



SỞ GIÁO DỤC VÀ ĐÀO TẠO HÀ NỘI

GIÁO TRÌNH

Kỹ thuật vi xử lý

DÙNG TRONG CÁC TRƯỜNG TRUNG HỌC CHUYÊN NGHIỆP



NHÀ XUẤT BẢN HÀ NỘI

SỞ GIÁO DỤC VÀ ĐÀO TẠO HÀ NỘI

KS. CHU KHÁC HUY

GIÁO TRÌNH **KỸ THUẬT VI XỬ LÝ**

(Dùng trong các trường THCN)

NHÀ XUẤT BẢN HÀ NỘI - 2006

Lời giới thiệu

*N*ước ta đang bước vào thời kỳ công nghiệp hóa, hiện đại hóa nhằm đưa Việt Nam trở thành nước công nghiệp văn minh, hiện đại.

Trong sự nghiệp cách mạng to lớn đó, công tác đào tạo nhân lực luôn giữ vai trò quan trọng. Báo cáo Chính trị của Ban Chấp hành Trung ương Đảng Cộng sản Việt Nam tại Đại hội Đảng toàn quốc lần thứ IX đã chỉ rõ: “Phát triển giáo dục và đào tạo là một trong những động lực quan trọng thúc đẩy sự nghiệp công nghiệp hóa, hiện đại hóa, là điều kiện để phát triển nguồn lực con người - yếu tố cơ bản để phát triển xã hội, tăng trưởng kinh tế nhanh và bền vững”.

Quán triệt chủ trương, Nghị quyết của Đảng và Nhà nước và nhận thức đúng đắn về tầm quan trọng của chương trình, giáo trình đối với việc nâng cao chất lượng đào tạo, theo đề nghị của Sở Giáo dục và Đào tạo Hà Nội, ngày 23/9/2003, Ủy ban nhân dân thành phố Hà Nội đã ra Quyết định số 5620/QĐ-UB cho phép Sở Giáo dục và Đào tạo thực hiện đề án biên soạn chương trình, giáo trình trong các trường Trung học chuyên nghiệp (THCN) Hà Nội. Quyết định này thể hiện sự quan tâm sâu sắc của Thành ủy, UBND thành phố trong việc nâng cao chất lượng đào tạo và phát triển nguồn nhân lực Thủ đô.

Trên cơ sở chương trình khung của Bộ Giáo dục và Đào tạo ban hành và những kinh nghiệm rút ra từ thực tế đào tạo, Sở Giáo dục và Đào tạo đã chỉ đạo các trường THCN tổ chức biên soạn chương trình, giáo trình một cách khoa học, hệ

thống và cập nhật những kiến thức thực tiễn phù hợp với đối tượng học sinh THCS Hà Nội.

Bộ giáo trình này là tài liệu giảng dạy và học tập trong các trường THCS ở Hà Nội, đồng thời là tài liệu tham khảo hữu ích cho các trường có đào tạo các ngành kỹ thuật - nghiệp vụ và đông đảo bạn đọc quan tâm đến vấn đề hướng nghiệp, dạy nghề.

Việc tổ chức biên soạn bộ chương trình, giáo trình này là một trong nhiều hoạt động thiết thực của ngành giáo dục và đào tạo Thủ đô để kỷ niệm “50 năm giải phóng Thủ đô”, “50 năm thành lập ngành” và hướng tới kỷ niệm “1000 năm Thăng Long - Hà Nội”.

Sở Giáo dục và Đào tạo Hà Nội chân thành cảm ơn Thành ủy, UBND, các sở, ban, ngành của Thành phố, Vụ Giáo dục chuyên nghiệp Bộ Giáo dục và Đào tạo, các nhà khoa học, các chuyên gia đầu ngành, các giảng viên, các nhà quản lý, các nhà doanh nghiệp đã tạo điều kiện giúp đỡ, đóng góp ý kiến, tham gia Hội đồng phản biện, Hội đồng thẩm định và Hội đồng nghiệm thu các chương trình, giáo trình.

Đây là lần đầu tiên Sở Giáo dục và Đào tạo Hà Nội tổ chức biên soạn chương trình, giáo trình. Dù đã hết sức cố gắng nhưng chắc chắn không tránh khỏi thiếu sót, bất cập. Chúng tôi mong nhận được những ý kiến đóng góp của bạn đọc để từng bước hoàn thiện bộ giáo trình trong các lần tái bản sau.

GIÁM ĐỐC SỞ GIÁO DỤC VÀ ĐÀO TẠO

Lời nói đầu

Trong những năm gần đây, kỹ thuật vi xử lý ở nước ta đã phát triển mạnh mẽ. Nó được ứng dụng rộng rãi trong nhiều lĩnh vực. Vì vậy việc đào tạo đội ngũ cán bộ kỹ thuật và công nhân lành nghề có khả năng sử dụng, bảo dưỡng sửa chữa các thiết bị có ứng dụng kỹ thuật vi xử lý là một việc rất cần thiết.

Giáo trình kỹ thuật vi xử lý ra đời nhằm cung cấp tài liệu học tập và nghiên cứu cho học sinh các hệ trung học chuyên nghiệp, các ngành kỹ thuật máy tính, ngành kỹ thuật viễn thông và một số ngành nghề khác trong các trường trung học chuyên nghiệp của thành phố Hà Nội. Nội dung của giáo trình được biên soạn theo tinh thần ngắn gọn, cơ bản phù hợp với đối tượng sử dụng.

Nội dung giáo trình gồm có 8 chương:

Chương 1: Khái quát chung về vi xử lý.

Chương 2: Biểu diễn thông tin trong máy tính.

Chương 3: Bộ vi xử lý 8088 và các bộ vi xử lý tiên tiến.

Chương 4: Lập trình hợp ngữ với bộ vi xử lý 8088.

Chương 5: Bộ nhớ.

Chương 6: Ghép nối vi xử lý với thiết bị ngoại vi.

Chương 7: Các mạch ghép nối phụ trợ khác.

Chương 8: Vi điều khiển.

Trong quá trình biên soạn giáo trình này, chúng tôi đã nhận được những đóng góp quý báu từ các đồng nghiệp, các chuyên gia ở các cơ sở sản xuất và các giảng viên đang giảng dạy ở các trường đại học. Đặc biệt là sự giúp đỡ của Thạc sĩ Nguyễn Hoàng Dũng và Thạc sĩ Dương Văn Phương, giảng viên trường Đại học Bách khoa Hà Nội đã có những ý kiến đóng góp rất quý báu cho kết cấu và nội dung của giáo trình. Chúng tôi xin chân thành cảm ơn tới tất cả các quý vị.

Mặc dù đã cố gắng nhưng chắc chắn không tránh khỏi những sai sót. Rất mong nhận được những ý kiến đóng góp từ bạn đọc để giáo trình ngày càng hoàn chỉnh hơn.

TÁC GIẢ

Bài mở đầu

GIỚI THIỆU MÔN HỌC KỸ THUẬT VI XỬ LÝ

1. Mục đích của môn học

Giới thiệu tổng quan một hệ thống vi xử lý dựa trên bộ vi xử lý cơ bản 8088, giới thiệu cấu trúc và các thiết bị phụ trợ của hệ vi xử lý và ứng dụng của hệ vi xử lý trong thực tế.

2. Yêu cầu của môn học

Học sinh cần nắm được mục đích, yêu cầu và nội dung của môn học để có được một cách nhìn tổng quát về môn học, từ đó rút ra được phương pháp học tập và nghiên cứu môn học một cách có hiệu quả.

3. Đối tượng của môn học

Môn học được sử dụng giảng dạy cho học sinh hệ trung học chuyên nghiệp, các ngành kỹ thuật máy tính, kỹ thuật viễn thông và có thể giảng dạy cho các ngành đo lường, điều khiển, tự động hoá,...

4. Nội dung của môn học

Môn học được chia làm các chương:

Chương 1: Khái quát chung về vi xử lý. Giới thiệu cho học sinh về lịch sử ra đời và phát triển các hệ vi xử lý, ứng dụng điển hình của hệ vi xử lý để xây dựng thành một hệ thống máy tính.

Chương 2: Biểu diễn thông tin trong máy tính. Giới thiệu các loại tín hiệu làm việc với máy tính, phương pháp biểu diễn số và thực hiện tính toán các phép toán trong máy tính.

Chương 3: Bộ vi xử lý 8088 và các bộ vi xử lý tiên tiến. Giới thiệu cấu trúc chung của một hệ vi xử lý dựa trên một hệ vi xử lý điển hình 8088 của Intel, giới thiệu tổng quát các hệ vi xử lý tiên tiến.

Chương 4: Lập trình hợp ngữ với bộ vi xử lý 8088. Giới thiệu phương pháp, cách thức lập trình để sử dụng và điều khiển một hệ vi xử lý thông qua lập trình cho hệ vi xử lý 8088.

Chương 5: Bộ nhớ. Giới thiệu cấu tạo, ứng dụng và các loại bộ nhớ bán dẫn được sử dụng trong hệ thống vi xử lý.

Chương 6: Ghép nối vi xử lý với thiết bị ngoại vi. Giới thiệu các phương pháp ghép nối và các phương pháp điều khiển ghép nối giữa một hệ vi xử lý với thiết bị ngoại vi để hình thành cho học sinh một phần nguyên lý hoạt động của hệ thống vi xử lý.

Chương 7: Các mạch ghép nối phụ trợ khác. Giới thiệu các vi mạch ghép nối phụ trợ cho một hệ thống vi xử lý để thực hiện ghép nối giữa vi xử lý với các thiết bị ngoại vi để tạo thành một hệ thống hoàn chỉnh.

Chương 8: Vi điều khiển. Giới thiệu các bộ vi xử lý được sử dụng rộng rãi trong công nghiệp, trong các thiết bị dân dụng và chuyên dụng để học sinh có thể khai thác và sử dụng chúng vào các ứng dụng thực tế.

5. Phương pháp học tập môn học

Học sinh cần chuẩn bị tài liệu và các dụng cụ học tập phục vụ cho môn học đầy đủ, nên nghiên cứu nội dung của bài học trước để tiếp thu bài học có hiệu quả hơn.

Học sinh cần học tập môn học theo hướng nghe giảng để hiểu bài mới, phân tích mở rộng kiến thức của bài học, nghiên cứu để ứng dụng vào thực tế (thông qua trao đổi với nhóm học tập hoặc giáo viên, kỹ thuật viên).

Nắm vững kiến thức lý thuyết, làm bài tập trong giáo trình và bài tập mở rộng do giáo viên cung cấp.

Chương 1

KHÁI QUÁT CHUNG VỀ VI XỬ LÝ

Mục tiêu:

- Giới thiệu khái quát về lịch sử phát triển của các bộ vi xử lý cũng như lịch sử phát triển của các thế hệ máy tính để cho học sinh có một tầm nhìn tổng quát về lĩnh vực này, từ đó tìm ra được hướng nghiên cứu và học tập môn học.

- Giới thiệu cấu trúc cơ bản của máy vi tính, một ứng dụng điển hình của bộ vi xử lý để học sinh nhận thấy vai trò và tầm quan trọng của bộ vi xử lý.

Nội dung chính:

1. Lịch sử phát triển của vi xử lý:

- Lịch sử phát triển của vi xử lý
- Lịch sử phát triển của máy vi tính

2. Cấu trúc chung của máy vi tính:

- Cấu trúc Von Neuman
- Cấu trúc chung

I. LỊCH SỬ PHÁT TRIỂN CỦA VI XỬ LÝ

1. Lịch sử phát triển của bộ vi xử lý

Sự phát triển của kỹ thuật vi xử lý được gắn liền với sự phát triển của các vi mạch số. Ngày nay, các thế hệ vi xử lý ra đời gắn liền với sự ra đời của các vi mạch số tổ hợp cấp cao. Sau đây là sự phát triển của chúng.

- *Mạch SSI*: Là loại mạch tổ hợp (IC) cỡ nhỏ, với số linh kiện nhỏ hơn 12, chúng được xây dựng bởi các phần tử cơ bản của kỹ thuật số (AND, OR, NOT...). Trong một thiết bị, giá thành của thiết bị đó sẽ tỷ lệ với số IC có trong thiết bị chứ không tỷ lệ với số phần tử có trong IC.

- *Mạch MSI*: Là mạch tổ hợp có thể chứa hơn 12 phần tử cho đến 90 phần tử logic cơ bản, được xuất hiện cuối năm 1960, người ta gọi đó là các mạch tổ hợp cỡ trung bình.

- *Mạch LSI*: Là mạch tổ hợp cỡ lớn xuất hiện vào năm 1970, được sử dụng trong việc thiết kế các hệ thống số. Đến năm 1971 xuất hiện vi mạch tổ hợp cỡ lớn mà có khả năng lập trình được. Việc ra đời các mạch LSI đã đánh dấu một bước khởi đầu của ngành tin học (vi tin). Cũng vào khoảng thời gian này đã bắt đầu xuất hiện các bộ vi xử lý đầu tiên (Microprocessor), từ đây các bộ vi xử lý phát triển rất nhanh về tốc độ xử lý cũng như số bit dữ liệu được xử lý, tính năng của chúng cũng ngày được mở rộng.

Sau đây là sự phát triển của các bộ vi xử lý qua các giai đoạn khác nhau:

- Vào những năm từ 1971 đến 1973, hãng Intel đã cho ra đời các bộ vi xử lý 4040, 8008 có thể xử lý 4 hoặc 8 bit thông tin. Tốc độ đồng hồ $0,1 \div 0,8$ MHz, đây là đồng hồ tạo nhịp hoạt động của hệ thống.

