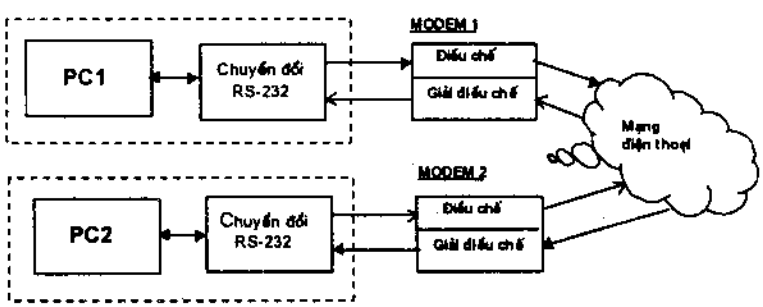
```
# include <dos.h>
# include <stdio.h>
# include <conio.h>
# define    COM1          0
# define    COM2          1
# define    Data_ready    0x100
# define    DataBit7 0x02
# define    DataBit8 0x03
# define    StopBit1 0x00
# define    StopBit2 0x04
# define    Noparity 0x00
# define    OddParity     0x08
# define    EvenParity    0x18
# define    Baud110       0x00
# define    Baud150       0x20
# define    Baud 300      0x40
# define    Baud 600      0x60
# define    Baud 1200     0x80
# define    Baud 2400     0xA0
```

```
# define    Baud 4800       0xC0
# define    Baud 9600       0xE0
# define    ESC             0x1B
int         main (void)
{
int         RS232_setting,status,in,ch;
    RS232_setting = Baud2400 | StopBit1 | NoParity | DataBit7;
    bioscom (0, RS232_setting, COM1);
    puts (" RS-232 COMBIOS  nhấn ESC để thoát ");
    do
    {
            status = bioscom (3, 0, COM1);
            if (kbhit ())
            {
                    ch = getch ( );
                    bioscom (1, ch, COM1);   /* truyền kí tự */
            }
                    if (status & DataReady)
                    if (( in = bioscom (2, 0, COM1)  &  0x7F  !=0  /* nhận kí tự */
                            putch (in);
    } while (ch!=ESC);
                    return 0;
}
```

8.3.5. Modem

Modem là tên viết tắt của thiết bị thông tin số liệu DCE xuất phát từ cụm từ tiếng Anh "MOdulation - DEModulation". Modem cho phép nối một thiết bị số (như máy vi tính) với một kênh thông tin có dải truyền hẹp (thí dụ kênh điện thoại có dải truyền giữa 400Hz và 3,4 kHz). Các Modem hiện đại còn hỗ trợ cả việc truyền và nhận các bản Fax. Với sự phổ biến của các dịch vụ trực tuyến (online) như Internet thì Modem là một thiết bị không thể thiếu được cho các máy vi tính hiện nay. Hiện có hai loại Modem: loại *lắp ngoài* được nối với PC qua các cổng nối tiếp COM và loại *lắp trong* là bản mạch ghép nối cắm vào các khe cắm mở rộng của PC. Nhìn chung cả hai loại đều như nhau nhưng do đây là một hệ thống truyền thông tin nối tiếp nên trên bản mạch Modem *lắp trong* rõ ràng phải có một chip UART gắn lên đó làm nhiệm vụ thay cho chip UART trong PC tại cổng COM.

Một hệ thông tin khi sử dụng Modem có thể được trình bày đơn giản như hình 8.15.



Hình 8.15 Hệ thông tin dùng Modem.

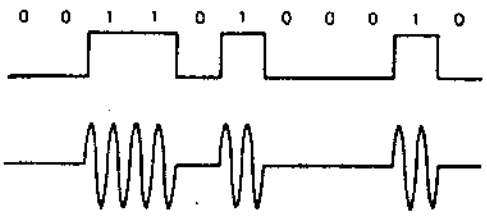
Modem phải có khả năng thực hiện được những việc sau:

- Tự động quay số kết nối với một Modem khác theo kiểu xung hay kiểu nhấn phím.
- Tự động trả lời các cuộc gọi và kết nối với Modem khác.
- Ngắt đường kết nối nếu số liệu đã được truyền xong hoặc khi có lỗi trong hệ.
- Biến đổi các bit thông tin thành các dạng thích hợp với đường truyền (điều chế).
- Biến đổi tín hiệu nhận được thành các bit thông tin (giải điều chế).
- Thực hiện thông tin có hoặc không có móc nối.

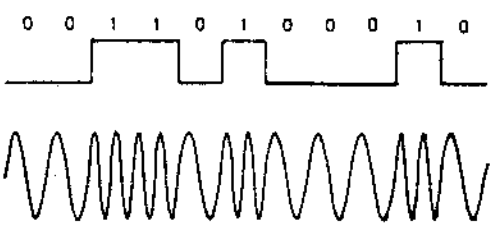
Có loại Modem đồng bộ hoặc không đồng bộ. Modem đồng bộ hồi phục lại tín hiệu đồng bộ khi thu và không cần các tín hiệu START cũng như STOP. Modem không đồng bộ là loại khá phổ biến. Tốc độ của Modem được đánh giá bằng Baud hay bps.

a, Điều chế tín hiệu trong Modem

Tốc độ cực đại mà một âm có thể được điều chế là bằng một nửa dải thông của đường truyền điện thoại. Thí dụ, dải thông của đường điện thoại hai dây là 2400Hz thì tốc độ điều chế cho truyền bán song công chỉ tối đa là 1200bps. Do vậy phải dùng các kĩ thuật để khắc phục hạn chế này. Có 3 dạng điều chế chính là: điều chế biên độ ASK, điều chế tần số FSK và điều chế pha FSK. Hình 8.16 là xung điều chế biên độ cho dãy bit thông tin. Loại điều chế này chỉ được dùng cho các kênh truyền có tốc độ rất thấp và tính chống nhiễu kém. Hình 8.17 là giản đồ xung cho điều chế tần số. Ở đây dùng một tần số đại diện cho mức lôgic 0 và một tần số khác cho mức lôgic 1. Để thông tin song công thì cần phải dùng đến 4 tần số. Thí dụ, theo tiêu chuẩn của Bell 103A - tốc độ 300Bd FSK dùng tần số 2025Hz cho mức 0 và 2225 Hz cho mức 1 theo một hướng còn 1070Hz cho mức 0 và 1270Hz cho mức 1 tại hướng kia. với các tốc độ cần cao hơn phải dùng điều chế pha. Dạng đơn giản nhất của điều chế pha là điều chế dịch khoá pha vi sai DPSK. Pha của một sóng mang hình sin có tần số không đổi cỡ 1700Hz được dịch đi 180° để biểu diễn cho một sự thay đổi số liệu từ 1 tới 0 hoặc từ 0 tới 1.

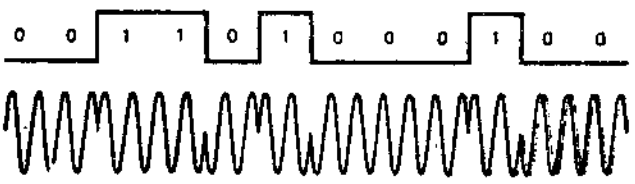


Hình 8.16 Điều chế biên độ.



Hình 8.17 Điều chế tần số.

Hình 8.18 chỉ ra một dạng sóng điều chế như vậy. Kiểu này không có ưu điểm gì hơn so với điều chế tần số về mặt tốc độ bit. Khi sử dụng thêm các góc pha bên cạnh 180° thì hai hoặc nhiều hơn bit số liệu có thể được gửi đi với một thay đổi về pha. Hình 8.19a và b chỉ ra giá trị 2 hoặc 3 bit có thể được đại diện bởi 4 hoặc 8 dịch pha khác nhau. Thí dụ, giá trị *dibit* (2 bit) là 00, pha của sóng mang sẽ được dịch đi 90° để biểu



Hình 8.18 Điều chế pha vi sai.

diễn cho 2 bit đó. Vấn đề ở đây là pha của sóng mang chỉ phải dịch đi một lần cho mỗi nhóm 2 bit được truyền.

Mã Gray dibit	Các góc độ dịch pha
00	0
01	90
11	180
10	270

Hình 8.19a

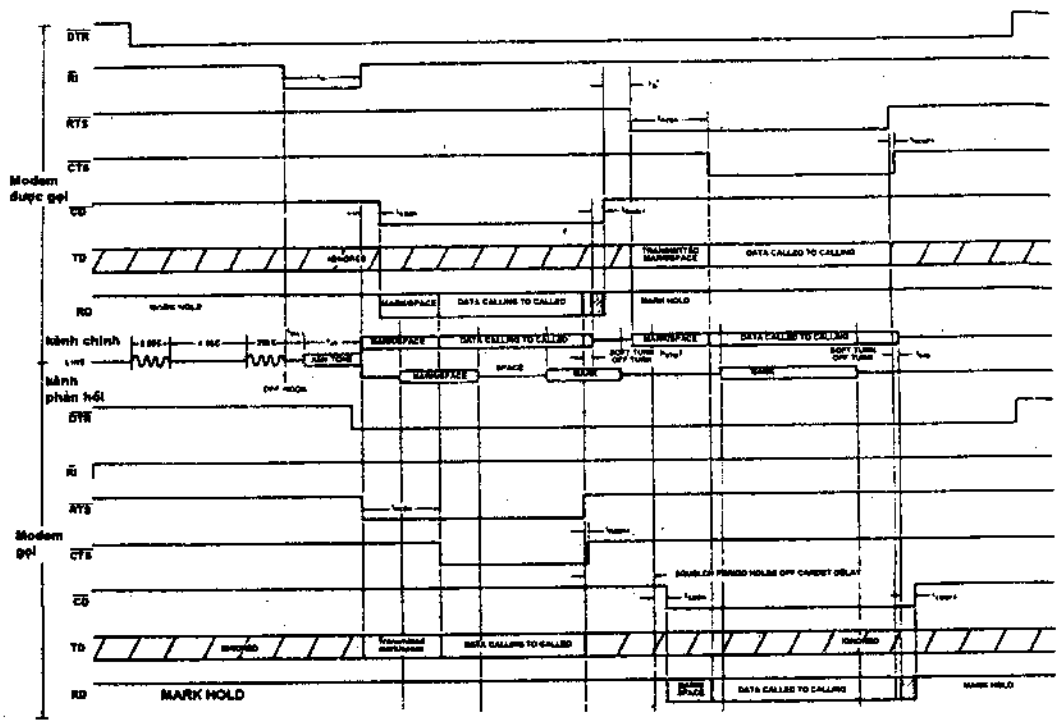
Mã Gray Tribit	Các góc dịch pha
001	22,5
000	67,5
010	112,.5
011	157,5
111	202,5
110	247,5
100	292,5
101	337,5

Hình 8.19b

Cũng có những sơ đồ điều chế pha phức tạp hơn là QAM (Quaternary amplitude modulation) cho phép truyền số liệu song công ở tốc độ 2400bps tại đường điện thoại 2 dây. Với truyền bán song công có thể lên tới 9600bps. QAM sử dụng 12 góc pha khác nhau và 3 biên độ khác nhau để mã hoá 4 bit số liệu trong mỗi thay đổi điều chế. Mỗi nhóm 4 bit đó được gọi là *quadbit*. Còn có những kĩ thuật điều chế khác như DAMQAM (Dynamically Adaptive Multicarrier Quadrature Amplitude Modulation) dùng tới 512 tần số mang khác nhau trên đường dây điện thoại cho phép tăng tốc độ truyền đến 19 200 bps.

b, Quá trình móc nối thông tin trong Modem

Hình 8.20 chỉ ra một thí dụ về số liệu và các dạng sóng cho một Modem đơn giản được lắp trên chip AM7910; các Modem khác có thể dùng các tín hiệu khác một chút nhưng về nguyên tắc là như nhau.



Hình 8.20 Các tín hiệu móc nối của Modem.

Hầu hết các Modem hiện nay đều có một vi xử lí chuyên dụng trong đó. Chỉ những Modem thông minh như vậy mới thoả mãn các nhiệm vụ đã nêu ở trên. Sau khi một Modem quay số gọi một Modem khác thì một loạt các tín hiệu móc nối phải được diễn ra. Phần bên trái của đồ thị sóng biểu diễn một cú gọi đang được tiến hành từ một Modem này tới một Modem khác. Giả thiết rằng tín hiệu $\overline{DTR}$ của *Modem được gọi* được xác nhận, tín hiệu chuông trên đường dây sẽ làm mạch ghép nối với đường điện thoại xác nhận *lối vào chuông* $\overline{RI}$ của 7910. Để đáp ứng với điều này, 7910 sẽ gửi ra một chu kì im lặng chừng 2 giây để dàn xếp với các tín hiệu billing rồi gửi ra một âm trả lời với tần số 2025Hz tới *Modem gọi* trong 2 giây. Nếu $\overline{DTR}$ và $\overline{RTS}$ của *Modem gọi* được xác nhận (tức là chỉ thị rằng số liệu đã sẵn sàng được gửi đi) thì Modem gọi sẽ đưa âm 2225Hz (mark) lên đường truyền trong 8 mili giây để cho *Mode được gọi* biết rằng kết nối đã hoàn thành. Để đáp lại với tín hiệu mark này, *Modem được gọi* phát tín hiệu ra *carrier-detect* $\overline{CD}$ để cho phép UART nhận. *Modem gọi* khi đó truyền số liệu cho tới khi lối vào $\overline{RTS}$ của nó được giải phóng khỏi PC hoặc hết số liệu. Trong khi đang nhận số liệu trên kênh chính, *Modem được gọi* có thể gửi số liệu tới *Modem gọi* trên 5-bit/s kênh phản hồi. Giả phóng tín hiệu $\overline{RTS}$ làm Modem giải phóng tín hiệu $\overline{CTS}$ tới *máy tính gọi* và lấy đi sóng mang khỏi đường truyền. *Modem được gọi* nhạy với việc mất sóng mang và không phát tín hiệu xác nhận carrier detect $\overline{CD}$.

Bây giờ nếu như hệ thống được gọi phải gửi một vài số liệu quay về hệ thống gọi trên kênh chính, nó sẽ đưa tín hiệu vào $\overline{RTS}$ tới Modem của mình. Modem được gọi gửi một âm đánh dấu tới Modem gọi trong 8 mili giây. Modem gọi gửi tín hiệu ra $\overline{CD}$ của nó tới UART của nó. Modem được gọi khi ấy gửi số liệu tới Modem gọi trên kênh chính cho tới khi lối vào $\overline{RTS}$ của nó không được xác nhận bởi hệ thống được gọi, điều này chỉ thị rằng không còn số liệu được gửi đi nữa. Trong khi Modem được gọi đang phát tín hiệu trên kênh chính, Modem gọi có thể phát tín hiệu trên kênh phản hồi nếu cần thiết. Việc móc nối tương tự như cho các hệ song công nhưng tốc độ truyền số liệu là tương đương trong cả hai hướng truyền.

c, Các chuẩn Modem

CCITT đã định nghĩa các chuẩn liên quan đến RS-232 và các thông tin modem. Mỗi chuẩn đều dùng chữ đầu là V để phân biệt loại này như bảng 8.3 sau. Hầu hết các modem hiện nay đều được chế tạo theo các chuẩn và tương thích với Heyes, đó là một công ti đầu tiên đi tiên phong trong lĩnh vực modem và xác định một phuương pháp lập trình cho modem với ngôn ngữ lệnh AT. Đó là tập lệnh mà hầu hết đều bắt đầu bằng hai chữ AT.

Bảng 8.3

CCITT	Mô tả
V.21	Song công, tốc độ phát 300bps
V.22	Bán song công, tốc độ phát 600bps và 1200bps
V.22bits	Song công, tốc độ phát 1200bps và 2400bps
V.23	Song công, tốc độ phát 1200bps và thu 75bps
V.24	Chuẩn CCITT cho ghép nối RS-232
V.32	Song công, phát ở 4800bps và 9600bps
V.25bits	Ngôn ngữ lệnh Modem
V.32bits	Song công, phát ở 7200bps, 12000bps và 14400bps
V.42	Giao thức điều khiển lỗi

Modem có thể ở một trong hai trạng thái: trạng thái bình thường và trạng thái lệnh Trong trạng thái bình thường, Modem phát và/hoặc nhận các kí tự từ máy tính. Trong trạng

thái lệnh, các kí tự gửi tới Modem được dịch ra thành lệnh. Mỗi khi lệnh được dịch xong, Modem tự động quay về trạng thái bình thường. Để ngắt Modem nhằm làm cho nó quay về mode lệnh phải gửi lệnh 3 dấu cộng là '+++'. Modem có các thanh ghi trạng thái gọi là các thanh ghi S lưu trữ các thông số đặt. Bảng 8.4 liệt kê một vài thanh ghi này.

Bảng 8.4

Thanh ghi	Chức năng	Dải (mặc định điển hình)
S0	Chuông để tự động trả lời	0-255 rings (0 rings)
S1	Đếm chuông	0-255 rings (0 rings)
S2	Kí tự Escape	(43)
S3	Kí tự Cariage Return	(13)
S6	Đợi cho âm mời quay số	2 - 255 s (2 s)
S7	Đợi sóng mang	1 - 255s (50 s)
S8	Tạm ngưng để quay số tự động	0 - 255 s (2 s)
S32	Kí tự X-ON	0 - 255 (11h)
S33	Kí tự X-OFF	0 - 255 (13h)

Máy tính gây sự chú ý của Modem khi gửi một lệnh "AT". thí dụ 'ATDT' là lệnh quay số gọi. Đầu tiên Modem vào mode lệnh và nhận lệnh từ máy tính. Các lệnh này được gửi với tốc độ 300bps hoặc 1200bps (Modem sẽ tự động tìm tốc độ đang được dùng). Hầu hết các lệnh được gửi đi với tiền tố là AT. Mỗi lệnh được theo sau bởi kí tự carriage return (mã ASCII là 13d). Một lệnh không có kí tự này sẽ bị bỏ qua. Nhiều lệnh có thể được đặt trên cùng một dòng và nếu cần thiết xen giữa các lệnh dấu cách (space). Các lệnh không phân biệt chữ thường hay chữ hoa. Bảng 8.5 dưới đây cho một vài thí dụ lệnh AT.

Bảng 8.5

Lệnh	Mô tả
ATDT12345	Quay gọi số điện thoại 12345 bằng âm nhấn phím (touch-tone)
ATPT12345	Quay số gọi điện thoại 12345 bằng điện thoại quay số (pulse-tone)
AT S0=2	Tự động trả lời cú gọi. Thanh ghi S0 chứa số gọi chuông mà modem sử dụng trước khi nó trả lời cuộc gọi. Trong trường hợp này có 2 chuông trước khi nó trả lời. Nếu S0 bằng 0 thì modem sẽ không trả lời cú gọi.
ATH	Treo đường dây (tương đương với nhấc máy điện thoại ra ngoài)
+++	Tách khỏi đường dây điện thoại và nhảy vào mode lệnh

Hầu hết các modem đều có bộ nhớ NVRAM được chương trình hoá bằng điện. Điều này cho phép modem tự động bật máy. Sau khi modem nhận được lệnh AT, nó đáp lại với một mã trả về như các thí dụ trong bảng 8.6. Thí dụ, nếu một modem gọi một modem khác mà modem đó lại đang bận thì mã số 7 sẽ được trả về.

Bảng 8.6

Thông báo	Digit	Mô tả
OK	0	Lệnh chạy không có lỗi
CONNECT	1	Một kết nối đã được thực hiện
RING	2	Phát hiện một cú gọi đến
NO CARRIER	3	Không có sóng mang
ERROR	4	Lệnh bất hợp lệ
CONNECT 1200	5	Nối với modem 1200bps

NO DIALTONE	6	Không có âm mời quay số
BUSY	7	Đường dây xa bận
NO ANSWER	8	Không có trả lời từ đường dây xa
CONNECT 600	9	Nối với modem 600bps
CONNECT 2400	10	Nối với modem 2400bps
CONNECT 4800	11	Nối với modem 4800bps

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

8.1 Cổng máy in song song trong phòng thí nghiệm đã nối sẵn 8 bộ chỉ thị lôgic cho 8 bit dữ liệu ra. Hãy viết và cho chạy một chương trình thoả mãn các yêu cầu sau:

- Ra các số từ 00h đến FFh tới cổng máy in với các khoảng thời gian trễ giữa 2 số là 0,5 giây.
- Tạo một đèn sáng chạy vòng trên 8 bộ chỉ thị.

8.2 Nhập và cho chạy chương trình sau, dùng ngắt trong BIOS để kiểm tra trạng thái của máy in và in một kí tự ra máy in.

Θ Chương trình 8.3

```
# include <dos.h>
# include <stdio.h>
# include <conio.h>
# define  PRINTERR  -1

void  print_character (int  ch);
int    init_printer (void);

int  main (void)
{
int  status, ch;
          status = int_printer ( );
          if (status == PRINTERR) return (1);
do
{
  printf(" nhập kí tự để in ra máy in ");
 ch = getch ( );
print_character (ch);
} while (ch !=4);
          return (0);
}
int       init_printer (void)
{
union REGS  inregs, outregs;
          inregs.h.ah = 0x01;          /* khởi động máy in */
          inregs.x.dx = 0;   /* LPT1 */
          int86 (0x17,&inregs, &outregs);
```

```
        if (inregs.h.ah & 0x20)
                {puts ("hết giấy"); return (PRINTERR); }
        else  if (inregs.h.ah & 0x08)
                {puts ("lỗi vào / ra");  return (PRINTERR); }
        else  if (inregs.h.ah & 0x01);
                {puts("Quá thời gian - Timer Out ");
        return (PRINTERR):  }
return (0);
}
void    print_character (int  ch)
{
        Union REGS  inregs,outregs;
                inregs.h.ah = 0x00;          /* In ký tự  */
                inregs.x.dx = 0;             /* LPT1 */
                inregs.h.al = ch;
                int86 (0x17,  &inregs, &outregs);
}
```

8.3 Viết một chương trình hiển thị trạng thái máy in, mô phỏng các hỏng hóc máy in và kiểm tra chương trình khi không có giấy in, cáp nối bị đứt (rút ra khỏi đầu cắm) v.v..

8.4 Xác định tốc độ cực đại khi gửi các kí tự ra một máy in không ghép nối vào mạng.

8.5 Viết một chương trình đếm sự xuất hiện của chữ 'a' trong file văn bản. Sử dụng hoặc là ch=fgetc ( ) hoặc fscan (in, "%c",&ch) để đọc các kí tự tách biệt trong file.

8.6 Nhập và cho chạy chương trình truyền một kí tự từ COM1 và nhận nó qua chính cổng này (đầu TxD được nối trực tiếp với đầu RxD).

- Viết trên Turbo C:
- Viết trên Turbo Pascal:

8.7 Nhập và cho chạy hai chương trình phát và thu được dùng để truyền số liệu từ PC này tới một PC khác qua các cổng COM như dưới đây. Nối cáp truyền TxD của máy PC phát với RxD của máy thu và ngược lại.

⊖ Chương trình 8.4

```
/* Chương trình phát */
# define Txdata          0x3F8
# define LSR             0x3FD
# define LCR             0x3FB
# include <stdio.h>
# include <conio.h>
# include <dos.h>
        void    setup_serial (void)              /* khởi tạo cổng nối tiếp */
        void    send_character (int  ch)         /* phát kí tự */
        int     main (void)
{
        int  ch;
        puts("chuong trinh phat. De nghi nhap kí tu (Ctrl-D : ket thuc)");
```

```
        setup_serial ( );
        do
        {
        ch=getche ();
        send_character (ch);
        }       while (ch!=4);
                return (0);
}
        void    setup_serial (void)
{
        outportb(LCR,0x80);             /* đặt bit 7 lên 1 để đặt bit địa chỉ thanh ghi */
        outportb(Txdata,0x0C);
        outportb(Txdata+1,0x00);        /* nạp thanh ghi số liệu số 12, tần số thạch anh
                                           bằng 1,8432 MHz  */
        outportb(LCR,0x0A);             /* 0Ah = 0000 1010 : 7 bit số liệu, 1 bit stop, 1 bit
                                                   chẵn lẻ-lẻ  */
}
        void    send_character (int  ch);
{
        char   status;
        do
        {
                status = inportb(LSR) & 0x40;
        }while (status !=0x40);          /* đợi đến khi bit 6 được đặt - bộ đệm Tx rỗng*/
                outportb(Txdata, (char)  ch);
    }
```

⊖ Chương trình 8.5

```
/* Chương trình thu */
# define Txdata          0x3F8
# define LSR             0x3FD
# define LCR             0x3FB
# include <stdio.h>
# include <conio.h>
# include <dos.h>
void    setup_serial (void)          /* khởi tạo cổng nối tiếp */
int     get_character (void)         /* nhận kí tự */
int     main (void)
{
int     inchar;
setup_serial ( );
do
{
        inchar = get_character ( );
        putchar (inchar);
```

```
    }       while (inchar!=4);
            return (0);
}
    void    setup_serial (void)
{
    outportb(LCR,0x80);             /* đặt bit 7 lên 1 để đặt bit địa chỉ thanh ghi */
    outportb(Txdata,0x0C);
    outportb(Txdata+1,0x00);        /* nạp thanh ghi số liệu số 12, tần số thạch anh
                                       bằng 1,8432 MHz  */
    outportb(LCR,0x0A);             /* 0Ah = 0000 1010 : 7 bit số liệu, 1 bit stop, 1 bit
                                       chẵn lẻ-lẻ  */
}
    int   get_character (void);
{
    int  status;
            do
            {
                    status = inportb(LSR) & 0x01;
            } while (status !=0x01);   / * đợi đến khi bit 1 */
    return ( (int) inportb(Txdata));
}
```

8.8 Viết một chương trình liên tục gửi các kí tự từ "A" đến "Z" trên đường dây nối tiếp. Quan sát lối ra trên một dao động kí.

8.9 Điền thời gian thực để truyền 1000 kí tự với các tốc độ Baud đã cho trong bảng dưới đây. So sánh với giá trị tính toán theo công thức.

Tốc độ Baud	Thời gian gửi 1000 kí tự [giây]
110	
300	
600	
1200	
2400	
4800	
9600	
19200	

8.10 Chương trình trong bài thực hành 8.7 có vấn đề là khi nhấn phím ENTER thì chỉ có 1 kí tự được gửi. Kí tự nhận được sẽ là một mã CR (carriage return) nó buộc con trỏ quay về đầu dòng mà không đi tiếp xuống dòng dưới. Thay đổi /* chương trình thu */ sao cho một dòng mới xuất hiện LF (line feed) mỗi khi phím ENTER được nhấn. Lưu ý rằng mã ASCII của kí tự CR là 13 và của LF là 10.

8.11 Thay đổi chương trình nhận kí tự *get_character ( )* sao cho nó sẽ trả về một cờ báo lỗi nếu nó phát hiện ra một lỗi hoặc nếu quá thời gian (time out). Bảng sau liệt kê các cờ lỗi

và trả về giá trị lỗi. Chương trình sau chỉ ra các lỗi, nếu một kí tự không nhận được trong vòng 10 giây thì một báo lỗi sẽ xuất hiện.

Θ Chương trình 8.6

```
# include <stdio.h>

#include <dos.h>
# define TxData          0x3F8
# define LSR             0x3FD
# define LCR             0x3FB
void     show_error (int ch);
int      get_character (void);
enum     RS232_error     (PARITY_ERROR = -1; OVERRUN_ERROR = -2,
         FRAMING_ERROR = -3, BREAK_DETECTED = -4, TIME_OUT = -5);
int      main (void)
{
  int    inchar;
         do
         {
                  inchar = get_character ( );
                  if (inchar < 0) show_error (inchar);
                  else printf("&c",inchar);
         } while (inchar !=4);
         return (0);
}
 void    show_error(int ch)
 {
         switch(ch)
         {
         case PARITY_ERROR: printf("Error: Parity error/n"); break;
         case OVERRUN_ERROR: printf("Error: Overrun error/n"); break;
         case FRAMING_ERROR: printf("Error: Framing error/n"); break;
         case BREAK_DETECTED: printf("Error: Break detected error/n"); break;
         case TIME_OUT: printf("Error: Time out /n"); break;
         }
 }
 int     get_character(void)
 {
 int instatus;
         do
         {
                  instatus = inportb(LSR) & 0x01;
                  if (instatus & 0x02) return (BREAK_DETECTED);
                                                    ::: etc
         } while (instatus != 0x01);
         return ( (int) inportb(TXDATA);
```

}

8.12 Thiết kế và cho hoạt động một kết nối bán song công (half-duplex) giữa hai máy tính. *Gợi ý*: cùng một chương trình sẽ chạy trong cả hai máy tính nhưng một sẽ ở chế độ phát khi một phím trên bàn phím được nhấn và máy kia ở chế độ thu. Khi máy phát truyền kí tự có mã ASCII là 04 (^D) hai máy sẽ trao hai chế độ.

8.13 Thiết kế và cho chạy hoạt động một kết nối mô phỏng song công (full-duplex) giữa hai máy tính (tức là cả hai đều có thể cùng nói và nghe). Gợi ý: cho chạy cùng một chương trình trong cả hai máy tính. Tạo một vòng lặp trong chương trình phụ *get_character ( )* và thoát ra khỏi nó nếu một kí tự được nhận từ một đường hoặc nếu một kí tự được nhập vào từ bàn phím. Nếu một kí tự được nhập vào từ bàn phím thì nó cần phải được gửi đi và chương trình khi đó sẽ quay về chương trình phụ *get_character ()*.

8.14 Xây dựng kết nối giữa hai máy tính nhằm một máy truyền các kí tự tới máy kia và hiện lên màn hình kí tự đó. Nếu phím cách (space bar) được nhấn trên máy thu thì nó gửi kí tự X-OFF đến máy phát. Máy phát khi đó sẽ hiện thông báo là máy thu đang bận. Khi phím cách được nhấn lần nữa, máy thu sẽ truyền kí tự X-ON và máy phát được truyền tiếp. Do vậy khi sử dụng phím cách, máy thu sẽ chuyển giữa hai trạng thái bận và rỗi.

8.15 Thiết kế và phát triển một chương trình truyền một file giữa hai máy tính. Tên của file cần được gửi theo sau kí tự NULL. Tiếp theo nội dung của file được gửi và cuối cùng một kí tự kết thúc file là EOF.

8.16 Giải thích tại sao các hệ thống dùng chuẩn RS-422A hoặc RS-423A lại có thể truyền số liệu trên một khoảng cách xa hơn và tốc độ baud lớn hơn so với hệ thống dùng chuẩn RS-232C ?

8.17 Giải thích một DTE (máy tính) gây sự chú ý của modem như thế nào ? Cũng giải thích cách mà DTE buộc mode nhảy vào mode lệnh khi nó đang ở mode bình thường.

8.18 Định nghĩa thuật ngữ Modem và giải thích tại sao một Modem lại đòi hỏi gửi tín hiệu số trên đường điện thoại.

8.19 Mô tả chức năng của các tín hiệu $\overline{DSR}$, $\overline{DTR}$, $\overline{RTS}$, $\overline{CTS}$, *TxD* và *RxD* được trao đổi giữa một thiết bị đầu cuối và một Modem.

8.20 Nối modem với máy tính và quay số gọi modem từ xa.

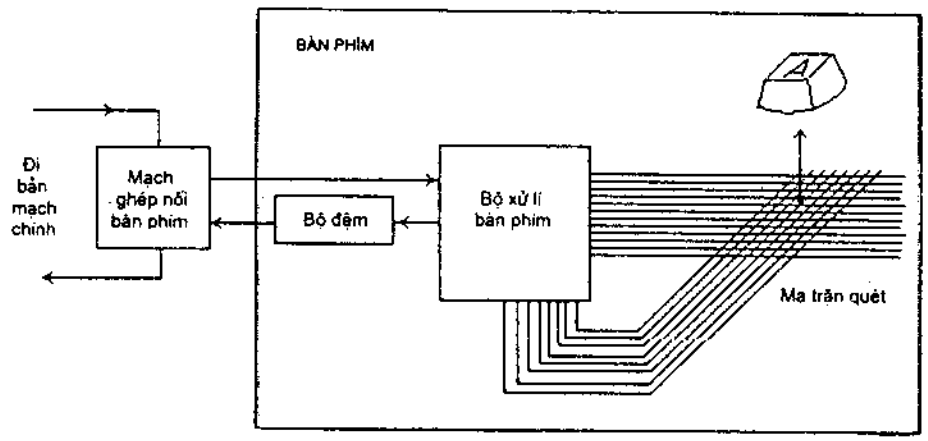
8.21 Nối hai modem với nhau theo mode đường dây riêng (lease line) và dùng chương trình như Terminal truyền các xâu kí tự từ máy tính này sang máy tính kia.

CHƯƠNG 9. GHÉP NỐI BÀN PHÍM VÀ CHUỘT

9.1. BÀN PHÍM

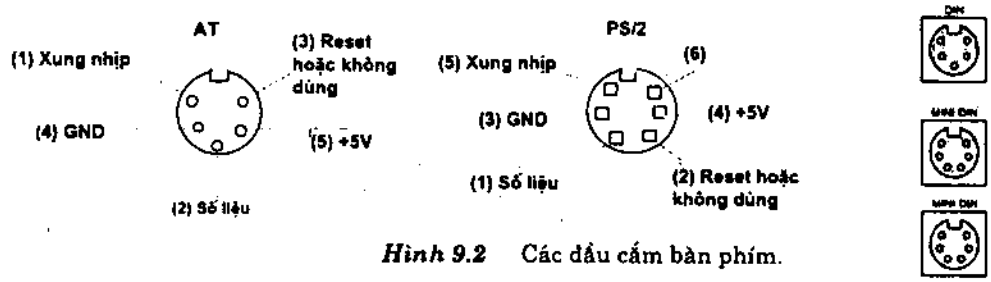
9.1.1. Hoạt động của bàn phím

Bàn phím có tác dụng quan trọng cho việc nhập các số liệu và chương trình vào PC. Nó có một vi xử lí riêng như chip 8042 cho PC/XT và 8048 cho máy AT. Một ma trận quét (bao gồm các đường dây dẫn dọc và ngang) được dẫn tới chip này. Ở các điểm vắt chéo được đặt các công tắc mà trên đó là các phím nhấn như hình 9.1. Nếu một phím được nhấn, công tắc sẽ đóng mạch tiếp xúc điện giữa các đường vắt chéo nhau tương ứng và mạch xử lí của bàn phím sẽ xác định phím nào được nhấn. Thông tin toạ độ của phím đó sẽ được truyền qua bộ đệm tới mạch ghép nối bàn phím trên bản mạch chính về CPU.



Hình 9.1 Ma trận quét bàn phím.

Mỗi khi một phím được nhấn hay nhả, một mã 8 bit gọi là *mã quét* (scan code) được phát chuyển về PC. Khi phím được nhấn, mã này được gọi là *make-code* nằm trong dải từ 01h đến E0h. Khi phím được nhả, mã này được cộng thêm 80h (128_{10}) và gọi là *break-code*. Thí dụ, khi nhấn phím Z bàn phím phát ra mã quét là 2Ch, khi nhả nó sẽ phát ra mã ACh (bằng 2Ch + 80h). Trong các máy từ AT trở về sau, break-code lại gồm 2 byte, byte đầu tiên là mã F0h và byte sau là bằng make-code. Do đó break-code cho phím Z ở đó sẽ là F0h và 2Ch. Các mã này được truyền từng byte một theo thủ tục truyền nối tiếp đồng bộ theo 2 kênh truyền hai hướng. Kênh số liệu truyền các bit dữ liệu nối tiếp và kênh đồng bộ truyền các xung nhịp đánh dấu đầu các bit dữ liệu. Một khung truyền gồm 1 bit start, 8 bit dữ liệu với LSB trước MSB sau, 1 bit chẵn lẻ (thường là chẵn lẻ-lẻ) và 1 bit stop. Hình 9.2 là các sơ đồ đầu cắm đực của bàn phím AT và PS/2, số trong ngoặc là số chân cắm trên đầu nối vào bản mạch chính.



Hình 9.2 Các đầu cắm bàn phím.

Việc thâm nhập bàn phím có thể được thực hiện trực tiếp qua các cổng vào/ra hoặc qua các ngắt trong BIOS.

9.1.2. Thâm nhập trực tiếp bàn phím qua các cổng vào/ra

Bàn phím cũng là một thiết bị ngoại vi nên về nguyên tắc có thể thâm nhập nó qua các cổng vào/ra. Kể từ máy AT trở đi với bàn phím MFII điều này có thể được thực hiện qua các cổng có địa chỉ 60h và 64h. Sử dụng hai địa chỉ này có thể thâm nhập các bộ đệm vào/ra, thanh ghi trạng thái và thanh ghi điều khiển của bàn phím. Các vùng đệm dùng để truyền các mã bàn phím từ bàn phím về PC và các lệnh mà PC truyền tới bàn phím. các lệnh được truyền qua cổng 64h còn dữ liệu vào PC hoặc ra bàn phím phải truyền qua cổng 60h. Thanh ghi trạng thái sẽ cho biết byte được truyền là lệnh (cổng 64h) hay dữ liệu (cổng 60h).

Cổng	Thanh ghi	Đọc(R),Viết (W)
60h	Đệm lối ra	R
60h	Đệm lối vào	W
64h	Thanh ghi điều khiển	W
64h	Thanh ghi trạng thái	R

Thanh ghi trạng thái xác định trạng thái hiện tại của *bộ điều khiển bàn phím*. Thanh ghi này là chỉ đọc (read only) và định dạng từ trạng thái như sau:

7							0
PARE	TIM	AUXB	KEYL	C/D	x	INPB	OUTB

PARE: Lỗi chẵn lẻ của byte cuối cùng được vào từ bàn phím, 1: có lỗi chẵn lẻ 0: không có
TIM: Lỗi quá thời gian (time - out), 1: có lỗi 0: không có
AUXB: Đệm ra cho thiết bị phụ (chỉ ở máy PS/2), 1: giữ số liệu cho thiết bị; 0: giữ số liệu cho bàn phím
KEYL: Trạng thái khoá bàn phím, 1: không khoá 0: khoá
C/D: Lệnh/số liệu, 1: viết qua cổng 64h; 0: viết qua cổng 60h
INPB: Trạng thái đệm vào, 1: số liệu CPU trong bộ đệm vào; 0: đệm vào rỗng
OUTB: Trạng thái đệm ra, 1: số liệu bộ điều khiển bàn phím trong bộ đệm ra; 0: đệm ra rỗng

Thanh ghi điều khiển ở địa chỉ 064h là loại chỉ viết, mã lệnh của từ điều khiển như sau.

D7	D6	D5	D4	D3	D2	D1	D0

Mã	Lệnh
A7h	Cấm thiết bị phụ
A8h	Cho phép thiết bị phụ
A9h	Kiểm tra ghép nối tới thiết bị phụ
AAh	Tự kiểm tra
ABh	Kiểm tra ghép nối bàn phím
ADh	Cấm bàn phím
AEh	Cho phép bàn phím
C0h	Đọc cổng vào
C1h	Đọc cổng vào ra (byte thấp)
C2h	Đọc cổng vào ra (byte cao)
D0h	Đọc cổng ra
D1h	Viết cổng ra
D2h	Viết đệm ra bàn phím
D3h	Viết đệm ra thiết bị phụ
D4h	Viết thiết bị phụ

E0h	Kiểm tra đọc cổng vào
F0h	Gửi một xung tới lối ra
FFh	Cổng.

Máy tính có thể ra các lệnh điều khiển bàn phím qua bộ đệm ra có địa chỉ 60h. Dưới đây là tóm tắt các lệnh đó. Nhớ rằng đây là địa chỉ của cổng B chip PPI-8255A trong đó 2 bit thấp nhất

Mã	Lệnh	Mô tả
EDh	Bật ON/OFF LEDs	Bật / tắt các đèn LED của bàn phím.
EEh	Echo	Trả về byte eeh.
F0h	Đặt / nhận diện	Đặt một trong ba mã quét và nhận diện các mã quét tập mã quét hiện tại.
F2h	Nhận diện bàn phím	Nhận diện ACK = AT, ACK+abh +41h = MFII .
F3h	Đặt tốc độ lặp lại/trễ	Đặt tốc độ lặp lại và thời gian trễ của bàn phím.
F4h	Enable	Cho phép bàn phím hoạt động.
F5h	Chuẩn/không cho phép	Đặt giá trị chuẩn và cấm bàn phím.
F6h	Chuẩn/cho phép	Đặt giá trị chuẩn và cho phép bàn phím.
FEh	Resend	Bàn phím truyền kí tự cuối cùng một lần nữa tới điều khiển bàn phím.
FFh	Reset	Chạy reset bên trong bàn phím.

Lệnh đặt tốc độ bàn phím và bật / tắt các đèn LED trên bàn phím là được dùng nhiều cho các chương trình ứng dụng. Lệnh thứ nhất liên quan đến tần số lặp nhấn phím (typematic rate) và thời gian trễ để lặp nhấn phím (delay rate). Đó là lượng các make-code mà bàn phím gửi cho PC trong một giây khi một phím được nhấn liên tục. Tốc độ này có thể thay đổi được từ 2 đến 30 lần/giây. Để tránh lặp nhấn phím , phím chỉ được lặp sau một thời gian trễ nào đó. Thời gian này được mã hoá như sau:

Mã	Tốc độ trễ (giây)
00	1/4
01	2/4
10	3/4
11	1

Thời gian trễ này có sai số cỡ 20%. Tốc độ lặp RpS (số lần lặp trong một giây) được mã hoá theo công thức sau:

(8+A)*2B*0,00417 *1 / giây

Trong đó A là giá trị nhị phân trong các bit 0, 1 và 2 còn biến B là giá trị nhị phân trong các bit 3 và 4. Thời gian trễ và tốc độ lặp được đặt trong byte tốc độ bàn phím; thời gian trễ chiếm ba bit cao, tốc độ lọc chiếm 5 bit thấp. Trước khi truyền tốc độ tới bàn phím phải truyền mã lệnh là byte F3h. Byte này được truyền cho bàn phím qua cổng 60h. Nhưng việc truyền từng byte phải tuân theo một thủ tục kiểm tra trạng thái bàn phím qua một vài lần (thí dụ 3 lần). Cổng trạng thái bàn phím được kiểm tra bằng một vòng lặp cho đến khi vùng đệm vào của bàn phím rỗng (bit 0 bằng 0). Chỉ sau đó byte cần truyền mới được gửi tới bàn phím. Vì có thể xuất hiện lỗi chẵn lẻ trong quá trình truyền, bàn phím cần trả lại một mã trả lời là 0FAh nếu việc truyền có kết quả (acknowledge). Mọi mã khác đều là thông báo lỗi. Đôi khi để đơn giản có thể thay các thủ tục kiểm tra bằng một lệnh trễ (thí dụ, delay(100) - trễ 100 ms).

9.1.3. Thâm nhập bàn phím qua BIOS

Việc thâm nhập bàn phím có thể được thực hiện qua các ngắt của BIOS. Mạch điều khiển bàn phím trên bản mạch chính luôn kiểm tra lối vào xem liệu có một byte từ bàn phím được chuyển tới không. Nó sẽ phát ra ngắt 09h mỗi khi nhận được 1 byte dữ liệu từ bàn phím BIOS chứa một chương trình con phục vụ ngắt 09h cho phép đọc byte từ bộ điều khiển bàn phím và xử lí nó từ cổng vào/ra 60h. Chương trình ngắt này sẽ dịch mã quét thành một giá trị 2 byte. Khi một phím kí tự ASCII chuẩn được nhấn, byte thấp chứa mã ASCII, byte cao chứa mã quét. Khi một phím đặc biệt được nhấn (thí dụ, phím Num-Lock) byte thấp bằng 0 và byte cao chứa mã quét. BIOS sẽ đặt 2 byte này vào bộ đệm bàn phím. Bộ đệm này được giữ trong vùng nhớ thấp khởi phát ở địa chỉ 0040: 001Eh. Số liệu được lưu giữ ở đó cho đến khi được gọi bởi chương trình. Việc dịch các mã quét thì tương đối phức tạp vì bàn phím IBM được phân làm hai loại phím: các phím dịch (shift-keys) và các phím lật (toggele-keys). Có 3 phím dịch là Ctrl, Shift và Alt. Chúng thay đổi trạng thái dịch của phím được nhấn với chúng; thí dụ khi nhấn Shift-C có nghĩa ta đã phát ra chữ C hoa. BIOS phát hiện ra hoạt động này ngay khi phím dịch được nhấn xuống. Hai phím lật là Caps Lock và Num Lock. Khi được kích hoạt, phím Caps Lock sẽ chỉ đảo trạng thái dịch của các phím kí tự an-phá-bê (A, B, C,...) mà không ảnh hưởng tới các phím còn lại. Khi được kích hoạt phím Num Lock sẽ vô hiệu chức năng điều khiển con trỏ trên các phím số. Các phím lật được kích hoạt với một lần nhấn và giữ nguyên hoạt động đến lần nhấn thứ 2. Các phím Ins (Insert) và Scroll Lock cũng là các phím lật nhưng chúng không tác động tới trạng thái của các phím khác.

Hai byte thông tin trạng thái của các phím dịch và lật được BIOS giữ ở vùng nhớ khởi phát từ địa chỉ 0040:0017h. Khi một phím dịch hoặc phím lật được nhấn, BIOS sẽ đặt một bit đặc biệt của một trong 2 byte này lên 1. Khi BIOS nhận được mã break-code của phím dịch nó sẽ chuyển bit trạng thái trở về trạng thái dịch ban đầu của nó.

Bất cứ khi nào BIOS nhận được một mã quét của một phím bình thường (chẳng hạn như phím Z hoặc phím dịch con trỏ sang phải →), nó sẽ kiểm tra trạng thái dịch rồi chuyển thành một mã 2 byte thích hợp.

Ngoài các hoạt động kể trên BIOS còn dịch các tổ hợp phím nhấn. Có 5 hoạt động tạo cho BIOS thực hiện các nhiệm vụ đặc biệt như sau:

Alt-Sys Req yêu cầu BIOS phát ra ngắt 15h với hàm 85h. Người sử dụng có thể tạo một chương trình con phục vụ ngắt 15 của riêng mình khi các phím này được nhấn.

Ctrl-Alt-Del bắt máy tính khởi động nóng.

Shift-PrtSc (Print Screen) cho phép ngắt 09h trong BIOS gọi ngắt mềm 05h gửi nội dung hiện tại của màn hình tới máy in.

Ctrl-Num Lock (Pause) ngừng hoạt động của chương trình tới khi một phím bất kì được nhấn.

Ctrl- Break yêu cầu BIOS phát ra ngắt mềm 1Bh và đặt bit 7 của byte ở 0040: 0071 lên 1.

Mạch điện bên trong bàn phím còn kiểm tra xem mỗi một phím được nhấn trong bao lâu. Nếu một phím được nhấn dài hơn một khoảng thời gian xác định mạch sẽ thực hiện thao tác lặp lại phím. Nó cho phép ngắt 09h phân biệt được một việc nhấn phím bình thường và nhấn phím quá dài. Thí dụ nếu ta nhấn phím A đủ dài để bàn phím phát ra tín hiệu thao tác lặp lại, BIOS tạo nên. Vi mạch bàn phím cũng còn phải xử lí *khử hiện tượng rung cơ khí* của các tiếp điểm phím nhấn. Do các tiếp điểm này đóng mở nhiều lần (dù chỉ trong thời gian rất ngắn cỡ mili giây) nên nếu không có xử lí đặc biệt thì CPU sẽ ghi nhận như là phím được nhấn nhả

nhiều lần. Việc khử nhiễu rung này được thực hiện bằng phần cứng như dùng các mạch lật (thí dụ mạch lật RS) hoặc bằng phần mềm theo nguyên tắc bộ xử lí chỉ ghi nhận một giá trị nào đó sau một khoảng thời gian đặt trước dài hơn thời gian rung.

Người dùng cũng có thể nhập một mã ASCII từ 01h tới FFh bằng việc trong khi nhấn phím Alt đồng thời nhập mã kí tự ASCII từ các phím số ở phần bên phải bàn phím. Ngắt 16h cho phép người sử dụng xâm nhập bàn phím bảng sau chỉ ra chức năng của ngắt này.

Chức năng	Các thanh ghi vào	Các thanh ghi ra
Nhận một nhấn phím	AH = 00h	AH = mã quét AL = mã ASCII
Kiểm tra nhấn phím	AH = 01h	Cờ zêrô ZF=1: không có nhấn phím ZF=0: có phím được nhấn AH = mã quét AL = mã ASCII

Chương trình 9.1 cho phép hiển thị các mã quét cho mỗi một phím được nhấn

Chương trình 9.1

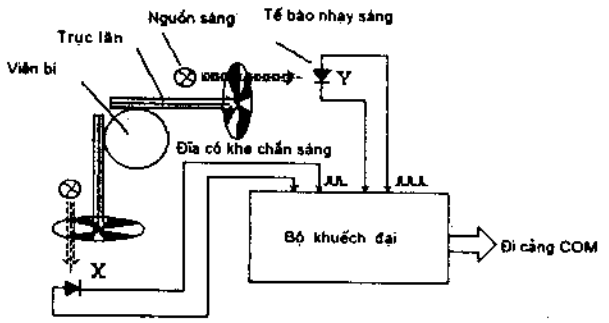
```
# include <stdio.h>
# include <dos.h>
void        show_key (void);
int         get_scan_code (void);
int         main (void)
{
int         ch;
            puts (" Chương trình hiện mã quét ");
            puts (" Nhấn F10 để thoát khỏi chương trình ");
            do
            {
                    ch=get_scan_code ( );
                    printf ("Mã quét: %02x (%d  decimal)\n", ch,ch);
            } while (ch !=68);
              return (0);
}
int         get_scan_code (void)
{
union REGS inregs,outregs;
int         scan_code;
            inregs.h.ah = 0;
            int 86 (0x16,&inregs,&outregs);
            scan_code = outregs.h.ah; / * nhận mã quét */
            return (scan_code);
}
```

9.2. GHÉP NỐI CHUỘT

Phần trung tâm của chuột (mouse) là một viên bi thép quay được khi dịch chuyển chuột (hình 9.3). Viên bi được phủ keo hoặc nhựa nhằm làm tăng ma sát cho truyền chuyển

động tới hai thanh nhỏ được đặt vuông góc với nhau. Các thanh này sẽ biến chuyển động của viên bi, tức là của chuột theo hướng X và Y thành sự quay tương ứng của hai đĩa gắn với chúng.

Trên hai đĩa có những khe liên tục đóng và ngắt hai chùm ánh sáng tới các sensor nhạy sáng để tạo ra các xung điện. Vậy số các xung này tỉ lệ với lượng chuyển động của chuột theo các hướng X và Y và số xung trên một giây biểu hiện tốc độ của chuyển động chuột. Kèm theo chuột có hai hoặc ba phím nhấn.



Hình 9.3 Sơ đồ cấu tạo của chuột.

Chuột được ghép nối với PC qua dây truyền tín hiệu (cáp nối) hoặc tia hồng ngoại theo một vài cách. Khi dịch chuyển hoặc nhấn một phím chuột, nó sẽ phát ra gói dữ liệu tới mạch ghép nối và mạch này sẽ phát ra một ngắt tới PC. Điều khiển ngắt này là một chương trình ghép nối quản lí chuột. Nó chặn ngắt lại để đọc dữ liệu và cập nhật các giá trị bên trong liên quan đến trạng thái hiện tại của bàn phím cũng như vị trí của chuột. Hơn nữa trình quản lí cung cấp một ghép nối mềm qua ngắt 33h của BIOS để dịch các giá trị bên trong này. Chương trình ghép nối chuột thực hiện việc nối con chuột với các chương trình ứng dụng. Chúng thường được cài đặt như các trình quản lí thiết bị (driver) khi khởi động hệ thống hoặc như là một chương trình nội trú TSR (như mouse.com).

Hiện có nhiều loại chuột. *Chuột bus* (bus mouse) là chuột được nối với PC qua một card ghép nối của riêng nó cắm trên một khe cắm mở rộng. *Chuột nối tiếp* (serial mouse) là chuột được nối qua cổng nối tiếp COM. *Chuột cổng vào* (inport mouse) là chuột được nối tới một đầu nối có cổng vào có thể được gắn trên một card đặc biệt hoặc là một phần của card có nhiều chức năng. IBM có *chuột PS/2* cho máy PS/2. Công ty Microsoft lần đầu tiên đưa vào loại *chuột PC* có 2 phím chuột. Hiện nay có nhiều nhà sản xuất khác đưa ra các loại chuột có thiết kế khác (thí dụ có 3 phím chuột và độ phân giải cho chuyển động của con trỏ chuột trên màn hình được nâng lên ...). Trong trường hợp này cần phải có một chương trình ghép nối chuột với PC riêng.

Con trỏ chuột cho biết vị trí tương đối của con chuột trên màn hình. Hình dạng của nó có thể thay đổi tuỳ theo chương trình điều khiển và ứng dụng. Trình điều khiển có trách nhiệm dịch chuyển con trỏ chuột trên màn hình. Con trỏ chuột có thể hiện trong chế độ đồ hoạ với *con trỏ chuột đồ hoạ* và *con trỏ cứng hoặc mềm* trong chế độ văn bản trong (text-mode). Con trỏ chuột cứng là một con trỏ thông thường chuyển động trên màn phù hợp với chuyển động chuột. Con trỏ chuột mềm lại có dạng là một kí tự do người lập trình tự chọn. Có thể định nghĩa loại và dạng con trỏ chuột bằng hàm 09h và 0Ah của ngắt 33h. Kích thước con trỏ cứng phụ thuộc vào ma trận kí tự hiện tại và chế độ màn hình. Trong khi đó con trỏ mềm được tạo bằng 2 giá trị 16 bit là mặt nạ màn hình (screen mask) và mặt nạ con trỏ (cursor mask). Các giá trị này sẽ được kết hợp với từ video cũ (gồm 2 byte mã kí tự và thuộc tính) trong RAM màn hình để tạo thành một từ nhớ video mới cho con trỏ như công thức sau:

Từ video mới = (từ video cũ AND *screen mask*) XOR *cursor mask*

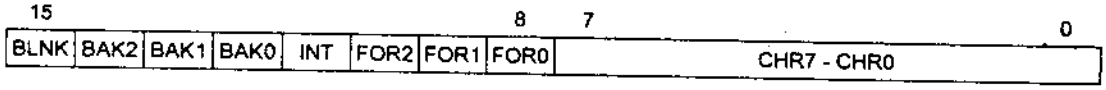
Thí dụ, từ cũ "A" có mã ASCII là 41h và thuộc tính là 01h. Như vậy từ cũ là 4101h. Screen nask là 4040h, cursor mask là 0F1Fh.

Vậy từ mới sẽ là: (4101h AND 4040h) XOR 0F1Fh = 4000h XOR 0F1Fh = 4F1Fh.

Đây là kí tự "O" có byte thuộc tính là 1Fh

Đơn vị đo trong ghép nối chuột là *Mickey* (1 mickey = 1/200 inch).

Ta có thể xoá, đặt hoặc đảo các bit trong từ nhớ màn hình. Cấu trúc của một từ nhớ video như sau:



CHR7 - CHR0: mã kí tự 8 bit

FOR2 - FOR0: màu cận (từ bảng màu hiện tại)

INT: cường độ sáng 1: sáng mạnh 0: sáng bình thường

BAK2 - BAK0: màu nền (từ bảng màu hiện tại)

Trong chế độ đồ hoạ, cách hiện con trỏ chuột cũng tương tự. Ở đây, một điểm ảnh (pixel) được gán bởi 1 hoặc nhiều bit. Do vậy phải bit được số bit trên một điểm ảnh. Hơn nữa kích thước con trỏ chuột ở đây luôn là 16*16 điểm ảnh.

Chuột hoạt động trên một tập các toạ độ chuột là x và y. Bảng 9.1 chỉ ra các toạ độ tại các chế độ màn hình khác nhau. Trong chế độ văn bản toạ độ x cực đại là 640 và y cực đại là 200. Trong chế độ đồ hoạ 640x480 các thông số này thay đổi là 640x480.

Bảng 9.1 **Độ phân giải chuột trong các chế độ màn hình**

Mode video	Card màn hình	Chế độ	Độ phân giải màn hình	Độ phân giải chuột
00h	CGA/EGA/VGA	Văn bản	80 x 25	640 x 200
0Dh	EGA/VGA	Đồ hoạ	320 x 200	640 x 200
12h	VGA	Đồ hoạ	640 x 480	640 x 480
25h	SVGA	Đồ hoạ	1024 x 768	1024 x 768

Nhìn vào bảng ta thấy độ phân giải chuột trong chế độ văn bản là 640x480 và độ phân giải màn hình là 80 kí tự trong một dòng và 25 dòng trên toàn màn hình (80 cột, 25 dòng). Như vậy toạ độ chuột lúc này phải được biến đổi thành một toạ độ hàng và cột bằng cách nhân toạ độ chuột x với 80/60 và toạ độ chuột y với 25/200). Chương trình 9.4 dùng sự biến đổi này để chọn một trong 3 tuỳ chọn.

Chương trình 9.2

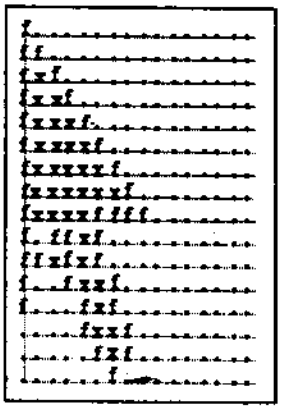
```
# include <stdio.h>
# include <dos.h>
# include <conio.h>
# define    Maxmouse_y  200          /* độ phân giải chuột trong mode text là  640x200 */
# define    N_Text_Row   25          /* 25 dòng văn bản */
int         mouse_init (void);
void        mouse_show (void)
 void       mouse_read (int *col, int *row, int *butt);
 int        main (void)
```

```
{
int             x,y,button;
clrscr( );
puts (" Chương trình tuỳ chọn thực đơn");
puts ("Dùng chuột để chọn");
puts ("Nhấn phím bất kì để tiếp tục");

getch ();
mouse_init ( );
mouse_show ( );
clrscr ( );
do
{
                gotoxy (1,1);
                puts ("Tuỳ chọn 1");
                puts ("Tuỳ chọn 2");
                puts ("Thoát chương trình");
                do
                {
                        mouse_read (&x, &y, &button);
                } while (button == 0);
                        if (button ==1)
                {
                        if (y<Maxmouse_Y/N_Text_Rows)
                           puts ("Tuỳ chọn 1 được chọn");
                        else  if (y<2*Maxmouse_Y/N_Text_Rows))
                           puts ("Tuỳ chọn 2 được chọn");
                        else  if (y<3*(Maxmouse_Y/N_Text_Rows)) break;
                }
} while  (!kbhit ( ));
  return (0);
}

int             mouse_int (void);
{
union REGS  inregs,outregs;
                inregs.x.ax = 0;
                int 86 (0x33,&inregs,&outregs);
                return(outregs.x.ax);
}

void            mouse_show (void)
{
union REGS  inregs,outregs;
                inregs.x.ax = 1;
                int 86 (0x33, &inregs,&outregs);
```

```
}

void        mouse_read (int *col, int *row, int *butt);
{
union REGS  inregs,outregs;
            inregs.x.ax = 3;
            int 86 (0x33,&inregs,&outregs);
            *butt = outregs.x.bx;
            *col = outregs.x.cx;
            * row = outregs.x.dx;
}
```

Hàm 09h cho phép định nghĩa con trỏ chuột về hình dạng và thuộc tính trong chế độ đồ hoạ. Các byte screen và cursor mask được giữ trong một bộ đệm. Con trỏ chuột luôn có 16*16 điểm ảnh và các điểm ảnh đó phải được phủ bởi các byte screen mask và cursor mask. Vậy tuỳ thuộc vào chế độ màn hình và số màu được sử dụng mà một bộ đệm đủ lớn để cần thiết cho các bit che (mask) này. Hình 9.4 chỉ ra một mặt nạ cho con trỏ hình mũi tên bao gồm các điểm ảnh thuộc các phần: đường bao (f), phần bên trong (x) và phần bên ngoài (.). Đối với cả hai byte screen mask và cursor mask, 256 byte (16 ký tự * 16 điểm / hàng) được đòi hỏi. Giả sử màu của đường bao (f) con trỏ chuột là FFh và màu phần bên trong (x) là 88h. Điểm hoạt động là điểm trên tận cùng góc trái mũi tên. Với tổ hợp

(bit màn hình AND screen mask) OR cursor mask



Hình 9.4 Mặt nạ con trỏ chuột hình mũi tên.

Các giá trị của các byte screen và cursor mask ứng với các điểm ảnh (f), (x) và (.) là:

Điểm ảnh	screen mask	cursor mask
f	00h	FFh
x	00h	88h
	FFh	00h

Gọi hàm 09h trong ngắt 33h cho việc này trong hợp ngữ như sau:

```
MOV  AX,09h              ; định nghĩa hàm
MOV  BX, 00h             ; điểm tận cùng góc trên bên trái
MOV  CX,00h              ; điểm tận cùng góc trên bên trái
MOV  DX, SEG  puffer     ; nạp đoạn đệm vào DX
MOV  ES, DX              ; nạp đoạn vào ES
MOV  DX, OFF puffer      ; nạp offset đệm vào DX
INT  33h                 ; gọi ngắt
```

Hàm 0Ah định nghĩa con trỏ chuột trong chế độ văn bản. Bằng cách xử lí screen mask và cursor mask như trên màu của con trỏ cũng như kí tự của nó có thể được thay đổi.

Thí dụ, tạo con trỏ chuột mềm sáng, nhấp nháy với màu số 3 trong bảng màu;

```
MOV  AX, 0Ah             ; chọn hàm
MOV  BX, 00h             ; con trỏ chuột mềm
MOV  CX, 00h             ; screen mask (xoá toàn bộ kí tự)
```

```
MOV DX, 8B02h          ; cursor mask (nhấp nháy=1Bh, màu cận=000h, cường độ
                       sáng=1, màu nền=011, kí tự=0000 0010)
INT 33h                ; gọi ngắt
```

Ngắt 33h trong BIOS cung cấp hầu hết các chức năng chính của chuột. Bảng 10.1 liệt kê một vài loại hàm khác của ngắt này.

Mô tả	Các thanh ghi vào	Các thanh ghi ra
Khởi tạo chương trình điều khiển chuột và đọc byte trạng thái	AX=0000h	AX=trạng thái: - 0000h phần cứng hoạt trình điều khiển không được lắp đặt. - FFFFh. được lắp đặt. BX=số phím chuột. - FFFFh hai phím - 0000h khác hai phím. - 0003h chuột Logitech
Hiện con trỏ chuột	AX=0001h	
Dấu con trỏ chuột	AX=0002h	
Trả về vị trí và trạng thái phím chuột	AX=0003h	- BX=trạng thái phím Bit 0 =1, phím trái được nhấn Bit 1=1, phím phải được nhấn Bit 2 =1, phím giữa được nhấn - CX=cột - DX=hàng
Vị trí con trỏ chuột	AX=0004h CX= cột DX= hàng	
Trả về số liệu nhấn phím	AX=0005h BX=phím 0000h: trái 0001h: phải 0002h: giữa	- AX= trạng thái phím Bit 0=1, phím trái được nhấn Bit 1=1, phím phải được nhấn Bit 2=1, phím giữa được nhấn - BX=số lần phím cần biết được nhấn kể từ lần gọi cuối cùng - CX=cột ở thời gian phím cần biết được nhấn lần cuối cùng - DX=hàng ở thời gian phím cần biết được nhấn lần cuối cùng
Định nghĩa dải ngang của con trỏ	AX=0007h CX=cột cực tiểu DX=cột cực đại	
Định nghĩa dải dọc của con trỏ	AX=0008h CX=hàng cực tiểu DX=hàng cực đại	
Trả về phiên bản phần mềm và loại chuột	AX=0024h	Trả về: AX=FFFFh khi có lỗi Các số còn lại chỉ thị: BH=phiên bản đa số BL=chứa phiên bản thiểu số CH=loại :1=bus, 2=nối tiếp, 3=cổng vào, 4=PS/2, 5=chuột HT. CL=ngắt: 0=PS/2, 2=IRQ2, 3=IRQ3, ...,7=IRQ7.

Chương trình 9.3 cho phép hiện số phiên bản của chuột, loại ghép nối chuột và các đường ngắt mà nó sử dụng.

Chương trình 9.3

```
# include <stdio.h>
# include <dos.h>
int  main(void)
{
Union REGS  inregs,outregs;
char          "mtyper[5]=("Bus","Serial","Inport","PS/2","HP");
char  "int_type[8]=("PS/2"."IRQ2","IRQ3","IRQ4"."IRQ5","IRQ6","IRQ7");
                inregs.x.ax=0x0024;
                int86(0x33, &inregs,&outregs);
                if  (outregs.x.ax=0xFFFF)
                {
                  puts("Không có chuột");
                      return (1);
                }
   printf("Phiên bản chuột là %x.%x.%x\n", outregs.h.bh,outregs.h.bl);
   printf("Chuột loại:  %s\n",mtypes [outregs.h.ch-1]);
   printf(" Ngắt:  %s\n", int_type[outregs.h.cl-1]);
   return (0);
}
```

Chương trình 9.4 hiện con trỏ chuột trong chế độ đồ hoạ. Ngắt số 10h và AX = 0011h khởi tạo màn hình đồ hoạ.có độ phân giải 640x480. Khi nạp AX = 0003 và gọi ngắt 10h sẽ quay trở về chế độ văn bản.

Chương trình 9.4

```
# include <stdio.h>
# include <dos.h>
# include <conio.h>
int          mouse_init (void);
void         mouse_show (void);
void         graphics (void);
void         text (void);
int          main (void)
{
             graphics ( );
             mouse_init ( );
             mouse_show( );
             do  { }  while  (!kbhit ());
             text ();
             return (0);
}

void         graphics (void)
{
```

```
union REGS  inregs,outregs;
            inregs.x.ax = 0x0011;
            int 86 (0x10,&inregs,&outregs);      /* màn hình 640x480 */
}
void        text (void)
{
union REGS  inregs,outregs;
            inregs.x.ax = 0x0003;
            int 86 (0x10,&inregs,&outregs);      /* chế độ văn bản */
}
int         mouse_int (void);
{
union REGS  inregs,outregs;
            inregs.x.ax = 0;
            Int 86 (0x33,&inregs,&outregs);
            return(outregs.x.ax);
}

void        mouse_show (void)
{
union REGS  inregs,outregs;
            inregs.x.ax = 1;
            int 86 (0x33, &Inregs,&outregs);
}
```

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

9.1 Viết một chương trình hiển thị các phím sau đây được nhấn: F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, Page up, Page down, Home, End, Insert hoặc Delete.

9.2 Viết một chương trình hiển thị một phím điều khiển được nhấn cùng với phím kí tự. Thí dụ Contrl-A

9.3 Viết một chương trình sử dụng phím chuyển dịch con trỏ (arrow keys) để chuyển một kí tự chạy vòng quanh màn hình. Gợi ý: xoá màn hình mỗi khi phím được nhấn rồi cập nhật kí tự với một toạ độ màn hình mới.

9.4 Viết một chương trình chứa thực đơn và thực đơn phụ theo bảng sau.

Thực đơn chính	Thực đơn phụ
F1 - File	F1 - New
	F2 - Open
	F3 - Close
	F4 - trở về thực đơn chính

F2 - Edit	F1 - Copy
	F2 - Paste
	F3 - Undo
	F4 - trở về thực đơn chính
F3 - Run	F1 - Step
	F2 - Go
	F3 - Trở về thực đơn chính
F4 - Exit	F1 - Không thoát
	F2 - Thoát

Các thực đơn được chọn khi sử dụng các phím chức năng thực đơn chính chứa 4 tuỳ chọn đó là File, Edit, Run và Exit. Khi nhấn F1 sẽ gọi thực đơn phụ New, Open, Close và Return v.v...khi thực đơn phụ được chọn một thông báo sẽ được hiện ra thí dụ File > Open. Sau khi thông báo được hiện, chương trình phải quay về thực đơn phụ. Người dùng có thể thoát khỏi thực đơn phụ bằng việc chọn F4 (trở về thực đơn chính). Việc thoát chương trình được chọn bằng tuỳ chọn Exit trong thực đơn chính.

9.5 Xác định các toạ độ cực đại x và y cho con trỏ chuột trong chế độ văn bản và trong chế độ đồ hoạ.

9.6 Viết một chương trình tự động chuyển con trỏ chuột trong chế độ văn bản từ góc trên tận cùng bên trái màn hình tới góc dưới tận cùng bên phải. Các bước chuyển động cách nhau 1 giây.

9.7 Làm lại bài tập trên nhưng làm cho con trỏ chuột chuyển động dọc theo các mép ngoài cùng của màn hình.

9.8 Viết chương trình trong đó con trỏ chuột trong chế độ văn bản được bật và tắt khi sử dụng phím chuột thứ nhất.

9.9 Làm lại bài tập trên nhưng dùng phím chuột thứ hai.

9.10 Viết chương trình giới hạn vị trí của con trỏ chuột ở nửa trên của màn hình.

9.11 Viết chương trình giới hạn vị trí của con trỏ chuột ở nửa dưới của màn hình.

9.12 Viết một chương trình phân tách màn hình ra thành 4 phần chữ nhật có quy ước số. Khi phím chuột thứ nhất được nhấn tại một trong các vùng chữ nhật này thì thông báo được hiện trên màn hình báo cho người sử dụng biết số của vùng đó.

9.13 Lặp lại các bài tập từ 2 đến 8 nhưng con trỏ chuột được hiện trong chế độ đồ hoạ.

9.14 Viết một chương trình hiện con trỏ chuột trong chế độ văn bản trên một thực đơn sau:

Mục 1
Mục 2
Mục 3
Thoát

Nếu con trỏ chuột được đặt trên dòng Mục 1 và nhấn phím chuột thứ nhất thì một thông báo " Mục 1 đã được chọn" và v.v...

9.15 Viết một chương trình trong chế độ đồ hoạ vẽ một hình chữ nhật trên màn hình khi dùng con trỏ chuột. Các điểm khởi phát và kết thúc của hình chữ nhật được định nghĩa bởi con trỏ chuột và việc nhấn phím chuột thứ nhất (phím trái).

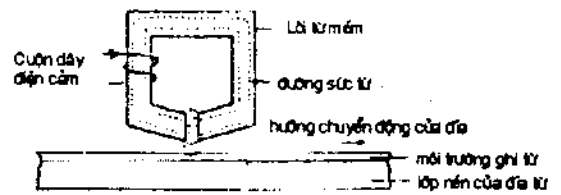
CHƯƠNG 10. GHÉP NỐI BỘ NHỚ KHỐI

Bộ nhớ khối của máy vi tính hiện nay chủ yếu dùng các đĩa từ và đĩa quang. Đặc điểm chính của chúng so với bộ nhớ chính là có tốc độ truy cập chậm hơn nhưng giá thành cho một Mbyte nhớ thấp và cái chính là có dung lượng nhớ rất lớn.

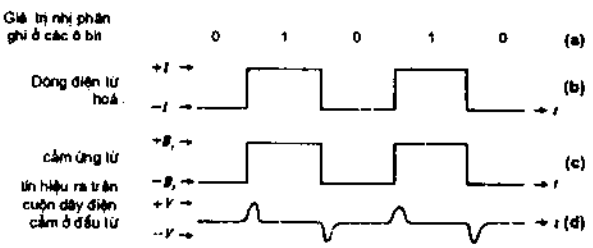
10.1. CÁC Ổ ĐĨA VÀ ĐĨA TỪ

10.1.1. Nguyên tắc đọc / viết trên đĩa từ

Các dữ liệu được ghi lại trên đĩa từ. Đĩa từ gồm một lớp nền làm bằng vật liệu nào đó trên mặt có phủ một lớp chất sắt từ làm môi trường ghi và đọc dữ liệu. Lớp đó bao gồm các vùng rất nhỏ gọi là domain (mỗi domain vẫn có đến hàng tỉ nguyên tử). Mỗi domain được đặc trưng bởi một *véctơ từ độ* riêng và được phân cách nhau bởi các biên giới gọi là vách domain. Ở trạng thái bình thường, các véctơ từ độ này được sắp xếp theo các hướng hỗn loạn và do vậy có thể coi một vùng nào đó trong lớp sắt từ (vẫn gồm rất nhiều domain) sẽ có từ độ bằng không. Khi vùng đó được đặt trong một từ trường ngoài biến thiên, dưới tác dụng của từ trường này các véctơ từ độ của các domain được sắp xếp theo cùng một hướng và tuỳ thuộc vào độ lớn và chiều của từ trường ngoài, vùng đó sẽ có một véctơ từ độ tổng cộng có hướng và độ lớn xác định. Với chất sắt từ, khi từ trường ngoài mất đi, véctơ từ độ của vùng đó vẫn giữ được một hướng và giá trị xác định còn đủ lớn gọi là *từ độ còn dư*. Đây gọi là quá trình *từ hoá* vùng vật liệu sắt từ. Có thể dùng quá trình này để lưu trữ thông tin số. Nếu vùng vật liệu được từ hoá có véctơ từ độ còn dư được sắp theo một chiều nào đó thì ta có được một ô nhớ lưu trữ một bit thông tin gọi là *ô bit* (bit cell). Tuỳ theo chiều này của véctơ từ độ mà ô bit lưu trữ giá trị "0" hoặc theo chiều ngược lại mà ô bit lưu trữ giá trị "1". Các thông tin được ghi lên đĩa bằng *đầu từ* viết (write head) gồm một lõi hình vòng làm bằng vật liệu từ mềm có độ từ thẩm cao (tức là có từ dư và trở kháng từ thấp). Trên lõi có quấn một cuộn dây điện cảm và có một khe hở nhỏ gọi là *khe từ* cho phép từ trường trong lõi được trải ra ngoài như hình 10.1. Từ trường này được dùng để từ hoá các vùng ghi thông tin trên mặt đĩa từ nhằm tạo ra các ô bit. Xử lí đọc thì ngược lại. Đầu từ đọc (read head) thông thường cũng giống như đầu từ viết là dựa trên nguyên tắc cảm ứng từ. Do đó có thể kết hợp cả hai chức năng trên một đầu từ để có đầu từ đọc/viết. Các ô bit chuyển động qua khe từ của đầu đọc sẽ gây ra sự biến thiên của từ thông trong khe và do vậy làm xuất hiện sức điện động cảm ứng trên hai đầu cuộn dây đầu từ. Sức điện động này là biểu hiện của các dữ liệu ghi lên đĩa. Hình 10.2 là một thí dụ về việc ghi dữ liệu nhị phân 01010 (hình 10.2a) cùng các dạng sóng điện của dòng chảy qua đầu từ ghi (hình 10.2b), từ độ $+Br$ ứng với mức lôgic 1, từ độ $-Br$ ứng với mức lôgic 0 (hình 10.2c) và

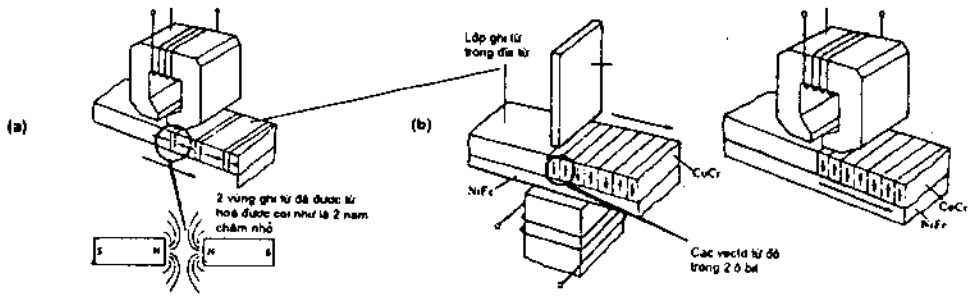


Hình 10.1 Đầu từ đọc/viết kiểu cảm ứng.



Hình 10.2 Các dạng sóng đọc/viết.

thế hiệu điện cảm ứng ra trên đầu cuộn dây của đầu từ đọc (hình 10.2d). Để tăng mật độ ghi tin (thí dụ từ 300.000 bit /cm^2 lên 2,5 triệu bit /cm^2) cần phải có kích thước các vùng ghi kể trên, phụ thuộc vào kích thước các hạt vật liệu ghi từ càng nhỏ càng tốt cũng như cải thiện thuộc tính từ của vật liệu. Phương pháp *ghi từ dọc* thay cho *ghi ngang* cũng là một giải pháp. Đó là việc thay vì cho việc từ hoá các domain theo mặt phẳng đĩa bằng việc từ hoá các domain theo chiều vuông góc với mặt phẳng này như hình 10.3a và b đã chỉ ra.



Hình 10.3 Ghi từ ngang (a) và ghi dọc (b).

10.1.2. Tổ chức vật lí của đĩa từ

Đĩa chỉ dùng được sau khi đã được định dạng, đó là việc xắp xếp, tổ chức các vùng trữ tin trong đĩa và "đặt tên", đánh số v.v.. cho chúng. Về mặt vật lí, đĩa từ được chia thành:

- Các *rãnh từ* (track) là các vùng vòng tròn đồng tâm có một bề dày nào đó dùng để ghi từ và các rãnh cách nhau bởi một vành hẹp không được từ hoá.
- Mỗi rãnh từ lại được phân ra thành các *cung từ* (sector). Mỗi cung từ gồm có các byte đánh dấu để chỉ thị số rãnh và cung từ đó cùng 512 byte chứa dữ liệu.
- DOS lại nhóm một số cung từ lại thành một đơn vị gọi là *liên cung* (cluster) để lưu trữ các tệp dữ liệu.
- Riêng với đĩa cứng còn một đơn vị nữa gọi là các *từ trụ* (cylinder), đó là quỹ tích vị trí các đầu từ của ổ đĩa ở mỗi thời điểm. Nguyên do là vì ổ đĩa cứng không chỉ có một mà là có một chồng đĩa xắp với nhau cùng số đầu từ tương ứng.

10.1.3. Đĩa từ mềm và ổ đĩa mềm

1. Đĩa và ổ đĩa mềm

Đĩa mềm được chế tạo gồm một đĩa nhựa hình tròn được phủ lớp sắt từ (thường là hợp chất của sắt hoặc cô-ban) để lưu trữ tin. Để bảo vệ khỏi bụi bẩn và hỏng hóc khác, đĩa này được lắp trong một phong bì giấy cứng hoặc trong hộp nhựa. Loại đĩa đường kính 5 1/4" được bọc trong giấy cứng có dung lượng trữ tin 320 kbyte hoặc 1,2 Mbyte. Loại đĩa 3 1/2" được lắp trong hộp nhựa có dung lượng 720 kbyte hoặc 1,44 Mbyte. Hầu hết các loại đĩa này đều là loại gồm 2 mặt ghi tin (double-side).

Để ghi hoặc đọc dữ liệu từ đĩa phải cho đĩa vào *ổ đĩa mềm* (floppy disk drive). Khi truy xuất dữ liệu, đĩa sẽ được điều khiển để quay bởi một môtơ điều khiển. Trên đĩa 5 1/4" có *lỗ chỉ số* cho phép xác định các điểm bắt đầu của rãnh từ. Khe bảo vệ chống viết trong đĩa 5 1/4" cho phép ngăn không cho ổ đĩa viết dữ liệu không mong muốn lên đĩa. Nếu muốn ghi lên đĩa chỉ việc bọc kín khe bằng một băng dính là được. Với đĩa 3 1/2" thì chỉ việc đẩy lẫy nhựa xuống (hở lỗ bảo vệ) ngăn không cho ghi và đẩy lẫy nhựa lên (lấp kín lỗ bảo vệ) khi cần ghi. Mô tơ điều khiển sẽ quay đĩa với tốc độ 300 vòng/phút (với đĩa 360 kbyte) hoặc 360 vòng/phút với các loại

đĩa còn lại khi thực hiện truy nhập thông tin. Các đầu từ được gắn ở một đầu cần truy xuất và dưới tác dụng của một môtơ bước cần này sẽ được chuyển dịch theo phương bán kính đĩa đến rãnh cần truy cập. Như vậy khác với trường hợp băng ghi từ ở máy ghi âm, chuyển động tương đối giữa đầu từ và đĩa là một chuyển động hai chiều. Nếu một đĩa mềm một mặt trong trường hợp chậm nhất cho một vòng quay để truy cập dữ liệu mất chừng 170 - 200 ms; chuyển động tịnh tiến của đầu từ nhiều nhất cỡ 200 ms. Như vậy mất 200 ms cho trường hợp truy cập dữ liệu chậm nhất.

Đĩa mềm quay dưới đầu từ đọc/viết để cho bộ điều khiển có thể truyền những xung dòng điện ngắn qua cuộn dây đầu từ nhằm tạo ra từ trường mong muốn đặt lên các ô bit của mặt đĩa ở những thời điểm cần thiết. Như vậy cần thiết phải có một sự tổ chức chính xác giữa việc điều khiển quay đĩa, vị trí của cần truy xuất và đồng bộ các xung dòng như thế nào đó để các giá trị xác định được viết vào các ô bit. Nếu cần viết liên tiếp các số liệu thì bộ điều khiển ổ đĩa phải cấp một chuỗi xung tới đầu từ đọc/viết đồng bộ với việc ghi số liệu. Kích thước của các khe và khoảng cách giữa đầu từ và mặt đĩa có ảnh hưởng chính đến mật độ ghi tin. Trong ổ đĩa mềm, khi truy xuất đĩa, đầu từ chạm nhẹ vào mặt đĩa (giống như ở băng cassette), do vậy khoảng cách này bằng 0. Như vậy sau một thời gian sử dụng, đĩa mềm sẽ bị hao mòn đi mặc dù đã được phủ thêm một lớp teflon bảo vệ. Tuy nhiên điều này được hạn chế bởi tốc độ quay của đĩa chậm và chỉ quay khi cần truy xuất dữ liệu. Để đọc một giá trị lưu trữ nào đó, bộ điều khiển chuyển dịch đầu từ tới vùng xác định và quay đĩa. Khi số liệu cần thiết xuất hiện dưới đầu từ thì bộ điều khiển mở một cổng lôgic để truyền các tín hiệu cảm ứng từ cuộn dây đầu từ tới bộ khuếch đại.

Ngày nay các đĩa từ mềm được định dạng từ 40 đến 80 rãnh và rãnh ngoài cùng (có đường kính lớn nhất) mang số 0. Trong đĩa 5 1/4" điểm bắt đầu của mỗi cung từ trên một rãnh được xác định bởi lỗ chỉ số. Với đĩa 3 1/2" vị trí cố định của một vết khía trong bản kim loại ở tâm đĩa làm nhiệm vụ này. Mỗi cung từ gồm có các byte đánh dấu để chỉ thị số rãnh và cung từ đó cùng 512 byte chứa dữ liệu. Thường có từ 8 đến 18 cung từ trên một rãnh. Bảng 10.1 cho một số tham số của các đĩa mềm.

Bảng 10.1 **Các thông số của các đĩa từ mềm**

Đĩa	Số track	Số sector/track	Kích thước cluster (số sector)	Bề rộng track (mm)	Tổng số sector
5 1/4" 360 kbyte	40	9	2	0,330	720
5 1/4" 1,2 Mbyte	80	15	1	0,160	2400
3 1/2" 720 kbyte	80	9	2	0,115	1440
3 1/2" 1,44 Mbyte	80	18	1	0,115	2880

Dung lượng trữ tin DLTT của đĩa được tính như sau:

DLTT = số track x số sector trên một track x số mặt đĩa x 512 [byte]

Khi định dạng bằng lệnh FORMAT, DOS tạo ra các rãnh và các cung từ. Nó viết giá trị 0F6h (kí tự "÷") vào các byte của các cung từ.

Việc đánh số nhận dạng các đơn vị được theo hai cách. BIOS nhận dạng theo tổ chức vật lí, tức là số rãnh bắt đầu từ 0 và số cung từ trên một rãnh bắt đầu từ 1 đến hết số cung từ trong rãnh đó. Trong khi đó thì DOS lại nhận dạng theo tổ chức lôgic mà ở đó các cung từ được đánh số lần lượt từ 0 liên tục đến hết số cung từ của đĩa.

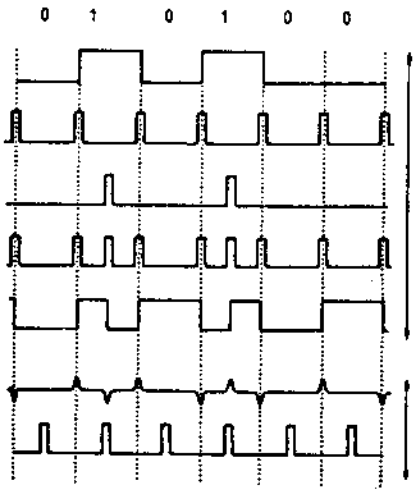
2. Các phương pháp ghi thông tin lên đĩa mềm và phát hiện lỗi

Thông tin được viết thành một dòng bit liên tục trên các rãnh từ. Các bit dữ liệu được lưu trữ trong các ô bit (bit cell). Mỗi ô bit gồm hai nửa chứa bit giữ nhịp (clock bit) và bit dữ liệu (data bit).

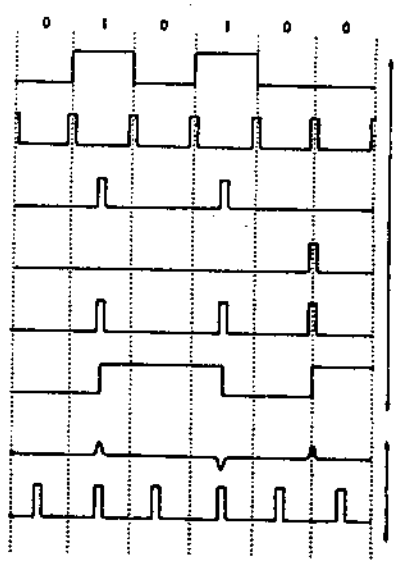
Có hai trong số các phương pháp ghi số liệu lên đĩa mềm thường được sử dụng là:

- *Phương pháp điều tần FM*: trong đó các nửa ô bit nhịp đều được ghi các xung nhịp và các nửa ô bit dữ liệu được ghi một xung khi giá trị bit dữ liệu bằng 1. Các dạng sóng được chỉ ra trong hình 10.4.

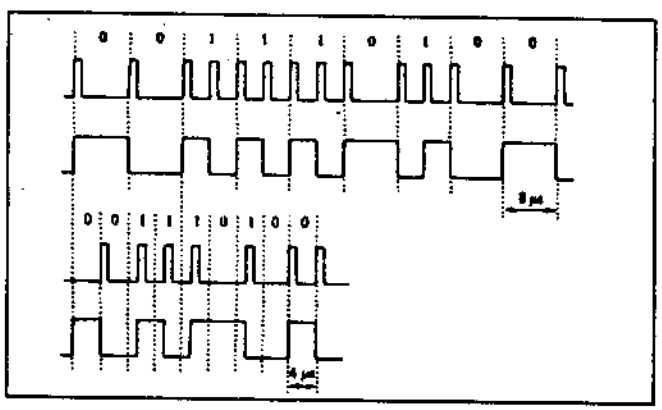
- *Phương pháp điều tần sửa đổi một lần MFM*: trong đó các nửa ô bit dữ liệu được ghi như trên nhưng các nửa ô bit nhịp chỉ được ghi một xung khi ô bit đó và ô bit trước đó có dữ liệu bằng 0. Các dạng sóng được chỉ ra trên hình 10.5. Phương pháp này cho một mật độ ghi tin tăng gấp 2 lần so với phương pháp FM như hình 10.6 chỉ ra.



Hình 10.4 Các dạng sóng ghi kiểu FM.



Hình 10.5 Ghi kiểu MFM.



Hình 10.6 Mật độ ghi tin của phương pháp MFM tăng gấp đôi so với phương pháp FM.

Chương trình định dạng đĩa FORMAT cho phép tạo ra các track và sector trên đĩa. Như trên đã nói một sector chứa 512 byte dữ liệu, ngoài ra các track và sector còn có nhiều byte bổ sung lưu trữ các thông tin dùng cho định vị và đồng bộ. Trường CRC gồm các byte phát hiện lỗi bằng phương pháp kiểm tra dư thừa vòng CRC (cycle redundancy check) như đã được trình bày trước đây. Với đa thức sinh 16 bit theo quy định của CCITT các byte CRC cho phép phát hiện

được các lỗi bit đơn là 100%, các lỗi bit kép là 100%, các lỗi xảy ra ở một số chẵn các bit là 100%, các lỗi nhóm nhỏ hơn hoặc bằng 16 bit là 100%, các lỗi nhóm 17 bit là 99,9969% và các lỗi khác là 99,9984% !

Định dạng kiểu FM

Phần mở đầu track
40 byte khe phân cách 4A (GAP 4A): FFh
6 byte đồng bộ (SYNC): 00h
1 byte đánh dấu địa chỉ chỉ số (IAM): FCh
26 byte khe phân cách1 (GAP1): FFh

Sector
6 byte đồng bộ (SYNC): 00h
1 byte dấu địa chỉ nhận dạng (IDAM): FEh
4 byte nhận dạng (ID): track, head, sector, size*
2 byte mã CRC
11 byte khe phân cách 2 (GAP2): FFh
6 byte đồng bộ (SYNC): 00h
1 byte FBh hoặc 4 byte F8h đánh dấu bắt đầu vùng lưu trữ dữ liệu (DAM)
512 byte lưu trữ dữ liệu (DATA)
2 byte mã CRC
40 byte khe phân cách 3 (GAP3): FFh

Kết thúc rãnh (EOT)
xx byte FFh khe phân cách (GAP 4B): FFh

Định dạng kiểu MFM

Phần mở đầu track
80 byte khe phân cách 4A (GAP 4A): 4Eh
12 byte đồng bộ (SYNC): 00h
4 byte đánh dấu IAM: C2h, C2h, C2h, FCh
50 byte khe phân cách1 (GAP1): 4Eh

Sector
12 byte đồng bộ (SYNC): 00h
4 byte dấu DAM: A1h, A1h, A1h, FEh
4 byte nhận dạng (ID): track, head, sector, size*
2 byte mã CRC
22 byte khe phân cách 2 (GAP2): FFh
12 byte đồng bộ (SYNC): 00h
4 byte A1h, A1h, A1h, FBh hoặc 4 byte A1h, A1h, A1h, F8h đánh dấu DAM
512 byte lưu trữ dữ liệu (DATA)
2 byte mã CRC
80 byte khe phân cách 3 (GAP3): 4Eh

Kết thúc rãnh (EOT)
xx byte FFh khe phân cách (GAP 4B): 4Eh

* kích thước sector ; xx tuỳ loại đĩa và ổ đĩa

3. Bộ điều khiển và ổ đĩa mềm

CPU và các ổ đĩa liên lạc được với nhau qua bộ điều khiển ổ đĩa mềm. Thường mạch điện này được lắp trên một bản mạch cắm vào khe cắm mở rộng của PC.