- Từ năm 1973 đến 1975 xuất hiện các bộ vi xử lý 8 bit như 8080, 6800 với tốc độ xử lý thông tin được tăng lên rất nhiều. Tần số đồng hồ $1 \div 3$ MHz. Trong thời gian này còn xuất hiện một số bộ vi xử lý khác như 8085, Z80 với tập lệnh phong phú hơn, có thể xử lý đến 16 bit dữ liệu một lúc, tần số xung nhịp khoảng 2 MHz. Các bộ vi xử lý thế hệ này được ứng dụng trong các thiết bị điện tử.

- Từ năm 1976 đến 1982, với sự phát triển nhanh chóng, các hãng sản xuất đã cho ra đời các bộ vi xử lý 8086, 8088, 80286, 68000. Các bộ vi xử lý này có tập lệnh phong phú, có thể xử lý thông tin 8, 16, 32 bit. Tần số đồng hồ $1 \div 5$ MHz và được ứng dụng trong các thế hệ máy vi tính XT, AT.

- Từ năm 1982 đến nay, hàng loạt các bộ vi xử lý tiên tiến ra đời với bộ lệnh và tốc độ xử lý thông tin ngày càng được cải thiện nhanh chóng. Đó là các bộ vi xử lý 80386 DX, 80486, 68020, 68030, Pentium, Powerpc... có thể xử lý thông tin 16, 32 bit... và tốc độ lên tới hàng trăm MHz, hàng GHz. Các bộ vi xử lý thế hệ này được sử dụng rộng rãi trong các máy tính hiện đại và trong các hệ thống điện tử tiên tiến.

2. Lịch sử phát triển của máy vi tính

Lịch sử phát triển của các bộ vi xử lý gắn liền với sự ra đời của các vi mạch số thì lịch sử ra đời và phát triển của máy vi tính lại bắt đầu từ thế hệ các bóng đèn điện tử cho đến các linh kiện bán dẫn. Cho đến nay, nếu xét theo sự phát triển công nghệ thì máy tính thường được phân thành 5 thế hệ:

2.1. Thế hệ thứ nhất

Từ 1950 - 1959. Các máy tính ở thế hệ này được xây dựng từ các bóng đèn điện tử. Chúng có năng lực tính toán chậm, tốc độ chỉ đạt hàng ngàn phép tính/giây, đồng thời tiêu thụ điện năng lớn và chiếm nhiều diện tích không gian.

2.2. Thế hệ thứ hai

Từ 1959 - 1963. Đặc trưng cơ bản của máy tính thế hệ này là sử dụng bóng bán dẫn để xây dựng bộ xử lý trung tâm (CPU) và các mạch điện của chúng. Nhờ vậy, các máy tính thế hệ này có tốc độ xử lý tăng đáng kể. Thời gian thực hiện lệnh của máy tính thế hệ này nhanh hơn hàng chục lần so với máy tính thế hệ thứ nhất. Ngoài ra, kích thước và độ tin cậy của chúng cũng được cải thiện đáng kể.

2.3. Thế hệ thứ ba

Từ 1964 - 1974. Máy tính thế hệ này gắn liền với sự xuất hiện và ứng dụng của các mạch vi điện tử, hay được gọi là IC (Integrated Circuit). Chúng đã có kích thước gọn hơn, khả năng tính toán lớn hơn, tốc độ xử lý có thể nhanh gấp hàng ngàn lần so với thế hệ thứ nhất (cỡ hàng triệu phép tính/giây). Việc xử lý các lệnh song song ngày càng được ứng dụng nhiều hơn ở các máy tính thế hệ này.

2.4. Thế hệ thứ tư

Những năm 1974. Đây là thời kỳ của các máy tính với năng lực tính toán lớn. Công nghệ máy tính thời kỳ này liên quan tới việc sử dụng các mạch tích hợp cực lớn VLSI (Very Large Scale Integration) - Chip, IC; với dung lượng trên 100.000 tranzistor/chip cho tới hàng chục triệu tranzistor/chip cho đến ngày nay. Nhờ có công nghệ VLSI mà toàn bộ CPU (bộ vi xử lý), bộ nhớ chính và các thành phần khác có thể được xây dựng gọn trên một chip. Tốc độ xử lý của máy tính ở thế hệ này có thể đạt tới hàng tỷ phép tính/giây.

2.5. Thế hệ thứ năm

Máy tính Neuron-Neural Network- một sản phẩm của kỹ thuật trí khôn nhân tạo được bắt mô phỏng theo cách thức tổ chức các tế bào thần kinh nối với bộ não của con người. Người ta cung cấp những thông tin cho hệ thống máy tính để huấn luyện cho nó nhận biết được các sự vật, những mâu thuẫn có thể xảy ra để đưa ra các dự báo hoặc các giải pháp xử lý thích ứng.

Tuy nhiên, thế hệ máy tính neuron hiện nay đang ở giai đoạn đầu nghiên cứu và phát triển, một số mẫu máy tính thử nghiệm đã xuất hiện trong vài năm trở lại đây và các khái niệm liên quan đang dần hình thành.

Theo sự phát triển của công nghệ, các máy tính hiện nay được thiết kế, xây dựng theo một hướng chung là:

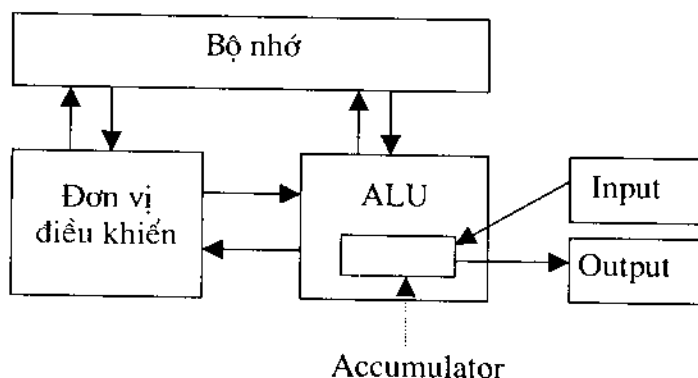
- Mạnh hơn về tốc độ và khả năng tính toán.

- Nhỏ hơn về kích thước.
- Tiết kiệm hơn về năng lượng...

II. CẤU TRÚC CHUNG CỦA MỘT MÁY VI TÍNH

1. Cấu trúc Von NeuMann

Cấu trúc máy tính Von NeuMann là cấu trúc máy tính đầu tiên có chương trình điều khiển được lưu trữ trong bộ nhớ. Cho đến nay, cấu trúc này vẫn là cơ sở cho hầu hết các máy tính số. Cấu trúc Von NeuMann có sơ đồ như sau:



Hình 1.1: Cấu trúc máy tính Von NeuMann

Máy tính Von NeuMann bao gồm 5 thành phần cơ bản, đó là:

- Bộ nhớ (Memory).
- Đơn vị số học và logic (ALU - Arithmetic and Logic Unit).
- Đơn vị điều khiển (CU - Control Unit).
- Thiết bị vào (Input).
- Thiết bị ra (Output).

Bộ nhớ gồm 4096 từ, mỗi từ chứa 40 bit. Như vậy, mỗi từ chứa được hoặc là hai chỉ thị 20 bit hoặc là một số nguyên có dấu dài 39 bit. Các chỉ thị bit có 8 bit dành riêng để chỉ kiểu của chỉ thị, 12 bit để chứa địa chỉ của 1 trong 4096 từ trong bộ nhớ.

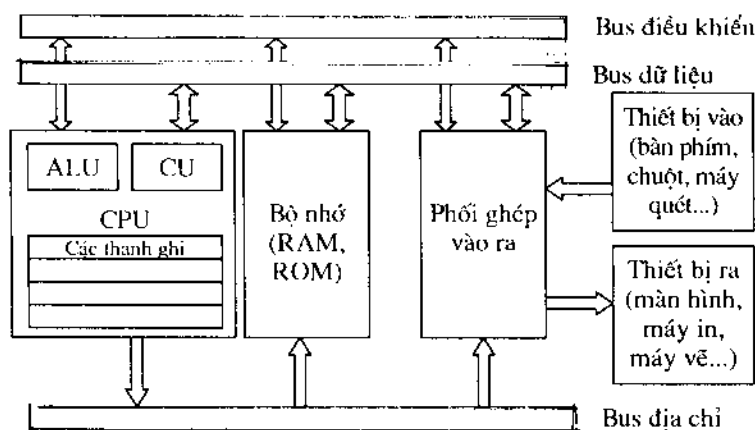
Bên trong ALU, tiền thân của các CPU hiện đại là một thanh ghi 40 bit, có tên gọi là Accumulator. Có một chỉ thị riêng biệt để cộng một từ của bộ nhớ vào Accumulator hoặc chứa nội dung của Accumulator vào bộ nhớ. Cấu trúc này không thực hiện được các phép toán số học và dấu phẩy động.

2. Cấu trúc chung

Để một hệ máy tính có thể hoạt động và thực hiện các chức năng đặt ra thì ngoài cấu trúc phần cứng, máy tính còn phải có phần mềm (để điều khiển phần cứng hoạt động).

Phần cứng là những đối tượng vật lý hữu hình như bảng mạch chính, ổ đĩa cứng, ổ đĩa mềm, bàn phím, màn hình, nguồn nuôi, máy in... Phần mềm là các chương trình của máy tính, chương trình có thể được nhập từ bàn phím, được ghi lên đĩa cứng, đĩa mềm hoặc trên các con chip (vi mạch). Chương trình bao gồm hệ điều hành giúp máy tính quản lý vận hành hệ thống và các chương trình ứng dụng. Tìm hiểu cấu trúc máy tính trước hết là tìm hiểu về cấu trúc phần cứng, tuy nhiên cũng không thể tách rời với các phần mềm.

Tổ chức các khối chức năng của máy tính được thể hiện ở hình vẽ sau:



Hình 1.2: Sơ đồ khối hệ máy tính cơ bản

Cho đến nay, nhiều thế hệ máy tính được xây dựng và phát triển dựa trên những tư tưởng ban đầu của Von Neuman với cấu hình và chủng loại rất đa dạng. Tuy nhiên, mọi hệ thống máy tính, dù là máy tính cá nhân nhỏ hay các hệ máy tính lớn đều có một cấu trúc chung. Một hệ máy tính điển hình bao gồm 4 khối cơ bản: Bộ xử lý trung tâm, bộ nhớ, thiết bị vào, thiết bị ra và nối ghép giữa chúng là hệ thống bus.

2.1. Bộ vi xử lý

Bộ vi xử lý là thành phần chủ đạo nhất của hệ thống máy tính, còn được gọi là bộ xử lý trung tâm CPU (Central Processing Unit). CPU là nơi thực hiện phần lớn các công việc như thực hiện các phép toán, phép tính số học, phép

tính logic và các biến đổi mã tương ứng... Ở các máy tính ngày nay, CPU thường được xây dựng trên một hoặc một vài vi mạch rồi được đặt trong một chip và được gọi là chip vi xử lý (microprocessor).

Hai thành phần cơ bản của CPU đó là: Đơn vị số học và logic - ALU (Arithmetic and Logic Unit) đảm nhiệm xử lý các phép toán số học - logic và đơn vị điều khiển CU (Control Unit) đảm nhiệm các công việc như chuyển lệnh từ bộ nhớ, giải mã, thực hiện lệnh hay chuyển tới ALU khi cần... Bên cạnh các đơn vị ALU, CU, trong CPU còn có tập hợp các thanh ghi. Các bộ vi xử lý hiện đại ngày nay có thể còn có thêm một đơn vị xử lý dấu phẩy động FPU (Floating Point Unit) chuyên để xử lý các số thực và một đơn vị đa phương tiện MU (Multimedia Unit) chuyên xử lý đa phương tiện.

2.2. Bộ nhớ

Bộ nhớ (Memory) hay còn gọi là bộ nhớ chính, bộ nhớ bán dẫn là một bộ phận hết sức quan trọng của máy tính. Bộ nhớ thực hiện việc lưu trữ thông tin để phục vụ cho quá trình trao đổi thông tin. Bộ nhớ này thực chất là bộ nhớ trong chứ không phải là bộ nhớ ngoài (bộ nhớ dùng để lưu trữ dữ liệu như đĩa cứng, đĩa mềm).

Ở trong các máy vi tính, người ta sử dụng bộ nhớ bán dẫn, bao gồm bộ nhớ truy cập ngẫu nhiên RAM (Random Access Memory) hay còn gọi là bộ nhớ ghi đọc và bộ nhớ chỉ đọc ROM (Read Only Memory).

2.3. Phối ghép vào/ra

Khối phối ghép vào/ra (I/O) để tạo khả năng giao tiếp giữa máy tính và thế giới bên ngoài. Các thiết bị ngoại vi như màn hình, bàn phím, máy in, bộ chuyển đổi từ tín hiệu số sang tín hiệu tương tự DAC, bộ chuyển đổi từ tín hiệu tương tự sang tín hiệu số ADC, các thiết bị đo lường điều khiển... đều liên hệ với máy tính qua khối này và được gọi là cổng vào/ra (I/O Port).

2.4. Thiết bị vào/ra

Thiết bị vào thực hiện chuyển đổi tín hiệu tự nhiên do con người tạo ra sang dạng mã máy để máy tính có thể xử lý được. Thiết bị vào thông dụng nhất có thể kể như: Bàn phím, con chuột, máy quét, thiết bị phân tích nhận dạng tiếng nói, CD-ROM...

Thiết bị ra chuyển đổi các mã máy bên trong máy tính sau khi đã xử lý sang các dạng tín hiệu tự nhiên để con người có thể hiểu được hoặc để điều khiển được các thiết bị khác. Thiết bị ra thông dụng nhất có thể kể như: Màn hình, máy in, máy fax, máy vẽ, loa hay các thiết bị điều khiển khác.

Tuy nhiên, ranh giới giữa thiết bị vào và thiết bị ra không phải lúc nào cũng rõ ràng. Nhiều thiết bị vừa đảm nhiệm vai trò là thiết bị vào song cũng thực hiện chức năng là thiết bị ra. Ví dụ như máy fax, CD-ROM đọc/ghi, thiết bị đo - điều khiển...

2.5. Hệ thống BUS

Hệ thống BUS của máy tính thực hiện kết nối các khối chức năng trong hệ thống với nhau. Trong các máy tính hiện đại ngày nay có hai nhóm BUS đó là: BUS hệ thống nối giữa CPU với bộ nhớ chính và các BUS vào ra nối ghép giữa các thiết bị ngoại vi với CPU thông qua các “cầu nối” hay còn gọi là các “cầu”. Tham gia điều khiển BUS có các vi mạch điều khiển BUS, các vi mạch này trước đây nằm riêng lẻ còn hiện nay thường được tích hợp vào trong các vi mạch tổng hợp chipset. Để có thể trao đổi được dữ liệu, hệ thống BUS bao gồm BUS địa chỉ, BUS dữ liệu và BUS điều khiển.

BUS địa chỉ: Phục vụ việc chọn ô nhớ hoặc thiết bị vào/ra. Khi ghi/đọc bộ nhớ hoặc thiết bị vào/ra, bộ xử lý trung tâm sẽ đưa lên BUS này địa chỉ của các thiết bị liên quan. Đây là BUS một chiều và xuất phát từ CPU.

BUS dữ liệu: Được dùng để chuyển dữ liệu và thường có từ 8, 16, 20, 24, 32 đến 64 đường dây tùy thuộc vào từng CPU cụ thể. BUS dữ liệu là loại hai chiều. Các phân tử đầu ra nối thẳng với BUS dữ liệu đều phải được trang bị đầu ra 3 trạng thái để đảm bảo cho BUS hoạt động được bình thường.

BUS điều khiển: Hỗ trợ cho việc trao đổi các thông tin điều khiển và trạng thái, như phân biệt thiết bị được CPU truy nhập là bộ nhớ hay thiết bị vào/ra, thao tác truy nhập là đọc hay viết... BUS điều khiển thường gồm hàng chục dây tín hiệu khác nhau và xét theo cả nhóm thì đó là loại BUS hai chiều.

Câu hỏi ôn tập

1. Hãy tóm tắt lịch sử phát triển của kỹ thuật vi xử lý.
2. Hãy tóm tắt lịch sử phát triển của máy vi tính.
3. Cấu trúc máy tính Von NeuMan gồm những khối nào? Chức năng các khối đó làm gì?
4. Cấu trúc chung một máy tính được chia làm bao nhiêu khối? Hãy trình bày chức năng các khối đó.
5. BUS hệ thống là gì? BUS hệ thống có mấy loại? Đặc điểm của từng loại như thế nào?

Chương 2

BIỂU DIỄN THÔNG TIN TRONG MÁY VI TÍNH

Mục tiêu:

Giới thiệu các loại dữ liệu cơ bản trao đổi với máy tính, cách thức trao đổi dữ liệu giữa máy tính với các thiết bị ngoại vi để học sinh nhận thấy khả năng ứng dụng của bộ vi xử lý và máy tính, sử dụng bộ vi xử lý và máy tính vào các ứng dụng trong đời sống.

Trình bày các cách biểu diễn số trong máy tính và các cách thực hiện các phép toán trong máy tính, để học sinh có thể hình thành được một phần nguyên lý hoạt động của hệ thống máy tính.

Nội dung chính:

- I. Các dạng dữ liệu cơ bản
- II. Biểu diễn số và các phép toán trong máy tính
 - 1. Biểu diễn số dấu phẩy tĩnh
 - 1.1. Các cách biểu diễn số dấu phẩy tĩnh
 - 1.2. Các phép toán với dấu phẩy tĩnh
 - 2. Biểu diễn số dấu phẩy động
 - 2.1. Các cách biểu diễn số dấu phẩy động
 - 2.2. Các phép toán với dấu phẩy động
 - 2.3. Vấn đề tràn số với số dấu phẩy động

I. CÁC DẠNG DỮ LIỆU CƠ BẢN

Trong máy tính, việc trao đổi (nhận và đưa ra) dữ liệu dưới dạng số với các mức logic là 0 và 1 (0V và 5V). Nhưng do nhu cầu trao đổi với bên ngoài nên máy tính có thể làm việc với các loại dữ liệu dưới dạng các dạng tin và loại tin khác nhau.

1. Các dạng tin

* *Dạng số (Digital):*

Là chuỗi các bit 0 hay 1 được biểu diễn theo hệ nhị phân (binary), hệ 8 (octal), hệ 16 (hexadecimal). Đó là tín của bàn phím đơn giản (đóng ngắt mạch mắc nối tiếp một nguồn điện 5V qua một điện trở cỡ 1kΩ). Tín này có thể đưa thẳng vào đường dây số của máy tính, đưa ra các đèn chỉ thị mức bằng điốt phát quang (LED) hay đưa ra đèn chỉ thị 7 đoạn (qua giải mã 2 - 7 đoạn).

** Dạng chữ - số mã ASCII:*

Có nhiều cách biểu diễn tín dạng các chữ (A ÷ Z) và các số (0 ÷ 9) trong đó sử dụng mã quốc tế ASCII là thông dụng. Mỗi chữ cái hoặc con số được biểu diễn bởi tổ hợp 7 hay 8 bit dạng nhị phân (0 hay 1).

Như vậy, bàn phím phải có một bộ phận tạo các mã ASCII khi người điều hành bấm một phím nào đó của bàn phím. Khi đưa tín ra màn hình, mã ASCII này cũng phải được biến đổi thành dạng các chữ và con số để biểu diễn cho người điều hành quan sát.

** Dạng tương tự (Analog):*

Các tín tức vật lý thu được dưới dạng một tín hiệu điện, tức một điện thế U hoặc dòng điện I kéo dài trong một thời khoảng t nào đó. Hàm $U(t)$ là một đại lượng liên tục (hay tương tự) trong thời gian t . Muốn máy tính thu nhận tín hiệu tương tự này, ta phải biến đổi nó thành dạng số (chuỗi 0, 1), tức là phải rời rạc hoá tín hiệu đó theo thời gian và lượng tử hoá theo biên độ (biến đổi A-D). Ngược lại, tín dạng số từ máy tính đưa ra cho thiết bị ngoài phải biến đổi thành tương tự (biến đổi số - tương tự D-A), vì thiết bị ngoài làm việc với tín hiệu tương tự.

** Dạng âm tần:*

Tiếng nói của con người là dạng tín âm tần cũng được máy tính nhận và truyền đi. Như vậy, phải qua một quá trình biến đổi dạng tín hiệu âm tần sang dạng số và ngược lại từ dạng số sang dạng âm tần.

2. Các loại tín

** Máy tính đưa ra thiết bị ngoài về một trong ba loại tín:*

- Tín về địa chỉ: Đó là tín hiệu của địa chỉ để xác định đó là thiết bị ngoại vi, bộ nhớ, hay một bộ phận nào đó của máy tính.

- Tín về điều khiển: Đó là các tín hiệu để điều khiển các thành phần của máy tính, điều khiển các thiết bị khác ghép với máy tính, ví dụ như tín hiệu điều khiển để ghi/đọc bộ nhớ, ghi/đọc thiết bị ngoại vi...

- Tin về số liệu: Đó là các dữ liệu để trao đổi giữa các bộ phận trong máy tính hay dữ liệu trao đổi giữa máy tính với các thiết bị ngoại vi. Ví dụ như dữ liệu về văn bản được in ra máy in, được hiển thị lên màn hình...

** Máy tính nhận tin vào từ thiết bị ngoài về một trong hai loại tin:*

- Tin về trạng thái: Đó là tín hiệu về các trạng thái của các bộ phận trong máy tính, hay trạng thái của các thiết bị ngoài đi cùng với máy tính. Ví dụ như trạng thái của ổ đĩa đọc hay ghi, trạng thái của máy in bận hay không bận...

- Tin về số liệu: Đó là tín hiệu dữ liệu được truyền vào trong máy tính. Ví dụ như đọc tín hiệu từ ổ đĩa để xử lý, tín hiệu lấy từ các hệ thống thiết bị bên ngoài đưa vào máy tính để quản lý...

II. BIỂU DIỄN SỐ TRONG MÁY VI TÍNH

1. Biểu diễn số dấu phẩy tĩnh

1.1. Các cách biểu diễn số dấu phẩy tĩnh

Chúng ta đã biết trong tất cả máy tính hiện nay đều sử dụng hệ thống số nhị phân. Chúng ta xét một thanh ghi n bit để biểu diễn một số, nếu không quan tâm đến dấu của số đang xét thì tất cả n bit của thanh ghi này đều biểu diễn giá trị của số như sau:

b^{n-1}		b_1	b_0
-----------	-------	-------	-------

Cách biểu diễn này người ta gọi là biểu diễn số nguyên không dấu. Như vậy, với n bit có thể biểu diễn các số nguyên trong phạm vi:

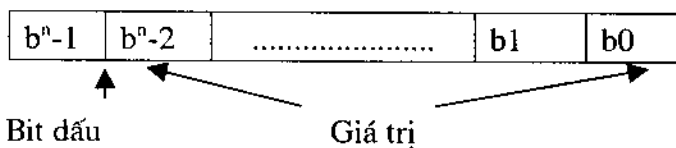
$$0 \dots 2^n - 1$$

Như vậy, nếu chúng ta biểu diễn một số lớn hơn $2^n - 1$ vào thanh ghi n bit thì sẽ xảy ra hiện tượng tràn số.

Khi tính toán, ngoài việc quan tâm đến giá trị, chúng ta thường phải quan tâm đến dấu của các số. Để có thể biểu diễn các số nguyên cả về giá trị và dấu có 3 cách biểu diễn như sau:

1.1.1. Biểu diễn theo dấu và độ lớn (Sign and Magnitude)

Với cách này người ta sử dụng bit tận cùng bên trái của số để biểu diễn dấu, gọi là bit dấu. Nếu bit dấu là 0 thì số được biểu diễn là số dương, nếu bit dấu là 1 thì số được biểu diễn là số âm. Các bit còn lại trừ bit dấu thì vẫn biểu diễn giá trị tuyệt đối của số.



Với thanh ghi n bit, nếu theo cách này thì sẽ biểu diễn được các số nguyên trong phạm vi $-(2^{n-1}-1)$ đến $+(2^{n-1}-1)$ và chỉ có $n-1$ bit để biểu diễn độ lớn của con số. Bảng sau đây minh họa sự biểu diễn các số nguyên 4 bit theo dấu và độ lớn.

Bảng 2.1: Biểu diễn các số theo dấu và độ lớn

Biểu diễn theo dấu và độ lớn	Giá trị biểu diễn
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1000	- 0
1001	- 1
1010	- 2
1011	- 3
1100	- 4
1101	- 5
1110	- 6
1111	- 7

1.1.2. Biểu diễn theo số bù 1

Các số dương được biểu diễn ở đây giống như cách biểu diễn số dương theo dấu và độ lớn và chỉ khác cách biểu diễn với các số âm. Trong cách biểu diễn

này các số âm được tạo ra bằng cách lấy bù từng bit của số dương tương ứng với nhau hoặc lấy hiệu của $(2^n - 1)$ trừ đi số dương tương ứng (n là số bit thanh ghi).

Ví dụ: Để biểu diễn giá trị -3 ta lấy bù từng bit của giá trị $+3$ ở dạng nhị phân như sau:

$$\begin{aligned} 3 &\rightarrow 0\ 0\ 1\ 1 \\ -3 &\leftarrow 1\ 1\ 0\ 0 \end{aligned}$$

Hoặc có thể tìm được số bù 1 biểu diễn -3 bằng cách (với $n=4$):

$$10000 - 1 - 0011 = 1100$$

Bảng sau đây minh họa sự biểu diễn các số nguyên 4 bit theo số bù 1.

Bảng 2.2: Biểu diễn số theo số bù 1

Biểu diễn theo số bù 1	Giá trị biểu diễn
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1000	-7
1001	-6
1010	-5
1011	-4
1100	-3
1101	-2
1110	-1
1111	-0

Qua bảng trên chúng ta thấy các số âm cũng có bit tận cùng bên trái là bit dấu luôn bằng 1, còn các số dương thì bit này luôn là 0. Phạm vi biểu diễn ở đây cũng giống như cách biểu diễn theo dấu và độ lớn.

1.1.3. Biểu diễn theo số bù 2

Các số dương được biểu diễn ở đây giống các cách ở trên và chỉ khác cách biểu diễn với các số âm. Trong cách biểu diễn này, các số âm được tạo ra bằng cách lấy số bù 1 của nó cộng với 1.

Ví dụ: Để biểu diễn giá trị - 5 ta lấy bù từng bit của giá trị + 5 ở dạng nhị phân, sau đó cộng thêm với 1:

$$5 \rightarrow 0101$$

$$1010 + 1 = 1011 \rightarrow (-5)$$

Bảng sau đây minh họa sự biểu diễn các số nguyên 4 bit theo số bù 2.

Bảng 2.3: Biểu diễn số theo số bù hai

Biểu diễn theo số bù 2	Giá trị biểu diễn
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	+0
1000	- 8
1001	- 7
1010	- 6
1011	- 5
1100	- 4
1101	- 3
1110	- 2
1111	- 1

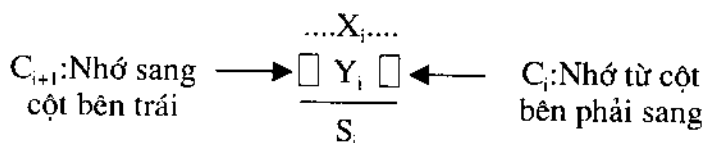
Qua bảng trên chúng ta thấy các số âm cũng có bit tận cùng bên trái là bit dấu luôn bằng 1, còn các số dương thì bit này luôn là 0. Một điểm khác ở đây là khi biểu diễn theo cách này không phải biểu diễn - 0. Vì vậy, phạm vi biểu diễn rộng hơn, đó là - 2^{n-1} đến $2^{n-1}-1$.