Ngày nay hầu hết mạch này được tích hợp ngay trên bản mạch chính. Bộ điều khiển có một vi xử lí riêng với chương trình trong ROM của nó. Cùng với ROM, chúng hình thành khối gọi là *bộ vi điều khiển* (microcontroller). Bộ vi điều khiển này có thể lập mã với định dạng FM hoặc MFM và tiến hành các bước đọc hoặc viết dữ liệu với đĩa. Thí dụ, việc viết dữ liệu được thực hiện qua các bước sau:

- Số liệu được truyền từ bus vào bộ ghép nối bus.
- Bộ điều khiển xác định byte CRC, biến đổi số liệu song song thành nối tiếp và định dạng chúng một cách thích hợp.
- Bộ tách số liệu biến đổi chuỗi số liệu này thành chuỗi mã FM hoặc MFM, đồng thời phát ra các xung đánh dấu (address mark).
- Mạch ghép nối truyền chuỗi số liệu đã được mã hóa tới ổ đĩa.
- Đầu từ viết số liệu được mã hóa lên đĩa.

Một cáp bẹt 34 dây nối mạch này với ổ đĩa mềm. Có thể nối 2 ổ đĩa mềm với cáp. Từ chân 10 đến chân 16 các dây được bắt xoắn để phân biệt giữa ổ đĩa A và B. Trong ổ đĩa mềm có một số cầu nối (jumper) để đặt cấu hình cho ổ đĩa như: chọn ổ đĩa, đặt terminator, thay đổi ổ đĩa và nhận dạng loại đĩa ...

4. Thâm nhập ổ đĩa mềm

a, Thâm nhập qua DOS: DOS cung cấp 3 ngắt 25h, 26h và 21h cho phép thâm nhập các đĩa mềm và đĩa cứng. Các ngắt 25h và 26h cho phép thâm nhập ở mức thấp, tức là có thể định địa chỉ trực tiếp tới các sectơ lôgic. Bảng sau cho các định dạng lời gọi và trả về của ngắt 25h dùng để đọc và ngắt 26h dùng để viết một sectơ lôgic.

INT 25h Đọc 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	số ổ đĩa (0:A, 1: B)	
AX		mã lỗi (bảng dưới).
CX	số sectơ	
DX	sectơ đầu tiên	
BX	offset của bộ đệm đọc	
DS	segment bộ đệm đọc	
Carry		lỗi nếu <> 0

INT 26h Viết 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	số ổ đĩa (0:A, 1: B)	
AX		mã lỗi (bảng dưới)
CX	số sectơ	
DX	sectơ đầu tiên	
BX	offset của bộ đệm viết	
DS	segment bộ đệm viết	
carry		lỗi nếu <> 0

Bảng mã lỗi như sau:

Mã	Lỗi
01h	Lệnh không hợp lệ
02h	Trường địa chỉ không hợp lệ
04h	Không tìm thấy sectơ
08h	Tràn DMA
10h	Lỗi CRC hoặc ECC
20h	Lỗi ở bộ điều khiển
40h	Lỗi tìm kiếm
80h	Ổ đĩa không sẵn sàng

Thí dụ, chương trình sau cho phép đọc sectơ lôgic 519 bằng ngắt 25h từ ổ đĩa B

Chương trình 10.1

```
unsigned char far buffer [512];
void  read_log_sector_519 (void);
{union REGS inregs, outregs;
  struct SREGS  segregs;
     inregs.h.al = 0x01;                    /* ổ đĩa B */
     inregs. x.cx = 0x0001;                 /* chỉ đọc 1 sectơ */
     inregs.x.dx = 0x0207;                  /* sectơ thứ nhất là 519 */
     inregs.x.bx = FP_OFF(buffer);          /* truyền offset bộ đệm tới bx */
     segregs.ds = FP_SEG(buffer);           /* truyền segment của bộ đệm tới ds */
     int86(0x25, &inregs, &outregs, &segregs);  /* gọi ngắt, sectơ được truyền
                                                tới bộ đệm */
     if((outregs.x.cflag & 0x01) == 0x01)   /* kiểm tra cờ được đặt ? */
            { printf("\n error code: %x", outregs.x.ax);  /* hiện lỗi */
            }
}
```

Chương trình sau cho phép viết sectơ lôgic 1482 vào đĩa mềm trong ổ A bằng ngắt 26h.

Chương trình 10.2

```
unsigned char far buffer [512];
void  write_log_sector_1482 (void);
{union REGS inregs, outregs;
  struct SREGS  segregs;
     inregs.h.al =0x00;                          /*  ổ đĩa A  */
     inregs. x.cx = 0x0001;                      /* chỉ viết 1 sectơ */
     inregs.x.dx = 0x05ca;                       /* sectơ thứ nhất là 1482 */
     inregs.x.bx = FP_OFF(buffer);               /*truyền offset bộ đệm tới bx*/
     segregs.ds = FP_SEG(buffer);                /* truyền segment của bộ đệm tới ds */
     int86(0x26, &inregs, &outregs, &segregs);   /*gọi ngắt, sectơ được truyền
                                                 tới bộ đệm*/
     if((outregs.x.cflag & 0x01) == 0x01)        /* kiểm tra cờ được đặt ?*/
              { printf("\n error code: %x", outregs.x.ax);  /* hiện lỗi*/
              }
}
```

b, Thâm nhập qua ngắt 13h của BIOS. 6 hàm của ngắt 13h từ 00h đến 05h cho phép thâm nhập ổ đĩa mềm với các chức năng như sau:

00h	Khởi động ổ đĩa
01h	Đọc trạng thái và tác vụ cuối cùng
02h	Đọc các sector
03h	Viết các sector
04h	Kiểm tra các sector
05h	Định dạng track.

Thí dụ, đọc sector 1 của track 0, head 0 của đĩa mềm trong ổ đĩa B (sectơ này chứa sector khởi động của đĩa):

```
buffer   DB  512  DUP  (?)        ; cấp vùng đệm đọc 512 bytes (=1 sector)
mov      AH,02h                   ; gọi hàm  02h (đọc sector)
mov      DL,01h                   ; ổ đĩa mềm thứ 2 (BĐ)
mov      DH,00h                   ; đầu từ 0
mov      CH,00h                   ; track 0
mov      CL,01h                   ; sector 1
mov       AL,01h                  ; chỉ đọc 1 sector
mov      ES, SEG buffer           ; địa chỉ đoạn vùng đệm đọc
mov      BX,offset buffer         ; địa chỉ offset vùng đệm đọc
INT 13h                           ; đọc sector vào vùng đệm bằng ngắt 13h.
```

c, Thâm nhập trực tiếp qua các thanh ghi.

Tất cả các bộ điều khiển ổ đĩa mềm đều sử dụng DMA kênh 2 để truyền số liệu giữa chúng và bộ nhớ chính trong quá trình ghi hoặc đọc dữ liệu. Sau khi kết thúc một lệnh, bộ điều khiển phát ra một ngắt cứng qua IRQ6 trong pha kết quả (xem sau). Ổ đĩa mềm là một thiết bị ngoại vi nên có thể thâm nhập trực tiếp nó qua các địa chỉ cổng của các thanh ghi như trình bày dưới đây:

	Địa chỉ sơ cấp	Địa chỉ thứ cấp	Viết (W) / Đọc (R)
Địa chỉ cơ sở	3F0	370	
Thanh ghi trạng thái A (PS/2)	3F1	371	R
Thanh ghi trạng thái B (PS/2)	3F1	371	R
Thanh ghi lối ra số DOR	3F2	372	R/W
Thanh ghi trạng thái chính	3F4	374	R
Thanh ghi chọn tốc độ dữ liệu	3F4	374	W
Thanh ghi dữ liệu	3F5	375	R/W
Thanh ghi lối vào số DIR (AT)	3F7	377	R
Thanh ghi điều khiển cấu hình	3F7	377	W
Kênh DMA	2	2	
IRQ	6	6	
INTR	0E	0E	

Dưới đây là nội dung một vài thanh ghi quan trọng trong bảng trên.

Thanh ghi lối ra số là thanh ghi chỉ có thể viết, nội dung như sau:

7							0
MOTOD	MOTOC	MOTOB	MOTOA	/DMA	/RES	DR1	DR0

MOTOD, MOTOC, MOTOB, MOTOA: điều khiển các môtơ cho các ổ đĩa mềm A, B, C, D.
1 = start 0 = stop

DMA : kênh DMA và IRQ. 1 = cho phép 0 = không cho phép

/RES : reset bộ điều khiển. 1 = cho phép 0 = xử lý reset bộ điều khiển

DR1, DR0: chọn ổ đĩa
00 = ổ 0 (A); 01 = ổ 1 (B) 10 = ổ 2 (C) 11 = ổ 3 (D)

Thanh ghi trạng thái chính là thanh ghi chỉ đọc:

7							0
MRQ	DIO	NDMA	BUSY	ACTD	ACTC	ACTC	ACTA

MRQ: yêu cầu chính 1 = thanh ghi số liệu đã sẵn sàng 0 = không sẵn sàng

DIO: hướng truyền số liệu. 1 = từ bộ điều khiển đến CPU 0 = từ CPU đến bộ điều khiển

NDMA: chế độ không DMA. 1 = bộ điều khiển không ở mode DAM 0 = ở mode DMA

BUSY: 1 = bộ điều khiển đang chạy lệnh 0 = không chạy lệnh

ACTD, ACTC, ACTB, ACTA: 1 = các ổ đĩa đang định vị đầu từ 0 = không định vị

Các bộ vi điều khiển có một tập lệnh (thí dụ µPD765 cùng ROM có 15 lệnh). Các lệnh này được xử lí theo 3 pha (giai đoạn):

- ***Pha lệnh:*** Pha này được phát bằng việc kích hoạt bộ điều khiển qua thanh ghi lối ra số DOR và chọn ổ đĩa cần truy nhập. Trong máy PC, vì việc truyền dữ liệu bằng DMA nên bit DMA trong DOR cần được đặt. Nếu cần thâm nhập đĩa thì phải khởi phát môtơ A, B, C hoặc D bằng việc đặt bit MOTx tương ứng. Thí dụ, phát pha lệnh:

```
MOV    DX, 3F2h      ; nạp địa chỉ DOR vào dx
OUT    DX,1Ch        ; ra 0001 1100 (môtơ A, DMA =1, AKTV =1, LW1/LW0 = 00)
```

Bộ điều khiển đợi một lệnh cấp qua thanh ghi số liệu. Các lệnh gồm một số byte lệnh. Thí dụ *lệnh đọc 1 sectơ* gồm 9 byte như được mô tả dưới đây. Lưu ý rằng bộ điều khiển chỉ chấp nhận một byte lệnh nếu bit MRQ trong thanh ghi trạng thái được đặt.

Như vậy chương trình phải đọc thanh ghi trạng thái chính để kiểm tra bit MRQ đã được đặt hay chưa trước khi truyền byte lệnh.

Định dạng lệnh đọc 1 sectơ

Byte\ bit	7	6	5	4	3	2	1	0
0	M	F	S	0	0	1	1	0
1	x	x	x	x	x	HD	DR1	DR0
2	Cylinder							
3	Đầu từ (head)							
4	Số sectơ (sector number)							
5	Kích thước sectơ (sector size)							
6	Độ dài rãnh / số sectơ cực đại							
7	Chiều dài của khe GAP 3							
8	Chiều dài dữ liệu							

- M: lệnh cho bộ điều khiển thực hiện cùng một lệnh cho nhiều rãnh khác, tức là nó không chỉ đọc sectơ trên một mặt đĩa mà cả với mặt khác. Trong khi truyền số liệu cho sectơ thứ nhất, bộ điều khiển DMA liên tục đếm lên địa chỉ đích, sectơ được đọc sau không viết đè lên số liệu được đọc trước mà được tiếp nối. Như vậy có thể đọc một vài sectơ một lúc bằng việc đặt bit này bằng 1. M=1 thực hiện lệnh với cylinder M=0 thực hiện với 1 rãnh đơn

- F: phương pháp ghi. 1=MFM 0=FM

- S: mode nhảy qua. 1=bỏ qua trường địa chỉ của số liệu đã bị xoá 0=không bỏ qua

- HD: số đầu từ (luôn bằng địa chỉ đầu từ trong byte 3).

- DR1, DR2: chọn ổ đĩa. 00=ổ A; 01=ổ B; 10=ổ C; 11=ổ D.

- Cylinder, head, sector number: địa chỉ của sectơ được đọc.

- Sector size: 0=128 byte; 1=256 byte; 2=512 byte; ... 7=16 kbyte.

- Byte lệnh thứ 6: số sectơ trên một rãnh hoặc số sectơ cực đại tại đó lệnh được chạy.

- Chiều dài khe 3: khe này như một bộ đệm giữa 2 sectơ nhằm làm cho việc đọc chính xác, tránh sự đè các số liệu của sectơ này lên sectơ sau do các sectơ bị dãn dài ra vì nhiệt, vì tốc độ quay đĩa v.v... Tiêu chuẩn=42; cực tiểu=32

- Chiều dài dữ liệu: nếu số byte trên một sectơ được định nghĩa là 00 (128 byte/sectơ) thì byte lệnh thứ 8 này chính là số byte muốn truy nhập. Nếu chiều dài dữ liệu nhỏ hơn 128 byte thì những byte còn lại được điền là 0. Vì PC dùng kích thước 512 byte/sectơ nên byte lệnh này không có ý nghĩa. Tuy nhiên vẫn phải truyền hết cả 9 byte lệnh; vì vậy phải nạp số FFh cho byte này.

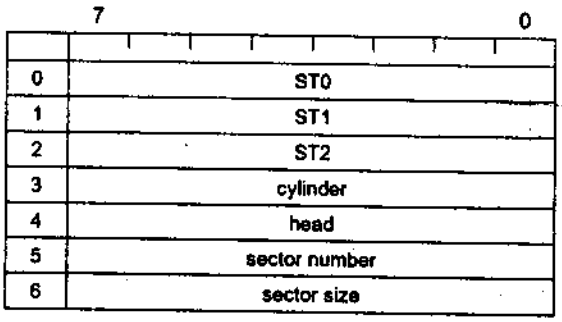
• ***Pha số liệu***: Sau khi byte lệnh cuối cùng kết thúc, bộ điều khiển đợi cho đến khi thời gian nạp đầu từ (head load) trôi qua rồi hạ đầu từ xuống mặt đĩa. Tiếp đó đầu từ và bộ điều khiển khởi phát đọc trường địa chỉ nhận dạng ID từ đĩa đang quay. Nếu nó phát hiện dấu địa chỉ nhận dạng trùng với địa chỉ sectơ cần thiết trong chương trình thì bộ điều khiển sẽ đọc vùng dữ liệu của sectơ và các byte kiểm tra tổng CRC rồi truyền vào bộ nhớ chính. Nếu xử lí này kết thúc, bộ điều khiển tăng địa chỉ sectơ lên 1 và tiếp tục xử lí với sectơ tiếp theo. Quá trình cứ tiếp tục cho đến khi bộ điều khiển nhận được tín hiệu TC (terminal count) từ chip DMA; hoặc bộ điều khiển phát ra một tín hiệu TC ẩn khi kết thúc rãnh được đọc/viết; hoặc số liệu không thể được truyền vào bộ nhớ chính đủ nhanh; hoặc đã tới số sectơ cực đại cần đọc/viết trong chương trình. Sau khi kết thúc quá trình đọc/viết, đầu từ đứng nguyên ở chỗ cuối cho đến khi hết thời gian nhả đầu từ (head unload) để cho nếu CPU phát ra một lệnh khác trong thời gian này thì có thể bỏ qua thời gian nạp đầu từ tiếp theo.

Nếu bộ điều khiển nhận được 2 xung IDX qua đường IDX từ ổ đĩa mà không phát hiện được dấu địa chỉ nhận dạng của sectơ mong muốn thì nó sẽ đặt mã ngắt IC1, IC0 tới giá trị 01

để chỉ báo. Thêm nữa, nó sẽ đặt bit ND trong thanh ghi trạng thái lên 1 để chỉ thị rằng nó đã không phát hiện ra sectơ mà chương trình cần. Hai xung lặp lại IDX có nghĩa là lỗ chỉ số của đĩa mềm đã đi qua đầu nhạy quang 2 lần hay bộ điều khiển đã phát hiện dấu địa chỉ chỉ số (IAM) ở đầu rãnh 2 lần và do đó đĩa mềm đã thực hiện được ít nhất trọn vẹn 1 vòng quay.

Trong pha lệnh, CPU phải kích hoạt bộ điều khiển và truyền các byte lệnh. Nhưng trong pha dữ liệu, bộ điều khiển và ổ đĩa mềm làm việc độc lập với CPU khi định vị đầu từ, tìm cylinder mong muốn, đọc/viết, thậm chí truyền dữ liệu từ hoặc đến các sectơ cùng các xử lí khác. Bộ điều khiển DMA được chương trình hoá trước sẽ thực hiện các công việc này một cách độc lập nhưng chương trình phải đặt bit DMA trong thanh ghi lối ra số nhằm cho bộ điều khiển thực sự phát ra các tín hiệu DRQ2 và IRQ6. Thêm nữa bit NDM trong byte thứ hai của lệnh các thông số đặc biệt của ổ đĩa (specify drive parametter command) phải được xoá để cho phép hoạt động DMA. Trong quá trình viết số liệu, bộ điều khiển kích hoạt điều khiển DMA truyền dữ liệu từ bộ nhớ chính tới bộ điều khiển qua kênh 2 DMA và viết sectơ dữ liệu cùng với các dữ liệu điều khiển và kiểm soát như các khe GAPs, IDAM, CRC v.v... vào đĩa. Nếu ta lập trình lệnh chuyển dữ liệu thì CPU có thể tiếp tục chạy các lệnh khác không đòi hỏi chu kì bus thâm nhập bộ nhớ (thí dụ, nhân hai giá trị của các thanh ghi trong CPU ...).

- ***Pha kết quả:*** Vào cuối pha dữ liệu (hoặc nếu chương trình thực hiện một lệnh không có pha dữ liệu như lệnh kiểm tra trạng thái ổ đĩa) bộ điều khiển sẽ phát ra một ngắt cứng. CPU sẽ ghi nhận rằng lệnh này đã thực hiện xong và nạp các byte trạng thái kết quả vào thanh ghi dữ liệu. Bộ điều khiển đĩa mềm có 4 thanh ghi trạng thái từ ST0 đến ST3. Thanh ghi trạng thái cung cấp một vài thông tin về đầu từ, ổ đĩa, v.v... của sectơ được xử lí cuối cùng. Cấu trúc của 4 thanh ghi trạng thái như được chỉ ra dưới đây.



ST0, ST1, ST2: thanh ghi trạng thái từ 0 ... 2
cylinder, head, sector number : địa chỉ của sectơ được đọc cuối cùng.
sector size: 0=12 byte, 1=256 byte, 2=512 byte ...7=16 kbyte.

7 byte kết quả này có thể được đọc qua thanh ghi số liệu bằng lệnh IN. Như trường hợp viết byte lệnh, ở đây ta phải kiểm tra liệu thanh ghi số liệu đã sẵn sàng hay chưa. Thí dụ sau cho phép đọc sectơ 12 , đầu từ 1, rãnh 40 của đĩa 1,2 Mbyte trong ổ đĩa B. Bộ điều khiển DMA được chương trình hoá trước.

```
main ( )
 { unsigned  char st0, st1, st2, cylinder, head, sector, byte_per_sector;
 st0 = read_controller();                /* đọc byte kết quả st0 từ bộ điều khiển */
 st1= read_controller();                 /* đọc byte kết quả st1 từ bộ điều khiển */
 st2 = read_controller();                /* đọc byte kết quả st2 từ bộ điều khiển */
 cylinder = read_controller()            /* đọc số cylinder liên quan từ bộ điều khiển*/
```

```
head = read_controller();                    /* đọc số đầu từ */
sector = read_controller();                  /* đọc sectơ*/
byte_per_sector = read_controller();         /* đọc kích thước sectơ */
unsigned  char read_controller(void)
{
    int mrq;
            unsigned  char  result_byte;
            for (; ;)   {                    /  * đợi đến khi bộ điều khiển sẵn sàng */
            mrq = inport(0x3f4)  && 0x80;    /*xác định trạng thái của MRQ*/
            if (mrq ==0x80)   {              /* kiểm tra liệu bộ điều khiển đã sẵn sàng ? */
          outport(0x3f5, result_byte);       /* xuất byte lệnh ra thanh ghi dữ liệu */
            return(result_byte);             /* trả về byte kết quả */
            }
        }
    }
```

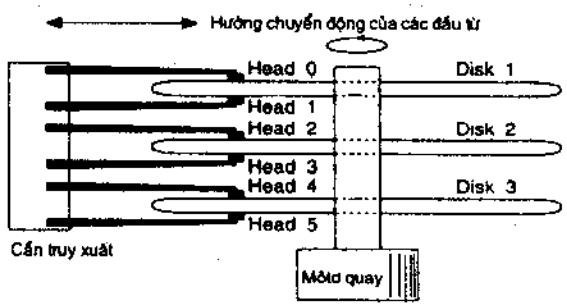
10.1.4. Đĩa cứng và ổ đĩa cứng

1. Đặc điểm cấu tạo

Về nguyên tắc, đĩa cứng khác đĩa mềm không nhiều nhưng về mặt kĩ thuật thì nó có những yêu cầu cao và phức tạp hơn hẳn. Đĩa cứng luôn quay trong quá trình máy tính hoạt động trong khi đĩa mềm chỉ quay khi truy xuất thông tin trên đĩa. Hiện nay có những chương trình cho phép tiết kiệm năng lượng, tức là khi gặp khoảng thời gian dài không truy xuất thì chương trình sẽ tự động giảm tốc độ quay hoặc ngừng hẳn việc quay đĩa cứng nhưng về nguyên tắc trong một phiên làm việc đĩa luôn được quay với vận tốc không đổi. Dữ liệu cũng được tổ chức thành các rãnh từ, cung từ, liên cung như trong đĩa mềm và đầu từ đọc/viết cũng được chuyển động bởi cần truy xuất. Nhưng các chi tiết cơ khí và mạch điều khiển ổ đĩa phải có những điểm đặc biệt để có thể làm việc được với đĩa có dung lượng nhớ rất cao (mật độ ghi tin rất lớn) cùng các chỉ tiêu kĩ thuật khác cao hơn so với ổ đĩa mềm nhiều. Các chi tiết này phải được bảo vệ cẩn thận để chống bụi. Các đĩa từ cứng, các đầu từ, cần truy xuất và cơ chế quay nằm trong *hộp ổ đĩa* được bọc kín, còn mạch điều khiển điện tử được lắp trên bản mạch bên ngoài ổ đĩa. Tốc độ của môtơ quay đĩa là 3600 vòng/phút được ổn định bởi một mạch phản hồi cho phép một sai số cỡ ± 0,3% và ở những ổ đĩa chất lượng cao là cỡ ± 0,1%. Điều đó có nghĩa rằng các phần tử trữ tin trên đĩa cứng chuyển động tương đối so với một vị trí của đầu từ nhanh gấp 10 lần so với ổ đĩa mềm. Các đĩa cứng đơn giản nhất cũng có từ 17 sectơ /rãnh tới 70 sectơ / rãnh cho phép với sự tăng tốc độ quay như vậy, lượng dữ liệu đi qua đầu từ được tăng lên cỡ 80 lần so với con số 360 kbyte ở ổ đĩa mềm. Điều này đòi hỏi ổ đĩa cứng phải có các bộ phận điều khiển mạnh hơn.

Các ổ đĩa cứng thường có từ 2 đến 8 đĩa hai mặt, do vậy cần từ 4 đến 16 đầu từ. Tất cả các đầu từ này đều được gắn ở đầu của một cần truy xuất nằm trong hộp kín và do vậy cùng được chuyển động cố kết với nhau như hình 10.7 chỉ ra.

Giống như ổ đĩa mềm, trong ổ đĩa cứng thường chỉ có 1 đầu từ được kích hoạt cho việc đọc hoặc viết dữ liệu tại một thời điểm. Cần truy xuất được chuyển động hoặc bởi môtơ bước hoặc bởi môtơ tuyến tính. Các môtơ này được dùng để định vị đầu từ trên các track hoặc cylinder cần truy xuất. Hầu hết ổ đĩa cứng đều có một đèn LED chỉ thị khi quá trình đọc hoặc viết dữ liệu xảy ra. Các ổ đĩa cứng thường dùng các đĩa có kích thước 5 1/4" hoặc 3 1/2".



Hình 10.7 Cần truy xuất, đầu từ và các đĩa nằm trong một hộp kín.

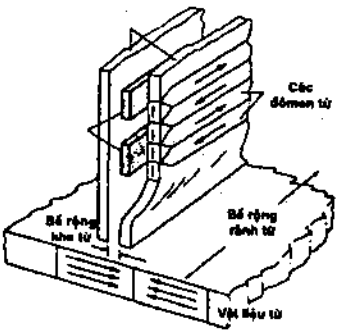
Các đĩa nhỏ hơn có ưu điểm là các đầu từ không cần chuyển dịch nhiều do đó thời gian truy xuất ngắn hơn. Ngày nay các hợp kim nhôm thường được dùng để làm đế mang đĩa vì nhẹ nhưng lại có độ ổn định về cơ học. Do đĩa nhẹ như vậy nên quán tính nhỏ và cho phép rút ngắn được thời gian đạt tới tốc độ danh định sau khi khởi phát hoạt động. Điều này đặc biệt quan trọng trong các ổ đĩa phải đóng cắt điện luôn luôn khi hoạt động trong chế độ tiết kiệm năng lượng như các ổ đĩa dùng trong máy tính xách tay (notebook). Môi trường ghi từ dùng để chứa dữ liệu là một lớp vật liệu có thể từ hoá được. Lớp đó phủ lên lớp đế mang kể trên và được chế tạo từ một chất cho một mật độ lưu trữ tin (các bit) cao hơn ở đĩa mềm nhiều. Thường các hợp chất của Cô-ban hoặc các gốm phe-rít được sử dụng. Việc phủ được thực hiện hoặc bằng công nghệ phủ dung dịch ferrite (ferrite coating) tạo được lớp mỏng cỡ 1 - 2μm hoặc bằng công nghệ điện phân (gavalnization methode) cho lớp mỏng cỡ 0,1 - 0,2μm. Ngày nay một số đĩa cứng chất lượng cao có lớp phủ từ được tạo bởi công nghệ phún xạ kim loại (sputtering) cho một lớp mỏng chỉ cỡ 0,05 đến 0,2 μm. Tiếp nữa, lớp phủ này lại được phủ bởi một lớp các-bon để bảo vệ các va chạm về cơ khí. Một đĩa cứng được chế tạo như vậy sẽ có bề mặt trông bóng sáng như gương.

Thông thường, các bit dữ liệu được ghi theo phương trùng với mặt phẳng đĩa, tức là các véctơ từ độ hoặc nằm dọc theo hướng của rãnh từ hoặc theo hướng ngược lại. Phương pháp ghi này có nhược điểm là mật độ ghi không cao. Vì vậy gần đây xuất hiện loại ghi từ dọc trong đó các véctơ từ độ có phương vuông góc với mặt phẳng đĩa như đã chỉ ra trên hình 10.3.

Ta đã biết chỉ có sự biến thiên của từ độ mới có thể tạo nên tín hiệu tại đầu từ đọc/viết. Sự biến thiên đó được đo bằng đơn vị FCI (flux changes per inch), tức là số điểm có sự biến đổi từ độ trên một đơn vị độ dài là inch. Tuy nhiên mật độ bit BPI (bit per inch) không chỉ phụ thuộc vào PCI mà còn phụ thuộc vào loại mã hoá dữ liệu. Các đĩa từ cứng được mã hoá theo phương pháp RLL (sẽ trình bày sau) có điểm mạnh do 1 bit có thể được mã hoá với ít hơn một lần biến đổi từ độ. Ngược lại phương pháp điều tần FM đòi hỏi 2 lần biến đổi từ độ cho 1 bit (1 cho dữ liệu và 1 cho xung đồng bộ). Do mật độ ghi tin rất cao so với đĩa mềm nên kĩ thuật chế tạo đầu từ đĩa cứng cũng đòi hỏi tinh tế hơn nhiều. Hình 11.7 cho thấy tất cả các đầu từ đều được gắn ở một đầu của cần truy xuất nên được chuyển động một cách cố kết với nhau. Các đầu từ càng nhỏ càng tạo được các ô bit có kích thước nhỏ. Một đầu từ nhỏ cũng có nghĩa là có trọng lượng nhỏ và do đó có quán tính nhỏ. Tức là các đầu từ như vậy sẽ cho phép chuyển động tới một vị trí mong muốn nào đó nhanh hơn. Để từ hoá hoặc phát hiện các vùng ghi từ nhỏ trên vùng mang dữ liệu, đầu từ càng ở gần mặt đĩa càng tốt. Hình 10.8 lí giải điều này. Do đầu từ ở quá xa như trường hợp hình bên trái nên từ trường được phát bởi đầu từ không tập trung được trên mặt đĩa như

Hình 10.8 Độ lớn của ô bit phụ thuộc vào khoảng cách và kích thước đầu từ.

trường hợp hình bên phải . Một cách gần đúng, có thể coi kích thước ngang nhỏ nhất của một ô bit bằng khoảng cách từ đầu từ đến mặt đĩa như hình bên phải. Tức là mật độ ghi tin có thể được tăng một cách đơn điệu khi giảm khoảng cách này. Vậy đầu từ càng nhỏ cũng như khoảng cách từ nó đến mặt đĩa càng nhỏ thì càng tốt. Tất nhiên lớp mang từ tính cũng phải có kích thước các domain không lớn hơn kích thước các ô bit này. Hiện nay có 2 công nghệ chính tạo ra các đầu từ: đầu từ ferrit và đầu từ màng mỏng (thin film). Đầu từ ferrit có giá thành thấp và có độ tin cậy do công nghệ chế tạo đã ổn định. Chúng phát ra từ trường bởi cuộn dây cuốn quanh lõi ferrit như được mô tả ở các đầu từ cho đĩa mềm. Đầu từ màng mỏng như hình 10.9 có kích thước và trọng lượng nhỏ hơn được chế tạo theo công nghệ vi cơ khí và vi điện tử như chế tạo các dụng cụ bán dẫn. Đó là việc phủ và ăn mòn để tạo nên các lớp bán dẫn khác nhau có những thuộc tính điện-từ nào đó thay thế cho cuộn dây và lõi ferrit, thí dụ như loại đầu từ đọc màng mỏng dùng hiệu ứng từ trở khổng lồ GMR (giant magnetic resistance). Các đầu từ màng mỏng này rất nhỏ và rất nhẹ nhưng lại có thể phát ra một từ trường hoặc thu được một tín hiệu điện đủ mạnh trong một khoảng cách giữa đầu từ và mặt đĩa còn nhỏ hơn so với đầu từ ferrite. Vậy đầu từ màng mỏng có hiệu suất cao hơn nhiều đầu từ ferrite.

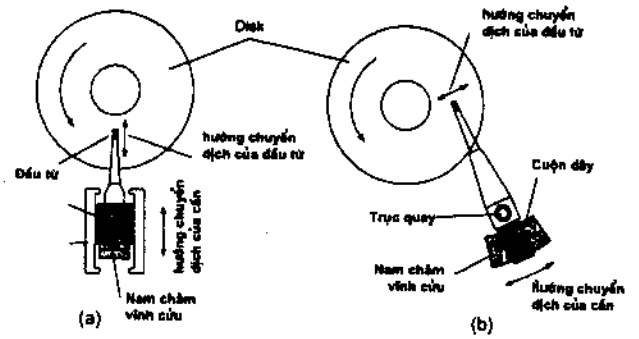


Hình 10.9 Đầu từ màng mỏng.

Khi ổ đĩa được cắt điện, đầu từ được "đỗ" lại trên một rãnh không dùng để trữ dữ liệu gọi là *landing zone*. Điều này tránh cho việc cọ sát vào các rãnh trữ dữ liệu khi tăng tốc cho đĩa quay lúc khởi động máy. Khi môtơ khởi phát quay đĩa, như một cái quạt gió, nó cuốn không khí xung quanh đĩa và tạo nên một dòng khí xoáy. Dòng khí này đủ lớn để phát ra một lớp đệm khí trên mặt đĩa đủ để nâng đầu từ lên khỏi mặt đĩa một khoảng cách rất nhỏ . Khoảng cách này là 1μm với đầu từ ferrit và từ 0,2 - 0,5 μm với đầu từ màng mỏng. Nó phụ thuộc chính vào hình dạng của đầu từ. Bề dày của lớp đệm không khí như vậy phụ thuộc vào áp suất không khí, áp suất càng thấp lớp đệm càng mỏng.

Cần truy xuất gồm tất cả các cấu kiện cơ khí cần thiết dùng cho chuyển động của các đầu từ. Hiện có thể dùng môtơ bước hoặc môtơ tuyến tính để chuyển dịch cần này. Môtơ bước trong đĩa cứng cũng được quay từng bước một bởi các xung bước (mỗi bước là một giá trị góc quay nào đó). Chuyển động quay của môtơ này qua các chi tiết cơ khí được biến thành chuyển động tịnh tiến của cần truy xuất theo phương bán kính của đĩa để định vị các đầu từ tới vị trí mong muốn. Thường một bước quay của môtơ được thiết kế tương ứng với một chuyển dịch của đầu từ đi một rãnh từ trên đĩa. Thường khi dùng môtơ bước thì không cần mạch điều khiển phản hồi. Do vậy, thí dụ nếu cần chuyển đầu từ từ rãnh 0 đến rãnh 219 thì bộ điều khiển chỉ đơn giản cấp 219 xung tới môtơ bước là xong. Với đĩa mềm có bề rộng rãnh cỡ 0,1 mm cách xử lí này là đủ độ chính xác này cho phép định vị đầu từ tới giữa rãnh. Nhưng với những đĩa cứng kích thước 3 1/2" với hơn 1000 rãnh thì việc sử dụng môtơ bước như vậy không đảm bảo độ chính xác khi định vị đầu từ. Để định vị chính xác đầu từ, không chỉ độ chính xác mà cả độ lặp lại phải cao. Việc nối ghép giữa trục môtơ bước với cần truy xuất được thực hiện bằng ma sát giữa trục và một phần tử cơ khí hình vòng cung. Do đó có thể xảy ra trường hợp bị trượt giữa hai bộ phận này đặc biệt trong quá trình tăng tốc và một sai lệch khi định vị sẽ xảy ra. Điều này có thể hiệu chỉnh được bằng việc đưa cần truy xuất về vị trí tương ứng với rãnh 0. Do đó một lệnh

recalibrate drive được sử dụng. Khi về tới đó, bộ điều khiển lại bắt đầu chuyển dịch cần truy xuất và đếm các xung bước. Một vấn đề khác là sự biến dạng nhiệt liên quan tới việc sử dụng môtơ bước. Giả sử rằng ta định dạng đĩa ngay sau khi khởi động máy tính. Lúc đó tất cả các linh kiện còn đang lạnh. Sau khi máy tính hoạt động được vài giờ thì đột nhiên xảy ra sai số đọc/viết. Nguyên do là sự biến dạng nhiệt của các linh kiện cơ khí, thí dụ như cần truy xuất bị dài ra do nhiệt và do vậy lượng dịch chuyển của nó coi như dài hơn tí chút và có thể gây nên lỗi định vị trên các rãnh từ. Tóm lại các sai số này xảy ra do không có một sự phản hồi về vị trí thực sự của đầu từ và lượng điều khiển của môtơ để mà hiệu chỉnh. Các ổ đĩa cứng hiện đại lại quay về sử dụng môtơ tuyến tính thông thường thay cho môtơ bước, chỉ có điều thêm vào đó một mạch điện tử cho phép so sánh vị trí hiện tại của đầu từ với vị trí cần định vị để hiệu chỉnh và do đó đầu từ luôn được định vị chính xác. Hiện có 2 loại cần truy xuất dùng mô tơ tuyến tính được sử dụng như hình 10.10: loại tuyến tính (a) và loại quay (b). Trong loại tuyến tính, dưới tác dụng của dòng điện điều khiển chảy qua cuộn dây, cần truy xuất có gắn nam châm vĩnh cửu sẽ chuyển động vào ra trên khung dẫn hướng và tạo nên chuyển dịch theo trục đĩa từ.

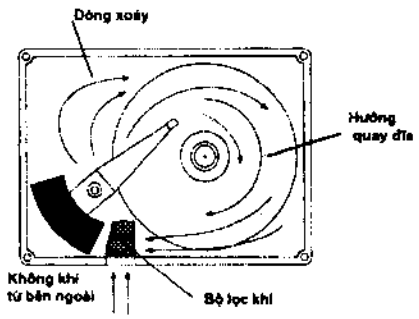


Hình 10.10 2 loại cần truy xuất: tuyến tính (a) và quay (b).

Trong loại quay, cần truy xuất có gắn nam châm hình vòng cung sẽ chuyển động quay quanh trục quay. Vì cánh tay đòn bên phía đầu từ dài hơn nên một dịch chuyển nhỏ (cũng có nghĩa là dòng chảy qua cuộn dây nhỏ) của nam châm cũng dẫn tới một dịch chuyển lớn của đầu từ theo phương bán kính của đĩa. Vậy phương pháp quay cho một tốc độ định vị đầu từ nhanh hơn với công suất tiêu thụ nhỏ hơn. Môtơ tuyến tính có một nhược điểm so với môtơ bước là việc định vị không phải là tuyệt đối. Một xung ngắn trong cuộn dây sẽ chuyển dịch cần truy xuất nhưng không thể dự đoán trước một cách chính xác xung đó như thế nào. Hơn nữa các từ trường được phát bởi cuộn dây sẽ bị thay đổi dần theo sự già hoá của nam châm vĩnh cửu. Vì vậy phải chọn một chiến lược khác là tạo một vòng phản hồi tích cực luôn dẫn tới một vị trí tuyệt đối của đầu từ và cần truy xuất. Điều này được thực hiện bởi cái gọi là các *rãnh chỉ số* (index track) và *đầu từ chỉ số* (index head) được dùng để chuẩn trong quá trình định vị đầu từ. Các rãnh chỉ số được nhà sản xuất tạo ra ở tại một mặt đĩa trong chồng đĩa cùng với đầu từ của mặt đó gọi là đầu từ chỉ số không dùng cho ghi/đọc dữ liệu mà chỉ được dùng để định vị các đầu từ khác. Khi ổ đĩa định thâm nhập tới một rãnh xác định thì đầu từ chỉ số được định vị trên rãnh chỉ số liên quan. Như vậy tất cả các đầu từ khác dùng cho việc ghi dữ liệu cũng được định vị trên rãnh này. Người dùng không thể thay đổi các rãnh chỉ số. Các chương trình định dạng đĩa cấp thấp được định hướng theo các rãnh chỉ số này và tạo ra các rãnh dữ liệu trên khắp đĩa ở cùng vị trí bán kính như các rãnh chỉ số. Nếu ổ đĩa định đưa đầu từ tới một rãnh nào đó thì

điều khiển ổ đĩa sẽ cấp năng lượng cho môtơ tuyến tính và đầu từ chỉ số phát hiện số của rãnh đó. Khi đầu từ chỉ số tới rãnh mong muốn, ổ đĩa sẽ dừng quá trình định vị thô này và tiến hành quá trình định vị tinh. Đầu từ chỉ số sẽ được dịch chuyển tới lui từng bước nhỏ bởi cuộn dây môtơ được cấp những xung dòng nhỏ cho tới khi tín hiệu nhận được từ rãnh chỉ số là cực đại. Lúc này đầu từ được định vị chính xác ở giữa rãnh cần thiết. Rõ ràng việc định vị bằng các rãnh chỉ số và đầu từ chỉ số chính xác hơn và không bị ảnh hưởng bởi nhiệt độ. Tuy nhiên mạch điện tử dùng cho phản hồi sẽ phức tạp hơn và việc tạo chính xác các rãnh chỉ số là phức tạp. Các ổ đĩa loại này đắt tiền hơn loại dùng môtơ bước.

Vấn đề lọc khí và thông khí, tất cả các ổ đĩa cứng đều có một bộ lọc khí. Bộ lọc này không dùng cho lọc khí từ bên ngoài xâm nhập vào ổ đĩa mà chủ yếu dùng để lọc khí đã ở bên trong hộp ổ đĩa. Nếu áp suất không khí ở môi trường thay đổi thì chỉ xảy ra một sự trao đổi khí nhỏ mà thôi. Vì vậy hầu như không phải thay bộ lọc khí trong suốt thời gian hoạt động của ổ đĩa. Hình 10.11 chỉ ra sơ đồ của một vòng không khí xoáy bên trong hộp ổ đĩa. Mỗi ổ đĩa đều có một khe thông khí dẫn tới bộ lọc khí bên trong. Tuỳ vào áp suất khí của môi trường mà không khí có thể thấm vào hộp ổ đĩa hoặc thoát ra từ đó.



Hình 10.11 Dòng xoáy không khí trong hộp ổ đĩa cứng.

2. Ghép nối bộ điều khiển và ổ đĩa cứng

Mạch ghép nối bus xác lập khâu nối giữa mạch điều khiển với PC. Các dữ liệu đọc/viết cho ổ đĩa cũng như các dữ liệu lệnh cho bộ điều khiển được trao đổi qua mạch ghép nối bus này.

Một chương trình máy trong ROM vi mã được một vi xử lý chuyên dụng trên bộ điều khiển sử dụng. Dữ liệu ghi/đọc trên sectơ được giữ bởi bộ đệm sectơ. Bộ đệm này luôn trữ đủ cho nội dung toàn bộ của 1 sectơ. Mạch ghép nối ổ đĩa được thiết kế rất khác nhau tuỳ thuộc chúng là loại ST412/506, ESDI hay điều khiển chuyên dụng. Mạch điều khiển chuyên dụng được tích hợp ngay trên ổ đĩa và chỉ dùng cho riêng ổ đĩa đó mà thôi.

Để đọc một sectơ, CPU truyền một khối lệnh (gồm các lệnh và các byte thông số) tới một thanh ghi trong mạch ghép nối PC. Vi xử lí của bộ điều khiển phát hiện các lệnh này và dịch mã lệnh. Bộ điều khiển thực hiện lệnh thích hợp (đây là việc đọc 1 sectơ) dưới sự điều khiển của chương trình trong ROM vi mã. Nó sẽ điều khiển ổ đĩa định vị đầu từ trên rãnh bằng các xung cấp tới môtơ. Đầu từ sẽ liên tục đọc các tín hiệu phát ra từ các ô bit đi qua dưới nó, truyền các tín hiệu phù hợp qua ghép nối ổ đĩa tới bộ phân tách dữ liệu. Tuỳ thuộc vào phương pháp ghi, bộ phân tách sẽ biến đổi các dữ liệu được lập mã MFM hay RLL thành các mã trung gian NRZ (non-return to zero). Vi xử lí sẽ xác định các trường địa chỉ nhận dạng và dữ liệu ... đã đi qua đầu từ. Nếu nó phát hiện được dấu địa chỉ mong muốn thì nó sẽ xác định xem có đúng là rãnh và sectơ cần thiết không. Nếu không đúng nó sẽ hướng dẫn ổ đĩa tiến hành việc định vị khác và đợi cho tới dấu địa chỉ tiếp sau. Nếu số sectơ đúng bằng số mong muốn, bộ điều khiển sẽ mở cổng đọc và chấp nhận dữ liệu đã đọc từ đĩa. Số liệu được giải mã sẽ được truyền vào bộ đệm sectơ.

Cùng lúc đó mạch sửa lỗi ECC (error correcting code) kiểm tra liệu rằng các dữ liệu của sectơ có phù hợp với các byte ECC hay không. Nếu không thì dữ liệu được sửa bởi các byte này. Các byte ECC này không chỉ phát hiện được các nhóm lỗi mà còn được dùng để sửa chúng khi các lỗi không quá nghiêm trọng. Nguyên lý của mã sửa lỗi ECC cũng tương tự như mã kiểm tra

tổng RCR nhưng đa thức sinh được sử dụng có phức tạp hơn. Một bộ điều khiển đĩa cứng ST412 thường sử dụng một mã ECC 4 byte với đa thức sinh:

$$ECC = x^{32} + x^{28} + x^{26} + x^{19} + x^{17} + x^{10} + x^{6} + x^{2} + 1.$$

Dữ liệu sau đó được truyền vào bộ nhớ chính. Bộ điều khiển XT dùng DMA kênh 3 cho mục đích này trong khi bộ điều khiển trên máy AT dùng quá trình trao đổi dữ liệu bằng ngắt với các lệnh IN và OUT. Nó sẽ phát ra một ngắt cứng qua IRQ14 sau khi sectơ đã được đọc vào bộ đệm sectơ và rồi chương trình ngắt sẽ truyền tất cả 512 byte dữ liệu được đọc từ bộ đệm sectơ vào bộ nhớ chính qua chương trình vào/ra. Khi viết một sectơ thì xử lí được tiến hành ngược lại. Đầu tiên 512 byte dữ liệu được viết và truyền từ bộ nhớ chính tới bộ đệm sectơ. Bộ điều khiển sẽ tiến hành định vị đầu từ và tìm rãnh và sectơ cần thiết như trường hợp đọc. Nếu vi xử lí phát hiện được dấu địa chỉ của sectơ mong muốn từ đầu từ, nó sẽ mở cổng viết và liên tục truyền dữ liệu qua bộ điều khiển lưu trữ. Bộ điều khiển này sẽ phát ra các dấu địa chỉ dữ liệu và các thông tin về khe phân cách, tính các byte ECC, biến đổi dữ liệu của sectơ thành mã NRZ và truyền dòng dữ liệu này tới bộ phân tách dữ liệu. Tại đây việc lập mã MFM hoặc RLL được tiến hành và bộ này sẽ truyền các tín hiệu đó tới ghép nối ổ đĩa và tới đầu từ đã được kích hoạt để ghi lên đĩa. Khi các dữ liệu đã được ghi xong, vi xử lí sẽ phát hiện và xử lí lệnh tiếp theo. Lí do trong các máy AT với các vi xử lí từ 286 trở đi không dùng phương thức trao đổi DMA nữa là vì việc thực hiện các lệnh vào/ra IN, OUT, MOV nhanh hơn tốc độ xử lí của chip DMA cũ.

3. Dung lượng đĩa và tốc độ truyền dữ liệu

- *Dung lượng trữ tin* của đĩa sau khi đã định dạng thường nhỏ hơn tới 20% trước khi định dạng vì rằng phải mất một số byte cho các trường địa chỉ, các khe phân cách v.v... Đơn vị chỉ thị dung lượng thường là *Mbyte* nhưng đôi khi có thể là *miobyte* (million byte) nhỏ hơn Mbyte là 5%.

- *Thời gian truy xuất dữ liệu* (data access time) của ổ đĩa là một thông số quan trọng ảnh hưởng mạnh mẽ đến việc xử lí số liệu trong PC, thời gian này là tổng hợp của *thời gian tìm kiếm* (seek time), *thời gian chuyển đầu từ* (head switch time) và *thời gian quay trễ* (rotational latency). Thời gian tìm kiếm là thời gian chuyển đầu từ từ một track này sang track khác; thời gian tìm kiếm chậm nhất ứng với hai track trong và ngoài cùng của đĩa; thời gian tìm kiếm trung bình là bằng thời gian cần có để đặt đầu từ tới một vị trí được yêu cầu ngẫu nhiên. Thời gian chuyển đầu từ là thời gian ổ đĩa phải bỏ ra để chuyển giữa hai trong số các đầu từ khi đọc hay viết dữ liệu. Thời gian quay trễ là thời gian tính từ khi đầu từ được đặt lên một track đến khi nó tới được sector mong muốn. Tất cả các thông số trên đều được đo bằng mili giây (ms). Ổ đĩa có thời gian truy xuất trên 40 ms là loại chậm, từ 28 ms đến 40 ms là trung bình, từ 18 ms đến 28 ms là loại nhanh, dưới 18 ms là loại cực nhanh.

Ba nhân tố chính quyết định đến *tốc độ truyền dữ liệu của ổ đĩa* là sự truyền tín hiệu giữa đĩa và bộ điều khiển, xử lí trong bộ điều khiển và việc truyền tín hiệu giữa bộ điều khiển và bộ nhớ chính. Nhân tố một bị hạn chế bởi số byte chuyển qua đầu từ trong mỗi giây. Với đĩa có 17 sectơ/rãnh tại tốc độ quay 3600 vòng/phút, giá trị này cỡ 600 kbyte/giây tương đương với tốc độ danh định 5Mbits/giây của ghép nối ST412 với mã MFM. Bộ điều khiển sẽ tách tất cả các byte không thuộc trường dữ liệu và như vậy chỉ còn lại 510 kbyte. Số kbyte được đi qua đầu từ trong 1 giây này chỉ cho một sectơ riêng biệt; ta phải viết 512 byte / 980μs vì một sectơ gồm 512 byte dữ liệu cùng các trường địa chỉ, khe ... đi qua đầu từ cỡ 980μs. Tiếp đó dữ liệu nằm trong bộ đệm séctơ và phải được truyền tiếp nữa vào bộ nhớ chính. Nếu bộ điều khiển và hệ thống nhớ đủ nhanh để thực hiện việc này trước khi sectơ tiếp theo xuất hiện (với hệ số đan xen của đĩa

bằng 1:1), thì tốc độ truyền này giữ nguyên trong toàn rãnh bằng tốc độ hiệu dụng là 510 kbyte/giây. Thời gian xử lí trong bộ điều khiển với kiểm tra CRC và ECC thường rất ngắn nên có thể bỏ qua nhưng tốc độ truyền giữa bộ điều khiển và bộ nhớ chính qua bus hệ thống PC trong khoảng thời gian ngắn đến khi sectơ tiếp theo xuất hiện là một vấn đề. Khoảng thời gian giữa hai trường dữ liệu của các sectơ kế tiếp cỡ 100μs. Như vậy tốc độ truyền vào bộ nhớ phải tương đương 512 byte/100μs hoặc chừng 5 Mbyte /giây.! Vậy là bus hệ thống của PC phải có khả năng đảm bảo một tốc độ ít nhất như vậy để có hệ số đan xen 1:1. Bus ISA chỉ có tốc độ cực đại là 8,33 MHz hay tốc độ truyền dữ liệu là 16,67 Mbyte/giây. Chỉ có mode burst của bus EISA hoặc mode DMA có tốc độ truyền là 33,3 Mbyte/giây. Do đó dữ liệu được lập mã RLL với ghép nối ST412 và đĩa 26 sectơ /rãnh sẽ làm cạn kiệt dung lượng của bus ISA. Do vậy, các ổ đĩa dung lượng cao tới 70 sectơ/rãnh và hệ số đan xen bằng 1:1 đều có một đĩa cache (hoặc ít nhất một rãnh thay cho bộ đệm sectơ). Cache hoạt động nhanh hơn việc truyền qua bus ISA hoặc EISA và có thể chứa được toàn bộ nội dung của một rãnh. Sự truyền vào bộ nhớ chính qua bus PC có thể được thực hiện một cách liên tục và dung lượng truyền 8,33 Mbyte/giây hoặc 16,67 Mbyte/giây có thể được sử dụng cho thời gian dài hơn 100μs giữa 2 sectơ trong trường hợp đĩa được lập mã MFM với ghép nối ST412.

Các chương trình Benchmark cho phép xác định tốc độ truyền dữ liệu bằng cách đọc lặp lại cùng một khối dữ liệu.

4. Định dạng cấp thấp và cấp cao

Đĩa cứng phải được định dạng cấp thấp (low-level format) trước khi tạo phân khu và định dạng cấp cao (high-level format). Đó là việc tạo ra những rãnh và sectơ cùng các thông tin liên quan đến chúng trên đĩa. Một rãnh đĩa cứng chứa các thông tin điển hình như thí dụ sau:

Định dạng kiểu MFM	**Định dạng kiểu RLL**
Phần mở đầu track	*Phần mở đầu track*
10 byte đồng bộ (SYNC): 00h	11 byte đồng bộ (SYNC): 00h
2 byte đánh dấu địa chỉ chỉ số (IAM): FCh	2 byte đánh dấu IAM: A1h, FCh
11 byte khe phân cách1 (GAP1): 4Eh	12 byte khe phân cách1 (GAP1): FFh
Sector	*Sector*
7 byte SPD: 4Eh	
10 byte đồng bộ (SYNC): 00h	10 byte đồng bộ (SYNC): 00h
2 byte dấu địa chỉ nhận dạng (IDAM): A1h, FEh	2 byte dấu IDAM: 5Eh, A1h
4 byte nhận dạng (ID): cyli., head, size, flag *	4 byte nhận dạng (ID): cylin., head, size, flag*
4 byte mã ECC	4 byte mã ECC
5 byte khe phân cách 2 (GAP2): 00h	5 byte khe phân cách 2 (GAP2): 00h
10 byte đồng bộ (SYNC): 00h	11 byte đồng bộ (SYNC): 00h
2 byte đánh dấu bắt đầu vùng lưu trữ dữ liệu (DAM): A1h, F8h	2 byte đánh dấu DAM: 5Eh, A1h
512 byte lưu trữ dữ liệu (DATA)	512 byte lưu trữ dữ liệu (DATA)
4 byte mã ECC	4 byte mã ECC
15 byte khe phân cách 3 (GAP3): 00h	3 byte 00h và 17 byte FFh làm khe phân cách 3 (GAP3)
Kết thúc rãnh (EOT)	*Kết thúc rãnh (EOT)*
56 byte 00h khe phân cách (GAP 4): FFh	93 byte 00h khe phân cách (GAP4)

* 4 byte xác định: cylinder, đầu từ, sector và cỡ sector.

Trong quá trình định dạng cấp thấp, chương trình cũng đánh dấu các sectơ hỏng để DOS không sử dụng đến nữa cùng việc phân định hệ số đan xen. Đó là việc đánh số các sectơ lôgic

liên tiếp không theo các tuần tự liên tiếp nhau mà cách ra n sectơ nếu thời gian xử lí dữ liệu của hệ thống cho 1 sectơ dài hơn thời gian phân cách giữa 2 sectơ liên tiếp. Với các máy tính hiện nay có tốc độ xử lí của hệ thống nhanh thì hệ số đan xen bằng 1 (n 0 1), tức là không có đan xen. Khác với đĩa mềm, lệnh định dạng cấp cao FORMAT trong đĩa cứng chỉ xác định cấu trúc của các phân khu đĩa. Nó tạo ra các sectơ khởi động, hai bảng FAT và thư mục gốc. Tất cả các điểm vào của FAT được điền số 00h để chỉ rằng phân khu còn trống.

Khi định dạng lại một đĩa cứng thì chỉ có cấu trúc lôgic bị xoá (giống như lệnh DEL hay RMDIR) chứ về mặt vật lí các dữ liệu vẫn được giữ nguyên.

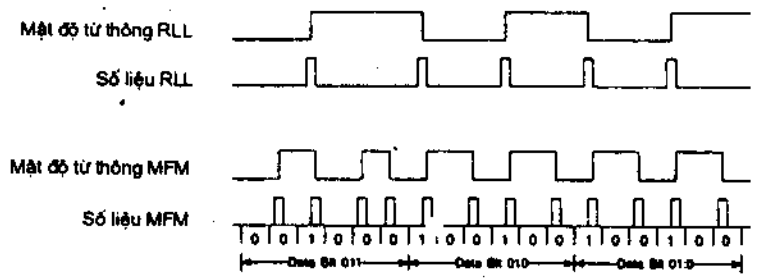
5. Định dạng RLL ghi trên đĩa từ cứng

Các bit thông tin được ghi trên đĩa cứng theo mã RLL (run length limited). Mã này được lập trên cơ sở quy định số bit 0 giữa 2 nhóm bit được hạn chế. Phương pháp ghi theo mã RLL không đòi hỏi phải ghi các xung bit nhịp. Mạch lôgic giải mã RLL sẽ tự xác định trong khoảng thời gian giữa 2 nhóm bit có bao nhiêu bit 0 được xen vào. Chính do vấn đề đồng bộ, sự biến đổi của tốc độ quay đĩa v.v.. sẽ ảnh hưởng đến chu kỳ giữa các bit 0 nên mã này chỉ hạn chế số bit 0 là vậy. Loại mã RLL được sử dụng rộng rãi là $RLL_{2,7}$. Tức là có ít nhất là 2 và nhiều nhất là 7 bit 0 giữa 2 nhóm bit được ghi. Bảng bên chỉ ra cách lập mã $RLL_{2,7}$. Nhìn vào đây ta thấy số bit 0 nhiều nhất đứng trước bit 1 là 4 và sau là 3 và như vậy nhiều nhất là có 7 bit 0 trong một nhóm bit. Hình 10.12 chỉ ra kết quả ghi mã RLL của byte 0110 1001 như thí dụ với phương pháp FM và MFM. Giả thử bit đầu tiên của byte tiếp theo là 0. Căn cứ vào bảng lập mã bên ta sẽ có mã RLL là:

Bit số liệu	Mã $RLL_{2,7}$
000	000100
10	0100
010	100100
0010	00100100
11	1000
011	001000
0011	00001000

011 010 010
↓ ↓ ↓
001000 100100 100100.

Chỉ có 7 bit có giá trị 1 được ghi, các bit 0 sẽ được phát bởi lôgic RLL dựa vào thời gian trễ giữa 2 nhóm bit. So với 9 bit số liệu cần ghi trong phương pháp MFM phải có 12 lần thay đổi từ thông thì phương pháp RLL chỉ cần 5 lần. Tổng quát phương pháp MFM cần gấp 3 lần thay đổi từ thông so với RLL. Nhưng mã RLL có độ dài bit lớn gấp 2 lần nên mật độ nén so với MFM chỉ còn 3/2 tức là số dữ liệu được trữ trên đĩa tăng hơn 50% so với MFM. Từ mã RLL gồm từ 4 đến 8 bit trong khi từ mã MFM dài cố định. Nếu trong từ mã RLL chỉ sai 1 bit sẽ dẫn đến sai các bit còn lại. Vì vậy lôgic



Hình 10.12 Phương pháp lập mã $RLL_{2,7}$ so với MFM.

phát hiện và sửa lỗi ECC cho mã này cần phải mạnh hơn. Trong các bộ điều khiển RLL chất lượng cao, mã ECC gồm 6 chứ không phải 4 byte như thường lệ. Ngoài ra còn có các phương

pháp lập mã ARLL (advanced RLL) như $RLL_{1,7}$ và $RLL_{3,9}$. Cả hai trường hợp này, số bit dư ra (overhead) lớn hơn nhưng mật độ ghi được tăng lên đến 90% so với MFM.

6. Chiến lược hồi phục lỗi

Vì mật độ ghi dữ liệu cực kì cao nên ở đĩa cứng thường xuyên xảy ra việc đọc và tìm kiếm lỗi. Các xử lí này khác nhau tuỳ vào lỗi xảy ra trong khi đọc dữ liệu hay trường địa chỉ của sectơ. Bộ điều khiển không thể phát hiện lỗi định vị một cách tuyệt đối mà chỉ bởi sự so sánh các điểm vào trong trường ID của sectơ với trường ID cần tìm. Nếu hai số cylinder không trùng nhau thì có nghĩa có lỗi định vị. Đối với các lỗi trong vùng dữ liệu cũng xử lí như vậy. Thí dụ, nếu do va chạm của đầu từ, một vùng sectơ bị hỏng thì bộ điều khiển không thể phát hiện nguyên nhân này một cách trực tiếp. Do vậy, nó sẽ chấp nhận các tín hiệu tiếp theo từ đầu từ và giải mã chúng. Từ phần bị hỏng nó chỉ nhận được thông tin sai lạc. Khi sử dụng các byte ECC, bộ điều khiển có thể tổ chức lại phần sectơ chứa các dữ liệu không hợp lệ. Nếu bộ điều khiển phát hiện một lỗi trong vùng dữ liệu vì một vài sai khác giữa số liệu sectơ và các byte ECC, nó sẽ xử lí như các cách dưới đây. Nếu ý đồ thành công (tức là sectơ đó lại có thể được đọc mà không xảy ra lỗi) thì bộ điều khiển sẽ kết thúc xử lí và truyền dữ liệu vào bộ nhớ chính.

- Đọc sectơ 8 lần không sửa bằng ECC. Nếu lí do lỗi là do vấn đề đồng bộ hay do tốc độ quay thì trong hầu hết trường hợp chỉ đọc lại một lần là thành công.
- Chuyển đầu từ về rãnh 0, sau đó tới rãnh có số cylinder cực đại rồi cuối cùng tới rãnh mong muốn. Nếu lỗi do định vị sai một chút của đầu từ (nằm ở không chính giữa rãnh làm cho tín hiệu yếu) thì có thể khắc phục được.
- Đọc lại sectơ và tiến hành sửa ECC.
- Đọc sectơ 6 lần bằng cách dịch cửa sổ xung nhịp và dữ liệu đi +1,5%, +6%, +7,5%, -1,5%, -6% và 7,5%. các lỗi do dịch bit vì mất bù trừ trước (precompensation) có thể hồi phục được theo cách này.
- Bộ điều khiển chuyển dịch đầu từ theo các bước nhỏ (vi dịch chuyển) và đọc sectơ 8 lần không có sửa ECC. Do vậy các lỗi do sai lệch nhỏ trên rãnh so với vị trí ban đầu do biến dạng (thí dụ vì nhiệt) có thể được khắc phục.
- Bộ điều khiển chuyển dịch đầu từ theo các bước nhỏ (vi dịch chuyển) và đọc sectơ 8 lần có sửa ECC.

Như vậy, nếu không sửa được lỗi, bộ điều khiển đã phải trải qua 6 bước kể trên với 31 lần thâm nhập đọc.

7. Tự động cấu hình

Khi lắp một đĩa cứng mới vào PC phải khai báo lại các thông số của ổ đĩa trong chương trình SETUP của BIOS. Trong các đĩa cứng hiện nay, nhà sản xuất đã lưu trữ các thông tin này trong cylinder vật lí thứ nhất, tại các sectơ 1 và 2 của rãnh 0 và 1 trong dạng 4 bản copy như nhau. Bảng sau cho cấu trúc của 4 sectơ chứa các thông số hình học của đĩa.

Byte	Nội dung
0 - 1	Chữ kí 0DABEh về thông tin hình học đĩa
2 - 3	Số cylinder
4	Số đầu từ
5	Số sectơ trên một rãnh
6 - 7	Dự trữ
8	Hệ số đan xen
9	Byte của BIOS

10	Số cylinder cho các rãnh khác
11	Dự trữ
12 - 13	Cylinder khởi phát của xử lí precompensation
14 - 20	Tên nhà sản xuất (mã ASCII)
21 - 39	Tên sản phẩm
40 - 511	Dự trữ

Do đường kính của các rãnh bên ngoài lớn hơn bên trong nên về nguyên tắc các cylinder có số thấp hơn có thể chứa nhiều sectơ hơn. Các ổ đĩa có bộ điều khiển của riêng mình và nó biết rãnh nào chứa bao nhiêu sectơ... Trong khi đó qua BIOS người dùng chỉ biết số sectơ trên một rãnh là cố định. Vậy, bộ điều khiển phải biến đổi các giá trị được chỉ định cho đầu từ, cylinder, sectơ trong chương trình BIOS thành các giá trị thực sự cho đầu từ, cylinder, sectơ trên đĩa cứng mà không cần sự can thiệp từ bên ngoài. Sự xử lí này được hoàn toàn dấu kín với người sử dụng. Qúa trình dịch này (translation) thường không thể can thiệp được thậm chí ở mức thanh ghi. Thực tế, đĩa cứng thường được chia thành một vùng bên trong và một vùng bên ngoài, mỗi vùng có một số sectơ cố định trên một rãnh. Như vậy số sectơ trên một rãnh chỉ thay đổi từ vùng này tới vùng kia. Việc này gọi là ghi theo vùng (zone recording).

8. Thâm nhập đĩa cứng qua DOS và BIOS

Giống như đĩa mềm, có thể thâm nhập đĩa cứng qua ngắt 25h và 26h để đọc hoặc viết các sectơ lôgic trong các phân khu. Với các phân khu có số sectơ nhỏ hơn 65536 thì các giá trị gọi và trả về giống như với đĩa mềm, với các phân khu có số sectơ lớn hơn thì có khác.

Phân khu lên tới 32 Mbyte hoặc 65536 sectơ

INT 25h Đọc 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	Số ổ đĩa (0:A, 1: B)	
AX		Mã lỗi (bảng dưới)
CX	Số sectơ	
DX	Sectơ đầu tiên	
BX	Offset của bộ đệm đọc	
DS	Segment bộ đệm đọc	Lỗi nếu <> 0
Carry		

INT 26h Viết 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	Số ổ đĩa (0:A, 1: B)	
AX		Mã lỗi (bảng dưới)
CX	Số sectơ	
DX	Sectơ đầu tiên	
BX	Offset của bộ đệm viết	
DS	Segment bộ đệm viết	Lỗi nếu <> 0
carry		

Phân khu lớn hơn 32 Mbyte hay có hơn 65536 sectơ

INT 25h Đọc 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	Số ổ đĩa (0:A, 1: B)	
AX		Mã lỗi (bảng dưới).
CX	0FFFFh	
DX		
BX	Tham số offset [1]	
DS	Tham số segment [1]	
Carry		Lỗi nếu <> 0

INT 26h Viết 1 hoặc nhiều sectơ lôgic

Thanh ghi	Giá trị gọi	Giá trị trả về
AL	Số ổ đĩa (0:A, 1: B)	
AX		Mã lỗi (bảng dưới).
CX	0FFFFh	
DX		
BX	Tham số offset [1]	
DS	Tham số segment [1]	
carry		Lỗi nếu <> 0

1) Khối tham số như sau:

Sectơ đầu tiên: 4 byte; Số của sectơ: 2 byte; Địa chỉ của bộ đệm đọc/viết: 4byte.

Bảng mã lỗi như sau:

Mã	Lỗi
01h	Lệnh không hợp lệ
02h	Trường địa chỉ không hợp lệ
04h	Không tìm thấy sectơ
08h	Tràn DMA
10h	Lỗi CRC hoặc ECC
20h	Lỗi ở bộ điều khiển
40h	Lỗi tìm kiếm
80h	Ổ đĩa không sẵn sàng

Thí dụ, đọc 3 sectơ bắt đầu từ sectơ số 189 063 tại đĩa cứng C.

```
type parm_block {
  unsigned long start_sector;
  unsigned num_of_sector;
  char  far *buffer;
}
main ( )
{
union REGS inregs, outregs;
struct SREGS  segregs;
struct  far parm_block p_block;
                                                    /* khối tham số construct */
p_block.start_sector = 189063;
p_block.num_of_sector = 3;
p_block.buffer = (char far *)_fmalloc (1536);       /* đệm cho 3 sectơ */
                                                    /* gọi hàm */
inregs.h.al = 0x02;                                 /* ổ C */
inregs.x.cx = 0xffff;                               /* định dạng gọi phần mở rộng */
inregs.x.bx = FP_OFF(p_block);                      /* truyền tham số khối offset vào bx */
segregs.ds = FP_SEG(p_block);                       /*truyền tham số khối segment vào ds */
int86x(0x25,&inregs, &outregs, &segregs);           /*gọi ngắt, đọc các sectơ */
if ((outregs.x.cflag & 0x01) == 0x01 {              /* kiểm tra xem cờ được đặt ? */
printf("\ nError code: %x", outregs.x.ax);          /* hiện mã lỗi */
exit (1);
}
.....                                               /* xử lí đọc các sectơ */
exit (0);
}
```

Ngắt 13h trong BIOS có các hàm sau:

- Hàm 05h, định dạng rãnh hoặc cylinder.
- Hàm 06h, định dạng và đánh dấu các rãnh xấu.
- Hàm 07h, định dạng và đánh dấu ổ đĩa.
- Hàm 08h, xác định các tham số ổ đĩa.
- Hàm 09h, đặt các tham số ổ đĩa.

- Hàm 0Ah, đọc các sectơ mở rộng.
- Hàm 0Bh, viết các sectơ mở rộng.
- Hàm 0Ch, tìm kiếm.
- Hàm 0Dh, reset đĩa cứng.
- Hàm 0Eh, đọc bộ đệm sectơ.
- Hàm 0Fh, viết bộ đệm sectơ.
- Hàm 10h, kiểm tra ổ đĩa đã sẵn sàng chưa.
- Hàm 11h, chuẩn lại ổ đĩa.
- Hàm 19h, đỗ (park) đầu từ.

9. Ghép nối CPU với ổ đĩa qua chuẩn IDE

Chuẩn IDE (intelligent drive electronics hay integrated drive electronics) ra đời thay thế cho chuẩn ghép nối cũ ST412/506.

Ghép nối IDE tích hợp cả ổ đĩa và bộ điều khiển vào cùng một khối trên cơ sở gắn trực tiếp chip điều khiển ST506 vào ổ đĩa cứng và nối ổ đĩa tới bản mạch chính bằng các mạch điện đơn giản. Việc nối giữa bus AT và mạch ghép nối IDE của bộ điều khiển trên các ổ đĩa cứng được thực hiện bởi bản mạch gọi là *host-adapter* cắm vào một khe cắm mở rộng. Bản mạch chính giữ vai trò chủ (host), *host-adapter* chỉ cung cấp một số bộ đệm và mạch giải mã dùng cho việc nối ổ đĩa IDE và bus hệ thống AT. Một cáp dẫn bẹt 40 sợi cho phép nối *host-adapter* với các ổ đĩa. Độ dài cực đại của cáp là 18 inch (46 cm). Ghép nối IDE có thể phục vụ nhiều nhất là hai ổ đĩa (một là chủ và một là tớ, ổ đĩa chủ được gán địa chỉ 0, ổ đĩa tớ được gán địa chỉ 1). Một số ổ đĩa IDE được trang bị thêm bộ nhớ cache cho ít nhất hai track nhằm làm giảm thời gian thâm nhập trung bình của ổ đĩa. Dưới đây liệt kê các dây IDE và chức năng của chúng:

Tín hiệu IDE	Chân cắm	Ý nghĩa	Tín hiệu AT	Hướng
RESET	1	Reset các ổ đĩa	RESETDRV(1)	host→drive
GND	2	nối đất (0V)	-	-
DD7	3	Bus số liệu bit 7	SD7	Hai hướng
DD8	4	Bus số liệu bit 8	SD8	Hai hướng
DD6	5	Bus số liệu bit 6	SD6	Hai hướng
DD9	6	Bus số liệu bit 9	SD9	Hai hướng
DD5	7	Bus số liệu bit 5	SD5	Hai hướng
DD10	8	Bus số liệu bit 10	SD10	Hai hướng
DD4	9	Bus số liệu bit 4	SD4	Hai hướng
D11	10	Bus số liệu bit 11	SD11	Hai hướng
DD3	11	Bus số liệu bit 3	SD3	Hai hướng
DD12	12	Bus số liệu bit 12	SD12	Hai hướng
DD2	13	Bus số liệu bit 2	SD2	Hai hướng
DD13	14	Bus số liệu bit 13	SD13	Hai hướng
DD1	15	Bus số liệu bit 1	SD1	Hai hướng
DD14	16	Bus số liệu bit 14	SD14	Hai hướng
DD0	17	Bus số liệu bit 0	SD0	Hai hướng
DD15	18	Bus số liệu bit 15	SD15	Hai hướng
GND	19	Đất	-	-
(2)	20	Khoá chân cắm 20	-	-
DMARQ (3)	21	Yêu cầu DMA	DRQx	Drive→host
GND	22	Đất	-	-
DIOW	23	Viết số liệu qua kênh I/O	IOW	Host→drive
GND	24	Đất	-	-
IOR	25	Đọc số liệu qua kênh I/O	IOR	Host→drive
GND	26	Đất	-	-
IORDY(3)	27	Hoàn thành truy nhập I/O	IOCHRDY	Drive→host
SPSYNC	28	Đồng bộ quay trục	-	-

DMACK(3)	29	Ghi nhận DMA	DACKx	Host→drive
GND	30	Đất	-	-
INTRQ	31	Yêu cầu ngắt	IRQx	Drive→host
IOCS16	32	Truyền 16 bit qua kênh I/O	I/OCS_{16}	Drive→host
DA1	33	Bus địa chỉ 1	SA1	Host→drive
PDIAG	34	Cho qua chẩn đoán từ ổ tớ	Drive→drive	
DA0	35	Bus địa chỉ 0	SA0	Host→drive
DA2	36	Bus địa chỉ 2	SA2	Host→drive
CS1Fx	37	Chọn chip cho địa chỉ cơ sở 1F0h	-	Host→drive
CS3Fx	38	Chọn chip cho địa chỉ cơ sở 3F0h	Host→drive	
DASP	39	Kích hoạt ổ đĩa/ có ổ tớ	-	Drive→host
GND	40	Đất	-	-

(1) Tín hiệu đảo của tín hiệu bus AT.
(2) Đầu được khoá để tránh cắm nhầm các chân nối. (3) Tuỳ chọn.

Các ổ đĩa IDE ban đầu cho phép nâng tốc độ truyền số liệu giữa ổ đĩa với bộ nhớ chính lên tới 700 kbyte/s đến 1Mbyte/s.

CPU thâm nhập bộ điều khiển IDE qua một vài thanh ghi số liệu và điều khiển. Chúng được phân thành hai nhóm với địa chỉ cơ sở của cổng ở 1F0h và 3F0h. Dưới đây liệt kê các thanh ghi đó.

Thanh ghi	Địa chỉ	Bề rộng thanh ghi	Đọc (R)/ Viết (W)
Thanh ghi số liệu	1F0h	16	R/W
Thanh ghi lỗi	1F1h	8	R
Bù trừ trước	1F1h	8	W
Số các sectơ	1F2h	8	R/W
Số sectơ	1F3h	8	R/W
Cylinder LSB	1F4h	8	R/W
Cylinder MSB	1F5h	8	R/W
Ổ đĩa/đầu từ	1F6h	8	R/W
Thanh ghi trạng thái	1F7h	8	R
Thanh ghi lệnh	1F7h	8	W
Thanh ghi trạng thái biến đổi	3F6h	8	R
Thanh ghi lối ra số	3F6h	8	W
Địa chỉ ổ đĩa	3F7h	8	R

Nội dung của một số thanh ghi như sau.

- Thanh ghi báo lỗi (1F1h)

D7							D0
NDM	NTO	ABT	x	NID	x	UNC	BBK

NDM:	1= không thấy dấu địa chỉ dữ liệu	0= không có lỗi
NTO:	1= không tìm thấy rãnh 0	0= không lỗi
ABT:	1= lệnh bị bỏ	0= lệnh được thực hiện
NID:	1= không tìm thấy dấu ID	0= không lỗi
UNC:	1= lỗi dữ liệu không sửa được	0= không lỗi hoặc lỗi có sửa được
BBK:	1= sectơ được đánh dấu là hỏng	0= không lỗi

x: không sử dụng

- Thanh ghi đầu từ / ổ đĩa (1F6h)

D7							D0
1	0	1	DRV	HD3	HD2	HD1	HD0

DRV:	1= tó	0= chủ
HD3 - HD0:	số nhị phân	0000=đầu từ 0; .... 1111=đầu từ 15

- Thanh ghi trạng thái (1F7h)

D7							D0
BSY	RDY	WFT	SKC	DRQ	CORR	IDX	ERR

BSY:	1= ổ đĩa bận	0= không bận
RDY:	1= ổ đĩa sẵn sàng	0= không sẵn sàng
WFT:	1= lỗi viết	0= không lỗi
SKC:	1= định vị đầu từ xong	0= đang định vị
DRQ:	1= dữ liệu có thể được truyền	0= không
CORR:	1= lỗi dữ liệu	0= không có lỗi
IDX:	1= chỉ số đĩa vừa qua	0= không qua
ERR:	1= thanh ghi báo lỗi có chứa thông tin lỗi	0= không chứa

- Thanh ghi lối ra số (3F6h)

D7							D0
x	x	x	x	x	SRST	/IEN	x

SRST:	1= reset mềm		0 = reset xong
/IEN:	1= IRQ14 bị cấm		0 = được phép
x:	không dùng;	1 = tó;	0 = chủ;

- Thanh ghi địa chỉ ổ đĩa (3F7h)

D7							D0
x	/WTGT	/HS3	/HS2	/HS1	/HS0	/DS1	/DS0

x:	không dùng	
/WTGT:	1= không viết	0= đang viết tới đĩa được kích hoạt
/ HS3 .. /HS0:	mã bù 1 của đầu từ đang được kích hoạt	
	0000= đầu từ 15;	0001= đầu từ 14; .... 1111= đầu từ 0
/ DS1:	1= ổ đĩa 1 cấm	0= ổ đĩa 1 được kích hoạt
/DS0:	1= ổ đĩa 0 cấm	0= ổ đĩa 0 được kích hoạt

10. Chương trình ghép nối IDE và các pha lệnh

Chương trình ghép nối IDE tương tự như với ổ đĩa mềm gồm 3 pha:

- Pha lệnh: CPU chọn các thanh ghi thông số và truyền mã lệnh để khởi phát chạy chương trình.
- Pha dữ liệu: thực hiện các lệnh liên quan tới thâm nhập đĩa, ổ đĩa định vị đầu từ và truyền dữ liệu giữa ổ đĩa và bộ nhớ chính.
- Pha kết quả: bộ điều khiển cung cấp những thông tin trạng thái về các lệnh đã chạy trong các thanh ghi tương ứng và phát ra ngắt cứng IRQ14 (tương ứng ngắt 76h).

Khác với máy XT, từ máy AT trở đi việc truyền dữ liệu được thực hiện bằng chương trình vào/ra qua CPU và thanh ghi số liệu. CPU sẽ viết các sectơ và định dạng dữ liệu vào hoặc đọc chúng từ các thanh ghi 16 bit. Chỉ có các byte ECC được đọc và viết từng phần 8 bit qua byte thấp của thanh ghi dữ liệu. Để đồng bộ CPU và bộ điều khiển trong quá trình trao đổi dữ liệu, bộ điều khiển phát ra một ngắt cứng ở những khoảng thời gian khác nhau qua IRQ14:

- Đọc sectơ: bộ điều khiển luôn cho phép IRQ14 khi CPU có thể đọc một sectơ thậm chí cùng với các byte ECC từ bộ đệm sectơ. Khác với các lệnh khác, lệnh này không phát một ngắt ở bắt đầu pha kết quả, như vậy số của ngắt cứng giống như số của sectơ được đọc.
- Viết sectơ: bộ điều khiển luôn kích hoạt IRQ14 khi nó đợi dữ liệu từ CPU.

Lưu ý rằng sectơ đầu tiên được truyền ngay lập tức sau khi phát lệnh và bộ điều khiển không phát một ngắt cho mục đích này. Thêm nữa, qua IRQ14 bộ điều khiển kích hoạt một ngắt cứng ở thời điểm bắt đầu của pha kết quả. Như vậy, số của ngắt cứng trùng với số của sectơ được viết.

- Tất cả các lệnh khác: bộ điều khiển sẽ phát một ngắt cứng qua IRQ14 ở thời điểm bắt đầu của pha kết quả.

Chương trình con xử lí ngắt INT76h tương ứng với IRQ14 trong PC phải có thể xác định liệu bộ điều khiển muốn ra số liệu hoặc liệu một ngắt đã xảy ra để chỉ thị bắt đầu pha kết quả. Nếu ta muốn vậy phải sử dụng các thanh ghi trạng thái và báo lỗi để xác định nguồn ngắt. Bộ điều khiển IRQ13 bị cấm ngay khi CPU đọc thanh ghi trạng thái (1F7h). Nếu IRQ14 vẫn còn tích cực ta phải đọc thông tin trạng thái qua thanh ghi trạng thái khác ở địa chỉ 36Fh.

Lưu ý rằng bộ điều khiển của ổ đĩa được định địa chỉ khởi phát chạy lệnh ngay sau khi CPU đã viết lệnh vào thanh ghi lệnh. Như vậy ta phải nạp tất cả các thông số cần thiết với các giá trị được đòi hỏi trước khi khởi phát chạy lệnh. Định dạng cho một lệnh được chỉ ra như dưới đây.

Thanh ghi	7							1
Lệnh (1F7h)	0	0	1	1	0	0	L	R
Đếm sectơ (1F2h)	Số sectơ được viết lên đĩa							
Số của sectơ (1F3h)	S7	S6	S5	S4	S3	S2	S1	S0
Cylinder LSB (1F4h)	Z7	Z6	Z5	Z4	Z3	Z2	Z1	Z0
Cylinder MSB (1F50	0	0	0	0	0	0	Z9	Z8
Ổ đĩa / đầu từ (1F6)	1	0	1	DRV	HD3	HD2	HD1	HD0

L=1 thì CPU sẽ cung cấp 4 byte ECC; L= 0 lệnh viết bình thường.

R= 1: bộ xử lí có gắng lặp lại lệnh để sửa lỗi bên trong;

R= 0 : bộ xử lí bỏ qua lệnh ngay lập tức nếu phát hiện có lỗi

DRV cho phép dùng 1 trong 2 ổ đĩa

HD3 - HD0 cho phép chọn 1 trong số 16 đầu từ của ổ đĩa được kích hoạt.

Sau đây là thí dụ đoạn chương trình về một lệnh: viết 4 sectơ bắt đầu với cylinder 167, đầu từ 3, sectơ 7 với các byte ECC vào ổ đĩa chủ.

```
unsigned int_buffer[1024];
unsigned char byte_buffer[16];
unsigned int *word_pointer;
unsigned char *byte_pointer;
int int_count;
main ( )
{ int word_count, byte_count;
  void far *old_irq14;
```

```
    word_pointer = &word_buffer;                /* khởi động */
    byte_pointer = &byte_bufer;                 /*con trỏ */
    init_buffers ( );                           /* khởi động vùng đệm */
    old_irq14 = dosgetvect(0x76);               /* đặt ngắt mới */
    dossetvect(0x76, new_irq14 ());             /* cho IRQ14 */
    while ((inport(0x1f7) &0x80) ==0x80;        /* đợi đến khi bt BSY trong thanh ghi trạng thái
                                                   được xoá */
    outport(0x1f2, 0x04);                       /* thanh ghi đếm sectơ: 4 sectơ */
    outport(0x1f3, 0x07);                       /*  thanh ghi số sectơ: 7 */
    outport(0x1f4, 0xa7);                       /* LSB cylinder: 167 */
    outport(0x1f5, 0x00);                       /* MSB cylinder: 0 */
    outport(0x1f6, 0xa3);                       /* ổ đĩa 0, đầu từ 3 */
    outport(0x1f7, 0x33);                       /* thanh ghi lệnh mã 001100, L= 1, R= 1 */
                                                /* viết sectơ thứ nhất gồm 512 byte số liệu + 4 byte ECC */
while ((inport(0x1f7) & 0x80) == 0x80 || (inport(0x1f7) &0x80) != 0x80);
                                                /* đợi bit BSY bị xoá và DRQ được đặt */
word_pointer = word_buffer;                     /* khởi động con trỏ */
for (word_count = 0; word_count < 256; word_count ++, word_pointer ++)
        { outportw(0x1f0, *word_pointer); }     /* truyền 256 từ = 512 byte dữ liệu */
byte_pointer = byte_buffer;                     /* khởi động con trỏ */
for (byte_count = 0, byte_count <4, byte_count++, byte_pointer++)
        { outport(0x1f0, byte_pointer);     }   / *truyền 4 byte ECC */
int_count = 0;                                  /* khởi động đếm ngắt */
while (int_cont < 4);                   /* đợi đến khi 4 sectơ được truyền xong */
_dos_setvect (0x76, old_irq14 ());              /* đặt ngắt cũ IRQ14 */
status_check ();                        /* kiểm tra trạng thái và xác định mã lỗi */
exit (0);
}
        void interrupt  far new_irq14 ()
        {  int word_count, byte_count;
                int_count++;
        if (int_count < 4)  {           /* bỏ qua ngắt ở bắt đầu pha kết quả */
         for (word_count = 0, word_count < 256, word_count++, word_pointer++)
        {  outportw(0x1f0, word_pointer);    }      / truyền 256 từ = 512 byte dữ liệu*/
         for (byte_count = 0, byte_count < 4, byte_count++, byte_pointer++)
        {  outport(0x1f0, byte_pointer);                        / truyền 4 byte ECC */
        }
}
```

return;

11. Ghép nối với chuẩn SCSI

SCSI (Small Computer System Interface) là một chuẩn ghép nối giữa ổ đĩa và PC khác IDE, nó mềm dẻo và mạnh. Chuẩn này xây dựng một bus ghép nối gồm 8 đơn vị SCSI trong đó một *host-adapter* nối 7 đơn vị còn lại với bus hệ thống. Khác với host-adapter của IDE, host-adapter này phức tạp hơn vì nó phải thực hiện các chức năng của bus SCSI, nhưng ngược lại nó lại không bị ràng buộc bởi những hạn chế của bus AT. Bus SCSI chỉ phục vụ cho việc trao đổi dữ liệu giữa những đơn vị được nối với nó. Trong một thời điểm chỉ có hai đơn vị có thể được kích hoạt. Việc trao đổi dữ liệu có thể được thực hiện giữa host-adapter và ổ đĩa hoặc giữa hai đơn vị SCSI như ổ băng cassette và ổ đĩa mà không có sự tham gia của CPU vì các ổ đĩa SCSI đủ "thông minh" để tự làm điều đó. Giới hạn dung lượng ổ đĩa của SCSI là 8,4GB và tốc độ truyền dữ liệu đạt tới 5 Mbyte/s.

Việc nối giữa các đơn vị SCSI được thực hiên bởi cáp dẫn bẹt 50 dây có các chân tín hiệu được chỉ ra như sau:

Tín hiệu	Chân cắm	Ý nghĩa	Tín hiệu	Chân cắm	Ý nghĩa
GND	1	Đất	TERMPWR	26	Đầu nối
$\overline{DB(0)}$	2	Bit số liệu 0	GND	27	Đất
GND	3	Đất	GND	28	Đất
$\overline{DB(1)}$	4	Bit số liệu 1	GND	29	Đất
GND	5	Đất	GND	30	Đất
$\overline{DB(2)}$	6	Bit số liệu 2	GND	31	Đất
GND	7	Đất	$\overline{ATN}$	32	Đất
$\overline{DB(3)}$	8	Bit số liệu 3	GND	33	Đất
GND	9	Đất	GND	34	Đất
$\overline{DB(4)}$	10	Bit số liệu 4	GND	35	Đất
GND	11	Đất	$\overline{BSY}$	36	Bận
$\overline{DB(5)}$	12	Bit số liệu 5	GND	37	Đất
GND	13	Đất	$\overline{ACK}$	38	Ghi nhận
$\overline{DB(6)}$	14	Bit số liệu 6	GND	39	Đất
GND	15	Đất	$\overline{RST}$	40	Reset
$\overline{DB(7)}$	16	Bit số liệu 7	GND	41	Đất
GND	17	Đất	$\overline{MSG}$	42	Thông báo
$\overline{DB(P)}$	18	Bit chẵn lẻ	GND	43	Đất
GND	19	Đất	$\overline{SEL}$	44	Chọn
GND	20	Đất	GND	45	Đất
GND	21	Đất	$\overline{C/D}$	46	Lệnh/ số liệu
GND	22	Đất	GND	47	Đất
GND	23	Đất	$\overline{REQ}$	48	Yêu cầu
GND	24	Đất	GND	49	Đất
(không nối)	25	-	I/O	50	I/O

Bất kì một đơn vị SCSI nào cũng có thể thực hiện chức năng như một *đơn vị khởi phát* (initiator) nắm quyền điều khiển bus bởi các tín hiệu điều khiển. Nó sẽ kích hoạt một đơn vị nào đó gọi là *đơn vị đích* (target) bắt đơn vị này thực hiện một nhiệm vụ nào đó.

Hoạt động trên bus SCSI gồm 8 pha như sau:

1. *Pha bus rỗi*: Không có đơn vị SCSI nào đang sử dụng và điều khiển bus. Pha này được kích hoạt khi cả hai dây $\overline{SEL}$ và $\overline{BSY}$ bị cấm (ở mức cao).

2. *Pha trọng tài*: Một đơn vị SCSI có thể điều khiển bus nhằm cho nó có thể hoạt động như là một đơn vị khởi phát hoặc đơn vị đích. Lúc này các quá trình như sau được thực hiện:
   - Bus phải đang ở trạng thái rỗi,
   - Kích hoạt dây $\overline{BSY}$ và đặt các trường nhận dạng SCSI-ID của đơn vị lên bus dữ liệu,
   - Sau một thời gian trễ ngắn để điều tra bus dữ liệu, nếu có một SCSI-ID khác có mức ưu tiên cao hơn đang được kích hoạt thì đơn vị này rút lui khỏi bus. Ngược lại, nó sẽ chiếm dụng bus và kích hoạt dây $\overline{SEL}$.
   - Sau khoảng trễ ngắn để xoá bus, đơn vị SCSI giờ đây đã có thể điều khiển bus SCSI và trao đổi dữ liệu trên bus.

3. *Pha chọn lựa*: Đơn vị khởi phát chọn đơn vị đích và yêu cầu đích thực hiện một nhiệm vụ nào đó (thí dụ, đọc hoặc viết một khối dữ liệu). Trong pha này, tín hiệu $\overline{I/O}$ được kích hoạt để phân biệt pha này với pha chọn lại của đơn vị đích. Nó sẽ xuất giá trị trường nhận dạng ID của nó OR với giá trị ID của đích lên bus. Do vậy sẽ có 2 bit dữ liệu đặc trưng cho đơn vị khởi phát và đơn vị đích được kích hoạt. Đơn vị đích được định địa chỉ phải phát hiện ra rằng nó đã được chọn bởi đơn vị khởi phát đã kích hoạt $\overline{BSY}$, nếu không pha chọn lựa này sẽ bị lỗi và đơn vị khởi phát sẽ cấm dây $\overline{SEL}$.

4. *Pha chọn lựa lại*: Đơn vị đích sẽ có thể thiết lập lại thông tin với đơn vị khởi phát để tiếp tục hoạt động đã bị ngắt trước đó. Thí dụ, hot adapter phát ra một lệnh đọc tới ổ đĩa đích. Việc định vị đầu từ và đọc sectơ mất chừng 40 ms là quá dài đối với PC. Do đó đơn vị đích tách rời ra khỏi bus SCSI và tự thực hiện việc này nhưng vẫn nhớ dữ liệu nhận dạng ID của đơn vị khởi phát. Sau khi thực hiện xong nhiệm vụ trên, ổ đĩa đích sẽ thiết lập lại tiếp xúc với đơn vị khởi phát này bởi pha chọn lựa lại và truyền các dữ liệu vừa được đọc. Sự xử lí tại pha này cũng giống 2 pha trên. Trong suốt 4 quá trình truyền thông tin về lệnh, dữ liệu, thông báo và trạng thái thì thông tin về số liệu và điều khiển được truyền qua bus dữ liệu. Các tín hiệu $\overline{C/D}$, $\overline{I/O}$ và $\overline{MSG}$ được dùng để phân biệt các giai đoạn truyền này. Nếu dây $\overline{I/O}$ được kích hoạt tức là thông tin được truyền từ đơn vị đích tới khởi phát, nếu không quá trình truyền sẽ theo hướng ngược lại. Các quá trình truyền này được thực hiện có móc nối. Bên phát đặt số liệu lên bus $\overline{DB0}$ đến $\overline{DB7}$ và thông tin chẵn lẻ lên $\overline{DB(P)}$ và kích hoạt dây $\overline{REQ}$ để báo cho bên thu biết số liệu đã ổn định. Bên thu nhận số liệu và kích hoạt dây $\overline{ACK}$ để ghi nhận.

5. *Pha lệnh*: Trong pha này đơn vị đích có thể yêu cầu dữ liệu lệnh từ đơn vị khởi phát bằng việc kích hoạt dây $\overline{C/D}$ và cấm các dây $\overline{MSG}$ và $\overline{I/O}$.

6. *Pha dữ liệu*: Đơn vị đích lệnh cho đơn vị khởi phát truyền dữ liệu đến đích (pha data-out) hoặc nó có thể cung cấp dữ liệu cho đơn vị khởi phát (pha data-in).

7. *Pha thông báo*: Đơn vị đích yêu cầu đơn vị khởi phát truyền thông báo tới đích (pha message-out) hoặc cung cấp thông báo tới đơn vị khởi phát (pha message-in).

8. *Pha trạng thái*: Đơn vị đích cấp thông tin trạng thái tới đơn vị khởi phát.

Bên cạnh các tín hiệu điều khiển còn có 2 tín hiệu nữa là:

$\overline{ATN}$ (chú ý): Đơn vị khởi phát kích hoạt dây này để thông tin cho đơn vị đích biết nó có ý định truyền thông báo.

$\overline{RST}$ (reset): Khi dây này được kích hoạt, tất cả các đơn vị được tách rời khỏi bus SCSI, tất cả các hoạt động bị bỏ qua và các đơn vị được đặt tới trạng thái được xác định.

10.1.5. Tổ chức lôgic của đĩa từ

Về mặt lôgic, đĩa từ bao gồm các đơn vị như sau:

- *Phân khu* (partition): Một đĩa cứng có thể được chia thành các phần khác nhau gọi là phân khu và DOS coi các phần đó như các ổ đĩa độc lập gọi là ổ đĩa lôgic được gán cho các tên từ C đến Z. Các phân khu trên đĩa cứng được tạo lập và kiểm tra bởi chương trình FDISK.EXE. Nội dung của chúng có thể được khảo sát bằng các chương trình như DISKEDIT.EXE trong gói chương trình Norton Utilities NU.

- *Sector khởi động* (boot sector): Đó là sectơ thứ nhất của đĩa mềm (đầu từ 0, rãnh 0, sectơ 1) hoặc sectơ khởi phát của một phân khu trong ổ đĩa cứng. Nó có một bảng mô tả môi trường gồm các điểm vào lưu trữ các thông tin như: tên và nhận dạng của nhà sản xuất và phiên bản của hệ điều hành, số byte trong một sectơ v.v... Tiếp đó là chương trình khởi động hệ thống. Chương trình này trước hết kiểm tra, nạp 2 file hệ thống có thuộc tính ẩn là IO.SYS và MSDOS.SYS cùng hệ điều hành DOS tại đĩa mềm hoặc phân khu đĩa cứng vào bộ nhớ chính và cho chạy 2 file đó. Nếu các file này không có, chương trình sẽ phát thông báo lỗi "không có đĩa hệ thống".

- *Thư mục gốc* (Root directory): Bằng các điểm vào trong sectơ khởi động DOS biết rằng dung lượng của đĩa mềm hoặc phân khu là bao nhiêu, hiện có bao nhiêu rãnh, số sectơ trên một rãnh và số byte trên một sectơ là bao nhiêu. Tuy nhiên để quản lí các file, hệ điều hành cần biết điểm bắt đầu của các vùng lưu trữ file. Chính thư mục gốc được đặt kề ngay sau sectơ khởi động chứa các thông tin đó. Thư mục này giữ các thông tin về vùng nhớ, kích thước, ngày và thời gian của lần thay đổi cuối cùng của file và thư mục con cũng như thuộc tính của các điểm vào thư mục. Thư mục gốc và các thư mục con gồm điểm vào 32 byte được lưu trữ theo định dạng Intel (byte thấp trước, byte cao sau). Với lệnh MKDIR có thể tạo ra các thư mục con. Giống như thư mục gốc, thư mục con cũng gồm các file và các thư mục con nữa (sub-subdirectories). Do đó, một cluster được dùng để giữ các điểm vào thư mục cho các file và thư mục con nữa. Cluster khởi phát trong thư mục gốc chỉ tới cluster này. Lệnh MKDIR tạo nên 2 điểm vào thư mục đầu tiên trong một thư mục con. Một điểm vào chỉ chính thư mục con (được đặt tên là ".") và điểm vào thứ hai chỉ tới thư mục ".." một mức cao hơn về thứ bậc (vì vậy lệnh CHDIR .. chuyển tới một mức cao hơn trong cây thư mục). Với mục đích như vậy, cluster khởi phát trong các điểm vào này chỉ tới điểm bắt đầu của thư mục mẹ phía trên. Một điểm vào 0 có nghĩa là thư mục gốc là thư mục mẹ vì rằng thư mục con hiện tại là thư mục con ở bậc 1. Lệnh FORMAT phát ra sectơ khởi động, bảng định vị file FAT và thư mục gốc. Nhưng các thư mục con có cấu trúc định vị file động nên phải được tạo bởi người dùng bằng lệnh MKDIR. Cũng vậy, các chương trình ứng dụng dĩ nhiên có thể tạo hoặc xoá các thư mục con bằng các hàm của DOS.

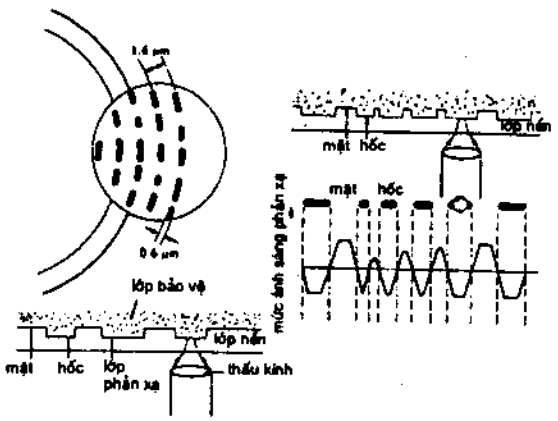
- *Bảng định vị file FAT* (file allocation table): Các file hoặc các thư mục con được định vị trong một cluster mà số của nó được lưu trữ trong điểm vào thư mục như cluster khởi phát. Một cluster thường có từ 1 đến 8 sectơ. Bảng FAT được xắp xếp giữa sectơ khởi động và thư mục gốc và được tự động tạo ra khi định dạng đĩa bằng lệnh FORMAT. Mỗi đĩa mềm hoặc đĩa cứng thường có 2 bảng copy của FAT. Số bảng này được lưu trữ ở địa chỉ offset 10h trong sectơ khởi động. Từ offset 16h ta có thể tìm thấy kích thước của FAT theo đơn vị sectơ.

FAT lưu trữ các điểm vào ứng với số cluster có trên đĩa. Bề dài của mỗi điểm vào tuỳ thuộc phiên bản DOS hoặc WINDOWS, có thể là 12, 16 hoặc 32 bit. Do đó có 3 loại bảng FAT-12, FAT-16 và FAT-32.

Nội dung của các điểm vào như sau: hai điểm vào đầu tiên để nhận dạng loại đĩa, các điểm vào sau cho biết: tình trạng của cluster (tự do, bị chiếm,...), dữ kiện được ghi ở cluster nào (phần này bao giờ cũng là có cluster đang trữ phần dữ liệu kế tiếp của file nếu file nằm trải dài trên nhiều cluster). Như vậy, khi muốn truy nhập một file, hệ điều hành sẽ căn cứ vào các điểm vào trên bảng FAT để tìm ra các cluster của file đó.

10.2. GHÉP NỐI CÁC Ổ ĐĨA QUANG

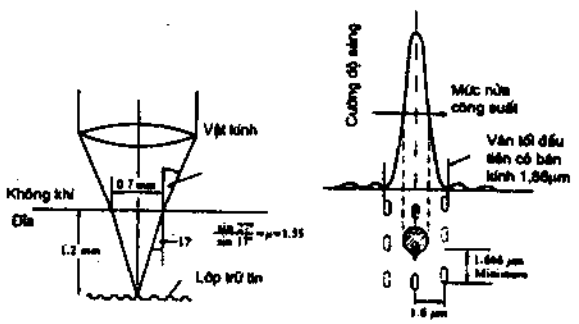
Đĩa quang như CD-ROM lưu trữ thông tin bằng việc phản xạ hoặc không phản xạ các điểm sáng dọc theo rãnh đĩa. Một tia sáng phát ra từ một Laser bán dẫn được hội tụ trên mặt rãnh và ánh sáng phản xạ được phát hiện bởi một transistor hoặc diode nhạy quang. Hình 10.13 minh hoạ cấu trúc của mặt đĩa quang.



Hình 10.13 Rãnh quang và ánh sáng phản xạ trên mặt và hốc.

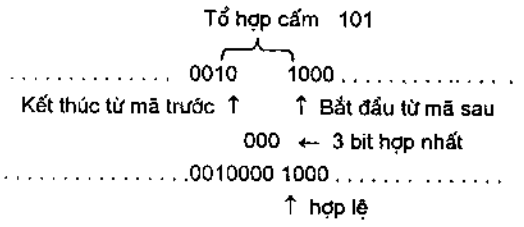
Dữ liệu được ghi lên đĩa thành một chuỗi các hốc có độ dài biến đổi dọc theo rãnh xoáy trôn ốc với bước rãnh không đổi (1,6µm). Các hốc và mặt đĩa được phủ lên lớp nền và được phủ tiếp bằng một lớp bảo vệ trong suốt. Ánh sáng từ nguồn laser được hội tụ trên các hốc sau khi qua biên giới phân cách giữa không khí và đĩa như hình 10.14.

Điểm sáng được hội tụ tuy vậy vẫn lớn hơn nhiều các hốc và do đó các vết bẩn cũng như bụi trên mặt đĩa ít ảnh hưởng tới kết quả xử lí đọc dữ liệu. Ánh sáng từ vùng giữa các hốc (tức là vùng mặt) được phản xạ trở lại trong khi ánh sáng từ các hốc bị tán xạ mất. Sự tán xạ là do chiều cao của hốc xấp xỉ 0,13µm trên bề mặt và ánh sáng được phản xạ từ hốc và mặt xung quanh giao thoa triệt tiêu lẫn nhau (do 0,13µm bằng một phần tư bước sóng ánh sáng laser trong môi trường đĩa và do đó ánh sáng từ hốc và mặt khác nhau một nửa bước sóng). Như vậy, sự thay đổi cường độ ánh sáng phản xạ từ đĩa đại diện cho sự



Hình 10.14 Sự phân bố cường độ sáng giữa hốc và mặt.

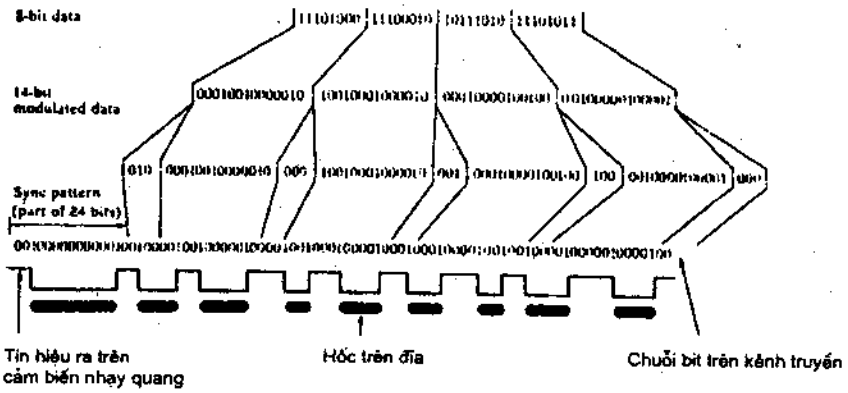
thay đổi vị trí từ hốc đến mặt và từ mặt đến hốc trên đĩa. Độ phân giải của hệ thống quang (tức là kích thước của phần tử nhỏ nhất còn có thể phân biệt được) quyết định mật độ ghi tin được xác định bởi bước sóng ánh sáng laser (700nm trong không khí và là 500nm trong môi trường đĩa) và tiêu cự của kính vật (0,45). Hình 10.15 cũng chỉ ra sự phân bố cường độ sáng từ một tia sáng chiếu lên một hốc trong một dãy các hốc (được tạo bởi các đường rãnh liền kề). Kích thước của các hốc được chế tạo sao cho nửa năng lượng của tia sáng rơi trên hốc, nửa rơi trên mặt. Năng lượng phản xạ từ hốc lí tưởng phải bằng 0 nếu ánh sáng từ hốc và mặt giao thao với nhau. Dữ liệu được lưu trữ trong CD-ROM được mã hoá nhằm tăng được mật độ ghi tin và chống lỗi. Chiều dài của các hốc được điều chế và sự chuyển tiếp từ hốc đến mặt (hoặc từ mặt đến hốc) được gán cho giá trị 1. Dữ liệu nguồn trong dạng byte và mỗi byte được mã hoá thành từ mã 14 bit. Từ mã được chọn sao cho các số 1 liên tiếp nhau được phân cách bởi ít nhất là 2 số 0. Thêm nữa số số 0 nhiều nhất là 10. Theo vậy, mã 14 bit này có 267 giá trị thoả mãn 256 số của một dữ liệu 1 byte. Bảng sau cho 10 giá trị đầu tiên từ dữ liệu 8 bit chuyển sang từ mã 14 bit như kể trên. Các từ mã này còn được phân cách nhau bởi 3 bit hợp nhất để đảm bảo quy tắc lập mã không bị vi phạm khi các bit cuối của từ trước được ghép với các bit đầu của từ sau. Các bit này chỉ có ý nghĩa như dấu phảy phân cách. Thí dụ dưới đây biểu thị sự cần thiết của chúng khi được dùng để phân cách 2 từ mã ...0010 và 1000...



	Dữ liệu 8 bit	Từ mã 14 bit
0	0000 0000	01 0010 0010 0000
1	0000 0001	10 0001 0000 0000
2	0000 0010	10 0100 0100 000
3	0000 0011	10 0010 0010 0000
4	0000 0100	01 0001 0001 0000
5	0000 0101	00 0001 0001 0000
6	0000 0110	00 0100 0010 0000
7	0000 0111	00 1001 0000 0000
8	0000 1000	01 0010 0100 0000
9	0000 1001	10 0000 0100 0000
10	0000 1010	10 0100 0100 0000

Nhịp đồng hồ được khôi phục từ các hốc và mặt là 4,3128 MHz vì đây là tốc độ thay đổi cực đại của tín hiệu từ hốc và mặt ở các đĩa CD chuẩn có tốc độ quét 1,3 m/s. Mật độ bit là 1,66 bit/μm hoặc 42 kbit/inch. Tại một hốc có bề rộng 1,6μm con số này tương ứng với 6 x 10^8 bit / $inch^2$ hay 10^6 bit / mm^2. Hình 10.15 là một thí dụ về lập mã dữ liệu trên đĩa.

Do mật độ ghi tin rất cao như vậy nên khó tránh khỏi lỗi, vì vậy ở đây phải dùng các mã sửa lỗi rất mạnh. Mã đan xen chéo CIRC (cross interleaved Reed-Solomon code) nhóm 24 byte dữ liệu và lập mã chúng thành các nhóm 32 byte. Thông tin được đan xen nhằm cho một nhóm các lỗi ở một vùng sẽ tác động đến một vài nhóm mã. Các nhóm này có thể được sửa lỗi riêng biệt. Nếu tất cả các lỗi rơi vào một nhóm đơn thì không thể sửa được. Một trong những sự khác nhau giữa CD sử dụng trong kĩ thuật Audio và CD-ROM là CD-ROM dùng một lớp mã hoá thêm để giảm hơn nữa tỉ lệ không phát hiện lỗi tới một phần 10^{11}bit. Hơn thế nữa kĩ thuật lập mã CIRC cho phép có khả năng sửa một nhóm lỗi lên tới 450 byte (tức là chiếm tới 2mm chiều dài rãnh ghi).

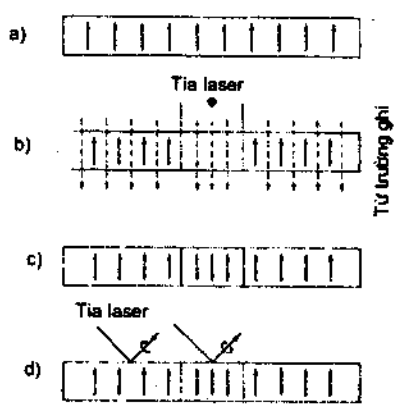


Hình 10.15 Lập mã dữ liệu trên đĩa quang

Dung lượng điển hình của đĩa CD-ROM là 553 Mbyte. Rãnh hình xoáy trôn ốc của CD-ROM được chia thành các sectơ, mỗi sectơ được địa chỉ hoá tách biệt. Địa chỉ của một sectơ là địa chỉ tuyệt đối so với điểm bắt đầu của rãnh và trong dạng phút, giây và block. Một sectơ hoặc một block gồm 12 byte đồng bộ (dùng để hồi phục nhịp đồng hồ), 4 byte tiêu đề để nhận dạng sectơ, một khối 2048 byte lưu trữ dữ liệu và 288 byte phụ thêm dùng cho mã sửa lỗi. Vì rằng kích thước của một hốc là không đổi và chúng được ghi dọc theo rãnh xoáy trôn ốc nên số hốc trên một vòng xoay biến đổi từ phần rãnh bên ngoài tới bên trong. Kết quả là tốc độ của đĩa phải phụ thuộc vào vùng chứa sectơ đang đọc (tức là đĩa cần phải chuyển động với vận tốc dài không đổi chứ không phải là vận tốc góc không đổi). Đĩa quay với tốc độ từ 200 đến 500 vòng /phút, hơn nữa đầu đọc tương đối nặng nên cũng làm giảm tốc độ thâm nhập từ 2 phần rãnh liền kề nhau. Những nhân tố này làm hạn chế thời gian thâm nhập của CD-ROM trong vùng từ 100 đến 200 ms (tức là một bậc chậm hơn so với một đĩa cứng thông thường).

Những đĩa quang WORM (write-once, read-many) hiện đang phổ biến trên thị trường cho phép ghi một lần và đọc nhiều lần. Nguyên tắc của việc đọc và tìm rãnh như ở CD-ROM. Một số ổ đĩa WORM dùng chất nhuộm hữu cơ bị bay hơi khi được chiếu các xung ánh sáng viết dữ liệu từ nguồn phát tia laser công suất lớn lên đó. Một số khác đơn giản chỉ cắt bỏ bề mặt một lớp không phản xạ của vật liệu trên nền phản xạ để tạo ra các hốc. Một số dùng tia laser mạnh đốt chảy các vùng có phủ chất telua (tellurium) để tạo ra các hốc. Các ổ đĩa quang ghi /đọc được không thể dùng các công nghệ kể trên được. Kĩ thuật phân cực từ đã được áp dụng cho mục đích này. Hình 10.16 minh hoạ nguyên tắc của một đĩa CD có thể xoá được.

Bề mặt đĩa được tạo sẵn cấu trúc rãnh và rãnh được phủ một số lớp. Lớp mang tin dùng loại vật liệu như TeFeCo. Chất này có khả năng thay đổi sự phân cực của tia sáng laser được phản xạ trên mặt đĩa. Từ độ của màng mỏng chất này sẽ quyết định chiều phân cực của tia sáng phản xạ. Ban đầu màng mỏng được đặt vào một từ trường đồng nhất nhằm xắp xếp các phân tử TeFeCo để có được các vùng có từ độ theo một hướng ban đầu nào đó quyết định đến chiều quay của mặt phẳng phân cực của tia phản xạ. Có thể coi các vùng có từ độ theo hướng



Hình 10.16 Các vectơ từ độ trên lớp ghi tin của đĩa quang ghi/đọc được.

ban đầu đó lưu trữ các bit 0 trên rãnh (hình 10.16a). Trong pha viết, một xung laser ngắn sẽ đốt nóng bề mặt ở vùng cần xoá tin lên quá nhiệt độ Curie làm chất mất từ tính, từ độ bằng 0 (hình 10.16b). Đồng thời ngay sau đó áp đặt một từ trường ngoài theo hướng nào đó nhằm để sau khi hết xung, nhiệt độ của chất hạ xuống dưới nhiệt độ Curie sẽ cho phép vùng đó có một từ độ theo hướng mong muốn, hoặc theo hướng cũ, hoặc theo hướng ngược lại (hình 10.16c). Khi đọc, một tia laser có cường độ yếu hơn nhiều được hội tụ trên đĩa và bộ phận nhạy quang sẽ phát hiện liệu tia sáng phân cực được phản xạ có vectơ phân cực được quay theo chiều kim đồng hồ hay ngược chiều kim đồng hồ (hình 10.16d).

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

10. 1 Một đĩa cứng gồm một chồng 10 đĩa và 18 mặt để ghi dữ liệu. Mỗi mặt gồm 200 track và được quay với tốc độ 200 vòng/phút. Mỗi track được chia thành 8 khối mỗi khối gồm 256 từ-32 bit. Có một đầu từ cho một mặt đĩa và như vậy có khả năng đọc 18 track một lần trong một cylinder. Thời gian tìm kiếm là 1ms. Khi không truy nhập dữ liệu, đầu từ được đỗ ở ngoài các track. Hãy tính:

Tổng dung lượng của ổ đĩa ?

Tốc độ truyền dữ liệu cực đại theo bit/giây ?

Thời gian truy cập của ổ đĩa ?

Tốc độ trung bình truyền dữ liệu khi đọc 256 khối từ đã được phân bố ngẫu nhiên trên đĩa ?

10. 2 Cho đường kính đĩa 12 inch và track ngoài cùng cách mép đĩa 1 inch; hãy tính mật độ ghi tin (bit / inch) của track trong cùng và ngoài cùng. Cho mật độ track trên đĩa là 200 track /in.

10. 3 Giả thiết rằng điều kiện chuyển động của đầu từ như bài tập trên không chính xác. Nếu ổ đĩa có N cylinder được đánh số từ 0 tới N-1 (cylinder trong cùng được đánh số 0). Hãy tìm biểu thức biểu diễn chuyển động ngẫu nhiên từ một cylinder này tới cylinder khác (theo số chuyển dịch đầu từ). Giả thiết rằng việc tìm kiếm cylinder bên cạnh có xác suất như nhau đối với các vị trí của đầu từ trên mặt đĩa. Chuyển động trung bình tiến tới N/3 đối với các giá trị N đủ lớn. Gợi ý: Xem xét cylinder thứ K và tính số các bước cần thiết để chuyển tới cylinder thứ J; trong đó J biến đổi từ 0 đến N-1.

10. 4 Dùng chương trình DISKEDIT.EXE trong gói phần mềm Norton Utility để khảo sát tổ chức vật lý và tổ chức lôgic của một đĩa qua các thực đơn xem các đơn vị như phân khu, sectơ khởi động, bảng FAT, thư mục gốc v.v... bằng định dạng text và số hệ mười sáu. Chú ý: dùng tuỳ chọn chỉ đọc READ-ONLY để tránh ghi đè lên nội dung đĩa.

10. 5 Tạo một đĩa mềm hệ thống bằng các phương pháp nào đó (thí dụ, format đĩa với tuỳ chọn /S hoặc chạy file SYS a:\ từ thư mục DOS của ổ đĩa cứng có thể khởi động). Khảo sát đĩa theo các yêu cầu như bài tập trên. Xoá tuỳ chon READ-ONLY trong chương trình DISKEDIT. Thử thay đổi nội dung các điểm vào rồi quan sát và nhận xét. Thí dụ, thay đổi điểm vào có offset 00h cho phép đĩa có thể khởi động; thử khởi động máy bằng ổ A; nhận xét.

10. 6 Khảo sát đĩa bằng các hàm trong ngắt 21h của DOS.

CHƯƠNG 11. GHÉP NỐI MÀN HÌNH

Có nhiều kĩ thuật khác nhau được sử dụng để hiển thị các kí tự và hình ảnh trong máy vi tính. Loại màn hình phổ biến nhất cho đến nay vẫn là màn hình sử dụng ống tia điện tử CRT (cathode-ray tube). Sau đó là các loại màn hình thường được sử dụng trong máy tính xách tay như màn hình tinh thể lỏng LCD (liquid crystal display) và màn hình plasma.

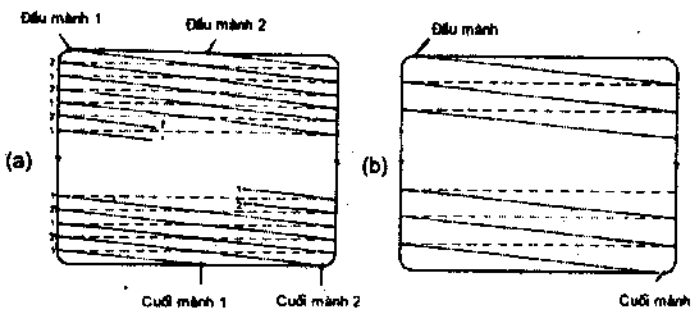
11.1. HIỂN THỊ KÍ TỰ BẰNG PHƯƠNG PHÁP QUÉT

11.1.1. Các khái niệm cơ bản về quét mành

CRT là một ống chân không hình chai tương đối lớn trong đó có một súng phóng điện tử gắn vào phía đuôi ống phát ra một chùm tia điện tử được định hướng tới mặt trước của ống dưới tác dụng của điện trường cao áp. Mặt trong thành ống phía trước được phủ một lớp huỳnh quang (hợp chất phosphor) có tác dụng phát sáng mỗi khi bị các điện tử đập vào. Màu sắc của ánh sáng phát ra tuỳ thuộc vào loại phosphor được sử dụng.

Phương pháp phổ biến nhất để tạo ra hình ảnh trên màn hình CRT là quét đi quét lại một chùm tia điện tử ngang qua màn hình từ trái sang phải bằng từ trường biến thiên được sinh ra do các cuộn lái tia. Khi chùm tia quét đến cạnh phải của màn hình, nó lập tức bị xoá và nhanh chóng trở về cạnh bên trái để bắt đầu lượt quét mới. Nếu như chùm tia chuyển dần từ trên xuống dưới đồng thời vẫn quét qua quét lại theo phương ngang thì do cảm giác lưu ảnh trong mắt người ta sẽ thấy toàn bộ màn hình bật sáng. Khi chùm tia quét tới góc dưới màn hình, nó lại bị xoá và được đưa về góc trên để lại bắt đầu quá trình quét tiếp theo. Hình ảnh được tạo ra theo cách này được gọi là ảnh *quét mành* (raster scan). Để có thể tạo ra hình ảnh trên màn hình trong suốt quá trình quét, cường độ của chùm tia điện tử tại các thời điểm đập vào các điểm trên màn hình được điều khiển liên tục để nhận được độ sáng tối mong muốn. Vấn đề quan trọng ở đây là cường độ chùm tia, hay tín hiệu video, phải được đồng bộ hoá với quá trình quét ngang cũng như quét dọc của chùm tia, nhờ vậy mà hình ảnh được hiển thị ổn định trên màn hình.

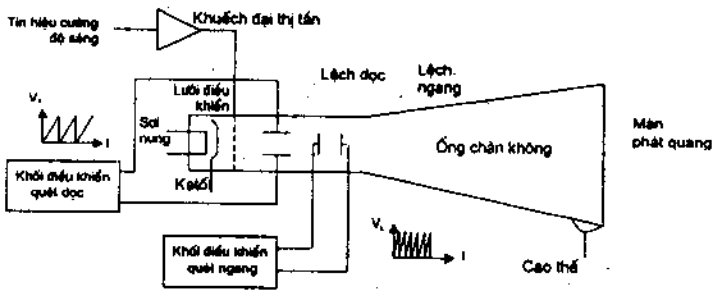
Hiện có hai kiểu quét: xen kẽ (interlaced) và không xen kẽ (non-interlaced) mà nguyên tắc được chỉ ra trên hình 11.1a và b.



Hình 11.1 Quét xen kẽ (a) và không xen kẽ (b).

Kĩ thuật *quét xen kẽ* (interlace scanning) như trình bày trên hình 11.1a, các dòng trong một lần quét từ cạnh đỉnh xuống cạnh đáy (gọi là một mành) được bù trước và xen kẽ với các dòng của mành kế tiếp. Các dòng quét của mành sau nữa sẽ lặp lại dòng quét của mành đầu tiên. Như vậy, để tạo thành một ảnh hoàn chỉnh cần hai mành liên tiếp, hay một khung. Cần

có ba mạch điện tử cơ bản để tạo ra hình ảnh hiển thị trên màn hình CRT là: máy phát dao động dọc dùng để phát tín hiệu điều khiển chùm tia quét theo chiều dọc, máy phát dao động ngang tạo ra các tín hiệu điều khiển chùm tia quét ngang và bộ khuếch đại tín hiệu video dùng để điều khiển cường độ chùm tia điện tử. Một màn hình CRT bao gồm một ống CRT và các mạch điện điều khiển cơ bản trên đây.



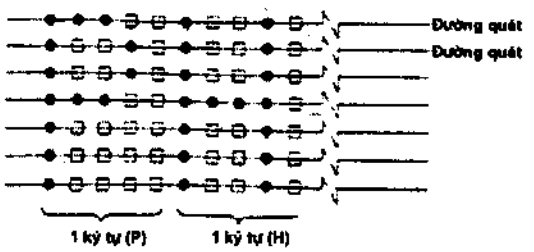
Hình 11.2 Ống tia điện tử và mạch điều khiển.

Mạch điện tử điều khiển một màn hình CRT đơn sắc yêu cầu 3 tín hiệu đầu vào. Nó cần có các xung đồng bộ dòng để đồng bộ máy phát dao động ngang, các xung đồng bộ mành dùng để đồng bộ máy phát dao động dọc và tín hiệu video dùng để điều khiển cường độ chùm tia điện tử quét trên màn hình. Một điểm đặc biệt quan trọng là ba tín hiệu này phải được đồng bộ với nhau để mỗi điểm ảnh luôn được hiển thị tại cùng một vị trí trên màn hình trong mỗi khung. Trường hợp hình ảnh trên màn hình bị trôi hoặc bị méo theo chiều ngang là khi các tín hiệu đồng bộ dòng, đồng bộ mành, và tín hiệu video bị mất đồng bộ. Xét cách tạo ra ba tín hiệu này như thế nào để hiển thị các kí tự lên một màn hình CRT.

11.1.2. Tổng quan về hệ thống điều khiển hiển thị kí tự

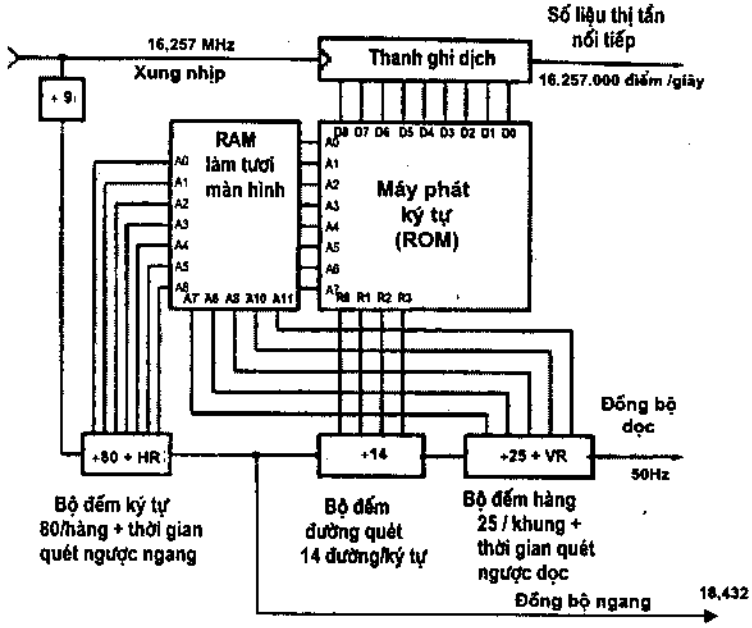
Các kí tự cũng như hình ảnh được hiển thị trên màn hình CRT theo dạng các chấm tối hoặc sáng. Các chấm này được tạo thành khi chùm tia điện tử đập hay không đập vào màn hình trong suốt quá trình quét. Hình 11.3 biểu diễn cách hiển thi các chữ P và H tại góc trên cùng bên trái của màn hình theo cách này.

Các chấm tròn trên hình biểu diễn các điểm ảnh mà tại đó chùm tia đập vào màn hình còn các ô vuông biểu diễn các điểm ảnh tại đó chùm tia không đập vào màn hình. Trong ví dụ này, có thể thấy ma trận điểm cho mỗi kí tự có kích cỡ rộng 5 điểm và cao 7 điểm. Các kích thước thường dùng khác cho các ma trận điểm biểu diễn kí tự là 7×9, 7×12 và 9×14.



Hình 11.3 Hiển thị hai kí tự P và H trên màn hình.

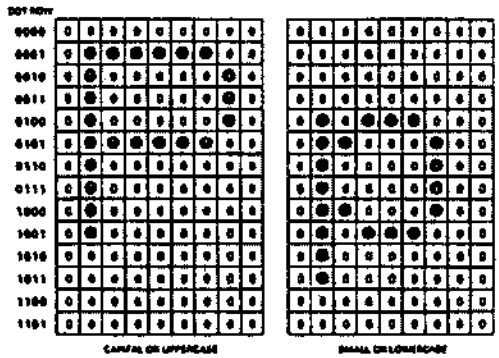
Hình 11.4 trình bày sơ đồ khối của mạch điện để tạo nên các điểm ảnh và hiển thị một trang kí tự lên màn hình. Trong thí dụ này, giả thiết màn hình hiển thị được 25 dòng, mỗi dòng có 80 kí tự. Mã ASCII của các kí tự hiển thị trên màn hình được lưu trong một bộ nhớ RAM. Bộ nhớ này thường được gọi là *bộ đệm khung* (frame buffer) hay *RAM làm tươi màn hình* (display refresh RAM). Trong bộ nhớ này, mỗi kí tự cần ít nhất là 1 byte nhớ. Một màn hình 25 dòng với 80 kí tự mỗi dòng cần bộ nhớ có kích thước 25×80 hay 2 kbyte nhớ.



Hình 11.4 Sơ đồ mạch hiển thị màn hình

Trên thực tế, bộ nhớ RAM này được cài đặt sao cho vi xử lí có thể truy cập để thay đổi các kí tự được lưu bên trong, đồng thời mạch điện làm tươi màn hình cũng có thể truy cập đến nó để làm tươi hình ảnh được hiển thị trên màn hình.

Vị trí các điểm ảnh của mỗi kí tự theo từng dòng quét được nhớ trong một bộ nhớ ROM gọi là *ROM phát kí tự* (character generator ROM). Hình 11.5 trình bày ma trận điểm ảnh của một ROM phát kí tự điển hình.



Hình 11.5 Ma trận điểm ảnh của ROM ký tự

Bộ nhớ ROM này sử dụng ma trận 7×9 cho riêng kí tự, nhưng toàn bộ không gian dùng để hiển thị một kí tự lại là một ma trận 9×14. Các điểm ảnh thêm vào được dùng để tạo khoảng cách giữa hai kí tự cũng như khoảng trống giữa các dòng kí tự. Ngoài ra, khoảng không gian thừa này còn cho phép hiển thị các chữ in thường với các nét kéo dài xuống dưới. Mỗi hàng điểm trong hình 11.5 biểu diễn vị trí các điểm ảnh trên một dòng quét ngang của mỗi kí tự.

Để bắt đầu hiển thị từ góc trái trên của màn hình, các lối ra của *bộ đếm kí tự* và *bộ đếm dòng kí tự* đều bằng 0 cho phép trỏ đến địa chỉ chứa kí tự đầu tiên trong bộ nhớ RAM. Nội dung vùng nhớ này chính là mã ASCII của kí tự đó sẽ được RAM đưa tới lối vào địa chỉ của ROM phát kí tự. Tín hiệu trên những lối vào này mới thực sự nói cho máy phát kí tự biết kí tự nào sẽ được hiển thị. Để theo dõi xem chùm tia đang quét trên dòng quét nào của một dòng kí tự, người ta sử dụng một *bộ đếm dòng quét*. Trong ví dụ đang xét, một dòng kí tự trên màn hình bao gồm 14 hàng điểm ảnh hay 14 dòng quét của tia điện tử nên bộ đếm dòng quét là một bộ đếm modulo 14. Các lối ra của bộ đếm này được nối với 4 lối vào địa chỉ bổ sung của ROM phát kí tự.

Với mã ASCII và số dòng quét đã cho, ROM phát kí tự sẽ phát ra 9 bit biểu diễn hình dạng các điểm ảnh trên 1 hàng điểm trong ma trận điểm của một kí tự. Với dòng quét ngang đầu tiên, bộ đếm dòng quét sẽ phát ra 4 bit 0000, do đó hình dạng các điểm ảnh được quét chính là hàng điểm số 0000 trong ma trận điểm của kí tự.

Dữ liệu được phát song song lên các lối ra của ROM phát kí tự. Để bật/tắt tia điện tử quét ngang màn hình đúng lúc, dạng điểm ảnh này phải được chuyển thành dạng dữ liệu nối tiếp bằng một thanh ghi dịch có lối vào song song, lối ra nối tiếp. Máy phát xung nhịp tần số cao cho thanh ghi dịch được gọi là *đồng hồ điểm ảnh* (dot clock) vì nó điều chỉnh tốc độ truyền thông tin về các điểm ảnh tới bộ khuếch đại tín hiệu video. Có thể thấy trong sơ đồ hình 11.4 đồng hồ điểm ảnh có tần số 16.257 MHz được sử dụng.

Sau khi 9 điểm trên dòng quét đầu tiên của kí tự thứ nhất được phát đi, bộ đếm kí tự tăng lên 1 đơn vị. Vì bộ đếm kí tự được nối với lối vào địa chỉ của RAM làm tươi màn hình nên khi bộ đếm này tăng lên 1 đơn vị, địa chỉ chứa mã ASCII của kí tự tiếp theo trên dòng đầu tiên sẽ được trỏ tới. Mã ASCII của kí tự thứ hai này được chuyển tới ROM phát kí tự. Vì bộ đếm dòng quét vẫn giữ các lối ra ở giá trị 0000, bộ nhớ ROM sẽ phát ra 9 bit thông tin về các điểm ảnh trên dòng quét đầu tiên của kí tự thứ hai của dòng kí tự đầu tiên trên màn hình. Khi tất cả các điểm ảnh này được phát đi, bộ đếm kí tự lại tăng lên 1 đơn vị, quá trình lặp lại đối với kí tự thứ ba của dòng đầu tiên. Quá trình cứ thế tiếp diễn cho tới khi dòng quét đầu tiên cho toàn bộ 80 kí tự hoàn thành.

Lúc này, xung đồng bộ dòng được tạo ra sẽ đưa chùm tia quay trở về cạnh trái màn hình. Sau khi chùm tia quay trở về, bộ đếm kí tự quay lại đếm từ 0, trỏ tới mã ASCII của kí tự đầu tiên. Bộ đếm dòng quét (R0-R3) được tăng lên 0001 khiến cho bộ phát kí tự giờ đây phát ra hình dạng các điểm ảnh của các kí tự theo dòng quét thứ hai. Sau khi dòng quét thứ hai đối với kí tự đầu tiên được đưa tới bộ khuếch đại tín hiệu video, bộ đếm kí tự được tăng lên 1 đơn vị, trỏ tới mã ASCII của kí tự thứ hai trong RAM màn hình. Quá trình lặp lại cho đến khi tất cả 14 dòng quét của dòng kí tự đầu tiên được thực hiện.

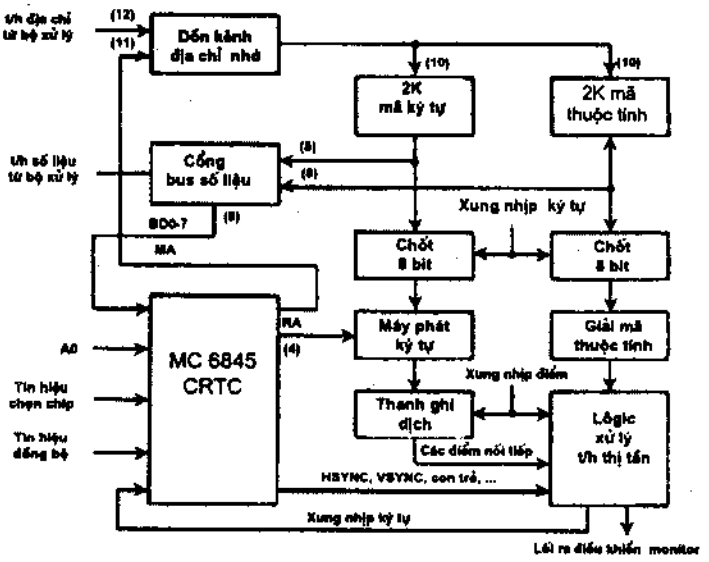
Bộ đếm dòng kí tự sẽ tăng lên 1 đơn vị. Lối ra của bộ đếm kí tự và bộ đếm dòng kí tự bây giờ sẽ trỏ tới địa chỉ vùng nhớ trong RAM màn hình chứa mã ASCII của kí tự đầu tiên trên dòng kí tự thứ hai. Quá trình lặp lại như đã mô tả đối với dòng kí tự thứ nhất. Sau khi quét xong dòng kí tự thứ hai, chùm tia điện tử tiếp tục quét sang dòng kí tự thứ 3, rồi thứ 4 cho đến khi quét hết 25 dòng kí tự trên màn hình.

Khi đã quét xong toàn bộ các dòng kí tự, chùm tia lúc này ở góc phải dưới của màn hình. Mạch đếm sẽ phát ra một xung đồng bộ dòng và một xung đồng bộ mành đưa chùm tia quay trở lại góc trái trên màn hình. Khi chùm tia trở về góc trái trên màn hình, toàn bộ *quá trình làm tươi màn hình* (screen-refresh process) mà ta vừa mô tả phía trên sẽ lặp lại. Như đã nhắc đến trên đây, để tránh hiện tượng màn hình nhấp nháy, cần phải quét (làm tươi) màn hình từ 30 đến 75 lần trong một giây. Trong sơ đồ hình 11.4 tốc độ làm tươi màn hình là 50Hz.

11.1.3. Mạch ghép nối đồ hoạ của màn hình đơn sắc

Hình 11.6 trình bày sơ đồ khối của bản mạch ghép nối đồ hoạ trong máy tính cá nhân IBM. Tuy hiện nay có những bản mạch có tính năng mạnh hơn nhiều nhưng do kết cấu đơn giản nên việc khảo sát bản mạch này vẫn là một thí dụ tốt cho các mạch điện tương tự. Nhìn vào hình 11.6 ta sẽ nhanh chóng tìm được bộ điều khiển CRT (CRT controller), bộ phát kí tự, và thanh ghi dịch giống như sơ đồ hình 11.3 đã nói. Tiếp đó, sẽ là bộ nhớ 2 kbyte dùng để lưu trữ mã ASCII của các kí tự được hiển thị. Bên phải bộ nhớ này là một bộ nhớ 2 kbyte khác

dùng để lưu trữ *mã thuộc tính* (attribute code) của mỗi kí tự. Mỗi mã thuộc tính xác định kí tự sẽ được hiển thị một cách bình thường hay có gạch dưới, cường độ sáng được tăng lên hay giảm đi, có nhấp nháy hay không.



Hình 11.6 Sơ đồ khối mạch màn hình đơn sắc.

Hình 11.7 chỉ ra ý nghĩa của các bit trong byte thuộc tính.

Byte mã kí tự								Byte thuộc tính							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Địa chỉ chẵn								Địa chỉ lẻ							

Thuộc tính	Byte thuộc tính							
	7	6	5	4	3	2	1	0
	B	R	G	B	1	R	G	B
	FG	màu nền			nền trước			
Hiện bình thường	B	0	0	0	1	1	1	1
Đảo màu	B	1	1	1	1	0	0	0
Không hiện (màu đen)	B	0	0	0	1	0	0	0
Không hiện (màu trắng)	B	1	1	1	1	1	1	1

Hình 11.7 Byte thuộc tính.

Như đã đề cập trong phần trước, các mã ASCII trong RAM làm tươi màn hình được chuyển tới bộ phát kí tự. Ngoài ra, đi vào bộ phát kí tự còn có 4 đường địa chỉ xác định số dòng quét. Bộ đếm dùng để phát ra địa chỉ này nằm trong vi mạch điều khiển CRT MC6845. Lối ra của bộ phát kí tự được đưa vào một thanh ghi dịch để biến đổi thành dữ liệu nối tiếp truyền đến bộ khuếch đại tín hiệu video. Thanh ghi dịch được cấp xung nhịp với tần số 16.257 MHz từ đồng hồ điểm ảnh. Mạch điện trong khối lôgic xử lí tín hiệu video (video process logic section) chia tần số này cho 9 để thu được tín hiệu xung nhịp đếm kí tự ở tần số 1 787 904 Hz. Tín hiệu này sẽ tạo thành một xung mỗi khi cần lấy một kí tự ASCII nào đó ra khỏi RAM màn hình cũng như byte thuộc tính ra khỏi RAM thuộc tính.

Nhận xét tiếp theo là có một bộ hợp kênh nối tiếp với các đường địa chỉ đi vào các bộ nhớ kí tự và bộ nhớ thuộc tính. Bộ hợp kênh này cho phép CPU hoặc bộ điều khiển CRT có thể truy cập RAM làm tươi màn hình.

Để làm tươi màn hình, vi mạch điều khiển 6845 gửi tới bộ nhớ các địa chỉ của mã kí tự và mã thuộc tính. Xung nhịp đếm kí tự sẽ chốt mã kí tự vào bộ phát kí tự và mã thuộc tính vào mạch giải mã thuộc tính (attribute decode circuitry). Xung nhịp đếm kí tự cũng tăng bộ đếm kí tự bên trong vi mạch 6845 để trỏ đến kí tự tiếp theo. Đến xung đếm kí tự tiếp theo, mã kí tự và mã thuộc tính kế tiếp sẽ được chuyển vào bộ phát kí tự và bộ giải mã thuộc tính. Quá trình tiếp diễn đối với tất cả các kí tự trên một trang màn hình rồi sau đó được lặp lại.

Bây giờ, mỗi khi muốn hiển thị các kí tự mới lên màn hình, chỉ cần cho CPU thực hiện một số câu lệnh ghi mã ASCII của những kí tự đó vào các địa chỉ thích hợp trong RAM màn hình. Mỗi khi mạch giải mã địa chỉ kiểm tra một địa chỉ nào đó trong RAM màn hình, nó sẽ phát một tín hiệu đến bộ hợp kênh cho phép CPU truy cập đến RAM màn hình. Đến đây phát sinh một câu hỏi thú vị, điều gì sẽ xảy ra nếu cả CPU lẫn vi mạch 6845 cùng muốn truy cập vào RAM màn hình tại cùng một thời điểm ? Có nhiều giải pháp cho vấn đề này, một trong số đó chỉ cho phép CPU truy cập RAM màn hình trong thời gian tia điện tử quay về sau khi quét ngang hoặc/và sau khi quét dọc. Một giải pháp khác là sắp xếp các chu trình truy cập bộ nhớ của CPU xen kẽ với các chu trình truy cập bộ nhớ của vi mạch 6845. Đây chính là cách làm việc của bản mạch đồ hoạ đơn sắc trong máy vi tính IBM.

CPU được phép truy cập RAM màn hình trong nửa chu kì xung nhịp đếm kí tự, nửa chu kì còn lại được dành cho vi mạch 6845. Nếu như CPU vẫn cố truy cập RAM màn hình trong khoảng thời gian dành cho vi mạch điều khiển, 6845 sẽ phát ra một tín hiệu chưa sẵn sàng (not-ready signal) khiến cho CPU phải xen vào một chu kì đợi (wait cycle) cho tới khi được phép truy cập trở lại. Vi mạch điều khiển 6845 trên hình 11.6 bao gồm một chuỗi các bộ đếm được vẽ trên hình 11.4 và các mạch điện tử khác dùng để phát ra các xung xoá tia quét ngược ngang và dọc, tạo con trỏ, cuộn màn hình v.v... Các nhà sản xuất đề xuất nhiều vi mạch điều khiển CRT với số mạch điện chức năng khác nhau. Vi mạch MC 6845 của hãng Motorola có thể sử dụng cho cả bản mạch ghép nối đồ hoạ đơn sắc cũng như màu trong máy tính cá nhân IBM.

11.1.4. Các tần số và vấn đề định thời trong màn hình CRT

Có rất nhiều tần số khác nhau được dùng cho tín hiệu đồng bộ dòng, đồng bộ mành và làm tần số quét điểm ảnh. Tần số tín hiệu đồng bộ dòng thường nằm trong khoảng từ 15 đến 50 kHz, tần số tín hiệu đồng bộ mành thường là 50 hoặc 60 Hz, còn tần số quét điểm ảnh thường từ 10 cho tới 100 MHz. Ví dụ thứ nhất: hãy quan sát kĩ các tần số được dùng trong bản mạch ghép nối đồ hoạ đơn sắc của máy tính cá nhân IBM mà ta đã thảo luận ở phần trên.

Để làm tươi bộ nhớ, mạch ghép nối đồ hoạ đơn sắc của máy tính cá nhân IBM hiển thị lên màn hình 25 dòng, mỗi dòng chứa 80 kí tự. Mỗi kí tự được tạo ra trong phạm vi một ma trận 7×9 điểm ảnh trong một không gian 9×14 điểm ảnh. Sở dĩ như vậy là do luôn có các khoảng trống bao quanh mỗi kí tự, cho nên trên thực tế một kí tự chiếm đến 9 điểm ảnh theo chiều ngang và 14 điểm ảnh theo chiều dọc.

Do vậy, diện tích hiển thị thực sự là 9 điểm ảnh/ kí tự × 80 kí tự/dòng hay 720 điểm ảnh/dòng. Số dòng quét thực tế trên màn hình là 25 dòng × 14 dòng quét/dòng hay 350 dòng quét. Theo tài liệu kĩ thuật tham khảo của hãng IBM, mạch ghép nối đồ hoạ đơn sắc có tần số quét điểm ảnh là 16,257 MHz. Điều này có nghĩa là thanh ghi dịch tín hiệu video phát ra thông tin của 16 257 000 điểm ảnh / giây. Tài liệu kĩ thuật cũng cho biết bản mạch này sử dụng tín

hiệu đồng bộ dòng có tần số 18 432 dòng/giây. Chia 16 257 000 điểm ảnh trong thời gian 1 giây cho 18 432 dòng quét được trong một giây, sẽ thấy bản mạch quét 882 điểm ảnh trên một dòng. Nhưng ta vừa chứng minh trong phần trên diện tích hiển thị thực sự chỉ là 720 điểm ảnh/dòng. Thời gian quét 162 điểm ảnh dôi ra này chính là thời gian để tia điện tử quét ngược trở lại cạnh trái của màn hình để chuẩn bị quét tiếp dòng mới. Sở dĩ cần số điểm ảnh dôi ra lớn như vậy là vì phần lớn màn hình đều có hiện tượng *quét quá* (over scan) tương đối lớn. Quét quá có nghĩa là chùm tia điện tử lượt đi và lượt về đều quét quá cạnh phải và cạnh trái màn hình khá xa.

Tài liệu kĩ thuật cũng cho biết tốc độ khung là 50 Hz. Nói cách khác, chùm tia quét từ cạnh trên xuống đến cạnh đáy màn hình 50 lần trong một giây. Để xem có bao nhiêu dòng quét trong một khung, lấy 18 432 dòng/giây chia cho 50 khung/giây sẽ được 369 dòng quét/khung. Chúng ta cũng đã tính ở phần trên màn hình chỉ có 350 dòng. Thời gian quét hết 19 dòng dư ra chính là thời gian cần thiết để chùm tia quay lên cạnh trên của màn hình.

Một điểm nữa đáng quan tâm ở đây là dải thông của bộ khuếch đại tín hiệu video trong màn hình. Để tạo ra hình ảnh sắc nét, bộ khuếch đại tín hiệu video cần phải chuyển từ trạng thái "on" sang trạng thái "off" đủ nhanh sao cho hai điểm ảnh kề nhau không cùng màu (không cùng trạng thái). Trong ví dụ của chúng ta, tần số quét điểm ảnh là 16,257 MHz, có nghĩa là thanh ghi dịch điểm ảnh "dịch" ra ngoài 16 257 000 điểm ảnh/giây. Nếu các điểm ảnh ở trạng thái sáng và tối xen kẽ nhau, thì tín hiệu tại chân ra nối tiếp của thanh ghi dịch là một sóng vuông có tần số 8,1285 MHz. Như vậy, để tạo được hình ảnh rõ nét trên màn hình, bộ khuếch đại tín hiệu video phải có dải thông nhỏ nhất là 8MHz.

Yêu cầu về dải thông chính là lí do khiến cho các máy thu hình bình thường khi nối với máy tính không thể hiển thị được 80 kí tự/dòng với độ nét cao. Để lọc sóng mang tín hiệu âm thanh có tần số 4,5 MHz (theo tiêu chuẩn FCC của Mĩ) và sóng mang tín hiệu màu có tần số 3,58 MHz thì bộ khuếch đại tín hiệu video của TV chỉ có dải thông là 3 MHz hoặc nhỏ hơn.

Điểm cuối cùng cần để cập đến là vấn đề định thời cho màn hình CRT: sau bao lâu RAM làm tươi màn hình phải được truy cập. Vì mạch điều khiển quét màn hình theo từng dòng một, do đó nó cần truy cập đến một kí tự mới trong RAM sau khi 9 điểm ảnh đã được "dịch" khỏi thanh ghi, với giả thiết 1 kí tự có độ rộng 9 điểm ảnh. Thực hiện phép chia tần số quét điểm ảnh 16 257 000 điểm ảnh/giây cho 9 điểm ảnh/kí tự sẽ cho ta tốc độ các kí tự được đọc từ RAM ra ngoài vào khoảng 1 806 333 kí tự/giây hay mỗi kí tự được đọc ra trong 553 ns! Như đã nói đến trong phần trước, bộ điều khiển CRT truy cập RAM màn hình trong một nửa thời gian này, nửa thời gian còn lại được dành cho CPU. Như vậy thời gian để truy cập RAM màn hình trên thực tế chỉ còn khoảng 200 ns trong mỗi nửa chu kì xung nhịp đếm kí tự. Sau này, ta sẽ thấy với các màn hình có độ phân giải cao hơn và màn hình màu thậm chí còn đòi hỏi bộ nhớ có tốc độ nhanh hơn nữa.

11.2. CÁC MÀN HÌNH QUÉT MÀNH TRONG CHẾ ĐỘ ĐỒ HOẠ

11.2.1. Màn hình đơn sắc trong chế độ đồ hoạ

Các kí tự được hiển thị lên màn hình bằng cách gửi tín hiệu về chuỗi các điểm ảnh sáng tối đến bộ khuếch đại video. Mã ASCII của các kí tự này được lưu trong RAM làm tươi màn hình. Trên hình 11.3, ROM phát kí tự sử dụng mã ASCII lấy từ RAM và một mã 4 bit lấy từ bộ đếm dòng quét để tạo nên hình mẫu điểm ảnh cho một dòng quét xác định của kí tự. Bây giờ giả sử bộ phát kí tự bị loại bỏ khỏi mạch điện và các lối ra của RAM được nối trực tiếp với một thanh

ghi dịch 8 bit. Thêm nữa, giả sử rằng thay vì lưu mã ASCII của các kí tự vào trong RAM, chúng ta lưu vào các ô nhớ hình mẫu (pattern) mong muốn cho cụm 8 điểm ảnh liên tiếp trên một dòng quét.

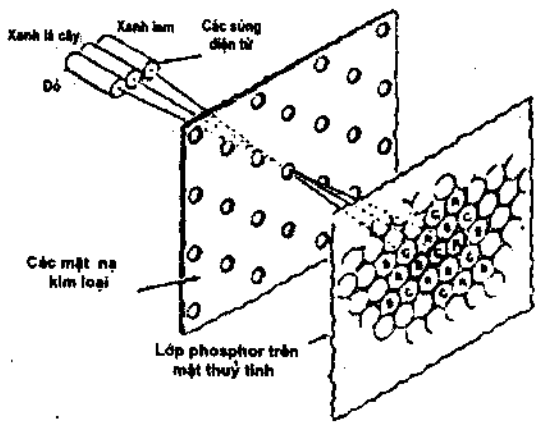
Khi một byte được đọc ra khỏi RAM và nạp vào thanh ghi dịch, trạng thái của từng điểm ảnh sẽ được phát ra lần lượt để điều khiển chùm tia điện tử tạo thành một dải 8 điểm ảnh mong muốn trên dòng quét. Quá trình này tiếp diễn cho tới khi mọi điểm ảnh trên màn hình đều đã được quét. Toàn bộ màn hình lúc này có thể xem là một ma trận các điểm ảnh. Mỗi điểm ảnh đều có thể lập trình cho sáng hoặc tối bằng cách ghi vào các ô nhớ tương ứng trong RAM các giá trị 1 hoặc 0. Màn hình đồ hoạ được tạo ra theo cách này được gọi là *màn hình quét mành ảnh bit (ánh xạ bit)* (bit-mapped raster scan display). Mỗi điểm ảnh (hay trong một số trường hợp là khối các điểm ảnh) được gọi là một *phần tử ảnh* (picture element), viết tắt là *pixel* hoặc *pel*. Ở đây, chúng ta giả thiết mỗi pixel là một điểm ảnh.

Giả sử một màn hình đơn sắc hiển thị 640 pixel theo chiều ngang và 200 pixel theo chiều dọc. Như vậy, trên màn hình có tổng cộng 200×640 hay 128 000 điểm ảnh. Vì mỗi điểm ảnh tương ứng với một ô nhớ nên ta cần một bộ nhớ RAM có dung lượng tối thiểu là 128 000 bit hay 16 kbyte để lưu giữ thông tin chỉ cho một trang màn hình. Hãy so sánh với dung lượng 4 kbyte để lưu giữ mã ASCII và thuộc tính cho một trang màn hình 25 dòng, mỗi dòng có 80 kí tự. Sẽ được thấy ngay sau đây để tạo ra một màn hình đồ hoạ màu có số pixel tương đối lớn còn cần nhiều bộ nhớ hơn nữa.

11.2.2. Màn hình màu và đồ hoạ màu

Màn hình CRT đơn sắc chỉ được phủ một loại phosphor phát ra một màu xác định mỗi khi bị bắn phá bởi các điện tử từ một súng phóng điện tử duy nhất. Để tạo ra màn hình CRT màu, người ta phủ lên mặt trong ống CRT từng cụm điểm ảnh hay một dải các loại phosphor đỏ, xanh lá cây và xanh da trời. Một trong những xu hướng thông dụng nhất là phủ 3 loại phosphor này trên một đường thẳng như vẽ trong hình 11.8.

Các cụm điểm ảnh được đặt gần nhau đến mức đối với mắt người, chúng xuất hiện chỉ như một điểm ảnh hay một pixel mà thôi. Người ta dùng ba súng phóng điện tử riêng biệt để bắn vào 3 loại phosphor này. Một mặt nạ đặt ngay sau màn hình ngăn không cho các điện tử của màu này rơi vào lớp phosphor của màu khác. Khoảng cách giữa các lỗ trên mặt nạ hay giữa các pixel trên màn hình được gọi là *pitch*. Khoảng pitch của các màn hình CRT thông thường nằm trong khoảng 0,21 mm đến 0,66 mm.



Hình 11.8 Các cụm điểm phosphor 3 màu dùng để tạo ra các điểm ảnh màu trên màn hình CRT.

Với khoảng pitch và kích thước điểm ảnh càng nhỏ thì càng tạo được nhiều điểm ảnh hơn trên màn hình và do vậy màn hình có được độ phân giải cao hơn. Tuy nhiên cần phải có sự thoả hiệp bởi nếu kích thước điểm ảnh và pitch nhỏ đi thì cường độ chùm tia điện tử phải tăng lên mới khiến cho cường độ sáng phát ra có thể chấp nhận được. Đồng thời, do khả năng chùm tia điện tử đập vào

mặt nạ che tăng lên và cường độ chùm tia mạnh dẫn đến việc làm cho mặt nạ che quá nóng và bị cong vênh khiến cho hình ảnh trên màn hình bị méo.

Màn hình CRT hiển thị màu theo kiểu này thường được gọi là màn hình RGB hay màn hình RGBI. Ngoài các lối vào cho tín hiệu đỏ, xanh lá cây và xanh da trời, màn hình RGB còn có một lối vào tín hiệu đồng bộ dòng, một lối vào tín hiệu đồng bộ mành, và trong một số trường hợp còn có thêm lối vào cường độ sáng. Một màn hình RGB phải được thiết kế để hoạt động được với định dạng hiện ảnh và các tần số đồng bộ phù hợp với các mạch điện trong máy vi tính. Tuy nhiên, một số màn hình như NEC Multisync, Sony Multiscan, và Magnavox MVX9CM còn có khả năng hoạt động tốt với nhiều định dạng và các tần số đồng bộ khác nữa.

Đối với mắt người, màu sắc của một pixel được xác định bởi tỉ lệ cường độ giữa 3 tia điện tử và cường độ tổng cộng của cả ba chùm tia. Bảng 11.1 liệt kê 16 màu có thể tạo ra nhờ sự pha trộn các trạng thái bật/ tắt của 3 tín hiệu màu R, G, B và tín hiệu cường độ I. Bit I được đặt bằng 1 khi ta muốn tăng cường độ tất cả các chùm tia để tạo ra các sắc màu tươi. Nếu tất cả các chùm tia đều được đặt ở trạng thái "tắt", điểm ảnh được quét sẽ có màu đen và ngược lại, nếu tất cả các chùm tia đều được "bật", điểm ảnh được quét sẽ có màu trắng. Các sự pha trộn khác sẽ tạo ra 14 sắc độ như trên bảng chỉ ra.

Bảng 11.1 **Bảng pha trộn 3 màu**

I	R	G	B	Màu
0	0	0	0	Đen
0	0	0	1	Xanh lam
0	0	1	0	Xanh lá cây
0	0	1	1	Xanh côban
0	1	0	0	Đỏ
0	1	0	1	Đỏ tươi
0	1	1	0	Nâu
0	1	1	1	Trắng
1	0	0	0	Xám
1	0	0	1	Lam sáng
1	0	1	0	Xanh lá cây sáng
1	0	1	1	Xanh Côban sáng
1	1	0	0	Đỏ sáng
1	1	0	1	Đỏ tươi sáng
1	1	1	0	Vàng
1	1	1	1	Trắng sáng

Để có được dải màu phong phú hơn, người ta phải thiết kế các màn hình mới có khả năng xử lí các tín hiệu màu RGB tương tự thay cho các tín hiệu RGB số như trên. Các tín hiệu này được tạo ra nhờ các bộ biến đổi D/A. Ví dụ nếu sử dụng một bộ biến đổi D/A 2 bit cho mỗi tín hiệu màu ta sẽ nhận được 4×4×4 hay 64 màu. Tuy nhiên, để tăng số màu sẽ phải tăng dung lượng nhớ cho bộ đệm khung cũng như tốc độ truy cập bộ nhớ.

11.2.3. Các bảng màu, trang điểm ảnh và VRAM

Đối với một màn hình đồ hoạ đơn sắc, dữ liệu về một pixel chỉ cần lưu vào 1 bit đơn trong RAM màn hình. Các màn hình màu đòi hỏi nhiều hơn một 1 bit cho mỗi pixel vì còn phải lưu các dữ liệu về màu: đỏ, xanh lá cây và xanh da trời cho mỗi pixel. Ví dụ phải cần 2 bit mới xác định được 1 màu trong dải 4 màu, cần 3 bit mới xác định được một màu trong dải 8 màu, 8 bit mới xác định được một màu trong dải 256 màu v.v... Như vậy số màu ta muốn hiển thị lên màn hình có ảnh hưởng trực tiếp đến dung lượng nhớ của bộ đệm khung. Giả sử ta muốn hiển thị trên một màn hình 256 màu có kích thước 640×480 pixel trong một hệ 8086. Tổng số pixel trên màn hình là 640×480 = 307 200. Để hiển thị 256 màu, mỗi pixel đòi hỏi 8 bit (1byte). Do đó, dung lượng bộ đệm khung là 307 200 byte. Bên cạnh giá cả thì con số này cũng đã vượt quá dung lượng nhớ dành cho màn hình vì hệ thống chỉ có thể định địa chỉ cho tối đa 1 Mbyte bộ nhớ. Để giảm số bit dùng để lưu trữ thông tin của một pixel mà vẫn hiển thị được màu sắc trong một dải rộng, người ta sử dụng một *bảng màu* (palette scheme). Thuật ngữ bảng màu ở đây cũng có ý nghĩa tương tự như các hoạ sĩ vẫn dùng. Bảng màu của các hoạ sĩ được dùng để chứa đựng các màu vẽ mà hoạ sĩ có. Có thể hoạ sĩ có tới 16 màu trên bảng màu nhưng để vẽ

một bức tranh cụ thể nào đó, hoạ sĩ chỉ cần đến 4 màu mà thôi. Trong trường hợp này chúng ta có thể nói hoạ sĩ đã chọn 4 màu từ bảng 16 màu.

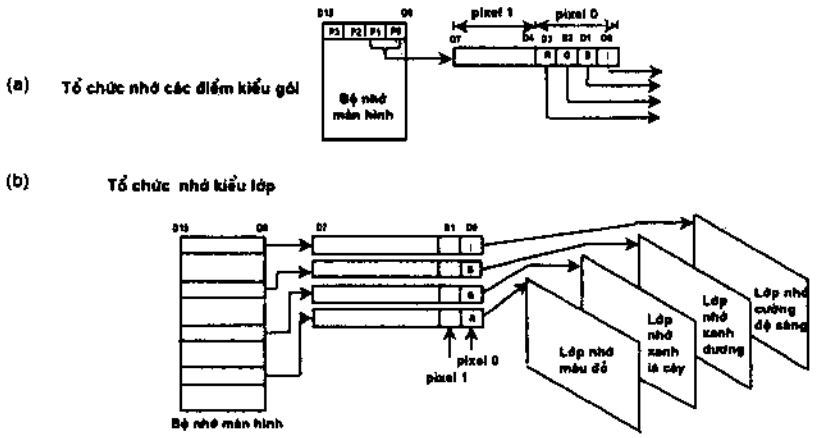
Thí dụ thứ nhất, một bản mạch ghép nối đồ hoạ màu (Color Graphics Adapter -CGA) của IBM có thể hiển thị ở độ phân giải trung bình (320×200 pixel) 4 màu từ một bảng 16 màu. Vì chỉ có 4 màu được hiển thị tại cùng một thời điểm nên chỉ cần 2 bit cho dữ liệu của mỗi pixel.

Thí dụ thứ hai, giả sử chúng ta cho một hệ 640×480 pixel hiển thị 16 màu từ một bảng 256 màu, thay vì hiển thị cả 256 màu. Chỉ cần 4 bit để chứa dữ liệu cho 1 trong 16 màu mà mỗi pixel có thể có, do vậy kích thước bộ đệm khung chỉ bằng một nửa so với hiển thị trực tiếp 256 màu.

Một giới hạn khác của việc thiết kế các hệ đồ hoạ màu có độ phân giải cao là tốc độ mà dữ liệu của pixel được đọc ra khỏi bộ đệm. Giả sử ta muốn một màn hình 640×480 pixel hiển thị 256 màu. Như đã nói phía trên, mỗi pixel yêu cầu 1 byte trong bộ nhớ hay bộ nhớ phải có tổng cộng 307 200 byte. Giả sử tốc độ khung là 50 Hz, mỗi byte phải được đọc ra khỏi bộ nhớ 50 lần trong một giây. Điều này có nghĩa là trong một giây phải đọc ra 15 360 000 byte trong một giây hay đọc một byte trong 65 ns (trên thực tế còn nhỏ hơn vì tất cả 307 200 lượt truy cập phải được thực hiện chỉ trong thời gian hiển thị thực sự mà thôi). Thời gian thực hiện một chu kì đọc đối với các chip DRAM thông thường lớn hơn nhiều 65 ns. Điều này có nghĩa là chúng ta không thể dùng DRAM cho bộ đệm khung trừ khi tìm ra cách thức cho phép mỗi lần truy cập xảy ra trong khoảng thời gian dài hơn. Có một số phương pháp cho phép thực hiện điều này.

Bước đầu tiên cần làm nhằm mang lại nhiều thời gian hơn cho bộ điều khiển làm tươi màn hình trong mỗi lần truy cập bộ đệm khung là chỉ cho phép bộ vi xử lí truy cập bộ đệm khi tia điện tử quét ngược. Điều này sẽ mang lại toàn bộ khoảng thời gian giữa hai kí tự hay hai pixel được dành cho bộ điều khiển thay vì phải chia sẻ thời gian (timesharing) như trong bản mạch ghép nối đồ hoạ đơn sắc trước đây.

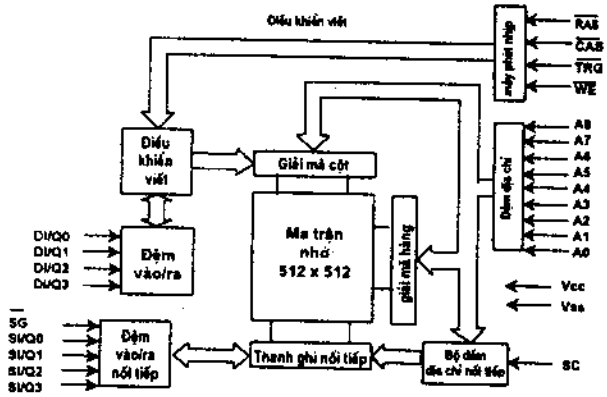
Cách thứ hai làm giảm tốc độ truy cập bộ nhớ là sử dụng bảng màu để giảm số bit lưu trữ dữ liệu của mỗi pixel. Như đã trình bày, hiển thị 16 màu từ bảng 256 màu thay vì hiển thị trực tiếp 256 màu giúp ta cắt giảm được một nửa dung lượng bộ nhớ. Vì chỉ cần 4 bit để biểu diễn 1 màu trong dải 16 màu nên dữ liệu của 4 pixel có thể gói gọn trong một từ (word) như hình 11.9a.



Hình 11.9 Các cấu hình bộ nhớ đệm khung pixel dạng gói (a) và nhớ lớp (b).

Do đó mỗi lần truy cập bộ nhớ sẽ lọc ra dữ liệu của 4 pixel. Trong phương pháp thứ 3, tốc độ truy cập bộ đệm khung được giảm xuống bằng cách chia bộ nhớ thành các lớp nhớ song

song với nhau. Hình 11.9b minh hoạ phương pháp này qua sơ đồ khối của một hệ thống yêu cầu 4 bit cho mỗi điểm ảnh. Có thể thấy 4 bit dữ liệu của mỗi điểm ảnh được lưu tại 4 bit có cùng vị trí trong 4 ô nhớ khác nhau. Khi bộ điều khiển đọc dữ liệu chứa trong 4 ô nhớ, độ dài mỗi ô nhớ là một từ (word) vào các thanh ghi của mình, nó sẽ có tất cả các số liệu cần thiết về 16 điểm ảnh. Trong khoảng thời gian tiến hành quét 16 điểm ảnh này, bộ nhớ DRAM sẽ được hồi phục và có thể truy cập trở lại. Người ta có thể thêm vào nhiều lớp nhớ song song nhằm lưu trữ được nhiều bit hơn cho mỗi điểm ảnh. Vẫn còn một cách nhằm để giải quyết vấn đề tốc độ truy cập bộ nhớ là xây dựng bộ đệm khung bằng các chip DRAM đặc biệt gọi là *video RAM* hay *VRAM*. Hình 11.10 giới thiệu sơ đồ khối của bộ nhớ video RAM TI TMS44C251.



Hình 11.10 Sơ đồ khối của VRAM TI TMS44C251.

Mỗi khối DRAM bao gồm 4 mảng, dung lượng mỗi mảng là 256 kbit. Đối với mạch điều khiển DRAM và bộ vi xử lí, vi mạch này có chức năng như 4 vi mạch 256 k dùng cho các hoạt động đọc và ghi dữ liệu. Bộ điều khiển DRAM sẽ cung cấp các tín hiệu RAS, CAS, địa chỉ hợp kênh, và tín hiệu làm tươi cho nó, giống như các chip DRAM bình thường khác. Để xuất dữ liệu tới bộ điều khiển, 512 bit được lưu trên mỗi hàng trong một mảng DRAM được đồng thời chuyển đến một thanh ghi 512 bit. Các lối ra của thanh ghi này được nối với một bộ hợp kênh 512 lối vào có nhiệm vụ dẫn mỗi một lối ra của thanh ghi đến một lối ra SDQ. Bộ hợp kênh xuất ra lần lượt 512 bit một tới lối ra SDQ. Ở đây VRAM có thể xuất ra dữ liệu của 512 pixel một cách nhanh chóng. Ví dụ TMS44C251 có thể xuất dữ liệu ở tốc độ 33 MHz, nhanh hơn tốc độ cần thiết đối với màn hình 800×512 pixel (không tính đến quá trình quét quá, tốc độ này là: (800×512 pixel/frame) × 60 frame/s = 24 576 000 pixel/s).

VRAM có thể lưu trữ dữ liệu theo dạng gói (packed pixel format) hay dạng mặt phẳng (planar format). Có lẽ khuôn dạng dễ hình dung nhất là dạng mặt. Một vi mạch đơn TMS44C251 có thể chứa được 4 mặt nhớ toàn bộ bit cho màn hình 512×512 pixel. Các vi mạch có thể ghép tầng với nhau để quét dòng dài hơn 512 pixel hay tạo ra nhiều mặt nhớ hơn. Tuy VRAM đắt hơn DRAM tiêu chuẩn nhưng chúng làm cho công việc trở nên tương đối dễ dàng khi ứng dụng vào trong các màn hình có độ phân giải cao.

11.3. CÁC CHUẨN MÀN HÌNH THÔNG DỤNG VÀ PHẦN CỨNG

11.3.1 . Giới thiệu chung

Có rất nhiều cấu hình phần cứng và định dạng dành cho màn hình máy tính. Những chuẩn màn hình chính của nhiều loại máy tính tương thích IBM được nêu trong bảng 11.2.

Theo thứ tự từ trên xuống dưới, bảng này điểm qua những bước phát triển chủ yếu trong lĩnh vực đồ hoạ dành cho máy tính cá nhân IBM trong vòng những năm trở lại đây.

Khi mua các máy tính này, cần phải lựa chọn tiêu chuẩn mong muốn rồi mua bản mạch ghép nối đồ hoạ tương ứng với chuẩn ấy, cắm vào một khe cắm trên bản mạch chính và nối nó với một màn hình tương thích.

Như đã nói trong phần trên, bản mạch ghép nối đồ hoạ đơn sắc IBM chỉ hiển thị được một màn hình 80×25 kí tự mà không thể tạo ra các hình ảnh đồ hoạ. Bước cải tiến dựa trên bản mạch ghép nối đơn sắc cơ bản là mạch ghép nối của hãng Hercules. Nó có thể hiển thị kí tự hay đồ hoạ đơn sắc với độ phân giải 720×348 pixel.

Mode đồ hoạ thông dụng nhất của bản mạch ghép nối đồ hoạ CGA (Color Graphics Adapter) là chế độ đồ hoạ có độ phân giải trung bình (320×200 pixel) hiển thị 4 màu từ palette 16 màu. Ngoài ra, bản mạch ghép nối đồ hoạ CGA còn có chế độ hiển thị các chữ số và kí tự. Tuy nhiên, trong chế độ này, mỗi kí tự chỉ được hiển thị trong khuôn khổ một ma trận 8×8 điểm ảnh khiến cho người sử dụng rất khó đọc.

Bảng 11.2

Card	Độ phân giải	Bảng màu	Tín hiệu
CGA	25 x 80	4/16	Số
	160 x 100	4/16	Số
	320 x 200	4/16	Số
	640 x 400	2/16	Số
Hercules	720 x 348	2	Số
EGA	640 x 350	16/64	Số
MGA	320 x 200	256	Analog
	640 x 480	2	Analog
VGA	640 x 480	2	Analog
	640 x 480	16/256k	Analog
	640 x 200	256/256k	Analog
SUPER-VGA1	640 x 480	256/256k	Analog
	1024 x 768	256/256k	Analog

Để khắc phục nhược điểm này, hãng IBM đã phát triển chuẩn EGA (Enhanced Graphics Adapter). Card đồ họa EGA hiển thị kí tự trên một ma trận 8×14 điểm ảnh trong chế độ đồ hoạ, do đó văn bản trở nên dễ đọc hơn nhiều so với card CGA. Mạch ghép nối đồ hoạ EGA ngoài các mode đồ hoạ CGA còn có thể hoạt động trong một số mode đồ hoạ khác như hiển thị màn hình 640×350 pixel với 16 màu từ palette 64 màu. Để có thể hiển thị được tất cả 64 màu trong mode này, ngoài các lối vào đỏ, xanh lá cây và xanh lam chuẩn màn hình còn phải có các lối vào đỏ đậm, xanh lá cây đậm và xanh lam đậm. Các cải tiến tiếp theo ra đời cùng dòng máy PS/2 của IBM với mạch điều khiển CRT được gắn trên bản mạch chính. Chuẩn MCGA (Multicolor Graphics Array) được áp dụng trên các máy tính PS/2 model 25 và 30 có thể hiển thị tất cả các mode EGA cộng thêm một vài mode khác. Trong số đó đáng chú ý là 2 chế độ 320×200 pixel 256 màu và 640×480 pixel 2 màu.

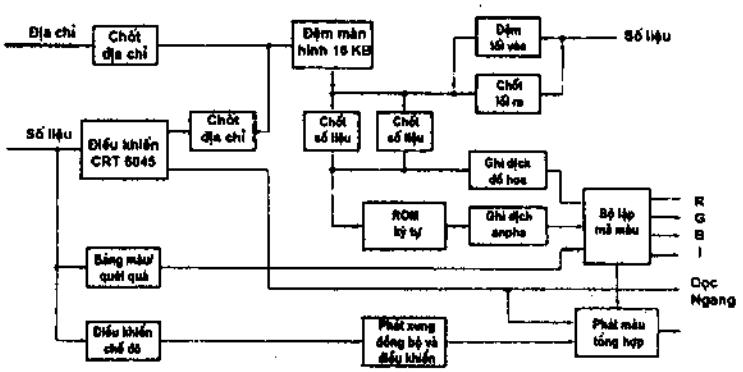
Trong các máy PS/2 model 50,60,70 và 80 các thiết bị VGA (Video Graphics Array) có thể tạo ra một số lượng lớn các chế độ hiển thị. Ngoài các mode tương thích EGA, màn hình VGA còn có các chế độ 640×480 16 màu từ palette 256 k, 320×200 với 256 màu từ palette 256 k và chế độ văn bản 80×25 sử dụng ma trận 8×16 điểm ảnh cho các kí tự.

Để tạo thành 256 màu, các mạch MCGA và VGA phải phát các tín hiệu màu tương tự thay cho các tín hiệu màu số trong các mạch ghép nối EGA trước đó. Như vậy các màn hình MCGA và VGA phải có khả năng tiếp nhận các tín hiệu màu tương tự. Hiện nay bản mạch ghép nối VGA và các mạch ghép nối đồ hoạ có độ phân giải cao khác như SVGA có thể hiển thị các mode đồ hoạ lên đến 1024×768 pixel với 256 màu trên một bảng màu 256 k.

11.3.2. Mạch ghép nối đồ hoạ trong máy tính cá nhân IBM

Hiện nay, các máy PC chủ yếu dùng các bản mạch VGA hoặc SVGA với các vi mạch tích hợp cỡ lớn được lắp trên đó. Nhưng để hiểu rõ chúng, ta sẽ xem xét lần lượt nguyên tắc hoạt động của các bản mạch ghép nối màn hình ra đời trước đó để làm căn cứ tìm hiểu, so sánh. Hình 11.11 trình bày sơ đồ khối của một bản mạch CGA. Bản mạch này vẫn sử dụng vi mạch MC6845 của hãng Motorola để thực hiện tất cả các nhiệm vụ điều khiển màn hình. Như đã đề cập phía trên, vi mạch 6845 tuần tự phát các địa chỉ yêu cầu đến RAM màn hình cũng như phát các xung đồng bộ dòng và đồng bộ mành.

Mạch ghép nối có thể chuyển tới màn hình các tín hiệu màu riêng biệt hoặc *tín hiệu video màu tổng hợp* (composite video color monitor) chứa mọi tín hiệu thành phần và chỉ truyền trên 1 đường dây. 16 kbyte RAM màn hình là loại *lưỡng cổng* (dual-ported), do vậy nó có thể được truy cập bởi bộ vi xử lí hệ thống hoặc vi mạch điều khiển CRT.



Hình 11.11 Sơ đồ bản mạch CGA.

Bản mạch ghép nối này có thể hoạt động trong chế độ văn bản hoặc chế độ đồ hoạ. Trong chế độ văn bản, nó sử dụng một ROM phát kí tự và các thanh ghi dịch (alpha serializer) để truyền nối tiếp thông tin về các điểm ảnh tới các lối ra RGB. Trong chế độ này, mỗi kí tự được biểu diễn bằng 2 byte trong RAM màn hình theo định dạng trên hình 11.11. Byte được đánh địa chỉ chẵn hay byte thấp chứa mã ASCII 8 bit của kí tự được hiển thị. Byte có địa chỉ lẻ hay byte cao chứa mã thuộc tính của kí tự. Bốn bit thấp của byte thuộc tính sử dụng các mã để xác định màu của kí tự. Các bit từ 4-7 của byte trạng thái cho phép xác định màu nền trong số 8 màu đầu tiên. Bit B của byte thuộc tính cho phép hiển thị kí tự nhấp nháy. Vì chỉ cần 4 kbyte để lưu trữ mã và thuộc tính của các kí tự trên màn hình 80×25 nên ta có thể lưu trữ tới 4 trang màn hình trong 16 kbyte RAM màn hình.

Có thể hiển thị một kí tự lên một vị trí bất kì trên màn hình CGA bằng cách ghi thông tin trực tiếp vào RAM màn hình. Bộ đệm khung trong một hệ CGA bắt đầu từ địa chỉ tuyệt đối B800h và cần 2 byte để lưu giữ mã và thuộc tính của mỗi kí tự. Chỉ cần cộng một bội số của 2 với B800h để có được địa chỉ của kí tự tại một vị trí bất kì trên màn hình. Sau đó có thể dùng lệnh MOV để ghi mã ASCII của kí tự vào địa chỉ ấy và ghi byte thuộc tính vào địa chỉ tiếp theo. Khi hoạt động trong chế độ đồ hoạ, bản mạch ghép nối CGA sử dụng các bộ ghi dịch để tạo ra thông tin về các điểm ảnh cho các súng tạo màu và cường độ tổng cộng. Dữ liệu của các pixel được truyền trực tiếp tới RAM màn hình.

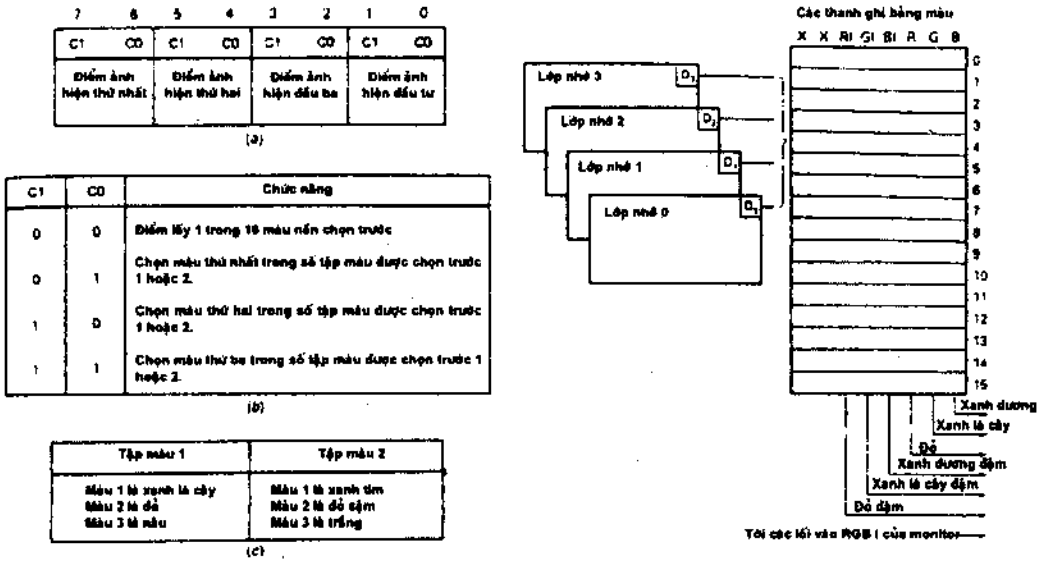
Để hiển thị đồ hoạ, một mạch ghép nối đồ hoạ CGA có thể hoạt động theo 3 chế độ: độ phân giải thấp, độ phân giải trung bình và độ phân giải cao. Chế độ phân giải thấp không thu hút nhiều sự quan tâm bởi nó chỉ hiển thị được 100 dòng, mỗi dòng 160 pixel. Chế độ phân giải

cao hiển thị được 200 dòng, mỗi dòng 640 pixel nhưng nó chỉ có khả năng tạo ra màn hình đồ hoạ đơn sắc.

Trong chế độ phân giải trung bình, màn hình bao gồm 200 dòng, mỗi dòng 320 pixel, tổng cộng là 64 000 pixel. 16 kbyte RAM màn hình tương ứng với 16 kbit × 8 hay 128 kbit. Thực hiện phép chia số lượng bit của bộ nhớ cho tổng số pixel, chỉ có 2 bit dùng để biểu diễn màu sắc cho mỗi pixel trong mode hoạt động này. Với 2 bit, chỉ có thể lựa chọn cho mỗi pixel một trong 4 màu.

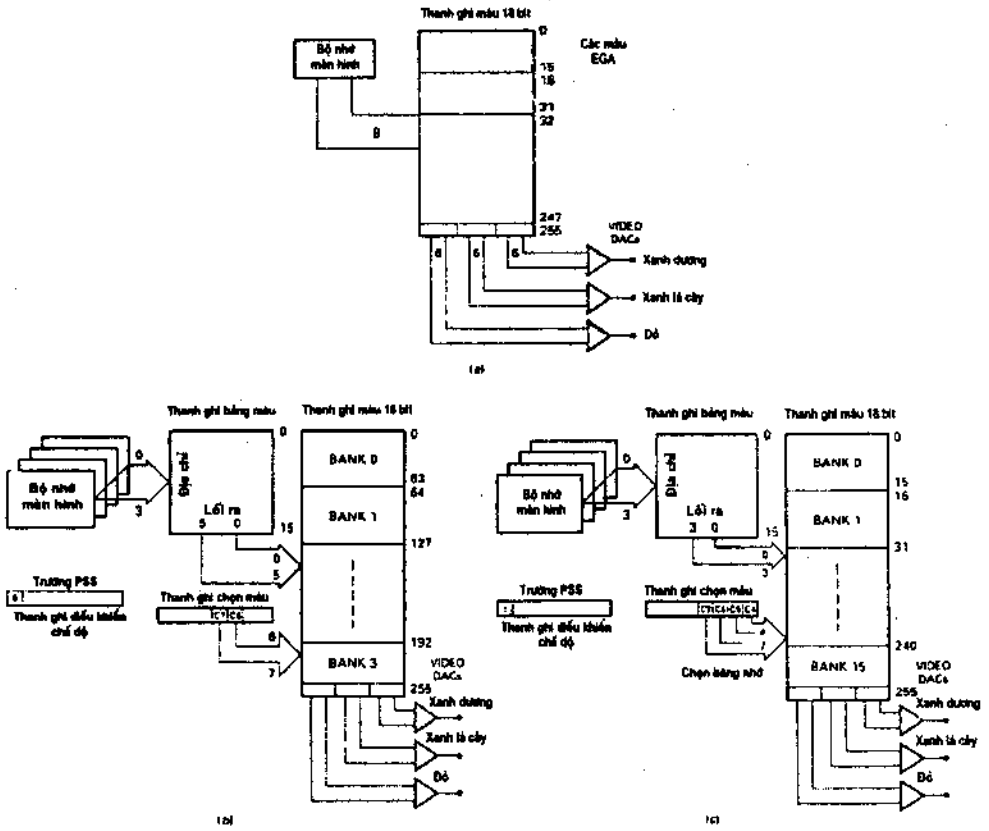
Hình 11.12a trình bày cách sắp xếp 2 bit màu cho mỗi pixel trong các byte nhớ của RAM màn hình. Hình 11.12b trình bày các mã được dùng để xác định màu sắc cho mỗi pixel. Hai bit này sẽ xác định pixel đó sẽ có màu nền hay một màu nào đó trong số hai tập màu. Hình 11.12c biểu diễn hai tập màu có sẵn. Ta có thể lựa chọn tập màu mong muốn bằng cách xuất một byte điều khiển qua cổng 3D9h tới mạch điều khiển bảng màu (pallette circuit) trên cạnh trái hình 11.12. Cách dễ nhất để thực hiện điều này là sử dụng ngắt 10h của BIOS.

Cũng có thể hiển thị các điểm ảnh trực tiếp lên màn hình CGA bằng cách ghi vào một vị trí nhớ thích hợp một byte cho 4 pixel liên tiếp trên cùng một hàng. Mạch ghép nối CGA sử dụng phương pháp quét dòng xen kẽ nên mã của những pixel trên các dòng chẵn được ghi vào bộ nhớ từ địa chỉ B800h còn các dòng lẻ được ghi từ địa chỉ BA00h.