Qua các cách biểu diễn trên chúng ta thấy cách biểu diễn theo dấu và độ lớn là đơn giản nhất, sau đó đến biểu diễn theo số bù 1 và biểu diễn theo số bù 2 là phức tạp nhất. Nhưng chính khi biểu diễn theo số bù 2 sẽ làm đơn giản nhất sự thực hiện các phép tính sau này.

1.2. Các phép toán với số dấu phẩy tĩnh

1.2.1. Phép cộng

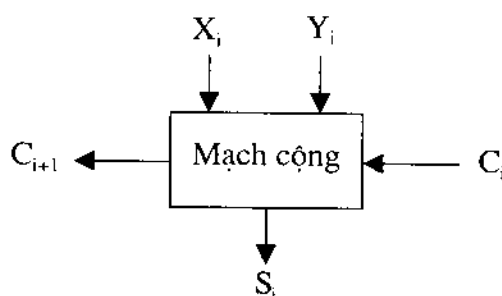
Chúng ta xét phép cộng ở đây là phép cộng 2 số dương, còn phép cộng với số âm có thể quy về phép trừ. Để cộng các số với nhau chúng ta cũng dựa trên nguyên tắc cộng từng cặp bit của các số hạng từ phải qua trái. Tuy nhiên, khi cộng 1 cặp bit nào đó có thể phải cộng thêm bit nhớ từ phép cộng của cặp bit bên phải nó (nếu có) hoặc có khi phải nhớ 1 sang cặp bit bên trái của nó (khi phép cộng bit hiện hành có nhớ).



Ví dụ:

$$\begin{array}{r}
 1\ 0\ 0\ 1 \\
 +\ 0\ 1\ 0\ 1 \\
 \hline
 1\ 1\ 1\ 0
 \end{array}
 \qquad
 \begin{array}{r}
 1\ 1\ 0\ 1 \\
 +\ 0\ 1\ 1\ 1 \\
 \hline
 1\ 0\ 1\ 0\ 0
 \end{array}$$

Phép cộng các cặp bit trên được thực hiện nhờ mạch cộng như sau:



Hình 2.1: Sơ đồ khối mạch cộng 1 bit

Bảng trạng thái (nguyên lý hoạt động) của mạch cộng như sau:

Bảng 2.4: Bảng trạng thái của mạch cộng 1 bit

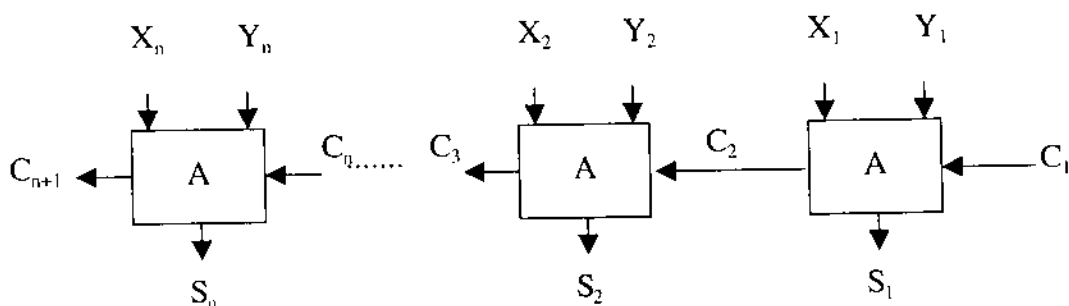
X_i	Y_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Có thể coi mạch cộng gồm 3 đầu vào X_i, Y_i, C_i , đầu ra S_i, C_{i+1} .

$$S_i = \overline{X_i} \overline{Y_i} C_i + \overline{X_i} Y_i \overline{C_i} + \overline{X_i} Y_i C_i + X_i Y_i \overline{C_i}$$

$$C_{i+1} = Y_i C_i + X_i C_i + X_i Y_i$$

Để có thể cộng được hai số n bit thì phải sử dụng n bộ cộng 1 bit như hình vẽ sau:

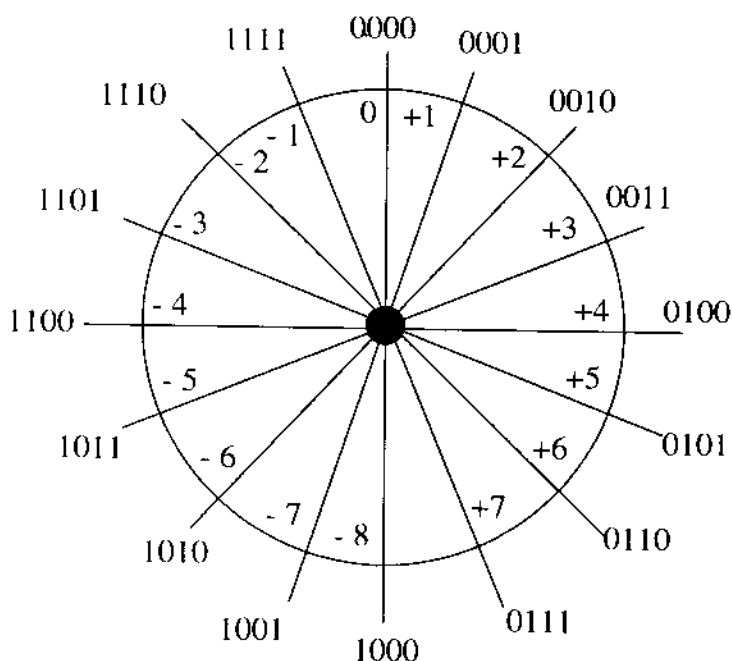


Hình 2.2: Sơ đồ khối mạch cộng n bit

1.2.2. Phép trừ

Thực chất phép trừ là một trường hợp đặc biệt của phép cộng. Ta nhận thấy $A - B = A + (-B)$. Vấn đề đặt ra là chúng ta phải tìm cách cộng được các số âm. Trước hết chúng ta hãy nhận biết một điều đặc biệt khi biểu diễn các số theo dạng số bù 2 trên một đường tròn đánh số thứ tự liên tục. Đó là các số biểu

diễn các giá trị đối nhau sẽ cách đều điểm biểu diễn giá trị 0 ở 2 phía. Hình sau đây sẽ mô tả điều này đối với các số 4 bit.



Hình 2.3: Biểu diễn các số nhị phân 4 bit theo số bù hai

Chính vì điều đặc biệt này mà việc thực hiện phép trừ sẽ dễ dàng được thực hiện bằng phép cộng khi các số được biểu diễn ở dạng số bù 2. Như vậy, để tính hiệu $A - B$ chỉ cần thực hiện phép cộng $A + C$ (trong đó C là số bù 2 của $-B$).

Xét ví dụ cụ thể

Tính: $7 - 5 = ?$ (Sử dụng 4 bit nhị phân)

Trước hết biểu diễn -5 sang dạng số bù 2 sẽ là $1010 + 1 = 1011$

Như vậy, kết quả $7 - 5$ là tổng: $0111 + 1011 = 10010$, nhưng chúng ta đang xét các số 4 bit do đó kết quả thực là 0010, hay bằng 2 hệ thập phân.

1.2.3. Phép nhân

Một điều nhận thấy rằng, tích 2 số n bit có độ dài tối đa là $2n$ bit. Vậy, khi nhân 2 số n bit giữ trong 2 thanh ghi chúng ta phải lưu trữ kết quả vào thanh ghi $2n$ bit. Để nghiên cứu phép toán nhân trong máy tính, ta xét ví dụ sau:

Xét phép nhân 2 số 4 bit: $M \times Q = ?$ trong đó $M = 1101$, $Q = 1011$. Trước hết chúng ta hãy thực hiện phép nhân này theo cách thông thường.

[illegible]

Qua cách thực hiện trên chúng ta thấy, để có được tích giữa M và Q ta chỉ thực hiện phép tính cộng và các phép dịch xen kẽ nhau. Cụ thể ở ví dụ trên chúng ta thực hiện các bước sau:

Bước 0: Khởi tạo tổng $A = 0$ (A để giữ tổng), $M = 1101$, $Q = 1011$ ($q_3 q_2 q_1 q_0$).

Bước 1: Xét giá trị $q_0 = 1$, do đó cộng thêm vào tổng A giá trị M. Sau đó dịch A sang phải.

Bước 2: Xét giá trị $q_1 = 1$, do đó cộng thêm vào tổng A giá trị M. Sau đó dịch A sang phải.

Bước 3: Xét giá trị $q_2 = 0$, do đó không cộng thêm vào tổng A giá trị M. Sau đó dịch A sang phải.

Bước 4: Xét giá trị $q_3 = 1$, do đó cộng thêm vào tổng A giá trị M.

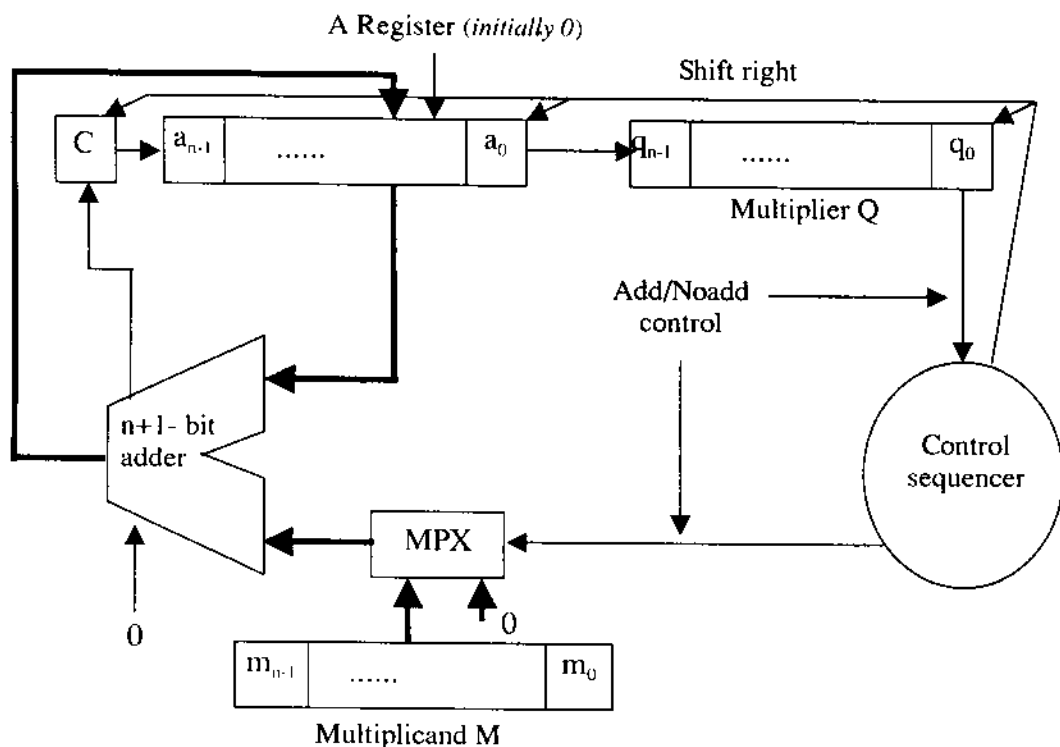
Quá trình thực hiện trên có thể tổng quát hoá khi M và Q gồm n bit, thì chúng ta sẽ phải thực hiện n chu trình như ở trên. Trong đó chu trình thứ i bất kỳ nào đó thực hiện hai nhiệm vụ sau:

- Kiểm tra bit thứ i (q_i) của Q : Nếu bit này bằng 1 thì cộng thêm M vào tổng A (trước khi thực hiện bước 1 tổng A được gán bằng 0). Nếu bit này bằng 0 thì không cộng thêm vào tổng A .

- Dịch tổng A sang phải 1 vị trí.

Giả thiết các giá trị M , Q được lưu trữ trong các thanh ghi n bit. Về nguyên tắc, nếu tổng A được lưu trữ trong một thanh ghi nào đó thì thanh ghi này phải có độ dài $2n$ bit (vì đây là tích hai số n bit). Nhưng trong thực tế thanh ghi lưu trữ Q sau mỗi chu trình đều dịch sang phải 1 vị trí, vì vậy trong các chu trình chúng ta chỉ cần kiểm tra bit q_0 vì các bit khác đều lần lượt qua đây. Một điều hữu ích nữa là, khi dịch thanh ghi Q sẽ lần lượt xuất hiện các bit trống ở đầu bên trái thanh ghi này và chúng có thể được tận dụng để lưu trữ các bit của tích. Chính vì vậy, thanh

ghi lưu trữ tổng A cũng thường chỉ có n bit. Để hiểu rõ hơn việc thực hiện phép nhân 2 số n bit M, Q chúng ta hãy quan sát lược đồ sau đây:



Hình 2.4: Lược đồ thực hiện phép toán nhân

Ghi chú:

MPX là bộ điều khiển nạp bit song song.

Multiplicand M: Số bị nhân M.

Mutiplier Q: Số nhân Q.