Hình 11.12 Định dạng 320 x 200. (a) vị trí của các bit điểm ảnh trong byte nhớ. (b) Các mã được dùng để phân biệt màu chọn cho điểm ảnh. (c) Các tập 2 màu.

Có thể tính toán từ những địa chỉ khởi đầu này để tìm ra ô nhớ tương ứng với vị trí của pixel đã cho trên màn hình. Ngắt 10h của BIOS được dùng để hiển thị các điểm ảnh lên màn hình. Chuẩn EGA có một bộ điều khiển CRT khả trình cho phép đặt nhiều chế độ hiển thị như với vi mạch MC6845 của mạch ghép nối CGA. Tuy nhiên, các thành phần chính của một card EGA mà ta cần nói đến là bộ đệm khung và các thanh ghi bảng màu. Hình 11.13 trình bày cách chúng thực hiện các chức năng trong format thông dụng nhất, hiển thị 640×350 pixel với 16 màu từ một palette 64 màu.



Hình 11.13 Cấu hình phần cứng cho mode hiện VGA thông dụng.

Trong chế độ này, bộ đệm khung được chia thành 4 plane. mỗi plane lưu giữ một trong số 4 bit màu của mỗi pixel. Mỗi giá trị 4 bit được đọc ra từ 4 plane này được dùng để trỏ tới địa chỉ của 1 trong 16 thanh ghi palette 8 bit. Sáu bit thấp của thanh ghi được trỏ tới sẽ được đưa tới màn hình màu. Chú ý: để hoạt động trong chế độ này, màn hình phải có các lối vào đỏ, xanh lá cây và xanh lam đậm bên cạnh các lối vào đỏ, xanh lá cây và xanh lam chuẩn. Có tất cả 64 khả năng tổ hợp 6 bit trong mỗi thanh ghi palette, nhưng vì chỉ có 16 thanh ghi palette nên tại một thời điểm chỉ có 16 trong số 64 khả năng ấy là được lưu giữ. Mười sáu giá trị trong các thanh ghi palette tại một thời điểm cụ thể sẽ xác định 16 màu từ một palette 64 màu. Trong quá trình khởi động, các thanh ghi palette của một hệ EGA được khởi tạo bằng các giá trị tương ứng với 16 màu trong một hệ CGA, nhưng sau đó có thể nạp vào các thanh ghi palette những giá trị ứng với các màu sắc tuỳ ý.

Chuẩn VGA (video gate array - video graphic adapter) với bộ điều khiển CRT cho phép lựa chọn tần số nhịp đồng hồ, số dòng quét ngang, tốc độ làm tươi, lượng quét quá, v.v... Có thể lập trình cho một hệ VGA hoạt động theo các chế độ CGA, EGA hoặc một số chế độ khác. Một trong những chế độ chuẩn của VGA cho phép hiển thị 256 màu cùng một lúc từ một palette 262 144 màu. Để tạo ra 256 màu, một hệ VGA phải phát ra các tín hiệu đỏ, xanh lá cây và xanh lam tương tự thay cho các tín hiệu màu số trong các mạch ghép nối CGA trước đó. Như trình bày trên hình 11.9, một bộ biến đổi D/A 6 bit, thường được gọi là *video DAC*, được dùng để ra từng tín hiệu màu thành phần. Với một bộ biến đổi D/A 6 bit, tín hiệu phát ra có khả năng nhận 2^6 hay 64 giá trị nên tổng các tổ hợp của 3 tín hiệu sẽ là 64×64×64 hay 262 144, tức 256 K . Cũng trên hình 11.12, giá trị 18 bit ứng với màu được hiển thị được lưu trong 256 thanh ghi màu

(color register). Vì chỉ có 256 thanh ghi màu nên số lượng màu lớn nhất có thể hiển thị được tại một thời điểm là 256. Trên thị trường, một số hãng sản xuất đã tung ra các vi mạch tích hợp (IC) có tên là *RAMDAC* trong đó có chứa cả các thanh ghi màu lẫn các bộ biến đổi D/A .

Mỗi pixel sử dụng một giá trị 8 bit để chọn thanh ghi màu điều khiển các bộ biến đổi D/A. Giá trị 8 bit này được tạo ra theo nhiều cách khác nhau, tuỳ thuộc vào chế độ hiển thị được chọn.

Bên trái hình 11.13 trình bày cách thức mà các "địa chỉ" 8 bit trỏ tới thanh ghi màu được tạo ra trong chế độ 320×200×256 màu. Các giá trị 8 bit của 4 pixel liên tiếp được lưu trữ trong 4 lớp nhớ như hình vẽ chỉ ra. Khi một giá trị 8 bit được đọc từ một trong những lớp nhớ, 4 bit cao được dùng trực tiếp làm một phần của địa chỉ thanh ghi màu. 4 bit thấp được dùng để địa chỉ 1 trong 16 thanh ghi bảng màu. 4 bit thấp từ địa chỉ thanh ghi bảng màu địa chỉ được dùng làm 4 bit thấp của địa chỉ thanh ghi màu.

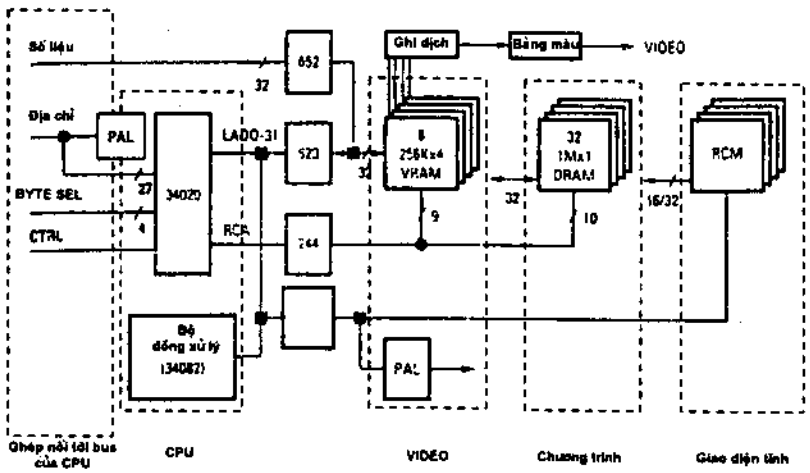
Trên hình chỉ ra một cách mà phần cứng VGA được đặt cho chế độ hiển thị 640×480 điểm ảnh với 16 màu. Các giá trị điểm ảnh 4 bit được lưu giữ trong 4 lớp nhớ. Đối với bộ điều khiển CRT, mỗi lớp bit này có cùng một địa chỉ, do đó bộ điều khiển sẽ đọc ra tất cả các bit cho một điểm ảnh cùng một lúc. 4 bít được đọc từ 4 lớp nhớ được dùng để định vị một trong 16 thanh ghi bảng màu. 6 bit thấp từ thanh ghi bảng màu này được dùng làm 6 trong số 8 bit địa chỉ của thanh ghi màu. 6 bit từ thanh ghi bảng màu có thể địa chỉ được 1 trong số 64 thanh ghi màu. Tuy nhiên vì chỉ có 16 thanh ghi bảng màu do đó chỉ có 16 trong số 64 thanh ghi màu được định vị một lần. Điều này có nghĩa rằng màn hình chỉ có thể hiện đồng thời 16 màu một lần.

Hai bit cao của địa chỉ thanh ghi màu nhận được từ các bit C6 và C7 từ một thanh ghi đặc biệt được gọi là thanh ghi chọn màu. Các bit này có thể được điều khiển bởi người sử dụng cho phép thay đổi nhanh chóng từ một tập 16 màu này tới tập 16 màu khác trong bảng 256 màu. Hình 11.13c chỉ ra một cách khác mà phần cứng VGA cấu hình cho hiển thị 640x480 điểm ảnh với 16 màu. Trong cấu hình này, 4 bit từ các lớp nhớ chọn một trong 16 thanh ghi bảng màu nhưng chỉ có 4 bit trong số 6 bit của thanh ghi bảng màu được sử dụng để định vị thanh ghi màu. 4 bit khác của thanh ghi màu nhận được từ thanh ghi chọn màu. Có thể tưởng tượng 4 bit này như việc chọn 1 trong 16 dải màu trong 256 thanh ghi màu. Ưu điểm của các phương pháp này là ở chỗ cho phép chuyển nhanh chóng giữa các tập 16 màu với nhau.

11.4. ĐỒ HOẠ PHÂN GIẢI CAO VÀ BỘ XỬ LÍ ĐỒ HOẠ

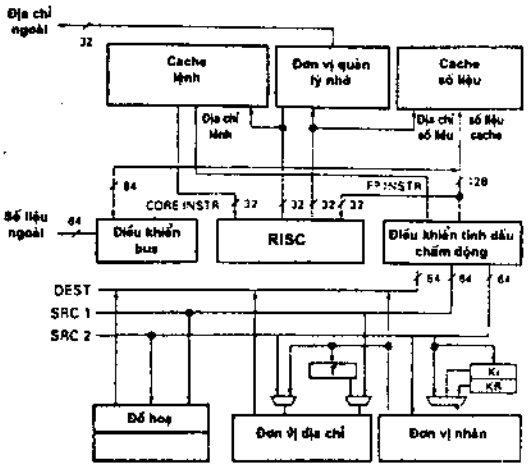
Trong nhiều ứng dụng như các phần mềm đồ hoạ như CAD (computer added design), CAE (computer-added enginnering) hoặc EDA (electronic design automation); độ phân giải cần phải có 1024x768 hoặc hơn nữa. Điều này liên quan tới quá trình làm tươi màn hình và làm tươi bộ nhớ RAM. Thêm nữa các chi tiết hình ảnh phức tạp sẽ đòi hỏi nhiều phép tính hơn trong quá trình vẽ các điểm ảnh. Trong những trường hợp này có thể dùng các bản mạch ghép nối đồ hoạ đặc biệt trong đó có các bộ xử lí đồ hoạ như TMS34020 của hãng Texas Instruments. Hình 11.14 chỉ ra sơ đồ nối bộ xử lí này với bộ vi xử lí chính để tạo ra các hình ảnh có độ phân giải 1024 x 1024 điểm ảnh 8 bit. Ở đây VRAM được dùng cho bộ đệm video vì nó có khả năng nhận số liệu ra đủ nhanh để làm tươi điểm ảnh. Các tín hiệu lối ra VRAM được nối tới các thanh ghi bảng màu và lối ra của các thanh ghi bảng màu được đưa vào các bộ biến đổi D/A để tạo ra các tín hiệu RGB analog cấp cho monitor. Các DRAM và ROM trong bản mạch được dùng để giữ các chương trình và dữ liệu cho bộ xử lí đồ hoạ. Những lệnh đồ hoạ cơ bản có sẵn trong chip như vẽ đường thẳng, chuyển điểm ảnh, tô màu một vùng v.v... được phát triển thành các thư

viện cho phép người sử dụng lập trình dễ dàng các chương trình thực hiện vẽ rất nhanh các hình phức tạp.



Hình 11.14 Nối TMS34020 với CPU.

Chip Intel i860 còn có ưu điểm hơn, sơ đồ khối của nó như hình 11.15. Nó thực sự như là một CPU trong một máy trạm có khả năng thực hiện một lượng lớn các phép tính dấu chấm động cho các xử lí đồ hoạ 3 chiều. Với nhịp đồng hồ 40 MHz nó có thể thực hiện được 80 triệu phép tính dấu chấm động trong một giây nhờ vào việc đưa vào nhiều chức năng và nhiều hoạt động trong đó được xử lí song song thay vì nối tiếp. Chip có một bộ xử lí trung tâm kiểu RISC, một bộ xử lí dấu chấm động và một bộ xử lý đồ hoạ. Nó cũng chứa 2 bộ nhớ cache lệnh và số liệu riêng biệt được bộ xử lí đọc trong cùng một thời gian. Bus dữ liệu ngoài rộng 64 bit cho phép cùng một lúc có thể đọc cả từ số liệu 32 bit và từ lệnh 32 bit.



Hình 11.15 Sơ đồ khối chip Intel i860.

11.5. MÀN HÌNH TINH THỂ LỎNG

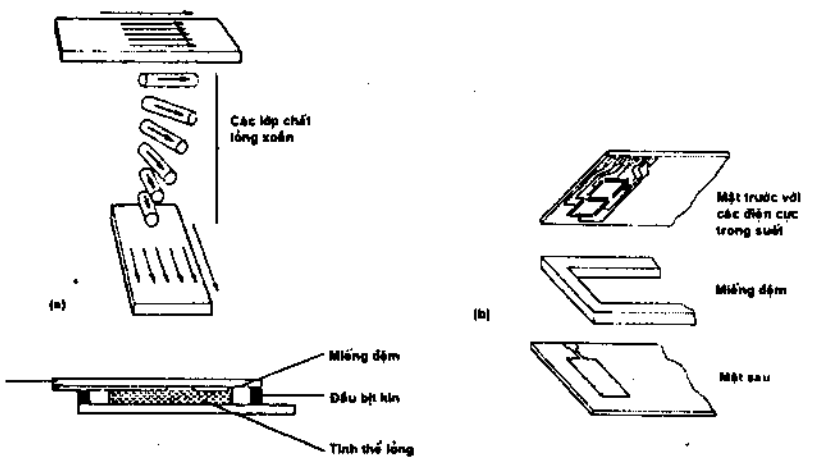
Ống tia điện tử CRT có nhược điểm là kích thước lớn và tiêu tốn năng lượng do vậy, trong các máy tính xách tay (notebook), màn hình tinh thể lỏng LCD (liquid crystal display) đã được dùng thay thế cho ống tia này.

Một số chất có những phân tử có hình thon dài cho phép định hình thành những thanh mảnh dưới tác dụng của trường tĩnh điện. Trong trạng thái được gọi là trạng thái tinh thể lỏng như vậy được tồn tại ở một nhiệt độ cao hơn nhiệt độ nóng chảy của tinh thể chất một chút. Lúc này chất có tính phân cực quang; tức là chỉ cho sóng ánh sáng phân cực (vectơ điện trường chỉ dao động trong một mặt phẳng) truyền qua khi mặt phẳng phân cực của ánh sáng trùng

với mặt phẳng phân cực của chất. Phương của mặt phẳng phân cực của tinh thể lỏng có thể điều khiển được bằng điện trường áp đặt lên nó. Nhóm chất được sử dụng nhiều nhất trong kĩ thuật hiện ảnh là các tinh thể lỏng hình xoắn mà mặt phẳng phân cực quay theo trường tĩnh điện áp đặt vào. Thí dụ, sẽ quay 90 độ khi điện trường thấp hơn 3 vôn.

Mỗi chi tiết cần hiện trên màn hình LCD được cấu tạo bởi việc ghép xen kẽ chi tiết làm bằng tinh thể lỏng giữa 2 lớp thuỷ tinh có phủ chất phân cực sáng có cửa sổ phân cực vuông góc nhau như hình 11.16a chỉ ra. Trường tĩnh điện được đặt tới mỗi chi tiết bằng các điện cực được phủ trên các lớp thuỷ tinh này. Vì rằng dòng một chiều DC sẽ làm hỏng tinh thể lỏng (do điện phân) nên màn hình sẽ được nuôi bằng dòng điện xoay chiều AC với tần số từ 25 đến 500Hz. Tuỳ thuộc vào tín hiệu đặt vào, mặt phẳng phân cực của tinh thể lỏng sẽ bị quay đi một góc và kết quả là ánh sáng chiếu tới bề mặt sẽ được phản xạ hoặc không để tạo nên hình ảnh. Loại màn hình dùng phương pháp phản xạ như vậy có ưu điểm là chỉ tiêu thụ một dòng rất nhỏ nhưng đáp ứng thời gian chậm hơn màn hình CRT (100ms) và tỉ số độ tương phản thấp.

Để tạo thành màn ảnh trong máy tính, các phần tử tinh thể lỏng được sắp xếp thành một ma trận các điểm ảnh (dots). Các phần tử trong mỗi hàng và cột được nối lần lượt với nhau. Một phần tử được kích hoạt chỉ khi cả 2 dây dòng và cột nối với nó được điều khiển. Số liệu cho một dòng các điểm ảnh của một kí tự hay toàn bộ một dòng qua màn hình được đưa tới trục X của ma trận và dòng này được kích hoạt. Sau một thời gian ngắn, dòng này không được kích hoạt nữa. Quá trình như vậy được quét từ đỉnh màn hình đến đáy rồi quay trở lại.



Hình 11.16 Màn hình tinh thể lỏng LCD.

Để tạo ra màn hình LCD màu người ta dùng các nguồn sáng huỳnh quang mạnh phía sau truyền qua các tinh thể lỏng và một số bộ lọc màu để tạo ra các hình ảnh có màu sắc mong muốn. Việc lập trình cho bản mạch ghép nối màn hình có thể thực hiện bằng việc thâm nhập trực tiếp tới các thanh ghi điều khiển CRT, các thanh ghi bảng màu, bộ đệm khung v.v... Cũng có thể thâm nhập qua ngắt 10h trong BIOS nhưng sẽ đạt được tốc độ chậm hơn. Và tất nhiên còn có thể sử dụng các hàm đồ hoạ trong các ngôn ngữ bậc cao nữa. Thí dụ, dùng ngắt 10h để khởi động adapter tới mode màu văn bản với 80 cột x 25 dòng:

```
MOV  Ah,00        ;     gọi hàm 00h
MOV  AL,03        ;     mode text 80 x 25
INT  10h          ;     gọi ngắt
```

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

11.1 Giải thích nguyên tắc tạo mành xen kẽ và không xen kẽ.

11.2 Dùng một hình vẽ đơn giản để mô tả cách hiện một chữ X trong ống CRT bằng việc quét không xen kẽ.

11.3 Căn cứ vào hình 11.3 trả lời các câu hỏi sau:
- Chức năng của RAM trong mạch?
- Tại điểm nào trong quá trình hiện một khung làm cho các lối vào địa chỉ của RAM này thay đổi? Làm cho các lối vào địa chỉ của ROM phát kí tự thay đổi ?
- Chức năng của thanh ghi dịch tại lối ra của ROM kí tự ?
- Xung đồng bộ ngang xuất hiện lúc nào trong quá trình hiện khung ?
- Xung đồng bộ dọc xuất hiện lúc nào trong quá trình hiện khung ?

11.4 Một ống hình CRT được thiết kế để hiện 24 dòng với 80 kí tự trong một dòng. Hệ sử dụng máy phát kí tự 7 x 9 trong ma trận 9 x 12 điểm ảnh. Giả thiết tốc độ khung quét không xen kẽ là 60 Hz, 3 kí tự dự trữ cho việc quét quá theo chiều ngang, 120 cho chiều dọc. Tìm các giá trị sau:
- Tổng số thời gian cho các kí tự trên một dòng ?
- Tổng số dòng quét trên một khung ?
- Tần số quét ngang (số dòng quét trong 1 giây) ?
- Tần số xuất hiện điểm ảnh (số điểm ảnh trong 1 giây) ?
- Dải thông cực tiểu cho bộ khuếch đại thị tần ?
- Thời gian giữa các lần thâm nhập RAM ?

11.5 Viết chương trình dùng ngắt BIOS để đọc một xâu kí tự từ bàn phím, đặt mã kí tự vào bộ đệm nhớ và hiện các kí tự lên màn hình khi một phím được nhấn.

11.6 Tại sao nhiều máy vi tính hiện nay dùng một bộ xử lí đồ hoạ chuyên dụng thay cho bộ xử lí chính khi tính toán các giá trị pixel cho các hình ảnh ?

11.7 Hãy dùng sơ đồ để giải thích sự khác nhau giữa việc lưu trữ các điểm ảnh kiểu gói và lưu trữ kiểu mặt ?

11.8 Dung lượng nhớ cần là bao nhiêu để lưu giữ số liệu cho một màn hình đơn sắc có độ phân giải 640 x 480 ?

11.9 Mô tả cách tạo ra hình ảnh màu bằng 3 chùm tia điện tử trong ống hình CRT màu.

11.10 Bao nhiêu bit nhớ cần để lưu giữ số liệu cho 1 điểm ảnh mà điểm ảnh đó có thể nhận một trong 256 màu ?

11.11 Viết một chương trình hợp ngữ dùng ngắt 10h của BIOS để khởi động một card VGA cho mode 320 × 200 × 256 màu.

11.12 Ưu điểm chính của màn hình LCD so với màn hình CRT là gì ?

CHƯƠNG 12. GHÉP NỐI MẠNG MÁY TÍNH, CHUẨN GHÉP NỐI GPIB - IEE488

12.1. TRUYỀN THÔNG TIN NỐI TIẾP ĐỒNG BỘ

Trong chương 8 ta đã khảo sát quá trình truyền thông tin giữa hai máy tính theo phương thức truyền nối tiếp không đồng bộ qua các cổng COM và Modem trên các đường dây điện thoại công cộng. Hiện nay khi các tín hiệu tương tự được truyền đến tổng đài, chúng được số hoá bằng các bộ biến đổi A/D theo phương pháp điều chế xung mã PCM là các bộ mã hoá *coder* và ngược lại, tín hiệu số từ nơi khác tới được giải mã bằng một bộ biến đổi D/A (pulse-code modulation) gọi là *decoder*. Vì rằng cả hai bộ luôn cần cho thông tin hai chiều nên chúng thường được các công ty sản xuất đóng gói trong cùng một chip và được gọi chung là bộ *codec*. Các bộ biến đổi A/D này thường không tuyến tính. Chúng được chế tạo sao cho có các bước lượng tử hoá nhỏ với các mức tín hiệu nhỏ và bước lượng tử hoá lớn cho các mức tín hiệu lớn. Tính phi tuyến này được gọi là sự nén tín hiệu. Điều đó cải thiện độ chính xác của các tín hiệu có mức nhỏ mà không cần tăng độ phân giải của bộ biến đổi. Tất nhiên bộ biến đổi D/A ở phía kia lại phải có tính phi tuyến ngược lại nhằm cho dạng sóng toàn bộ không bị méo. Lối ra A/D của codec thường được dồn kênh với các lối ra khác theo cách thức thường gọi là dồn kênh theo thời gian TDM (time-division multiplexing) nhằm cho phép dùng chung một đường truyền cho nhiều kênh thông tin. Ta lấy một thí dụ, dồn 24 kênh thoại PCM vào một đường dây truyền. Giả sử mỗi kênh có độ phân giải 8 bit và các kênh được gửi trên đường truyền xen kẽ liên tiếp nhau với tốc độ 8 kHz. Nếu thêm vào 1 bit cho khung truyền thì như vậy bộ dồn kênh sẽ truyền đi các khung gồm 193 bit với tốc độ 8000 bit/giây và tốc độ truyền dữ liệu sẽ là 193 × 8000 = 1,544 Mbit/giây. Nhưng nếu cần đến 4032 kênh truyền như vậy thì tốc độ sẽ lên tới 274,176 Mbit/giây và phải mở rộng dải thông của đường truyền bằng các môi trường truyền khác như cáp quang sợi nếu không sẽ gặp phải hiện tượng nghẹt cổ chai. Một trong những phương pháp để giải quyết vấn đề này là sử dụng hệ thống gọi là mạng số đa dịch vụ ISDN (integrated services digital network) trong đó thay thế các đường truyền analog giữa các máy điện thoại và tổng đài bằng các đường truyền số có tốc độ cao hơn.

Truyền dữ liệu kiểu không đồng bộ có nhược điểm là lượng thông tin lãng phí nhiều do phải gán các bit không phải là dữ liệu (bit start, stop, chẵn lẻ) cho mỗi kí tự cần truyền. Thí dụ lượng lãng phí đến 30% trong tổng số: 7 bit dữ liệu + 1bit start + 1 bit stop + 1 bit chẵn lẻ. Phương pháp truyền có hiệu suất cao hơn là truyền đồng bộ trong đó gửi và nhận một khối lớn kí tự không ngừng, kí tự này sau kí tự kia không có các bit start và stop. Khi ngừng truyền đường dây sẽ được giữ ở mức quy định (thí dụ mức cao - mark). Để chỉ thị khởi phát quá trình truyền, máy phát gửi đi một hoặc một số các kí tự duy nhất được gọi là các kí tự đồng bộ hoặc một bit gọi là bit cờ. Máy thu sử dụng các kí tự đồng bộ hoặc bit cờ để đồng bộ đồng hồ bên trong của nó với các tín hiệu thu được. Nó sẽ dịch các số liệu và biến đổi chúng thành dạng thông tin song song (thí dụ 8 bit một) để có thể đọc vào máy tính. Các Modem và mạng ISDN có tốc độ cao dùng loại thông tin nối tiếp này. Rõ ràng giữa máy phát và thu trong trường hợp này phải có một sự hợp tác cao để đảm bảo truyền dữ liệu một cách chính xác và có trật tự. Nói cách khác hoạt động của chúng phải tuân thủ theo một giao thức (protocol) nào đó. Tức là tuân thủ theo một tập hợp các quy tắc liên quan đến dạng tín hiệu được truyền. Hai giao thức thông

dụng là BISYNC (binary synchronous communications protocol) và HDLC (high-level data link control protocol).

12.2. GHÉP NỐI MẠNG CỤC BỘ LAN

12.2.1. Hình trạng của mạng

Các máy tính có thể ghép nối với nhau thành một mạng cục bộ LAN (Local Area Network) để cùng nhau thông tin và dùng chung các tài nguyên (như máy in, ổ đĩa cứng v.v..) của nhau. Để thông tin trong khoảng cách ngắn giữa một thiết bị đầu cuối và một máy tính thì chuẩn RS-232C là đủ. Nếu ở khoảng cách xa thì cần phải có Modem và đường dây điện thoại hoặc đường dây riêng. Nhưng nếu có nhiều máy tính cần được kết nối với nhau trên một diện tích nào đó thì một trong những cách giải quyết là nối các dây từ một chuyển mạch dữ liệu trung tâm tới tất cả các máy tính rồi truyền dữ liệu trên đường dây có dải thông rộng. Lúc đó hoặc là phải dồn các kênh thông tin (mỗi kênh cho một máy tính) theo phương pháp dồn kênh thời gian hoặc dồn kênh tần số. Có 5 cách nối các thiết bị trong mạng LAN (gọi là các hình trạng của mạng - nework topology):

- *Nối star:* trong đó tất cả các thiết bị trong mạng được nối tới một bộ điều khiển trung tâm.
- *Nối loop*: một thiết bị được chọn là bộ điều khiển. Nếu thiết bị nào đó muốn thông tin với một hoặc một vài thiết bị khác nó sẽ gửi yêu cầu tới bộ điều khiển. Nếu đường truyền rỗi, bộ điều khiển sẽ cho phép một thiết bị xuất ra tín hiệu và các thiết bị khác nhận tín hiệu.
- *Nối bus*: việc điều khiển bus được dàn trải khắp trên tất cả các thiết bị được nối với bus. Hệ chỉ gồm một dây bus thông tin (không cần phải là loại đồng trục) trong đó một số thiết bị bất kỳ đều nối tới dây đó. Bất kì thiết bị nào đều có thể chiếm quyền truyền dữ liệu trên bus. Dữ liệu được truyền thành từng khối có độ dài cố định gọi là gói dữ liệu. Để ngăn chặn hai hay nhiều thiết bị cùng phát một thời gian phải thực hiện các giao thức đặc biệt như CSMA/CD (carrier sense multiple access with collision detection).
- *Nối ring*: việc điều khiển cũng được phân tán tới tất cả các thiết bị trong mạng. Mỗi thiết bị có chức năng như một bộ lặp lại (repeater) tức là nó đơn giản chỉ nắm rồi nhả dữ liệu đến thiết bị tiếp theo trên mạng vòng nếu không có ý định nhận dữ liệu. Dữ liệu luôn chạy vòng trên mạng theo một chiều. Thiết bị bất kì có thể phát dữ liệu trên mạng. Một thẻ bài được dùng để tránh hai hay nhiều thiết bị cùng phát dữ liệu trong một lúc. Đó là một byte đặc biệt (thí dụ 01111111) chạy dọc theo vòng mạng khi không có thiết bị nào phát dữ liệu. Khi một thiết bị cần phát, nó lấy đi thẻ bài trên mạng để ngăn chặn không cho thiết bị khác phát dữ liệu. Sau khi phát một hoặc nhiều gói dữ liệu, thiết bị này đặt lại thẻ bài trở về vòng mạng để các thiết bị khác có thể tóm lấy nó mà giành quyền phát.
- *Nối tree*: được dùng trong việc truyền dữ liệu trên dải băng rộng. Việc truyền trên dải băng rộng dựa trên các sơ đồ dồn kênh tần số như truyền hình cáp CATV (communication antenna television) trong đó phổ tần số radio được chia thành các kênh rộng 6 MHz. Một hoặc một nhóm thiết bị được gán cho một kênh phát và nhóm khác cho kênh thu. Mỗi kênh hoặc một phần của kênh được coi như một nhánh cây. Các Modem đặc biệt được dùng để biến đổi các tín hiệu số thành các tín hiệu tần số radio được điều chế. Các kênh và độ rộng 6MHz trong mạng băng rộng cho phép tiếng nói, số liệu và tín hiệu video được phát đồng thời qua mạng. Đây là một ưu điểm so với hệ băng gốc trong đó chỉ có thể phát một loại tín hiệu số một lần.

12.2.2. Các kết nối vật lí của mạng

Các thiết bị trong mạng LAN được kết nối vật lí với nhau trên cơ sở các thiết bị và đường truyền.

1. Đường truyền: mạng cục bộ thường dùng 3 loại cáp truyền:

- *Cáp xoắn đôi*: giữa hai dây điện được xoắn với nhau nhằm tránh các can nhiễu điện từ, can nhiễu 50Hz. Cáp này lại gồm 2 loại: có bọc kim STP (shield twisted pair) và không bọc kim UTP (unshield twisted pair). Tốc độ cho UTP từ <4MB/s đến 100MB/s.
- *Cáp đồng trục* (coaxial cable): gồm một dây đồng ở giữa chất cách điện, xung quanh được quấn, bện bằng dây kim loại là vỏ bọc kim tránh can nhiễu điện. Có hai loại với các trở sóng 50 và 75 Ohm. Về hình dạng có 2 loại: cáp gày (thin cable) có đường kính 0,25 inch (thí dụ, RG-58) và cáp béo (thick cable) có đường kính 0,5 inch (thí dụ, loại RG-8, RG-11). Các cáp này được dùng trong dải băng gốc (base band) có dải thông từ 2,5 MB/s tới 10 MB/s. Chúng có độ suy hao nhỏ hơn cáp xoắn và có thể sử dụng xa tới vài ngàn mét.
- *Cáp quang*: các tín hiệu điện được chuyển thành ánh sáng sau khi cho qua bộ chuyển đổi điện-quang E-O (electric to optical) và được truyền trên các sợi quang (optical fiber) trong cáp quang rồi được chuyển đổi lại thành điện qua bộ chuyển đổi quang-điện O-E (optical to electric). Có hai loại cáp quang: đơn mode và đa mode. Loại đa mode có đường kính lớn, giá thành thấp nhưng do sai pha giữa các tia sáng lối ra nên sẽ gây nên nhưng sai lệch trong tốc độ truyền dữ liệu. Loại đơn mode có đường kính chỉ bằng một vài lần bước sóng ánh sáng cho phép chỉ cho một tia sáng rất gần song song với trục quang sợi được truyền nên không mắc phải khiếm khuyết trên. Giá thành loại này đắt và phải được chế tạo bởi nguyên liệu đặc biệt nếu không khoảng cách truyền sẽ không được xa. Loại này thường cho một tốc độ truyền tới 1 Gbit/s với khoảng cách 60 km.

2. Các thiết bị mạng

Có thể liệt kê vắn tắt các loại thiết bị mạng như sau:

- *Bản mạch ghép nối mạng NIC* (network interface card): được cắm vào các khe cắm mở rộng của các máy vi tính trên mạng. Nó gồm có : Bộ thu phát (tranceiver), nhằm phát tín hiệu của PC lên mạng và thu tín hiệu từ mạng vào PC; các đầu nối như K-45 cho cáp UTP, BNC cho cáp gầy, AUT cho cáp béo, DB-15 cho STP và RJ-45 cho UTP.
- *Bộ tập trung Hub* (concentrator): gồm các hub bị động (pasive hub), hub tích cực (active hub) có thể khuếch đại và xử lý tín hiệu, hub thông minh (intelligent hub) như hub tích cực nhưng có thêm các chức năng quản trị hub, chuyển mạch v.v...
- *Bộ chuyển tiếp* (repeater): cho phép tiếp nhận và chuyển tiếp các tín hiệu dữ liệu dùng để mở rộng mạng.
- *Cầu* (Brigde): giống repeater nhưng hoạt động mềm dẻo hơn vì có chọn lọc tín hiệu chỉ cần cho trạm đích bên kia.
- *Bộ định tuyến* (Router): thông minh hơn bridge vì có thể thực hiện các thuật giải chọn đường tối ưu. Nó cho phép nối các mạng khác nhau thành các liên mạng, tức là router phải hiểu được các giao thức khác nhau.

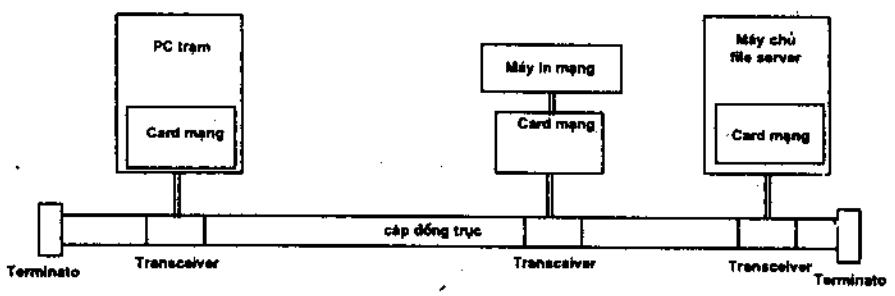
12.2.3. Giao thức mạng

Ở mức thấp nhất, những máy tính trong mạng truyền thông tin với nhau bằng cách gửi các gói thông điệp gọi là các khung truyền. Các card mạng cùng các phần mềm hỗ trợ của chúng gửi và nhận những khung này. Mỗi máy tính trên mạng LAN được nhận dạng bằng một địa chỉ duy nhất để những khung này có thể gửi tới. Hiện nay người ta phân tầng cho các giao

thức truyền thông mạng. Kiến trúc phân tầng là một cách thức cho phép hiểu và thực hiện rạch ròi các nhiệm vụ trong quá trình truyền thông trên mạng. Thí dụ, tầng thấp nhất chỉ quan tâm đến cấu trúc vật lí của đường truyền, nó cho phép một card mạng biết và nhận thông điệp nhưng không cần chú ý tới những máy chủ và các file truyền tập tin ở tầng cao hơn; ngược lại tầng cao hơn hiểu những máy chủ quản lí tập tin nhưng không cần biết về các kết nối vật lí của mạng v.v... Như vậy lí do chủ yếu tách chức năng mạng thành các tầng là cho phép các nhà sản xuất phần cứng và phần mềm khác nhau chế tạo những sản phẩm sử dụng trên mạng (những ứng dụng hệ điều hành, những card mạng, cáp...) đảm bảo chắc chắn các khả năng hoạt động của chúng trong hệ mạng. Mặc dù có những trường hợp riêng hơi khác nhau nhưng đến nay hầu hết đều tán thành tiêu chuẩn phân tầng OSI. Tổ chức quốc tế về tiêu chuẩn hoá chia những truyền thông trong mạng LAN thành 7 tầng. Mỗi tầng độc lập với các tầng khác bằng một giao diện xác định chính xác. các tầng được mô tả tóm tắt như sau:

- *Tầng 1, vật lí* (physical): xác định các tính chất vật lí và điện của các kết nối mạng (cáp, đầu nối, card, hub ...). Tầng này được coi là phần cứng.
- *Tầng 2, liên kết dữ liệu* (data link): xử lí những xung dữ liệu, sửa lỗi v.v... Tầng này thường được chia thành các tầng con: điều khiển truy cập môi trường MAC (media access control) và tầng điều khiển liên kết lôgic LLC (logical link control). Tầng MAC giải quyết truy cập mạng và điều khiển mạng. Tầng LLC quan tâm đến việc gửi và nhận các thông điệp của người dùng.
- *Tầng 3, mạng* (network): chuyển đổi và định truyền các gói tin cần thiết để đưa chúng tới đích. Nó lập địa chỉ và chuyển giao các gói thông điệp.
- *Tầng 4, giao vận* (transport): điều khiển sự nối tiếp trật tự những thành phần của thông điệp khi chúng được tách thành các gói tin nhỏ.
- *Tầng 5, phiên* (session): cho phép những ứng dụng chạy ở hai trạm làm việc phối hợp những truyền thông của chúng thành một phiên truyền. Tầng này hỗ trợ tạo ra phiên truyền ấy, quản lý những gói tin được gửi và nhận trong phiên truyền và kết thúc phiên truyền.
- *Tầng 6, biểu diễn* (presentation): chuyển đổi thông điệp thành các mã và định dạng mà máy thu có thể hiểu được.
- *Tầng 7, ứng dụng* (application): gồm các dịch vụ chung của mạng như thư điện tử, quản lí file, thông tin giữa các chương trình và chia sẻ các thiết bị ngoại vi v.v...

Ta sẽ khảo sát qua về *chuẩn Ethernet* điển hình cho một mạng LAN. Về mặt vật lí nó sử dụng tôpô-bus với cáp đồng trục 50 Ω. Số liệu được truyền ở dải băng gốc với tốc độ 10 Mbit/s. các bit số liệu được lập mã Manchester. Ưu điểm của mã này là ở mỗi ô bit đều có một chuyển mức tín hiệu. Một hệ thống muốn truyền tín hiệu trên mạng trước hết phải kiểm tra xem mạng có bận không. Hình 12.1 là sơ đồ đơn giản của một mạng Ethernet.



Hình 12.1 Sơ đồ mạng Ethernet đơn giản.

Đường xương sống (backbone) của mạng là một cáp đồng trục. Các terminator là các điện trở có giá trị bằng trở kháng sóng đường dây được đặt ở hai đầu cuối đường dây để ngăn phản xạ tín hiệu.

Một transceiver gửi ra tín hiệu trên cáp hoặc nhận tín hiệu từ cáp và phát hiện bất kì một ý định phát trong khi cáp đang được sử dụng. Transceiver được nối tới một bản mạch ghép nối với đầu nối 15 chân và 4 cặp dây xoắn. Cáp transceiver có thể dài tới 15 m. Card ghép nối mạng thực hiện hầu hết các công việc nhận dữ liệu một cách chính xác. Nó ghép nối và phân tách các khung dữ liệu, gửi ra các địa chỉ nguồn và đích, phát hiện các lỗi truyền phát và ngăn chặn việc phát trong khi có một trạm nào đó đã đang phát rồi. Phương pháp để xâm nhập mạng gọi là phương pháp đa truy nhập sử dụng sóng mang có phát hiện xung đột CSMA/CD (carrier sense multiple access with collision detection). Xuất phát điểm của nó là phương pháp CSMA hay còn gọi là phương pháp "nghe trước khi nói" LBT (listen before talk). Trạm sẽ chỉ phát dữ liệu khi biết đường truyền không bận. Nếu bận thì phải thực hiện một trong 3 giải thuật đợi: hoặc tạm rút lui chờ trong thời gian ngẫu nhiên rồi lại bắt đầu thăm dò (nghe ngóng) đường truyền xem đã rỗi chưa; hoặc tiếp tục nghe đến khi đường truyền rỗi với xác suất bằng 1; hoặc tiếp tục nghe đến khi đường truyền rỗi với xác suất lớn hơn 0 nhỏ hơn 1. Phương pháp này có nhược điểm là có thể xảy ra xung đột trong khi trạm đang phát vậy phải dùng CSMA/CD hay còn gọi là phương pháp "nghe trong khi nói" LWT (Listen while talk). Trong đó khi truyền trạm vẫn tiếp tục nghe ngóng đường truyền. Nếu phát hiện thấy có xung đột thì ngừng ngay việc truyền dữ liệu nhưng vẫn tiếp tục gửi sóng mang một khoảng thời gian nữa để báo cho các trạm khác. Chờ một khoảng thời gian nữa rồi thử truyền theo quy tắc CSMA với 3 giải thuật đợi kể trên.

Số đơn vị cực đại có thể nối vào một mạng Ethernet là 1024. Cáp đồng trục thường đắt nên các tiêu chuẩn mới cho phép dùng loại thin Ethernet hay 10Base-T là loại truyền tới 10Mbit/s với cáp xoắn đôi với các khoảng cách từ 100m đến 2500m. Khi lưu lượng truyền thông trong mạng tăng thì thời gian mà một đơn vị ở một đầu mạng phải đợi đến khi được truyền trở nên dài. Điều này làm giảm các chỉ tiêu thông thường của mạng. Lúc đó phải dùng các loại mạng khác có ưu thế hơn như mạng Token-ring v.v...

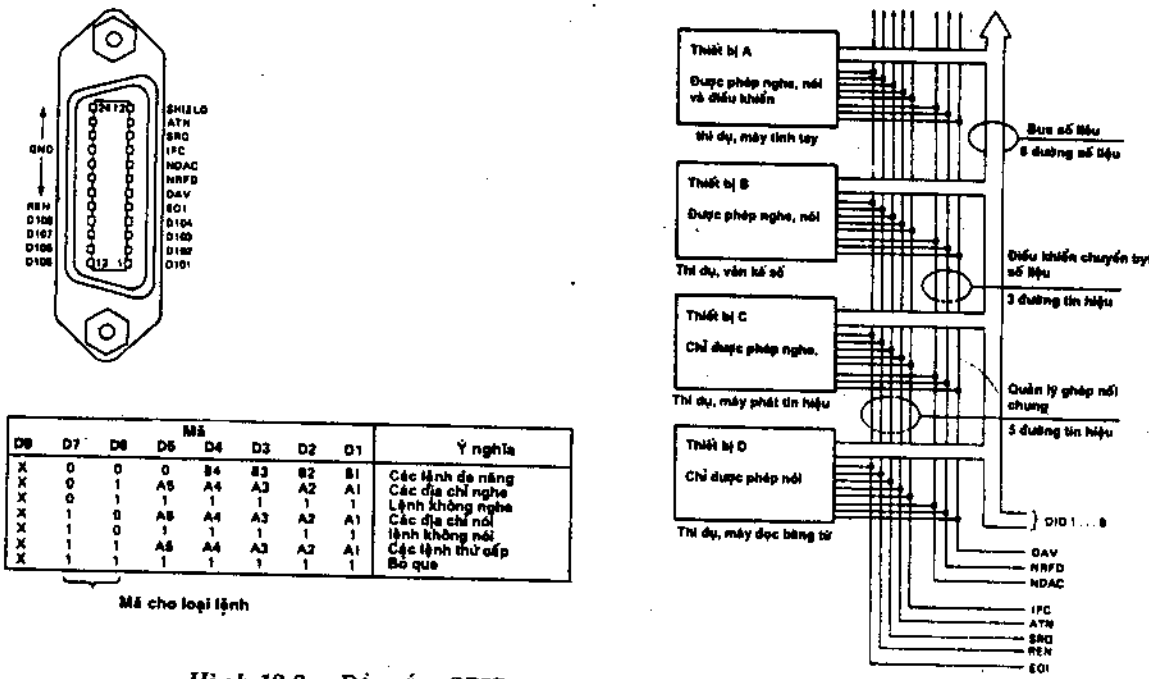
12.3. CHUẨN GHÉP NỐI GPIB VÀ BUS IEE488

Được phát triển ban đầu bởi hãng Hewlett-Packard và sau được IEEE (Institute of Electrical Electronical Engineering) nâng thành chuẩn IEEE-488; chuẩn này hiện đùng dùng rất phổ biến trong công nghiệp và các phòng thí nghiệm khi cần nối ghép các thiết bị đo lường, kiểm tra thông minh và máy vi tính. Chuẩn định nghĩa 3 loại thiết bị:

- Thiết bị nghe (listener) là thiết bị nhận số liệu từ thiết bị khác hoặc từ bộ điều khiển.
- Thiết bị nói (talker) là thiết bị gửi số liệu đến thiết bị khác.
- Thiết bị điều khiển (controller) là thiết bị nắm quyền điều khiển, phân định thiết bị nào là nói hoặc nghe trên bus.

Một thiết bị có thể có cả hai chức năng nghe hoặc nói. Về mặt vật lí, bus GPIB gồm 24 dây với các đầu nối được vẽ ra như hình 12.2.

Mỗi đầu cáp đồng thời vừa có cả các chân nối đực và cái nhằm cho phép nối móc xích các thiết bị trên bus với nhau. Các thiết bị dùng trong bus GPIB thường có 5 chuyển mạch hai vị trí (0,1) cho phép người dùng chọn cho nó 1 trong 32 địa chỉ. Tín hiệu trên bus tương thích với mức lôgic TTL. Như thấy trên hình, bus GPIB có 8 đường dữ liệu hai hướng. Những đường này được dùng để truyền số liệu, địa chỉ, lệnh và các byte trạng thái giữa các thiết bị.



Mã								
D8	D7	D6	D5	D4	D3	D2	D1	Ý nghĩa
X	0	0	0	B4	B3	B2	B1	Các lệnh đa năng
X	0	1	A5	A4	A3	A2	A1	Các địa chỉ nghe
X	0	1	1	1	1	1	1	Lệnh không nghe
X	1	0	A5	A4	A3	A2	A1	Các địa chỉ nói
X	1	0	1	1	1	1	1	lệnh không nói
X	1	1	A5	A4	A3	A2	A1	Các lệnh thứ cấp
X	1	1	1	1	1	1	1	Bỏ qua

Mã cho loại lệnh

Hình 12.2 Đầu cắm GPIB và các chân tín hiệu.

Có 5 đường dùng cho quản lí bus với các tín hiệu:

- IFC (clear) cho phép reset tất cả các thiết bị trên bus về trạng thái ban đầu.
- ATN (attention) khi ở mức thấp nó báo rằng thiết bị điều khiển đang đặt một lệnh hoặc một lệnh địa chỉ lên bus dữ liệu. Khi dây này ở mức cao, các đường dữ liệu chứa số liệu hoặc byte trạng thái.
- SRQ (service request) giống như trường hợp ngắt, khi một thiết bị khi cần truyền số liệu trên bus nó sẽ phát ra tín hiệu này xuống mức thấp. Thiết bị điều khiển lúc đó sẽ thăm dò tất cả các thiết bị để xác định xem thiết bị nào có yêu cầu phục vụ.
- REN (remote enable) thiết bị điều khiển làm tích cực dây này để cho phép một thiết bị được nó điều khiển trực tiếp thay vì bởi các chuyển mạch trên thiết bị đó.
- EOI (end of identify) được thiết bị nói phát ra để chỉ thị rằng đã truyền xong một khối số liệu.

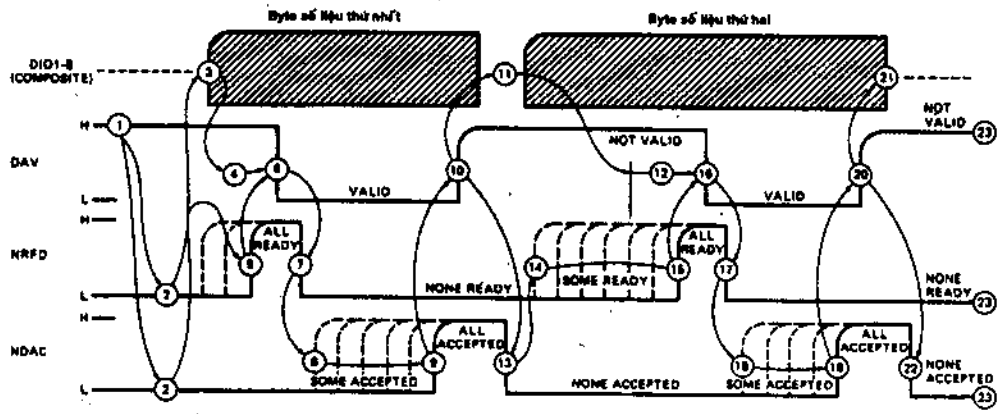
Có 3 đường tín hiệu móc nối, các đường này là: DAV (data valid - số liệu đã ổn định), NRFD (not ready for data - chưa sẵn sàng có số liệu) và NDAC (not data accepted - chưa nhận số liệu). Những tín hiệu này như trên đã nói cho phép các thiết bị có tốc độ truyền số liệu rất khác nhau có thể được nối với nhau trên cùng hệ thống.

Ngay khi hệ thống được cấp điện, Controller sẽ nắm lấy quyền điều khiển bus và gửi ra tín hiệu IFC để reset tất cả các thiết bị trên bus vào trạng thái ban đầu. Tiếp đó nó sẽ xử lí chuỗi hoạt động đo hoặc kiểm tra theo chương trình. Nó gửi một chuỗi các lệnh bằng cách cho đường ATN xuống mức tích cực thấp. Trên hình 12.2 ta thấy các mã lệnh địa chỉ mà nó gửi cho các talker và listener. Bit 6 và 7 phân định lệnh nào đang được gửi, và các bit từ 1 đến 5 xác định địa chỉ đích. Thí dụ, để đặt một thiết bị ở địa chỉ 04 trở thành talker, thiết bị điều khiển làm dây ATN xuống mức thấp và xuất ra byte lệnh x1000100. Khi truyền xong số liệu, tất cả các listener bị cấm nghe bởi controller phát ra byte lệnh unlisten (cấm nghe) x0111111. Controller

cũng cấm nói các talker bằng byte lệnh untalk x1011111. Các lệnh thông thường được phát ra bởi controller với các bit 7, 6 và 5 bằng 0 sẽ đi tới tất cả các listener và talker. 4 bit thấp của byte lệnh này xác định một trong 16 lệnh thông thường.

Controller chú ý đến bus bằng việc kiểm tra định kỳ dây SQR. Nếu dây này ở mức thấp, tức là có một thiết bị nào đó muốn được phục vụ, thì controller sẽ thực hiện thăm dò các thiết bị theo cách lần lượt hoặc đồng thời cho đến khi tìm ra thiết bị đó.

Hình 12.3 trình bày các dạng sóng điển hình trong truyền tín hiệu có móc nối trên bus GPIB từ một talker tới một vài listener.



Hình 12.3 Các dạng sóng trong thông tin với chuẩn GPIB.

Các đường DAV, NRFD và NDAC được nối với các cổng collector hở. Do đó một listener bất kì đều có thể giữ dây NRFD ở mức thấp để báo rằng nó chưa sẵn sàng nhận số liệu. Khi tất cả các listener rời dây NRFD (điểm số 5 trên hình) tức là chúng đã sẵn sàng nhận số liệu thì talker hạ đường DAV xuống thấp để chỉ thị rằng một byte số liệu hợp lệ đã ở trên bus. Những listener được định vị khi đó sẽ kéo dây NRFD xuống thấp và khởi phát việc đọc số liệu. Khi listener chậm nhất đọc xong, dây NDAC sẽ lên mức cao (điểm số 9). Talker khi nhận ra dây NDAC ở mức cao, nó sẽ nâng dây DAV lên mức cao và kéo dây NDAC xuống mức thấp lần nữa và chuỗi hoạt động như vậy sẽ lặp lại đến khi tất cả các byte số liệu đã được truyền. Cuối cùng talker sẽ đặt dây EOI xuống thấp để báo cho controller biết rằng đã hoàn tất công việc. Controller lúc này sẽ nắm quyền điều khiển bus và gửi một lệnh untalk tới talker.Nó cũng đồng thời gửi lệnh unlisten để cấm các listener và tiếp tục thực hiện hoạt động khác theo chương trình. Chuẩn ghép nối GPIB - IEEE488 cho phép một tốc độ truyền dữ liệu tới 10 Mbps và có thể điều khiển được một lúc 15 thiết bị.

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

12.1 Tại sao công ti điện thoại lại truyền các tín hiệu trên khoảng cách dài dưới dạng số chứ không phải analog ?

12.2 Bộ codec là gì ?

12.3 Tại sao bộ codec lại được thiết kế phi tuyến ?

12.4 Tại sao các công ti điện thoại thường truyền nhiều tín hiệu thoại trên 1 dây (hay 1 kênh) ?

12.5 Tại sao thông tin nối tiếp đồng bộ hiệu quả hơn nhiều thông tin không đồng bộ ?

12.6 Vẽ sơ đồ đơn giản chỉ ra 5 tôpô mạng cục bộ như đã được trình bày trong sách ? Thử so sánh tính kinh tế, độ tin cậy, an toàn giữa các tôpô.

12.7 Nêu sự khác nhau giữa mạng dùng băng gốc và băng rộng ?

12.8 Liệt kê kiến trúc vật lí của mạng LAN: các kết nối đường truyền và các thiết bị mạng ?

12.9 Liệt kê 7 tầng của hệ thống mở OSI ? Tầng nào có trách nhiệm ghép các thông điệp thành khung truyền hoặc thành các gói ? Tầng nào có trách nhiệm đảm bảo thông điệp được truyền và nhận một cách chính xác ?

12.10 Mô tả tôpô và các kết nối vật lí của mạng Ethernet ?

12.11 Phương pháp CSMA/CD là gì ? Mô tả chi tiết phương pháp này được dùng trên mạng Ethernet ?

12.12 Bus GPIB được thiết kế dùng cho mục đích gì ? Tên của 3 loại thiết bị mà GPIB định nghĩa ?

12.13 Liệt kê và mô tả tóm tắt chức năng của 3 nhóm tín hiệu trong GPIB ?

12.14 Mô tả chuỗi tín hiệu móc nối xảy ra khi một talker trên bus GPIB truyền số liệu tới một vài listener. Tại sao quá trình móc nối này lại cho phép các talker và listener có tốc độ truyền số liệu rất khác nhau lại có thể làm việc trao đổi chính xác số liệu cho nhau trên bus GPIB ?

CHƯƠNG 13. VÀO/ RA SỐ BẰNG VI MẠCH PPI-8255A

Các cổng *song song* LPT và *nối tiếp* COM cũng như các *khe cắm mở rộng* (expansion slots) cho phép máy vi tính có khả năng thu thập số liệu số rồi xuất ra các tín hiệu số điều khiển ngoại vi sau khi đã xử lí. Chính khả năng này đã mở ra những ứng dụng cực kì phong phú của máy vi tính trong đo lường và điều khiển tự động.

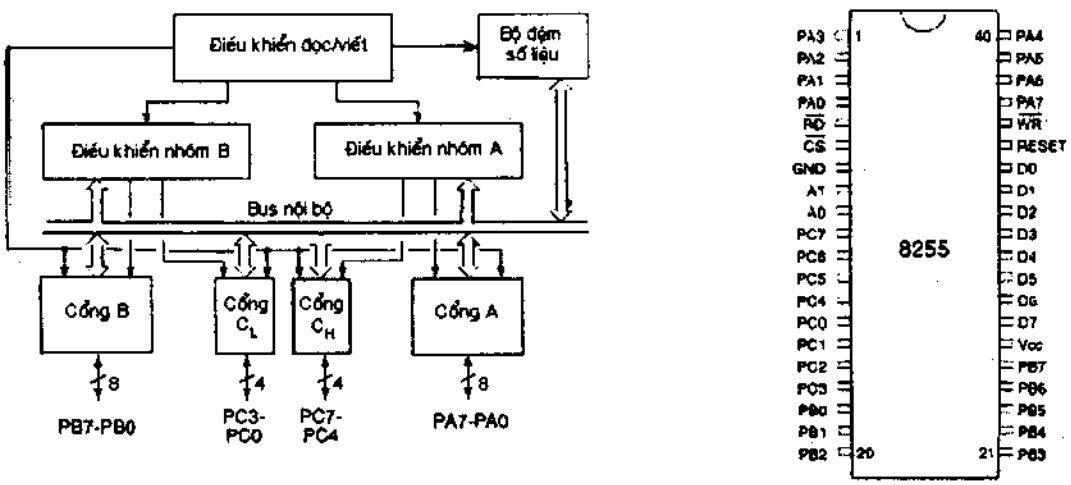
Khi thiết bị ngoại vi là một thiết bị số (thí dụ như các bộ chỉ thị mức lôgic, lối ra các bộ so sánh v.v...) thì việc thông tin giữa chúng và PC hoàn toàn là bằng tín hiệu số. Quá trình vào/ ra như vậy gọi là vào/ra số. Ngoài các cổng LPT và cổng COM như đã trình bày ở các chương trước, cũng có thể xây dựng các bản mạch ghép nối vào/ ra số cắm trên các khe cắm mở rộng của máy vi tính (thí dụ khe cắm ISA). Máy tính sẽ trao đổi dữ liệu số với các ngoại vi qua các vi mạch ghép nối như PPI-8255A lắp trên đó. Các vi mạch này cũng được thiết kế những địa chỉ cơ sở của riêng chúng. Những địa chỉ này phải nằm trong vùng dành cho người sử dụng.

PPI-8255A (Programmable Peripheral Interface) là một vi mạch vào/ra đa năng có thể chương trình hoá được Nó được thiết kế để ghép nối các thiết bị ngoại vi với hệ thống máy tính dùng vi xử lí của Intel theo phương thức truyền dữ liệu song song. Độ tích hợp của vi mạch này tạo một ưu điểm là bản mạch ghép nối dùng 8255A chỉ cần thêm rất ít các linh kiện phụ trợ.

13.1. CẤU TẠO CỦA VI MẠCH PPI 8255A

13.1.1. Sơ đồ khối và chân tín hiệu

Sơ đồ khối và các chân tín hiệu của của vi mạch 8255A như hình 13.1.



Hình 13.1 Sơ đồ cấu tạo và các chân tín hiệu của PPI-8255.

- 24 đầu của 3 cổng vào/ra 8 bit hai hướng: PA0-PA7 của cổng A, PB0-PB7 của cổng B và PC0-PC7 của cổng C được chia làm 2 nhóm A và B: nhóm A gồm cổng A và nibble cao của cổng C_H ; nhóm B gồm cổng B và nibble thấp của cổng C_L. Vi mạch được chương trình hoá để hoạt động trong 3 chế độ (0, 1 và 2).

- *Bộ đệm số liệu* là loại 8 bit-2 hướng-3 trạng thái được dùng để phối ghép 8255A với bus hệ thống của PC. Số liệu sẽ được bộ đệm phát hoặc thu bằng việc chạy các lệnh OUT

hoặc IN của CPU. Các từ điều khiển và thông tin trạng thái cũng được truyền qua bộ đệm này.

• *Các mạch lôgic đọc/viết và điều khiển* quản lí các quá trình truyền số liệu, truyền các từ điều khiển bên trong và bên ngoài vi mạch. Nó chấp nhận các tín hiệu vào từ các bus địa chỉ và điều khiển của CPU và lần lượt phát ra các lệnh tới cả hai nhóm điều khiển A và B.

• Chân $\overline{CS}$ khi ở mức tích cực thấp cho phép 8255A thông tin được với CPU.

• Chân $\overline{RD}$ khi ở mức tích cực thấp cho phép CPU đọc số liệu hoặc thông tin trạng thái từ 8255A.

• Chân $\overline{WR}$ khi ở mức tích cực thấp cho phép CPU viết số liệu hoặc từ điều khiển vào 8255A.

• Hai chân *A0* và *A1* thường được nối với 2 bit địa chỉ thấp nhất của bus địa chỉ PC. Khi kết hợp với các chân RD và WR sẽ cho phép truy cập hoặc 1 trong 3 cổng hoặc thanh ghi từ điều khiển theo quy luật sau:

A1	*A0*	*RD*	*WR*	
0	0	0	1	đọc cổng A vào bus số liệu.
0	1	0	1	đọc cổng B vào bus số liệu.
1	0	0	1	đọc cổng C vào bus số liệu.
1	1	0	1	đọc từ điều khiển vào bus số liệu.
0	0	1	0	viết từ bus số liệu vào cổng A.
0	1	1	0	viết từ bus số liệu vào cổng B.
1	0	1	0	viết từ bus số liệu vào cổng C.
1	1	1	0	viết từ bus số liệu vào từ điều khiển.

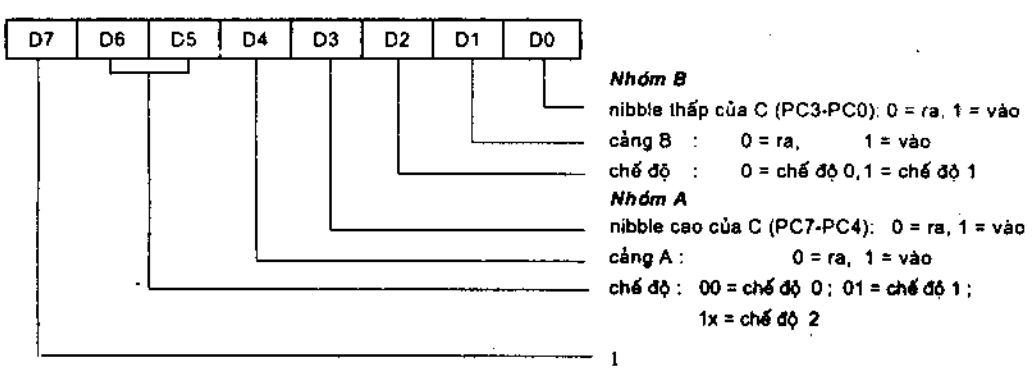
• Chân RESET ở mức cao sẽ xoá các thanh ghi điều khiển và đặt tất cả các cổng A, B và C ở chế độ vào.

13.2. LẬP TRÌNH VỚI PPI 8255A

13.2.1. Chọn chế độ hoạt động

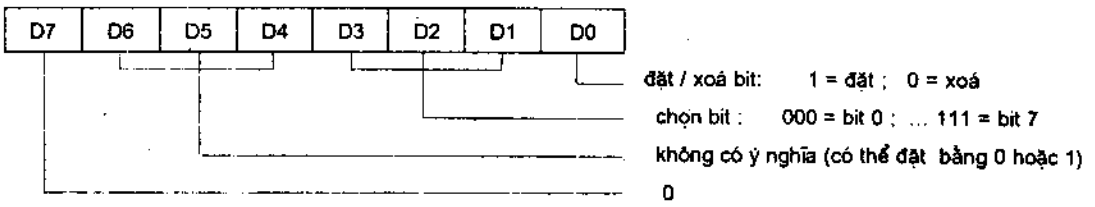
PPI-8255A có 3 chế độ hoạt động cơ bản từ 0 đến 2 được chọn bởi phần mềm. Sau khi được reset, tất cả các cổng của PPI-8255A sẽ được đặt ở mode vào (tức là cả 24 chân ở 3 cổng A, B và C sẽ ở trạng thái trở kháng cao). Trong quá trình chạy chương trình, bằng các lệnh viết ra địa chỉ của thanh ghi từ điều khiển có thể chọn các chế độ hoạt động tuỳ ý. Định dạng của từ điều khiển như sau.

Chế độ viết từ điều khiển (D7 = 1)



Đặt/xoá các bit trên cổng C.

Riêng các bit của cổng C có thể được đặt hay xoá bằng việc viết từ điều khiển với bit D7=0 vào thanh ghi từ điều khiển. Đặc điểm này làm giảm bớt các yêu cầu phần mềm trong những ứng dụng điều khiển. Khi cổng C được dùng để điều khiển hoặc giữ trạng thái của hai cổng A và B, những bít này có thể được đặt/xoá bằng thao tác nêu trên như thể chúng là những cổng ra số liệu.



13.2.2. Các chức năng điều khiển ngắt

Khi PPI-8255A được lập trình để hoạt động trong chế độ 1 hoặc 2, các tín hiệu điều khiển sẽ được cung cấp nhằm có thể sử dụng chúng như các tín hiệu yêu cầu ngắt tới CPU. Các tín hiệu yêu cầu ngắt được phát từ cổng C có thể được cho phép (enable) hoặc bị cấm (inhibited) bằng việc đặt (set) hoặc đặt lại (reset) một mạch lật INTE có liên quan khi sử dụng chức năng đặt/ xoá các bit trên cổng C. Chức năng này cho phép người lập trình cấm hoặc cho phép một thiết bị vào/ra nào đó gửi ngắt tới CPU mà không phải tác động tới bất kì một thiết bị khác trong cấu trúc ngắt của PC.

Định nghĩa trạng thái của mạch lật INTE như sau:

Đặt bit - INTE được đặt - cho phép ngắt

Xoá bit - INTE được xoá - cấm ngắt.

Lưu ý rằng tất cả các mạch lật che (mask flip-flop) được tự động xoá trong suốt thời gian chọn chế độ và khởi động lại thiết bị.

13.2.3. Các chế độ hoạt động

Ta sẽ xét tỉ mỉ các chế độ hoạt động như sau.

a) Chế độ 0: Vào/ra một chiều đơn giản không có móc nối (no-handshake).

- Bất kì cổng nào cũng có thể là cổng vào hoặc cổng ra.
- Trong mode ra (out), số liệu viết ra cổng được chốt. Byte số liệu tồn tại trên các đầu ra của cổng cho tới khi một byte mới được viết tới đó.
- Trong mode vào (in) sẽ đọc được giá trị của byte hiện diện trên các đầu nối của cổng. Các số liệu này không được chốt.

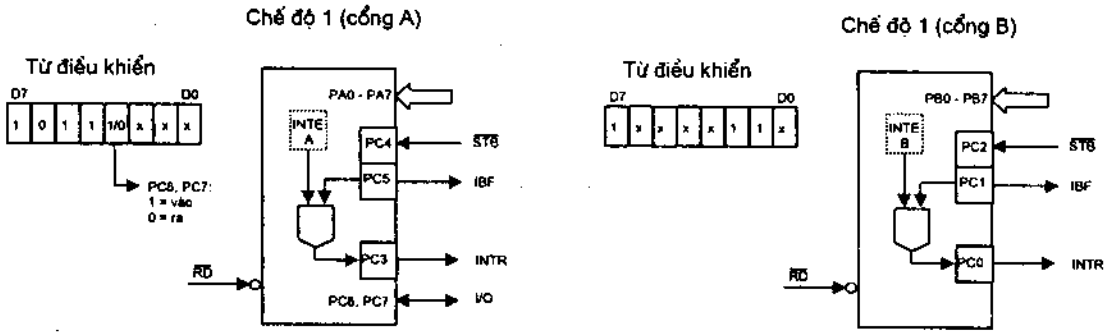
b) Chế độ 1: Vào/ra hai chiều có móc nối qua nibble cao C_H của cổng C. Các tín hiệu móc nối được định nghĩa như dưới đây.

- ***Trong mode vào***:

STB (strobe input): khi thiết bị ngoại vi nối với cổng làm đường này ở mức tích cực thấp thì byte hiện diện trên cổng được nạp vào thanh ghi chốt lối vào.

IBF (input buffer full): lối ra này nhảy lên mức cao để chỉ thị rằng 1 byte đã được chốt vào bộ chốt lối vào. IBF trở về mức thấp bởi sườn lên của xung đọc lối vào RD.

INTR (interrrupt request): mức cao của lối ra này có thể được dùng để ngắt CPU khi một thiết bị vào đòi hỏi phục vụ ngắt. Lối ra INTR được đặt lên 1 khi *STB* =1, IBF=1 và INTE=1. Sườn xuống của xung đọc RD sẽ làm lối ra này trở về mức thấp. Thủ tục này cho phép một thiết bị vào yêu cầu phục vụ từ CPU bằng việc đơn giản là gửi một xung *STB* tới cổng. INTE A được điều khiển bằng việc đặt/xoá bit PC4, INTE B được điều khiển bằng việc đặt/xoá bit PC2.



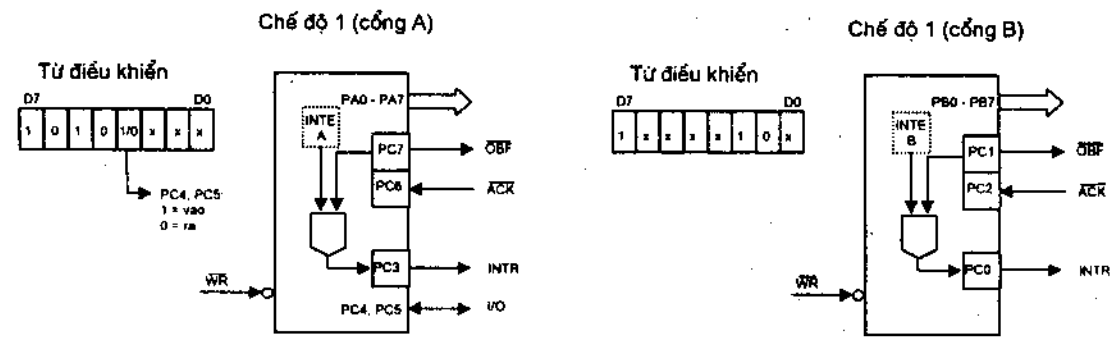
Hình 13.2 Chế độ 1 ở mode vào.

- ***Trong mode ra***:

OBF (output buffer full): lối ra này xuống mức thấp để chỉ thị rằng CPU đã viết xong số liệu ra cổng. Lối ra được đặt (1) bằng sườn lên của xung lối vào WR và được xoá (0) bằng mức thấp của xung lối vào ACK.

ACK (acknowledge): thiết bị ngoại vi làm tín hiệu đến lối vào ở mức thấp thông báo cho PPI-8255 biết rằng số liệu từ cổng A hoặc B đã được nhận. Điều này làm lối ra *OBF* nhảy lên mức cao để cho phép chương trình có thể gửi một byte mới nữa.

INTR (interrupt request): nếu cho phép, một ngắt được phát ra khi bộ đệm ra rỗng (thiết bị ra sẵn sàng nhận số liệu mới). INTR được đặt khi ACK=1, OBF=1 và INTE =1. Sườn xuống xung viết WR làm lối ra này trở về mức thấp.



Hình 13.3 Chế độ 1 ở mode ra.

c) Chế độ 2: Vào/ ra hai chiều qua cổng A có móc nối qua nibble cao và bit C3 của cổng C. Chức năng này cung cấp một phương tiện cho việc thông tin với các thiết bị ngoại vi theo phương thức phát và thu đồng thời (vào/ra bus hai hướng). Các tín hiệu móc nối được sử dụng để điều khiển luồng số liệu tương tự như chế độ 1.

- Chế độ này chỉ dùng cho nhóm A.
- Cả hai cổng vào và ra đều có chốt số liệu

- 5 bit trên cổng C được dùng để điều khiển và lưu trữ trạng thái cho cổng A hai chiều vào/ra, 8 bit.

Định nghĩa các tín hiệu điều khiển vào/ra trên bus 2 hướng như sau:

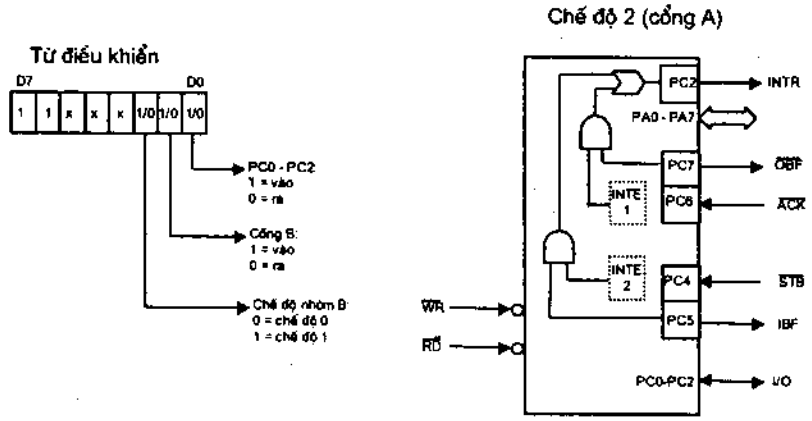
INTR (interrupt request): mức cao trên lối ra này có thể được dùng để ngắt CPU cho cả hai hoạt động vào hoặc ra.

Với hoạt động ra:

$\overline{OBF}$ (output buffer full): mức thấp để chỉ thị rằng CPU đã viết xong số liệu tới cổng A.

$\overline{ACK}$ (acknowledge): mức thấp cho phép bộ đệm ra 3 trạng thái phát số liệu. Ngược lại bộ đệm này sẽ ở trạng thái trở kháng cao.

INTE 1 (mạch lật INTE liên quan đến OBF): được điều khiển bởi việc đặt/xoá bit PC6.



Hình 13.4 Chế độ 2 (ở cả 2 hướng thông tin).

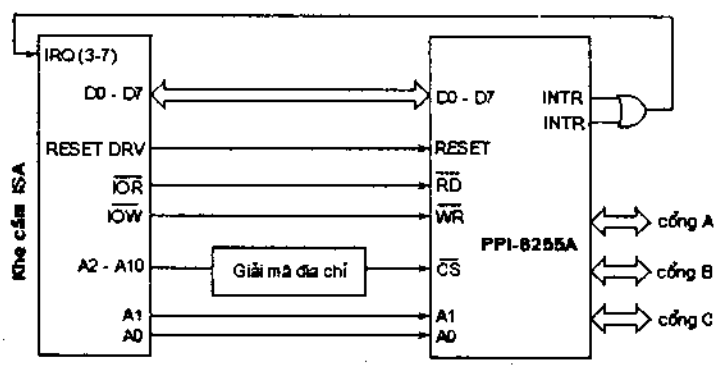
Với hoạt động vào:

$\overline{STB}$ (strobe input): mức thấp để nạp số liệu vào chốt lối vào.

IBF (input buffer full): mức cao chỉ thị rằng số liệu đã được nạp vào bộ chốt lối vào.

INTE 2 (mạch lật INTE liên quan đến IBF): được điều khiển bởi việc đặt/xoá bit PC4.

Hình 13.5 là sơ đồ nối các chân tín hiệu giữa vi mạch 8255A lắp trên bản mạch ghép nối và máy vi tính cắm qua khe cắm mở rộng ISA. Bộ giải mã địa chỉ có thể gồm các cổng AND và NOT.



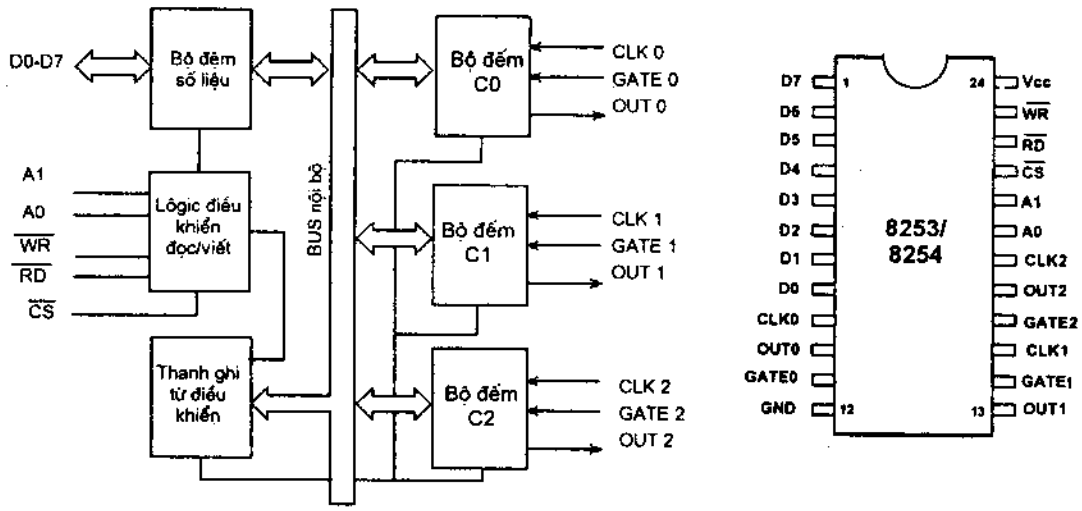
Hình 13.5 Sơ đồ nối giữa 8255A và PC qua khe cắm ISA.

CHƯƠNG 14. ĐO VÀ TẠO THỜI GIAN CHÍNH XÁC BẰNG CHIP PIT-8253/54 GHÉP NỐI MÁY TÍNH

Khi cần đo hoặc tạo ra những xung nhịp thời gian chính xác và ổn định bằng một hệ thống ghép nối máy tính người ta thường hay sử dụng vi mạch định thời khả trình PIT-8253 (Programmable Interval Timer). Ngoài việc thực hiện 3 chức năng trong PC đã được nói tới như một chip bổ trợ, loại vi mạch này có thể sử dụng tốt với các vi tính dùng họ vi xử lí của Intel khi được thiết kế lắp ráp trên các bản mạch ghép nối cắm vào một khe cắm mở rộng. Nó có thể phát ra các khoảng thời gian trễ chính xác dưới sự điều khiển của phần mềm. Thay vì phải khởi tạo các vòng lặp (loop) trong chương trình hệ thống, người lập trình chỉ việc định cấu hình của 8253 để thoả mãn yêu cầu của mình: khởi động một trong các bộ đếm của nó với định lượng mong muốn. PIT-8253 khi được lắp trên các bản mạch ghép nối có thể phục vụ được rất nhiều mục đích như: tạo máy phát sóng tỉ lệ khả trình, đếm các sự kiện, tạo đồng hồ thời gian thực, tạo các xung đơn, đo các khoảng thời gian chính xác, đo tần số, điều khiển môtơ bước v.v...

14.1. CẤU TẠO VÀ CÁC CHÂN TÍN HIỆU CỦA PIT 8253/54

Sơ đồ khối và các chân tín hiệu như hình 14.1



Hình 14.1 Sơ đồ khối và các chân tín hiệu của 8253/54.

• *Bộ đệm số liệu* là loại 8 bit-2 hướng-3 trạng thái được dùng để phối ghép 8253 với bus hệ thống của PC. Số liệu sẽ được bộ đệm phát hoặc thu bằng việc chạy các lệnh OUT hoặc IN của CPU. Nó có 3 chức năng cơ bản: chương trình các chế độ hoạt động của 8253/54, nạp các thanh ghi của bộ đếm và đọc các giá trị đếm

• *Các mạch lôgic đọc/viết* nhận các tín hiệu vào từ bus hệ thống và lần lượt phát các tín hiệu điều khiển cho thiết bị hoạt động. Nó bị cấm hoặc được cho phép bởi tín hiệu chọn chip CS nhằm không để một hoạt động nào xảy ra chừng nào mà chip chưa được chọn bởi hệ thống PC.

• Chân *RD* khi ở mức thấp cho phép CPU đọc số liệu từ 8253 trong dạng một giá trị các bộ đếm.

• Chân $\overline{WR}$ khi ở mức thấp cho phép CPU viết dữ liệu vào 8253, đó là các thông tin về chế độ hoạt động hoặc nạp các số đếm ban đầu vào bộ đếm.

• Hai chân *A0* và *A1* thường được nối với 2 bit địa chỉ thấp nhất của bus địa chỉ PC. Khi kết hợp với các chân RD và WR sẽ cho phép truy cập hoặc 1 trong 3 bộ đếm hoặc thanh ghi từ điều khiển theo quy luật sau:

A1	*A0*	$\overline{RD}$	$\overline{WR}$	
0	0	0	1	đọc bộ đếm 0
0	1	0	1	đọc bộ đếm 1
1	0	0	1	đọc bộ đếm 2
1	1	0	1	đọc từ điều khiển (chỉ có ở 8254)
0	0	1	0	viết (nạp dữ liệu) vào bộ đếm 0
0	1	1	0	viết vào bộ đếm 1
1	0	1	0	viết vào bộ đếm 2
1	1	1	0	viết từ điều khiển vào thanh ghi từ điều khiển

• Chân $\overline{CS}$ chọn chip khi ở mức thấp sẽ cho phép 8253 hoạt động. Chừng nào chip chưa được chọn, tất cả các hoạt động đọc hoặc viết đều bị ngừng.

• *Thanh ghi từ điều khiển* được chọn khi 2 bit A0 và A1 bằng 1. Thông tin được lưu trữ trong thanh ghi này sẽ điều khiển chế độ hoạt động của các bộ đếm, việc chọn cách đếm (nhị phân hay BCD) và phương thức đọc/viết của mỗi thanh ghi đếm số liệu của các bộ đếm. Trong 8253 đây là thanh ghi chỉ được viết. Trong 8254 có thể cả đọc cả viết được.

• *Ba bộ đếm C0, C1 và C2* mỗi bộ đếm là loại đếm lùi, 16 bit và có thể đặt trước giá trị đếm ban đầu. Mỗi bộ đếm có 2 lối vào xung nhịp đếm CLK, lối vào cửa GATE và 1 lối ra OUT. Khi GATE bằng 1, bộ đếm được phép đếm. Khi GATE bằng 0, bộ đếm bị cấm.

14.2. LẬP TRÌNH CHO 8253/54

14.2.1. Định dạng từ điều khiển

Bằng các lệnh vào/ra đơn giản có thể lập trình cho các hoạt động của 8253. Bước đầu là phải viết từ điều khiển vào thanh ghi từ điều khiển. Định dạng của từ điều khiển như sau.

D7	D6	D5	D4	D3	D2	D1	D0
SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

Trong đó: SC_1, SC_0 là hai bit *chọn bộ đếm*,

00 : bộ đếm 0 được chọn;
01 : bộ đếm 1 được chọn;
10 : bộ đếm 2 được chọn.
11 : lệnh đọc thanh ghi từ điều khiển trong 8254.

RW_1, RW_0 là hai bit *quy định phương thức đọc / viết*,

00 : lệnh chốt bộ đếm
01 : chỉ đọc/viết byte thấp
10 : chỉ đọc/viết byte cao
11 : đọc/viết byte thấp trước, byte cao sau.

M_2, M_1, M_0 *chọn chế độ đếm*,

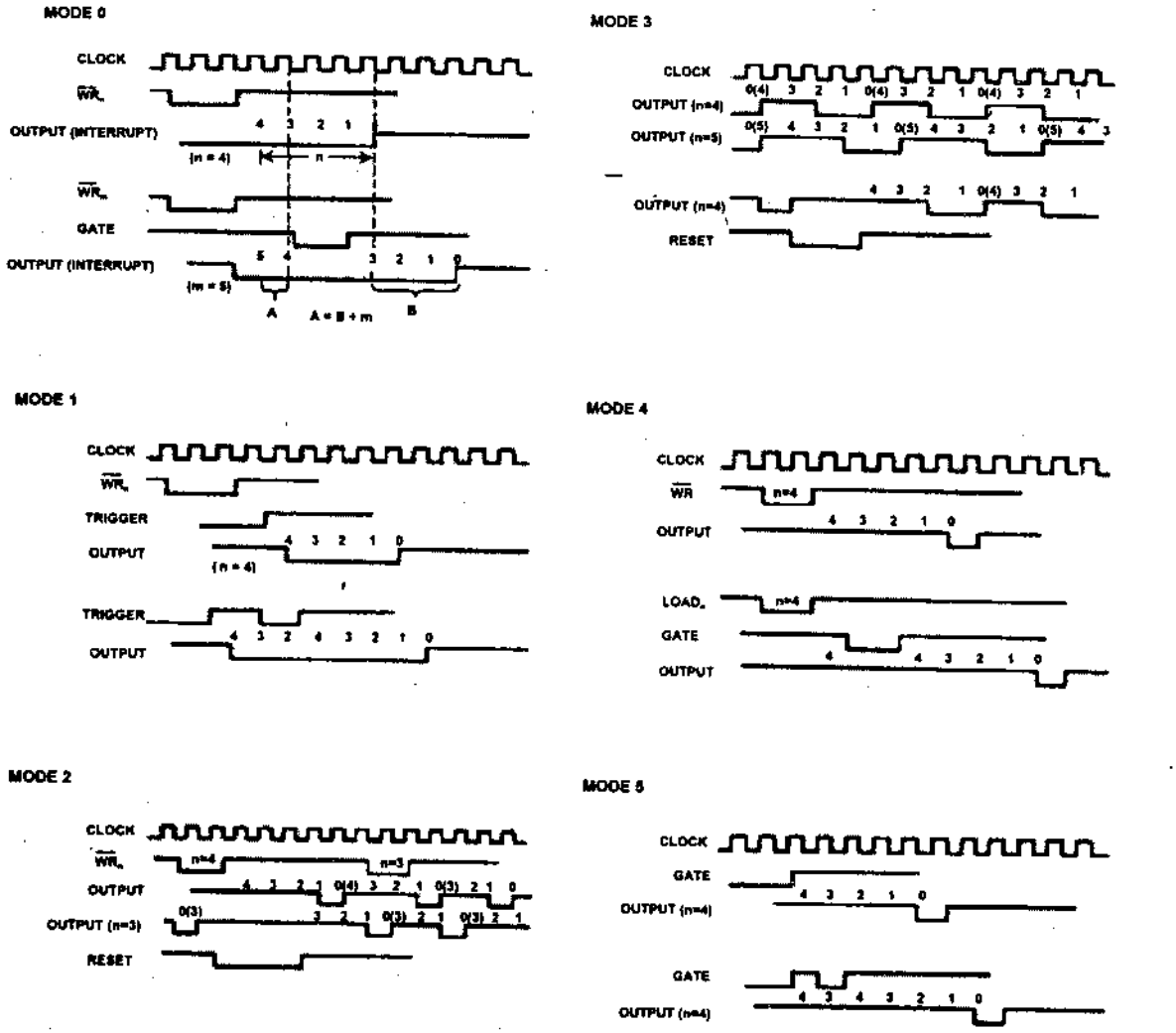
000 = chế độ 0 001 = chế độ 1 010 = chế độ 2
011 = chế độ 3 100 = chế độ 4 101 = chế độ 5.

BCD là bit *định dạng đếm*,

0 : đếm nhị phân 16 bit, từ 0 h đến ffff h.
1 : đếm với mã BCD từ 0 đến 9999.

14.2.2. Các chế độ đếm

Hình 14.2 là tóm tắt các dạng sóng trong các chế độ đếm từ 0 đến 5 theo tài liệu của hãng Intel.



Hình 14.2 Các chế độ đếm của PIT-8253/54.

• *Chế độ 0* (Interrupt on Terminal Count): phát ra một sườn xung lên (thường được dùng làm tín hiệu yêu cầu ngắt trong các hệ thống) khi kết thúc đếm. Tín hiệu lối ra được khởi phát đặt ở mức thấp ngay sau khi chọn chế độ; cho tới khi số xung đếm kết thúc thì sẽ nhảy lên mức cao. Giản đồ thời gian hình 14.2a cho khái niệm về chế độ đếm này với giá trị ban đầu được nạp vào bộ đếm là 4, bộ đếm sẽ đếm lùi xuống tới giá trị 0. Việc nạp lại thanh ghi bộ đếm sẽ gây nên các kết quả sau: nạp byte đầu tiên (byte thấp) sẽ làm ngừng việc đếm hiện tại; nạp byte thứ hai (byte cao) sẽ khởi phát quá trình đếm mới.

• *Chế độ 1* (Programmable Monoflop): phát ra một xung đơn với thời gian tồn tại xung biến đổi được. Lối ra OUT sẽ chuyển xuống mức thấp khi khởi phát đếm và duy trì

trạng thái này cho đến khi kết thúc đếm thì sẽ nhảy lên mức cao. Bộ đếm được khởi động bởi sườn lên của xung cửa GATE. Trong quá trình đếm, nếu xung cửa xuất hiện những sườn lên thì với mỗi lần như vậy bộ đếm lại được đếm lại từ giá trị ban đầu. Chế độ này cho phép tạo ra các xung đơn có độ rộng biến đổi được tuỳ theo giá trị nạp ban đầu. Hình 14.2b cho giá trị nạp là 4.

• *Chế độ 2* (Rate Generator): phát ra các sóng tỉ lệ. Các xung nhịp đưa vào bộ đếm được chia bởi giá trị nạp trong thanh ghi bộ đếm. Lối ra sẽ xuống mức thấp trong một chu kì khi số đếm đạt về 0 và sau đó được nâng lên mức cao trong một chu kì xung nhịp. Quá trình lại cứ tiếp tục như vậy mãi. Giản đồ thời gian hình 14.2c cho giá trị được nạp vào bộ đếm là 4, trong trường hợp này lối ra ở mức cao trong ba chu kì và ở mức thấp trong một chu kì nhịp. Vậy tần số vào CLK đã được chia 4 ở lối ra OUT.

• *Chế độ 3* (Square-Wave Generator): phát ra sóng hình vuông. Chế độ này tương tự như chế độ 2 nhưng xung điện lối ra là sóng vuông khi giá trị đếm là chẵn hoặc thêm một chu kì ở mức cao khi giá trị đếm là lẻ. Giản đồ thời gian như hình 14.2d với giá trị nạp vào bộ đếm là 4, lối ra ở mức cao trong hai chu kì và ở mức thấp trong hai chu kì.

• *Chế độ 4* (Software-Triggered Pulse): tạo các xung bằng phần mềm. Chế độ này giống như chế độ 2 nhưng xung cửa (GATE) không khởi phát quá trình đếm mà quá trình này được khởi phát bằng lệnh nạp số đếm ban đầu. Ngay sau khi nạp chế độ hoạt động, lối ra sẽ ở mức cao. Khi số đếm ban đầu được nạp, bộ đếm bắt đầu đếm. Khi kết thúc đếm, lối ra sẽ nhảy xuống mức thấp trong một chu kì nhịp đồng hồ rồi lại nhảy lên mức cao.

• *Chế độ 5* (Hardware-Trigered Pulse): tạo các xung bằng xung cửa GATE. Cũng giống chế độ 2, sườn lên xung cửa GATE khởi phát quá trình đếm sau đó xuống mức thấp trong một chu kì nhịp khi việc đếm kết thúc. Khi quá trình đếm chưa kết thúc, nếu xung cửa xuất hiện những sườn lên thì cứ mỗi lần như vậy bộ đếm lại được đếm lại từ giá trị ban đầu.

Có thể tóm tắt vai trò của xung cửa GATE trong các chế độ như bảng 14.1 sau.

Bảng 14.1 **Tóm tắt hoạt động của xung cửa GATE 8253/54**

Chế độ \ Trạng thái tín hiệu	Mức thấp hoặc Sườn xuống	Sườn lên	Mức cao
0	Cấm bộ đếm	---	Cho phép đếm
1	---	1) Khởi phát đếm 2) Reset lối ra sau xung nhịp tiếp theo	---
2	1) Cấm đếm 2) Đặt lối ra ở mức cao ngay lập tức	Khởi phát đếm	Cho phép đếm
3	1) Cấm đếm 2) Đặt lối ra ở mức cao ngay lập tức	Khởi phát đếm	Cho phép đếm
4	Cấm đếm	---	Cho phép đếm
5	---	Khởi phát đếm	---

14.2.3. Viết bộ đếm. Chương trình phải thực hiện hai thao tác viết bộ đếm: viết từ điều khiển để xác lập chế độ hoạt động của các bộ đếm và nạp số đếm ban đầu cho các bộ đếm. Sau khi được nạp giá trị ban đầu, bộ đếm khi đếm sẽ đếm lùi từ đó tới 0. Giá trị ban đầu này ứng với hệ số chia tần của bộ đếm trong các chế độ 2 và 3. Số đếm ban đầu

được nạp là 0 sẽ tương đương với việc nạp số đếm cực đại 2^{16} (đếm nhị phân) hoặc 10^4 (đếm BCD).

14.2.4. Đọc bộ đếm

Nếu cần đọc giá trị tức thời của bộ đếm khi quá trình đếm chưa kết thúc thì phải sử dụng các lệnh chốt bộ đếm hoặc lệnh đọc thanh ghi từ điều khiển (Read-back Command). Lệnh đọc thanh ghi từ điều khiển chỉ có trong 8254.

Tại thời điểm lệnh chốt bộ đếm được viết vào thanh ghi từ điều khiển, giá trị đếm được chốt vào thanh ghi đệm ra của PIT.

Dưới đây là định dạng của từ điều khiển cho lệnh chốt bộ đếm

D7	D6	D5	D4	D3	D2	D1	D0
SC_1	SC_0	0	0	x	x	x	x

SC_1, SC_0 là hai bit chọn bộ đếm được chốt

00: bộ đếm 0 được chọn

01: bộ đếm 1 được chọn

10: bộ đếm 2 được chọn

x : không có nghĩa (có thể đặt = 0).

Lệnh đọc thanh ghi từ điều khiển cho phép chốt giá trị đếm và chốt byte lưu trữ trạng thái của bộ đếm được chọn.

Dưới đây là định dạng cho lệnh đọc thanh ghi điều khiển:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	count	status	count2	count1	count0	0

count: 0 = chốt giá trị đếm của bộ đếm được chọn; 1 = không xác định

status: 0 = chốt byte trạng thái của bộ đếm được chọn; 1 = không xác định

count2= 1: chọn bộ đếm 2, count1= 1: chọn bộ đếm 1, count0= 1: chọn bộ đếm 0.

Dưới đây là định dạng byte trạng thái:

D7	D6	D5	D4	D3	D2	D1	D0
pin	null	RW_1	RW_0	M_2	M_1	M_0	BCD

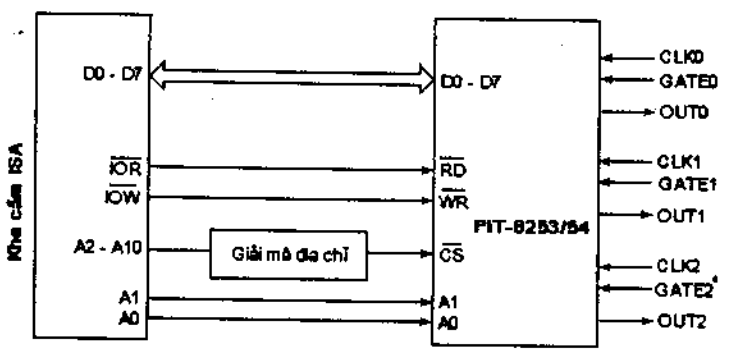
pin: trạng thái của đầu ra OUTx (x = 0, 1 hoặc 2) 1 = đầu ra ở mức cao; 0 = thấp

null: bộ đếm đã được nạp giá trị ban đầu chưa ? 1 = chưa nạp, chưa đọc được; 0 = nạp rồi, đọc được

RW_1, RW_0, M_2, M_1, M_0, BCD: xem phần thanh ghi từ điều khiển

14.2.5. Ghép nối PIT-8253/54 với PC qua khe cắm mở rộng ISA

8253/54 cũng được nối với các chân tín hiệu trên khe cắm mở rộng như 8255A như hình 14.3 sau.



Hình 14.3 Nối 8253/54 với PC qua khe cắm ISA.

CHƯƠNG 15. CÁC MẠCH GHÉP NỐI SỐ - TƯƠNG TỰ (D/A) VÀ TƯƠNG TỰ - SỐ (A/D)

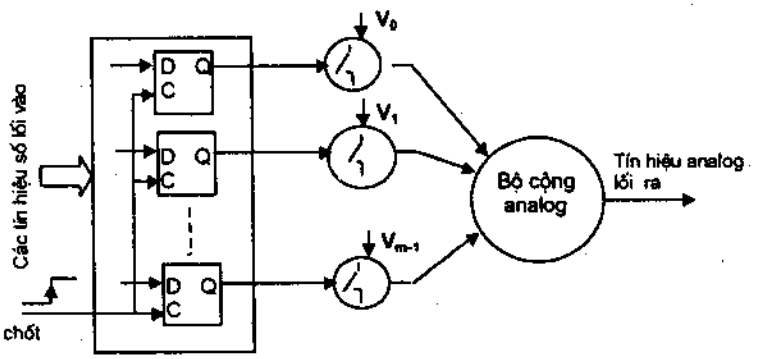
Máy vi tính là thiết bị số trong khi nhiều thiết bị ngoại vi dùng trong đo lường và điều khiển lại là thiết bị thu thập và phát ra các tín hiệu tương tự. Vì vậy, các *bộ biến đổi số-tương tự* (DAC) và *tương tự-số* (ADC) là cầu nối giữa máy vi tính và thế giới tương tự ở bên ngoài.

15.1. BỘ BIẾN ĐỔI SỐ-TƯƠNG TỰ D/A

Các bộ biến đổi D/A (digital to analog convertion) thường được chế tạo trên một vi mạch đơn khối. Các mạch điện đều dựa trên nguyên lí cơ bản như hình 15.1 trong đó cần một mạch cộng analog để cộng tất cả các điện thế analog tỉ lệ với từng bit nhị phân. Trên hình là một tín hiệu vào gồm *m* bit số được chốt bởi *m* mạch chốt là các trigơ D. Mỗi bit thứ *i* sẽ điều khiển các công tắc tương ứng nhằm cho qua điện thế V_i tỉ lệ với các trọng số thứ i tới bộ cộng mỗi khi giá trị nó bằng 1. Như vậy lối ra của bộ cộng sẽ là:

$$V = B_{m-1}V_{m-1} + B_{m-2}V_{m-2} + ... + B_0V_0$$

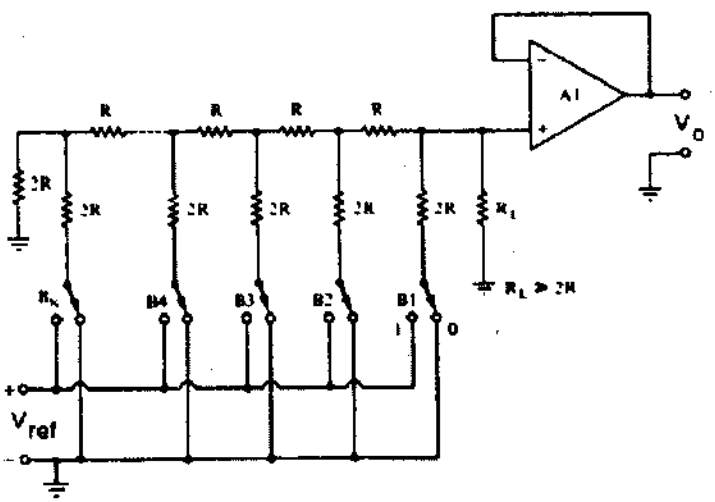
Trong đó B_i là giá trị nhị phân (0 hoặc 1) của bit thứ i của mạch.



Hình 15.1 Bộ biến đổi D/A cơ bản.

Hiện thông dụng nhất là loại dùng thang các điện trở R-2R để cấp dòng cho bộ cộng analog, sơ đồ như hình 15.2. Với các giá trị điện trở được chọn chỉ gồm 2 trị số R và 2R, thế lối ra tỉ lệ với giá trị của các bit mã số lối vào là các công tắc B_N như sau:

$$V_O = -V_{ref}\left(\frac{B_1}{2^1} + \frac{B_2}{2^2} + ... + \frac{B_N}{2^N}\right)$$



Hình 15.2 Mạch biến đổi D/ A kiểu R-2R.

15.1.1. Các thông số và đặc trưng của biến đổi D/A

a, Độ phân giải là đại lượng được xác định bởi số bit của từ nhị phân đưa tới lối vào. Một bộ biến đổi với 8 lối vào nhị phân có thể tạo ra 2^8 hay 256 mức điện áp ra, do đó có độ phân giải là 1 phần 256. Lấy ví dụ khác, một bộ biến đổi 12 bit có độ phân giải là 1 phần 2^{12} hay 1 phần 4096. Đôi khi độ phân giải còn được biểu diễn theo phần trăm. Độ phân giải của bộ biến đổi D/A 8 bit nếu biểu diễn theo phần trăm sẽ là (1/256) × 100% = 0,39%.

b, Điện áp ra toàn thang FS (full scale) là điện áp cực đại ứng với trường hợp tất cả các bit lối vào bằng 1. Giá trị điện áp ra cực đại của một bộ biến đổi luôn nhỏ hơn giá trị danh định ít nhất là 1 bit ít ý nghĩa nhất. Ví dụ, giả sử có một bộ biến đổi D/A 12 bit, điện áp ra toàn thang là 10V thì giá trị của 1 bit LSB là 10V/4096 hay 2,44 mV. Điện áp ra lớn nhất của bộ biến đổi này nếu được điều chỉnh đúng sẽ phải là (10,0000 – 0,0024) V hay 9,9976 V.

c, *Độ chính xác* của một bộ biến đổi D/A là sự sai lệch giữa giá trị lối ra thực tế và giá trị lối ra mong muốn. Nó thường được xác định theo số phần trăm của điện áp hoặc dòng điện toàn thang. Nếu một bộ biến đổi có điện áp toàn thang 10V và sai số là ±0,2% thì *sai số lớn nhất* đối với bất kì giá trị ra nào cũng sẽ là 0,002 × 10,00 V hay 20 mV. Trong trường hợp lí tưởng, sai số lớn nhất không vượt quá giá trị ±1/2 bit LSB.

d, Độ tuyến tính là số đo độ lệch của lối ra so với đường thẳng nối điểm chuyển đổi từ trạng thái 00...0 (không có chuyển mạch nào được đóng) cho tới trạng thái 11...1 (mọi chuyển mạch đều được đóng). Trong trường hợp lí tưởng, độ lệch của lối ra so với giá trị trên đường thẳng không được vượt quá giá trị ±1/2 bit LSB nhằm duy trì độ chính xác toàn cục. Tuy nhiên rất nhiều bộ biến đổi D/A trên thị trường có sai số tuyến tính lớn hơn giá trị này. Ví dụ hãng National Semiconductor đã tung ra thị trường các loại DAC 1020, DAC 1021, DAC 1022 của bộ biến đổi loại phân giải 10 bit. Độ tuyến tính của DAC 1020 là 0,05 %, phù hợp với loại biến đổi 10 bit. Nhưng các bộ biến đổi DAC 1021 lại có độ tuyến tính là 0,1 %, còn độ tuyến tính của DAC 1022 thì lên tới 0,20 %. Câu hỏi được đặt ra là được lợi gì khi dùng một bộ biến đổi 10 bit mà độ tuyến tính của nó chỉ tương đương với các bộ biến đổi 8 hay 9 bit ? Câu trả lời cho vấn đề này nằm ở chỗ có rất nhiều ứng dụng cần đến độ phân giải 10 bit cho các tín hiệu nhỏ nhưng không coi trọng yêu cầu giá trị ra có tuyến tính hay không đối với những tín hiệu lớn. Cái giá phải trả cho một bộ biến đổi D/A không những tỉ lệ với độ phân giải mà còn tỉ lệ với cả độ tuyến tính của nó nữa.

e, *Thời gian xác lập* là thời gian để bộ biến đổi đạt được giá trị trong khoảng ±1/2 bit LSB so với giá trị xác lập cuối cùng. Bộ biến đổi 10 bit DAC1020 của hãng National có thời gian xác lập trung bình là 500 ns cho một biến đổi toàn thang trên lối ra. Thông số này sẽ trở nên cực kì quan trọng nếu bộ biến đổi phải hoạt động ở tần số cao vì có thể nó sẽ không có đủ thời gian để xác lập một giá trị trước khi chuyển sang giá trị tiếp theo.

Trong thực tế có thể có nhiều loại mã cho các số lối vào trên những bộ biến đổi D/A như *mã nhị phân thuận (straight binary), mã BCD, mã nhị phân dịch (offset binary) v.v...*

15.1.2 . Các ứng dụng của bộ biến đổi D/A

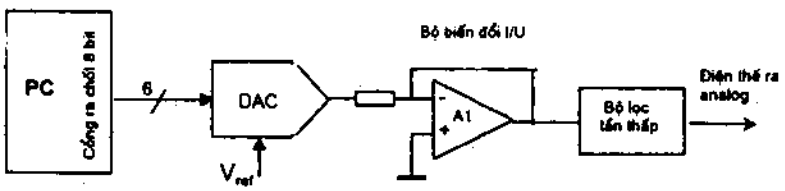
Ngoài những ứng dụng khi được ghép nối với máy vi tính, các bộ biến đổi D/A còn có rất nhiều ứng dụng. Ví dụ trong các máy nghe nhạc sử dụng đĩa compact, người ta dùng một bộ biến đổi D/A 14 hoặc 16 bit để biến đổi dòng số liệu nhị phân được đọc ra từ đĩa bởi tia laser thành tín hiệu audio tương tự. Phần lớn các vi mạch tổng hợp tiếng nói (speech synthesizer IC) đều chứa một bộ biến đổi D/A dùng để biến đổi số liệu ở dạng nhị phân được lưu trong vi mạch thành tín hiệu audio tương tự. Tuy nhiên, ở đây chúng ta chủ yếu quan tâm đến việc sử dụng bộ biến đổi D/A cùng với máy vi tính.

Các lối vào trong sơ đồ bộ biến đổi D/A có thể nối trực tiếp với các chân tín hiệu của một cổng ra trong máy tính. Như vậy, bằng chương trình phần mềm có thể tạo ra bất kì dạng điện áp nào mong muốn trên lối ra của bộ biến đổi D/A bằng cách xuất các số ra cổng theo một quy luật thích hợp. Ngoài ra, còn có thể dùng điện áp lối ra biến đổi này điều khiển dòng cấp cho các mô tơ điện nhằm điều khiển tốc độ quay của mô tơ v.v...

15.1.3. Ghép nối bộ biến đổi D/A với máy tính

Sơ đồ phần cứng đơn giản nhất ghép nối máy tính với vi mạch biến đổi D/A 8bit như hình 15.3a trong đó sử dụng cổng ra chốt 8 bit.

Thường các vi mạch D/A là loại có dòng lối ra tỉ lệ với mã số lối vào nên ở đây sử dụng mạch khuếch đại thuật toán A1 làm bộ biến đổi dòng điện thành điện thế I/U. Do điện thế lối ra biến đổi theo các mức lượng tử nên phải dùng một bộ lọc tần thấp nhằm làm trơn tín hiệu. Có thể sử dụng các cổng vào/ra của vi mạch PPI-8255A làm cổng ra 8 bit trong trường hợp này. Nếu số bit vào của D/A nhiều hơn 8 (thí dụ 12 hoặc 16) ta phải sử dụng tới hai cổng của 8255. Trong trường hợp cần điều khiển nhiều bộ biến đổi D/A phải dùng thêm các mạch chốt số liệu ở ngoài như 74244 v.v.. Khi lập trình xuất ra cổng các số là các hàm biến đổi theo thời gian ta có thể tạo được các sóng tín hiệu hiệu ra tuỳ ý như sin, răng cưa, vuông, tỉ lệ v.v...với tần số và biên độ tuỳ ý mà nếu sử dụng các mạch điện tử lắp cứng có thể khá phức tạp. Ưu điểm nữa ở đây là độ ổn định của tín hiệu ra chỉ phụ thuộc vào nguồn điện thế chuẩn V_{ref} cấp cho DAC.



Hình 15.3 Điều khiển D/A bằng máy vi tính.

Sự ghép nối này chỉ đơn giản là nối các lối vào của bộ biến đổi với các bít của một cổng. Như trên đã nói, trong trường hợp số bit của bộ biến đổi D/A (thí dụ 12 bit) lớn hơn số bit của cổng (thí dụ 8 bit) thì cần phải thiết kế mạch phần cứng và chương trình phần mềm hợp lý. Cách đơn giản nhất là, thí dụ, nối 8 bit thấp trong 12 bit lối vào của bộ biến đổi với một cổng và nối 4 bit cao còn lại với một cổng khác. Lúc này có thể gửi đi 8 bit thấp bằng một chu trình viết và gửi 4 bit cao bằng một chu trình viết khác. Tuy nhiên, nếu đi theo hướng này, khoảng thời gian giữa hai chu trình thường ẩn chứa rất nhiều rắc rối. Lấy thí dụ, giả sử muốn thay đổi lối ra của bộ biến đổi 12 bit từ trạng thái 0000 1111 1111 thành 0001 0000 0000. Khi thực hiện lệnh ghi 8 bit thấp, lối ra sẽ chuyển từ 0000 1111 1111 thành 0000 0000 0000. Phải đến khi viết xong 4 bit cao thì lối ra mới trở về trạng thái mong muốn 0001 0000 0000. Điểm cần chú ý ở đây là trong khoảng thời gian giữa hai lệnh viết lối ra sẽ chuyển sang một giá trị không muốn. Trong nhiều hệ thống, điều này có thể gây ra những thảm hoạ. Giải pháp cho vấn đề này là nối các lối vào với các bộ chốt. Những bộ chốt này có thể nạp số liệu riêng rẽ nhưng sau đó sẽ truyền toàn bộ 12 bit tới bộ biến đổi D/A một cách đồng thời. Rất nhiều bộ biến đổi D/A hiện có trên thị trường có sẵn các bộ chốt được tích hợp trong vi mạch.

Các bộ biến đổi D/A đòi hỏi *điện áp chuẩn* V_{ref} phải rất chính xác. Các bộ biến đổi D/A có lối ra tỉ lệ với dòng điện, do đó thường dùng một bộ khuếch đại thuật toán để biến đổi dòng điện ra

của bộ biến đổi thành điện áp tương ứng I/U. Thường hay sử dụng bộ khuếch đại có lối vào là transistor trường vì dòng điện lối vào phân cực cho bộ khuếch đại lưỡng cực có thể ảnh hưởng đến độ chính xác của lối ra. Các vi mạch D/A thường có sẵn các điện trở phản hồi ở bên trong với đặc trưng nhiệt độ phù hợp với các điện trở chia dòng bên trong vi mạch, nên ta chỉ cần nối thêm bên ngoài một điện trở 50Ω dùng để điều chỉnh. Với điện áp chuẩn là -10V, điện áp ra sẽ là (từ dữ liệu số lối vào/4096) × (+10,000V). Chú ý rằng các chip biến đổi D/A có hai đường nối đất, một đường nối đất tương tự và một đường nối đất số. Để tránh nhiễu số lên các thành phần tương tự, hai chân này chỉ được nối với nhau tại nguồn.

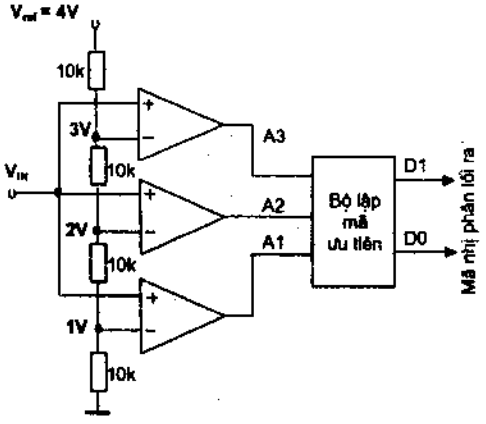
15.2. BỘ BIẾN ĐỔI TƯƠNG TỰ - SỐ A/D

Các bộ biến đổi tương tự-số A/D (analog to digital convertion) có nhiệm vụ biến đổi điện thế hiệu hoặc dòng điện tương tự ở lối vào thành các mã số tương ứng ở lối ra. Nếu số bit lối ra là N ta nói ADC đó có độ phân giải N bit. Hiện nay các chip biến đổi A/D được chế tạo trên các vi mạch đơn khối hoặc mạch lai tuỳ yêu cầu về độ phân giải, tốc độ chuyển đổi v.v.. . Các thông số của một bộ biến đổi A/D rất giống với các thông số của bộ biến đổi D/A. Độ phân giải của bộ biến đổi A/D liên quan đến số bit lối ra. Ví dụ bộ biến đổi 8 bit có độ phân giải là 1 phần 256. Độ chính xác và độ tuyến tính của bộ biến đổi A/D cũng có cùng ý nghĩa như đối với bộ biến đổi D/A. Một thông số quan trọng khác của bộ biến đổi A/D là ***thời gian biến đổi***. Đây là thời gian cần thiết để tạo ra từ mã nhị phân ổn định trên lối ra đối với mỗi điện áp tương tự được đưa tới lối vào. Khi nói một bộ biến đổi có tốc độ cao là muốn ám chỉ rằng nó có thời gian biến đổi ngắn. Có rất nhiều cách để thực hiện một biến đổi A/D, nhưng chúng ta sẽ chỉ đề cập đến 4 phương pháp được sử dụng rộng rãi với thời gian biến đổi rất khác nhau.

15.2.1. Bộ biến đổi A/D kiểu so sánh song song

Hình 15.3b là sơ đồ của một bộ biến đổi A/D 2 bit sử dụng 3 bộ so sánh song song. Người ta dùng một bộ phân áp bằng điện trở để thiết lập điện áp chuẩn đưa tới các lối vào đảo của mỗi bộ so sánh. Điện áp tại mắt điện trở trên cùng quyết định giá trị toàn thang của bộ biến đổi. Điện áp cần được chuyển đổi sang dạng số được đưa song song tới các lối vào không đảo của tất cả các bộ so sánh. Nếu điện áp vào trên một bộ so sánh lớn hơn điện áp chuẩn tại lối vào đảo thì lối ra của bộ so sánh nhảy lên mức cao. Như vậy, lối ra của các bộ so sánh sẽ cho ta biểu diễn số mức điện áp tín hiệu vào.

Thí dụ, trên hình có V_{ref} = 4V, nếu điện áp vào là 2,6V thì lối ra của các bộ so sánh A1 và A2 sẽ ở mức cao. Ưu điểm nổi bật của bộ biến đổi A/D song song, hay còn gọi là bộ biến đổi A/D *chớp nhoáng* (flash), là ở tốc độ chuyển đổi của nó chỉ là thời gian trễ tại các bộ so sánh điện tử. Mã lối ra của các bộ so sánh chưa phải là mã nhị phân chuẩn nhưng có thể biến đổi thành bất kì loại mã nào bằng một số cổng lôgic đơn giản. Nhược điểm chính của các bộ biến đổi A/D chớp nhoáng là số lượng các bộ so sánh cần thiết để thu được một kết quả có độ phân giải chấp nhận được tăng nhanh theo độ phân giải. Bộ biến đổi 2 bit như hình đòi hỏi 3 bộ so sánh. Để lắp một bộ

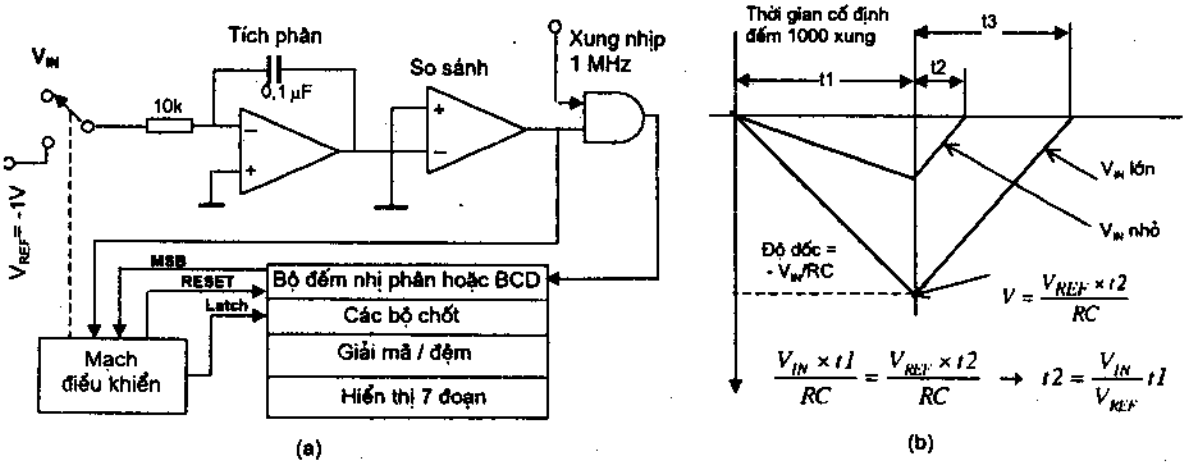


Hình 15.3b Bộ biến đổi A/D song song.

biến đổi có độ phân giải N bit phải cần tới $(2^N - 1)$ bộ so sánh. Do vậy, với một chuyển đổi 8 bit cần 256 bộ so sánh, còn với chuyển đổi chớp nhoáng 10 bit cần tới 1023 bộ so sánh ! Các bộ chuyển đổi chớp nhoáng đơn chip hiện có cho những ứng dụng đòi hỏi tốc độ cao, nhưng chúng tương đối đắt tiền. Các bộ biến đổi chớp nhoáng hiện nay có thể thực hiện một chuyển đổi 8 bit trong thời gian dưới 10 ns.

15.2.2. Các bộ biến đổi A/D hai sườn xung

Hình 15.4a trình bày sơ đồ khối chức năng của một bộ biến đổi A/D hai sườn xung. Các bộ biến đổi loại này được coi là trái tim của các đồng hồ đo hiện số vì chúng có thể cung cấp độ phân giải cao với giá thành hạ. Sau đây là nguyên tắc hoạt động. Khởi đầu, mạch điều khiển xoá tất cả các bộ đếm về 0 và nối lối vào của mạch tích phân với điện áp vào cần chuyển đổi. Giả sử điện áp vào là dương, khiến cho lối ra mạch tích phân lúc này âm dần như trên hình 15.4b. Ngay khi lối ra của mạch tích phân giảm xuống dưới 0 vài micrô vôn thì lối ra bộ so sánh sẽ chuyển lên mức cao. Lối ra bộ so sánh ở mức cao cho phép mở cổng AND đưa xung nhịp 1 MHz tới chuỗi các bộ đếm. Sau một số đếm nhất định, thường là 1000, mạch điều khiển sẽ chuyển lối vào bộ tích phân xuống mức điện áp chuẩn âm, đồng thời reset tất cả các bộ đếm về 0. Với điện áp lối vào âm, lối ra mạch tích phân sẽ tăng dần như vẽ bên phải hình 15.5b. Khi lối ra bộ tích phân vượt qua 0V, lối ra bộ so sánh chuyển xuống mức thấp, cấm xung nhịp chuyển vào các bộ đếm. Số đếm cần thiết để lối ra mạch tích phân quay về 0V tỉ lệ với điện áp vào. Đối với mạch điện trên hình 15.5a, ví dụ điện áp vào là +2V sẽ tạo ra số đếm tương ứng là 2000. Vì điện trở và tụ điện trên mạch tích phân được dùng cho cả tích phân điện áp vào lẫn tích phân điện áp chuẩn nên những biến đổi nhỏ trong giá trị của chúng do nhiệt độ không ảnh hưởng đến độ chính xác của quá trình biến đổi. Những bộ biến đổi A/D hoàn chỉnh kiểu sườn xung hiện có bán trên thị trường dưới dạng các IC đơn chip. Ví dụ vi mạch ICL 7136 của hãng Intersil bao gồm toàn bộ mạch điện của một bộ biến đổi A/D 3 1/2 digit và tất cả các mạch điện cần thiết để ghép nối nó với một bộ chỉ thị LCD 3 1/2 digit. Một ví dụ khác, vi mạch ICL 7135 của hãng Intersil bao gồm toàn bộ mạch điện của bộ biến đổi A/D 4 1/2 digit và một lối ra hợp kênh BCD. Chú ý rằng do các ứng dụng thông thường của các bộ biến đổi loại này mà chúng ta thường biểu diễn độ phân giải của chúng dưới dạng số các chữ số. Giá trị toàn thang đọc được của một bộ biến đổi 3 1/2 digit là 1999, do đó độ phân giải tương đương với 1 phần 2000. Các vi mạch lưỡng chip ICL 8068 và ICL 7104-16 chứa các mạch điện của bộ biến đổi A/D loại sườn xung 16 bit nhị phân lối ra.

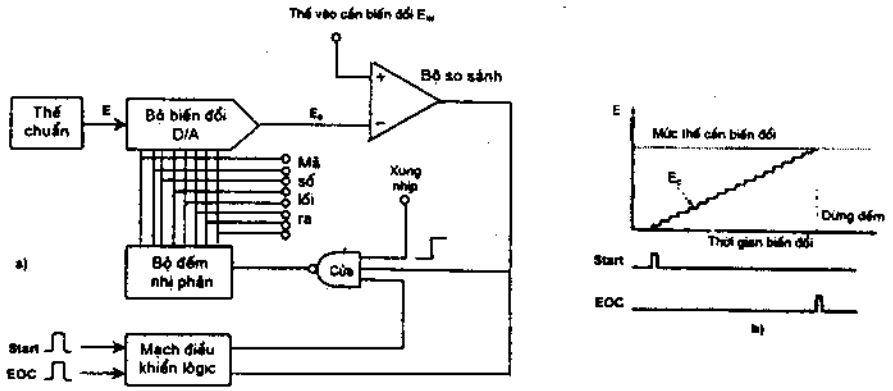


Hình 15.5 Bộ biến đổi A/D hai sườn xung.

Nhược điểm chính của các bộ biến đổi loại sườn xung là tốc độ chậm. Một bộ biến đổi 4 1/2 digit có thể phải mất tới 300 ms để thực hiện một chuyển đổi.

15.2.3. Bộ biến đổi A/D kiểu quét bậc thang

Sơ đồ của nó như hình 15.6 trong đó sử dụng một DAC cùng các bộ so sánh, bộ đếm nhị phân và mạch điều khiển lôgic cho xung khởi phát biến đổi START và xung báo kết thúc quá trình biến đổi EOC.

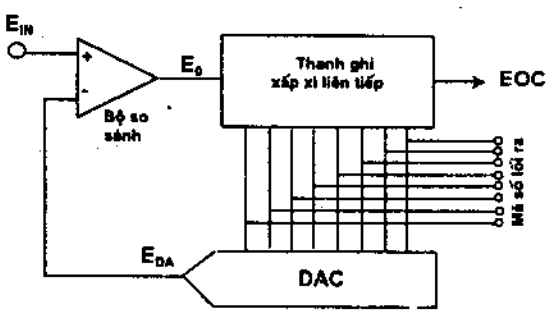


Hình15.6 A/D quét bậc thang.

Khi xung Start xuất hiện, bộ đếm nhị phân được xoá về 0 → lối ra của bộ biến đổi số-tương tự DA là $E_0 < E_{IN}$ → lối ra của bộ so sánh bằng 1 mở cổng cho các xung nhịp vào bộ đếm. Bộ đếm bắt đầu đếm với số đếm lối ra tăng dần từng bit một ứng với mỗi xung nhịp vào. Điều đó dẫn tới lối ra E_0 của bộ biến đổi DA tăng từng mức một theo kiểu bậc thang. Đến khi $E_0 >= E_{IN}$ bộ so sánh lật trạng thái về 0 → đóng mạch cửa, cấm các xung nhịp vào bộ đếm đồng thời kích lôgic điều khiển phát ra xung báo kết thúc quá trình biến đổi EOC. Giá trị không đổi ở lối ra bộ đếm (cũng chính là lối vào của bộ DA) là giá trị chuyển đổi được. Nhược điểm của phương pháp biến đổi này là thời gian biến đổi không cố định mà phụ thuộc vào độ lớn của điện thế vào E_{IN}. Thế này càng lớn, thời gian biến đổi càng dài.

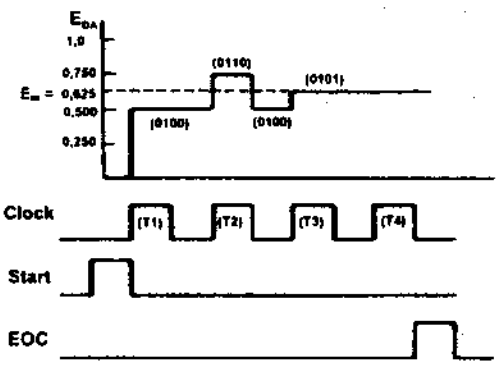
15.2.4. Bộ biến đổi A/D kiểu xấp xỉ liên tiếp

Một trong những sơ đồ loại này như hình 15.7 chỉ ra, sử dụng một thanh ghi đặc biệt gọi là thanh ghi xấp xỉ liên tiếp (XXLT), một bộ biến đổi D/A và một bộ so sánh. Phương pháp này sử dụng thuật toán chia đôi dây cung khi tìm nghiệm tiệm cận của một đường cong. Nếu ADC có N bit thì quá trình biến đổi xảy ra trong N bước. Bước thứ nhất, thanh ghi XXLT xuất ra số nhị phân có giá trị số bằng 1/2 toàn dải, nghĩa là số có bit nhiều ý nghĩa nhất MSB = "1", các bit còn lại bằng "0". Như vậy thế lối ra E_{DA} của DAC nhảy lên một mức điện thế bằng 1/2 thế toàn dải FS. Nếu $E_{DA} > E_{IN}$, lối ra bộ so sánh E_0 sẽ ở mức lôgic = "0" điều khiển thanh ghi XXLT xoá bit MSB và đặt bit tiếp theo của số xuất ra lên "1". Nếu thế

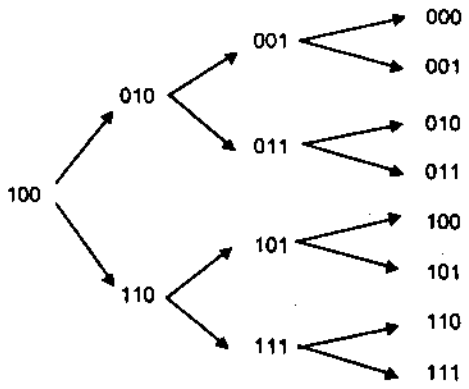


Hình 15.7 Bộ biến đổi A/D xấp xỉ liên tiếp.

này còn nhỏ hơn E_{IN}, thế lối ra của bộ so sánh E_0 = "1" sẽ điều khiển thanh ghi XXLT giữ nguyên giá trị MSB = "1" và đặt bit tiếp theo của số xuất ra lên "1". Quá trình cứ lặp lại cho đến N lần và giá trị cuối lần đó chính là giá trị chuyển đổi số mong muốn. Hình 15.8 là giản đồ thế lối ra E_0 của bộ biến đổi D/A phụ thuộc vào các nhịp biến đổi, ở đây điện thế toàn thang FS là 1V. Trong đó điện thế vào cần biến đổi E_{IN} gần mức 0,625V. Nếu liệt kê tất cả các trường hợp mà thanh ghi XXLT có thể xuất ra số nhị phân trong trường hợp với bộ biến đổi AD xấp xỉ liên tiếp 3 bit (N = 3) ta có các giá trị trong 8 trường hợp xảy ra như hình 15.9. Ngoài các phương pháp trên còn nhiều phương pháp biến đổi A/D khác như: ADC kiểu bám dùng bộ đếm tiến/lùi, ADC kiểu dùng điện áp răng cưa được tạo bởi quá trình nạp/phóng qua tụ, v.v...



Hình 15.8 Sự phụ thuộc của thế E_0 vào các nhịp biến đổi 3 bit.



Hình 15.9 8 giá trị có thể xảy ra khi biến đổi A/D 3 bit.

15.2.5. Các loại mã lối ra của bộ biến đổi A/D

Để được thuận tiện trong những ứng dụng khác nhau, các bộ biến đổi A/D thường có sẵn những mã lối ra khác nhau, thường gây nhầm lẫn cho người sử dụng. Cách tốt nhất để phân biệt chúng là xem xét toàn bộ một cách đồng thời theo các giá trị mà chúng biểu diễn như trên bảng sau. Các giá trị này là của chuyển đổi 8 bit nhưng có thể mở rộng chúng cho bất kì số bit nào.

Bảng 15.3 **Các mã nhị phân đơn cực**

Giá trị	10 V toàn thang	Nhị phân (BIN)	Bù nhị phân (CB)	Đảo nhị phân (IB)	Bù đảo nhị phân (ICB)
+FS - 1 LSB	9,9609	1111 1111	0000 0000		
+1/2 FS	5,0000	1000 0000	0111 1111		
+1/2Fs - 1 LSB	4,9609	0111 1111	1000 0000		
+ 1 LSB	0,0391	0000 0001	1111 1110		
Zero	0,0000	0000 0000	1111 1111	0000 0000	1111 1111
- 1 LSB	-0,0391			0000 0001	1111 1110
-1/2 FS + 1 LSB	-4,9609			0111 1111	1000 0000
-1/2 FS	-5,0000			1000 0000	0000 0001
- FS + 1 LSB	-9,9609			1111 1111	0000 0000

Các mã BCD đơn cực

Giá trị	10 V toàn thang	BCD	Bù bcd (CBCD)	Đảo bcd (IBCD)	Bù đảo bcd (ICBD)
+FS - 1 LSB	9,9	1001 1001	0110 0110		
+1/2 FS	5,0	0101 0000	1010 1111		
+1/2Fs - 1 LSB	0,1	0000 0001	1111 1110		
+ 1 LSB	0,0	0000 0000	1111 1111		
Zero	-0,1			0000 0001	1111 1110
- 1 LSB	-5,0			0101 0000	1010 1111
-1/2 FS + 1 LSB	-9,9			1001 1001	0110 0110
-1/2 FS					
- FS + 1 LSB					

Các mã nhị phân lưỡng cực

Giá trị	10 V toàn thang	Nhị phân dịch (OB)	Bù nhị phân dịch (COB)	Bù 2 (TC)
+ FS	5,0000			
+FS - 1 LSB	4,9609	1111 1111	0000 0000	0111 1111
+ 1 LSB	0,0391	1000 0001	0111 1110	0000 0001
Zero	0,0000	1000 0000	0111 1111	0000 0000
- 1 LSB	-0,0391	0111 1111	1000 0000	1111 1111
- FS + 1 LSB	-4,9609	0000 0001	1111 1110	1000 0001
- FS	-5,0000	0000 0000	1111 1111	1000 0000

Các bộ biến đổi A/D có dải điện áp vào dương (*đơn cực*) thường sử dụng mã nhị phân thuận (straight binary code) hoặc mã nhị phân đảo (inverted binary code). Nếu các lối ra của một bộ biến đổi A/D sẽ được ghép với một màn hiển thị, sẽ thuận tiện hơn nếu có mã lối ra BCD. Còn với các ứng dụng mà dải điện áp vào bao gồm cả giá trị âm lẫn dương (*lưỡng cực*) người ta thường sử dụng mã nhị phân dịch. Có thể thấy trên các bảng các giá trị từ 00000000 cho tới 11111111 đều bị dịch xuống dưới, do đó 00000000 biểu diễn giá trị âm nhất còn 10000000 biểu diễn giá trị zero. Sơ đồ mã hoá này có ưu điểm là các mã bù 2 được tạo ra một cách đơn giản: chỉ việc đảo bit MSB. Một số bộ biến đổi xuất ra các giá trị số dưới dạng mã bù 2 một cách trực tiếp.

15.2.6. Ghép nối các bộ biến đổi A/D với máy tính

Trong bất kì ứng dụng nào có bộ biến đổi A/D sử dụng các bộ so sánh song song, bộ biến đổi dường như tạo lập các giá trị lối ra nhanh hơn khả năng đọc chúng của các máy tính nhiều lần. Do vậy, cần phải dùng phương pháp *truy cập bộ nhớ trực tiếp* DMA. Khi vi xử lý thả nổi bus, chip điều khiển DMAC sẽ nắm quyền điều khiển bus và cho dữ liệu từ bộ biến đổi A/D truyền trực tiếp vào các ô nhớ liên tiếp.

Phần lớn các bộ biến đổi A/D loại sườn xung sẵn có trên thị trường được thiết kế để ghép với các bộ hiển thị 7 đoạn như trong đồng hồ đo chỉ thị số. Do đó, chúng thường xuất dữ liệu dưới dạng mã BCD hợp kênh hoặc mã 7 đoạn. Để đọc dữ liệu từ bộ biến đổi này thường dùng phương pháp hỏi vòng. Các bộ đếm tần số, đồng hồ hiện số và các thiết bị kiểm tra khác thường có sẵn các lối ra BCD hợp kênh ở mặt sau của chúng.

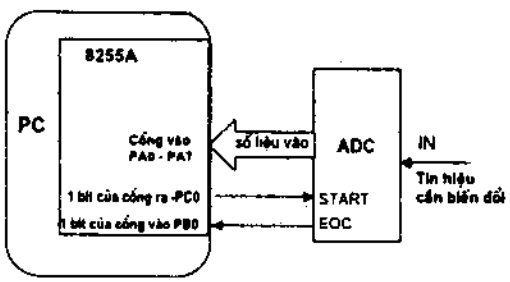
Các bộ biến đổi A/D xấp xỉ liên tiếp thường có các lối ra cho từng bit một. Mã xuất ra trên các đường này thường là loại mã nhị phân thuận hoặc mã nhị phân dịch. Có thể nối các lối ra

song song của bộ biến đổi với số chân yêu cầu của một cổng vào và đọc số liệu bằng một chương trình phần mềm. Ngoài các đường dữ liệu, có hai đường tín hiệu khác của bộ biến đổi xấp xỉ liên tiếp cần phải nối với máy tính. Đầu tiên là tín hiệu START CONVERT (khởi phát) được xuất từ máy tính đến bộ biến đổi để báo cho nó thực hiện một quá trình chuyển đổi AD. Tín hiệu thứ hai là EOC (kết thúc) phát ra từ bộ biến đổi báo hiệu quá trình biến đổi đã hoàn thành và dữ liệu trên các lối ra đã ổn định. Sau đây là các bước của chương trình dùng để nhận số liệu của một mẫu từ các bộ biến đổi loại này.

Đầu tiên, phát một xung lên lối vào START CONVERT trong một khoảng thời gian do bộ biến đổi yêu cầu. Sau đó, kiểm tra xem tín hiệu EOC đã hạ xuống mức thấp hay chưa bằng phương pháp hỏi vòng hoặc phương pháp ngắt. Tiếp theo, đọc các giá trị đã được số hoá từ các lối ra song song của bộ biến đổi.

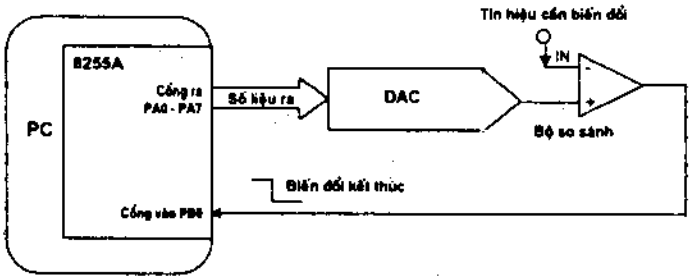
Trong trường hợp có sẵn một vi mạch biến đổi A/D rồi thì vấn đề chỉ là dùng các cổng vào/ra của PC tạo ra các xung START, ghi nhận các xung EOC và thu thập các dữ liệu. Trường hợp khác là có thể xây dựng hệ biến đổi AD với độ phân giải N bit trên cơ sở một vi mạch biến đổi DAC có N bit và một phần cứng tối thiểu gồm các thanh ghi trên các cổng vào/ra.

Sơ đồ ghép nối bộ biến đổi ADC là một vi mạch tích hợp đơn khối với các cổng vào/ra của 8255A như hình 15.10 sau. Sự lựa chọn các bit thu thập số liệu và điều khiển là tuỳ ý, trên hình chỉ lấy ra làm thí dụ.



Hình 15.10 Ghép nối ADC với 8255A.

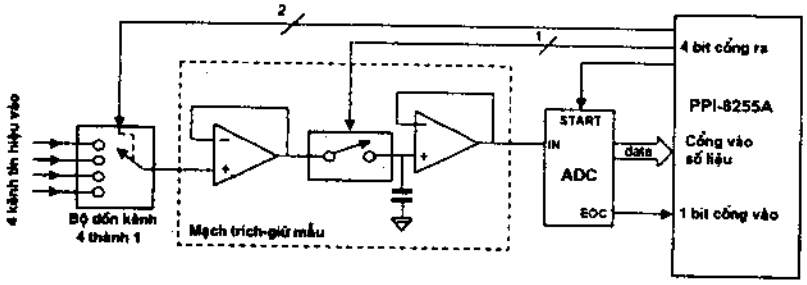
Trong trường hợp phải xây dựng một bộ biến đổi tương tự - số A/D trên cơ sở một DAC ta có sơ đồ khối hình 15.11 sau. Có thể chọn cổng ra A ở chế độ 0, mode ra. Vì cổng luôn ở mode ra trong suốt quá trình hoạt động nên số liệu được PC viết vào đó sẽ tồn tại mãi cho tới khi một số liệu mới được viết tới cổng. Chương trình sẽ viết số liệu ra cổng tuỳ thuộc vào quy luật của các phương pháp biến đổi (bậc thang hay xấp xỉ liên tiếp) và trạng thái ra của bộ so sánh được đọc vào 1 bit trên một cổng vào (thí dụ PB0). Giá trị viết ra cuối cùng chính là giá trị cần biến đổi.



Hình 15.11 Xây dựng bộ biến đổi A/D bằng chip D/A với phần cứng tối thiểu.

15.2.7. Điều khiển bộ dồn kênh analog và mạch trích - giữ mẫu

Để đảm bảo mức tín hiệu analog là không đổi trong quá trình biến đổi A/D thì tín hiệu này phải được cho qua bộ trich và giữa mẫu (Sample & hold). Nhiều kênh đo lối vào analog cũng có thể được dồn kênh cho một bộ biến đổi A/D. Các bit trên cổng ra của 8255A cũng có thể được dùng để điều khiển bộ dồn kênh tương tự (analog multiplexor) và mạch trích-giữ mẫu tín hiệu. Một bộ dồn kênh là một chuyển mạch điện tử nhiều đầu vào và một đầu ra cho phép tạo được một bộ biến đổi AD nhiều kênh, thí dụ 4 kênh như hình 15.12 sau.



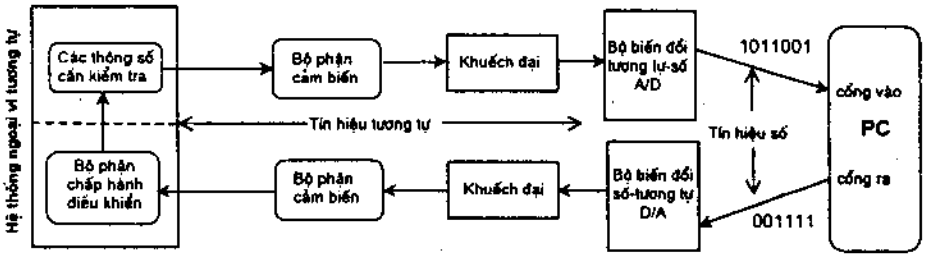
Hình 15.12 Điều khiển dồn kênh và trích giữ mẫu tín hiệu.

Trong sơ đồ trên, 8255A dùng 2 bit của cổng ra để điều khiển chọn 1 trong 4 kênh vào. 1 bit cổng ra để điều khiển công tắc điện tử cho mạch trích (đóng công tắc) và giữ mẫu (ngắt công tắc). 1 bit ra tạo xung khởi phát biến đổi A/D và 1 bit vào để phát hiện thời điểm kết thúc quá trình biến đổi EOC.

CHƯƠNG 16. ĐO LƯỜNG VÀ ĐIỀU KHIỂN TỰ ĐỘNG BẰNG KĨ THUẬT GHÉP NỐI MÁY VI TÍNH

Một lĩnh vực mà ở đó các máy vi tính được sử dụng rộng rãi là các quá trình *đo lường, điều khiển tự động trong công nghiệp và trong các phòng thí nghiệm.*

Trong kĩ thuật ghép nối máy vi tính với hệ thống đo lường- điều khiển của các dây chuyền sản xuất công nghiệp hay trong các thực nghiệm vật lí, hoá học, sinh học v.v..., các tín hiệu điện tỉ lệ với các thông số phi điện cần đo do các cảm biến (sensor) sinh ra phải được biến đổi như thế nào đó cho tương thích được với các yêu cầu về tín hiệu lối vào số của máy vi tính. Sơ đồ tổng quát của một hệ thống thu thập và xử lí tín hiệu ghép nối vi tính như hình 16.1. Biên độ lối ra của các tín hiệu trên các cảm biến thường là nhỏ (thí dụ, thế nhiệt điện trên các cặp nhiệt điện đo nhiệt độ chỉ cỡ mili vôn) nên chúng phải được khuếch đại lên trước khi được đưa vào bộ biến đổi A/D. Các tín hiệu số lối ra được đưa vào máy vi tính để xử lí. Các kết quả điều khiển sẽ được xuất ra các cổng vào/ra rồi được đưa vào bộ biến đổi số-tương tự D/A . Bộ này có nhiệm vụ ngược lại với bộ A/D là biến đổi các mã số ở lối vào từ máy tính thành điện thế hoặc dòng điện tương tự ở lối ra. Các tín hiệu tương tự đó được đưa trở lại hệ thống nhằm điều khiển các quá trình một cách tự động.



Hình 16.1 Một hệ thống đo lường và điều khiển tự động ghép nối máy tính.

16.1. HỆ ĐIỀU KHIỂN TỰ ĐỘNG

16.1.1. Các thông số của một hệ đo và điều khiển

Các thông số cần đo của một hệ cần điều khiển được xem xét theo các tiêu chuẩn sau:

1. *Độ nhạy* (sensitivity) là sự thay đổi giá trị cực tiểu còn phát hiện được so với giá trị trung tâm của đại lượng cần đo. Nó được quyết định bởi tỷ số tín hiệu trên ồn của thiết bị. Thường được coi là bằng hai lần giá trị bình phương trung bình của ồn. Theo cách như vậy, một thông số là *dải chết* (dead band) của một thiết bị được định nghĩa là dải biến đổi của tín hiệu vào mà không gây nên một sự thay đổi nào giá trị tín hiệu ra.
2. *Độ chính xác* (accuracy) là sai lệch nhỏ nhất của các giá trị đo được so với một giá trị chuẩn được coi là giá trị đúng.
3. *Độ lặp lại* (reproducibility hay precision) khác với độ chính xác, một thiết bị được chuẩn độ kém có thể thiếu chính xác nhưng những kết quả không chính xác này vẫn có tính lặp lại cao. Do đó một thiết bị đo cần có cả độ chính xác và lặp lại cao nhưng thiết bị điều khiển chỉ cần có độ lặp lại cao là đủ. Liên quan đến độ chính xác là đơn vị độ lệch chuẩn (standard deviation)

$$S \equiv \sqrt{\frac{\sum(\overline{X} - X_i)}{N-1}}$$

trong đó $\overline{X}$ là giá trị trung bình của các phép đo X_T

N là số lần đo.

4. *Độ chọn lọc* (Selectivity) là khả năng phân biệt giữa các giá trị cần quan tâm và các can nhiễu có thể có.
5. *Dải đo* là khoảng giá trị trong đó phép đo định lượng có thể thực hiện được.
6. *Dải động* (dinamic range) của một thiết bị đo là tỷ số giữa độ rộng dải đo trên độ nhạy của phép đo.
7. *Độ trôi* (drift), một khía cạnh rất quan trọng về chỉ tiêu của các hệ đo trong công nghiệp và phòng thí nghiệm là thiên hướng "trôi" của các giá trị đo có phổ rất gần thành phần 1 chiều DC. Thí dụ như điểm zêro có thể thay đổi từ từ do quá trình biến đổi nhiệt độ phòng hoặc sự già hoá của thiết bị v.v...
8. *Độ phân giải* (resolution) là độ chính xác mà phép đo có thể hiển thị được. Thí dụ, kích thước nhỏ nhất của một màu mắt người còn có thể phân biệt được, số digit được dùng để hiển thị kết quả đo số: 999 có độ phân giải 3 digit, (1/0)999 có độ phân giải 31/2 digit ...
9. *Độ trễ* (delay): Trong các thiết bị điện tử luôn có một sự trì trễ giữa đại lượng biến đổi cần đo và giá trị đo được. Thường định nghĩa *thời gian phản ứng* (responsive time) là thời gian để thiết bị đạt được số % giá trị xác định nào đó (thí dụ, 98%) trên toàn bộ dải biến đổi. *Thời gian tăng* (raising time) là thời gian đòi hỏi để tín hiệu lối ra của thiết bị thay đổi từ giá trị này (thí dụ 10%) đến giá trị khác (thí dụ 90%) trong dải biến đổi của tín hiệu vào. *Hằng số thời gian* (time constant) là thời gian để giá trị lối ra đạt được từ 0% đến 63% giá trị toàn dải.

16.1.2. Các cảm biến và mạch thích ứng tín hiệu trong hệ điều khiển

Nhìn trong sơ đồ 16.1 ta thấy các các cảm biến (sensor) giữ vai trò quan trọng trong một hệ đo và điều khiển. Nói chung, đó là những dụng cụ dựa trên các hiệu ứng vật lý, hoá học ... cho phép biến đổi các đại lượng phi điện ở lối vào của nó thành các đại lượng điện ở lối ra. Các thông số kể trên của hệ đo (như độ nhạy, độ phân giải, tính lặp lại, độ chính xác v.v...) phụ thuộc vào các cảm biến. Ta liệt kê sơ bộ một vài loại cảm biến phổ biến trong thực tế như sau:

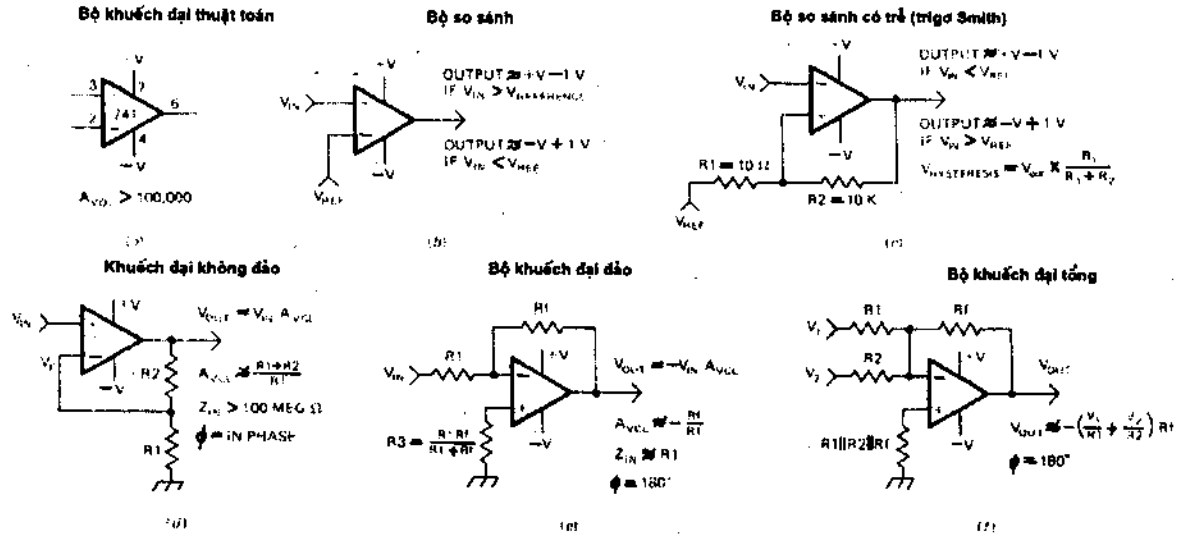
1. Các cảm biến đo độ dịch chuyển:
   - Điểm dịch chuyển cần đo được gắn cố kết với vị trí con chạy trên một biến trở. Giá trị biến đổi điện trở của nhánh tỷ lệ với độ dịch chuyển.
   - Cảm biến sức căng (train gauss) khi được gắn cố kết với đối tượng cần đo độ dịch chuyển sẽ có giá trị điện trở phụ thuộc vào sức căng và do vậy phụ thuộc vào độ dịch chuyển (thí dụ độ dãn nở) của đối tượng cần đo. Mạch cầu Whestone là một thí dụ rất tốt về một cảm biến cho phép đo những thay đổi nhỏ về điện trở trong đó cảm biến điện trở được gắn trên một nhánh của cầu.
   - Các cảm biến đo độ dịch chuyển bằng các cuộn điện cảm dùng nguyên tắc tự cảm, hỗ cảm và biến thế vi sai. Các thông số ghép giữa các cuộn dây này sẽ bị biến đổi bởi sự chuyển dịch của một vật có độ từ thẩm cao đặt giữa chúng. Vậy bằng cách xác định giá trị điện phụ thuộc vào thông số ghép ta có thể xác định được độ dịch chuyển của vật.
   - Các cảm biến đo độ dịch chuyển bằng phương pháp điện dung: điện dung giữa 2 bản cực song song của một tụ điện tỉ lệ với khoảng cách giữa chúng. Vậy nếu cố kết vật cần đo độ

dịch chuyển với một bản cực thì bằng cách đo giá trị biến đổi của điện dung ta có thể biết được độ dịch chuyển của vật.

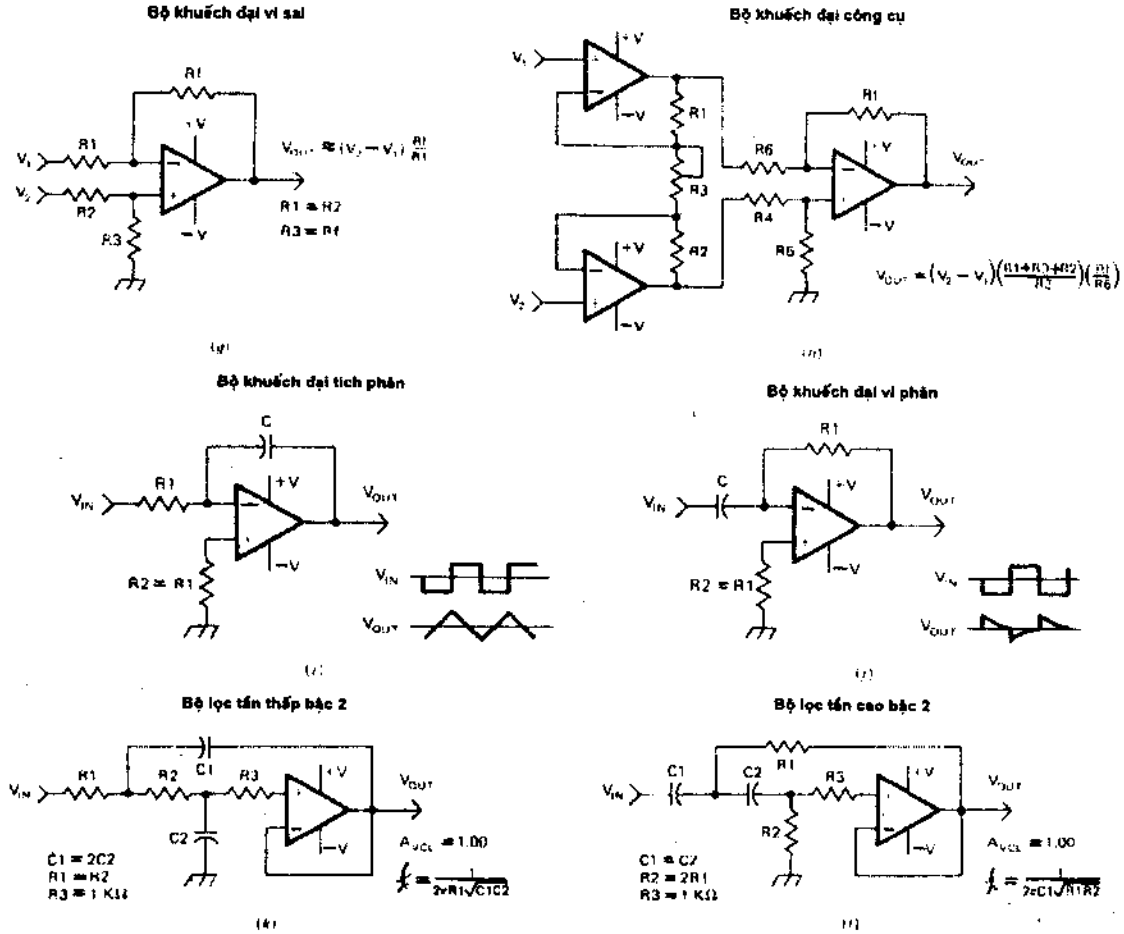
- Các cảm biến dựa trên hiệu ứng áp điện thuận và nghịch cho phép đo được sự dịch chuyển cũng như lực tác dụng lên vật được gắn cố kết với cảm biến.

2. Các cảm biến đo nhiệt độ.
   - Cặp nhiệt điện sử dụng hiệu ứng Seebeck tồn tại một sức điện động ở điểm tiếp giáp giữa 2 kim loại, sức điện động nhiệt điện này tỉ lệ với nhiệt độ tại điểm đó.
   - Nhiệt điện trở thuộc loại này có nhiệt điện trở kim loại, nhiệt trở bán dẫn v.v... trong đó giá trị điện trở của cảm biến tỉ lệ với nhiệt độ của nó.
3. Các cảm biến nhạy ánh sáng.
   - Quang trở: là vật liệu có điện trở phụ thuộc vào quang thông chiếu lên nó.
   - Photodiode là loại diode có dòng ngược tăng tỷ lệ với quang thông chiếu lên tiếp giáp phân cực ngược của nó.
   - Phototransistor là loại transistor có dòng collector tăng theo quang thông chiếu vào các lớp tiếp giáp p-n của nó.
4. Các cảm biến điện hoá.
   - Loại tế bào gavalnic biến đổi năng lượng điện hoá thành năng lượng điện khi hai điện cực điện hoá của nó được nối với nhau bằng một dây dẫn.
   - Loại tế bào điện phân (electrolytic) phản ứng hoá học được xảy ra bởi một nguồn điện thế bên ngoài nối tới hai điện cực khi điện thế này lớn hơn điện thế gavalnic của tế bào.

Thường thì tín hiệu analog ở lối ra các cảm biến này không đủ lớn hoặc không có dạng cần thiết nên phải cho qua các bộ khuếch đại analog hoặc bộ thích ứng tín hiệu (signal conditioning), sau đó mới cho qua mạch trích và giữ mẫu và bộ chuyển đổi A/D. Hình 16.2 là tóm tắt một số các sơ đồ thông dụng dùng các bộ khuếch đại thuật toán cho các xử lí analog đó. Đặc điểm của các mạch này đã được trình bày kĩ trong các giáo trình về điện tử analog và số, tên của chúng cũng như tóm tắt các thông số (như hệ số khuếch đại) cũng được cho trên hình.



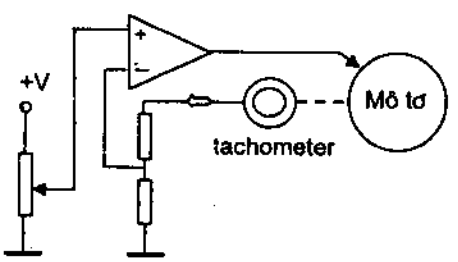
Hình 16.2 Các sơ đồ khuếch đại thuật toán (a, b, c, d, e, f).



Hình 16.2 Các sơ đồ khuếch đại thuật toán (g, h, i, j, k, l).

16.1.3. Hệ điều khiển tự động

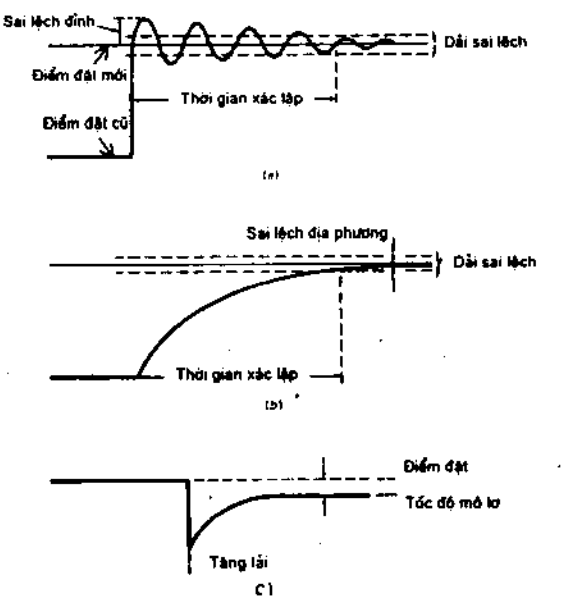
Điều khiển quá trình trước hết liên quan đến các hệ đo những đại lượng biến thiên như tốc độ của mô tơ, nhiệt độ, lưu lượng các chất phản ứng, mức chất lỏng trong bồn chứa, bề dày của mẫu vật liệu, v.v... Sau đó, tín hiệu lối ra của bộ điều khiển hệ thống sẽ điều chỉnh giá trị của mỗi biến cho tới khi bằng một giá trị đã được xác định trước, giá trị đó gọi là *điểm đặt* (set point). Bộ điều khiển hệ thống phải duy trì mỗi biến số càng gần điểm đặt của nó càng tốt và sẽ phải bù lại một cách nhanh nhất và chính xác nhất những thay đổi đối với biến số, thí dụ do tải của môtơ tăng lên. Một thí dụ đơn giản sẽ trình bày cách tiếp cận cổ điển để điều khiển một quá trình biến thiên và giải thích một số thuật ngữ được sử dụng trong các hệ điều khiển. Mạch điện trên hình 16.3 trình bày phương pháp tương tự đối với quá trình điều khiển tốc độ của một mô tơ điện một chiều. Gắn với trục của mô tơ là một máy phát điện một chiều gọi là *máy đo vận tốc góc (tachometer)*, phát ra điện áp tỉ lệ với tốc độ của mô tơ. Điện áp lối ra thông thường cỡ một vài vôn khi tốc độ



Hình 16.3 Mạch điều khiển tốc độ của mô tơ một chiều sử dụng máy đo vận tốc.

đạt tới 1000 vòng/phút. Một phần điện áp ra của máy đo vận tốc góc được đưa trở lại lối vào đảo của bộ khuếch đại công suất điều khiển mô tơ. Điểm đặt là một điện áp dương được đưa tới lối vào không đảo của bộ khuếch đại. Khi cấp nguồn cho mô tơ, nó sẽ tăng tốc cho tới khi điện áp phản hồi từ máy đo vận tốc góc về lối vào đảo gần bằng điện áp chuẩn. Nếu tải trong mô tơ tăng lên, vận tốc của nó sẽ bắt đầu giảm xuống khiến cho điện áp ra trên máy đo vận tốc góc giảm đi, làm tăng điện áp vi sai giữa các lối vào của bộ khuếch đại dẫn đến việc dòng điều khiển mô tơ được tăng lên. Dòng điều khiển tăng sẽ tăng tốc độ mô tơ lên giá trị gần bằng mức trước khi tải tăng. Phản ứng tương tự cũng sẽ xảy ra nếu tải trong mô tơ giảm đi.

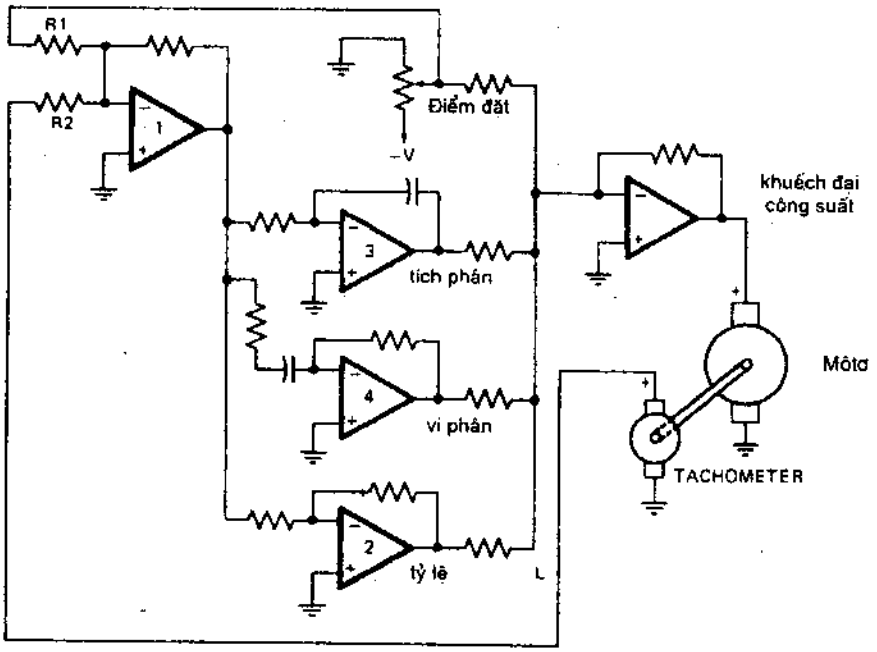
Việc sử dụng cơ chế phản hồi âm để điều khiển một hệ thống như trên thường được gọi là *điều khiển trợ động* (servo control). Vòng điều khiển kiểu này sẽ giữ cho tốc độ mô tơ khá ổn định đối với những ứng dụng mà tải trong mô tơ không thay đổi quá nhiều. Một số mô tơ điều khiển ổ cứng và các máy nghe nhạc quay đĩa chất lượng cao sử dụng phương pháp này để điều khiển tốc độ. Đối với những ứng dụng mà tải hoặc/và điểm đặt thay đổi đáng kể, ta sẽ gặp phải một số rắc rối. Đầu tiên là hiện tượng vượt mức khi điều chỉnh điểm đặt. Hình 16.4a trình bày một ví dụ về hiện tượng đó. Lúc này biến điều khiển - là tốc độ mô tơ - vượt qua điểm đặt mới và dao động lên xuống một lúc. Thời gian dao động trước khi nằm trong dải sai số cho phép được gọi là thời gian xác lập. Đáp ứng loại này được gọi là *thiếu giảm chấn* giống như một ô tô có bộ phận giảm xóc kém khi gặp ổ gà. Sự dao động có thể tránh được nếu ta thêm yếu tố hãm vào hệ. Tuy nhiên, nếu thêm vào quá nhiều yếu tố hãm thì đáp ứng của hệ sẽ có dạng như hình 16.4b. Đáp ứng loại này được gọi là *thừa giảm chấn*. Đáp ứng kiểu này khiến cho biến điều khiển phải rất lâu sau mới đạt tới giá trị điểm đặt mới. Để có được chất lượng tốt nhất, yếu tố hãm phải được thiết kế phù hợp với từng hệ thống cụ thể.



Hình 16.4 Đáp ứng của hệ điều khiển môtơ.

Một vấn đề khác nảy sinh trong bất kì hệ thống điều khiển quá trình nào là *sai số dư*. Hình 16.4c vẽ đáp ứng của một hệ điều khiển giống như bộ điều khiển tốc độ mô tơ trong hình 16.3 khi tải trong mô tơ tăng lên. Ban đầu, tốc độ mô tơ hạ xuống nên điện áp ra của máy đo vận tốc góc bị giảm đi. Như đã nói trước đây, điều này làm tăng điện áp vi sai giữa các lối vào của bộ khuếch đại khiến cho điện áp ra bộ khuếch đại tăng lên. Điện áp này tăng sẽ làm tăng tốc độ mô tơ và do đó tăng cả điện áp ra máy đo vận tốc góc. Tuy nhiên, khi hệ đạt tới trạng thái cân bằng, vẫn còn sự sai lệch đáng kể giữa điểm đặt và điện áp phản hồi từ máy đo vận tốc góc. Chính độ lệch hay sai số dư này sẽ được khuếch đại bởi bộ khuếch đại để tạo ra tín hiệu điều khiển bổ sung cho mô tơ. Vì các lí do ổn định, hệ số khuếch đại của các hệ điều khiển thường không quá cao. Do vậy, ngay cả khi đã điều chỉnh được tốc độ của mô tơ về chính xác giá trị cho trước đối với một tải, khi thay đổi tải vẫn sẽ bắt gặp sai số dư giữa điểm đặt và giá trị ra thực tế. Để giải quyết vấn đề này, người ta phải sử dụng các mạch phản

hồi phức tạp hơn. Hình 16.5 trình bày một mạch điện đại diện cho rất nhiều loại phản hồi thường được sử dụng.



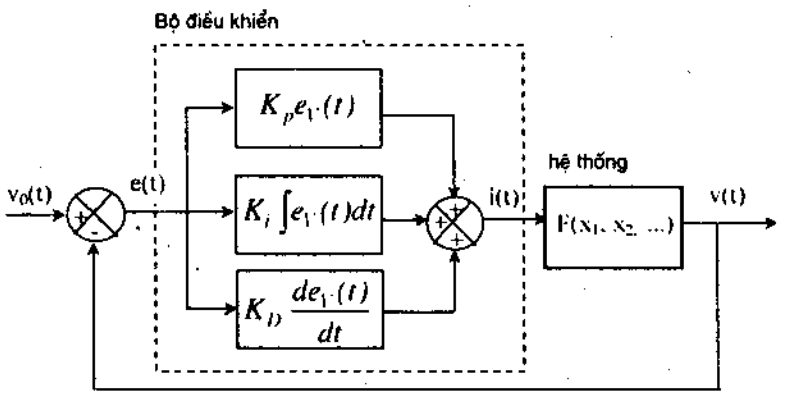
Hình 16.5 Mạch điều khiển sử dụng phản hồi phức tạp.

Trước hết, chú ý rằng điện áp ra của bộ khuếch đại là tổng của 4 điện áp vào khác nhau. Dòng điện đi tới điểm lấy tổng của bộ cộng từ điểm đặt tạo nên dòng điều khiển nền. Ba lối vào còn lại không cấp dòng trừ khi có sự sai lệch giữa điểm đặt và điện áp phản hồi từ máy đo vận tốc góc. Bộ khuếch đại số 1 là một bộ cộng khác có chức năng so sánh điện áp điểm đặt với điện áp phản hồi từ máy đo vận tốc góc. Giả sử hai điện trở vào R1 và R2 bằng nhau. Vì điện áp điểm đặt là âm còn điện áp phản hồi từ máy đo vận tốc góc lại dương nên sẽ không có dòng điện chạy qua điện trở phản hồi của khuếch đại thuật toán nếu hai điện áp này bằng nhau về độ lớn. Nói cách khác, nếu tốc độ của môtơ bằng giá trị ứng với điểm đặt thì điện áp ra bộ khuếch đại thuật toán số 1 sẽ bằng không, dẫn đến các bộ khuếch đại thuật toán số 2, 3 và 4 không cấp dòng cho điểm lấy tổng của bộ khuếch đại công suất. Bây giờ nếu tăng tải của môtơ lên, khiến cho nó chạy chậm lại. Điện áp ra từ máy đo vận tốc góc sẽ không còn bằng điểm đặt nữa, nên khuếch đại thuật toán số 1 có điện áp trên lối ra. Tín hiệu sai số này sẽ tạo ra 3 loại phản hồi khác nhau lên điểm lấy tổng của bộ khuếch đại công suất. Khuếch đại thuật toán số 1 đơn giản chỉ tạo ra tín hiệu phản hồi tỉ lệ với sai số giữa điểm đặt và lối ra máy đo vận tốc góc, giống hệt vai trò của bộ chia điện thế trong máy đo vận tốc góc ở hình 16.3. *Phản hồi tỉ lệ*, như nó vẫn được gọi, sửa được hầu hết các hiệu ứng của sự tăng tải nhưng như ta đã đề cập trước đây sẽ duy trì một sai số dư. Giải pháp để triệt sai số dư là sử dụng một mạch ***phản hồi tích phân***. Khuếch đại thuật toán số 3 trên hình 16.4 cung cấp loại phản hồi này. Hãy nhớ lại mạch điện này sẽ tạo ra điện áp tăng dần mỗi khi đưa một điện áp đến lối vào của nó. Trong thí dụ này, điện áp ra mạch tích phân sẽ tăng hoặc giảm mỗi khi có tín hiệu sai số trên lối vào của nó. Bằng cách tăng hoặc giảm chỉ một chút xung quanh điểm chuẩn, bộ tích phân có thể loại bỏ

được phần lớn sai số dư. Tuy nhiên, nếu có quá nhiều phản hồi tích phân sẽ khiến cho điện áp ra dao động lên xuống. Ngoài ra, tín hiệu phản hồi chỉ ảnh hưởng rất chậm đến điện áp ra vì điện áp sai số phải tồn tại một lúc trước khi bộ tích phân có được tín hiệu ra đáng kể. Để cải thiện đáp ứng của hệ thống, khuếch đại thuật toán số 4 trên hình 16.5 cung cấp loại phản hồi thứ ba gọi là *phản hồi vi phân*. Phản hồi vi phân là tín hiệu tỉ lệ với tốc độ biến đổi của sai số. Nếu tải của hệ bất ngờ thay đổi, mạch khuếch đại vi phân sẽ nhanh chóng phát ra một xung phản hồi để sửa sự biến đổi này. Khi tín hiệu sai số lần đầu được đưa tới mạch vi phân, tụ điện nối tiếp với lối vào không được tích điện từ trước sẽ hành động như bị đoản mạch. Sự tác động này cho phép một dòng điện lớn chạy qua, do đó khuếch đại thuật toán sẽ có được một điện áp ra đáng kể. Khi tụ nạp điện, cường độ dòng điện giảm đi khiến cho phản hồi từ mạch vi phân cũng giảm xuống. Mạch vi phân chủ yếu chỉ cấp cho bộ khuếch đại một xung phản hồi nhanh nhằm sửa sự tăng lên đột ngột của tải. Quá nhiều phản hồi vi phân có thể gây ra hiện tượng vượt mức và dao động.

Điểm mấu chốt ở đây là bằng cách sử dụng kết hợp một số hoặc tất cả các loại phản hồi mà hệ điều khiển phản hồi đã cho sẽ có được đáp ứng tối ưu đối với các thay đổi trong tải cũng như trong điểm đặt. Các vòng điều khiển quá trình sử dụng cả 3 loại phản hồi này được gọi là *vòng điều khiển vi tích phân tỉ lệ* PID (propotional integral derivative control loop).

Tóm lại, ngoài kiểu điều khiển đơn giản hai trạng thái đóng/ngắt (ON/OFF) trong đó bất kì một sai lệch nào của giá trị đo khỏi điểm đặt sẽ điều khiển thiết bị chấp hành hoặc đóng van cấp năng lượng cho tải hoàn toàn hoặc tắt van hoàn toàn; còn có loại điều khiển phức tạp là PID. Sơ đồ tổng quát của một hệ điều khiển như vậy được cho trong hình 16.6. Ở đây hệ thống (gồm thiết bị chấp hành, tải, môi trường v.v...) phụ thuộc vào các thông số x_1, x_2, ... được đặc trưng bởi hàm $F(x_1, x_2, ...)$. Về nguyên tắc các thông số này là khó xác định chính xác. Hệ có lối vào là đại lượng $i(t)$ và lối ra là biến cần điều khiển $v(t)$. Hệ được mắc nối tiếp với bộ điều khiển. Tín hiệu lối ra $v(t)$ được so sánh với điểm đặt $v_0(t)$ và sai lệch của chúng là $e(t)$ sẽ là tín hiệu vào bộ điều khiển. Biến lối ra bộ điều khiển là $i(t)$.



Hình 16.6 Sơ đồ một hệ điều khiển PID.

Mối quan hệ giữa lối ra bộ điều khiển và lối vào giống như đã thí dụ trong trường hợp trên có thể được viết như sau:

$$i(t) = K_p e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \qquad (16.1)$$

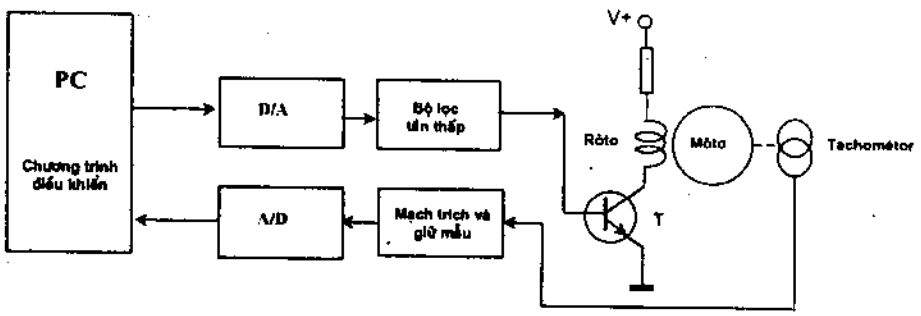
K_p là hệ số điều khiển tỉ lệ. K_I là hệ số điều khiển tích phân được đưa vào để triệt sai lệch dư, việc chọn nó phải phù hợp để tránh hoặc là biến điều khiển ra bị dao động hoặc là rất lâu mới đạt tới sai lệch dư bằng 0. K_D là hệ số điều khiển vi phân cho phép làm nhụt các dao động của hệ do đột biến của các thông số.

16.2. ĐIỀU KHIỂN BẰNG GHÉP NỐI MÁY TÍNH

Thường sự biến đổi theo thời gian của các thông số hệ thống cần điều khiển chậm hơn tốc độ tính toán của PC nhiều nên một máy tính với vài mạch điện vào ra đơn giản có thể thực hiện tất cả các chức năng của mạch tương tự trên hình 16.5 đối với các vòng điều khiển PID. Việc biến đổi thông số của các mạch điện tử loại tỉ lệ với vi và tích phân theo thời gian lúc này có thể thay thế bằng các thông số trong phương trình điều khiển bởi vì các giá trị điện áp và dòng điện tương tự đã được bộ biến đổi A/D đổi ra thành các đại lượng số, mà các con số thì dễ gia công, xử lí hơn việc thay đổi các linh kiện điện tử trong mạch điện nhiều.

Các hệ thống điều khiển quá trình trên cơ sở máy vi tính bao gồm từ các bộ điều khiển khả trình nhỏ như trên hình 16.5 có thể dùng để điều khiển một cỗ máy trong nhà máy cho đến những máy tính mini cỡ lớn dùng để điều khiển toàn bộ một nhà máy tự động.

Với thí dụ trên ta có thể vẽ ra sơ đồ khối của hệ điều khiển môtơ ghép nối với máy vi tính như hình 16.7 sau. Dưới sự điều khiển của máy vi tính bằng các mã số ra, tín hiệu ra analog của bộ biến đổi D/A tỉ lệ với mã số này sẽ điều khiển dòng góp của transistor T và là dòng qua cuộn dây rôto của động cơ. Với 1 tải nào đó, dòng càng lớn môtơ quay càng nhanh. Tốc độ của môtơ được đo bằng tachomét, thế lối ra của nó (tỉ lệ với tốc độ môtơ) được trích-giữ mẫu và cho qua bộ biến đổi A/D thành tín hiệu số đưa vào máy tính. Sai lệch của điện thế này so với điện thế đặt (tương ứng với một tốc độ cần thiết) trong chương trình sẽ được đưa vào tính toán và cho ra các mã số phù hợp với các loại điều khiển PID mong muốn. Với sơ đồ như vậy ta chỉ việc lập trình và điều chỉnh các thông số tỉ lệ, tích phân, vi phân trong chương trình là có thể đạt được một tốc độ quay mong muốn.



Hình 16.7 Điều khiển tốc độ môtơ bằng mạch ghép nối máy tính.

Những thảo luận về quá trình điều khiển của chúng ta đến đây mới chỉ dừng ở hệ điều khiển và vòng phản hồi với các tín hiệu analog. Trong khi đó trên hình 16.7 của hệ điều khiển ghép nối vi tính thì tín hiệu phản hồi sau khi qua mạch trích giữ mẫu và A/D là tín hiệu số rời rạc. Vì vậy cần phải xây dựng thuật toán xử lí phương trình 16.1 cho các mẫu tín hiệu số rời rạc của $e(t)$. Ta giả thiết rằng các mẫu được lấy đều với khoảng thời gian Δt. Biến số lối ra sẽ được đo thành một chuỗi v_n, với n = 0, 1, 2, ...

Ta có thể viết $v_n \equiv v(n\Delta t)$, với n = 0, 1, 2, ...

Số hạng tỉ lệ $K_P e(t)$ có thể viết thành: $K_P.e_{v_n}$. đó là dòng điện được cấp bởi điều khiển tỉ lệ trong khoảng thời gian Δt sau mẫu thứ n. Dòng này tỉ lệ với tín hiệu sai lệch $e_{r_n} = v_0 - v$ với điều kiện mẫu thứ n là không đổi.

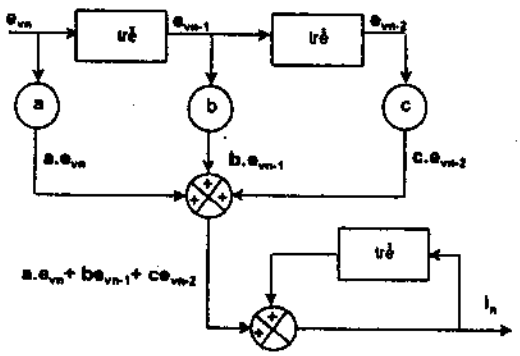
Từ đó có thể viết biểu thức 16.1 dưới dạng rời rạc:

$$i_n = K_P e_{v_n} + K_I \Delta t \sum_{n=0}^{n} e_{v_n} + \frac{K_D}{\Delta t}(e_{v_n} - e_{v_{n-1}}) \qquad (16.2)$$

Trong thực tế thường biểu thức trên hay được viết dưới dạng:

$$i_n = \sum \left(a e_{v_m} + e_{v_{m-1}} + c e_{v_{m-2}} \right) \qquad (16.3)$$

Biểu thức này đại diện cho sơ đồ điều khiển sau:



Ở đây :

$$a = K_P + \frac{K_D}{\Delta t} + K_I \Delta t$$

$$b = -\frac{2K_D}{\Delta t} - K_P$$

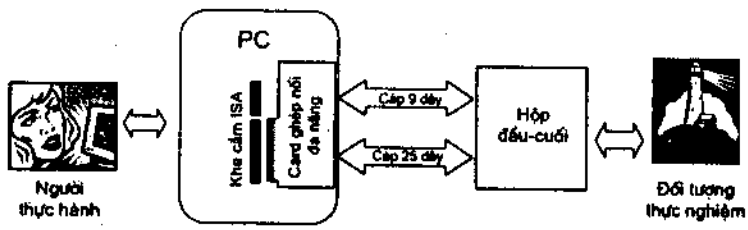
$$c = \frac{K_D}{\Delta t}$$

CHƯƠNG 17. THỰC HÀNH TRÊN BẢN MẠCH GHÉP NỐI ĐA NĂNG

17.1. GIỚI THIỆU CHUNG

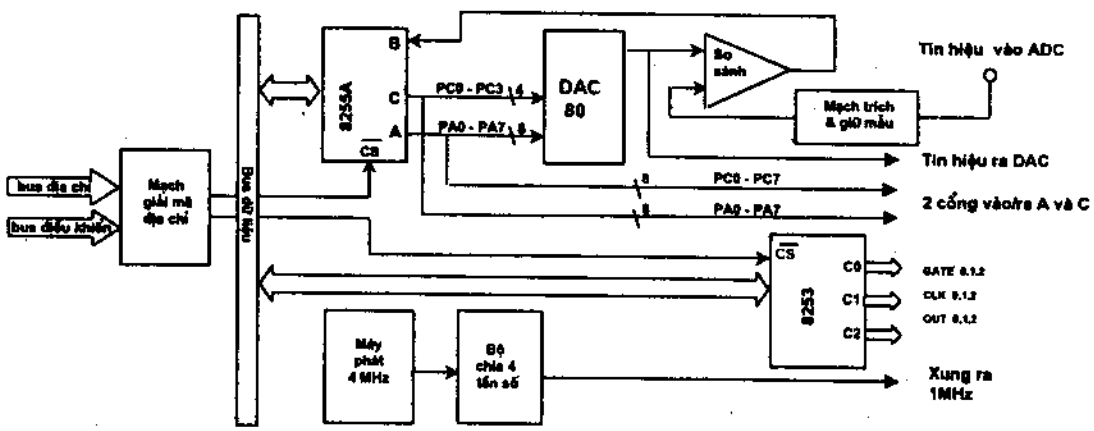
Các bản mạch ghép nối máy tính dùng cho thông tin vào/ra số và tương tự được bán trên thị trường cùng các chương trình phần mềm điều khiển và ứng dụng kèm theo. Trong điều kiện Việt Nam ta có thể tự xây dựng được các bản mạch như vậy với một số linh kiện phần cứng tối thiểu có thể dùng cho các đo lường và điều khiển không yêu cầu chỉ tiêu về tốc độ, độ chính xác quá cao. Những bản mạch như vậy rất hữu ích cho việc nghiên cứu học tập trong các cơ sở đào tạo đại học, cao đẳng, trường dạy nghề và cả trong các trường phổ thông cần những bài thực hành về vật lí, hoá học v.v.. được ghép nối với máy vi tính.

Dưới đây là giới thiệu về sơ đồ nguyên lí và các thực hành ứng dụng của một bản mạch ghép nối đa năng như vậy. Bản mạch này được thiết kế với một số ít nhất các chip điện tử dùng cho kĩ thuật vi xử lí và ghép nối. Để giao tiếp với CPU, nó được cắm trên một khe cắm mở rộng ISA của máy vi tính. Các tín hiệu vào/ra số và tương tự được đưa ra các đầu cắm D-25 và D-9 nhằm cho phép nối chúng tới các đầu cắm trên một hộp đầu-cuối (front-end) bằng các cáp nối 25 và 9 dây. Học viên sẽ đấu nối dây với các đối tượng cần đo và điều khiển cũng như quan sát trạng thái của các cổng vào/ra trên hộp đầu cuối một cách dễ dàng và thuận tiện, tránh gây nguy hiểm cho máy vi tính và bản mạch ghép nối đa năng. Sơ đồ vị trí của máy vi tính, bản mạch ghép nối đa năng và hộp đầu cuối được chỉ ra trên hình 17.1.



Hình 17.1 Vị trí của các khối dùng trong thí nghiệm.

Sơ đồ khối của bản mạch ghép nối đa năng như hình 17.2.



Hình 17.2 Sơ đồ khối của bản mạch ghép nối đa năng.

Bản mạch ghép nối được sử dụng cho các mục đích cụ thể sau:

- Vào/ra dữ liệu số với mức lôgic TTL qua chip PPI-8255A có địa chỉ cơ sở tại 300h.
- Thực hiện những ứng dụng đo đạc liên quan tới việc tạo ra những khoảng thời gian chính xác qua chip PIT-8253 có địa chỉ cơ sở tại 304h.
- Biến đổi dữ liệu nhị phân 12 bit hoặc tương đương trong máy tính thành tín hiệu tương tự nhằm điều khiển các ngoại vi bằng máy tính. Nhiệm vụ này được thực hiện bởi sự tổ hợp chip PPI-8255A và chip chuyển đổi số - tương tự DAC-80.
- Thu thập tín hiệu tương tự từ một ngoại vi, chuyển đổi ra dữ liệu số tương ứng với mã nhị phân 12 bit đưa vào máy vi tính cũng bằng tổ hợp trên gọi là bộ chuyển đổi AD với phần cứng tối thiểu.