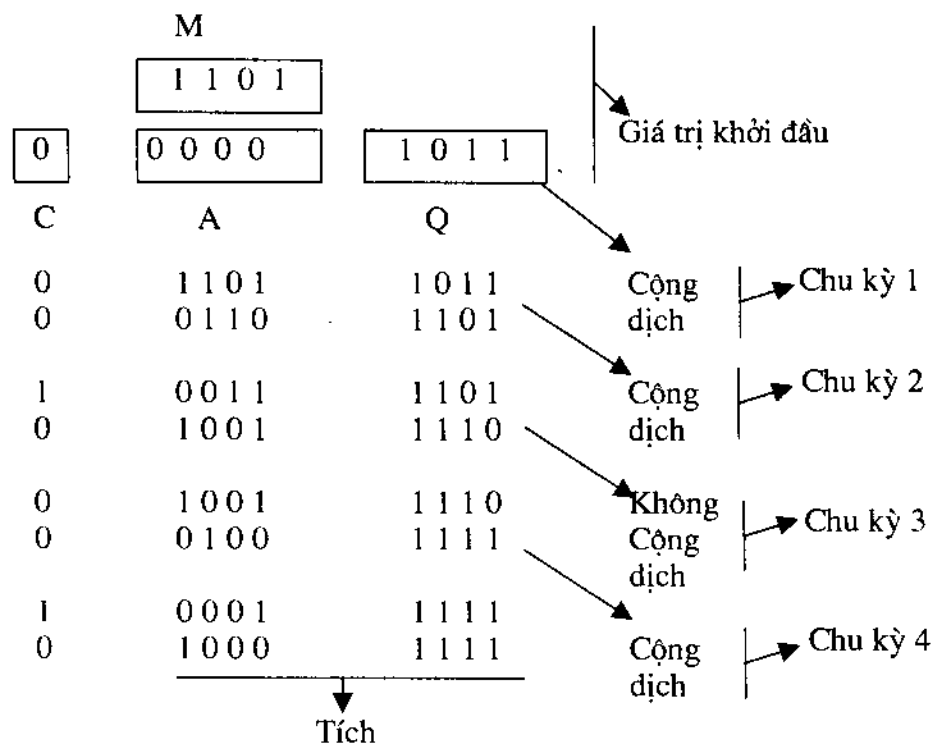
A Register (initally 0): Thanh ghi A (khởi đầu bằng 0).

n -bit adder: Bộ cộng n bit.

Control sequencer: Điều khiển nối tiếp (tuần tự).

Add/Noadd control: Điều khiển cộng/không cộng.

Áp dụng lược đồ tính toán này để tính tích 1101×1011 trong ví dụ trên.



Hình 2.5: Các bước thực hiện phép toán nhân

1.2.4. Phép chia

Để nghiên cứu phép toán chia trong máy tính, ta xét ví dụ sau:

Xét phép chia 2 số, Q : M

Trong đó:

Q = 100010010

M = 1101

Chúng ta hãy thực hiện theo cách thông thường:

$$\begin{array}{r}
 100010010 \text{ (Q)} \quad | \quad 1101 \text{ (M)} \\
 - 1101 \\
 \hline
 100000 \\
 - 1101 \\
 \hline
 1110 \\
 - 1101 \\
 \hline
 1
 \end{array}$$

Xét chi tiết thì các bước đã thực hiện để tìm ra phương pháp chia ở trên như sau:

Lấy dần từng bit của Q bắt đầu từ bên trái nhất. Sau mỗi lần lấy được 1 bit từ Q hãy so sánh giá trị A được tạo bởi các bit vừa lấy được với M. Nếu A lớn hơn M thì tại thương đặt bit tương ứng là 1 và lúc này A được nhận giá trị mới là hiệu của A và M là $A = A - M$. Nếu A nhỏ hơn M thì tại thương đặt bit tương ứng là 0 và A giữ nguyên giá trị. Giá trị A có thể được thay đổi hoặc giữ nguyên sau mỗi lần như vậy, lại được kết hợp với các bit được lấy ra ở lần kế tiếp. Quá trình trên được lặp lại cho đến khi các bit của Q được lấy ra hết. Giá trị A cuối cùng sẽ là số dư của phép chia.

Thuật toán trên cũng được áp dụng khi M, Q gồm n bit. Thông thường các giá trị này được lưu trữ vào các thanh ghi n bit. Một thanh ghi A có n bit là một bộ tổng được sử dụng trong khi thực hiện (lúc đầu khởi gán giá trị 0). Như vậy, việc thực hiện phép chia Q:M sẽ gồm n chu trình, mỗi chu trình gồm các thao tác như sau:

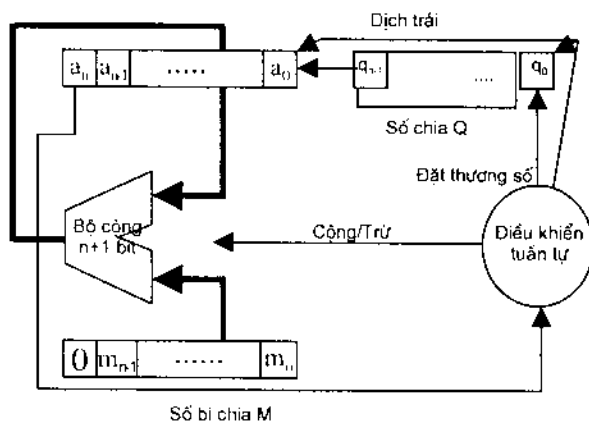
- Dịch A và Q sang trái 1 vị trí.

- Trừ A đi một giá trị M.

- Nếu dấu A là 1 (A sau khi trừ âm) thì đặt thêm bit mới ở thương là 0 và khôi phục lại A bằng cách cộng thêm M vào A. Nếu dấu của A là 0 (A dương sau khi trừ) thì đặt thêm bit mới ở thương là 1 và A nhận giá trị mới.

Lưu ý việc dịch các bit của Q sang trái sẽ lần lượt được đưa vào A và khi dịch như vậy, các bit bên phải của Q sẽ dần được giải phóng và chính đây là nơi để cất các bit của thương. Như vậy, sau khi thực hiện được n chu trình, thì Q sẽ là nơi lưu trữ thương còn A là nơi lưu trữ số dư của phép chia.

Để có thể hiểu kỹ hơn thuật toán trên, hãy quan sát biểu đồ thực hiện ở hình sau:



Hình 2.6: Lưu đồ thực hiện phép toán chia

Một số thực nói chung có thể được biểu diễn dưới dạng gồm 4 thành phần như sau:

$$(-1)^S.M.R^E$$

Trong đó:

S: là phần dấu, để biểu diễn số thực là dương hay âm.

M: Là phần định trị (Mantissa).

E: Là phần mũ (Exponent).

R: Là cơ số (Radix).

Ví dụ:

Trong hệ thập phân số có giá trị 123,456 có thể biểu diễn là:

$$(-1)^0 1,23456.10^2$$

Như vậy $S = 0$, $M = 1,23456$, $R = 10$, $E = 2$.

Do trong máy tính chỉ sử dụng hệ nhị phân, nên thông tin về cơ số R là không cần lưu trữ do R luôn bằng 2. Vậy, để biểu diễn các số thực trong máy tính chúng ta cần 3 thông tin: S, M, E.

Trong đó, S biểu diễn dấu của số thực.

- Nếu $S = 0$ biểu diễn số dương.

- Nếu $S = 1$ biểu diễn số âm.

M thường được chuẩn hoá dưới dạng: 1, X và khi biểu diễn M chỉ cần biểu diễn giá trị X.

Trong thực tế thường có 2 dạng biểu diễn chính theo chuẩn IEEE 754 (Institute of Electrical and Electronic Engineers) sau đây:

2.1.1. Dạng 32 bit (độ chính xác đơn - Single precision)

31	30	23	22	0
S	E'	M		

Với dạng biểu diễn này M gồm 23 bit biểu diễn từ bit 0 đến bit 22.

E' gồm 8 bit biểu diễn từ bit 23 đến bit 30 và một bit dấu S.

Trong đó: $E' = E + 127$

Ở đây E' không có dấu nên có thể biểu diễn được một số nguyên trong phạm vi từ 0 đến 255. Ta có $E = E' - 127$, nên E được biểu diễn trong phạm vi từ - 127 đến 128, đó chính là phạm vi số mũ có thể biểu diễn được. Như vậy, giá trị của số (V) được biểu diễn với mô hình trên đây là:

$$V = (-1)^S \cdot 1, M \cdot 2^{E' - 127}$$

Ví dụ: Biểu diễn số 7.

Biểu diễn sang hệ nhị phân có số 7 là: $1,11 \cdot 2^3$

Ta có $M = 1,11$ (nhưng khi biểu diễn chỉ cần 11).

$S = 0$ do đây là số dương, $E = 2$ nên $E' = 2 + 127 = 129 = (10000001)_2$

Vậy số được biểu diễn là:

0	10000001	11000... 00
---	----------	-------------

Với cách biểu diễn 32 bit này thì phạm vi trị tuyệt đối với các số thực có thể biểu diễn xấp xỉ trong khoảng: từ 2^{-127} đến 2^{128} (10^{38} đến 10^{37}).

2.1.2. Dạng 64 bit (độ chính xác kép - double precision)

63	62	52	51	0
S	E'	M		

Cách biểu diễn số ở đây cũng tương tự dạng biểu diễn trên nhưng số lượng bit lớn hơn. Cụ thể là M gồm 52 bit biểu diễn từ bit 0 đến bit 51, E' gồm 11 bit biểu diễn từ bit 52 đến bit 62 và một bit dấu S.

Trong đó $S' = E + 1023$

Tương tự có giá trị số (V) được biểu diễn với mô hình trên là:

$$V = (-1)^S \cdot 1, M \cdot 2^{E' - 1023}$$

Ví dụ: Biểu diễn số 12,5.

Biểu diễn sang hệ nhị phân có số 12,5 là: $1,1001 \cdot 2^3$

Ta có $M = 1,1001$ (nhưng khi biểu diễn chỉ cần 1001)

$S = 0$ do đây là số dương, $E = 3$ nên $E' = 3 + 1023 = 1026 = 1000\ 0000\ 0010$

Vậy số 12,5 được biểu diễn như sau:

0	10000000010	100100... 00
---	-------------	--------------

Với cách biểu diễn 64 bit này thì phạm vi trị tuyệt đối của các số thực có thể biểu diễn xấp xỉ trong khoảng: từ 10^{-306} đến 10^{306}

2.2. Các phép toán với số dấu phẩy động

Việc thực hiện các phép toán đối với số dấu phẩy động hoàn toàn có thể quy về phép cộng đối với số dấu phẩy tĩnh. Sau đây chúng ta xét chi tiết vấn đề này.

2.2.1. Phép cộng, trừ

Để thực hiện phép cộng, trừ hai số dấu phẩy động hãy thực hiện các bước sau đây:

- Biến đổi số thực có phần bậc nhỏ hơn để nó có phần bậc bằng phần bậc của số kia, bằng cách dịch phải phần định trị.
- Cộng hoặc trừ phần định trị của hai số, còn phần bậc của tổng hoặc hiệu là phần bậc chung.
- Chuẩn hoá kết quả nếu cần thiết. Để phần định trị có dạng 1,X.

Ví dụ:

$$\begin{aligned}\text{Thực hiện phép cộng: } 1,0001.2^7 + 1,1.2^9 \\ = 0,01001.2^9 + 1,1.2^9 = 1,11001.2^9\end{aligned}$$

$$\begin{aligned}\text{Thực hiện phép trừ: } 1,101.2^8 - 1,1.2^7 \\ = 1,101.2^8 - 0,11.2^8 = 0,111.2^8 = 1,11.2^7\end{aligned}$$

2.2.2. Phép nhân

Để thực hiện phép nhân hai số dấu phẩy động hãy thực hiện các bước sau đây:

- Cộng phần bậc hai số và trừ 127.
- Nhân phần định trị hai số và xác định dấu kết quả.
- Chuẩn hoá kết quả nếu cần thiết.

Ví dụ:

$$1,011.2^{130} \times 1,111.2^5 = 10,000101.2^{135-127} = 1,01000101.2^9$$

2.2.3. Phép chia

Để thực hiện phép chia hai số dấu phẩy động hãy thực hiện các bước sau đây:

- Trừ phần bậc hai số và cộng thêm 127.
- Chia phần định trị hai số và xác định kết quả.
- Chuẩn hoá kết quả nếu cần thiết.

Ví dụ:

$$1,1001.2^8 : 1,01.2^3 = 1,01.2^{5+127} = 1,01.2^{123}$$

2.3. Vấn đề tràn số với số dấu phẩy động

Khi cộng hai số khác dấu ở dạng số bù 2, thì cho ta kết quả luôn đúng. Nhưng kết quả sai nếu chúng ta cộng hai số cùng dấu mà trị tuyệt đối của kết quả vượt ra ngoài phạm vi có thể biểu diễn được. Nếu sử dụng hệ n bit thì phạm vi các số dương có thể biểu diễn ở dạng số bù 2 là 0 đến $+(2^{n-1}-1)$. Còn phạm

vì các số âm có thể biểu diễn là -1 đến 2^{n-1} . Để hiểu rõ vấn đề này, chúng ta xét các ví dụ sau:

Ví dụ 1:

Tính kết quả: $7 + 5 = ?$ (xét hệ 4 bit).

Chúng ta hãy tính như cách thông thường:

$$7 + 5 = 0111 + 0101 = 1100$$

Nếu ở dạng số bù 2 thì kết quả 1100, tương đương giá trị biểu diễn là:

-4 và đây là kết quả sai.

Ví dụ 2:

Tính kết quả: $-3 - 8 = ?$ (xét hệ 4 bit)

Đổi $-3 - 8$ sang dạng số bù 2 là: 1101, 1000

Kết quả tính được $= 1101 + 1000 = (1)0101 = 0101$, tương đương giá trị biểu diễn là 5. Đây cũng là kết quả sai.

Những hiện tượng sai số ở trên gọi là hiện tượng tràn số. Do đó, khi thực hiện các phép tính để có kết quả chính xác, ngoài việc quan tâm đến hiện tượng nhớ (CARRY), chúng ta phải quan tâm đến hiện tượng tràn số.

Khi thực hiện phép cộng n bit, người ta có thể phát hiện ra hiện tượng tràn số của kết quả bằng một dấu hiệu Ov, trong đó Ov được tính theo công thức sau:

$$Ov = X_{n-1}Y_{n-1}S_{n-1} + X_{n-1}Y_{n-1}S_{n-1}$$

Trong đó: $X_{n-1}...X_0, Y_{n-1}...Y_0$: Là các bit tương ứng các số hạng cần tính tổng.