2 cổng vào/ra 8 bit A và C của chip PPI-8255A trên bản mạch được dẫn ra đầu cắm tới hộp đầu cuối là:

PA0, PA1, PA2, PA3, PA4, PA5, PA6, PA7 và
PC0, PC1, PC2, PC3, PC4, PC5, PC6, PC7.

9 lối vào và ra của 3 bộ đếm C0, C1, C2 chip PIT-8254 được dẫn trực tiếp ra hộp đầu cuối là:

CLK0, GATE0, OUT0
CLK1, GATE1, OUT1
CLK2, GATE2, OUT2

Xung nhịp cho các bộ đếm được có sẵn tại một máy phát sóng thạch anh 1 MHz trong bản mạch được dẫn ra hộp đầu cuối.

Lối ra số - tương tự của vi mạch chuyển đổi D/A loại DAC-80 được dẫn ra hộp đầu cuối.

Chuyển đổi A/D ở đây được xây dựng trên cơ sở vi mạch DAC-80 cùng phần cứng tối thiểu là các cổng ra A và C (8 bit cổng A làm thành byte thấp và 4 bit thấp cổng C_L làm thành 4 bit cao của từ 12 bit đưa vào DAC) cùng với vi mạch so sánh LM311 có đầu ra collector hở. Lối ra của bộ so sánh được đưa vào bit PB0 của cổng B. Vậy trong quá trình biến đổi D/A hoặc A/D cổng A và C_L phải ở mode ra, cổng B ở mode vào.

Như vậy, hộp đầu cuối có các bộ phận và đầu nối sau:

a. 2 cổng vào/ra 8 bit A và C của PPI-8255A gồm:
   - 2 x 8 ổ cắm cái cho 2 x 8 bit vào/ ra của các cổng A và C.
   - 1 công tắc 3 vị trí: vị trí giữa cho mode ra, 2 vị trí 2 bên cho mode vào tương ứng của cổng A hoặc C.
   - 2 x 8 công tắc chọn giá trị bit cho 2 cổng trong mode vào.

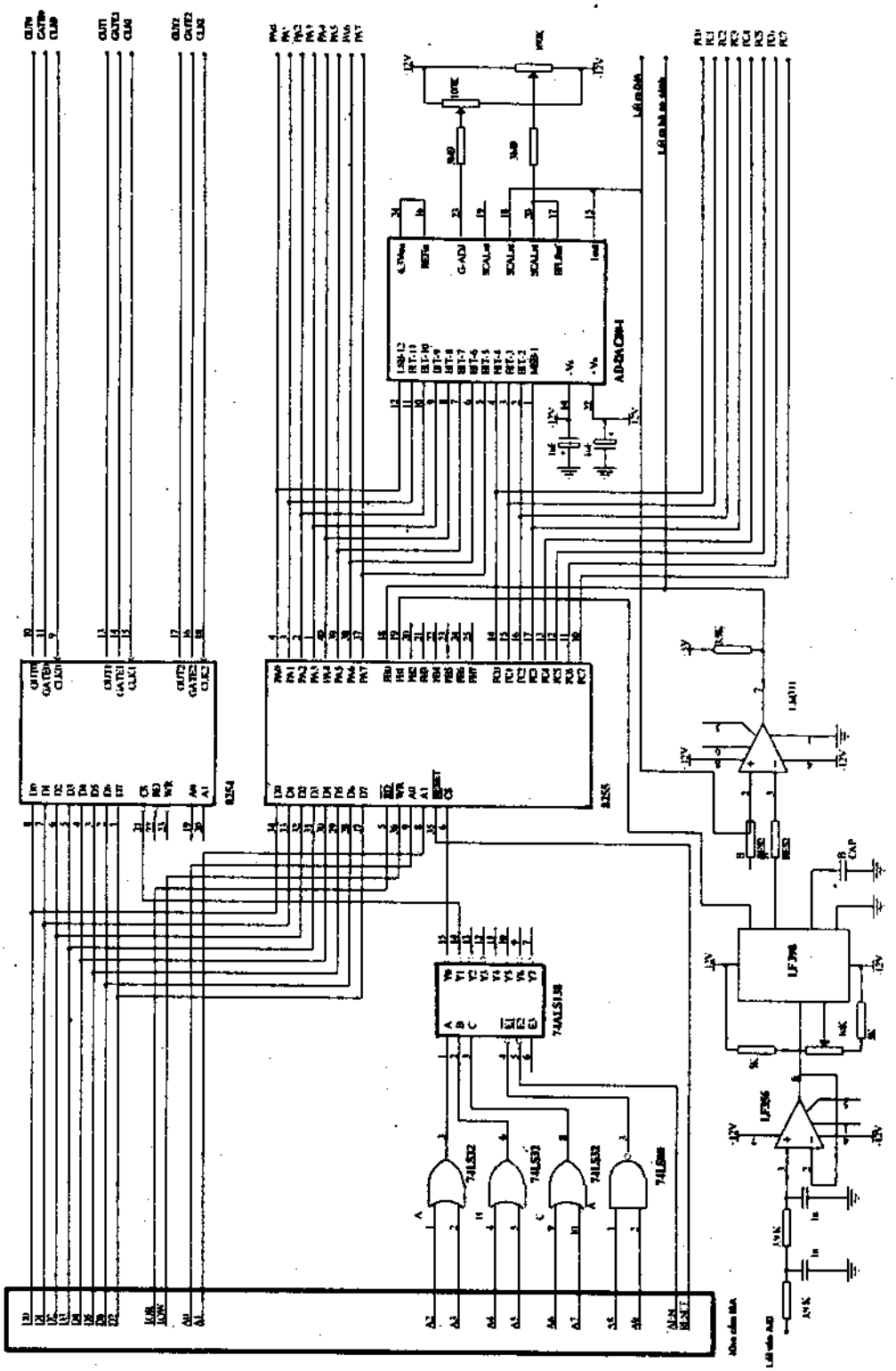
b. 9 đầu cắm chân tín hiệu của của PIT-8253/54 gồm:
   - $GATE_i$, CLK_i và OUT_i với i là số các bộ đếm từ 0 đến 2.
   - 1 lối ra xung nhịp1 MHz từ máy phát sóng thạch anh có độ ổn định tần số cao.

c. 1 lối vào cho tín hiệu tương tự cần chuyển đổi A/D có độ phân giải 12 bit.

d. 1 lối ra tín hiệu tương tự từ bộ chuyển đổi D/A 12 bit.

e. Trong hộp đầu cuối, các lối vào/ra được thiết kế lắp ráp thêm các vi mạch hoặc transistor đệm để bảo vệ cho các linh kiện ghép nối. Ngoài ra còn có các bộ phận phụ trợ cho thực nghiệm sau:
   - Phím nhấn và lối ra của 1 công tắc chống rung (anti-baund).
   - Phím nhấn và lối ra của công tắc tạo xung đơn (one-shot).
   - Các lối ra nguồn nuôi ổn áp 1 chiều DC: ±5V , ±12V, biến đổi từ -10V đến +10V.
   - Von mét hiện số 3 1/2 digit với 2 dải đo: 0 ÷ 200mV và 0 ÷ 20V.

Sơ đồ nguyên lí của bản mạch ghép nối đa năng đơn giản được cho trên hình 17.3 sau:



Hình 17.3 Sơ đồ bản mạch ghép nối đa năng.

CÂU HỎI, BÀI TẬP VÀ THỰC HÀNH

Để làm các bài tập thực hành trong chương này nên xem trước phần ba, chương 18 và 19 là " Các công cụ dùng cho khảo sát phần cứng và thực hành ghép nối máy tính".

17.1. Nhập và cho chạy chương trình sau cho phép đếm từ 0 đến 255 (00h đến FFh) và viết chúng ra cổng A với thời gian 1 giây một lần đếm.

⊙ **Chương trình 17.1** (viết bằng ngôn ngữ C)

```
# define port_A          0x300
# define port_B          0x301
# define port_C          0x302
# define CRPPI           0x303          / * địa chỉ thanh ghi từ điều khiển */
# include <dos.h>
int     main (void)
{
        unsigned        char    i=0;
        outportb(CRPPI,0x80);          / * 1000 0000 : cổng A, B, C ra */
        for (i=0, i<=255, i++)
        {
                outportb(port_A, i);
                sleep (1);
        }
return (0);
}
```

▪ **Chương trình 17.2** (viết bằng Pascal)

```
uses dos,crt;
const   port_A = $300;
        port_B = $301;
        port_C = $302;
        CRPPI = $303;  { địa chỉ thanh ghi điều khiển là 303h }
var     i: integer;
begin
        i:=0;
        port[CRPPI]:=$80;       { 1000 0000: cổng A ra }
        for i:=0 to 255 do
                begin
                  port[port_A]:=i;
                  delay(1000);
                end;
end.
```

⊙ **Chương trình 17.3** (viết bằng hợp ngữ)

```
code    segment
        assume cs:code
port_A          equ     0300h
port_B          equ     0301h
```

```
port_C          equ     0302h
CRPPI           equ     0303h
start:
        mov dx, CRPPI
        mov al,80h
        out dx,al
        mov al,00h
loop:
        mov dx,port_A
        out dx,al
        call    delay
        inc al
        cmp     al,0ffh
        jnz     loop
        mov     ah,4ch          ; thoát khỏi chương trình
        int 21h
delay           ; chương trình phụ cho trễ 1 giây bằng PIT-8253 trong PC
        push    ax
        push    bx
        mov     ax,18           ; 18,2 nhịp trong một giây
                                ; địa chỉ của 8253 được chứa trong 0000:046c (từ
                                ; thấp) và 0000:046e (từ cao)
        mov     bx,0
        mov     es,bx           ; cộng số nhịp của đồng hồ (là 18)
        add     ax,es:[46ch]
        mov     bx,es:[46eh]
loop2:  ; so sánh số đếm hiện tại với nội dung của ax và bx, xem liệu 1giây đã qua chưa
        cmp     bx,es:[46eh]
        ja      loop2
        jb      over
        cmp     ax,es:[46ch]
        jg      loop2
over:   pop     bx
        pop     ax
        ret
code            ENDS
                END     start
```

17.2 Nhập và cho chạy chương trình sau cho phép đọc byte từ cổng A rồi viết byte này ra cổng C. Nếu một phím bất kỳ được nhấn hoặc tất cả các bit trên cổng A đều bằng 1 thì chương trình được kết thúc.

Θ Chương trình 17.4 (viết bằng ngôn ngữ C)

```
# define port_A         0x300
# define port_B         0x301
# define port_C         0x302
# define CRPPI          0x303   / * địa chỉ thanh ghi từ điều khiển */
```

```
# include       <dos.h>
# include       <conio.h>
int     main (void)
{
int     i;
        outportb(CRPPI,0x90);
        do
        {
                i=inportb(port_A);
                outportb(port_C,i);
        } while (!kbhit () &&(i!=0xff) );
return (0);
}
```

Θ Chương trình 17.5 (viết bằng Pascal)

```
uses crt;
const           port_A = $300;
                port_B = $301;
                port_C = $302;
                CRPPI = $303; { địa chỉ thanh ghi điều khiển là 303h }
var   i: integer;
begin
port[CRPPI]:=$90;               {1001 0000  A:vào; B, C: ra}
repeat
      i:=port[port_A];          {gán giá trị cổng A cho biến i}
      port[port_C]:=i;          {xuất ra cổng C}
until  (keypressed) or (i=$ff);
end.
```

Θ Chương trình 17.6 (viết bằng hợp ngữ)

```
code    segment
assume  cs: code
start:
mov     dx, 303h
mov     al,90h
out     dx,al
loop:
mov     dx,300h
in      al,dx           ; đọc cổng A
mov     dx,301h
out     dx,al           ; viết ra cổng B
cmp     al,ffh
jnz     loop:
mov     ah,04ch
int 21
```

```
code    ENDS    start
```

17.3 Từ bài tập này trở đi, cho địa chỉ cơ sở của 8255A là 300h. Viết một chương trình thoả mãn lần lượt các điều kiện sau:

- Nâng 2 bit PA5 và PA7 lên 1 mà không làm ảnh hưởng tới các bit còn lại.
- Trên 2 cổng A và C đã lắp sẵn các bộ chỉ thị mức lôgic bằng các đèn LED; hãy tạo một điểm sáng chạy vòng theo 2 cổng A và C theo quy luật lần lượt PA0 sáng → PA1 sáng .. PA7 sáng → PC7 sáng → PC6 sáng → ...PC0 sáng.
- Yêu cầu như trên nhưng điểm sáng chạy vòng ngược trở lại.

17.4 Mã Gray có ưu điểm hơn mã nhị phân là chỉ có một bit thay đổi khi số đếm thay đổi từ 0 đến 9. Bảng biến đổi từ mã thập phân sang nhị phân và Gray như sau:

Thập phân	Nhị phân	Mã Gray
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101

Nếu gọi giá trị các bit trong mã nhị phân lần lượt là B1, B2, B3, B4 thì các bit trong mã Gray G1, G2, G3, G4 tương ứng sẽ được tính như sau:

$G1 = B1 \oplus B2$

$G2 = B2 \oplus B3$

$G3 = B3 \oplus B4$

$G4 = B4$

Hãy viết chương trình phát ra các mã Gray ra cổng A theo 2 cách:

- Dùng một khai báo dãy hằng căn cứ vào bảng trên.
- Dùng toán tử *bitwise* hoặc tuyệt đối (^ trong C và xor trong Pascal) như công thức trên. *Gợi ý:* Dùng các *bitmask* như che các bit còn lại

để được giá trị nhị phân 8 bit của Bi:

```
bit1 = (i&1);          /* che bit 1 */
bit2 = (i&2);
bit3 = (i&4);
bit4 = (i&8);
```

17. 5 Viết chương trình xuất ra cổng A hai số BCD tương ứng với các số thập phân từ 0 đến 9999. Thời gian trễ giữa 2 số là 1 giây. *Gợi ý*: Có thể dùng hai biến để biểu diễn cho các số hàng đơn vị và hàng chục. Kết quả giá trị ra sẽ là (số hàng chục - thí dụ biến ***tens*** -dịch trái 4 bit rồi cộng với số hàng đơn vị - thí dụ biến ***units***). Thí dụ câu lệnh trong ngôn ngữ C:

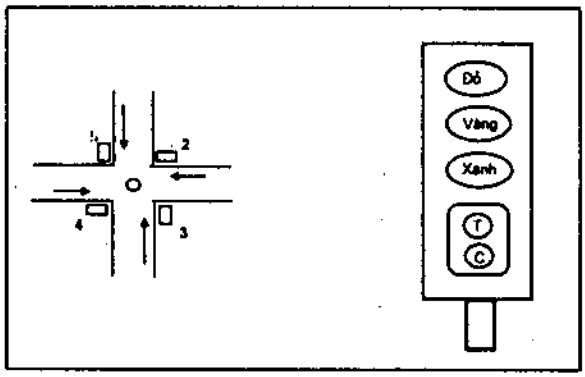
```
for (tens=0; tens <10; tens ++)
for (units =0; units <10; units ++)
```

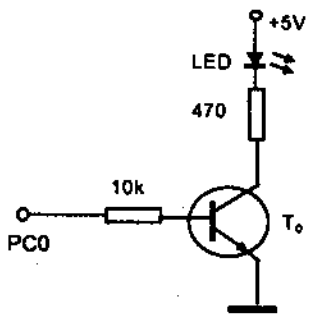
```
.... (tens <<4) + units
}
```

17. 6 Viết chương trình kiểm tra và thông báo ra màn hình máy vi tính giá trị của các bit trong cổng A ở chế độ 0, mode vào.

17. 7 Một bộ đèn điều khiển giao thông đơn giản được lắp trên 3 bit của cổng C như hình sau. Hã y viết chương trình điều khiển các đèn theo quy luật sau:

- Đèn vàng luôn sáng sau đèn xanh, đèn đỏ luôn sáng sau đèn vàng và đèn xanh luôn sáng sau đèn đỏ.
- Mỗi đèn sáng trong 2 phút.
- Các đèn sẽ sáng khi mức lôgic của của các bit tương ứng bằng 1.
- Hệ sẽ tắt hết các đèn và ngừng hoạt động khi bit PC7 bằng 1.

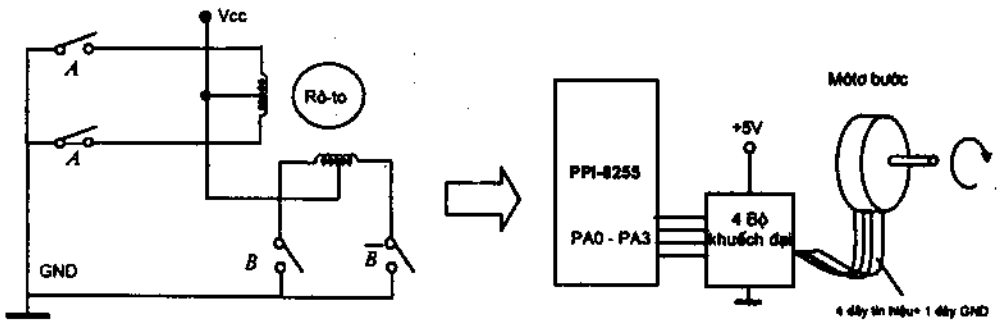




17. 8 Thiết kế hệ và viết chương trình cho một hệ đèn điều khiển giao thông phức tạp tại một ngã tư. Hệ được điều khiển bởi vi mạch 8255A như bài tập trên. Hệ gồm 4 bộ đèn 1, 2, 3, 4 như trên và mỗi bộ đèn gồm 3 đèn xanh, vàng, đỏ cho xe cộ và 2 đèn T(thông) và C(cấm) cho khách bộ hành đi cùng chiều. Lưu ý rằng, đèn T luôn sáng cùng đèn xanh và đèn C luôn sáng cùng đèn đỏ trên cùng một bộ đèn. Mỗi đèn được điều khiển bằng 1 transistor lắp trên bản breadboard theo sơ đồ đơn giản như hình trên, thí dụ đèn xanh trong bộ đèn 1 được điều khiển bởi transistor T_0, khi bit PA0 của cổng A ở mức cao (+5V) transistor sẽ thông làm đèn sáng v.v...

17. 9 Thực hành điều khiển môtơ bước bằng ghép nối vi tính trên cơ sở vi mạch 8255A.

Môtơ bước là loại thiết bị cho phép điều khiển chuyển động quay chính xác từng góc một theo các xung điện cấp vào các cuộn stator như hình sau:



Số bước trong một vòng quay (tương ứng với độ lớn của mỗi góc quay) tuỳ thuộc vào loại môtơ. Có 3 kiểu kích thích các cuộn dây như bảng sau. Trong đó 1 là chỉ trạng thái đóng của các công tắc, tức là khi có dòng điện chảy qua các cuộn dây tương ứng. 0 là trạng thái ngắt các công tắc. Trong bài thực hành, các lối ra của 4 bộ khuếch đại công suất được nối tới các cuộn Stator của môtơ bước. Khi cấp mức điện áp lôgic cao (+5V) tới các lối vào những bộ khuếch đại này tức là đã cấp dòng cho qua các cuộn Stator, tương ứng với việc đóng các công tắc. Như vậy, các công tắc A, B, $\overline{A}$ và $\overline{B}$ được thay bằng 4 bit điều khiển ra của cổng A hoặc C.

Bước	Kích thích 1 pha				Kích thích 2 pha				Kích thích 1-2 pha			
	A	B	$\overline{A}$	$\overline{B}$	A	B	$\overline{A}$	$\overline{B}$	A	B	$\overline{A}$	$\overline{B}$
1	1	0	0	0	1	1	0	0	1	0	0	0
2	0	1	0	0	0	1	1	0	1	1	0	0
3	0	0	1	0	0	0	1	1	0	1	0	0
4	0	0	0	1	1	0	0	1	0	1	1	0
5	1	0	0	0	1	1	0	0	0	0	1	0
6	0	1	0	0	0	1	1	0	0	0	1	1
7	0	0	1	0	0	0	1	1	0	0	0	1
8	0	0	0	1	1	0	0	1	1	0	0	1

Hãy viết chương trình điều khiển mô tơ bước theo các yêu cầu sau:

- Điều khiển quay phải. Tốc độ quay được biến đổi bằng lệnh delay (thời gian).
- Điều khiển quay trái.
- Xác định độ lớn (góc) của một bước.
- Điều khiển quay trái hoặc phải tuỳ vào phím nhấn trên bàn phím (thí dụ, nhấn phím T: quay trái; nhấn phím P: quay phải).

17.10 Thiết kế và viết chương trình cho một hệ thống bảo vệ cơ quan (security system) ghép nối với vi tính. Có 6 cảm biến được đặt tại các vị trí sau:

- Cảm biến 1 cho cửa trước.
- Cảm biến 2 cho cửa sau
- Cảm biến 3 cho cửa sổ 1.
- Cảm biến 4 cho cửa sổ 2.
- Cảm biến 5 cho cửa sổ 3.
- Cảm biến 6 cho cửa sổ 4.

Các cảm biến sẽ cho ra giá trị 0 khi cửa đóng, giá trị 1 khi cửa bị mở.

Có một công tắc ghi nhận báo động: bình thường cho mức lôgic 0, nếu nhấn cho mức lôgic 1 để ghi nhận.

Chương trình phải làm được các yêu cầu sau:

- Nếu bất kì một cửa hoặc cửa sổ nào bị mở thì một thông báo sẽ hiện lên mà hình để báo chính xác vị trí đã bị đột nhập. Hệ sẽ phát một tín hiệu ra điều khiển 1 đèn sáng nhấp nháy 2 giây một (1 giây sáng, 1 giây tắt) bằng mức lôgic 1 (điều khiển sáng) và mức lôgic 0 (điều khiển tắt).
- Nếu đã phát hiện có kẻ đột nhập, hệ vẫn tiếp tục kiểm tra tất cả các cửa còn lại và hiện thông báo như thường lệ.
- Thông báo có kẻ đột nhập được tồn tại cho đến khi người bảo vệ nhấn công tắc ghi nhận báo động.

- Hệ chỉ làm việc ngoài giờ hành chính (từ 8 giờ sáng đến 5 giờ chiều).

17.11 Nối lối ra của máy phát xung nhịp 1 MHz tới lối vào CLK2 của bộ đếm 2 chip PIT-8253/54; nối lối vào GATE2 lên +5V để luôn cho phép đếm. Nhập và cho chạy chương trình phát sóng vuông qua bộ đếm 2 trong chip PIT-8253/54 với tần số bất kì được nhập từ bàn phím như sau:

Θ Chương trình 17.7 (Phát sóng hình vuông qua bộ đếm 2 của PIT-8253)

```
uses crt;
var      f,N: real;
begin
clrscr;
port[$307]:= $B6;                {1011 0110 : Chọn C2, đọc/viết byte thấp-byte cao, mode3, nhị phân }
write(' Nhập tần số  =  ');
readln(f);
gotoxy(10,12);
N:=int(1E6/f+0.5);               { tính hệ số chia hay là số đếm ban đầu cần nạp }
if N <2 then writeln (' >> Tần số quá cao <<');
         else  i N >65535  then writeln ('>>Tần số quá thấp <<');
         else begin
         port[$306]:= lo(round(N));        { Nạp số đếm ban đầu vào bộ đếm 2 }
         port[$306]:= hi(round(N));
         writeln(' >> Tần số  =  ', 1E6/N:10:2, 'Hz <<');
         end;
gotoxy(1,24);
end.
```

17.12 Cũng thực hành như bài trên nhưng tần số có thể được điều chỉnh tăng hoặc giảm liên tục từng bước 100 Hz một bằng việc nhấn phím "T" (tăng) hoặc nhấn phím "G" (giảm).

Gợi ý: Cho chương trình vào trong một vòng lặp gồm thủ tục tính số nạp ban đầu của bộ đếm, nạp số đếm và một vòng lặp con phát hiện một phím bị nhấn trên bàn phím.

17.13 Dùng các bộ đếm trong 8253/54 và nguồn xung nhịp 1 MHz, viết chương trình để phát ra các xung tỷ lệ lặp lại có độ rỗng xung thoả mãn điều kiện sau: mức thấp 40 μs mức cao 200 μs. Quan sát và đo trên dao động kí.

17.14 Nối đầu ra của công tắc tạo xung đơn tới lối vào CLK1 của bộ đếm 1 vi mạch PIT-8253/54. Nhập và cho chạy chương trình liên tục hiển thị số xung được đếm. Nhấn công tắc để tạo xung. Quan sát kết quả trên màn hình.

Θ Chương trình 17.8 (Đếm xung bằng bộ đếm C1, PIT- 8254)

```
uses crt;
Type  BitVal      = 0..1;
         Timer    = C0 .. C2;
var      ch: char;
         f: word;
```

```
{---------}
Function  OutLevelC1: byte;          {Hàm xác định bit trạng thái pin 7}
var       staus: byte;
begin
          port[$307]:= $E4;          {1110 0100 : Chốt trạng thái bộ đếm 1,lệnh đọc TG điều khiển }
          status:= port[C1];         { đọc trạng thái thanh ghi từ điều khiển}
          OutLevel1 := status shr 7; { dịch phải 7 bit để xác định giá trị bit 7
                                       nếu bit này bằng 1 tức là OUT1 =1}
end;
{----------}
Function Count Value C1: word;     { Hàm đọc giá trị hiện tại của bộ đếm }
var       byte_thap, byte_cao: byte;
begin
          port[$307]:= $D4;                {1101 0100: chốt số đếm bộ đếm 1}
          byte_thap:=port[$305];     {đọc bộ đếm 1}
          byte_cao:= port[$305]:
          CountValue:= byte_thap + byte_cao * 256;
end;
{-----------}
Function CountLoadedC1: Boolean;
var       status: byte;
begin
          port[$307] := $E4;         {1110 0100: chốt trạng thái bộ đếm 1}
          status := port[$305];
          CountLoaded := ((status and $40) = 0);
end;
{-----------}
BEGIN          { Chương trình chính }
clrscr;
port[$307]:=$70;                     {0111 0000 :  chọn C1, đọc/viết byte thấp-
                                     cao, mode 0, đếm nhị phân }
port[$305]:= lo(65535);
port[$305]:= hi(65535);
repeat
          gotoxy(10,10);
          if OutLevel C1 =1 then write (' Tràn số đếm !!!')
                    else  if not Countloaded C1 then write (0, 'xung');
                    else write (65535-CountValueC1: 8, ' xung');
          delay(150);
until keypressed;
gotoxy(1,24);
END.
```

Giải thích các bước tiến hành như sau:

- Đặt chế độ cho bộ đếm 1
- Nạp số đếm ban đầu 65535 (tức là có thể đếm được cực đại 65535 xung).

- Khởi phát vòng lặp 'repeat ... until'. Sườn lên của OUT1 được dùng như một ngắt để chỉ thị rằng bộ đếm 1 đã đạt tới giá trị 0.
- Khi số đếm ban đầu chưa được nạp vào bộ đếm, chữ ' 0 xung' được hiện lên màn hình.
- Số xung nhịp (= giá trị ban đầu - giá trị hiện tại) được hiện lên màn hình.
- Lặp lại cho đến khi một phím trên bàn phím được nhấn.

17. 15 Nhập và cho chạy cũng chương trình đếm xung như trên nhưng dùng bộ đếm 0 và được viết trong ngôn ngữ C. Giá trị đếm ban đầu luôn là 1111 1111 1111 1111 = FFFFh được thực hiện bởi lệnh:

```
outportb(0x307, 0x30);     / * 0011 0000: chọn C0, đọc/viết byte thấp-cao,
                           mode 0, đếm nhị phân */

outportb(0x304,0xFF);      / * nạp byte thấp */

outportb(0x304,0xFF);      / * nạp byte cao */
```

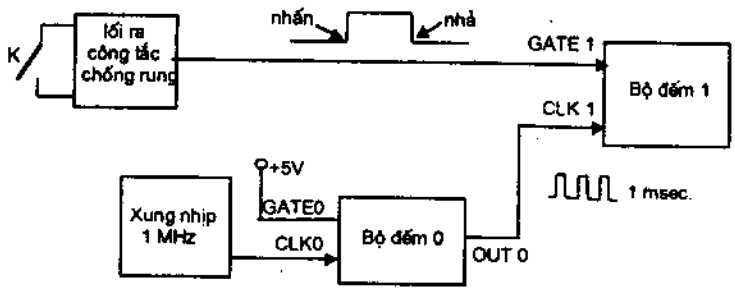
Tiếp đó số này được chốt vào bộ đếm bằng lệnh: outportb (0x307,0x00);

Θ Chương trình 17.9. (Đếm xung bằng bộ đếm C0)

```
# define C0              0x304
# define C1              0x305
# define C2              0x306
# define CRPIT           0x307   / * địa chỉ thanh ghi từ điều khiển */

# include        <stdio.h>
# include        <dos.h>
# include        <conio.h>
int      getcount (int portid);
void     set_up_PIT (void);
int      main (void)
{
int      count;
         set_up_PIT ( );
         do
         {
                  clrscr ( );
                  count = getcount (C0);
                  printf("số đếm là %d\n", count);
                  delay(1000);
         } while (!kbhit ( ));
         return (0);
}

void     set_up_PIT (void)
         {
         outportb(CRPIT,0x30);
         outportb(C0,0xFF);
```

```
        outportb(C0,0xFF);
        outportb(CRPIT,0x00);
        }

int     getcount (int portid)
        {
        int     lsb,msb;
        lsb=inportb(portid);
        msb =inportb(portid)'
        return(0xFFFF - (lsb + (msb <<8 )));
        }
```

17. 16 Thực hành đo thời gian chính xác giữa hai sự kiện (thí dụ giữa các sự kiện nhấn và nhả công tắc chống rung K) như sơ đồ hình sau. Các xung đếm chuẩn có độ rộng 1 ms được tạo ra từ nguồn xung nhịp 1MHz (có sai số tương đối về tần số rất thấp 10^{-5} đến 10^{-6}) và lối ra OUT0 của bộ đếm 0 được đưa tới lối vào CLK1 của bộ đếm 1. Bộ đếm sẽ chỉ đếm được số xung trong khoảng thời gian giữa 2 lần nhấn công tắc K tức là khi giá trị GATE1 bằng 1.



Chương trình có thể được tiến hành theo các bước hướng dẫn sau:

- Đặt mode đếm 3 cho bộ đếm 0
- Viết số đếm ban đầu 1000.
- Đặt mốt đếm 0 cho bộ đếm 1.
- Viết số đếm ban đầu lớn nhất 65535.
- Đợi cho đến khi số đếm đã nạp xong.
- Đợi cho đến khi giá trị đếm nhỏ hơn 65535. Tức là, chương trình sẽ đợi cho đến khi bộ đếm 1 khởi phát đếm (GATE1 = 1) .
- Đợi cho đến khi bộ đếm 1 ngừng đếm (GATE1 = 0). Vòng lặp này gồm các thao tác như sau:
- Đọc số đếm cho biến x.
- Đợi cho đến khi xung 1ms từ OUT0 lên mức cao.
- Đợi cho đến khi xung 1ms từ OUT0 xuống mức thấp.
- Kết thúc vòng lặp nếu sau một chu kì (OUT0 từ 1 xuống 0) giá trị đếm không thay đổi.
- Nếu số đếm bằng 0 báo quá thời gian (overflow) trên màn hình.

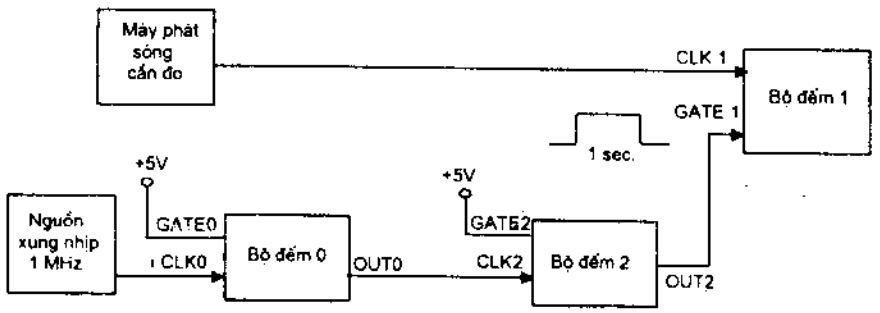
- Tính số xung đếm bằng giá trị dầu trừ đi giá trị đếm được, đó chính là khoảng thời gian tính ra mili giây.

Θ Chương trình 17.10 (Đo thời gian giữa hai sự kiện)

```
uses crt;
var     x:word;
{----------}
Function  OutLevelC1: byte;          {Hàm xác định bit trạng thái pin 7}
var     staus: byte;
begin
        port[$307]:= $E4;            {1110 0100 : Chốt trạng thái bộ đếm 1 }
        status:= port[C1];           { đọc trạng thái thanh ghi từ điều khiển}
        OutLevel1 := status shr 7; { dịch phải 7 bit để xác định giá trị bit 7
                                          nếu bit này bằng 1 tức là OUT1 =1}
end;
{----------}
Function Count Value C1: word;     { Hàm đọc giá trị hiện tại của bộ đếm }
var     byte_thap, byte_cao: byte;
begin
        port[$307]:=$E4;                    {1101 0100: chốt số đếm bộ đếm 1}
        byte_thap:=port[$305];              {đọc bộ đếm 1}
        byte_cao:= port[$305]:
        CountValue:= byte_thap + byte_cao * 256;
end;
{----------}
Function CountLoadedC1: Boolean;
var     status: byte;
begin
        port[$307] := $E6;           {1110 0100: chốt trạng thái bộ đếm 1}
        status := port[$305];
        CountLoaded := ((status and $40) = 0);
end;
{----------}

BEGIN           { Chương trình chính }
  Clrscr;
port[$307] := $36;                   { 0011 0110: Chọn C0, mode 3 }
port[$304] := lo(1000);              { nạp số 1000 cho bộ đếm 0}
port[$304] := hi(1000);
port[$307] := $70;                   { 0111 0000: chọn C1, mode 0 }
port[$305] := lo(65535);                    { nạp số 65535 cho bộ đếm 1 }
port[$305] := hi(65535);
repeat until CountLoadedC1;
repeat until CountValueC1 < 65535;
repeat
        x:= Count ValueC1;
        repeat until OutLevelC0 = 1;   { thoát không đếm nữa nếu giá trị của C1 }
```

```
        repeat until OutLevelC0 = 0;   { không đổi sau 1 chu kì xung nhịp }
until  CountValueC1 = x;
Gotoxy(30,10);
if OutLevelC1 = 1 then write (' Tràn số !!!');
else  write (' Thời gian trôi qua là:     ', 65535 - x :8, ' ms');
gotoxy(1,24);
END.
```

17. 17 Sơ đồ tạo một bộ đo tần số xung tuần hoàn chính xác dùng vi mạch PIT-8253/54 như hình sau



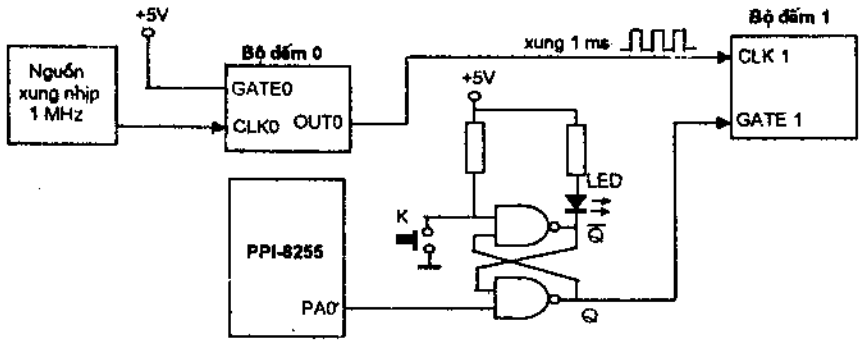
Khi viết chương trình cần chú ý các điểm sau đây:

- Bộ đếm 1 sẽ chỉ đếm xuống khi GATE1 = OUT2 = 1.
- Một chu kì của OUT0 (từ mức cao xuống thấp) mất 50ms.
- OUT2 sẽ khởi phát ở mức cao. Khi số đếm giảm đến 1, OUT2 sẽ nhảy xuống mức thấp trong một chu kì xung nhịp rồi lại lên mức cao. Bộ đếm lúc đó lại được nạp lại số đếm ban đầu và xử lí lại được lặp lại. GATE2 =0 thì cấm đếm, GATE2 =1 thì cho phép đếm. Ngay khi GATE2 =0, OUT2 sẽ ở mức cao.Vì vậy bộ đếm 2 được nạp với giá trị ban đầu là 21.
- Khi GATE1 = OUT2 ở mức thấp, bộ đếm 1 bị cấm đếm. Việc đếm các xung trên đường CLK1 (tức là các xung của máy phát cần đo) chỉ được phép trong 1 giây (20 x 50 ms) rồi lại bị cấm trong 50 ms (1 x 50ms).

Do vậy các bước chương trình sẽ có thể được tiến hành như sau:

- Đặt mode đếm 3 (phát sóng vuông) cho bộ đếm 0. Nạp số đếm ban đầu 50 000.
- Đặt mode đếm 2 (phát sóng tỉ lệ) cho bộ đếm 2. Nạp số đếm ban đầu là 21.
- Đặt mode đếm 0 (đếm) cho bộ đếm 1. Nạp số đếm ban đầu là 65 535.
- Tạo vòng lặp hiện tần số sóng mỗi khi một phím trên bàn phím được nhấn; cho đến khi bấm phím thoát (thí dụ 'x'):
- Vòng lặp lớn cho tới khi một phím trên bàn phím được nhấn thì thoát và dừng chương trình.
- Vòng lặp đợi cho đến khi OUT2 = 1 (nếu xảy ra báo tràn số).
- Vòng lặp đợi cho đến khi OUT2= 0.
- Nếu OUT1 = 1 thì báo tràn số, nếu không đọc số đếm ở bộ đếm 1. Nếu đã nạp.xong số đếm thì tần số f = 65535 – giá trị đếm hiện tại.
- Viết kết quả giá trị tần số ra màn hình.
- Nạp lại số đếm ban đầu 65 535 cho bộ đếm 1.

17. 18 Kết hợp giữa hai chip PPI-8255A và PIT-8253/54 hãy lắp đặt và viết chương trình để đo thời gian phản ứng của một người. Đó là khoảng thời gian tính từ khi 1 tín hiệu được hẹn trước xuất hiện (thí dụ 1 đèn được bật sáng) đến thời điểm người đó nhấn một công tắc đã cho.



Bộ đếm 0 được đặt ở mode 3, số đếm ban đầu được nạp là 1000. Do vậy, lối ra của nó là sóng 1MHz/1000 = 1kHz, tức là sẽ có các xung chu kì 1ms để đưa vào bộ đếm 1. Số xung mà bộ đếm 1 đếm được trong khoảng thời gian GATE1 ở mức 1 chính là thời gian đó tính ra mili giây. Chip PPI-8255A được đặt ở chế độ 0, cổng A ở mode ra và bít PA0 được đặt lên 1. Trước khi đo công tắc K được nhấn để RESET lại mạch lật. Lối ra Q của mạch lật bằng 0 cấm bộ đếm 1. Lối ra $\overline{Q}$=1 làm đèn LED sáng. Có thể tạo một khoảng thời gian ngẫu nhiên kể từ lúc khởi phát chạy chương trình đến khi bật sáng đèn bằng một vòng lặp tìm điều kiện để một hàm ngẫu nhiên nhận giá trị nằm trong một khoảng nào đó. Khoảng giá trị càng nhỏ xác xuất thoát khỏi vòng lặp càng nhỏ và khoảng thời gian này có thể càng dài.

Thí dụ, `Repeat until (((60 < RANDOM (100)) and (65 > RANDOM(100)));`

Khi thoát khỏi vòng lặp, chương trình tạo xung xoá bit PA0 rồi đặt lên 1 ngay và điều đó làm cho mạch lật chuyển trạng thái.

Thí dụ,
```
port [$300] := 0;
port [$300] := 1;
```

Lối ra $\overline{Q}$ lật về 0 làm đèn LED bật sáng, đồng thời lối ra Q nhảy lên mức 1 làm GATE1=1 cho phép bộ đếm 1 đếm các xung 1ms. Số đếm sẽ giảm theo thời gian chừng nào mà công tắc K chưa được nhấn. Khi số đếm không thay đổi nữa chính là lúc GATE1=0 hay mạch lật đã chuyển trạng thái lại, hay công tắc K đã được nhấn. Do vậy, ngay sau khi tạo xung ở PA0 chương trình nhảy vào vòng lặp đếm số xung tại bộ đếm 1 cho đến khi số đếm không thay đổi trong một vài chu kì. Thời gian phản ứng chính bằng số đếm nạp ban đầu (thí dụ, 65535) trừ đi số đếm hiện tại không đổi.

17.19 Viết chương trình biến đổi các số 0, 255, 511, 1023, 2047 và 4095 thành các tín hiệu tương tự ở lối ra DAC của chip DAC-80. Đo và ghi lại các giá trị thế tương ứng bằng

von-mét trên front-end Vẽ đồ thị sự phụ thuộc điện thế ra vào giá trị số lối vào. Nhận xét tính tuyến tính của đồ thị.

17.20 Viết chương trình phát ra một sóng hình răng cưa với tần số và biên độ lớn nhất có thể được bằng bộ biến đổi DA. Đo và quan sát dạng sóng lối ra trên front-end.

17.21 Viết chương trình như trên nhưng có thể biến đổi liên tục biên độ và tần số bằng việc nhấn các phím trên bàn phím (thí dụ, biến đổi liên tục biên độ bằng 2 phím T(tăng) hoặc G (giảm) và tần số bằng 2 phím I (tăng) hoặc D (giảm).

17.22 Thực hiện các nội dung như các bài tập 17.19 và 17.20 nhưng lần này phát ra các xung vuông với độ rỗng biến đổi được.

17.23 Thực hiện các nội dung như các bài tập 17.19 và 17.20 cho việc phát ra sóng hình sin.

17.24 Cấp điện thế một chiều biến đổi độ lớn được (± 10V) trên bộ front-end tới lối vào ADC. Viết chương trình biến đổi AD điện áp này theo phương pháp bậc thang. Đo liên tục (0,5 giây một lần) và hiện giá trị số lên màn hình máy vi tính.

17.25 Làm bài thực hành trên nhưng cho chương trình tính toán và hiện kết quả ra số Vôn. Dùng Vôn-mét trên front-end để kiểm tra, đối chứng. *Gợi ý*: phải dùng một vôn mét bên ngoài để chuẩn và tính các hệ số chuyển đổi.

17.26 Làm bài thực hành trên nhưng dùng thuật toán chia đôi dây cung để biến đổi AD theo phương pháp xấp xỷ liên tiếp. Dùng dao động kí quan sát thế lối ra bộ biến đổi DAC-80, nhận xét.

17.27 Với mục đích như bài tập trên nhưng hãy viết chương trình dùng phương pháp biến đổi xấp xỉ liên tiếp .

17.28 Với mục đích như bài tập trên nhưng hãy viết chương trình dùng phương pháp biến đổi AD theo kiểu bám (tracing).

17.30 Xây dựng một dao động kí nhớ một kênh dùng bộ biến đổi ADC trên cở sở chip DAC-80 và phần cứng tối thiểu theo phương pháp xấp xỉ liên tiếp. Khảo sát tốc độ biến đổi nhanh nhất.

PHẦN BA

CÁC CÔNG CỤ
DÙNG CHO
KHẢO SÁT PHẦN CỨNG
VÀ
THỰC HÀNH GHÉP NỐI MÁY TÍNH

CHƯƠNG 18. CÁC THIẾT BỊ ĐIỆN TỬ

Các tín hiệu trong mạch điện tương tự (như bộ khuếch đại không có phản hồi hoặc các mạch số tổ hợp) thường có thể được kiểm tra từng bước một, tầng nọ sau tầng kia vì tín hiệu ở tầng sau phụ thuộc vào tầng trước một cách rõ ràng. Ngược lại, trong nhiều hệ thống số, có rất nhiều tín hiệu độc lập với nhau. Máy vi tính là một hệ thống số phức tạp nên cũng nằm trong tình trạng đó. Thí dụ, một mạch vi xử lí 8 bit như 8086 luôn có đến hơn 30 loại tín hiệu trên bus địa chỉ, số liệu và bus điều khiển hoàn toàn độc lập với nhau, nhưng thường phải được kiểm tra đồng thời nếu muốn khảo sát, sửa chữa mạch điện liên quan đến chúng. Mặt khác tất cả các tín hiệu kể trên lại đều phụ thuộc vào sự điều khiển của chip vi xử lí. Nhưng sự phụ thuộc này thường rất khó xác định vì hầu hết các vi mạch số chuyển tải các tín hiệu đó đều là mạch lôgic dãy, có nhớ. Chính vì vậy, có những lỗi phát hiện ra ở một thời điểm nào đó có khi lại đã xảy ra ở một thời điểm trước đó rồi. Các tín hiệu điện ở trong máy tính thường là không tuần hoàn, cho nên một dao động kí thông thường nếu là rất tốt cho việc kiểm tra tín hiệu tuần hoàn thì lại không thể sử dụng ở đây được. Chỉ có một đặc điểm khá thuận lợi của việc đo các tín hiệu trong máy vi tính là các tín hiệu số loại này chỉ có hai mức điện thế nên không cần đòi hỏi một thiết bị đo điện áp thật chính xác.

Chương này sẽ mô tả tóm tắt một số thiết bị dùng cho các mục đích đo khác nhau trong mạch điện tử số nói chung và máy vi tính nói riêng. Các chương trình phần mềm về kiểm tra chẩn đoán cũng được đề cập tới.

18.1. ĐỒNG HỒ VẠN NĂNG (MULTIMETER)

Đồng hồ vạn năng (A-V-Ω mét) có 3 chức năng chính là đo điện áp (vôn mét), đo dòng điện (ampe mét) và đo điện trở (ôm mét). Các vôn mét cũng như ampe mét thường chỉ được dùng để đo điện áp cũng như dòng tiêu thụ của nguồn nuôi và các tín hiệu tĩnh vì đáp ứng thời gian của chúng thường rất chậm so với sự thay đổi của các tín hiệu trên mạch điện máy tính. Một đồng hồ vạn năng thường chỉ được thiết kế đo điện áp một chiều DC và xoay chiều AC trong dải tần từ 50 đến 400Hz. Trong khi đó các tín hiệu trên các bản mạch chính cũng như các bus mở rộng thường có tần số cao hơn nhiều. Thí dụ, bus mở rộng có tốc độ chậm như ISA cũng cho phép tần số xung nhịp lên tới 8,33 MHz. Dầu sao, đồng hồ vạn năng cũng rất hữu ích trong các quá trình đo đạc sơ bộ khi khảo sát, bảo dưỡng và sửa chữa máy vi tính.

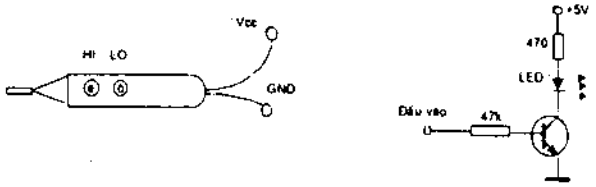
18.2. DAO ĐỘNG KÍ

Dao động kí hay máy hiện sóng (Oscilloscope) thông thường có thể được dùng để phân tích các tín hiệu điện được phát ra do các xử lí có tính lặp lại. Các sai hỏng về mức điện áp có thể được phát hiện khi dùng dao động kí. Một dao động kí nhiều chùm tia có thể được dùng để cung cấp các thông tin định lượng về mối quan hệ thời gian giữa các tín hiệu. Dao động kí thông thường chỉ có thể kiểm tra các tín hiệu có tính tuần hoàn. Với các dao động kí nhớ (storage oscilloscopes) có thể ghi được các xung đơn lẻ nhưng điều kiện trigơ thường đòi hỏi khắt khe, do vậy có trường hợp phải lập lại phép đo nhiều lần mới thu được kết quả mong muốn.

18.3. ĐẦU DÒ LÔGIC

Đầu dò lôgic (logic probe) là một công cụ đơn giản nhưng rất hữu ích trong các công việc kiểm tra cơ bản mạch điện số, thí dụ như xác định trạng thái lôgic cao hay thấp của đường

truyền tín hiệu. Một đầu dò lôgic như hình 18.1 gồm một đèn LED đỏ để chỉ thị mức thế ở đầu dò là mức cao (Hi) và một đèn LED xanh chỉ thị cho mức thấp (Lo). Với các trạng thái lôgic tĩnh ta được các chỉ thị sáng của một trong hai đèn tương ứng với mức lôgic nào đó. Khi đường dây ở trạng thái truyền tín hiệu biến đổi có tần số đủ thấp ta nhận được chỉ thị nhấp nháy của hai đèn. Hiện nay các chỉ thị lôgic tổ hợp nhiều đầu vào cũng trở nên thông dụng. Thực chất đó là một tập hợp các đầu dò lôgic được chế tạo cùng trên một giá đỡ có thể kẹp ngay vào 2 hàng chân của các vi mạch số 14, 16 hay nhiều chân. Giá trị mức lôgic ở mỗi chân vi mạch sẽ được chỉ thị đồng thời bởi các đèn LED tương ứng. Đầu dò lôgic đơn giản nhất được thiết kế như sơ đồ hình 18.1, transistor hoạt động ở một trong hai chế độ cấm và thông bão hoà tương ứng với một trong hai trạng thái lôgic thấp hoặc cao ở đầu vào đầu dò. Đèn LED sẽ sáng để chỉ thị mức lôgic cao.



Hình 18.1 Sơ đồ nguyên lí một đầu dò lôgic đơn giản.

18.4. THIẾT BỊ PHÂN TÍCH LÔGIC (Logic Analyzer)

18.4.1. Nguyên lí hoạt động

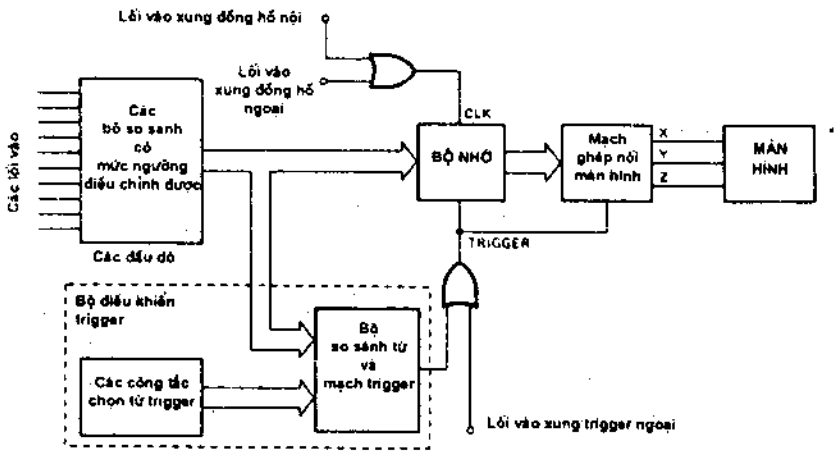
Ngoài những chức năng giống các thiết bị kể trên, thiết bị phân tích lôgic cho phép thực hiện những yêu cầu mà các thiết bị trên không thể đáp ứng nổi, đó là:

Cho phép kiểm tra đồng thời một số lớn các điểm trên mạch điện,

Phân tích các tín hiệu không liên tục.

Phân tích các xung đơn lẻ trên hệ thống.

Sơ đồ khối nguyên lí của bộ phân tích lôgic được trình bày trong hình 18.2.

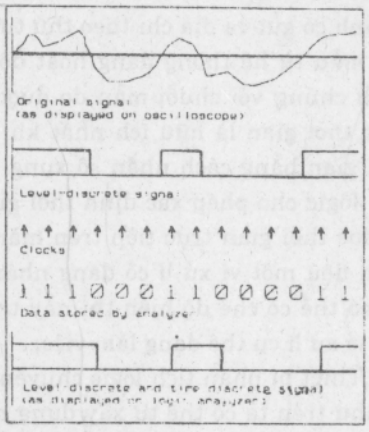


Hình 18.2 Sơ đồ khối của bộ phân tích lôgic.

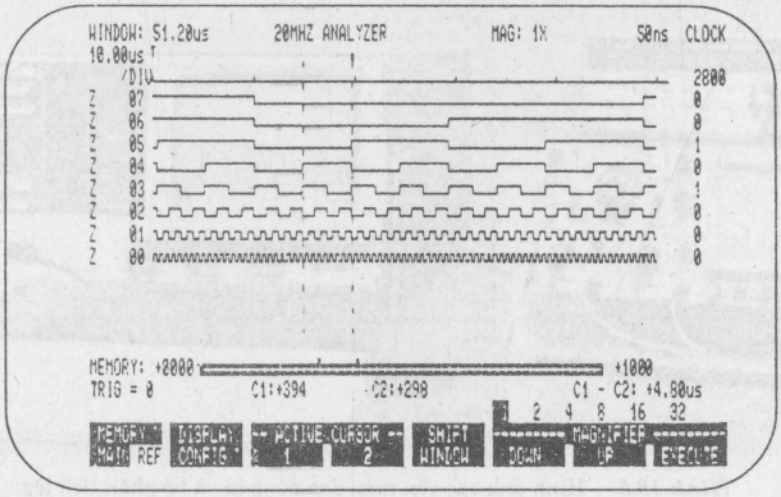
Nó bao gồm một bộ nhớ, một bộ điều khiển trigơ, một máy phát nhịp đồng hồ, một bộ nhớ, một monitor được nối với máy tính và các đầu dò xung nhịp và số liệu bên ngoài. Các đầu dò

thực chất là các mạch điện so sánh (comparators) được chương trình hoá phù hợp với các mức ngưỡng của họ lôgic cần phân tích và do đó tín hiệu lối ra của chúng sẽ chỉ còn tồn tại hai mức lôgic cố định (thí dụ mức "0" và "1" của họ TTL).

Nếu bộ phân tích nhận được tín hiệu nhịp đồng hồ thì nó liên tục lấy mẫu dữ liệu và cất giữ các mẫu này trong bộ nhớ RAM khi có xung điều khiển trigơ. Các mẫu đo được cất giữ trong bộ nhớ là chuỗi các địa chỉ hoặc từ dữ liệu mà mạch cần kiểm tra lần lượt phát ra. Tín hiệu nhịp đồng hồ có thể được lấy từ bên ngoài (external clock) hay được phát từ máy phát bên trong bộ phân tích lôgic (internal clock). Bộ điều khiển trigơ là một mạch điện được chương trình hoá bằng máy tính trước khi ghi các số liệu vào và kiểm tra chúng cũng như yêu cầu bộ nhớ lưu trữ lại số liệu. Tín hiệu "trigơ" sẽ báo cho bộ phân tích biết khi nào dừng lấy mẫu và lưu giữ mẫu đó trong RAM. Có thể sử dụng tín hiệu bên trong làm "trigơ" hay có thể dùng bộ "nhận dạng từ" (word recognizer) để tạo ra tín hiệu trigơ. Bộ nhận dạng từ sẽ so sánh từ nhị phân trên đường tín hiệu vào với một từ được đặt bằng khoá (switch) hay bàn phím (keyboard). Khi hai từ giống nhau thì bộ nhận dạng từ sẽ gửi ra một tín hiệu trigơ. Ở cuối quá trình ghi, số liệu được thu thập trong bộ nhớ được máy tính xử lí và cho hiện trên màn hình. Để so sánh với dao động kí, hình 18.3 chỉ ra tín hiệu được hiện trên dao động kí và trên màn hình bộ phân tích lôgic. Đường cong trên cùng ghi lại tín hiệu tương tự thực tế được hiện ra trên dao động kí. Cũng tín hiệu này được đưa vào một đầu dò của bộ phân tích lôgic. Nếu ta đặt mức ngưỡng là 0V, ta sẽ nhận được đường cong thứ hai là tín hiệu có mức điện thế rời rạc ở lối vào của bộ nhớ. Nhưng tín hiệu này chỉ được đặt tới lối vào của bộ nhớ khi hiện diện xung nhịp đồng hồ.



Hình 18.3 Tín hiệu hiện trên dao động kí và màn hình bộ phân tích lôgic.

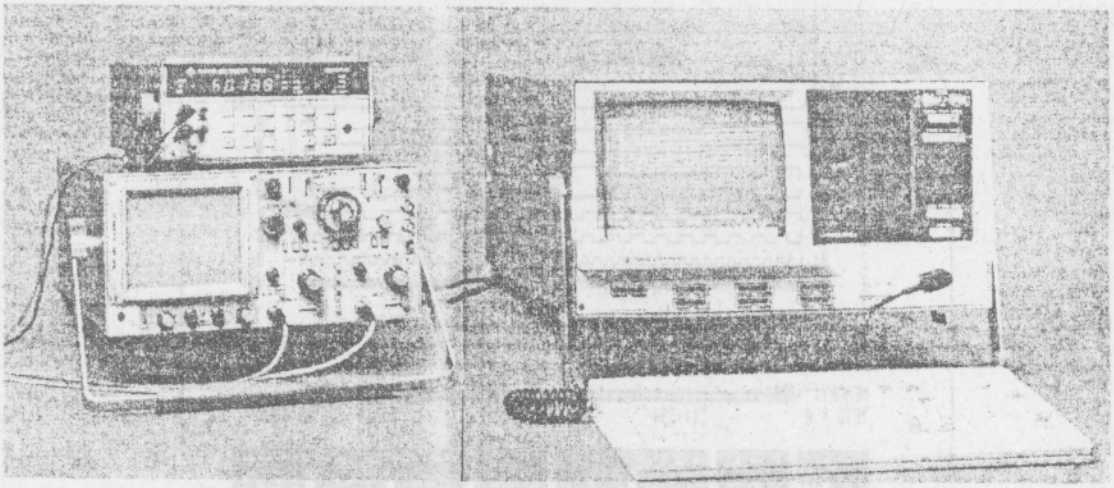


Hình 18.4 Màn hình bộ phân tích lôgic.

Kết quả là một tín hiệu có mức điện thế rời rạc và thời gian cũng rời rạc sẽ được lưu trữ vào bộ nhớ như một chuỗi số liệu "0" và "1". Kết quả các số liệu này được hiện ra trên màn hình của bộ phân tích lôgic như đường cong cuối cùng. Hình 18.4 là hình ảnh kết quả trên màn hình một bộ phân tích lôgic 8 kênh (8 đầu dò).

Có một vài cách để hiển thị các kết quả. Bảng danh sách trạng thái rất hữu ích khi quan sát chuỗi địa chỉ được gửi ra hay chuỗi dữ liệu được vi xử lí đọc. Để xác định liệu đường địa chỉ cụ thể nào đó có bị viết tắt đi không thì có thể chọn hiển thị bảng dưới dạng nhị phân, để có thể quan sát từng bit 1, 0. Tuy nhiên nhìn theo dạng số hệ mười sáu (hex) thì dễ dàng nhận ra máy vi tính có gửi ra địa chỉ theo thứ tự đúng chưa. Một vài bộ phân tích lôgic cho phép lấy một loạt các mẫu từ hệ thống đang hoạt động và lưu các mẫu này trong bộ nhớ thứ hai rồi sau đó so sánh chúng với chuỗi mẫu đo được lấy khi hệ thống không hoạt động. Dạng hiển thị giản đồ theo thời gian là hữu ích nhất khi đo thời gian bằng các xung định thời bên trong. Có thể đo thời gian bằng cách nhân số xung đồng hồ giữa hai sự kiện với chu kì xung. Một vài bộ phân tích lôgic cho phép xác định thời gian giữa hai sự kiện bằng cách đặt con trỏ vào từng sự kiện và đọc thời gian trực tiếp trên màn hình tại hai vị trí con trỏ. Dạng giản đồ rời cho phép xác định liệu một vi xử lí có đang nhận lệnh và thi hành lệnh theo một tuần tự đúng hay không. Để có thể có chế độ hiển thị này thì bộ phân tích lôgic phải có thêm phần cứng và phần mềm cho vi xử lí cụ thể đang làm việc...

Thiết bị phân tích lôgic chuyên dụng rất đắt tiền, nhưng với sơ đồ khối chức năng được mô tả như trên ta có thể tự xây dựng một bộ phân tích lôgic đơn giản với máy vi tính sẵn có trong phòng thí nghiệm cùng các linh kiện điện tử tương tự và số sẵn có tại Việt Nam. Sơ đồ nguyên lí của một bộ phân tích lôgic 8 kênh như vậy cùng phần mềm điều khiển được trình bày ở phần phụ lục cuối sách có thể hoàn toàn tự viết lấy dễ dàng bằng các ngôn ngữ bậc cao như Pascal hoặc hợp ngữ. Hình 18.5 là ảnh chụp các vôn mét điện tử, dao động kí và thiết bị phân tích lôgic. Trong đó bên trái từ trên xuống là vôn mét điện tử và dao động kí 2 kênh vào; bên phải là thiết bị phân tích lôgic gồm hộp máy chứa máy vi tính và màn hình cùng các núm điều khiển, trước nó là bàn phím và các đầu dò lôgic.



Hình 18.5 Hình ảnh các vôn mét, dao động kí và bộ phân tích lôgic.

18.4.2. Sử dụng thiết bị phân tích lôgic để đo các tín hiệu trên bus

Để quan sát tín hiệu trên bus với một dao động kí thông thường là một việc khó vì các tín hiệu này là không tuần hoàn. Một bộ phân tích lôgic (Logic Analyzer) sẽ cho phép quan sát và đo đạc từ 16 đến 64 đường tín hiệu cùng một lúc. Có thể nối các đường tín hiệu vào đầu vào của bộ đo khi dùng các đầu kẹp có vỏ bọc kim. Sau đây là một số phép đo có thể thực hiện được trên bộ phân tích lôgic.

a, Hiển thị một danh sách các địa chỉ

Bước đầu tiên trong sử dụng bộ phân tích lôgic để quan sát tín hiệu của máy vi tính là phải quyết định sẽ quan sát tín hiệu nào và nối đầu vào của bộ đo với các tín hiệu đó. Nếu muốn hiện thị một chuỗi các địa chỉ mà 8086 đã gửi ra khi nó thực hiện một chương trình kiểm tra, thì nối đầu vào của bộ đo với các chân từ AD0-tới AD15 của 8086.

Bước tiếp theo là quyết định tín hiệu nào sẽ là tín hiệu định thời cho bộ phân tích. Để quyết định thì nên nhìn vào dạng xung theo thời gian trong hình 3.2, chương 3 để tìm ra sườn xung mà tại đó xuất hiện tín hiệu địa chỉ ổn định trên đường AD0-AD15. Một tín hiệu có thể dùng để định thời cho bộ đo là tín hiệu CLK của 8086. Tín hiệu này có sườn âm khi tín hiệu địa chỉ ổn định xuất hiện trên AD0-AD15 nhưng tín hiệu này cũng có sườn âm khi các đường AD0-AD15 ở trạng thái treo hay khi có dữ liệu trên các đường này. Nói cách khác, bộ đo khi được định thời bởi tín hiệu CLK này thì chuỗi địa chỉ gồm cả dữ liệu, địa chỉ và tín hiệu thừa và như vậy phải lọc bỏ được chúng.

Một cách lựa chọn tốt hơn cho tín hiệu định thời của bộ đo là tín hiệu ALE của 8086 bởi vì tín hiệu này chỉ xuất hiện khi địa chỉ có trên đường A0- A15. Để sử dụng ALE như một tín hiệu định thời, ta nối đầu vào của bộ đo với chân ALE của 8086. Để xác định xem sườn nào của tín hiệu ALE sẽ kích hoạt bộ đo thì ta sẽ xem chi tiết dạng xung theo thời gian của 8086 trong hình 3.2. Vào thời điểm mà sườn dương của tín hiệu ALE xuất hiện, 8086 vẫn chưa gửi ra địa chỉ vì vậy mà xung định thời kích vào bộ đo tại sườn dương sẽ không chốt lại địa chỉ. Sườn âm của tín hiệu ALE xuất hiện khi địa chỉ đã được đặt cố định trong các đường dây AD0- AD15, vì vậy ta nên đặt xung định thời của bộ đo tại sườn âm của tín hiệu ALE.

Bước cuối cùng là xác định tín hiệu nào sẽ nối vào đầu vào trigger của bộ đo. Bởi ta muốn tạo ra một chuỗi các địa chỉ nên ta chọn tín hiệu trigơ là một tín hiệu bên trong bộ phân tích và đặt *bộ nhận dạng từ* để tạo ra tín hiệu trigơ khi địa chỉ đầu tiên của chương trình xuất hiện tại lối vào dữ liệu của bộ đo. Thí dụ nếu lệnh đầu tiên tại ô nhớ 00100h ta sẽ đặt bộ đo được khởi động khi địa chỉ này xuất hiện. Khi xác định được vị trí của tín hiệu trigơ, khởi động bộ đo để chuỗi địa chỉ sẽ bắt đầu tại địa chỉ đã xác định (00100h).

b, Hiển thị một danh sách các từ dữ liệu

Giả sử muốn tạo một danh sách các từ dữ liệu đọc được từ bộ nhớ khi 8086 đang thực hiện một chương trình thử. Ta nối đầu vào dữ liệu của bộ đo với chân AD0-AD15 của 8086 bởi vì dữ liệu sẽ đi vào bộ đo trên các đường dữ liệu này.

Để xác định tín hiệu nào và sườn xung nào của nó sẽ định thời cho bộ đo thì nhìn kĩ dạng xung theo thời gian trong hình 3.2. Từ giản đồ này ta thấy tín hiệu $\overline{RD}$ được kích hoạt trong một quá trình đọc bộ nhớ vì vậy đây là tín hiệu được nối vào đầu vào Clock ngoài của bộ đo. Tại sườn dương của tín hiệu $\overline{RD}$ thì dữ liệu ổn định xuất hiện trên bus dữ liệu nên đặt thời gian kích hoạt (clock) tại sườn dương. Bởi ta muốn tạo ra một danh sách các từ dữ liệu đọc được từ bộ nhớ nên cần nhìn vào chương trình thử để xác định tín hiệu nào là tín hiệu trigơ. Thí dụ một chương thử đơn giản được đưa vào bộ nhớ và chạy:

```
00100 EB HERE:  JMP HERE        ; Vòng lặp không kết thúc mà không làm gì.
00101   FE
00102   90      NOP             ; Hơn một từ được nhận
00103   90      NOP
00104   04      ADD  AL,55h
```

Bởi 8086 có bus dữ liệu 16 bit nên 8086 có thể đọc một từ (2 byte) một lúc nếu từ bắt đầu ở địa chỉ chẵn. Khi 8086 đọc byte mã cho chương trình, 8086 sẽ gửi ra địa chỉ 00100h và xác lập cả hai tín hiệu A0 và $\overline{BHE}$. Byte EBh sẽ đi vào 8086 trên đường dây AD0-AD7. Byte EFh sẽ đi vào 8086 trên đường dây A8-A15. Dữ liệu đầu tiên đọc từ bộ nhớ là FEEBh nên đây là từ mà ta cho vào bộ nhận dạng từ để kích hoạt (trigơ) bộ đo.

Khi bảng dữ liệu được hoàn thành thì bảng sẽ chỉ ra tuần tự các từ : FEEBh,9090h và 0455h. Phần chương trình duy nhất mà 8086 thực hiện là lệnh HERE:JMP HERE có mã EBh và FEh. Khi 8086 đang giải mã lệnh JMP thì 8086 sẽ nhận mã lệnh tiếp theo và lưu giữ trong hàng nhận lệnh trước sẵn sàng được sử dụng. Tuy nhiên trong chương trình này lệnh JMP chỉ cho 8086 biết phải trở lại nhận và thực hiện lệnh JMP. Số 9090h và 0455h được nhận từ bộ nhớ và lưu trong hàng đợi của 8086 nhưng chúng không bao giờ được thực hiện.

c, Sử dụng đầu vào clock qualifier trong đo đạc

Trong thí dụ tiếp theo sẽ chỉ ra cách tạo một danh sách các dữ liệu mà 8086 đọc từ bộ nhớ. Giả sử rằng ta đang thực hiện một chương trình đọc dữ liệu từ bộ nhớ và dữ liệu từ cổng vào/ra. Nếu chỉ đơn giản định thời (clock) cho bộ đo tại sườn dương của tín hiệu $\overline{RD}$ thì bảng dữ liệu tạo thành sẽ chứa cả dữ liệu đọc từ cổng và dữ liệu đọc từ bộ nhớ. Ta có thể dùng tín hiệu $M/\overline{IO}$ để tạo ra bảng dữ liệu mà chỉ chứa dữ liệu đọc từ bộ nhớ hay dữ liệu đọc từ cổng.

Nhớ lại từ lần trước, khi 8086 viết dữ liệu vào ô nhớ hay đọc dữ liệu từ ô nhớ thì 8086 sẽ kích hoạt tín hiệu $M/\overline{IO}$ lên mức cao. Còn khi 8086 đọc và viết dữ liệu vào cổng thì nó kích hoạt $M/\overline{IO}$ xuống mức thấp. Để tạo ra một bảng các dữ liệu đọc từ cổng thì nối tín hiệu $\overline{RD}$ với lối vào xung nhịp ngoại (external clock) và tín hiệu $M/\overline{IO}$ nối vào Clock Qualifier của bộ đo. Một điểm chính ở đây là nếu đầu vào Clock qualifier được dùng thì bộ đo sẽ đáp ứng lại tín hiệu định thời (clock) chỉ khi lối vào Clock Qualifier có một mức lôgic xác định. Ví dụ nếu muốn bộ đo lấy mẫu tại sườn dương của tín hiệu $\overline{RD}$ và tín hiệu M/ $\overline{IO}$ thấp. Do đó ta chỉ định mức thấp là mức kích hoạt của lối vào clock qualifier. Tuỳ thuộc vào từng loại bộ đo mà mức kích hoạt của lối vào clock qualifier có nhiều cách đặt trước khác nhau- có thể bằng cách đặt trong thực đơn hay bật công tắc trên đầu thử hay nối tín hiệu clock vào lối vào trên đầu kẹp.

Để tạo ra một bảng toàn dữ liệu đọc từ bộ nhớ thì ta định thời tại sườn dương của tín hiệu $\overline{RD}$ và nối tín hiệu M/ $\overline{IO}$ vào đầu vào clock qualifier và chỉ định mức logic kích hoạt của clock qualifier là cao. Một điểm cần chú ý ở đây là cẩn thận chọn tín hiệu clock và tín hiệu qualifier để tạo ra một bảng các dữ liệu bạn muốn.

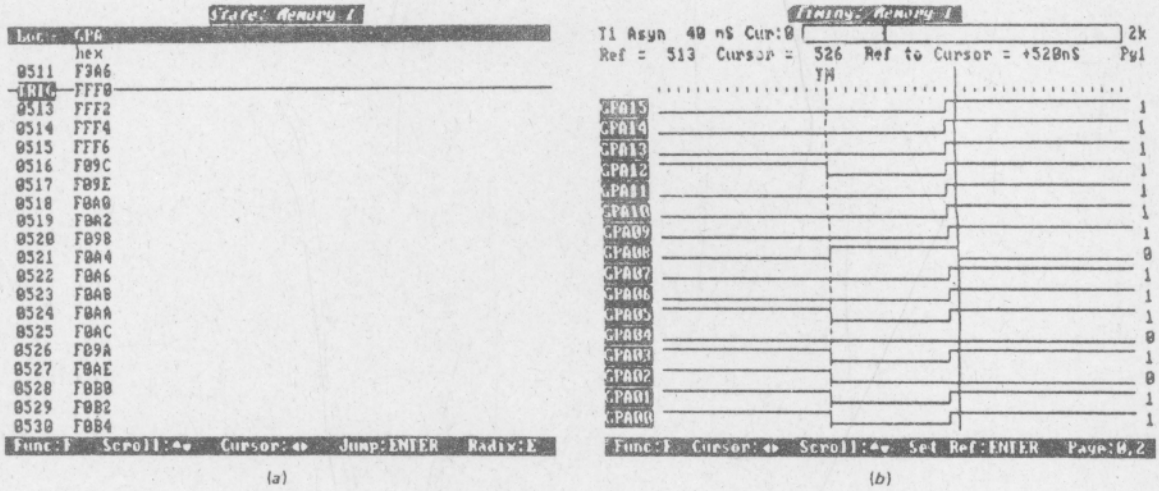
d, Đo thời gian truy cập bộ nhớ

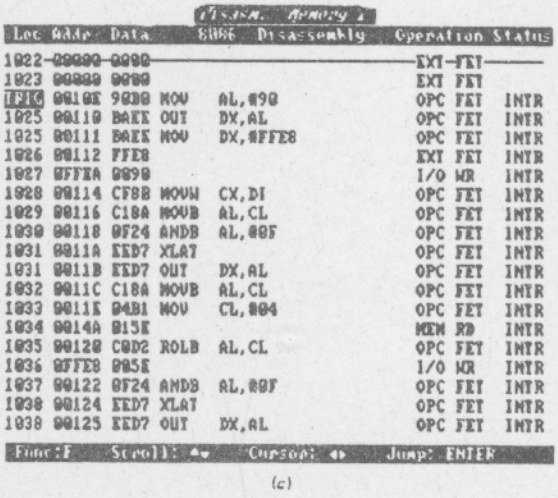
Như đã chỉ ra ở giản đồ xung theo thời gian trong hình 3.2, thời gian truy xuất bộ nhớ là khoảng thời gian cần thiết để bộ nhớ đặt dữ liệu ổn định lên đầu ra sau khi bộ nhớ nhận được địa chỉ ở đầu vào. Chỉ cần suy nghĩ một chút là có thể biết cách dùng bộ đo để xác định thời gian truy cập bộ nhớ thực sự của hệ thống.

Bước đầu tiên là viết và chạy chương trình liên tục đọc dữ liệu từ bộ nhớ mong muốn. Ví dụ như ta sử dụng một chương trình giống như ví dụ trước:

```
00100 EB HERE: JMP HERE        ; Vòng lặp không kết thúc mà không làm gì.
00101    FE
00102    90        NOP         ; Hơn một từ được nhận
00103    90        NOP
00104    04        ADD AL,55h
```

Bước tiếp theo là tìm xem tín hiệu nào nối với các đầu vào của bộ đo. Xác định khoảng thời gian giữa địa chỉ hợp lệ từ 8086 và địa chỉ hợp lệ từ bộ nhớ, ta phải quan sát tín hiệu trên đường ADR/DATA. Số con số có thể hiển thị phụ thuộc từng loại bộ đo cụ thể. Bộ đo Tektronix 1230 sẽ lấy mẫu và hiển thị 16 lối vào trong mode hiển thị theo thời gian- nếu dùng mode này để đo. Nếu có thiết bị này thì nối các chân AD0-AD15 của 8086 với các đầu vào dữ liệu của bộ đo. Bốn đường địa chỉ cao A16-A19 không thay đổi trong khi thực hiện thí dụ này nên không cần phải xem xét đến chúng. Để đo thời gian bằng bộ phân tích thì dùng tín hiệu định thời gian của máy phát sóng thạch anh để chỉ cho bộ đo biết khi nào lấy mẫu và biết chính xác khoảng thời gian giữa các lần lấy mẫu. Để chọn từ bắt đầu cho quá trình đo thì nhìn lại hình 3.2. Địa chỉ đi ra trên bus dữ liệu trước, sau đó dữ liệu đi vào cũng trên bus dữ liệu. Bởi vì gửi địa chỉ là hoạt động đầu tiên nên đặt bộ nhận dạng từ trong bộ đo được bắt đầu khi địa chỉ đầu tiên được gửi ra.





Hình 18.6 Ba kiểu hiển thị kết quả trên màn hình bộ phân tích lôgic:

a) Liệt kê trạng thái, chuỗi các địa chỉ ra sau khi RESET.

b) Hiện đồ thị theo thời gian, địa chỉ ra của 8086 và số liệu ra từ RAM.

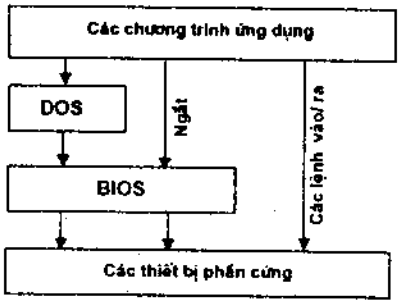
c) Liệt kê rời rạc, một phần của chương trình trên kit đào tạo về máy vi tính SDK-86 của hãng Intel.

Bởi vì ta tạo ra một bảng các dữ liệu nên có thể tính thời gian truy cập bộ nhớ bằng cách đếm số mẫu giữa địa chỉ 0100h và địa chỉ FEEBh. Nếu bộ phân tích lôgic có con trỏ thì có thể đặt con trỏ tại thời điểm mà địa chỉ là hợp lệ và đặt con trỏ kia tại một thời điểm khác và đọc khoảng thời gian giữa hai thời điểm này trên màn hình. Chú ý rằng chu kì lấy mẫu chỉ là cỡ, thí dụ, 40ns. Nói cách khác, không có hoạt động nào xảy ra giữa hai điểm lấy mẫu được hiển thị lên màn hình cho đến khi lần lấy mẫu tiếp theo. Trong nhiều bộ đo ta có thể giảm chu kì lấy mẫu bằng cách giảm số đường tín hiệu được đo. Với thiết bị Tektronix 1230 thì ta có thể đặt chu kì lấy mẫu 10ns nếu chỉ lấy mẫu cho 4 đường tín hiệu. Thường thì ban đầu ta chọn lấy mẫu cả 16 đường sau đó giảm còn 4 đường để giảm chu kì lấy mẫu.

Chúng ta không thể mô tả hết các cách sử dụng bộ phân tích lôgic ở đây. Hãy cẩn thận trong việc chọn tín hiệu cần quan sát, tín hiệu định thời, từ khởi phát đo. Hình 18.6 là các kiểu hiển thị trên màn hình bộ phân tích lôgic.

CHƯƠNG 19. CÁC CÔNG CỤ PHẦN MỀM

Các chương trình kiểm tra máy vi tính thường được viết ra trên cơ sở phân tích cấu trúc của máy vi tính và các yêu cầu của việc ghép nối nó với các thiết bị ngoại vi. Các chương trình này thâm nhập các bộ phận phần cứng qua các *ngắt* của BIOS, các ngắt của hệ điều hành hay thâm nhập trực tiếp qua các *lệnh truy nhập cổng vào/ra* như cách mô tả trong hình 19.1



Hình 19.1 Ba cách thâm nhập phần cứng.

Hợp ngữ cũng như các ngôn ngữ bậc cao đều cho phép gọi các chương trình con phục vụ ngắt để giao tiếp với các bộ phận phần cứng. Tùy mục đích và đối tượng cụ thể như kiểm tra loại thiết bị gì, tốc độ hoạt động của nó nhanh hay chậm v.v... mà các chương trình ứng dụng được viết trên các ngôn ngữ bậc cao như C, Pascal, Basic hay hợp ngữ. Hiện trên thị trường cũng có sẵn các gói phần mềm tiện ích dùng cho các mục đích này.

19.1. GỌI NGẮT TRONG CÁC NGÔN NGỮ

Các chương trình con phục vụ ngắt được nạp sẵn trong ROM BIOS hoặc DOS là các công cụ mạnh cho phép người sử dụng giao tiếp với các bộ phận phần cứng của máy vi tính.

19.1.1. Gọi ngắt trong hợp ngữ

Các ngắt được gọi bằng lệnh **INT** kèm theo số ngắt trong hợp ngữ. Trước đó các thanh ghi đa năng thường được nạp các giá trị của hàm ngắt (chức năng ngắt).

Thí dụ, gọi ngắt số 33h, hiện con trỏ chuột trong BIOS:

```
MOV  AX,01h        ; nạp giá trị 01h của hàm hiện con trỏ chuột
INT 33h            ; gọi ngắt 33
```

19.1.2. Gọi ngắt trong Pascal

Để gọi một ngắt, Turbo Pascal cung cấp thủ tục INTR, nó được gọi với các tham số sau:

- intr (NumberInterrupt: byte, Regs : Register);
- Number Interrupt là số ngắt. Do vậy nó có thể là số bất kì trong số từ 0-255
- Thủ tục MSDOS là một dạng đặc biệt của INTR:
- MSDOS (Regs: Register);

Số ngắt ở đây bị bỏ qua vì nó tự động gọi ngắt 21h mà trong nó chứa hầu hết các hàm của hệ điều hành.

Hai thủ tục trên có một tham số kiểu Register. Tham số này nhận các giá trị dành cho các thanh ghi của bộ xử lí trước khi gọi các thủ tục INTR và MSDOS. Các giá trị này sẽ được nạp vào các thanh ghi tương ứng trước khi gọi ngắt. Kiểu Register được định nghĩa như sau:

```
Type Registers = record
case integer of
        0: (AX, BX, CX, DX, BP, SI, DI, DS, ES, Flag : word);
        1: (AL, AH, BL, BH, CL, CH, DL, DH : byte);
end;
```

Khi biến Regs được định nghĩa bằng lệnh

```
var Regs :Registers
```

Kiểu này gồm các thành phần: Regs.ax Regs.bx ...

Cũng thí dụ như trên, ta có thể viết đoạn chương trình gọi ngắt trong Turbo Pascal:

```
var regs. : registers;
        Begin
        . . .
        regs.ax:=$01;   { gán giá trị hàm 01h cho thanh ghi AX }
        intr ($33,regs);  { gọi ngắt 33h }
        . . .
        end;
```

19.1.3. Gọi ngắt trong C

Các chương trình C cung cấp các hàm gọi ngắt như sau:

```
int86    int86x    intdos  intdosx    segread
```

Các hàm này cũng như các cấu trúc dữ liệu tương ứng được định nghĩa, khai báo trong file DOS.h. Mỗi khi chương trình muốn dùng một trong các hàm trên, nó buộc phải ghép file này vào bằng lệnh #include của bộ tiền xử lí.

Để truyền giá trị các thanh ghi, trong DOS.h định nghĩa 3 cấu trúc:

WORDREGS chứa các thanh ghi 16 bit AX, BX, CX, DX, SI, DI và cờ Carry.
BYTEREGS chứa các thanh ghi AH, BH, CH, DH, AL, BL, CL, DL.
SEGREGS chứa các thanh ghi đoạn DS, CS, ES, SS.

Các cấu trúc BYTEREGS và WORDREGS được kết hợp với nhau trong union REGS. Điều này cho phép thao tác hoặc là với các thanh ghi 8 bit, hoặc là với các thanh ghi 16 bit. Mỗi khi một biến kiểu REGS được tạo ra bằng lệnh: union REGS ta có thể truy nhập đến các thanh ghi theo cách sau: AX: Register.x.ax BX: Register.xbx ...

Cờ nhớ (carry) được biểu diễn dưới dạng một biến Register.x.cflag. Nếu biến này bằng 0, có nghĩa cờ nhớ bằng 0. Mọi giá trị khác thông báo cờ này bằng 1.

Trong trường hợp các thanh ghi đoạn, một biến SREGS được tạo ra như sau:

```
struct SREGS SegRegister;
```

Các thành phần của biến này : SegRegister.ds , SegRegister.es ... tương ứng với các thanh ghi đoạn của vi xử lí.

Các hàm gọi ngắt được bắt đầu bằng từ "int' còn hàm SegRead dùng để lấy nội dung hiện thời của các thanh ghi đoạn. Trước khi gọi ngắt, nội dung của các biến được nạp vào các thanh ghi tương ứng của bộ xử lí, sau khi thực hiện ngắt, nội dung của các thanh ghi này được copy vào các biến ra.

- Hàm int86 được gọi như sau:

```
int86 (NumberInt, RegisterIn, RegisterOut);
```

NumberInt là một biến hoặc một hằng, nó cho biết số ngắt cần gọi. RegisterIn, RegisterOut là địa chỉ của hai biến (hoặc chỉ một biến) kiểu Regs. Như trên, RegisterIn chứa nội dung các thanh ghi trước khi gọi ngắt và RegisterOut chứa nội dung các thanh ghi sau khi gọi ngắt.

Hàm int86x cần thêm tham số thứ ba là địa chỉ của một biến kiểu SREGS. Nội dung của biến này được copy vào các thanh ghi đoạn trước khi gọi ngắt, nhưng giá trị của các thanh ghi này không được copy vào biến sau khi gọi ngắt. Hàm này được gọi như sau:

```
int86x (NumberIn, RegisterIn, RegisterOut, RegistersSeg);
```

- Các hàm intdos và intdosx khác hai hàm trên ở chỗ chúng không cần số ngắt vì chúng tự động gọi ngắt 21h của DOS:

```
intdos (RegisterIn, RegisterOut);
intdosx (RegisterIn, RegisterOut, RegistersSeg);
```

Việc lấy địa chỉ của biến dùng toán tử địa chỉ &. Toán tử này cho phép tính địa chỉ offset của một biến. Hàm SEGREAD nhận địa chỉ của một biến kiểu SREGS. Hàm này sẽ copy nội dung của các thanh ghi đoạn vào biến này.

Thí dụ,

- Chương trình phụ đưa kí tự ra màn hình thông qua hàm số 9 của ngắt 21h trong DOS:

```
# include <dos.h>          /* Ghép file dos.h */
void main ()
{ static char PC () = "\ n This is PC\ n$";
union REGS Register;              /* biến thanh ghi để gọi ngắt */
Register.h.ah = 9;          /* hàm 09h hiện chuỗi kí tự */
Regster.x.dx = (int) PC           /* địa chỉ text của PC */
}
indos (&Register, &Register);     /* Gọi ngắt 21h trong DOS */
```

- Đoạn chương trình gọi hàm 01h ngắt 33h hiện con trỏ chuột trong PC:

```
int  mouse_show (void)
{
Union REGS inregs, outregs;
   inregs.x.ax = 1;
   int86 (0x33, &inregs, &outregs);
}
```

19.2. THÂM NHẬP TRỰC TIẾP CÁC CỔNG VÀO/RA

19.2.1. Các lệnh truy nhập cổng vào/ra

Các thiết bị ngoại vi thường được định địa chỉ trong không gian vào/ra. Việc truy xuất các địa chỉ đó có thể được thực hiện trực tiếp bằng các lệnh đọc hoặc viết từ/tới các cổng vào/ra.

Phần mềm điều khiển các thiết bị ngoại vi này thường được phát triển trên ngôn ngữ bậc thấp, thí dụ như hợp ngữ, do yêu cầu chính để đảm bảo tốc độ giao tiếp. Thời gian gần đây, với các thế hệ PC mới có khả năng tính toán xử lí dữ liệu rất nhanh cho phép người dùng có thể phát triển các phần mềm ghép nối trực tiếp bằng các ngôn ngữ bậc cao như Pascal, C. Các lệnh vào/ra cho phép truy nhập các cổng trực tiếp không qua các ngắt của DOS hoặc BIOS. Đó là các lệnh được viết bằng hợp ngữ hoặc ngôn ngữ bậc cao. Sau đây là liệt kê cú pháp các lệnh đó:

- ***Đọc một byte từ cổng:***
  - *Hợp ngữ:* IN AL,DX

    trong đó DX là thanh ghi số liệu chứa địa chỉ của cổng vào. Giá trị 8 bit cần đọc từ cổng được nạp thanh ghi AL.

  - *Pascal:* Gía trị := port [địa chỉ cổng];

    thí dụ, x:= port [$300h] ; {gán giá trị thanh ghi đệm số liệu cổng 300h vào biến x (byte) }
  - *C:* Gía trị = inportb (địa chỉ cổng);

    thí dụ, x= inport (0x300) ; {gán giá trị thanh ghi đệm số liệu cổng 300h vào biến x (byte) }
- ***Đọc một từ (word) từ cổng:***
  - *Hợp ngữ:* IN AX,DX

    trong đó DX là thanh ghi số liệu chứa địa chỉ của cổng vào. Thanh ghi AX chứa giá trị từ 16 bit cần đọc.
  - *Pascal:* gía trị := portw [địa chỉ cổng];
  - *C:* giá trị = inport (địa chỉ cổng);
- ***Viết một byte ra cổng:***
  - *Hợp ngữ:* OUT DX,AL

    trong đó DX là thanh ghi số liệu chứa địa chỉ của cổng ra. Thanh ghi AL chứa giá trị byte 8 bit cần viết ra cổng.

    Thí dụ, viết số 89h ra cổng 300h

```
MOV AL, 89h    ; nạp số 89h vào thanh ghi AL
MOV DX, 300h   ; nạp địa chỉ cổng vào thanh ghi DX
OUT DX,AL      ; xuất số 89h ra cổng 300h
```

  - *Pascal:* port [địa chỉ cổng]:= gía trị;

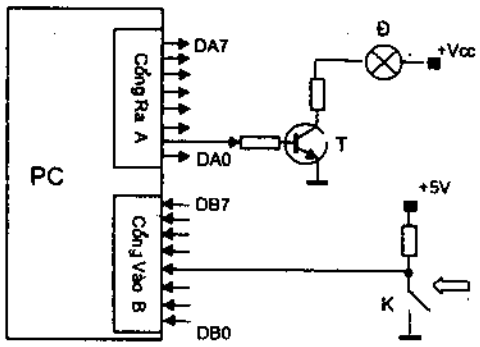
    Thí dụ như trên, port [$300] := $89;
  - *C:* outportb (địa chỉ cổng, giá trị);

    Thí dụ như trên, outportb(0x300, 0x89);
- ***Viết một từ ra cổng:***
  - *Hợp ngữ:* OUT DX,AX

    trong đó DX là thanh ghi số liệu chứa địa chỉ của cổng ra. Thanh ghi AX chứa giá trị từ 16 bit cần viết ra cổng.
  - *Pascal:* portw [địa chỉ cổng]:= giá trị;
  - *C:* outport (địa chỉ cổng, giá trị);

19.2.2. Các phép xử lí bit

Quá trình khảo sát các thiết bị phần cứng máy vi tính hoặc quá trình ghép nối nó với các thiết bị ngoại vi liên quan tới việc điều khiển các *bit ra* hoặc đọc trạng thái của các *bit vào* trên các thanh ghi của một cổng hoặc một từ nhớ. Lí do là vì các bit đó được dùng để điều khiển hoạt động của một thiết bị nào đó (khi xuất dữ liệu ra) hay được dùng để lưu trữ trạng thái hiện tại của một ô nhớ hoặc thiết bị ngoại vi nào đó (khi đọc dữ liệu vào). Thí dụ như bài toán đặt ra ở hình 19.2. Cổng A gồm 8 bit ở chế độ ra trong đó dùng bit DA1 để điều khiển đèn Đ các bit còn lại dùng cho mục đích khác; B là cổng vào 8 bit trong đó bit DB3 dùng để kiểm tra trạng thái của công tắc K. Yêu cầu là phải viết đoạn chương trình bật sáng đèn Đ mỗi khi công tắc K được nhấn.



Hình 19.2 Xử lí bit.

Bit D1 cổng A cần được đặt ở mức 1 mỗi khi bit D3 cổng B ở mức thấp. Nhìn trên sơ đồ ta thấy, mỗi khi công tắc K được nhấn, đường dây nối nó với bit DB3 trên cổng vào B sẽ ở mức thấp. Bật sáng đèn Đ chính là điều khiển cho transistor T ở trạng thái thông, tức là đưa bit DA1 của cổng A có đầu nối dây tới cực Base của transistor T lên mức cao. Bài toán ở đây quy về việc đọc liên tục byte số liệu được chốt tại cổng B để kiểm tra trạng thái bit DB3 cho tới khi nó bằng 0 thì viết ra cổng A một byte số liệu sao cho bit DA1 bằng 1 mà không làm ảnh hưởng tới các bit còn lại trên cổng.

Tóm lại, việc đặt hay xoá (trong chế độ *ra*) cũng như kiểm tra giá trị hiện tại của một hoặc vài bit nhất định (trong chế độ *vào*) của các ô nhớ, các thanh ghi của vi xử lí hoặc các thanh ghi của một cổng vào/ra mà không làm ảnh hưởng đến trạng thái hiện tại của các bit còn lại trong từ dữ liệu là một điều cần phải xử lí. Các phép xử lí bit cho phép ta thực hiện được điều đó trong các lệnh chương trình.

- ***Toán tử bitwise*** giống như các toán tử lôgic AND, OR, XOR và NOT nhưng toán tử bitwise tác động với từng cặp bit tương ứng của hai toán hạng. Bảng chân lí của các toán tử bitwise với 2 số nhị phân 1 bit cũng giống như với 2 biến lôgic nhận một trong hai giá trị True (1) và False (0).

Thí dụ: bitwise 2 số nhị phân A = 1011 0011b và B = 0101 1011b

```
AND   A = 1 0 1 1 0 0 1 1          OR    A = 1 0 1 1 0 0 1 1
      B = 0 1 0 1 1 0 1 1                B = 0 1 0 1 1 0 1 1
          ---------------                    ---------------
          0 0 0 1 0 0 1 1                    1 1 1 1 1 0 1 1
```

- ***Đặt (hay xoá) một bit trên thanh ghi, ô nhớ hoặc cổng ra (Out):*** Việc đặt (=1) hay xoá (=0) một bit mà không làm ảnh hưởng đến các bít còn lại được thực hiện bằng

việc bitwise giá trị byte (hay từ) với ***từ bitmask*** (từ này có tác dụng để "che" các bit còn lại) trên cơ sở 2 nhận xét sau:

Một bit nhị phân bất kì AND với 0 sẽ cho kết quả bằng 0 và AND với 1 sẽ cho kết quả bằng chính nó.

Một bit nhị phân bất kì OR với 1 sẽ cho kết quả bằng 1 và OR với 0 sẽ cho kết quả bằng chính nó.

* Vậy khi muốn đặt một bit thứ i của cổng n bit lên "1" ta thực hiện theo hai bước:

1- Đọc giá trị hiện tại của cổng (không cần quan tâm đến kết quả bằng bao nhiêu).

2- Xuất ra cổng giá trị bằng giá trị đó OR với từ *bitmask* bằng 2^i (tức là một số nhị phân có giá trị 1 ở bit thứ i, các bit còn lại bằng 0).

Thí dụ, câu lệnh Pascal đặt bit 5 của cổng 8 bit có địa chỉ 300h lên "1":

```
x := port[$300];              { gán giá trị đọc được từ 300h cổng vào biến x kiểu byte }
port[$300] := x OR $20;       { 20h = 0010 0000b  ; xuất ra cổng }
```

* Khi muốn xoá một bit thứ i của cổng n bit về "0" ta cũng thực hiện theo hai bước. Bước 1 như trên nhưng bước 2 là xuất ra cổng giá trị bằng giá trị hiện tại AND với từ *bitmask* bằng $(2^n - 1) - 2^i$. Nói cách khác, *bitmask* là một số nhị phân có giá trị 0 ở bit thứ i, các bit còn lại bằng 1.

Thí dụ, câu lệnh Pascal xoá bit 5 của cổng 8 bit có địa chỉ 300h về "0":

```
x := port[$300];              { gán giá trị cổng vào biến x byte }
Port[$300] := x AND $DF ;     { DFh = 1101 1111b }
```

• ***Xác định giá trị của một bit trên cổng vào (In)***

Cũng dựa vào 2 nhận xét trên có thể dùng các toán tử *bitwise* với các *từ bitmask* để "che" các bit còn lại trong việc xác định giá trị của bit cần kiểm tra. Muốn xác định giá trị của bit thứ i trong cổng vào n bit ta AND với từ bitmask có giá trị bằng 2^i (tức là từ có bit thứ i bằng 1, các bit còn lại bằng 0). Nếu kết quả bằng 0 thì bit thứ i bằng 0 và ngược lại nếu kết quả khác 0 (bằng 2^i) thì bit thứ i bằng 1.

Thí dụ, xác định giá trị của bit 5 trên cổng vào 8 bit có địa chỉ 301h.

```
If (port[$301] AND $20 = 0) then write( ' Bit thứ 5 bằng 0 ' )
 else write( ' Bit thứ 5 bằng 1 ' ) ;          { 20h = 0010 0000 }
```

Ngoài ra còn có thể vận dụng các toán tử AND, OR và toán tử dịch phải SHR theo các cách khác để đạt được các mục đích kể trên cũng như xử lí cho nhiều hơn một bit trên cổng.

• Các phép toán xử lí bit trong vi xử lí còn có: dịch trái / phải SHL / SHR ; phép quay trái / quay phải ROL / ROR; phép Và hai toán hạng TEST v.v..

Quay trở lại giải bài toán đặt ra ở hình 19.2 ta sẽ viết đoạn chương trình như sau:

```
Port[<địa chỉ thanh ghi từ điều khiển>] := <từ điều khiển> ;
                        { xác lập chế độ hoạt động của các cổng:  A ra, B vào, . . . }
Repeat
```

```
  x:= port[< địa chỉ cổng B>];
Until (x AND $08 = 0);                    {đợi đến khi bit DB3 ="0"}

Port[ <địa chỉ cổng A>] := port[<địa chỉ cổng A>] OR $02;

                                  {nâng bit DA1 lên "1" mà không ảnh hưởng đến các bit khác}
```

19.3. CÁC CHƯƠNG TRÌNH DÙNG CHO CHẨN ĐOÁN

Các chương trình dùng cho mục đích khảo sát, kiểm tra (chẩn đoán) cấu hình phần cứng cũng như cho phép sửa sai (gỡ rối) các chương trình khác hiện cũng có sẵn trên thị trường. Ta chỉ đề cập một cách tóm tắt ở đây một vài chương trình thông dụng giúp cho những người quan tâm đến các vấn đề về hệ thống máy giảm nhẹ đi việc lập trình cụ thể.

19.3.1. Chương trình gỡ rối Debug

Debug.com là một chương trình đặc biệt trong DOS, rất tiện lợi cho việc khảo sát, gỡ rối và sửa sai các chương trình khác từng byte một. Nó có thể được dùng để kiểm tra hoặc thay đổi các dữ liệu của một tập tin bất kì. Khi chạy chương trình Debug, màn hình sẽ hiện con trỏ nháy là dấu nhắc cho biết Debug sẵn sàng nhận lệnh đánh vào. Các lệnh của Debug đều bắt đầu bằng một kí tự từ A đến Z không phân biệt chữ hoa hay thường. Kí tự này được xem như là mã tác vụ (op code) của một lệnh Debug. Thí dụ, A hoặc a là lệnh dịch trong bộ nhớ các lệnh hợp ngữ . Đa số các lệnh Debug đều bao gồm một số thông số đi sau kí tự mã tác vụ. Những thông số này thường là số Hex. Hai thông số liên tiếp được phân cách bằng một kí tự trống (blank) hoặc bằng dấu phẩy.

Sau đây là bảng liệt kê các lệnh của debug.

Lệnh	Chức năng
A [địa chỉ]	Assembler: dịch trong bộ nhớ các lệnh hợp ngữ từ <địa chỉ>
C [dải địa chỉ] [địa chỉ]	Compare: So sánh hai vùng nhớ từng byte một
D [dải địa chỉ]	Dump: Hiển thị lên màm hình nội dung một vùng nhớ
E [địa chỉ] [liệt kê dữ liệu]	Enter: Thay thế một vùng nhớ bởi giá trị ghi trong bảng liệt kê các giá trị byte và/hoặc chuỗi.
F [dải địa chỉ] [liệt kê dữ liệu]	Fill: điền một vùng nhớ bởi dữ liệu ghi trong bảng liệt kê các byte và/hoặc chuỗi
G [= địa chỉ] [địa chỉ ..]]	Go: chạy chương trình từ <địa chỉ> hiện trong bộ nhớ
H [gía trị 1] [gía trị 2]	Hexarithmetic: tính cộng và trừ 2 số thập lục
I [địa chỉ]	Input: nhận và hiện thị ở dạngHex một byte ở cổng vào có <địa chỉ>
L [địa chỉ]	Load: nạp một tập tin hay các cung từ đĩa vào bộ nhớ
M [dải địa chỉ] [địa chỉ]	Move: chép khối dữ liệu trong bộ nhớ
N [tên file]	Name: đặt tên tập tin để dùng với lệnh Load và Write
O [địa chỉ] [gía trị byte]	Output: viết 1 byte ra cổng có <địa chỉ>
P [= địa chỉ] [giá trị]	Proceed: thi hành một vòng lặp, hoặc một chương trình con, hoặc thi hành một ngắt mềm, hoặc theo rõi các lệnh kế tiếp
Q	Quit: thoát khỏi Debug về DOS
R [thanh ghi]	Register: hiện thị nội dung của các thanh ghi và các cờ
S [dải địa chỉ] [liệt kê dữ liệu]	Search: tìm trong bộ nhớ chuỗi dữ liệu khai báo bởi <liệt kê>
T [= địa chỉ] [gía trị]	Trace: chạy một lệnh đồng thời hiện thị nội dung các thanh ghi, các cờ hiện tại và dạng giải mã chỉ thị mà Debug sẽ cho chạy tiếp

U[dải địa chỉ]	Unassemble: dịch ngược lệnh trong bộ nhớ (mã máy) ra lệnh hợp ngữ (mnemmonic)
W [địa chỉ] [ổ đĩa] [đầutừ] [cung từ]	Write: viết tập tin đang được Debug xử lí lên đĩa từ
XA [số]	eXpanded memory Allocation: cấp vùng nhớ phân trang EMS
XD [đầu vào]	eXpanded memory Deallocation: làm mất tác dụng bộ mô tả đoạn bộ nhớ phân trang.
XM [trangL] [trangP] [đầu vào]	eXpanded memory Mapping: định bản đồ một <trang lôgic L> của bộ nhớ phân trang lên một khung.<trang vật lí P> của bộ nhớ phân trang
XS	eXpanded memory Status: hiện thị trạng thái vùng nhớ phân trang

Có thể mô tả tóm tắt các nhóm lệnh trong Debug như sau:

1. Nhóm lệnh biên soạn chương trình

• **A [địa chỉ]**. Cho phép nhập các lệnh hợp ngữ vào mỗi [địa chỉ] ô nhớ.

Thông số [địa chỉ] ghi vị trí bắt đầu của các lệnh dạng hợp ngữ của chương trình bằng chữ số thập lục 4 chữ số. Địa chỉ có thể là dạng CS:OFFSET hay chỉ có OFFSET (ngầm định trong thanh ghi CS hiện tại).

Thí dụ: A 200 ↵ hay A 0200 ↵

Màn hình sẽ hiện ra địa chỉ ô nhớ xxxx : 0200 để ta gõ tên lệnh vào địa chỉ này. Thí dụ, gõ MOV AX, D2 ↵. Sau khi nhấn Enter (↵) dòng lệnh này sẽ được dịch ra mã máy, ghi vào ô nhớ có địa chỉ trên và màn hình hiện tiếp địa chỉ nối tiếp sau ở dòng tiếp theo để chờ nhập lệnh mới.

• **E [địa chỉ] [danh sách]** ↵ Nhập hay sửa mã của ô nhớ.

Địa chỉ có thể đầy đủ dạng CS:OFFSET hay chỉ có OFFSET (giá trị đoạn ngầm định trong thanh ghi CS hiện tại). [danh sách] chỉ các số liệu muốn đặt vào các byte nhớ liên tiếp trong bộ nhớ. Nếu có danh sách các số liệu cần ghi vào nhiều ô nhớ liên tiếp thì địa chỉ là địa chỉ bắt đầu cuả danh sách nhiều byte nhớ.

Thí dụ; E ↵ thì nội dung có địa chỉ xxxx : 00100 hiện ra. E 25 ↵ thì nội dung có địa chỉ xxxx : 0025 hiện ra cho phép nhập số liệu mới vào đó.

• **F [dải địa chỉ] [danh sách]** ↵ Điền đầy các ô nhớ có địa chỉ trong [dải địa chỉ] bằng các giá trị ghi trong [danh sách].

Thí dụ, F 100 L 100 42 45 52 54 41 ↵ sẽ ghi đầy vùng nhớ có địa chỉ ban đầu DS:100 đến địa chỉ DS:1FF (100 địa chỉ) bằng các số có giá trị 42, 45, 52, 54, 41.

F 04BA: 100 100 42 45 52 54 41 ↵ sẽ ghi đầy 100h byte nhớ có địa chỉ bắt đầu là 04BA:0100 lần lượt bởi các mã 42, 45, 52, 54, 41.

• **U [dải địa chỉ]** ↵ . Dịch ngược từ mã máy sang dạng hợp ngữ 32 byte bắt đầu từ địa chỉ đầu tiên của [dải địa chỉ].

Thí dụ, dịch ngược 32 byte từ địa chỉ CS:IP.

```
U 100 ↵
149D:0100  B089          MOV   AL,89
149D:0102  BA0003        MOV   DX,0300
149D:0105  EE            OUT   DX,AL
149D:0106  0000          ADD   [BX+SI],AL
```

Trong đoạn nhớ hiện tại 149D, ta thấy các mã máy B089 đã được dịch ra thành lệnh MOV AL,89 hay nói cách khác lệnh này dài 2 byte; nằm ở địa chỉ offset 0100 và 0101. Trong khi đó lệnh OUT DX,AL chỉ dài 1 byte nằm ở địa chỉ 0105 v.v...

2. Nhóm lệnh di chuyển dữ liệu

- **I [cổng]** ↵ . Đọc và hiển thị giá trị byte ở địa chỉ [cổng].

Thí dụ, I 2F8 ↵ . Đọc giá trị 8 bit của cổng có địa chỉ 2F8h.

- **O [cổng] [giá trị]** ↵ . Viết byte [giá trị] ra cổng có địa chỉ [cổng].

Thí dụ, O 61 3A ↵ . Xuất số 3Ah ra cổng có địa chỉ 061h.

- **L [địa chỉ]** ↵ . Ghi nội dung của một số byte đã chỉ định trong các thanh ghi BX:CX của một tệp từ đĩa vào bộ nhớ bắt đầu từ [địa chỉ]. Nếu không ghi [địa chỉ] thì địa chỉ mặc định là CS:0100.
- **W [địa chỉ]** ↵ . Ghi vào đĩa nội dung số byte được chỉ định trong các thanh ghi BX:CX.
- **N [đĩa:][đường dẫn] tệp** ↵ . Đặt tên cho tệp sắp được nạp hoặc ghi.

Thí dụ,

- Đặt tên file UIB.com trong thư mục C:\DOS\ gồm 9 địa chỉ nhớ và ghi lên đĩa:

```
n C:\DOS\UIB.com ↵
r CX ↵
: 09 ↵
w ↵
```

- Ghi 10 séctơ trong bộ nhớ bắt đầu từ địa chỉ CS:BX vào séctơ thứ 200 của đĩa.

```
w 0000 1000  200          10
  ↓    ↓     ↓            ↓
  BX   CX  séctơ 200   số séctơ
```

- **M [dải địa chỉ] [địa chỉ]** ↵. Chuyển nội dung vùng nhớ có địa chỉ thuộc [dải địa chỉ] (địa chỉ đầu địa chỉ cuối) sang vùng nhớ có địa chỉ đầu là [địa chỉ].

Thí dụ: Chuyển vùng nhớ có địa chỉ đầu DS:0100 đến DS:02FE sang vùng nhớ có địa chỉ đầu là 3000:0000.

```
m 100 2FE 3000:0000 ↵
```

- **R [thanh ghi]** ↵ . Hiển thị nội dung của các thanh ghi vi xử lí.

Thí dụ, hiển thị nội dung của mọi thanh ghi, kể cả thanh ghi cờ.

- R ↵

```
AX=0000  BX=0000  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=149D  ES=149D  SS=149D  CS=149D  IP=0100   NV UP EI PL NZ NA PO NC
149D:0100 1332          ADC     SI,[BP+SI]
   SS:0000=20CD
```

Các bit cờ được chỉ thị trong Debug như sau:

Tên cờ	Set (đặt)	Clear (xoá)
Overflow (yes / no) - cờ Tràn	OV	NV
Direction (decrement / increment) - cờ Hướng tăng/giảm	DN	UP

Interrupt (enable / disable) - cờ Ngắt	EI	DI
Sign (negative / positive) - cờ Dấu. âm/dương	NG	PL
Zero (yes / no) - cờ Số không	ZR	NZ
Auxiliary carry (yes / no) - cờ Phụ	AC	NA
Parity (even / odd) - cờ Chẵn Lẻ	PE	PO
Carry (yes / no) - cờ Nhớ	CY	NC

- R CX ↵ hiển thị và cho phép thay đổi nội dung thanh ghi CX.

• **D [địa chỉ]** ↵ . Hiển thị 128 byte nhớ bắt đầu từ địa chỉ [địa chỉ].
hoặc **D [dải địa chỉ]** ↵ . In vùng nhớ trong [dải địa chỉ] ra màn hình.

Thí dụ: Cho hiện lên nội dung các ô nhớ trong vùng dài 128 byte từ ô nhớ có địa chỉ offset 100. Màn hình sẽ hiện lên các thông tin như sau: đó là 8 hàng bắt đầu từ *địa chỉ đoạn hiện tại: offset* (ở đây là 1695:100). Mỗi hàng chia làm 3 phần như sau: phần bên trái là địa chỉ lôgic ô nhớ đầu tiên của hàng (thí dụ, 16950:0100), phần giữa biểu thị 16 byte nhớ chia làm 2 nhóm 8 byte cách nhau bởi dấu gạch ngang; phần bên phải là các mã kí tự ASCII của các byte tương ứng trong phần giữa (các byte không tương ứng với các kí tự hiển thị được sẽ được thay bằng dấu chấm).

```
-d 100 L80
1695:0100  54 72 75 6F 6E 67 20 44 - 61 69 20 68 6F 63 20 51   Truong Dai hoc Q
1695:0110  75 6F 63 20 67 69 61 20 - 48 61 20 6E 6F 69 0D 0A   uoc gia Ha noi..
1695:0120  4B 68 6F 61 20 43 6F 6E - 67 20 6E 67 68 65 0D 0A   Khoa Cong nghe..
1695:0130  44 69 65 6E 20 74 75 20 - 56 69 65 6E 20 74 68 6F   Dien tu Vien tho
1695:0140  6E 67 0D 0A 48 65 6C 6C - 6F 20 21 20 57 65 6C 63   ng..Hello ! Welc
1695:0150  6F 6D 65 20 21 0D 0A D9 - E3 C7 06 CF E2 DD E4 01   ome !...........
1695:0160  06 CD E2 01 06 CF E2 C3 - 53 8B 1E D3 E2 80 BF D8   ........S.......
1695:0170  E3 5C 74 12 80 BF D8 E3 - 3A 74 0B C6 87 D9 E3 5C   .\t.....:t.....\
```

• **C [dải địa chỉ] [địa chỉ]** ↵ So sánh vùng nhớ 1 có dải địa chỉ [dải địa chỉ] với vùng nhớ 2 có địa chỉ đầu là [địa chỉ]. Nếu kết quả không in gì lên màn hình thì 2 vùng nhớ đó giống nhau. Nếu có khác nhau thì debug in ra màn hình các ô nhớ khác nhau.

Thí dụ, So sánh vùng nhớ có địa chỉ từ DS:0100 tới DS:01EF với vùng nhớ có địa chỉ đầu là 4000:0000.

```
c 100 1EF 4000:0000
```

• **H [giá trị 1] [giá trị 2]** ↵ Tính và in kết quả của hai phép tính cộng và trừ 2 số [giá trị 1] [giá trị 2] lên màn hình. Kết quả trước là tổng, kết quả sau là hiệu. Các giá trị bao gồm giữa 0 và FFFFh.

• **T [=địa chỉ][giá trị]** ↵ Thực hiện từng lệnh một, bắt đầu từ địa chỉ [địa chỉ]. Sau mỗi lệnh nội dung các thanh ghi sẽ được hiện lên màn hình. Thông số [giá trị] xác định số liệu cần thực hiện nên gọi là lệnh duyệt. Nếu không có [địa chỉ] và [giá trị] thì chỉ thực hiện 1 lệnh ở địa chỉ xác định bởi nội dung thanh ghi CS:IP.

• **P [= địa chỉ] [giá trị]** ↵ Thực hiện một vòng lặp, một lệnh chuỗi được lặp lại, một ngắt lôgic hay một chương trình con hoặc theo dõi một lệnh. [địa chỉ] là địa chỉ của lệnh đầu tiên được thi hành Nếu không ghi thì giá trị ngầm định là CS:IP. [giá trị] chỉ số lệnh cần thực hiện trước khi trả sự điều khiển cho debug theo ngầm định là 1.

• **G [= địa chỉ][điểm dừng]** ↵ Chạy chương trình từ [địa chỉ] đến [điểm dừng]. Dấu = để phân biệt giữa địa chỉ khởi đầu và điểm dừng.

19.3.2. Chương trình MEM.EXE trong DOS

Khảo sát việc sử dụng bộ nhớ trong máy tính có thể được thực hiện bằng lệnh MEM.EXE trong DOS. Nếu chạy lệnh không có tuỳ chọn nào ta sẽ được một bảng tổng kết nhanh về tình trạng bộ nhớ của máy tính như thí dụ dưới đây. Từ bảng liệt kê này ta có thể thấy rằng bộ nhớ tổng cộng (Total) là 16MB trong máy và chỉ có chừng 1,5MB là hiện còn trống (Free). Trong số 640 kB nhớ của bộ nhớ quy ước, chương trình chạy trên DOS lớn nhất là 517 kB; vùng nhớ trên cũng không còn trống v.v...

Memory Type	Total	= Used	+ Free
Conventinal	640KB	123KB	517KB
Upper	139KB	139KB	0KB
Reserved	384KB	384KB	0KB
Extended (XMS)	15,221KB	14,197KB	1,024KB
Total Memory	16,384KB	14,843KB	1,541KB

Total Under 1MB	779KB	262KB	517KB
Largest executable program size		517KB	(528,896 bytes)
Largest free upper memory block			0KB (0bytes)

MS-DOS is resident in the high memory area.

Các tuỳ chọn:

• /C(lassify) chương trình sẽ cho thấy có bao nhiêu bộ nhớ trong mỗi loại đang được dùng bởi mỗi chương trình.

• /D(ebug) phân tích kích thước và vị trí trong bộ nhớ của mỗi thành phần được nạp, loại của mỗi thành phần đó (chẳng hạn như "device driver" hoặc "program" hoặc phiên bản của trình quản lí mở rộng, v.v...

• /M(odule) <tên chương trình> cho phép hiển thị chương trình đó được đặt ở các địa chỉ nào trong bộ nhớ.

• /F(ree) chỉ rõ những vùng nhớ còn trống và kích thước của chúng trong bộ nhớ.

19.3.3. Các chương trình kiểm tra hệ thống trong DOS và Windows

♦ ***Chương trình MSD.exe*** trong DOS cho phép hiển thị các thông tin kĩ thuật của hệ thống máy vi tính với những đề mục sau:

- Model and Processor - Kiểu và loại vi xử lí
- Memory - Bộ nhớ
- Video Adapter - Màn hình
- Version of MS-DOS - Phiên bản của DOS
- Mouse - Con chuột
- Other Adapter - Các bộ ghép nối khác

- Disk Drive - Các ổ đĩa
- LPT port - Các cổng song song LPT
- COM port - Các cổng nối tiếp COM
- IRQ status - Các trạng thái ngắt
- Terminale-and-stay-resident (TSR) program - Các chương trình thường trú
- Device drivers - Các chương trình quản lí thiết bị.

Nếu sử dụng màn hình, MSD sẽ hiện lên các menu gồm các mục chọn như trên. Từng mục được chọn sẽ cho thấy những thông tin chi tiết.

♦ Các chương trình trong phần mềm Norton Utility, QAPlus Avanced Diagno-stics v.v.. cũng cho phép hiển thị các thông số của hệ thống và các trình tiện ích cho bảo dưỡng sửa chữa đĩa như: Chương trình Sysinfo.exe, Diskedit.exe, NDD.exe, Disktool.exe v.v...

♦ ***Nếu máy tính dùng hệ điều hành Windows*** thì đã có sẵn những công cụ để thông báo tình trạng của hệ thống. Khi lần theo các đề mục sau:

Start → Setting → Control Panel → System → Device Manager

Windows sẽ thông báo cho người dùng các thông tin tỉ mỉ về các thiết bị ngoại vi và tình trạng của chúng.

PHỤ LỤC 1. CÁC LỆNH MÁY TRONG 8086

1. CÁC LỆNH TRUYỀN DỮ LIỆU

- **MOV <đích>,<nguồn>** (chuyển một từ hay một byte dữ liệu)

Mô tả thao tác: đích ← nguồn

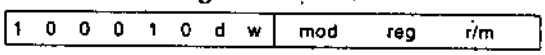
Toán hạng đích và nguồn phải có cùng độ dài và không được phép đồng thời là 2 ô nhớ hoặc 2 thanh ghi đoạn. Lệnh này không tác động đến các cờ.

Các thí dụ:
```
MOV AL,86          ; AL ← 86
MOV CL,BL          ; CL ← BL
MOV DL,[SI]        ; DL ← {DS:SI}
MOV AL,Table[BX]   ; AL ← {DS: (Table+BX)}
```

(ở đây kí hiệu {xxx} là ô nhớ có địa chỉ là nội dung của xxx).

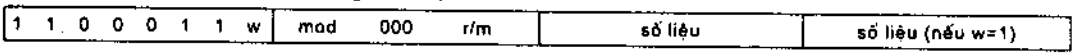
Mã máy và số nhịp đồng hồ trong các trường hợp như sau:

Giữa thanh ghi và bộ nhớ



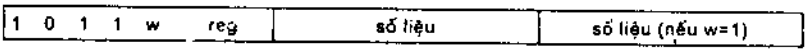
Số nhịp đồng hồ:
- thanh ghi tới thanh ghi: 2 nhịp
- bộ nhớ tới thanh ghi: 8 + nhịp EA
- thanh ghi tới bộ nhớ: 9 + nhịp EA.

Giá trị tức thời tới thanh ghi / bộ nhớ



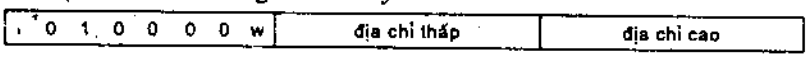
Số nhịp đồng hồ: 10 + nhịp EA.

Giá trị tức thời tới thanh ghi



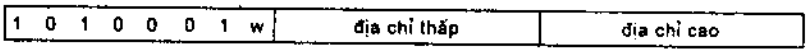
Số nhịp đồng hồ: 4.

Bộ nhớ tới thanh ghi tích luỹ



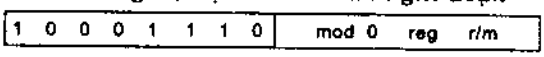
Số nhịp đồng hồ: 10.

Thanh ghi tích luỹ tới bộ nhớ



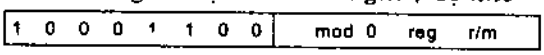
Số nhịp đồng hồ: 10.

Thanh ghi / bộ nhớ tới thanh ghi đoạn



Số nhịp đồng hồ:
- thanh ghi tới thanh ghi: 2
- bộ nhớ tới thanh ghi: 8 + nhịp EA.

Thanh ghi đoạn tới thanh ghi / bộ nhớ



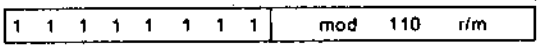
Số nhịp đồng hồ:
- thanh ghi tới thanh ghi: 2
- thanh ghi tới bộ nhớ: 9 + nhịp EA.

- **PUSH <nguồn>** (cất một từ vào ngăn xếp)

Mô tả thao tác: SP ← SP - 2 (con trỏ ngăn xếp giảm 2)
nguồn → {SP}
ở đây, {SP} là kí hiệu ô nhớ ngăn xếp có địa chỉ là nội dung của thanh ghi SP.

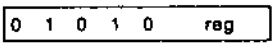
Thí dụ:
```
PUSH  BX              ; cất nội dung BX vào ngăn xếp tại vị trí do
                        con trỏ ngăn xếp PS chỉ ra.
PUSH  Table [BX]      ; cất 2 byte của vùng dữ liệu DS có địa chỉ
                        đầu tại (Table +BX).
```

Nguồn là thanh ghi / bộ nhớ



Số nhịp đồng hồ: thanh ghi 10
bộ nhớ 16 + nhịp EA.

Nguồn là thanh ghi



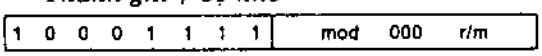
Số nhịp đồng hồ: 10.

- **POP <đích >** (hồi phục lại một từ vào thanh ghi từ đỉnh ngăn xếp)

Mô tả thao tác: đích ← {SP}
SP ← SP +2 (con trỏ ngăn xếp tăng 2)

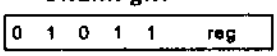
Thí dụ:
```
POP  DX               ; lấy 2 byte từ đỉnh ngăn xếp nạp vào DX
POP  Table [BX]       ; lấy 2 byte ở đỉnh ngăn xếp nạp vào đoạn DS
                        có địa chỉ đầu tại Table +BX.
```

Thanh ghi / bộ nhớ



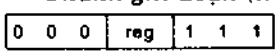
Số nhịp đồng hồ: thanh ghi 8
bộ nhớ 17 + nhịp EA.

Thanh ghi



Số nhịp đồng hồ: 8.

Thanh ghi đoạn (trừ CS)



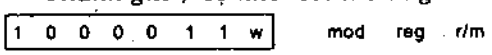
Số nhịp đồng hồ: 8.

- **XCHG <đích>,<gốc>** (chuyển đổi nội dung 2 toán hạng)

Mô tả thao tác: đích ↔ nguồn.

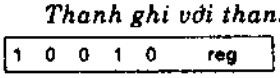
Thí dụ:
```
XCHG  AH,AL           ; tráo nội dung của AH và AL.
```

Thanh ghi / bộ nhớ với thanh ghi



Số nhịp đồng hồ: thanh ghi với thanh ghi 4
bộ nhớ với thanh ghi 17 + nhịp EA.

Thanh ghi với thanh ghi tích luỹ



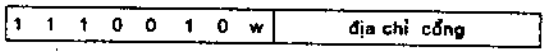
Số nhịp đồng hồ: 3.

- **IN <thanh ghi tích luỹ>,<địa chỉ cổng>** (đọc dữ liệu từ cổng vào AL hoặc AX)

Mô tả thao tác: thanh ghi tích luỹ ←{dữ liệu tại cổng}.

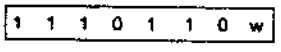
Thí dụ: IN AL,80 ; đọc dữ liệu từ cổng 80h vào AL
IN AX,DX ; đọc dữ liệu từ cổng có địa chỉ là nội dung của DX (thường dùng khi địa chỉ cổng lớn hơn 255d = FFh).

Địa chỉ cổng nhỏ hơn FFh



Số nhịp đồng hồ: 10.

Địa chỉ cổng lớn hơn FFh, thanh ghi DX chứa địa chỉ cổng



Số nhịp đồng hồ: 8.

- **OUT <địa chỉ cổng>, <thanh ghi tích luỹ>** (xuất 1 byte hay từ ra cổng)

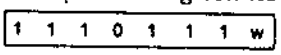
Mô tả thao tác: thanh ghi tích luỹ→ {dữ liệu tại địa chỉ cổng}.

Thí dụ: OUT 78,AL ; viết nội dung thanh ghi AL ra cổng 78h
OUT DX,AL ; viết nội dung AL ra cổng có địa chỉ là nội dung của DX (thường dùng khi địa chỉ cổng lớn hơn 255d = FFh).

Địa chỉ cổng nhỏ hơn FFh

Số nhịp đồng hồ: 10.

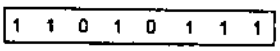
Địa chỉ cổng lớn hơn FFh, thanh ghi DX chứa địa chỉ cổng



Số nhịp đồng hồ: 8.

- **XLAT** (đổi nội dung của AL theo bảng cho trước)

Mô tả thao tác: AL ← {AL + BX}



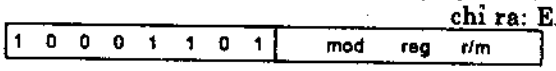
Số nhịp đồng hồ: 11.

- **LEA <đích>,<nguồn>** (nạp địa chỉ hiệu dụng EA tới thanh ghi)

trong đó: đích là một trong các thanh ghi BX, CX, DX, BP, SI, DI
nguồn là tên biến trong đoạn DS được chỉ ra trong lệnh hoặc ô nhớ.

Mô tả thao tác: đích ← địa chỉ offset (hoặc hiệu dụng) của nguồn
lệnh không tác động đến các cờ.

Thí dụ: LEA CX, [BX][DI] ; nạp vào CX địa chỉ hiệu dụng do BX và DI chỉ ra: EA = BX + DI.



Số nhịp đồng hồ: 2 + nhịp EA

- **LES <đích>,<nguồn>** (nạp một từ, từ bộ nhớ, vào thanh ghi cho trong lệnh và từ tiếp theo vào ES)

trong đó: đích là một trong các thanh ghi AX, BX, CX, DX, SP, BP, SI, DI
nguồn là ô nhớ trong đoạn DS được chỉ ra trong lệnh.

Mô tả thao tác: đích ← nguồn, ES ← nguồn + 2.

Thí dụ: LES DI,[BX] ; nạp vào DI nội dung 2 ô nhớ do BX và BX+1 chỉ ra và nạp vào ES nội dung 2 ô nhớ tiếp theo do BX+2 và BX+3 chỉ ra. Các ô nhớ này đều nằm trong đoạn dữ liệu DS và chứa địa chỉ của chuỗi đích. Sau đó ES:DI chỉ vào đầu chuỗi đích cần thao tác.

1	1	0	0	0	1	0	1	mod	reg	r/m

Số nhịp đồng hồ: 16 + nhịp EA.

- **LAHF** (nạp byte thấp của thanh ghi cờ vào AH)

Mô tả thao tác: AH ← byte thấp thanh ghi cờ.

1	0	0	1	1	1	1	1

Số nhịp đồng hồ: 4.

- **SAHF** (cất nội dung thanh ghi AH vào byte thấp thanh ghi cờ)

Mô tả thao tác: byte thấp thanh ghi cờ ← AH.

1	0	0	1	1	1	1	0

Số nhịp đồng hồ: 4.

- **PUSHF** (cất thanh ghi cờ vào ngăn xếp)

Mô tả thao tác: SP ← SP - 2
RF → {SP}.

1	0	0	1	1	1	0	0

Số nhịp đồng hồ: 10.

- **POPF** (hồi phục dữ liệu thanh ghi cờ từ đỉnh ngăn xếp)

Mô tả thao tác: RF ← {SP}
SP → SP + 2.

1	0	0	1	1	1	0	1

Số nhịp đồng hồ: 8.

2. CÁC LỆNH TÍNH SỐ HỌC

- **ADD <đích>,<nguồn>** (cộng 2 toán hạng)

Mô tả thao tác: đích ← đích + nguồn.

Thanh ghi/ bộ nhớ với thanh ghi

0	0	0	0	0	0	d	w	mod	reg	r/m

Số nhịp đồng hồ:

thanh ghi tới thanh ghi	3
bộ nhớ tới thanh ghi	9 + nhịp EA
thanh ghi tới bộ nhớ	16 + nhịp EA.

Giá trị tức thời tới thanh ghi / bộ nhớ

1	0	0	0	0	0	s	w	mod	000	r/m	số liệu	số liệu (nếu sw = 01)

Số nhịp đồng hồ:

tới thanh ghi	4
tới bộ nhớ	17 + nhịp EA.

Giá trị tức thời từ thanh ghi / bộ nhớ

1	0	0	0	0	0	s	w	mod	101	r/m	số liệu	số liệu (nếu sw = 01)

Số nhịp đồng hồ:

từ thanh ghi	4
từ bộ nhớ	17 + nhịp EA.

Giá trị tức thời từ thanh ghi tích luỹ

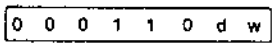
0	0	1	0	1	1	0	w	số liệu	số liệu (nếu w=1)

Số nhịp đồng hồ: 4.

- **SBB <dich>,<nguồn>** (trừ có mượn)

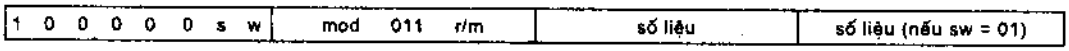
Mô tả thao tác: đích ← đích - nguồn - cờ nhớ CF.

Thanh ghi / bộ nhớ và thanh ghi



Nhịp đồng hồ: thanh ghi trừ thanh ghi 3
thanh ghi trừ bộ nhớ 9 + nhịp EA
bộ nhớ trừ thanh ghi 16 + nhịp EA.

Tức thời từ thanh ghi / bộ nhớ



Nhịp đồng hồ: tới thanh ghi 4
tới bộ nhớ 17 + nhịp EA.

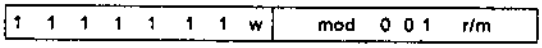
Tức thời từ thanh ghi tích luỹ

Nhịp đồng hồ: từ thanh ghi 4

- **DEC <đích>** (giảm toán hạng đích đi 1)

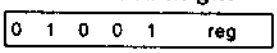
Mô tả thao tác: đích ← đích – 1.

Giảm thanh ghi / ô nhớ



Số nhịp đồng hồ: từ thanh ghi 2
từ ô nhớ 15 + nhịp EA.

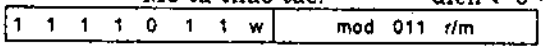
Giảm thanh ghi



Số nhịp đồng hồ: 2.

- **NEG <đích>** (Đảo dấu một toán hạng)

Mô tả thao tác: đích ←0 - (đích).



Số nhịp đồng hồ: thanh ghi 3
bộ nhớ 16 + nhịp EA.

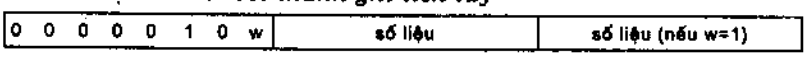
- **CMP <đích>, <nguồn>** (so sánh 2 byte hay 2 từ)

Mô tả thao tác đích – nguồn.

Thanh ghi / bộ nhớ và thanh ghi

Nhịp đồng hồ: thanh ghi với thanh ghi 3
bộ nhớ với thanh ghi 9 + nhịp EA
thanh ghi với bộ nhớ 9 + nhịp EA.

Giá trị tức thời với thanh ghi tích luỹ



Nhịp đồng hồ: từ thanh ghi 4.

- **ADC <đích>, <nguồn>** (cộng có nhớ)

Mô tả thao tác: đích ← đích + nguồn + CF.

Thanh ghi /bộ nhớ với thanh ghi

0 0 0 1 0 0 d w	mod reg r/m

Số nhịp đồng hồ: thanh ghi tới thanh ghi 3
bộ nhớ tới thanh ghi 9 + nhịp EA
thanh ghi tới bộ nhớ 16 + nhịp EA.

Giá trị tức thời tới thanh ghi / bộ nhớ

1 0 0 0 0 0 s w	mod 000 r/m	số liệu	số liệu (nếu sw = 01)

Số nhịp đồng hồ: tới thanh ghi 4
tới bộ nhớ 17 + nhịp EA.

Giá trị tức thời tới thanh ghi tích luỹ

0 0 1 0 1 1 0 w	số liệu	số liệu (nếu w=1)

Số nhịp đồng hồ: 4.

- **INC <đích>** (Tăng toán hạng đích lên 1)

Mô tả thao tác: đích ← đích + 1.

Tăng thanh ghi / bộ nhớ

1 1 1 1 1 1 1 w	mod 000 r/m

Số nhịp đồng hồ: thanh ghi 2
bộ nhớ 15 + nhịp EA.

Tăng thanh ghi

0 1 0 0 0 reg

Số nhịp đồng hồ: 2.

- **AAA** (chỉnh sau khi cộng 2 số ở dạng ASCII)

0 0 1 1 0 1 1 1

Số nhịp đồng hồ: 4.

- **DAA** (chỉnh AL sau khi cộng số BCD)

0 0 1 0 0 1 1 1

Số nhịp đồng hồ: 4.

- **SUB <đích>,<nguồn>** (trừ 2 toán hạng)

Mô tả thao tác: đích ← đích – nguồn.

Thanh ghi / bộ nhớ và thanh ghi

0 0 1 0 1 0 d w	số liệu	số liệu (nếu w=1)

Số nhịp đồng hồ: trừ thanh ghi từ thanh ghi 3
trừ bộ nhớ từ thanh ghi 9 + nhịp EA
trừ thanh ghi từ bộ nhớ 16 + nhịp EA.

Giá trị tức thời từ thanh ghi / bộ nhớ

1 0 0 0 0 0 s w	mod 111 r/m	số liệu	số liệu (nếu w=1)

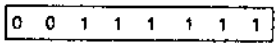
Số nhịp đồng hồ: trừ giá trị tức thời từ thanh ghi 4
trừ giá trị tức thời từ bộ nhớ 17 + nhịp EA.

Giá trị tức thời từ thanh ghi tích luỹ

0 0 1 1 1 1 0 w	số liệu	số liệu (nếu w=1)

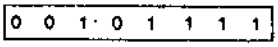
Số nhịp đồng hồ: 4.

- **AAS** (chỉnh sau khi trừ 2 số hạng ở dạng ASCII)



Số nhịp đồng hồ: 4.

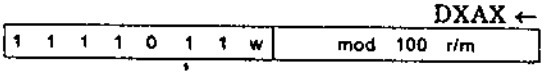
- **DAS** (Chỉnh AL sau khi trừ 2 số BCD)



Số nhịp đồng hồ: 4.

- **MUL <nguồn>** (nhân số không dấu)

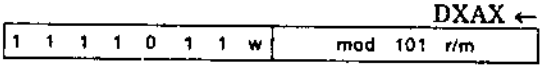
Mô tả thao tác: AX ← AL x <nguồn> nếu nguồn là số 8 bit
DXAX ← AX x <nguồn> nếu nguồn là số 16 bit.



Số nhịp đồng hồ: số 8 bit 71 + nhịp EA
số 16 bit 124 + nhịp EA

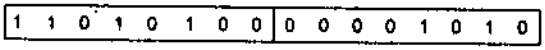
- **IMUL <nguồn>** (nhân số có dấu)

Mô tả thao tác: AX ← AL x <nguồn> nếu nguồn là số 8 bit
DXAX ← AX x <nguồn> nếu nguồn là số 16 bit.



Số nhịp đồng hồ: số 8 bit 90 + nhịp EA
số 16 bit 144 + nhịp EA.

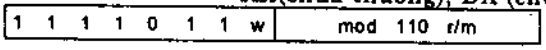
- **AAM** (chỉnh sau khi nhân 2 số ở dạng ASCII)



Nhịp đồng hồ: 83.

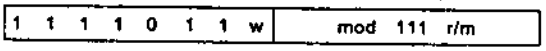
- **DIV <nguồn>** (chia 2 số không có dấu)

Mô tả thao tác: AL(chứa thương), AH (chứa số dư) ← AX / nguồn (số 8 bit)
AX(chứa thương), DX (chứa số dư) ← DXAX / nguồn (số 16 bit).



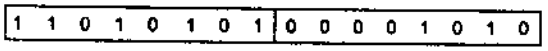
Số nhịp đồng hồ: số 8 bit 90 + nhịp EA
số 16 bit 155 + nhịp EA.

- **IDIV <nguồn>** (chia số có dấu)



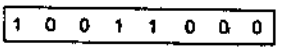
Số nhịp đồng hồ: số 8 bit 112 + nhịp EA
số 16 bit 177 + nhịp EA.

- **AAD** (chỉnh trước khi chia 2 số ở dạng ASCII)



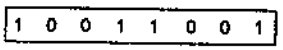
Số nhịp đồng hồ: 60.

- **CBW** (chuyển byte thành từ)



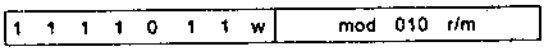
Số nhịp đồng hồ: 2.

- **CWD** (chuyển từ thành từ kép)



Số nhịp đồng hồ: 5.

- **NOT <dích>** (đảo bit của một toán hạng)

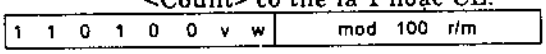


Số nhịp đồng hồ: thanh ghi 3
bộ nhớ 16 + nhịp EA.

- **SHL <dích>,<Count>/ SAL <dích>,<Count>** (Dịch trái số học / lôgic)

<Đích> có thể là thanh ghi hoặc ô nhớ

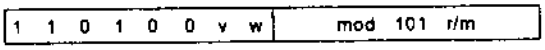
<Count> có thể là 1 hoặc CL.



Số nhịp đồng hồ: thanh ghi bit đơn 2
bộ nhớ bit đơn 15 + nhịp EA
thanh ghi bit biến đổi 8 + 4/bit
bộ nhớ bit biến đổi 20 + nhịp EA + 4/bit.

- **SHR<dích>,<Count>** (dịch phải lôgic)

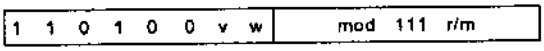
<Đích> có thể là thanh ghi hoặc ô nhớ
<Count> có thể là 1 hoặc CL.



Số nhịp đồng hồ: thanh ghi bit đơn 2
bộ nhớ bit đơn 15 + nhịp EA
thanh ghi bit biến đổi 8 + 4/bit
bộ nhớ bit biến đổi 20 + nhịp EA + 4/bit.

- **SAR <dích>,<Count>** (dịch phải số học)

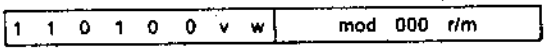
<Đích> có thể là thanh ghi hoặc ô nhớ
<Count> có thể là 1 hoặc CL.



Số nhịp đồng hồ: thanh ghi bit đơn 2
bộ nhớ bit đơn 15 + nhịp EA
thanh ghi bit biến đổi 8 + 4/bit
bộ nhớ bit biến đổi 20 + nhịp EA + 4/bit.

- **ROL <dích>, <count>** (Quay vòng trái)

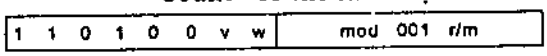
<Đích> có thể là thanh ghi hoặc ô nhớ
<Count> có thể là 1 hoặc CL.



Số nhịp đồng hồ: thanh ghi bit đơn 2
bộ nhớ bit đơn 15 + nhịp EA
thanh ghi bit biến đổi 8 + 4/bit
bộ nhớ bit biến đổi 20 + nhịp EA + 4/bit.

- **ROR <dích>, <count>** (quay vòng phải)

<Đích> có thể là thanh ghi hoặc ô nhớ
<Count> có thể là 1 hoặc CL.

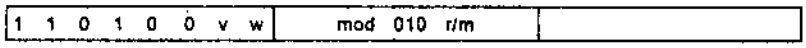


Số nhịp đồng hồ: thanh ghi bit đơn 2
bộ nhớ bit đơn 15 + nhịp EA
thanh ghi bit biến đổi 8 + 4/bit
bộ nhớ bit biến đổi 20 + nhịp EA + 4/bit.

- **RCL <đích>, <count>** (quay trái thông qua cờ nhớ)

<Đích> có thể là thanh ghi hoặc ô nhớ

<Count> có thể là 1 hoặc CL.

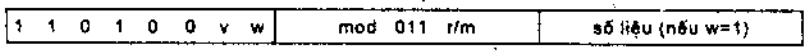


Số nhịp đồng hồ:

thanh ghi bit đơn	2
bộ nhớ bit đơn	15 + nhịp EA
thanh ghi bit biến đổi	8 + 4/bit
bộ nhớ bit biến đổi	20 + nhịp EA + 4/bit.

- **RCR <đích>, <count>** (quay phải thông qua cờ nhớ)

<Đích> có thể là thanh ghi hoặc ô nhớ
<Count> có thể là 1 hoặc CL.



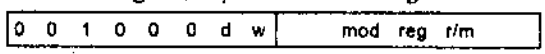
Số nhịp đồng hồ:

thanh ghi bit đơn	2
bộ nhớ bit đơn	15 + nhịp EA
thanh ghi bit biến đổi	8 + 4/bit
bộ nhớ bit biến đổi	20 + nhịp EA + 4/bit.

- **AND <đích>,<gốc>** (và 2 toán hạng)

Mô tả thao tác: đích ← đích ^ nguồn.

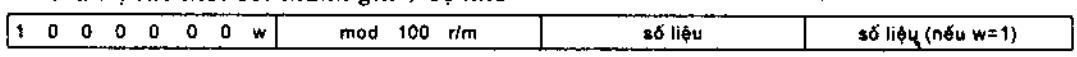
Thanh ghi / bộ nhớ và thanh ghi



Số nhịp đồng hồ:

thanh ghi tới thanh ghi	3
bộ nhớ tới thanh ghi	9 + nhịp EA
thanh ghi tới bộ nhớ	16 + nhịp EA.

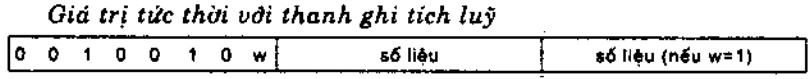
Giá trị tức thời với thanh ghi / bộ nhớ



Số nhịp đồng hồ:

tức thời tới thanh ghi	4
tức thời tới bộ nhớ	17 + nhịp EA.

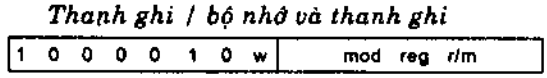
Giá trị tức thời với thanh ghi tích luỹ



Số nhịp đồng hồ: 4

- **TEST <đích>, <nguồn>** (và 2 toán hạng để tạo cờ)

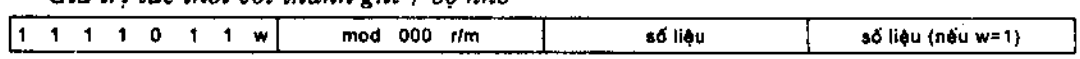
Thanh ghi / bộ nhớ và thanh ghi



Số nhịp đồng hồ:

thanh ghi với thanh ghi	3
thanh ghi với bộ nhớ	9 + nhịp EA.

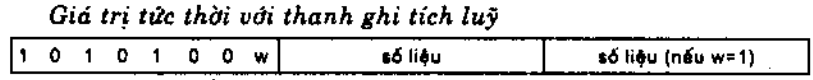
Giá trị tức thời với thanh ghi / bộ nhớ



Số nhịp đồng hồ:

với thanh ghi	2
với bộ nhớ	15 + nhịp EA.

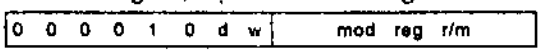
Giá trị tức thời với thanh ghi tích luỹ



Số nhịp đồng hồ: 4.

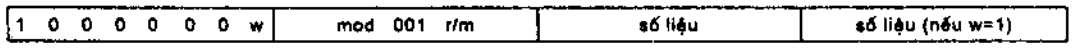
• **OR <đích>, <nguồn>** (hoặc 2 toán hạng)

Thanh ghi / bộ nhớ và thanh ghi



Số nhịp đồng hồ: thanh ghi với thanh ghi 3
bộ nhớ với thanh ghi 9 + nhịp EA
thanh ghi với bộ nhớ 16 + nhịp EA.

Giá trị tức thời với thanh ghi / bộ nhớ



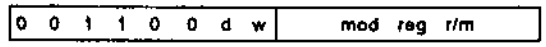
Số nhịp đồng hồ: với thanh ghi 4
với bộ nhớ 17 + nhịp EA.

Giá trị tức thời với thanh ghi tích luỹ

Số nhịp đồng hồ: 4.

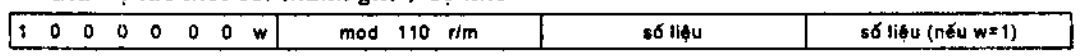
• **XOR <đích>, <nguồn>** (hoặc tuyệt đối 2 toán hạng)

Thanh ghi / bộ nhớ và thanh ghi



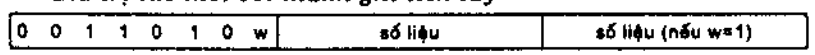
Số nhịp đồng hồ: thanh ghi với thanh ghi 3
bộ nhớ với thanh ghi 9 + nhịp EA
thanh ghi với bộ nhớ 16 + nhịp EA.

Giá trị tức thời với thanh ghi / bộ nhớ



Số nhịp đồng hồ: với thanh ghi 4
với bộ nhớ 17 + nhịp EA.

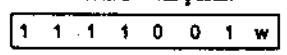
Giá trị tức thời với thanh ghi tích luỹ



Số nhịp đồng hồ: 4.

3. CÁC LỆNH XỬ LÍ XÂU

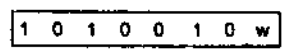
• **REP <Lệnh>** (lặp lại lệnh viết sau đó cho tới khi CX=0)



Số nhịp đồng hồ: 6 cho một vòng lặp.

• **MOVS <chuỗi đích>,<chuỗi nguồn>**

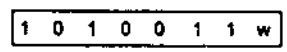
(chuyển một phần tử của một chuỗi sang một chuỗi khác)



Số nhịp đồng hồ: 17.

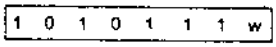
• **CMPS <chuỗi đích>, <chuỗi nguồn>**

(so sánh 2 chuỗi byte hay 2 chuỗi từ)



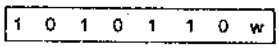
Số nhịp đồng hồ: 22.

- **SCAS <chuỗi đích>** (quét chuỗi byte hay chuỗi từ)



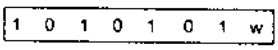
Số nhịp đồng hồ: 15.

- **LODS <chuỗi nguồn>** (nạp vào AL/AX một phần tử của chuỗi byte/từ)



Số nhịp đồng hồ: 12.

- **STOS <chuỗi đích)** (cất AL/AX vào một phần tử của chuỗi byte/từ)

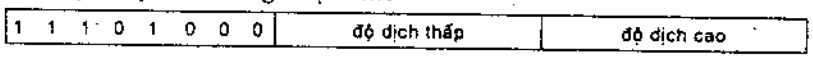


Số nhịp đồng hồ: 10.

4. CÁC LỆNH TRUYỀN ĐIỀU KHIỂN

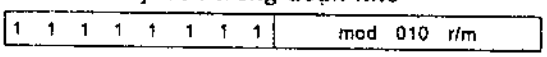
- **CALL** (gọi chương trình con)

Trực tiếp bên trong đoạn nhớ



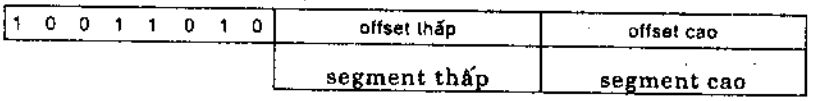
Số nhịp đồng hồ: 11.

Gián tiếp bên trong đoạn nhớ



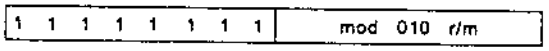
Số nhịp đồng hồ: 13 + nhịp EA.

Trực tiếp giữa các đoạn nhớ



Số nhịp đồng hồ: 20.

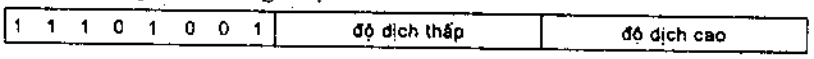
Gián tiếp giữa các đoạn nhớ



Số nhịp đồng hồ: 29 + nhịp EA.

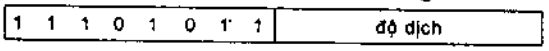
- **JMP** (nhảy vô điều kiện đến một đích nào đó)

Trực tiếp bên trong đoạn nhớ



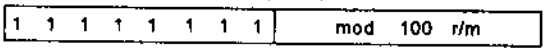
Số nhịp đồng hồ: 7.

Trực tiếp bên trong đoạn nhớ ngắn



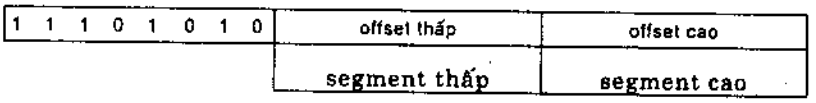
Số nhịp đồng hồ: 7.

Gián tiếp bên trong đoạn nhớ



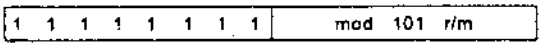
Số nhịp đồng hồ: 7 + nhịp EA.

Trực tiếp giữa các đoạn nhớ



Số nhịp đồng hồ: 7.

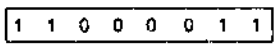
Gián tiếp giữa các đoạn nhớ



Số nhịp đồng hồ: 16 + nhịp EA.

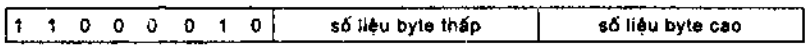
- **RET** (trở về chương trình chính từ chương trình con)

Bên trong đoạn nhớ



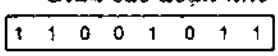
Số nhịp đồng hồ: 8.

Bên trong đoạn nhớ tức thời tới con trỏ ngăn xếp



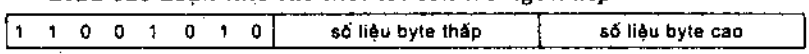
Số nhịp đồng hồ: 12.

Giữa các đoạn nhớ



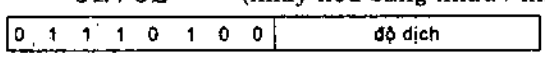
Số nhịp đồng hồ: 18.

Giữa các đoạn nhớ tức thời tới con trỏ ngăn xếp



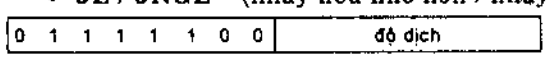
Số nhịp đồng hồ: 17.

- **JE / JZ** (nhảy nếu bằng nhau / nhảy nếu kết quả bằng không)



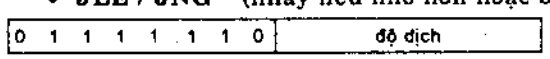
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JL / JNGE** (nhảy nếu nhỏ hơn / nhảy nếu không lớn hơn hoặc bằng)



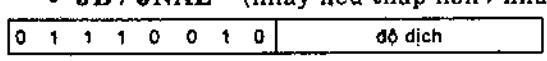
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JLE / JNG** (nhảy nếu nhỏ hơn hoặc bằng / nhảy nếu không lớn hơn)



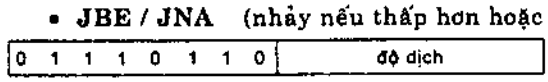
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JB / JNAE** (nhảy nếu thấp hơn / nhảy nếu không cao hơn hoặc bằng)



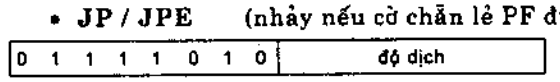
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JBE / JNA** (nhảy nếu thấp hơn hoặc bằng / nhảy nếu không cao hơn)



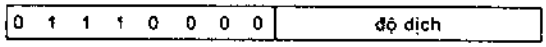
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JP / JPE** (nhảy nếu cờ chẵn lẻ PF được đặt)



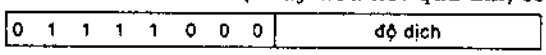
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JO** (nhảy nếu cờ tràn OF được đặt)



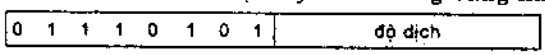
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JS** (nhảy nếu kết quả âm, cờ dấu SF = 1)



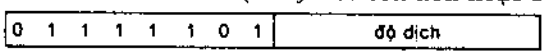
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNE / JNZ** (nhảy nếu không bằng nhau / nhảy nếu kết quả không rỗng)



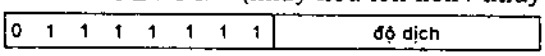
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNL / JGE** (nhảy nếu lớn hơn hoặc bằng / nhảy nếu không nhỏ hơn)



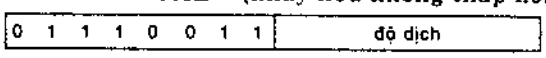
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNLE / JG** (nhảy nếu lớn hơn / nhảy nếu không nhỏ hơn hoặc bằng)



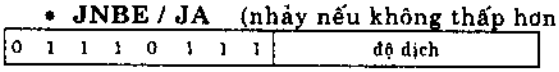
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNB / JAE** (nhảy nếu không thấp hơn / nhảy nếu cao hơn hoặc bằng)



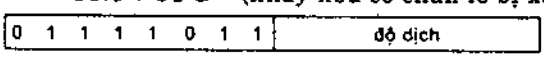
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNBE / JA** (nhảy nếu không thấp hơn hoặc bằng / nhảy nếu cao hơn)



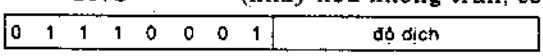
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNP / JPO** (nhảy nếu cờ chẵn lẻ bị xoá PF=0)



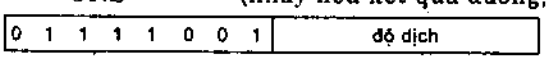
Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

- **JNO** (nhảy nếu không tràn, cờ OF=0)



Số nhịp đồng hồ: xảy ra nhảy 8
không xảy ra nhảy 4.

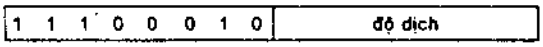
- **JNS** (nhảy nếu kết quả dương, cờ SF=0)



Số nhịp đồng hồ: xảy ra nhảy 8

không xảy ra nhảy 4.

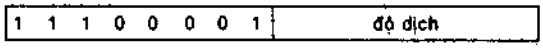
- **LOOP <nhãn>** (lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0)



Số nhịp đồng hồ: xảy ra nhảy 9
không xảy ra nhảy 5.

- **LOOPZ <nhãn> / LOOPE <nhãn>**

(lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0 hoặc ZF=0)



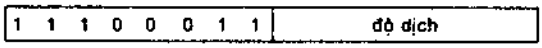
Số nhịp đồng hồ: xảy ra nhảy 11
không xảy ra nhảy 5.

- **LOOPNZ <nhãn>/ LOOPNE <nhãn>**

(lặp lại đoạn chương trình do nhãn chỉ ra cho đến khi CX=0 hoặc ZF=1)

Số nhịp đồng hồ: xảy ra nhảy 11
không xảy ra nhảy 5.

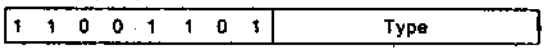
- **JCXZ <nhãn>** (nhảy nếu nội dung thanh ghi đếm rỗng)



Số nhịp đồng hồ: xảy ra nhảy 9
không xảy ra nhảy 5.

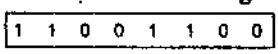
- **INT <số ngắt>** (gọi ngắt với <số ngắt> trong khoảng từ 00h đến FFh)

Ngoại lệ



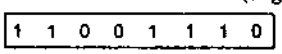
Số nhịp đồng hồ: 50.

Loại bình thường



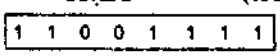
Số nhịp đồng hồ: 51.

- **INTO** (Ngắt nếu có tràn, cờ OF=1)



Số nhịp đồng hồ: 52 nếu được 4 nếu không.

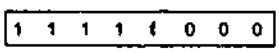
- **IRET** (trở về chương trình chính từ chương trình con phục vụ ngắt)



Số nhịp đồng hồ: 24.

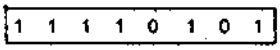
5. Các lệnh trong nội bộ vi xử lí

- **CLC** (xoá cờ nhớ)



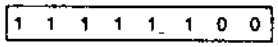
Số nhịp đồng hồ: 2

- **CMC** (đảo cờ nhớ)



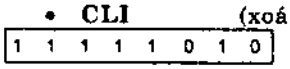
Số nhịp đồng hồ: 2.

- **CLD** (xoá cờ hướng)



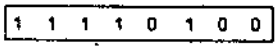
Số nhịp đồng hồ: 2.

- **CLI** (xoá cờ ngắt)



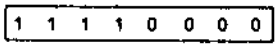
Số nhịp đồng hồ: 2.

- **HLT** (dừng hoạt động của vi xử lí)



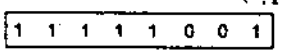
Số nhịp đồng hồ: 2.

- **LOCK** (đưa ra tín hiệu khoá bus)



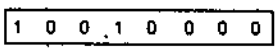
Số nhịp đồng hồ: 2.

- **STC** (lập cờ nhớ)



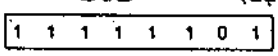
Số nhịp đồng hồ: 2.

- **NOP** (CPU không làm gì cả)



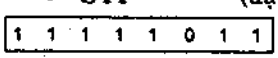
Số nhịp đồng hồ: 3.

- **STD** (đặt cờ hướng)



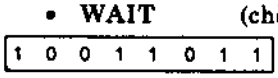
Số nhịp đồng hồ: 2.

- **STI** (đặt cờ ngắt)



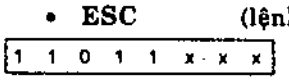
Số nhịp đồng hồ: 2.

- **WAIT** (chờ tín hiệu từ chân TEST hoặc INTR)



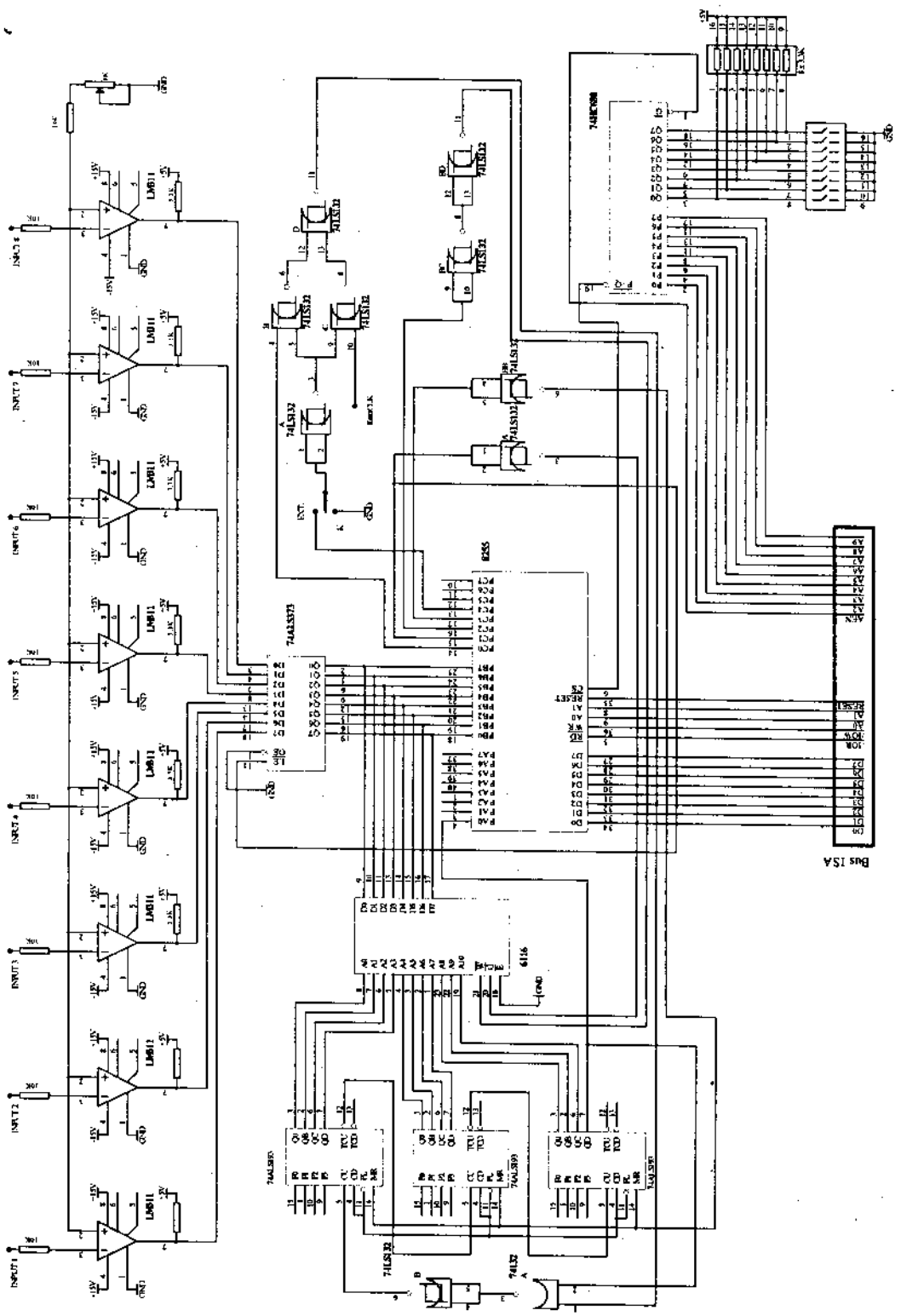
Số nhịp đồng hồ: 3.

- **ESC** (lệnh đồng xử lí toán)



Số nhịp đồng hồ: 7 + nhịp EA.

PHỤ LỤC 2 Sơ đồ bản mạch phân tích lôgic 8 kênh đơn giản



PHỤ LỤC 3. MÁY VI TÍNH DÙNG CHO ĐÀO TẠO INTEL SDK-86

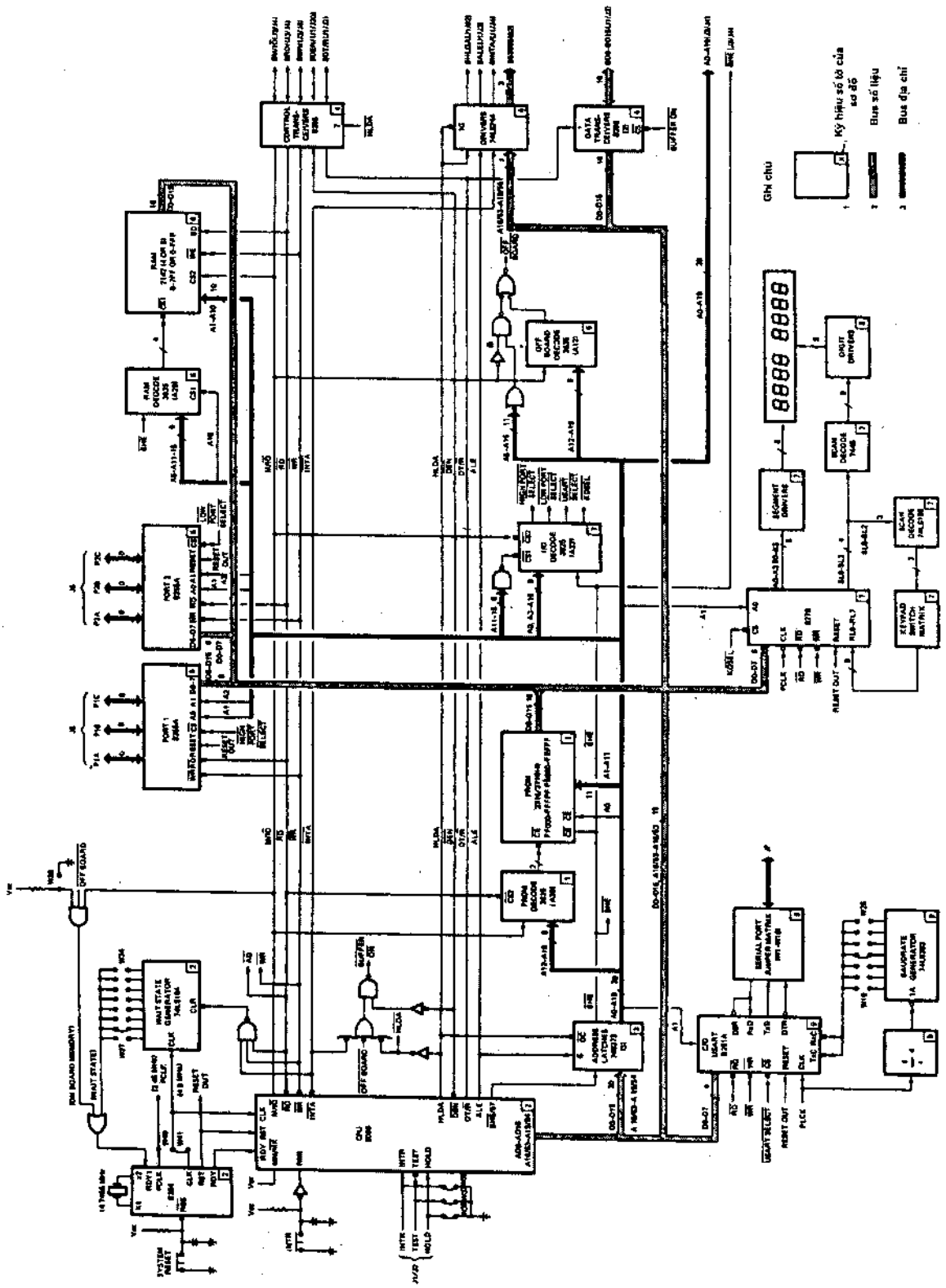
Hãng Intel đã sản xuất ra một bộ máy vi tính nhỏ với vi xử lí 8086 hoạt động trong chế độ MIN dùng cho giảng dạy, đào tạo về vi tính. Đó là kit SDK-86 được thiết kế lắp ráp trên các vi mạch rời có độ tích hợp không quá lớn. Các máy vi tính hiện nay được lắp ráp trên các chip có độ tích hợp rất lớn và được thiết kế đặc biệt theo công nghệ ASIC nhưng về nguyên tắc vẫn như vậy. Do đó việc khảo sát sơ đồ nguyên lí của kit Intel SDK-86 cũng vẫn hữu ích khi học viên muốn hiểu rõ về việc thiết kế phần cứng trong một hệ thống máy vi tính.

Sơ đồ khối chung của kit như hình P3.1 chỉ ra. Bộ phát xung đồng hồ, bộ chốt địa chỉ và bộ đệm bus dữ liệu được nối với 8086 để tạo thành một nhóm. Nhóm này phát ra và thu nhận các tín hiệu trên bus dữ liệu, bus địa chỉ, bus điều khiển. Nếu nối nhóm đó với các bộ nhớ ROM, RAM, các cổng và các thiết bị khác ta sẽ có một hệ thống hoàn chỉnh. Từ sơ đồ ta thấy là ngoài các IC thì trên bản mạch còn có bàn phím hệ 16, thiết bị hiển thị 7 đoạn và một vùng mở rộng để thêm ROM, RAM, các cổng hay các mạch ghép nối. Một chương trình kiểm tra trong ROM trên bản mạch cho phép nhập vào, thực hiện và gỡ rối các chương trình mã máy sử dụng bàn phím onboard hay CRT bên ngoài nối với cổng nối tiếp trên board. Trên bản mạch có 2 Kbytes RAM và vài đế cắm để người dùng có thể thêm vào 2 kbyte nữa. Hệ thống cũng có 6 cổng vào/ra song song 8-bit có thể lập trình được.

Phần đầu tiên trong hình là CPU-8086 và bộ phát xung đồng hồ 8284. Chú ý rằng 8284 có tinh thể dao động 14,7456 MHz nối với nó. Tần số của tinh thể nối với 8284 sẽ chia ba để tạo ra tín hiệu đồng hồ gửi tới 8086. Vì vậy tần số đồng hồ thực sự của 8086 trên bản mạch này là 4.915 MHz. Một tín hiệu đồng hồ khác tên là PCLK cũng được tạo ra bởi 8284 có tần số bằng một nửa tần số tín hiệu đồng hồ (bằng 2,45 MHz). Tín hiệu này được dùng như một tín hiệu đồng hồ đa năng trong toàn hệ thống. Tín hiệu RST và RDY cũng nối qua 8284 để đồng bộ với tín hiệu đồng hồ trước khi chúng được gửi đến 8086. Như nhìn thấy trên hình, một mạch được nối với đầu vào RDY1 để trong vài điều kiện có thể tạo ra trạng thái đợi xen vào một chu kì máy. Khối có tên W27 tới W34 phía trên bộ phát trạng thái đợi trong sơ đồ khối là các chân cắm mà có thể cắm các cầu nối (jumper) lên đó để xác định số trạng thái đợi mong muốn xen vào trong một chu kì máy. Ta sẽ thảo luận chi tiết trong phần sau. Trong các biểu tượng cho khối 8086, 8284, bộ phát trạng thái đợi đều có chứa số 2 trong góc phải bên dưới. Số này chỉ ra số sơ đồ chi tiết cho những phần này sẽ trình bày trên các hình vẽ sau. Hình P3.2a và P3.2b chỉ ra sơ đồ chi tiết của các khối này.

Phần tiếp theo trong sơ đồ khối của SDK-86 là các chốt địa chỉ mà như ta đã biết nó sẽ giữ thông tin địa chỉ trong chu kì T1 của chu kì máy. Hộp ngay bên dưới 8086 trong hình chỉ ra là có ba vi mạch 74S373 được dùng trong bộ chốt địa chỉ. AD0-AD15, A16 - A19 và $\overline{BHE}$ được nối với đầu vào của bộ chốt. Tín hiệu ALE được dùng để kích hoạt bộ chốt. Thông tin được giữ trên đầu ra bộ chốt là A0 - A19 và $\overline{BHE}$ sau khi ALE xuống mức thấp. Kí hiệu /20 sau A0 - A19 ở đầu ra các bộ chốt chỉ ra rằng có 20 đường dây trong nhóm này. Đường dây được vẽ to đậm được dùng để phân biệt bus địa chỉ và bus dữ liệu.

Tiếp theo đường địa chỉ phía bên phải là ROM của hệ thống. Hộp có tên PROM chỉ ra là gồm chip nhớ 2316 hay 2716 được dùng trong ROM. Mỗi chip chứa 2 kbyte bộ nhớ. Hộp PROM cũng chỉ ra địa chỉ tuyệt đối của các chip 2316 hay 2716. Hai EPROM chiếm không gian địa chỉ



Hình P3.1 Sơ đồ khối của kit SDK - 86 .

từ FE000h tới FEFFFh, hai PROM khác chiếm địa chỉ từ FF000h tới FFFFFh. Bộ giải mã PROM 3625 được nối với các EPROM này có hai mục đích. Mục đích thứ nhất là tạo ra tín hiệu để bật ROM mong muốn khi gửi địa chỉ của ROM đó đến. Mục đích thứ hai là đảm bảo chỉ một thiết bị đang gửi ra tín hiệu trên bus dữ liệu tại một thời điểm. Ta sẽ thảo luận chi tiết sau cách thức mà bộ giải mã địa chỉ tạo ra địa chỉ mong muốn của một thiết bị cụ thể trong hệ thống. Chú ý chân lối vào $\overline{CS2}$ của PROM được nối với tín hiệu $\overline{RD}$ của 8086. Việc nối này để bộ giải mã PROM chỉ hoạt động khi 8086 đang thực hiện quá trình đọc. Ta không muốn ROM sẽ bật lên khi khi tình cờ gửi đến ROM một địa chỉ trong một quá trình viết, bởi vì nếu cố gắng để viết vào đầu ra của ROM thì có thể làm chập mạch điện cả ROM và đầu ra bộ đệm. Kí hiệu A26 trong PROM là thể hiện IC 3625 sẽ có số A26 trong sơ đồ chi tiết.

Theo bus địa chỉ tới góc phải trên của sơ đồ khối hình P3.1 là RAM. Bản mạch nhớ này là 2 kbyte RAM tĩnh được lắp trên 4 chip nhớ 2142 nhưng có thêm các đế cắm cho 4 chip 2142 nữa. Bốn chip 2142 ban đầu này chiếm không gian địa chỉ là từ 00000h đến 007FFh. Nếu có thêm bốn chip 2142 nữa thì không gian địa chỉ là từ 00800h đến 00FFFh. Một PROM 3625 khác được dùng ở đây như là bộ giải mã RAM. Giống bộ giải mã PROM thì nhiệm vụ của thiết bị này là cho phép bộ nhớ tương ứng với địa chỉ gửi đi trên bus địa chỉ và đảm bảo rằng chỉ duy nhất một thiết bị gửi dữ liệu ra tại một thời điểm. 8086 có thể đọc và viết một byte hay một từ vì vậy đường dữ liệu 16 dây được nối với khối RAM.

Ta xem xét đến cổng hệ thống. Hai cổng 8255A trên đầu trang giấy là các cổng song song có thể lập trình được. Hai cổng 8255A có thể được sử dụng độc lập để nhập hay xuất các byte song song. Hay hai cổng cũng có thể sử dụng cùng nhau để nhập hay xuất các từ (từ dữ liệu, từ điều khiển). Với quá trình nhập xuất byte thì chỉ một trong hai cổng được chọn bằng cách xác lập đầu vào $\overline{CS}$ của cổng xuống mức thấp. Với quá trình nhập xuất từ thì cả hai cổng 8255A đều được chọn bằng cách đặt cả hai đầu vào $\overline{CS}$ xuống mức thấp. Byte cao của của từ sẽ được gửi đến *PORT 1*. Byte thấp của từ được gửi đến *PORT 2*. Để rõ hơn, nếu byte cao của từ đến cổng *P1A* thì byte thấp đến cổng *P2A*. Trong phần sau ta trình bày cách định địa chỉ cho các cổng này.

Phần lớn hệ thống cần một cổng nối tiếp để chúng có thể liên lạc với các thiết bị đầu cuối có yêu cầu dữ liệu phát và thu dạng nối tiếp. Góc trái bên dưới hình P3.1, SDK-86 có sử dụng chip USART-8251A là cổng nối tiếp (*Universal Synchronous/ Asynchronuos Receiver Transmitter).* Chip này có hai thanh ghi dịch thuận nghịch. Một thanh ghi dịch nhận byte dạng song song từ bus dữ liệu và dịch byte đó ra đầu ra TxD có dạng nối tiếp. Thanh ghi còn lại sẽ dịch dữ liệu dạng nối tiếp từ đầu vào RxD và biến đổi nó thành dạng byte song song mà 8086 có thể đọc được từ bus dữ liệu. 8251A có 8 lối vào dữ liệu vì vậy dữ liệu chỉ có thể đọc và viết từng byte từ 8251A tại mỗi thời điểm. Chỉ có 8 bit thấp của bus dữ liệu được nối với 8251A. Mỗi thanh ghi dịch trong 8251A cần một tín hiệu đồng hồ có tần số bằng 16 lần hay 64 lần tốc độ mà ta muốn dịch bit dữ liệu ra hay vào. Lối vào clock cho việc truyền dữ liệu của thanh ghi dịch là TxC còn lối vào clock cho việc nhận dữ liệu của thanh ghi dịch là RxC. Các tín hiệu clock này có quan hệ mật thiết với nhau vì ta thường muốn tốc độ gửi và nhận dữ liệu là bằng nhau. Tín hiệu clock cho các đầu vào này được tạo ra bằng cách chia tín hiệu PCLK 2,45 MHz của bộ phát 8284. Các chân W19-W25 cho phép chọn tần số lối vào TxC và RxC mong muốn từ một loạt các bộ chia tần trong bộ phát tốc độ baud là tốc độ mà bit dữ liệu được dịch vào hay ra của một thiết bị nối tiếp. Với chip 8251A thu phát thông tin với mã nhị phân thì tốc độ này chính bằng số bit được dịch trong 1 giây bps (bit per second). Thí dụ nếu thời gian để dịch 1 bit

là 416μs thì tốc độ baud là 2400 baud. Tốc độ baud thông dụng cho truyền dữ liệu nối tiếp là 300, 600, 1200, 2400, 9600 và 19200. Thiết bị vào/ra cuối cùng mà ta đề cập ở đây là 8279 nằm ở phần giữa dưới cùng của sơ đồ. 8279 là một chip vào/ra chuyên dụng có hai chức năng chính. Chức năng đầu tiên là để quét bàn phím mã Hex, dò tìm phím nào được nhấn, chống nhiễu rung cơ khí (debounce) từ phím vừa ấn, lưu giữ mã cho phím vừa ấn vào trong RAM nơi mà 8086 có thể được đọc. Chức năng chính thứ hai là làm tươi tám thanh LED 7 đoạn hiển thị. Mã bảy đoạn cho từng digit sẽ được gửi tới RAM chứa trong 8279. 8279 tự động gửi mã của một digit này và bật sáng digit đó. Sau vài ms thì 8279 gửi tiếp mã 7 đoạn của digit tiếp theo và bật sáng digit đó. Quá trình này tiếp tục cho đến khi tất cả các digit được sáng lần lượt. Đó là một phần trong quá trình giải mã động cho bộ hiển thị LED 7 đoạn. Điểm chính bây giờ cần biết là chip 8279 sẽ quét bàn phím và làm tươi hiển thị để mà ta không phải làm các quá trình này như một phần trong chương trình.

Bây giờ ta đã có cái nhìn tổng quan về các cổng trong hệ thống này, tiếp theo ta sẽ xem xét bộ giải mã chọn cổng được chỉ định. Đó là PROM 3625 ở chính giữa của sơ đồ. Sau đây ta trình bày cách thức các thiết bị này tạo ra tín hiệu chọn cổng từ địa chỉ cổng mà 8086 gửi đến.

Phần cuối cùng trong sơ đồ khối của SDK-86 là bộ đệm bên tay phải. Mục đích của bộ đệm là hỗ trợ cho bus dữ liệu và bus điều khiển để điều khiển ROM, RAM và các cổng thêm vào phần mở rộng trên bản mạch. Chú ý rằng các đường địa chỉ đã được đệm bởi bộ chốt địa chỉ 74LS373. Đến đây ta đã có cái nhìn tổng quan về SDK-86, bước tiếp theo là xem xét từng sơ đồ thực của các mạch. Nếu ta không có kit SDK-86 thì qua các sơ đồ thiết kế này ta vẫn có thể học được rất nhiều về chức năng của hệ thống 8086.

Do các phần sơ đồ được in tách rời nhau nên một vài biểu tượng chung trong các sơ đồ hệ thống vi xử lí này cần được giải thích để có thể nhanh chóng hiểu được chúng. Các con số ở phần trên cùng và dưới cùng của sơ đồ và các chữ cái dọc theo cạnh của từng tờ là toạ độ vùng. Các toạ độ này để xác định vị trí các phần hay các đầu nối trên sơ đồ với nhau, điều này hoàn toàn giống như dùng toạ độ trên bản đồ để giúp tìm vị trí một nơi nào đó nhanh chóng hơn. Thí dụ trên tờ 1 các đường có tên từ A1 - A7 ở góc trái trên, bên cạnh các đường này ta thấy có kí hiệu 3ZC2. Kí hiệu này chỉ ra rằng các đường địa chỉ này đến từ vùng C2 trên tờ 3. Để biết thực sự các đường này nối với đâu thì trước tiên phải tìm trên tờ 3 sau đó đi dọc theo mũi tên có tên C cho đến khi gặp cột có tên 2. Vùng này đủ nhỏ để ta có thể tìm thấy các đường địa chỉ này từ đâu đến. Toạ độ vùng cạnh các đường trên tờ 3 chỉ tờ và vùng khác mà các đường này đi tới. Để thực hành hãy thử tìm vài đường khác mà đi đến và đi ra từ tờ này.

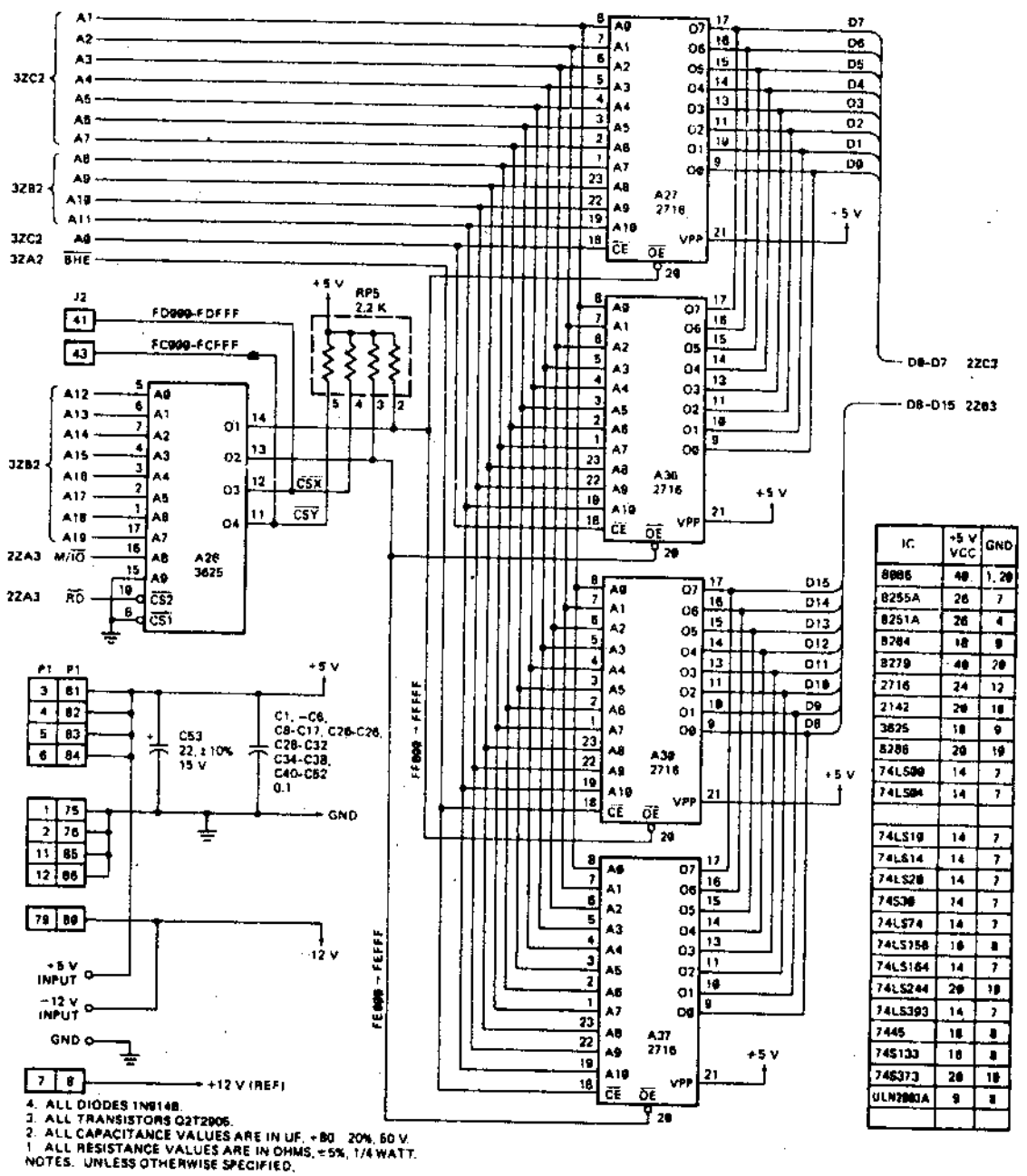
Điểm tiếp theo, trên sơ đồ còn có kí hiệu số trên các IC. Ví dụ như thêm vào bên cạnh số 2716 mỗi IC còn có con số dạng A36. Kí hiệu số này giúp xác định vị trí của IC trên bản mạch in. Các kí hiệu số này được in trên bản mạch gần với IC tương ứng. Thường thường các kí hiệu số của IC được viết tuần tự bắt đầu từ góc trái trên bên cạnh linh kiện. Có rất nhiều 2716 trên bản mạch nhưng chỉ có một cái có tên là A36.