$S_{n-1}...S_0$: Là các bit tương ứng của tổng.

Khi Ov có giá trị là 1 thì hiện tượng tràn số xảy ra.

Câu hỏi ôn tập

1. Hãy cho biết máy tính làm việc và trao đổi với thiết bị bên ngoài qua các loại tín và dạng tín nào?
2. Có bao nhiêu cách biểu diễn số dấu phẩy tĩnh? Trình bày các cách biểu diễn đó.
3. Hãy cho biết cách máy tính thực hiện các phép toán với số dấu phẩy tĩnh như thế nào?
4. Có bao nhiêu cách biểu diễn số dấu phẩy động? Trình bày các cách biểu diễn số dấu phẩy động.
5. Trình bày các phép toán thực hiện với số dấu phẩy động.

Chương 3

BỘ VI XỬ LÝ 8088 VÀ CÁC BỘ VI XỬ LÝ TIỀN TIẾN

Mục tiêu:

Trình bày cấu trúc, nguyên lý hoạt động của bộ vi xử lý 8088, là một bộ vi xử lý điển hình để trang bị cho học sinh có được một kiến thức cơ bản, từ đó có phương pháp học tập và nghiên cứu các bộ vi xử lý khác.

Giới thiệu các bộ vi xử lý tiên tiến, cấu trúc chung của các bộ vi xử lý tiên tiến để giúp cho học sinh tìm hiểu, nghiên cứu và khai thác các bộ vi xử lý này.

Nội dung chính:

I. Bộ vi xử lý 8088

1. Đặc trưng kỹ thuật
2. Sơ đồ cấu trúc
3. Đơn vị giao tiếp BUS và điều khiển
4. Các thanh ghi và khả năng địa chỉ hoá bộ nhớ

II. Nguyên tắc hoạt động của 8088

1. Tổ chức các chân
2. Chức năng các chân

III. Các bộ vi xử lý tiên tiến

1. Giới thiệu chung
2. Sơ đồ cấu trúc chung
3. Các khối chức năng

I. BỘ VI XỬ LÝ 8088

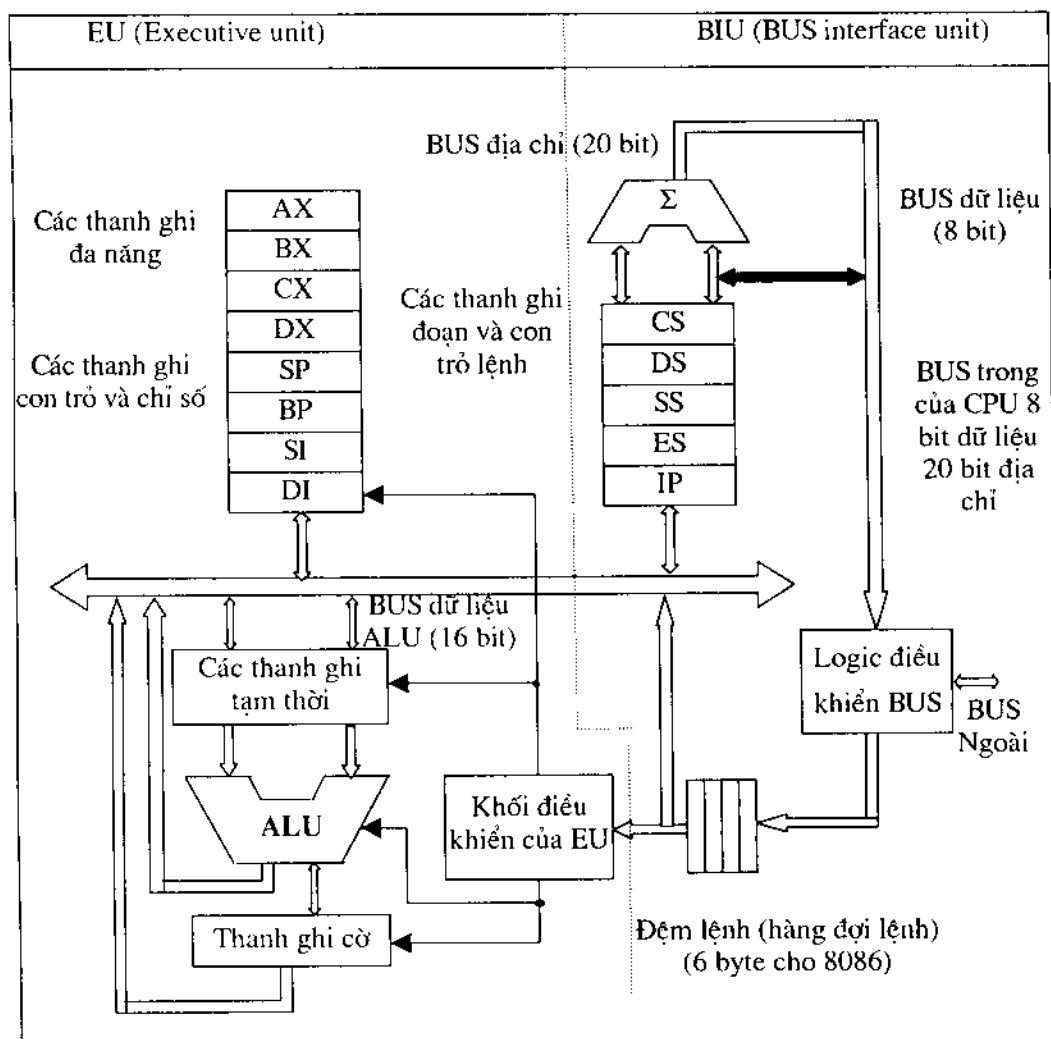
1. Đặc trưng kỹ thuật

Bộ vi xử lý 8088 có các thông số kỹ thuật sau:

- Là bộ vi xử lý gồm 40 chân.
- Trong bộ vi xử lý chứa 29000 transistor.
- Tốc độ xung nhịp $5 \div 8$ MHz.
- Công nghệ chế tạo NMOS.

- Bộ nhớ vật lý 1MB, không có bộ nhớ ảo bên trong.
- Số lệnh là 133 lệnh.
- BUS dữ liệu bên trong có 16 bit.
- BUS dữ liệu bên ngoài có 16 bit.
- BUS địa chỉ là 20 bit.

2. Sơ đồ cấu trúc



EU: Execution Unit - Khối thực hiện lệnh

BIU: BUS interface unit - Khối phối ghép bit

ALU: Arithmetic and logic unit - Khối số học và logic

Hình 3.1: Sơ đồ cấu trúc bộ vi xử lý 8088

3. Đơn vị giao tiếp BUS và điều khiển

Theo sơ đồ khối ta thấy bên trong CPU 8088 có 2 khối chính: Khối phối ghép BUS (Bus Interface Unit - BIU) và khối thực hiện lệnh (Execution Unit - EU). Việc chia CPU ra thành 2 phần làm việc đồng thời có liên hệ với nhau qua đệm lệnh làm tăng đáng kể tốc độ xử lý của CPU. Các BUS bên trong CPU có nhiệm vụ chuyển tải các tín hiệu của các khối khác. Trong số các BUS đó có BUS dữ liệu 16 bit của ALU, BUS tín hiệu điều khiển ở EU và BUS trong của hệ thống ở BIU. Trước khi đi ra BUS ngoài hoặc đi vào BUS trong của bộ vi xử lý, các tín hiệu truyền trên BUS thường được cho đi qua các bộ đệm để nâng cao tính tương thích cho nối ghép hoặc nâng cao khả năng phối ghép.

BIU đưa ra địa chỉ đọc mã lệnh từ bộ nhớ, đọc/ghi dữ liệu từ/vào cổng hoặc bộ nhớ. Nói cách khác, BIU chịu trách nhiệm đưa địa chỉ ra BUS và trao đổi dữ liệu với BUS.

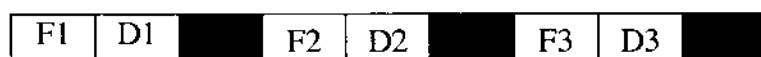
Trong EU ta thấy có một khối điều khiển (Control Unit, CU). Chính tại bên trong khối điều khiển này có mạch giải mã lệnh. Mã lệnh đọc vào từ bộ nhớ được đưa đến đầu vào của bộ giải mã, các thông tin thu được từ đầu ra của nó sẽ được đưa đến mạch tạo xung điều khiển, kết quả là ta thu được các dãy xung khác nhau (tuỳ theo mã lệnh) để điều khiển hoạt động của các bộ phận bên trong và bên ngoài CPU. Trong khối EU còn có khối số học và logic (Arithmetic and Logic Unit, ALU) dùng để thực hiện các thao tác khác nhau với các toán hạng của lệnh. Tóm lại, khi CPU hoạt động, EU sẽ cung cấp thông tin về địa chỉ cho BIU để khối này đọc lệnh và dữ liệu, còn bản thân nó thì giải mã lệnh và thực hiện lệnh.

Trong BIU còn có một bộ nhớ đệm lệnh với dung lượng 4 byte dùng để chứa các mã lệnh đọc được nằm sẵn để chờ EU xử lý (bộ đệm lệnh này còn được gọi là hàng đợi lệnh). Đây là một cấu trúc mới được cấy vào bộ vi xử lý 8088 để thực hiện xử lý xen kẽ liên tục dòng mã lệnh (Instruction Pipelining) và ứng dụng trong các bộ vi xử lý thế hệ mới. Pipeline là một cơ chế đã được ứng dụng từ những năm 60 trong các máy tính lớn. Sau đây là cơ chế thực hiện lệnh xen kẽ pipeline:

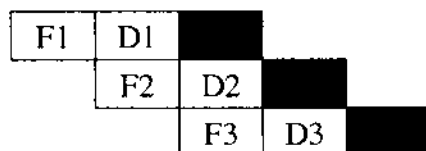
Trong các bộ vi xử lý ở các thế hệ trước (như ở 8085 chẳng hạn), thông thường hoạt động của CPU gồm 3 giai đoạn đọc mã lệnh (Opcode Fetch), giải mã lệnh (Decoder) và thực hiện lệnh (Execution). Trong một thời điểm nhất định, CPU thế hệ này chỉ có thể thực hiện một trong ba công việc nói trên và vì

vậy, tùy theo từng giai đoạn sẽ có những bộ phận nhất định của CPU ở trạng thái nhàn rỗi. Chẳng hạn khi CPU giải mã lệnh hoặc khi nó đang thực hiện những lệnh không liên quan đến BUS (thao tác nội bộ) thì các BUS không được dùng vào việc gì dẫn đến tình trạng lãng phí khả năng của chúng. Trong khi đó từ bộ vi xử lý 8088, Intel sử dụng cơ chế xử lý xen kẽ liên tục dòng mã lệnh thì CPU được chia thành 2 khối và có sự phân chia công việc cho từng khối: Việc đọc mã lệnh là do khối BIU thực hiện, việc giải mã lệnh và thực hiện lệnh là do khối EU đảm nhiệm. Các khối chức năng này có khả năng làm việc đồng thời và các BUS sẽ liên tục được sử dụng, trong khi EU lấy mã lệnh từ bộ đệm 4 byte để giải mã hoặc thực hiện các thao tác nội bộ thì BIU vẫn có thể đọc mã lệnh từ bộ nhớ chính rồi đặt chúng vào bộ nhớ đệm lệnh. Bộ đệm lệnh này làm việc theo kiểu "vào trước - ra trước" (First In - First Out, FIFO), nghĩa là byte nào được cất vào đệm trước sẽ được lấy ra xử lý trước. Nếu có sự vào/ra liên tục của dòng mã lệnh trong bộ đệm này thì có nghĩa là có sự phối hợp hoạt động hiệu quả giữa 2 khối EU và BIU theo cơ chế xử lý xen kẽ liên tục dòng mã lệnh, để làm tăng tốc độ xử lý tổng thể. Kỹ thuật xử lý xen kẽ liên tục dòng mã lệnh sẽ không còn tác dụng tăng tốc độ xử lý chung của CPU nữa nếu như trong đệm lệnh có chứa các mã lệnh của các lệnh Call (gọi chương trình con) hoặc JMP (nhảy), bởi vì lúc gặp các lệnh này nội dung cũ của bộ đệm sẽ bị xóa và thay thế vào đó là nội dung mới được nạp bởi các mã lệnh mới do lệnh nhảy hoặc gọi quyết định. Việc này tiêu tốn nhiều thời gian hơn so với trường hợp trong đệm chỉ có mã lệnh của các lệnh tuần tự.

Không có pipelining



Có pipelining



4. Các thanh ghi và khả năng địa chỉ hoá bộ nhớ

4.1. Các thanh ghi đoạn và khả năng địa chỉ bộ nhớ của 8088

Khối BIU đưa ra trên BUS địa chỉ 20 bit địa chỉ, như vậy 8088 có khả năng phân biệt ra được $2^{20} = 1.048.576 = 1\text{M}$ ô nhớ hay 1 Mbyte, vì các bộ nhớ nói chung tổ chức theo byte. Nói cách khác, không gian địa chỉ của 8088 là 1 Mbyte. Trong không gian 1 Byte này bộ nhớ cần được chia thành các vùng

khác nhau (điều này rất có lợi khi làm việc ở chế độ nhiều người sử dụng hoặc đa nhiệm) dành riêng để:

- Chứa mã chương trình.

- Chứa dữ liệu và kết quả trung gian của chương trình và tạo ra một vùng nhớ đặc biệt gọi là ngăn xếp (stack) dùng vào việc quản lý các thông số của bộ vi xử lý khi gọi chương trình con.