Một loại linh kiện khác thêm vào trên bản mạch là các "gói điện trở" (resister pack) như là tại toạ độ C5 của tờ 1. Như nhìn thấy trên sơ đồ, "gói điện trở" này chứa 4 điện trở 2,2 kΩ. "Gói điện trở" thường rất nhỏ, gắn dạng chữ nhật dọc hay toàn bộ điện trở được đóng gói như một IC nhỏ. Ưu điểm của chúng là chiếm ít không gian bản mạch in và dễ dàng để lắp vào mạch hơn là từng điện trở rời rạc.

Vài biểu tượng khác trong giản đồ là các cấu trúc có tên là J2 hay P1, ví dụ như trong toạ độ C7 và B7 của tờ 1. Các biểu tượng này dùng để chỉ các đầu nối. Con số bên trong hộp chữ

nhật chỉ số chân trên đầu nối có tín hiệu đi đến. Chữ P viết tắt của chữ Plug. Một đầu nối được gọi là Plug (phích cắm) nếu nó cắm vào một thiết bị khác nữa. Trong SDK-86 thì đầu nối tên P1 ở bên rìa của bản mạch in. Chữ J bên cạnh đầu nối viết tắt của chữ Jack (ổ cắm). Một đầu nối được gọi là Jack nếu có một thiết bị khác cắm vào nó. Trên board SDK-86 các jack từ J1 - J6 là một đầu nối 50 chân mà ta có thể cắm đầu nối dây cáp vào nó. Các jack này cho phép bus địa chỉ, bus số liệu, bus điều khiển, các cổng song song được nối với các mạch điện. Thêm một điểm nữa cần chú ý trong sơ đồ của SDK-86 là các tụ điện mắc ở lối vào nguồn ví dụ ở toạ độ B6 của tờ 1. Như trên hình vẽ thì một lượng lớn các tụ điện 0,1 μF mắc song song với tụ điện 22 μF. Phần lớn các hệ thống đều có mạch lọc giống như thế này trên đường dây nguồn. Tại sao lại mắc song song nhiều tụ điện nhỏ với một tụ điện lớn hơn nhiều lần? Bởi vì tụ điện lớn lọc đi nhiễu tần số thấp trên đường dây nguồn và tụ điện nhỏ lọc nhiễu tần số cao. Các thiết bị chuyển từ trạng thái lôgic này sang trạng thái lôgic khác sinh ra nhiễu trên đường dây nguồn. Nếu nhiễu này không bị lọc bỏ đi bởi các tụ điện thì nhiễu này có thể trở nên đủ lớn để gây rối loạn hoạt động của hệ thống.

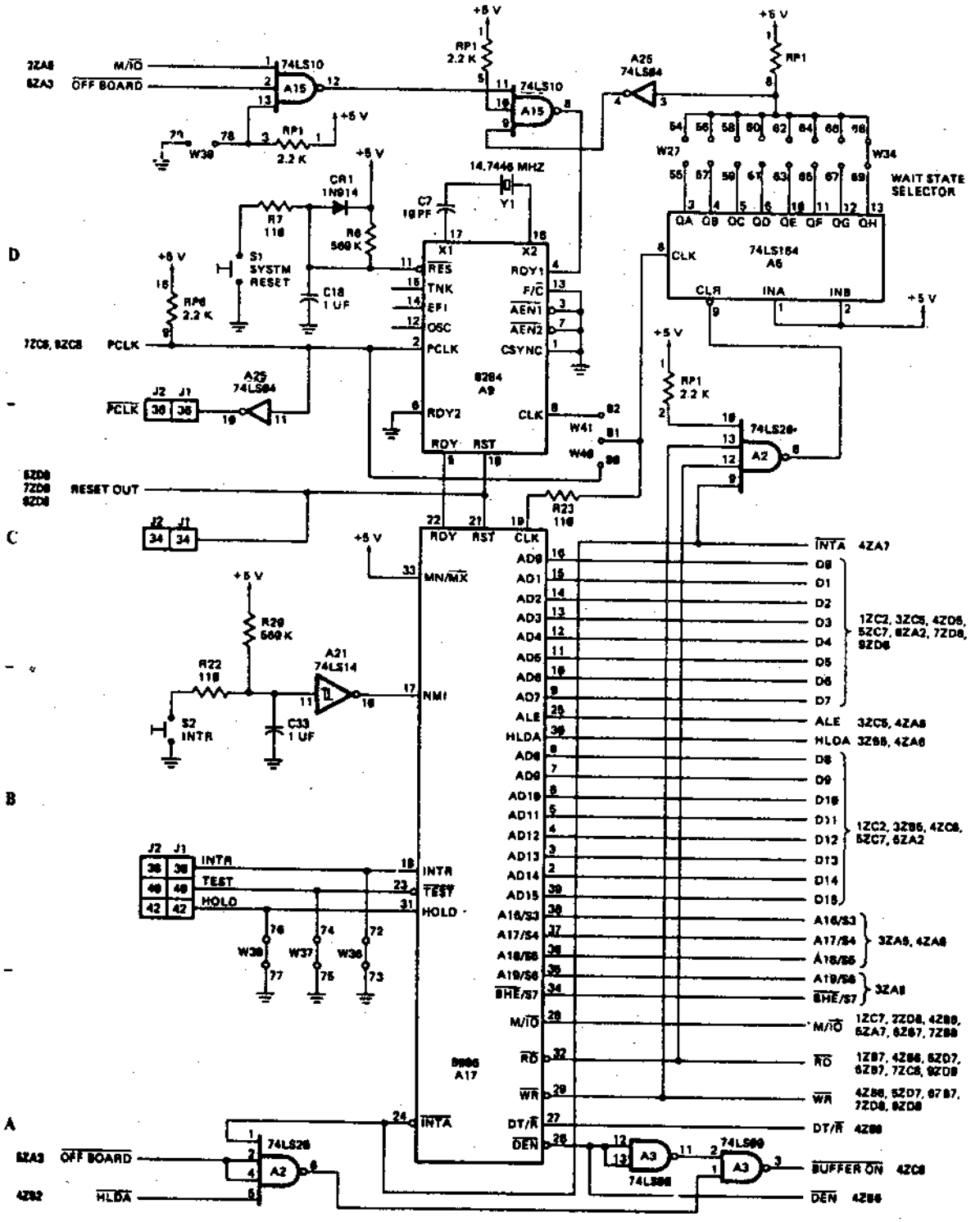
Ta sẽ xem lướt qua các sơ đồ SDK-86 để biết các phần khác nữa được đặt ở đâu và xem thông tin thêm nào có thể thu được từ các kí hiệu trên các thiết bị.



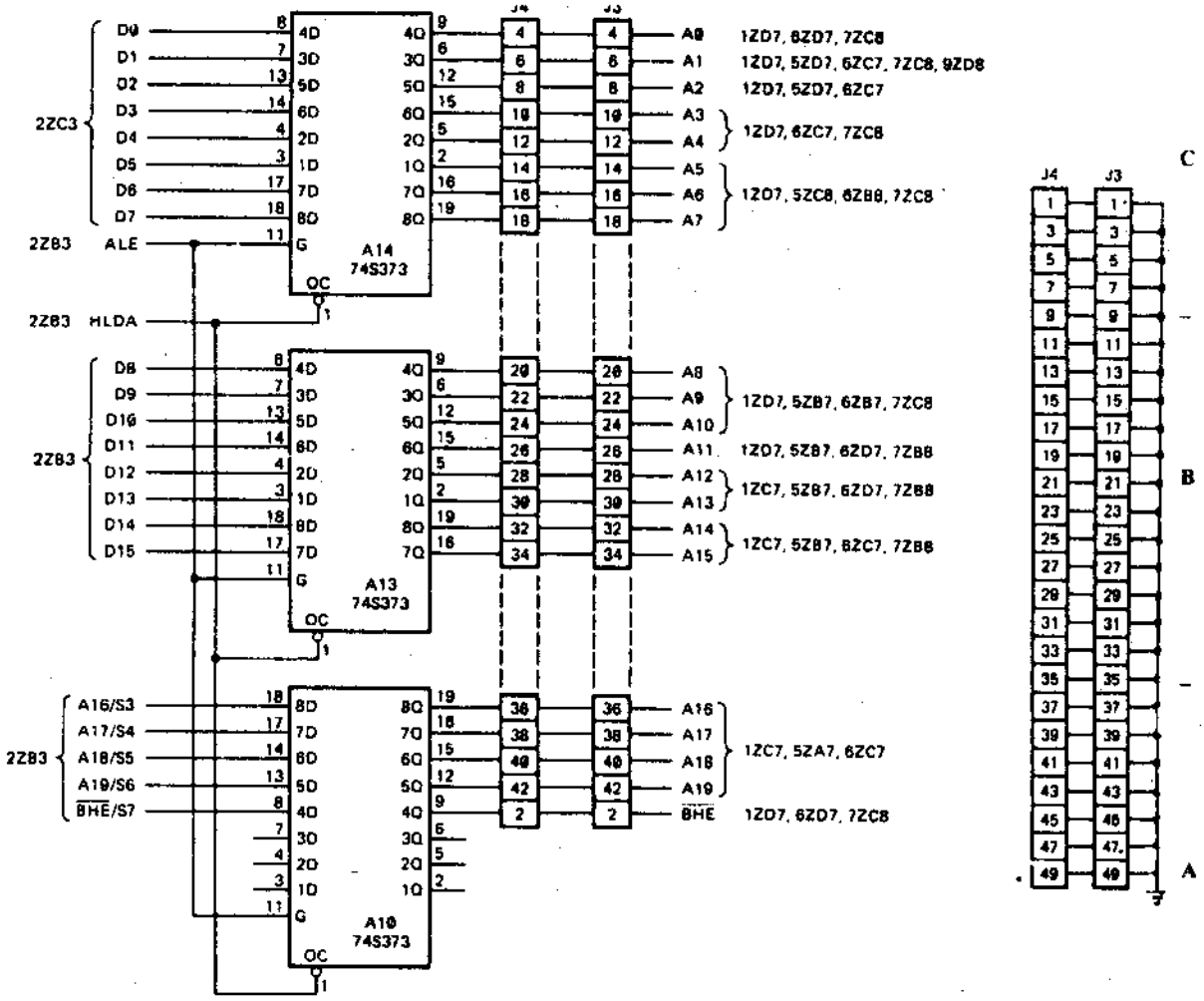
IC	+5 V VCC	GND
8086	40	1, 20
8255A	26	7
8251A	26	4
8284	18	9
8279	40	20
2716	24	12
2142	20	10
3625	18	9
8286	20	10
74LS00	14	7
74LS04	14	7
74LS10	14	7
74LS14	14	7
74LS20	14	7
74S30	14	7
74LS74	14	7
74LS158	16	8
74LS164	14	7
74LS244	20	10
74LS393	14	7
7445	16	8
74S133	16	8
74S373	20	10
ULN2803A	9	8

4. ALL DIODES 1N914B.
3. ALL TRANSISTORS Q2T2905.
2. ALL CAPACITANCE VALUES ARE IN UF, +80 20%, 50 V.
1. ALL RESISTANCE VALUES ARE IN OHMS, ±5%, 1/4 WATT.
NOTES. UNLESS OTHERWISE SPECIFIED.

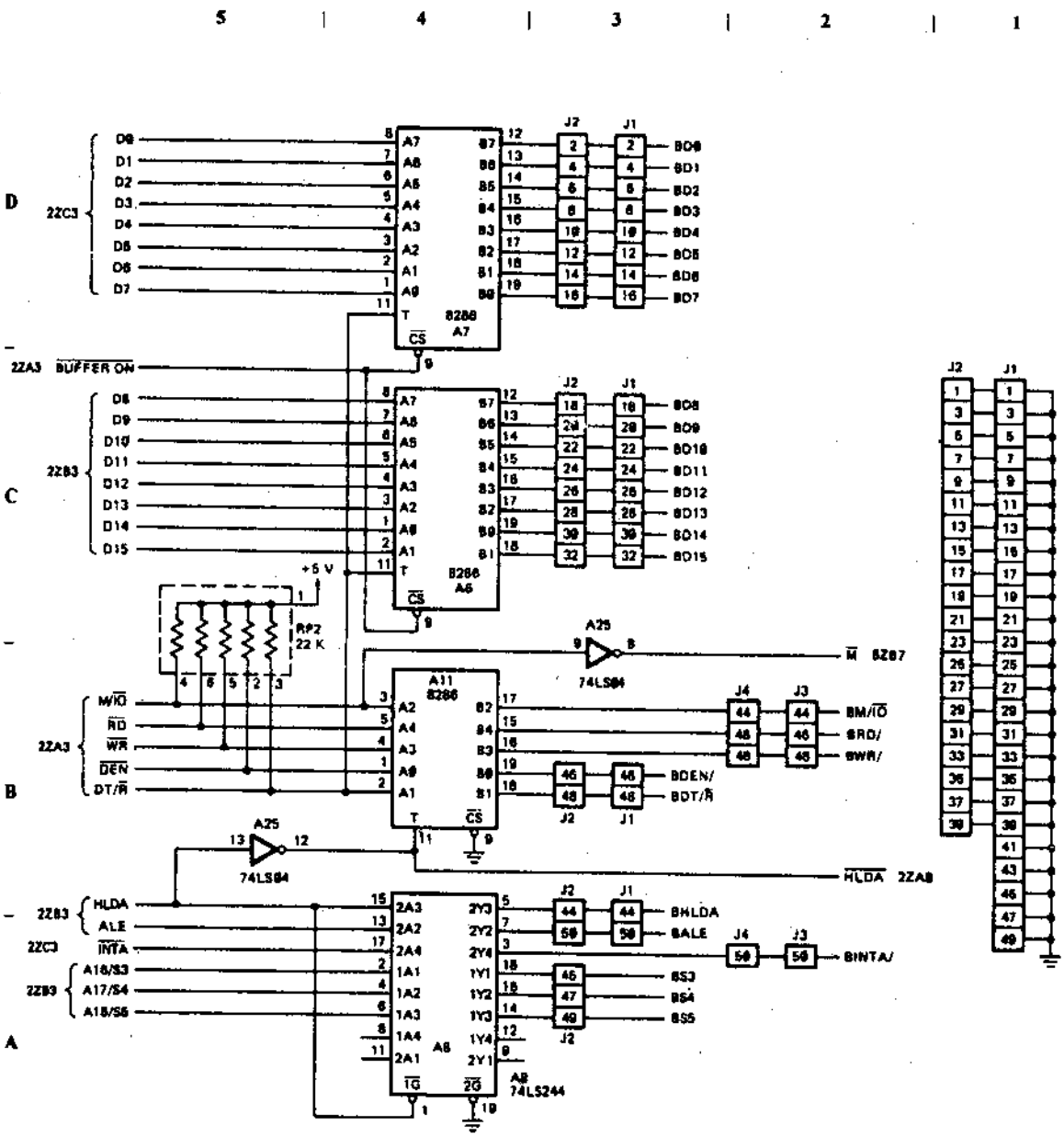
Hình P3.2a Sơ đồ chi tiết kit SKD - 86 / Tờ 1.



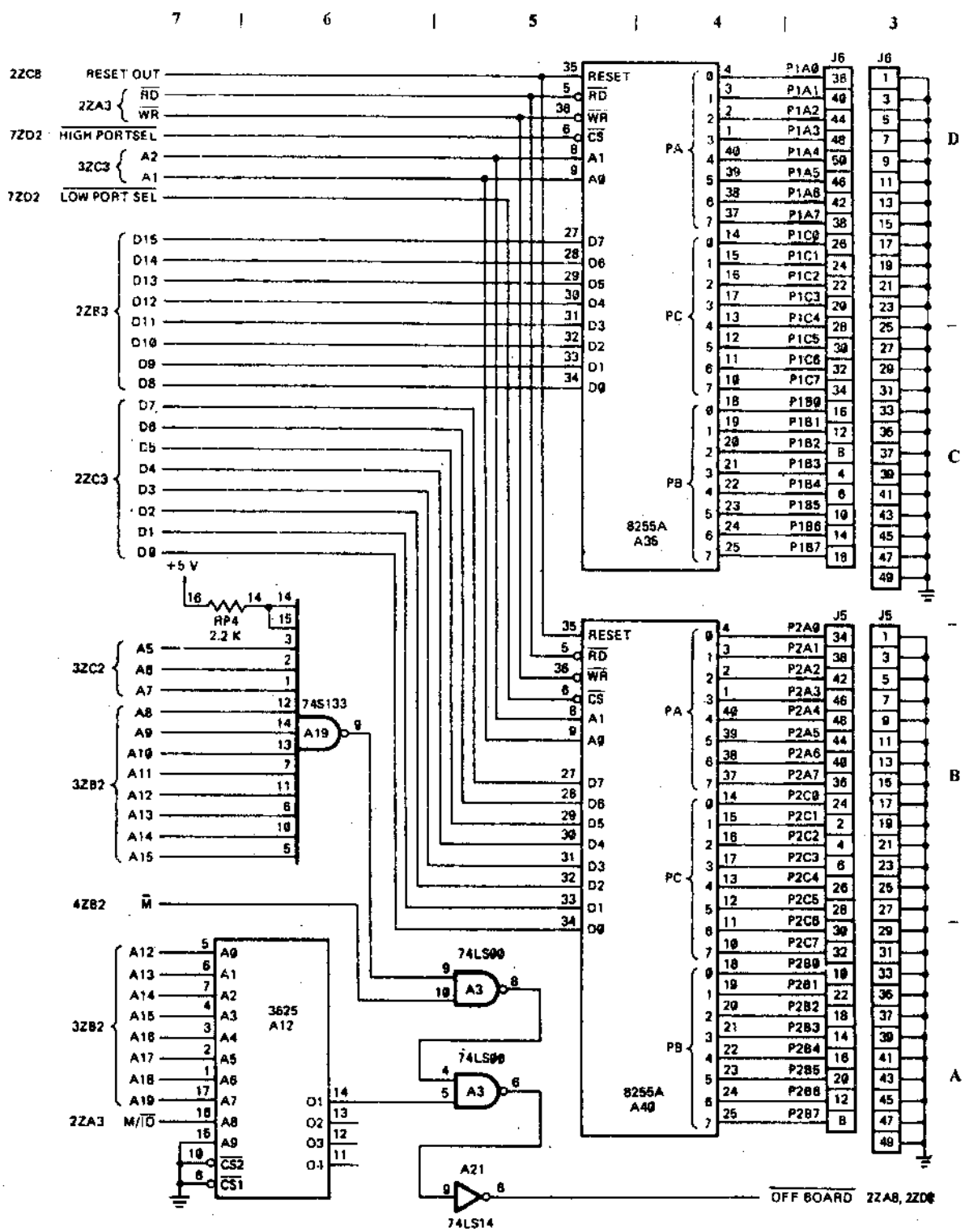
Hình P3.2b Sơ đồ SDK - 86 / Tờ 2.



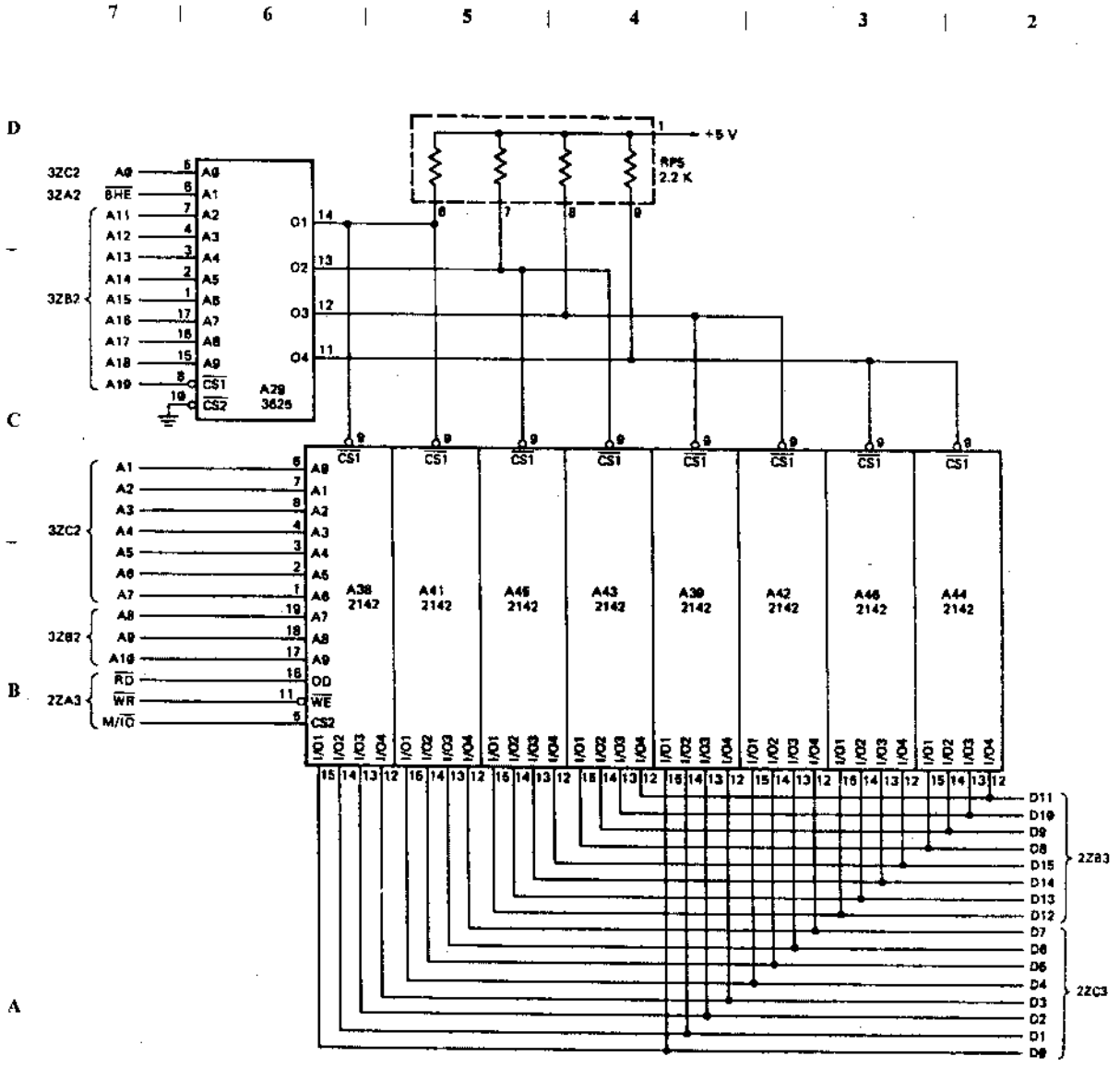
Hình P3.2c Sơ đồ SDK - 86 / Tờ 3.



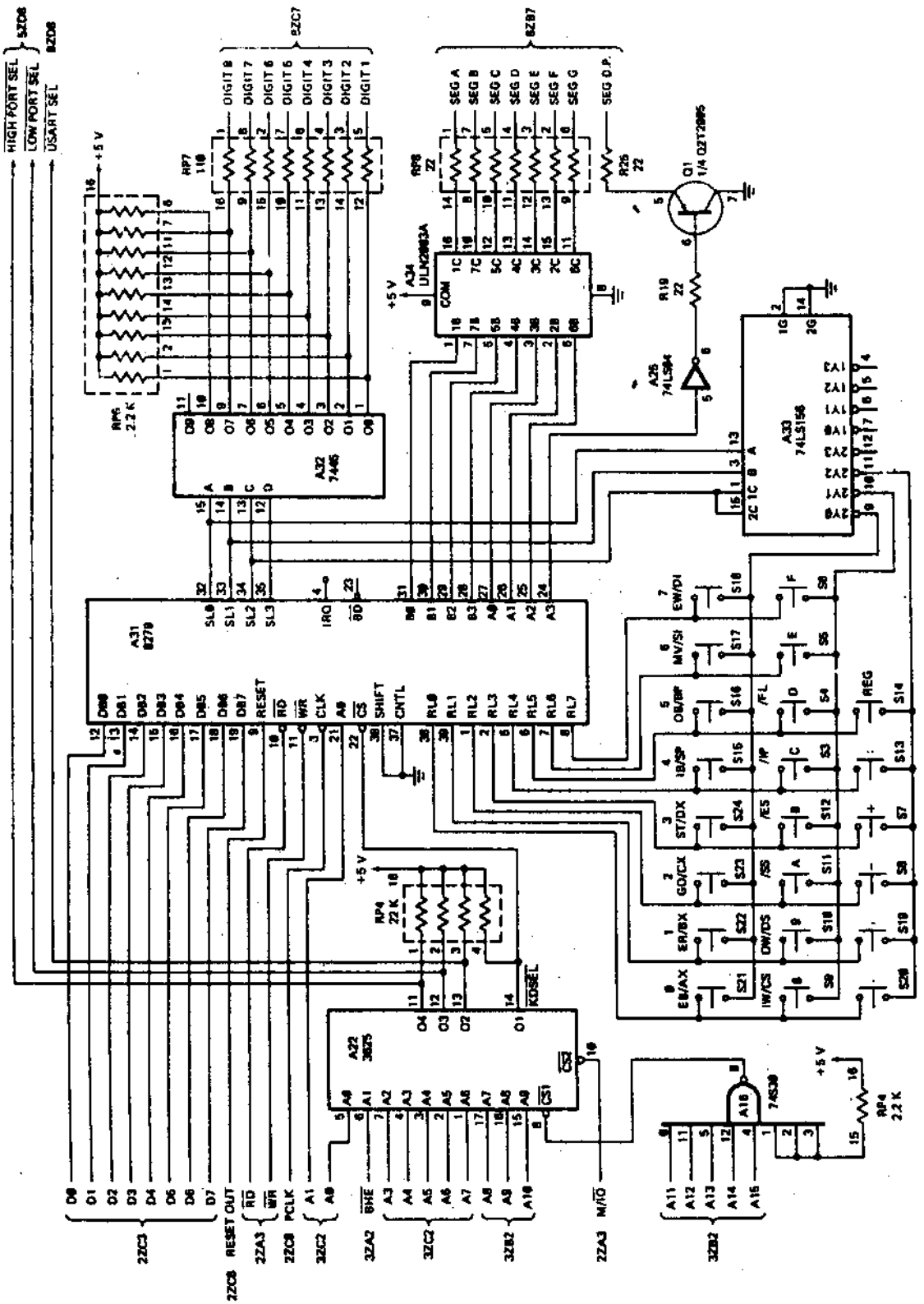
Hình P3.2d Sơ đồ SDK - 86 / Tờ 4.



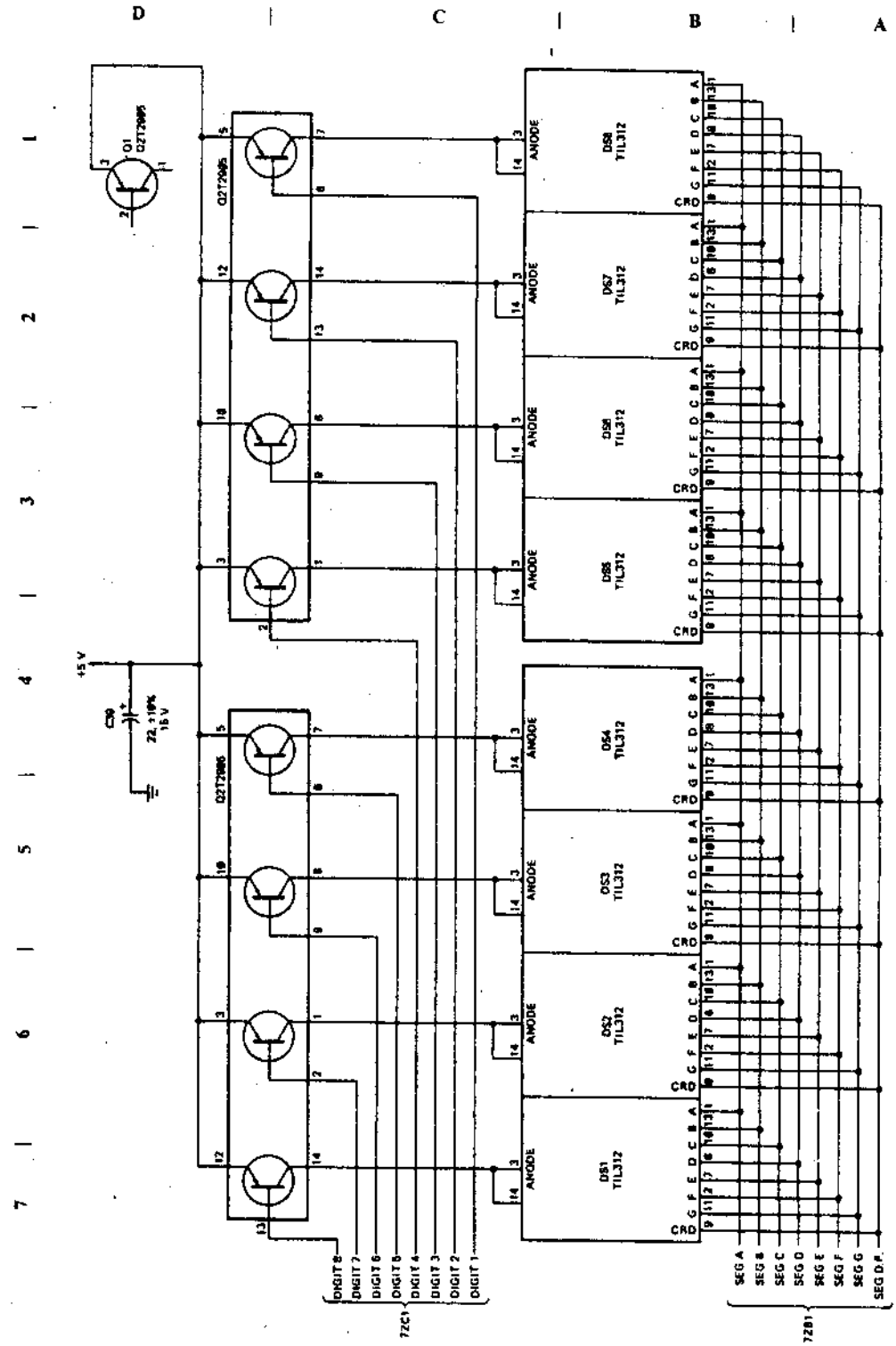
Hình P3.2e Sơ đồ SDK - 86 / Tờ 5.



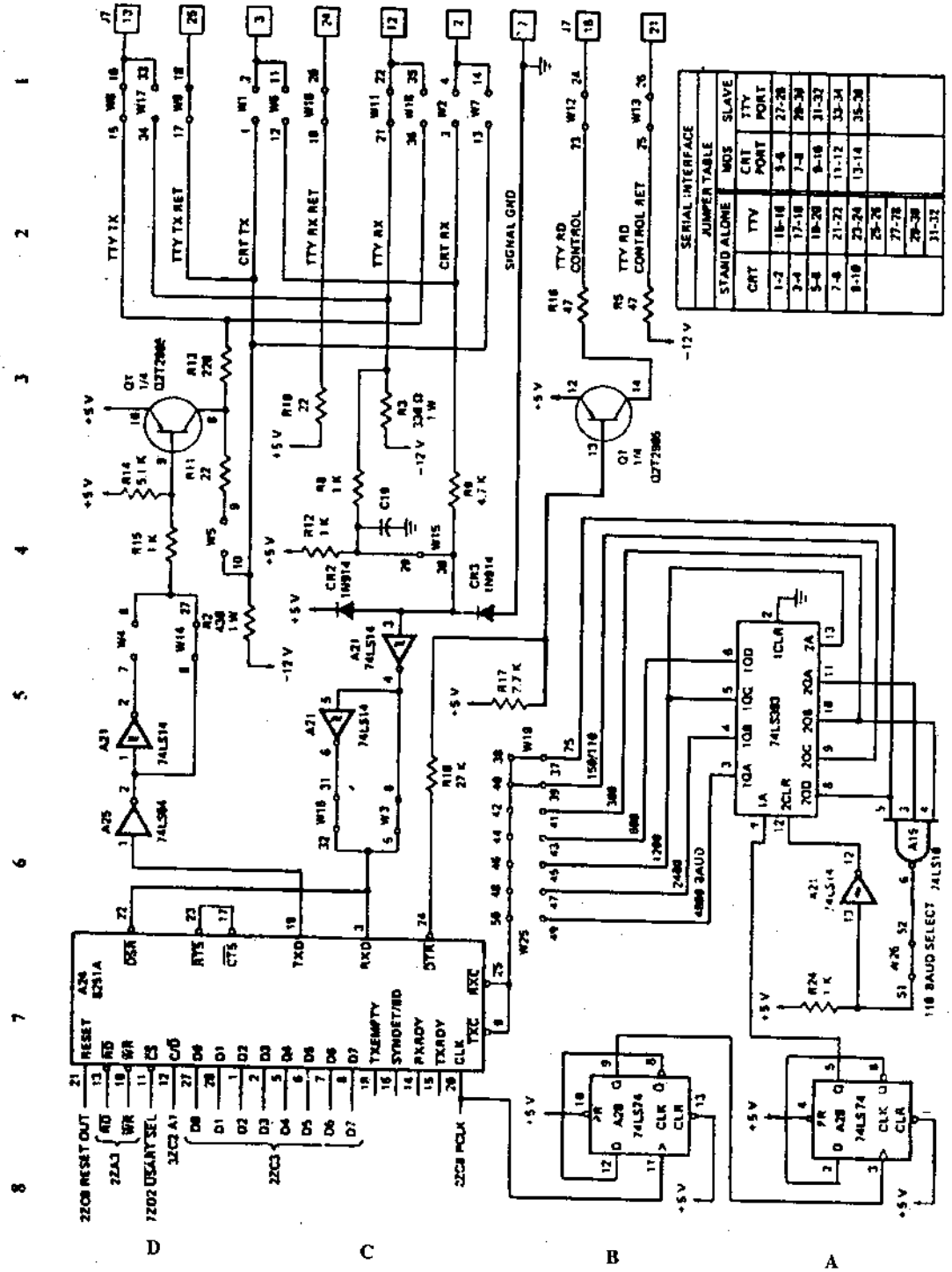
Hình P3.2f Sơ đồ SDK - 86 / Tờ 6



Hình P3.2g Sơ đồ SDK - 86 / Tờ 7.



Hình P3.2h Sơ đồ SDK - 86 / Tờ 8.



D C B A

Hình P3.2k Sơ đồ SDK - 86 / Tờ 9.

Như trên đã nói, chip 3625 trên bản mạch là bộ giải mã địa chỉ. Chức năng của bộ giải mã địa chỉ là tạo ra một tín hiệu chọn chip cho phép thiết bị ROM, RAM hay cổng định truy xuất và đảm bảo chỉ một chip đó được phép phát dữ liệu lên bus dữ liệu tại một thời điểm.

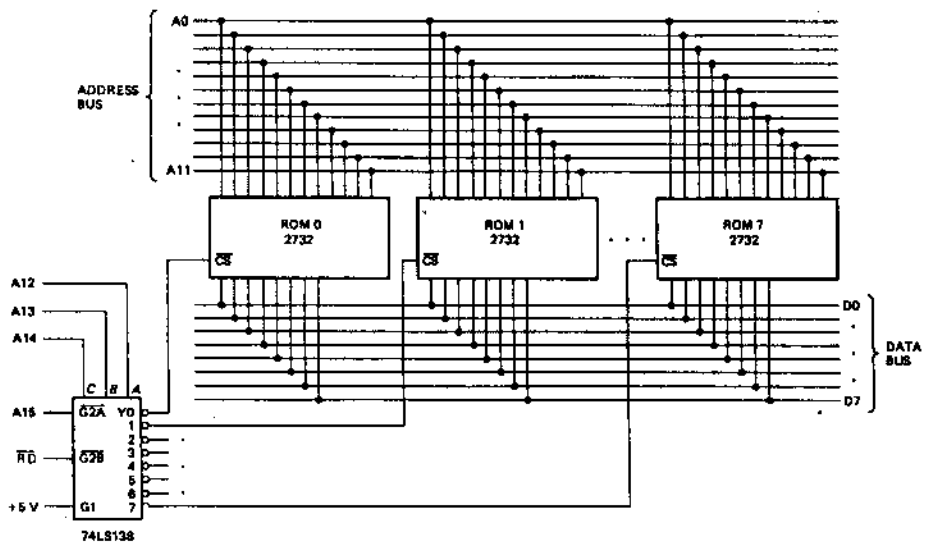
Hệ thống giải mã địa chỉ theo các cách khác nhau.

a, Tổ chức băng nhớ trong hệ thống 8086

Vì 8086 là vi xử lí có bus dữ liệu 16 bit (rộng 2 byte) nên về nguyên tắc nó truy xuất các từ (word) có địa chỉ chẵn chỉ trong 1 chu kì bus trong khi truy xuất các từ có địa chỉ lẻ phải mất 2 chu kì bus. Để đọc/viết 1 từ chỉ trong 1 chu kì máy, bộ nhớ 1MB = 1 048 576 Byte được chia thành hai băng, mỗi băng 524 288 Byte như hình 3.12 chương 3 đã chỉ ra. Các đường số liệu của băng chẵn được nối tới 8 đường thấp D7-D0 của bus dữ liệu 8086. Các đường số liệu của băng lẻ được nối tới 8 đường cao D15-D8. Đường địa chỉ A0 được dùng để cho phép (enable) băng thấp khi nó ở mức lôgic 0. Các đường địa chỉ còn lại từ A1 đến A19 được dùng để chọn địa chỉ cần truy nhập trong băng. Tín hiệu $\overline{BHE}$ được đưa ra từ 8086 cùng một thời gian với tín hiệu địa chỉ qua mạch chốt nhờ xung chốt địa chỉ ALE.

b, Giải mã ROM trong SDK - 86

Trong hình P3.3 là một thí dụ có tám EPROM được nối song song trên cùng một bus dữ liệu. Từ sơ đồ ta thấy các EPROM này xuất ra dữ liệu dạng byte bởi vì chỉ có tám đầu ra được nối với bus dữ liệu. Số các đường địa chỉ đi vào từng chip EPROM đã chỉ ra các EPROM này có thể lưu trữ được bao nhiêu byte. Mỗi EPROM có 12 đường địa chỉ từ A0 đến A11 nên số byte lưu trữ trong từng EPROM sẽ là 2^{12} hay 4096 Byte



Hình P3.3 Các chip ROM và bộ giải mã 74LS138.

Chú ý rằng mỗi ROM 2732 trong hình đều có một đầu vào chọn chip $\overline{CS}$ (chip select). Khi đầu vào $\overline{CS}$ được đặt ở mức tích cực thấp thì byte được định địa chỉ trong *ROM* sẽ được gửi ra bus dữ liệu. Chip 74LS138 sẽ đảm bảo rằng lối vào $\overline{CS}$ của duy nhất một chip ROM được đặt xuống thấp tại một thời điểm.

Nếu đặt đầu vào $\overline{G2A}$, $\overline{G2B}$ thấp và G1 cao thì sẽ kích hoạt 74LS138 hoạt động nhưng chỉ có một lối ra của 74LS138 là ở mức thấp. Lối ra nào sẽ ở mức thấp là do các lối vào chọn C, B, A quyết định. Ví dụ nếu CBA là 000 thì lối ra Y0 sẽ ở mức thấp và các lối ra còn lại ở mức

cao. Y0 thấp sẽ làm $\overline{CS}$ của ROM0 ở mức thấp và ROM0 được chọn. Nếu CBA là 001 thì lối ra Y1 thấp và ROM1 được chọn. Nếu CBA là 111 thì ROM7 được chọn v.v...

Xét dải địa chỉ mà các đầu nối trên 74LS138 quy định cho từng ROM trong hệ thống. Để xác định địa chỉ của ROM, RAM và các cổng trong hệ thống thì cách tiếp cận tốt nhất là sử dụng bảng P3.1. Trước tiên phải viết bit địa chỉ và trọng số của nó theo hàng ngang trên cùng của trang giấy. Để biến đổi dễ dàng từ số nhị phân sang số hex thì ta nhóm bốn đường địa chỉ thành một. Tiếp theo vẽ đường thẳng đứng để đánh dấu riêng ba đường địa chỉ nối với ba lối vào C, B, A của bộ giải mã. Với bộ giải mã trong hình P3.1, các đường địa chỉ A14, A13, A12 được nối theo thứ tự vào C, B, A. Cuối cùng viết dưới các bit địa chỉ mức logic để định địa chỉ ô nhớ đầu tiên trong EPROM đầu tiên.

Bảng P3.1 **Bảng giải mã địa chỉ cho 8 chip ROM 2732**

		HEX DIGIT				HEX DIGIT				HEX DIGIT				HEX DIGIT				Địa chỉ tương đương (HEX)
		2^{15} A15	2^{14} A14	2^{13} A13	2^{12} A12	2^{11} A11	2^{10} A10	2^{9} A9	2^{8} A8	2^{7} A7	2^{6} A6	2^{5} A5	2^{4} A4	2^{3} A3	2^{2} A2	2^{1} A1	2^{0} A0	
BLOCK 1	START	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 0000
	END	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	= 0FFF
BLOCK 2	START	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	= 1000
	END	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	= 1FFF
BLOCK 3	START	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	= 2000
	END	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	= 2FFF
BLOCK 4	START	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	= 3000
	END	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= 3FFF
BLOCK 5	START	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	= 4000
	END	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	= 4FFF
BLOCK 6	START	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	= 5000
	END	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	= 5FFF
BLOCK 7	START	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	= 6000
	END	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	= 6FFF
BLOCK 8	START	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	= 7000
	END	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	= 7FFF

Các lối vào địa chỉ bộ giải mã

Để định địa chỉ cho ô nhớ đầu tiên trong bất cứ một EPROM nào thì các đường địa chỉ từ A0 - A11 phải là thấp. Vì vậy viết 0 dưới các bit địa chỉ trong bảng. Để kích hoạt EPROM 0 thì lối vào chọn C, B, A của bộ giải mã tất cả phải là 0. Bởi vì các đường địa chỉ A14, A13, A12 được nối với C, B, A của bộ giải mã nên tất cả cũng là 0. Viết số 0 dưới các bit này. Đường địa chỉ A15 nối vào $\overline{G2A}$ của bộ giải mã nên A15 sẽ là 0 để bộ giải mã hoạt động. Viết 0 dưới bit A15. Chú ý rằng tín hiệu $\overline{RD}$ từ bus điều khiển của vi xử lí nối vào $\overline{G2B}$ nên bộ giải mã chỉ được kích hoạt trong quá trình đọc. Điều này đảm bảo rằng dữ liệu không thể tình cờ được viết vào ROM. Lối vào G1 của bộ giải mã được nối cố định vào nguồn +5V bởi vì ta không cần nó cho bất cứ thao tác nào khác.

Ta cũng có thể đọc địa chỉ bắt đầu của EPROM 0 trực tiếp từ bảng là 0000h. Địa chỉ cao nhất trong EPROM 0 là khi A0 - A11 là 1 nên địa chỉ sẽ là 0FFFh. Nhớ rằng A12-A14 là 0 để chọn EPROM 0. Dải địa chỉ của EPROM là từ 0000h đến 0FFFh, 4kbyte.

Ta dùng bảng để xác định dải địa chỉ của EPROM 1. EPROM 1 được kích hoạt khi A14, A13 là 0 và A12 là 1. Địa chỉ đầu tiên của EPROM 1 là khi A1-A11 là 0 nên địa chỉ là 1000h. Địa chỉ kết thúc của EPROM 1 là khi các A0-A11 là 1 nên địa chỉ kết thúc của ROM 1 là 1FFFh. Nhìn vào bảng ta thấy dải địa chỉ của sáu EPROM còn lại là 2000h-2FFFh, 3000h-3FFFh, 4000h-4FFFh, 5000h-5FFFh, 6000h-6FFFh, 7000h-7FFFh. Trong hệ thống này ta dùng các đường địa chỉ A14, A13, A12 để chọn một trong tám EPROM có tổng dải địa chỉ từ

0000h-7FFFh. Nếu nghĩ các đường địa chỉ là đầu ra của một bộ đếm 16 bit thì sẽ hiểu cách thức mà các bước trên thực hiện. Địa chỉ cuối cùng của các EPROM là tất cả các bit A0-A11 đều là 1. Khi tăng địa chỉ để truy cập vào byte tiếp theo trong bộ nhớ thì các bit A0-A11 xuống 0 và số trong ba bit A14, A13, A12 sẽ tăng lên 1. Số đếm trong ba bit A14, A13, A12 sẽ đi từ 000 đến 111.

Tờ 1 của sơ đồ SDK-86 hình P3.2a trình bày mối liên hệ giữa EPROM và bộ giải mã EPROM. Có 8 chip EPROM 2716/ 2 kbyte. Hai trong tám chip này có 8 đầu ra dữ liệu nối song song với bus dữ liệu hệ thống D0-D7. Hai EPROM này đưa ra 4 kbyte nhớ trong băng nhớ thấp. Hai EPROM khác có đầu ra nối song song với bus dữ liệu D8-D15 và đưa ra 4 kbyte nhớ trong băng cao.

Ta cần mười một đường địa chỉ để định địa chỉ cho 2 kbyte trong từng EPROM. Do đó, các đường địa chỉ của hệ thống A1-A11 được nối song song với tất cả EPROM. Nhớ rằng A0 không thể dùng để chọn byte trong bộ nhớ vì nó dùng để kích hoạt băng nhớ thấp (chứa địa chỉ chẵn).

Mỗi chip 2716 có hai đầu vào $\overline{CE}$ và $\overline{OE}$ được dùng trong xuất ra byte dữ liệu được chỉ định, cả hai đầu vào này phải đặt ở mức thấp. Đầu vào $\overline{CE}$ của hai EPROM trong băng nhớ thấp được nối với đường địa chỉ A0 nên $\overline{CE}$ sẽ thấp khi A0 thấp. Đầu vào $\overline{CE}$ của hai EPROM trong băng nhớ cao được nối với tín hiệu $\overline{BHE}$ nên đầu vào $\overline{CE}$ được xác lập bất cứ khi nào $\overline{BHE}$ ở mức thấp. Tóm lại, hai EPROM có kí hiệu A27 và A36 tạo thành băng nhớ thấp (chứa địa chỉ chẵn) và hai EPROM có kí hiệu A30, A37 tạo thành băng nhớ cao (chứa địa chỉ lẻ). Để biết lối vào $\overline{OE}$ của các EPROM được xác lập như thế nào và biết địa chỉ các EPROM nhận được trong hệ thống ra sao thì cần phải xem bộ giải mã địa chỉ 3625 có kí hiệu A26 trong tờ 1 của hình P3.2a.

3625 là một EPROM 1 kbyte x 4 có chức năng như bộ giải mã địa chỉ (giống chức năng 74LS138 trong hình 4.10). Bởi vì 3625 có các đầu ra collector hở nên cần một điện trở nối giữa nguồn +5V và các đầu ra này. Hình chữ nhật có nét đứt bao quanh 4 điện trở trong sơ đồ tờ 1 hình 4.6a có nghĩa là các điện trở này được "đóng gói" trong một khối- khối điện trở 5 (RP5). 3625 sẽ dịch một địa chỉ thành tín hiệu dùng để kích hoạt thiết bị mong muốn. Tuy nhiên, sử dụng PROM làm bộ giải mã địa chỉ sẽ mạnh và tốt hơn nhiều dùng bộ giải mã đơn giản 74LS138. Thứ nhất vì 3625 có thể lập trình được, điều này có nghĩa là ta có thể di chuyển bộ nhớ đến một địa chỉ khác đơn giản chỉ bằng cách lập trình PROM mới. Thứ hai, nhiều đầu vào của PROM cho phép ta chọn vùng nhớ bất kì không cần sử dụng các mạch cổng lôgic bên ngoài. Thí dụ nếu ta muốn lối vào $\overline{G2A}$ của 74LS138 được xác lập khi A11-A15 là cao thì phải dùng cổng NAND bên ngoài để dò tìm trạng thái này. Với PROM, ta chỉ cần tạo ra trạng thái này từ bảng chân lí.

Để phân tích mạch giải mã địa chỉ, trước tiên phải xác định tín hiệu nào cần để kích hoạt bộ giải mã. Đầu vào $\overline{CS1}$ của bộ giải mã EPROM 3625 được nối xuống đất để bộ giải mã luôn được kích hoạt. Đầu vào $\overline{CS2}$ nối với tín hiệu $\overline{RD}$ từ 8086 để bộ giải mã chỉ bị kích hoạt khi 8086 đang thực hiện quá trình đọc. Bởi vì như đã giải thích ở phần trước ta không muốn tình cờ kích hoạt ROM khi ta gửi ra địa chỉ sai trong quá trình viết vào bộ nhớ hay cổng.

Bước tiếp theo, là tra cứu sách hướng dẫn. Điều này là cần thiết vì mối quan hệ giữa đầu vào và đầu ra của bộ giải mã không thể xác định trực tiếp từ sơ đồ.

P3.2 là bảng chân lí của PROM lấy ra từ sách hướng dẫn SDK-86.

Bảng P3.2 **Bảng chân lí cho bộ giải mã PROM A26**

Các lối vào PROM				Các lối ra PROM				Dải địa chỉ PROM được chọn
M/$\overline{IO}$	A14-A19	A13	A12	O4	O3	O2	O1	
1	1	1	1	1	1	1	0	FF000H-FFFFFH
1	1	1	0	1	1	0	1	FE000H-FEFFFH
1	1	0	1	1	0	1	1	FD000H-FDFFFH ($\overline{CSX}$)
1	1	0	0	0	1	1	1	FC000H-FCFFFH ($\overline{CSY}$)
Tất cả các trạng thái khác				1	1	1	1	NONE

Bảng chân lí này chỉ là dạng viết ngắn lại của bảng mà ta đã thảo luận ở phần trước về giải mã địa chỉ. Từ bảng chân lí ta có thể thấy rằng để lối ra O1 của 3625 được xác lập thấp thì tín hiệu M/ $\overline{IO}$ phải là cao. Việc này là hợp lí vì bộ giải mã đang kích hoạt bộ nhớ chứ không phải thiết bị vào/ra. Đường địa chỉ A12-A19 phải là cao để lối ra O1 của PROM được xác lập xuống thấp. Do O1 được xác lập thấp khi mà tất cả 8 bit địa chỉ cao A12-A19 là cao nên địa chỉ thấp nhất là FF000h. Nếu nhìn lại tờ 1 hình P3.2a, ta thấy lối ra O1 được nối với đầu vào $\overline{OE}$ của hai EPROM 2716, A27 và A30. Đầu vào $\overline{OE}$ được kích hoạt bất cứ khi nào 8086 gửi ra một địa chỉ trong dải FF000h-FFFFFh. Tuy nhiên để kích hoạt các chip PROM thì lối vào $\overline{CE}$ cũng phải được xác lập. Đầu vào $\overline{CE}$ của EPROM A27 được nối vào dây A0, vì vậy EPROM này được kích hoạt bất cứ khi nào 8086 đọc bộ nhớ từ địa chỉ chẵn (A0 = 0) trong dải FF000h-FFFFFh.

Đầu vào $\overline{CE}$ của EPROM A30 được nối với đường $\overline{BHE}$. $\overline{BHE}$ sẽ được xác lập mức thấp bất cứ khi nào 8086 truy cập một byte ở địa chỉ lẻ hay cả một từ ở địa chỉ chẵn. Do đó, EPROM A30 sẽ được kích hoạt khi 8086 đọc byte từ địa chỉ lẻ trong dải địa chỉ FF000h-FFFFFh. EPROM A30 cũng được kích hoạt khi 8086 xác lập cả A0 và $\overline{BHE}$ thấp để đọc từ bất đầu từ địa chỉ chẵn.

Trong hình P3.1 và tờ 1 trong hình P3.2a, lối ra O2 của bộ giải mã PROM 3625 sẽ được kích hoạt khi địa chỉ trong dải FE000h-FEFFFh. Tín hiệu O2 cùng với A0 và $\overline{BHE}$ được dùng để kích hoạt cho EPROM A36 và A37, giống ta đã miêu tả trong phần trước cho thiết bị A27 và A30.

Chú ý trong tờ 1 hình P3.2a, bộ giải mã ROM 3625 có hai lối ra không dùng nhưng ta có thể dùng nó để kích hoạt EPROM mở rộng được thêm vào bản mạch. Như chỉ ra trong hình P3.4, dải địa chỉ dùng cho hai lối ra này là FD000h-FDFFFh và FC000h-FCFFFh.

Bốn EPROM trong SDK-86 thực sự chứa hai chương trình giám sát (monitor programs). Một chương trình trong EPROM A27, A30 cho phép sử dụng bàn phím hex để nhập và chạy chương trình. Chương trình kia chứa trong A36 và A37 cho phép sử dụng thiết bị CRT bên ngoài để nhập và chạy chương trình. Các EPROM được đặt ở địa chỉ cao trong bộ nhớ bởi vì sau khi RESET 8086 đến địa chỉ FFFF0h để nhận lệnh đầu tiên. Vì ta muốn SDK-86 thực hiện chương trình giám sát sau khi ấn nút RESET nên ta đặt EPROM chứa chương trình này tại vị trí sao cho EPROM có địa chỉ FFFF0h. Ta có thể thay đổi chip EPROM thực sự sao cho hoặc chương trình giám sát bàn phím hoặc chương trình giám sát nối tiếp (serial monitor) thực hiện khi ấn nút RESET.

c, Giải mã RAM trong SDK - 86

Tờ 6 trong sơ đồ hình P3.2f của SDK-86 chỉ ra mạch điện của hệ thống RAM và bộ giải mã RAM. Trước tiên, nhìn vào đầu vào và đầu ra của thiết bị RAM tĩnh 2142. Từ thực tế rằng mỗi chip RAM có 4 đường dữ liệu vào/ra ta kết luận rằng RAM lưu trữ dữ liệu 4 bit. Trong khi đó,

mỗi RAM có 10 đường địa chỉ, A0-A9 nên mỗi RAM có thể chứa 2^{10} hay 1024 từ dữ liệu 4 bit. Để có thể lưu trữ được byte (8 bit) thì hai RAM 2142 được kích hoạt song song. Ví dụ A38, A41 được hích hoạt cùng nhau để lưu trữ byte từ 8 đường địa chỉ thấp và A43, A45 được kích hoạt cùng nhau để lưu trữ byte dữ liệu của 8 đường địa chỉ cao. Các tín hiệu bus điều khiển $\overline{RD}$, $\overline{WR}$, M/ $\overline{IO}$ được nối tất cả vào 2142, $\overline{RD}$ được nối với chân *OD* (output disable) của 2142. Khi tín hiệu $\overline{RD}$ cao hay RAM không được kích hoạt thì đầu ra bộ đệm bị cấm. Khi đang đọc tín hiệu $\overline{RD}$ được xác lập là mức thấp. Khi 2142 được kích hoạt và OD thấp thì đầu ra bộ đệm sẽ được mở để từ dữ liệu gửi ra trên bus dữ liệu. Tín hiệu $\overline{WR}$ được nối từ 8086 vào chân $\overline{WR}$ (write enable) của 2142. Nếu 2142 được kích hoạt thì dữ liệu trên bus dữ liệu sẽ được viết vào ô nhớ trong RAM khi 8086 xác lập $\overline{WR}$ xuống mức thấp.

2142 có hai đầu vào $\overline{CS1}$, $\overline{CS2}$. Tín hiệu *M/* $\overline{IO}$ được nối với đầu vào *CS2* của tất cả 2142. Bởi vì lối vào *CS2* có mức tích cực dương nên nó được xác lập bất cứ khi nào 8086 làm việc với bộ nhớ. Lối vào $\overline{CS1}$ được nối thành từng cặp để lối ra của PROM 3625 có chức năng như bộ giải mã địa chỉ.

Để xác lập bất cứ đầu ra nào của 3625 và kích hoạt RAM thì 3625 phải tự nó được kích hoạt. Vì đầu vào $\overline{CS}$ 2 của PROM 3625 được gắn xuống đất nên PROM luôn được kích hoạt. Đầu vào $\overline{CS}$ 1 được xác lập khi đường địa chỉ A19 là thấp. Để tìm hiểu thêm thông tin về PROM thì ta cần nhìn vào bảng chân lí cho chip. Tuy nhiên trước khi ta tiếp tục, chú ý rằng A0 và $\overline{BHE}$ nối với hai đầu vào địa chỉ của PROM 3625. Lí do lại dùng A0 và $\overline{BHE}$ là một phần để xác định đầu ra của bộ giải mã được giải thích trên vào bảng chân lí P3.3.

Bảng P3.3 **Bảng chân lí cho bộ giải mã PROM (A29).**

Các lối vào PROM				Các lối ra PROM				Các byte được chọn (khối địa chỉ)
A12-A18	A11	$\overline{BHE}$	A0	O4	O3	O2	O1	
0	0	0	0	1	1	0	0	2 BYTES (0H-07FFH)
0	0	0	1	1	1	0	1	Byte cao (0H-07FFH)
0	0	1	0	1	1	1	0	Byte thấp (0H-07FFH)
0	1	0	0	0	0	1	1	2 BYTES (0800H-0FFFH)
0	1	0	1	0	1	1	1	Byte cao (0800H-0FFFH)
0	1	1	0	1	0	1	1	Byte thấp (0800H-0FFFH)
Tất cả các trạng thái khác				1	1	1	1	không làm gì

Theo như dòng thứ ba của bảng bộ giải mã, lối ra O1 của PROM được xác lập thấp nếu A12-A18 là thấp, A11 là thấp, $\overline{BHE}$ là cao và A0 là thấp. Lối ra O1 được xác lập tại địa chỉ chẵn bắt đầu từ 00000h. Lối ra O1 thấp thì sẽ kích hoạt RAM A38 và A41 (RAM nối với bus địa chỉ thấp). Hai RAM này là một phần của băng nhớ RAM thấp (chứa địa chỉ chẵn).

Từ dòng thứ hai của bảng chân lí *PROM* trong hình, lối ra O2 của PROM sẽ được xác lập thấp nếu A12-A18 là thấp, A11 là cao, $\overline{BHE}$ là thấp, A0 là cao. O2 được xác lập tại địa chỉ lẻ bắt đầu từ 00001h. Lối ra O2 là thấp sẽ kích hoạt RAM A43 và A45 (RAM này nối với bus địa chỉ cao). Hai RAM là một phần của băng nhớ RAM cao (chứa địa chỉ lẻ).

Giả sử rằng ta muốn viết một từ 16 bit vào RAM tại địa chỉ chẵn. Ta muốn cả O1 và O2 được xác lập là thấp để cả hai băng RAM được kích hoạt. Theo dòng đầu tiên của bảng chân lí, O1 và O2 được kích hoạt nếu cả A0 và $\overline{BHE}$ là thấp. Nhớ rằng A0 và $\overline{BHE}$ là thấp bất cứ khi nào ta viết hay đọc một từ vào địa chỉ chẵn. Hai đầu vào này chỉ ra cho bộ giải mã PROM xác lập cả O1 và O2 cho quá trình đọc hay viết một từ. Dải địa chỉ của RAM A38, A41, A43 và A45 là 00000h-007FFh. Nhìn vào hình 3.10 ta thấy rằng A39, A42, A44 và A46 chứa hơn hai byte

trong dải 00800h-00FFFh. Hai khối nhớ thêm vào của RAM được kích hoạt nếu cả A0 và $\overline{BHE}$ là thấp, khi mà các RAM đang đọc hay viết từ tại địa chỉ chẵn.

d, Định địa chỉ và giải mã các cổng của SDK - 86

Trong hệ thống có vào/ra ánh xạ bộ nhớ (memory-mapped I/O) thì bộ giải mã sẽ định địa chỉ và chọn bộ nhớ. Ưu điểm chính của của vào/ra theo kiểu này là bất cứ một lệnh nào về viết và đọc byte dùng cho bộ nhớ đều được dùng cho cổng. Lệnh: ADD BH,DS:BYTE PTR [437Ah] cũng có thể được dùng cho đọc dữ liệu từ cổng và cộng byte đọc được vào thanh ghi BH. Nhược điểm của phương pháp này là cổng sẽ chiếm một không gian nhớ và không gian này không được dùng cho lưu trữ dữ liệu hay lệnh.

Để tránh việc dùng một phần bộ nhớ hệ thống cho cổng, họ vi xử lí 8086 có một không gian địa chỉ riêng biệt cho cổng. Việc có một không gian địa chỉ riêng cho cổng được gọi là vào/ra tách biệt, bởi vì vùng nhớ riêng biệt này được truy cập trực tiếp bằng lệnh IN và OUT.

Bất cứ khi nào 8086 thực hiện lệnh IN, OUT để truy cập vào bộ nhớ thì không có một thanh ghi đoạn nào có liên quan đến việc tạo ra địa chỉ vật lí mà 8086 gửi đi. Địa chỉ cổng được gửi đi trực tiếp từ 8086 trên bus AD0-AD15 và các đường A16-A19 được đặt bằng 0. Tín hiệu M/ $\overline{IO}$ lúc này được dùng để kích hoạt bộ giải mã bộ nhớ hay bộ giải mã cổng. Nhớ rằng tín hiệu M/ $\overline{IO}$ cao là một điều kiện để kích hoạt ROM và giải mã RAM ta đã đề cập ở phần trước. Còn tín hiệu M/ $\overline{IO}$ là thấp sẽ kích hoạt bộ giải mã cổng.

Khi đang thực hiện lệnh IN, tín hiệu $\overline{RD}$ của 8086 là thấp. Tín hiệu này có thể được dùng để kích hoạt thiết bị vào - được chỉ ra trong địa chỉ. Trong khi thực hiện lệnh OUT tín hiệu $\overline{WR}$ của 8086 là thấp và tín hiệu này được dùng để kích hoạt thiết bị ra - được chỉ ra trong địa chỉ. Vì 8086 gửi ra địa chỉ 16 bit trong quá trình thực hiện vào/ra tách biệt nên nó có thể định địa chỉ bất cứ một trong số 2^{16} hay 65.536 địa chỉ cổng nhập hay cổng xuất.

Một ví dụ về cách định địa chỉ và chọn cổng vào/ra tách biệt trong sơ đồ SDK-86 trong tờ 7 hình P3.2g. Một PROM 3625 khác kí hiệu A26 được dùng để tạo ra tín hiệu chọn chip cho 4 thiết bị vào/ra. Lối ra O1 của PROM dùng để kích hoạt thiết bị ghép nối với màn hình/bàn phím 8279. Lối ra O2 dùng để kích hoạt USART 8251A trong tờ 9 của sơ đồ. 8251A cho phép liên lạc với các hệ thống khác theo cách nối tiếp. Lối ra O3 và O4 nối với hai cổng song song 8255A như trong tờ 5 hình P3.2e. Các cổng này có thể được kích hoạt độc lập để nhập hay xuất dữ liệu dạng byte. Chúng cũng có thể kích hoạt cùng nhau để nhập và xuất dữ liệu dạng word.

Nhìn bộ giải mã PROM 3625 để xác định trong trạng thái nào thì nó được kích hoạt. Đầu vào $\overline{CS2}$ được xác lập khi tín hiệu M/ $\overline{IO}$ là thấp và đầu vào $\overline{CS1}$ được xác lập khi A11-A15 là cao. Bây giờ xem những địa chỉ nào làm cho lối ra của PROM được xác lập.

Nhìn vào bảng chân lí P3.4a, ta thấy lối ra O1 được xác lập thấp khi A5-A15 là cao, A4 là thấp, A3 là cao, A0 là thấp, $\overline{BHE}$ có thể cao hay thấp. Tuy nhiên chú ý rằng chỉ có 8 đường địa chỉ thấp D0-D7 là nối với 8279. Vì vậy, dữ liệu gửi vào hay đọc ra từ 8279 là tại địa chỉ chẵn. Nói cách khác, dữ liệu chỉ gửi dưới dạng byte tới địa chỉ chẵn hay byte thấp của một từ gửi tới địa chỉ chẵn.

Địa chỉ cơ sở của 8279 là FFE8h. Đường địa chỉ A1 nối với 8279 để chọn một trong hai địa chỉ nội bên trong thiết bị 8279 này- A1 là thấp thì chọn địa chỉ FFE8h và A1 là cao thì chọn địa chỉ FFEAh. Hai địa chỉ trên là hai địa chỉ cho 8279 trong hệ thống. Từ bảng chân lí P3.4a, lối ra O2 của bộ giải mã PROM được xác lập là thấp khi A4-A15 là cao, A3 và A0 là thấp, $\overline{BHE}$ có thể là cao hay thấp. Nhưng bởi vì chỉ có 8 đường địa chỉ thấp được nối vào USART 8251A nên dữ liệu đọc ra hay gửi vào dưới dạng byte tại địa chỉ chẵn. Đường dữ liệu A1 được dùng để chọn

một trong hai địa chỉ nội trong 8251A (hình P3.2i, tờ 9). Hai địa chỉ cho thiết bị 8251A trong hệ thống là **FFF0h và FFF2h.**

Bảng P3.4 **Bảng chân lí (a) và bản đồ (b) của bộ giải mã cổng SDK - 86**

Các lối vào PROM						Các lối ra PROM			
A11-A15	A5-A10	A4	A3	$\overline{BHE}$	A0	O4 Chọn cổng cao	O3 Chọn cổng thấp	O2 Chọn USART	O1 KDSEL
1	1	0	1	0	0	1	1	1	0
1	1	0	1	1	0	1	1	1	0
1	1	1	0	0	0	1	1	0	1
1	1	1	0	1	0	1	1	0	1
1	1	1	1	0	0	0	0	1	1
1	1	1	1	0	1	0	1	1	1
1	1	1	1	1	0	1	0	1	1
Tất cả các trạng thái khác						1	1	1	1

(a)

Địa chỉ cổng	Chức năng của cổng
0000 to FFDF	OPEN
FFE8 E9 EA EB EC ED EE FFEF	Đọc/viết RAM display 8279 hoặc đọc FIFO 8279 Đọc trạng thái 8279 hoặc viết lệnh 8279 Dự trữ Dự trữ
FFF0 F1 F2 F3 F4 F5 F6 FFF7	Đọc/viết số liệu 8251A Đọc trạng thái 8251A hoặc viết từ điều khiển 8251A Dự trữ Dự trữ
FFF8 F9 FA FB FC FD FE FFFF	Đọc/viết 8255A cổng P2A Đọc/viết 8255A cổng P1A Đọc/viết 8255A cổng P2B Đọc/viết 8255A cổng P1B Đọc/viết 8255A cổng P2C Đọc/viết 8255A cổng P1C Viết thanh ghi điều khiển P2 Viết thanh ghi điều khiển P1

(b)

Trước khi thảo luận về lối ra O3 và O4 của bộ giải mã PROM, ta nhìn lướt qua hai cổng song song 8255 mà các lối ra O3 và O4 này sẽ kích hoạt chúng (trong tờ 5 hình P3.2e). Mỗi thiết bị 8255 này có 3 cổng song song 8-bit và một thanh ghi điều khiển.

Đường địa chỉ A1 và A2 dùng để định địa chỉ cho cho cổng, thanh ghi mong muốn trong thiết bị 8255 giống như dùng các đường địa chỉ thấp để định địa chỉ cho ô nhớ trong bộ nhớ. Chú ý rằng, 8 đường địa chỉ D0-D7 nối với thiết bị A40 và 8 đường địa chỉ A8-A15 nối với A35. Ta có thể đọc hay viết dữ liệu dạng byte vào cổng địa chỉ chẵn trong A40 hay vào cổng địa chỉ lẻ A35. Ta cũng có thể đọc hay viết một từ 16-bit tạo thành từ cổng A35 và A40. Để nhập và xuất dữ liệu dạng word thì cả hai cổng đồng thời phải được kích hoạt. Nhìn vào bảng chân lí

của bộ giải mã để xác định địa chỉ nào kích hoạt cổng nào. A40 sẽ được kích hoạt bởi lối ra O3 của PROM nếu A3-A15 là cao, A0 là thấp. A1, A2 dùng để chọn cổng nào bên trong 8255A. Để chọn cổng A trong 8255 thì cả hai A1 và A2 đều là thấp. Những trạng thái khác của A1, A2 sẽ chọn các cổng khác hay thanh ghi điều khiển. Bảng P3.4b trình bày địa chỉ hệ thống cho các cổng và thanh ghi điều khiển trong 8255. Chú ý các cổng trong thiết bị A40 được gọi là cổng 2A, cổng 2B và cổng 2C. Tất cả các cổng này có địa chỉ chẵn bởi vì A0 phải là thấp thì thiết bị A40 mới được chọn.

8255A có tên A35 - chứa các cổng 1A, 1B, 1C được kích hoạt bởi lối ra O4 của PROM khi A3-A15 là cao và $\overline{BHE}$ là thấp. Nếu A35 đang được kích hoạt cho việc đọc hay viết thì A0 lên cao. A1 và A2 lại được dùng để định địa chỉ cho cổng hay thanh ghi điều khiển bên trong 8255A. Trong hình 4.32b, địa chỉ của cổng 1A là FFF9h, cổng 1B là FFFBh, cổng 1C là FFFDh và thanh ghi điều khiển là FFFFh.

Như đã nói ở trên thì 8086 có thể nhập giá trị 16-bit bằng cách kích hoạt một cổng chứa dữ liệu thấp và một cổng chứa dữ liệu cao cùng một lúc. Khi 8086 thực hiện tuần tự lệnh : MOV DX,FFF8h - IN AX,DX thì cả A0 và $\overline{BHE}$ là thấp trong lệnh IN thì lối ra O3, O4 của bộ giải mã là thấp. Các tín hiệu này kích hoạt cả A40 và A35. Byte dữ liệu trong cổng 2A là byte thấp còn byte trong cổng 1A là byte cao.

Bảng chân lí P3.4a chỉ ra bộ giải mã PROM 3625 kích hoạt cổng chỉ khi địa chỉ xác định cổng này được gửi từ 8086 đến PROM. Trường hợp này còn được gọi là giải mã hoàn toàn (complete decoding) bởi vì tất cả các đường địa chỉ tham gia chọn cổng hay thanh ghi trong 8255 này.

d, Bộ giải mã "off-board" của SDK-86

Nhìn vào mạch off-board trong vùng A5, tờ 5 sơ đồ hình P3.2e của kit SDK-86. Mạch điện này có mục đích tạo ra tín hiệu $\overline{OFF-BOARD}$ mỗi khi 8086 gửi ra một địa chỉ cổng hay địa chỉ nhớ nhưng địa chỉ này không đúng với thiết bị cần giải mã.

Tín hiệu $\overline{OFF-BOARD}$ được xác lập khi chân 4 hoặc chân 5 của NAND A3 là thấp. Theo bảng chân lí P3.5 của PROM A12 thì đầu ra O1 là thấp nếu 8086 đang thao tác với bộ nhớ và địa chỉ gửi ra không thuộc dải địa chỉ của ROM hay RAM trên bản mạch.

Bảng P3.5 **Bảng chân lí bộ giải mã off-board.**

Các lối vào PROM									Lối ra PROM (Q1)	Khối địa chỉ tương ứng
M/$\overline{IO}$	A19	A18	A17	A16	A15	A14	A13	A12		
1	0	0	0	0	0	0	0	0	1 (không kích hoạt)	0H-0FFFH (ON-BOARD RAM)
1	1	1	1	1	1	1	1	0	1 (không kích hoạt)	FE000H-FEFFFH (ON-BOARD PROM)
1	1	1	1	1	1	1	1	1	1 (không kích hoạt)	FF000H-FFFFFH (ON-BOARD PROM)
Tất cả các trạng thái khác									0 (kích hoạt)	01000H-F0FFFH (OFF-BOARD)

Để chân 4 của NAND- A3 là thấp thì cả hai chân 9 và 10 của A3 phải là cao. Chân 10 là cao nếu 8086 đang thực hiện việc nhập hay xuất dữ liệu. Chân 9 là cao nếu một trong các đầu vào NAND-A19 là thấp. Bởi vì các đường địa chỉ A5-A15 nối với đầu vào của NAND 74LS133 nên tín hiệu trên chân 9-A3 là cao với bất cứ địa chỉ nào nhỏ hơn FFE0h. Nói tóm lại, chân 4 của NAND-A3 được xác lập là thấp trong thao tác I/O và địa chỉ không được bộ giải mã A22 chọn.

Nếu tín hiệu $\overline{OFF-BOARD}$, hay $\overline{INTA}$, hay $\overline{HLDA}$ là thấp thì lối ra cổng NAND-A2 là cao. Nếu tín hiệu $\overline{OFF-BOARD}$ là thấp thì chân 1 của NAND-A3 là cao. Nếu tín hiệu $\overline{DEN}$

cũng được xác lập là thấp thì tín hiệu $\overline{BUFFER-ON}$ được xác lập là thấp. Tín hiệu DEN được xác lập bất cứ khi nào 8086 đọc dữ liệu từ ô nhớ hay cổng. Tín hiệu $\overline{BUFFER-ON}$ dùng để kích hoạt bộ đệm dữ liệu 8286 (A6 và A7) được chỉ ra trên tờ 4 của sơ đồ.

Có thể lắp thêm vào một bộ giải mã vào/ra và các thiết bị khác vào bản mạch SDK-86. Để điều khiển các thiết bị thêm vào này thì bus dữ liệu, bus điều khiển và bus địa chỉ phải có bộ đệm. Bus địa chỉ được đệm bởi bộ chốt địa chỉ 74S373 trong tờ 3, hình P3.2c. Không cần bộ đệm bus dữ liệu và bộ đệm bus điều khiển. Tuy nhiên khi đọc hay viết dữ liệu vào thiết bị bên ngoài thì bus dữ liệu được đệm bởi hai thiết bị 8286, như A6, A7 trong tờ 4. Tín hiệu $\overline{BUFFER-ON}$ là thấp sẽ mở hai bộ đệm 8286 này. Nhớ rằng bộ đệm 8286 là hai chiều nên khi hai bộ đệm này được kích hoạt thì tín hiệu Data Transmit/Receive, DT/$\overline{R}$ từ 8086 xác định bộ đệm thông theo chiều nào. Nếu DT/$\overline{R}$ là cao thì bộ đệm được kích hoạt để viết dữ liệu vào thiết bị ngoài còn khi DT/$\overline{R}$ là thấp thì bộ đệm được kích hoạt để đọc dữ liệu từ các thiết bị bên ngoài.

Bus điều khiển được đệm bởi một thiết bị 8286 tên A11 và 74LS244 tên A8 trong tờ 4, hình P3.2d. Các thiết bị này luôn được kích hoạt để gửi ra tín hiệu điều khiển ngoại trừ trong trạng thái HOLD.

e, Mạch phát trạng thái đợi của SDK-86

Sau khi đã biết cách tín hiệu $\overline{OFF-BOARD}$ được tạo ra như thế nào trong SDK-86, ta sẽ giải thích mạch tạo trạng thái đợi trong tờ 2, sơ đồ P3.2b.

Ta đã biết nếu đầu vào RDY của 8086 là thấp thì 8086 sẽ xen thêm vào một hay nhiều trạng thái đợi trong chu kì máy đang thực hiện. Trong trạng thái đợi thì thông tin trên bus là không đổi. Mục dích chính của xen thêm các trạng thái đợi là để thiết bị bộ nhớ được gọi có thời gian để nhận hay xuất dữ liệu. Trong phần sau ta sẽ xem liệu có cần thêm trạng thái đợi không với tần số xung đồng hồ là cho trước. Bây giờ ta xem cách thức 8086 xen thêm các trạng thái đợi.

Trạng thái đợi được xen vào bằng cách đặt đầu vào RDY1 của 8284 xuống mức thấp. 8284 đồng bộ đầu vào RDY với tín hiệu định thời ở đầu vào RDY của 8086. Đầu vào RDY1 được xác lập là thấp khi tất cả ba đầu vào của NAND A15 là cao. Chân 10 của NAND A15 nối với +5V, chân 11 là cao nếu một trong các đầu vào của NAND trong vùng D7 là thấp. Chân 1 của A15 là thấp bất cứ khi nào 8086 thực hiện việc nhập hay xuất dữ liệu. Chân 2 của A15 là thấp bất cứ khi nào 8086 truy cập cổng hay ô nhớ mà không được giải mã trên cổng. Với cách nối như thế này thì số trạng thái đợi được xen vào khi 8086 đang đọc hay viết vào thiết bị vào/ra khi 8086 đọc hay viết vào thiết bị không được giải mã trên cổng. Nếu cầu nối W39 được nối vào chân 13 của A15 thì chân 11 của A15 sẽ luôn cao. Số trạng thái đợi được chọn bởi W27-W34 sẽ xen vào quá trình đọc hay viết.

Số trạng thái đợi xen vào là do các cầu nối của chân W27-W34 quyết định. Thí dụ nếu cầu nối cắm trên chân W27 thì không có trạng thái đợi nào được xen vào, nối trên chân W28 sẽ có một trạng thái đợi được xen vào, nối trên chân W34 sẽ có 7 trạng thái được xen vào.

Khối phát trạng thái đợi là một thanh ghi dịch 8-bit 74LS164. Tại thời điểm ban đầu của chu kì máy, tín hiệu $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ từ 8086 là cao. Ba tín hiệu này là cao thì làm cho NAND A2 kích hoạt đầu vào clear của thanh ghi dịch và đầu ra thanh ghi dịch là thấp. Một trong các đầu ra này được đảo trạng thái và nối qua cầu nối với chân 9 của NAND A15. Chân 9 và chân 11 ở mức cáo sẽ làm cho đầu vào RDY1 được xuống thấp. Tuy nhiên trạng thái chưa được xen vào trừ khi RDY1 duy trì ở mức thấp đủ dài. Khi tín hiệu $\overline{RD}$, $\overline{WR}$, $\overline{INTA}$ xuống thấp thì đầu vào $\overline{CLR}$ của 74LS164 là cao và thanh ghi dịch hoạt động bình thường. Đầu vào INA và INB

được nạp vào đầu ra QA tại sườn dương của xung định thời. Nếu cầu nối trên chân W27 thì đầu ra QA là cao làm cho đầu vào RDY1 của 8284 lên cao. Trong trường hợp này đầu vào RDY1 chuyển lên mức cao đủ nhanh để không có trạng thái đợi nào được xen vào.

Mức cao nạp vào thanh ghi dịch 74LS164 được dịch từng bước một sang bên phải bởi một chuỗi xung đồng hồ. Khi mức cao đến cầu nối được nối với A25 thì đầu vào RDY1 của 8284 lên cao. 8086 sẽ thoát khỏi trạng thái đợi tại xung đồng hồ tiếp theo. Số trạng thái đợi được quyết định bởi có bao nhiêu lần dịch cho đến khi tới cầu nối.

Tóm lại, 8086 xen số trạng thái đợi mong muốn vào một chu kì máy dùng để truy cập bất cứ thiết bị nào không được định địa chỉ trên bản mạch hay thiết bị vào/ra trên bản mạch. Nếu cầu nối W39 được xác lập thì số trạng thái đợi được xen vào quá trình truy cập trên bản mạch hay off-board. Mục đích của xen trạng thái đợi là có thêm thời gian cho nhập hay xuất.

TÀI LIỆU THAM KHẢO

1. The Peter Norton. *PC Programmer's Bible*. Microsoft Press,. Third edition 1993.
2. Douglas V. Hall. *Microprocessors and interfacing programming and hardware*. Macmillan/McGraw-Hill, 1995.
3. Hans-Peter Messmer . *The indispensable PC hardware book*. Addison-Wesley, 1994.
4. Alan Clements. *Principles of Computer Hardware*. PWS-KENT Publishing Company, 1993.
5. H.M. Staudenmaier (Ed.). *Physics Experiments Using PCs*. Springer-Verlag, 1995.
6. David J. Comer. *Electronis design with integrated circuits*. Addison-Wesley Publising Company., 1999,
7. Joseph J. Carr. *Microcomputer Interfacing, a practical guide for technicians, engineers and scientistc*. Prentice hall, 1991.
8. *Peripheral design handbook*. Intel. 1991.
9. Nguyễn Mạnh Giang. *Lập trình bằng ngôn ngữ assembly cho máy tính PC-IBM*. Nhà xuất bản Giáo dục, 2000.
10. William H. Rig by. *PC Interfacing*. Prentice Hall Inc. 1998.

MỤC LỤC

Lời nói đầu

PHẦN MỘT MÁY VI TÍNH VÀ BỘ VI XỬ LÍ

Chương 1. **Phần cứng và các phép tính số trong máy tính** 6

1.1. Vai trò của máy vi tính trong thực tế 6

1.2. Quan hệ giữa phần cứng và phần mềm – Cấu trúc phân mức của máy tính 10

1.3. Các phép tính số trong máy tính 13

1.4. Các linh kiện điện tử số trong máy tính 24

• Câu hỏi và bài tập. 31

Chương 2. **Cấu trúc của vi xử lí** 34

2.1. Bộ xử lí trung tâm CPU 34

2.2. Thực hiện lệnh máy bên trong vi xử lí 46

2.3. Các bộ xử lí RISC 62

• Câu hỏi và bài tập 71

Chương 3. **Hệ thống máy vi tính với họ vi xử lí 80x86, hoạt động đa nhiệm** 72

3.1. Hệ thống máy vi tính 8086 cơ bản 72

3.2. Các vi xử lí họ 80x86 dùng trong hệ điều hành đa nhiệm 78

3.3. Hiệu năng và độ tin cậy của máy vi tính 100

• Câu hỏi và bài tập. 104

Chương 4. **Vi xử lí và bộ nhớ chính** 112

4.1. Các chip nhớ bán dẫn 112

4.2. Bộ nhớ Cache 119

• Câu hỏi và bài tập 124

PHẦN HAI GHÉP NỐI MÁY VI TÍNH VỚI CÁC THIẾT BỊ NGOẠI VI

Chương 5. **Cấu trúc bus** 126

5.1. Cấu trúc các bus 126

5.2. Năng suất truyền dữ liệu 130

• Câu hỏi và bài tập 131

Chương 6. **Các phương pháp thông tin cơ bản giữa CPU và thiết bị ngoại vi** 132

6.1. Định địa chỉ thiết bị ngoại vi 132

6.2. Quá trình móc nối thông tin 132

6.3. Các quá trình trao đổi dữ liệu - Quá trình vào/ra 135

• Câu hỏi, bài tập và thực hành 147

Chương 7. **Ghép nối vi xử lí với các chip bổ trợ** 148

7.1. Các bộ định thời trong máy vi tính 148

7.2. Ngắt cứng với chíp điều khiển ưu tiên ngắt PIC-8259A 150

7.3. Truyền số liệu DMA bằng chip 8237A 160

• Câu hỏi, bài tập và thực hành 170

Chương 8. **Các cổng song song và nối tiếp, ghép nối máy in và Modem** 179

8.1. Các cổng song song và nối tiếp trong máy tính 179

8.2. Cổng song song ghép nối với máy in 179

8.3. Cổng thông tin nối tiếp và ghép nối Modem 183

• Câu hỏi, bài tập và thực hành 199

Chương 9. **Ghép nối bàn phím và chuột** 205

9.1. Bàn phím 205

9.2. Ghép nối chuột 209
• Câu hỏi, bài tập và thực hành 216
Chương 10. **Ghép nối bộ nhớ khối** 218
10.1. Các ổ đĩa và đĩa từ 218
10.2. Ghép nối các ổ đĩa quang 247
• Câu hỏi, bài tập và thực hành 250
Chương 11. **Ghép nối màn hình** 251
11.1. Hiển thị kí tự bằng phương pháp quét 251
11.2. Các màn hình quét mành trong chế độ đồ hoạ 256
11.3. Các chuẩn màn hình thông dụng và phần cứng 261
11.4. Đồ hoạ phân giải cao và bộ xử lí đồ hoạ 266
11.5. Màn hình tinh thể lỏng 267
• Câu hỏi và bài tập 269
Chương 12. **Ghép nối mạng máy tính, Chuẩn ghép nối GPIB-IEE 488** 270
12.1. Truyền thông tin nối tiếp đồng bộ 270
12.2. Ghép nối mạng cục bộ LAN 271
12.3. Chuẩn ghép nối GPIB và Bus IEE-488 274
• Câu hỏi và bài tập 277
Chương 13. **Vào/ra số bằng vi mạch PPI-8255A** 278
13.1. Cấu tạo của vi mạch PPI-8255A 278
13.2. Lập trình với PPI-8255A 279
Chương 14. **Đo và tạo khoảng thời gian chính xác bằng PIT-8253/54** 283
14.1. Cấu tạo và các chân tín hiệu của PIT-8253/54 283
14.2. Lập trình cho 8253/54 284
Chương 15. **Các mạch D/A và A/D** 288
15.1. Bộ biến đổi số-tương tự D/A 288
15.2. Bộ biến đổi tương tự-số A/D 291
Chương 16. **Đo lường và điều khiển tự động bằng kĩ thuật ghép nối máy tính** 298
16.1. Hệ điều khiển tự động 298
16.2. Điều khiển bằng ghép nối máy vi tính 305
Chương 17. **Thực hành trên bản mạch ghép nối đa năng** 307
17.1. Giới thiệu chung 307
17.2. Các câu hỏi và bài tập thực hành 310

PHẦN BA CÁC CÔNG CỤ DÙNG CHO KHẢO SÁT PHẦN CỨNG VÀ THỰC HÀNH GHÉP NỐI MÁY TÍNH

Chương 18. **Các thiết bị điện tử** 325
18.1. Đồng hồ vạn năng (multimeter) 325
18.2. Dao động kí 325
18.3. Đầu dò lôgic 325
18.4. Thiết bị phân tích lôgic 326
Chương 19. **Các công cụ phần mềm** 333
19.1. Gọi ngắt trong các ngôn ngữ 333
19.2. Thâm nhập trực tiếp các cổng vào/ra 335
19.3. Các chương trình dùng cho chẩn đoán 339
Phụ lục 1 Các lệnh máy trong 8086 345
Phụ lục 2 Sơ đồ bộ phân tích lôgic 8 kênh 360
Phụ lục 3 Máy vi tính dùng cho đào tạo Intel SDK-86 361

Chịu trách nhiệm xuất bản :

Chủ tịch HĐQT kiêm Tổng Giám đốc NGÔ TRẦN ÁI
Phó Tổng Giám đốc kiêm Tổng biên tập NGUYỄN QUÝ THAO

Biên tập lần đầu và tái bản :

DƯƠNG VĂN BẰNG

Biên tập kĩ thuật :

LƯU CHÍ ĐỒNG

Trình bày bìa :

NGUYỄN KIM DUNG

Sửa bản in :

DƯƠNG VĂN BẰNG

Chế bản :

TRẦN QUANG VINH

NGUYÊN LÝ PHẦN CỨNG VÀ KĨ THUẬT GHÉP NỐI MÁY VI TÍNH

Mã số : 7B594T7 – DAI

In 1.000 cuốn, khổ 19 x 27 cm, tại Công ty Cổ phần In Phúc Yên
Giấy phép do Cục xuất bản cấp: 11- 2007/CXB/128 - 2119/GD.
In xong và nộp lưu chiểu tháng 1 năm 2007

CÔNG TY CỔ PHẦN SÁCH ĐẠI HỌC - DẠY NGHỀ
HEVOBCO
25 HÀN THUYÊN – HÀ NỘI
Website : www.hevobco.com.vn

TÌM ĐỌC SÁCH THAM KHẢO KĨ THUẬT
CỦA NHÀ XUẤT BẢN GIÁO DỤC

LẬP TRÌNH BẰNG NGÔN NGỮ ASSEMBLY CHO MÁY TÍNH PC - IBM
NGUYỄN MẠNH GIANG

NGÔN NGỮ LẬP TRÌNH C^{++} VÀ CẤU TRÚC DỮ LIỆU
NGUYỄN VIỆT HƯƠNG

CƠ SỞ KĨ THUẬT MẠNG INTERNET
PHẠM MINH VIỆT – TRẦN CÔNG NHƯỢNG

KĨ THUẬT GHÉP NỐI MÁY VI TÍNH
NGUYỄN MẠNH GIANG

CẤU TRÚC MÁY VI TÍNH
TRẦN QUANG VINH

CƠ SỞ ĐỒ HOẠ MÁY VI TÍNH
PHAN HỮU PHÚC

AUTOCAD CHO TỰ ĐỘNG HOÁ THIẾT KẾ
NGUYỄN VĂN HIẾN

Bạn đọc có thể mua tại các Công ti Sách - Thiết bị trường học ở các địa phương hoặc các Cửa hàng của Nhà xuất bản Giáo dục :

Tại Hà Nội : 25 Hàn Thuyên ; 187B Giảng Võ ; 232 Tây Sơn ; 23 Tràng Tiền ;

Tại Đà Nẵng : Số 15 Nguyễn Chí Thanh ; Số 62 Nguyễn Chí Thanh ;

Tại Thành phố Hồ Chí Minh : 104 Mai Thị Lựu, Quận 1 ; Cửa hàng 451B - 453, Hai Bà Trưng, Quận 3 ; 240 Trần Bình Trọng – Quận 5.

Tại Thành phố Cần Thơ : Số 5/5, đường 30/4 ;

Website : www.nxbgd.com.vn

8 934980 758267

Giá : 41.000 đ