Trong thực tế bộ vi xử lý 8088 có các thanh ghi 16 bit liên quan đến địa chỉ đầu của các vùng (các đoạn) kể trên và chúng được gọi là các thanh ghi đoạn (Segment Registers). Đó là thanh ghi đoạn mã CS (Code Segment), thanh ghi đoạn dữ liệu DS (Data Segment), thanh ghi đoạn ngăn xếp SS (Stack Segment) và thanh ghi đoạn dữ liệu phụ ES (Extra Segment). Các thanh ghi đoạn 16 bit này chỉ ra địa chỉ đầu của 4 đoạn trong bộ nhớ, dung lượng lớn nhất của mỗi đoạn nhớ này là 64 Kbyte và tại một thời điểm nhất định, bộ vi xử lý chỉ làm việc được với 4 đoạn nhớ 64 Kbyte này. Việc thay đổi giá trị của các thanh ghi đoạn làm cho các đoạn tương ứng có thể dịch chuyển linh hoạt trong phạm vi không gian 1 Mbyte. Vì vậy, các đoạn này có thể nằm cách nhau khi thông tin cần lưu trong chúng đòi hỏi dung lượng đủ 64 Kbyte hoặc cũng có thể nằm trùm nhau do có những đoạn không cần dùng hết độ dài 64 Kbyte và vì thế những đoạn khác có thể bắt đầu nối tiếp ngay sau đó. Điều này cũng cho phép ta truy nhập vào bất kỳ đoạn nhớ 64 Kbyte nào nằm trong toàn bộ không gian 1 Mbyte.

Nội dung các thanh ghi đoạn sẽ xác định địa chỉ của ô nhớ nằm ở đâu đoạn, địa chỉ này còn gọi là địa chỉ cơ sở. Địa chỉ của các ô nhớ khác nằm trong đoạn tính được bằng cách cộng thêm vào địa chỉ cơ sở một giá trị gọi là địa chỉ lệch hay độ lệch (Offset), gọi như thế vì nó ứng với khoảng lệch của tọa độ một ô nhớ cụ thể nào đó so với ô đầu đoạn. Độ lệch này được xác định bởi các thanh ghi 16 bit khác đóng vai trò thanh ghi lệch (Offset Register). Cụ thể, để xác định địa chỉ vật lý 20 bit của một ô nhớ nào đó trong một đoạn bất kỳ, CPU 8088 phải dùng đến 2 thanh ghi 16 bit (một thanh để chứa địa chỉ cơ sở, còn thanh kia chứa độ lệch) và từ nội dung của cặp thanh ghi đó nó tạo ra địa chỉ vật lý theo công thức sau:

Địa chỉ vật lý = thanh ghi đoạn x 16 + thanh ghi lệch

Việc dùng 2 thanh ghi để ghi nhớ thông tin về địa chỉ, thực chất tạo ra một loại địa chỉ gọi là địa chỉ logic và được ký hiệu như sau:

“Thanh ghi đoạn : thanh ghi lệch” hay “segment : offset”

Địa chỉ kiểu **segment : offset** là logic vì nó tồn tại dưới dạng giá trị của các thanh ghi cụ thể bên trong CPU và khi cần thiết truy nhập ô nhớ nào đó thì nó phải được đổi ra địa chỉ vật lý để rồi được đưa lên BUS địa chỉ. Việc chuyển đổi này do một bộ tạo địa chỉ thực hiện (phần tử Σ).

Ví dụ, cặp thanh ghi CS:IP sẽ chỉ ra địa chỉ của lệnh sắp thực hiện trong đoạn mã. Nếu tại một thời điểm nào đó ta có CS = F000H và IP = FFF0H thì:

$$\text{CS:IP} \sim \text{F000H} \times 16 + \text{FFF0H} = \text{F0000H} + \text{FFF0H} = \text{FFFF0H}$$

Địa chỉ FFFF0H chính là địa chỉ khởi động của 8088. Dấu ~ ở đây là để chỉ sự tương ứng. Địa chỉ các ô nhớ thuộc các đoạn khác cũng có thể tính được theo cách tương tự như vậy. Từ nay khi cần nói đến địa chỉ của một ô nhớ, ta có thể sử dụng cả địa chỉ logic lẫn địa chỉ vật lý, vì bao giờ cũng tồn tại sự tương ứng giữa 2 loại địa chỉ này (thông qua bộ tạo địa chỉ Σ).

Trước khi nói đến các thanh ghi khác ta nói thêm chút ít về tính đa trị của các thanh ghi đoạn và thanh ghi lệch trong địa chỉ logic ứng với một địa chỉ vật lý. Điều này cũng nói lên tính linh hoạt của cơ chế **segment : offset** trong việc định địa chỉ của 8088. Nhìn vào giá trị cuối cùng của địa chỉ vật lý ta thấy có thể tạo ra địa chỉ đó từ nhiều giá trị khác nhau của thanh ghi đoạn và thanh ghi lệch.

Ví dụ, địa chỉ vật lý 12345H có thể được tạo ra từ các giá trị:

Thanh ghi đoạn	Thanh ghi lệch
1000H	2345H
1200H	0345H
1004H	2305H
0300H	E345H
....	

4.2. Các thanh ghi đa năng

Trong khối EU có 4 thanh ghi đa năng 16 bit: AX, BX, CX, DX. Điều đặc biệt là khi cần chứa các dữ liệu 8 bit thì mỗi thanh ghi này có thể tách ra thành 2 thanh ghi 8 bit cao và thấp để làm việc độc lập, đó là các cặp thanh ghi AH và AL, BH và BL, CH và CL, DH và DL (trong đó H chỉ phần cao, L chỉ phần thấp). Mỗi thanh ghi có thể được dùng một cách vạn năng để chứa các loại dữ liệu khác nhau, nhưng cũng có những công việc đặc biệt nhất định chỉ thao tác với một vài thanh ghi nào đó và chính vì vậy, các thanh ghi thường được gán cho những cái tên đặc biệt rất có ý nghĩa, cụ thể như sau:

- AX (Accumulator, Acc): Thanh chứa. Các kết quả của các thao tác thường được chứa ở đây (kết quả của phép nhân chia). Nếu kết quả là 8 bit thì thanh ghi AL được coi là Acc.

- BX (Base): Thanh ghi cơ sở, thường chứa địa chỉ cơ sở của một bảng dùng trong lệnh XLAT.

- CX (Count): Bộ đếm. CX thường được dùng để chứa số lần lặp trong trường hợp các lệnh LOOP (lặp), còn CL thường chứa số lần dịch hoặc quay trong các lệnh dịch hoặc quay thanh ghi.

- DX (Data): Thanh ghi dữ liệu. DX cùng AX tham gia vào các thao tác của phép nhân hoặc chia các số 16 bit. DX còn dùng để chứa địa chỉ của các cổng trong các lệnh vào ra dữ liệu trực tiếp (IN/OUT).

4.3. Các thanh ghi con trỏ và chỉ số

Trong 8088 còn có 3 thanh ghi con trỏ và 2 thanh ghi chỉ số 16 bit. Các thanh ghi này (trừ IP) đều có thể được dùng như các thanh ghi đa năng, nhưng ứng dụng chính của mỗi thanh ghi là chúng được ngầm định như là thanh ghi lệnh cho các đoạn tương ứng, cụ thể như sau:

- IP: Con trỏ lệnh (Instruction Pointer), IP luôn trỏ vào lệnh tiếp theo sẽ được thực hiện nằm trong đoạn mã CS. Địa chỉ đầy đủ của lệnh tiếp theo này ứng với CS:IP và được xác định theo cách tính địa chỉ logic ở trên.

- BP: Con trỏ cơ sở (Base Pointer), BP luôn trỏ vào một dữ liệu nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của một phần tử trong đoạn ngăn xếp ứng với SS:BP và được xác định theo cách tính địa chỉ logic ở trên.

- SP: Con trỏ ngăn xếp (Stack Pointer), SP luôn trỏ vào đỉnh hiện thời của ngăn xếp nằm trong đoạn ngăn xếp SS. Địa chỉ đầy đủ của đỉnh ngăn xếp ứng với SS:SP và được xác định theo cách tính địa chỉ logic ở trên.

- SI: Chỉ số gốc hay nguồn (Source Index). SI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:SI và được xác định theo cách tính địa chỉ logic ở trên.

- DI: Chỉ số đích (Destination Index), DI chỉ vào dữ liệu trong đoạn dữ liệu DS mà địa chỉ cụ thể đầy đủ ứng với DS:DI và được xác định theo cách tính địa chỉ logic đã nói ở trên.

Riêng trong các lệnh thao tác với dữ liệu kiểu chuỗi thì cặp ES:DI luôn ứng với địa chỉ của phần tử thuộc chuỗi đích, còn cặp DS:SI ứng với địa chỉ của phần tử thuộc chuỗi gốc.

4.4. Thanh ghi cờ FR (Flag Register)

Đây là thanh ghi khá đặc biệt trong CPU, mỗi bit của nó được dùng để phản ánh một trạng thái nhất định của kết quả phép toán do ALU thực hiện hoặc một trạng thái hoạt động của EU. Dựa vào các cờ này người lập trình có thể đưa ra các lệnh thích hợp tiếp theo cho bộ vi xử lý (các lệnh nhảy có điều kiện). Thanh ghi cờ gồm 16 bit nhưng người ta chỉ dùng hết 9 bit của nó để làm các bit cờ. Cụ thể các bit của thanh ghi cờ như sau:

x	x	x	x	O	D	I	T	S	Z	x	A	x	P	x	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

* Các cờ cụ thể

- C hoặc CF (Carry Flag): Cờ nhớ. CF = 1 khi có nhớ hoặc mượn từ MSB.
- P hoặc PF (Parity Flag): Cờ parity, PF phản ánh tính chẵn lẻ (parity) của tổng số bit 1 có trong kết quả. Cờ PF = 1 khi tổng số bit 1 trong kết quả là chẵn (even parity, parity chẵn). Ở đây ta tạm dùng từ parity dạng nguyên gốc để tránh sự tách cùm từ “even parity” thành tính chẵn lẻ chẵn hoặc “odd parity” thành tính chẵn lẻ.
- A hoặc AF (Auxiliary Carry Flag): Cờ nhớ phụ, rất có ý nghĩa khi ta làm việc với các số BCD. AF = 1 khi có nhớ hoặc mượn từ một số BCD thấp (4 bit thấp) sang một số BCD cao (4 bit cao).
- Z hoặc ZF (Zero Flag): Cờ rỗng, ZF = 1 khi kết quả bằng 0.
- S hoặc SF (Sign Flag): Cờ dấu, SF = 1 khi kết quả âm.
- O hoặc OF (Over Flow Flag): Cờ tràn. OF = 1 khi kết quả là một số bù hai vượt ra ngoài giới hạn biểu diễn dành cho nó.

Trên đây là 6 bit cờ trạng thái phản ánh các trạng thái khác nhau của kết quả sau một thao tác nào đó, trong đó 5 bit cờ đầu thuộc byte thấp của thanh ghi cờ, chúng được lập hoặc xóa tùy theo các điều kiện cụ thể sau các thao tác của ALU. Ngoài ra, bộ vi xử lý 8088 còn có các cờ điều khiển sau đây, các cờ này được lập hoặc xóa bằng các lệnh riêng:

- T hoặc TF (Trap Flag): Cờ bẫy. TF = 1 thì CPU làm việc ở chế độ chạy từng lệnh, chế độ này dùng khi cần tìm lỗi một chương trình.
- I hoặc IF (Interrupt Enable Flag): Cờ cho phép ngắt IF = 1, thì CPU cho phép các yêu cầu ngắt (che được) được tác động.
- D hoặc DF (Direction Flag): Cờ hướng. DF = 1 khi CPU làm việc với chuỗi ký tự theo thứ tự từ phải sang trái, vì vậy D chính là cờ lùi.

Ý nghĩa của các cờ đã khá rõ ràng. Riêng cờ tràn cần phải làm rõ hơn để ta hiểu được bản chất và cơ chế làm việc của nó. Cờ tràn thường được dùng đến khi ta làm việc với số bù hai có dấu. Để cho việc giải thích được đơn giản, đầu tiên giả thiết ta làm việc với số bù hai dài 8 bit kết quả để ở AL. Gọi C_{67} là cờ nhớ từ bit 6 (b6) lên bit 7 (b7), trong đó b7 là MSB và cũng chính là bit dấu (SF) của AL. Ta có thể chứng minh được rằng, quan hệ giữa cờ OF với các cờ CF và C_{67} tuân theo phương trình sau:

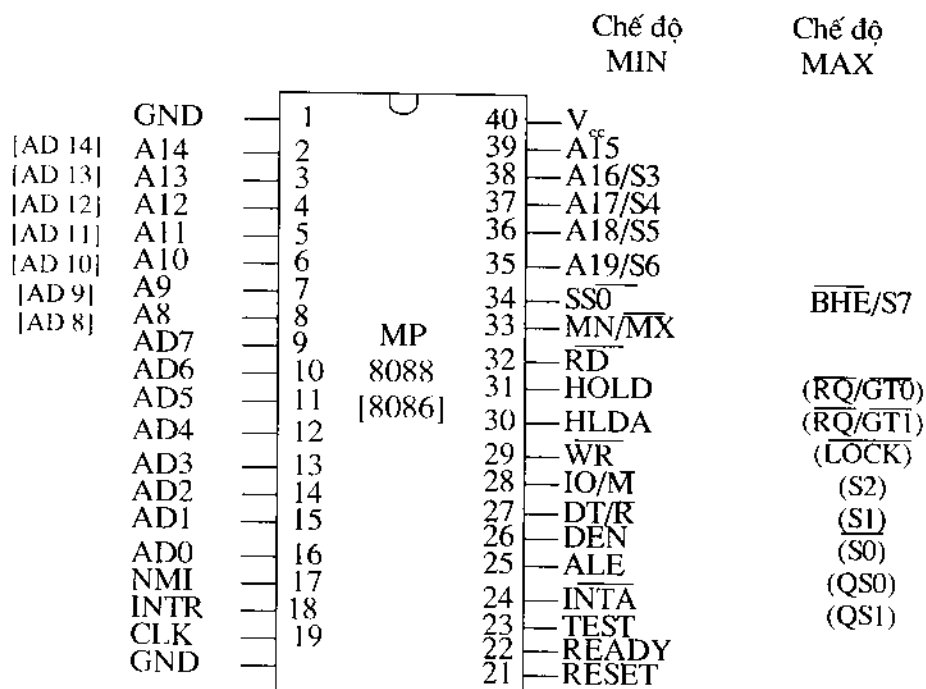
$$OF = CF \oplus C_{67}$$

Nghĩa là khi thực hiện các phép toán với số bù hai có dấu, hiện tượng tràn sẽ xảy ra (cờ OF = 1), nếu có nhớ từ MSB (tức là từ SF) sang CF nhưng lại không có nhớ vào chính nó (SF) hoặc ngược lại. Điều này có thể tổng quát hoá cho các trường hợp làm việc với số bù hai có dấu với độ dài 16/32 bit.

II. NGUYÊN TẮC HOẠT ĐỘNG CỦA 8088

1. Tổ chức các chân

Sơ đồ chức năng các chân của bộ vi xử lý 8088 được thể hiện ở hình sau đây:

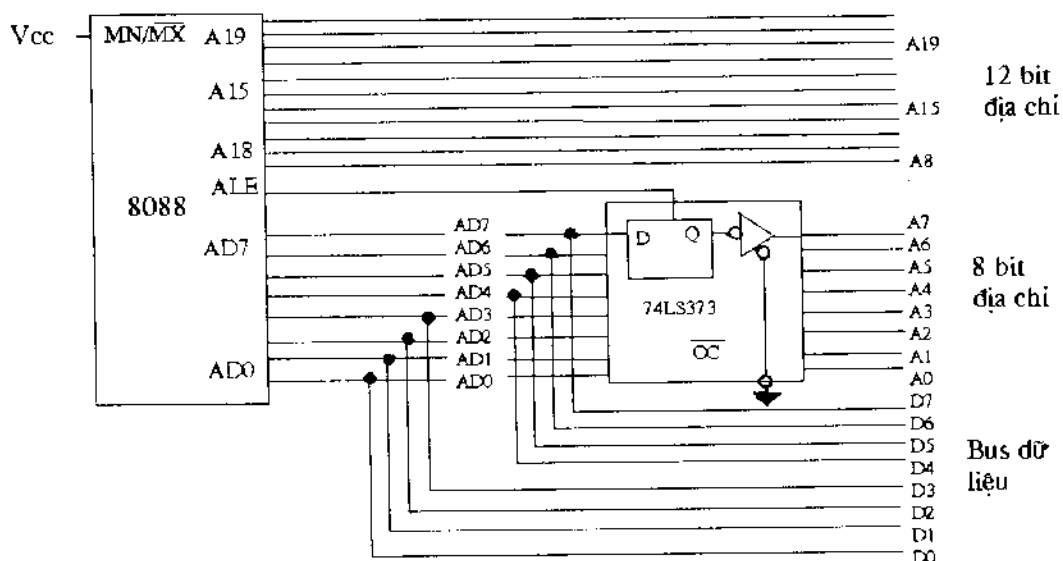


Hình 3.2: Sơ đồ chân của bộ vi xử lý 8088

2. Chức năng các chân

2.1. Các chân địa chỉ (BUS địa chỉ)

Để tách các tín hiệu địa chỉ từ các chân địa chỉ/dữ liệu, cần dùng mạch chốt. Các chân AD_0-AD_7 của bộ vi xử lý 8088 được đưa tới mạch chốt 74LS373 là 8 bit địa chỉ A_0-A_7 , còn 8 bit địa chỉ A_8-A_{15} được lấy trực tiếp từ bộ vi xử lý. 4 bit địa chỉ cuối cùng $A_{16}-A_{19}$ là các chân 35-38. Ở tất cả các máy tính, địa chỉ phải được chốt để đảm bảo cho BUS có khả năng phối ghép và ổn định.



Hình 3.3: Sử dụng ALE tách kênh địa chỉ/dữ liệu

2.2. Các chân của bộ vi xử lý 8088 được dùng cho cả hai chế độ tối thiểu và tối đa

2.2.1. NMI (Nonmaskable Interrupt - Ngắt không che được)

Là tín hiệu vào dùng sườn lên kích hoạt bộ vi xử lý nhảy về vector ngắt sau khi kết thúc lệnh hiện thời. Chân ngắt này không che được bằng phần mềm.

2.2.2. INTR (Interrupt Request - Yêu cầu ngắt)

INTR là tín hiệu vào tích cực cao và được bộ vi xử lý giám sát ở chu kỳ đồng hồ cuối cùng của từng lệnh. Nếu được kích hoạt, nó sẽ kết thúc thực hiện lệnh hiện hành và đáp lại bằng thao tác nhận ngắt. Ở máy tính IBM PC, chân này được nối với vi mạch điều khiển ngắt 8259 còn INTA do 8228 cấp.

2.2.3. CLOCK (Đồng hồ)

Các bộ vi xử lý đều cần một đồng hồ rất chính xác để đồng bộ các hoạt động và điều khiển CPU. Vì vậy Intel thiết kế bộ tạo xung đồng hồ 8284 và được sử dụng với các bộ vi xử lý này. CLOCK là chân vào và được nối với mạch tạo xung đồng hồ 8284. Mạch này hoạt động như nhịp tim của CPU. Bất kỳ một sai sót nào cũng có thể làm CPU hoạt động không đúng.

2.2.4. RESET

Chân RESET được nối với 8284, khi chuyển lên mức cao sẽ buộc vi xử lý ngừng mọi hoạt động và khởi động lại, đồng thời các thanh ghi tự động nhận các giá trị theo bảng sau:

Bảng 3.1: Nội dung của CPU khi thực hiện lệnh RESET

CPU	Nội dung
CS	FFFFH
DS	0000H
SS	0000H
ES	0000H
IP	0000H
Flags	Clear
Hàng đợi	Rỗng

2.2.5. READY

Tín hiệu vào READY có mức tích cực cao, được dùng để chèn trạng thái đợi khi truy cập cổng vào/ra hoặc bộ nhớ tốc độ chậm.

2.2.6. TEST

Trong máy tính IBM PC, tín hiệu TEST là chân vào từ bộ đồng xử lý 8087. Bộ vi xử lý sẽ kiểm tra tín hiệu này khi thực hiện lệnh WAIT. Nếu tín hiệu thấp thì tiếp tục thực hiện chương trình, ngược lại sẽ ngừng thực hiện. Tín hiệu này được sử dụng để đồng bộ hoạt động 8087 và 8088.

2.2.7. Chế độ tối thiểu/tối đa

Các chân 24-31 của 8088 có các chức năng không cố định, mà tùy thuộc vào bộ vi xử lý ở chế độ tối thiểu hay tối đa. Chế độ tối thiểu được chọn khi

chân 33 nối nguồn +5V. Trong chế độ tối thiểu, các chân 24-31 được dùng làm tín hiệu điều khiển IO và bộ nhớ. Các tín hiệu này được 8088 tạo ra ở bên trong bộ vi xử lý. Bảng sau liệt kê các tín hiệu điều khiển ở chế độ tối thiểu.

Bảng 3.2: Các tín hiệu điều khiển 8088 ở chế độ tối thiểu

CPU	Nội dung
24	$\overline{\text{INTA}}$
25	ALE
26	$\overline{\text{DEN}}$
27	DT/ $\overline{\text{R}}$
28	MEM/ $\overline{\text{IO}}$
29	$\overline{\text{WRITE}}$
30	HLDA
31	HOLD

Trong chế độ tối đa, một số tín hiệu điều khiển được tạo ra nhờ mạch ngoài (bằng chip điều khiển BUS 8288) và một số trong các chân 24-31 sẽ đảm nhận chức năng mới. Chế độ tối đa này được sử dụng khi bộ vi xử lý nối ghép với bộ đồng xử lý toán học 8087.

2.3. Các chân ở chế độ tối đa

2.3.1. QS_0, QS_1 (Trạng thái hàng đợi, chân 24, 25)

Hai chân tín hiệu này thông báo cho hệ thống về trạng thái hàng đợi lệnh bên trong bộ vi xử lý. Trong máy tính IBM PC, các chân này được nối với bộ đồng xử lý 8087 để đồng bộ hoạt động giữa 8088 và 8087. Bảng sau là trạng thái của hàng đợi:

Bảng 3.3: Tín hiệu trạng thái hàng đợi

QS_0	QS_1	Đặc điểm
0	0	Không thực hiện
0	1	Byte mã lệnh đầu tiên từ hàng đợi
1	0	Hàng đợi rỗng
1	1	Byte tiếp theo từ hàng đợi

2.3.2. $\overline{S_0}$, $\overline{S_1}$ và $\overline{S_2}$ (Tín hiệu trạng thái, các chân 26, 27 và 28)

Ba chân tín hiệu trạng thái $\overline{S_0}$, $\overline{S_1}$ và $\overline{S_2}$ được nối trực tiếp từ 8088 với 8288 tạo ra toàn bộ các tín hiệu điều khiển như bảng sau:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	Tín hiệu	Ý nghĩa
0	0	0	$\overline{INTA}$	Báo bận ngắt
0	0	1	$\overline{IORC}$	Đọc cổng vào/ra
0	1	0	$\overline{IOWC}$	Ghi cổng vào/ra
0	1	1	None	Dừng
1	0	0	$\overline{MRDC}$	Đọc bộ nhớ
1	0	1	$\overline{MRDC}$	Truy nhập mã lệnh
1	1	0	$\overline{MWTC}$	Ghi bộ nhớ
1	1	1	None	Thụ động

Bảng 3.4: Các tín hiệu điều khiển 8288

2.3.3. $\overline{LOCK}$ (Chân 29)

Tín hiệu ra tích cực thấp ở chân $\overline{LOCK}$ để ngăn ngừa bộ vi xử lý hoặc vi mạch khác chiếm quyền điều khiển BUS hệ thống. Việc khoá BUS được kích hoạt nhờ lệnh Assembler với tiền tố "LOCK", khi đó ở đầu ra $\overline{LOCK}$ (chân 29) tín hiệu sẽ duy trì mức thấp. Ở máy IBM PC, chân này cùng các tín hiệu trạng thái được sử dụng để ngăn không cho DMA chiếm BUS ngoài ý muốn.

2.3.4. $\overline{RQ/GT0}$, $\overline{RQ/GT1}$ (Các chân 30, 31)

Đây là các chân hai chiều cho phép bộ vi xử lý chiếm các quyền điều khiển BUS cục bộ. $\overline{RQ/GT0}$ có mức ưu tiên cao hơn so với $\overline{RQ/GT1}$. Ở máy tính IBM PC, $\overline{RQ/GT1}$ được nối với bộ đồng xử lý toán học 8087 để cho phép và $\overline{RQ/GT0}$ được nối cố định lên mức cao để không cho phép 8087 tham nhập BUS.

2.4. Các chân 24-31, 34 ở chế độ tối thiểu

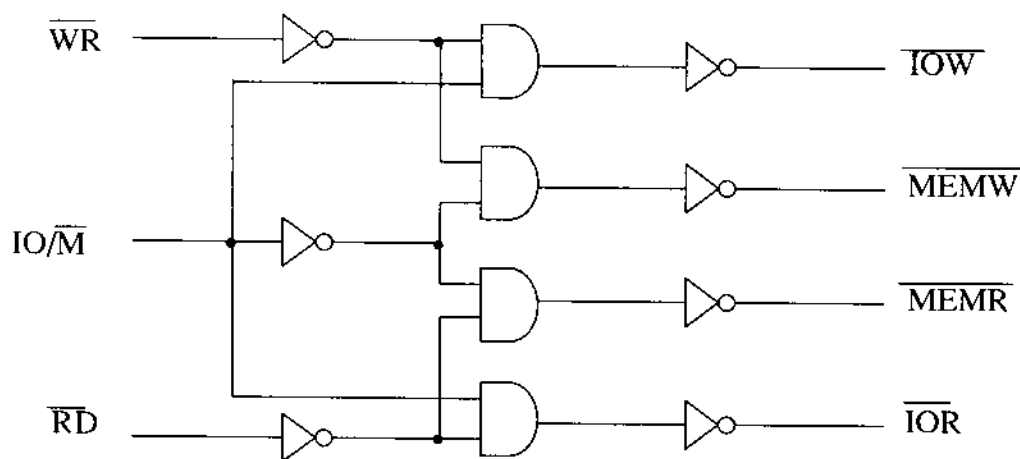
Các chân này chủ yếu cung cấp các tín hiệu điều khiển bộ nhớ và IO. Chế độ tối thiểu được chọn bằng cách nối trực tiếp chân MN/MX lên nguồn +5V. Bảng sau tóm tắt các chức năng của 8088 ở chế độ tối thiểu:

Bảng 3.5: Các chân của bộ vi xử lý 8088 ở chế độ tối thiểu

Chân	Tên	Chức năng
24	INTA (Interrupt Acknowledge)	Tín hiệu ra tích cực thấp. Chân này thông báo cho bộ điều khiển ngắt đang có một ngắt INTR và số hiệu ngắt ở 8 bit thấp của BUS dữ liệu.
25	ALE (Address Latch Enable)	Tín hiệu ra tích cực cao. Chân này xác định địa chỉ hợp lệ đang có trên BUS địa chỉ ngoài.
26	DEN (Data Enable)	Tín hiệu ra tích cực thấp. Tín hiệu này mở mạch 74LS245 nhằm cách ly CPU với BUS hệ thống.
27	DT/R (Data Transmit/Receive)	Tín hiệu ra tích cực thấp để điều khiển chiều của luồng dữ liệu qua mạch thu phát 74LS245.
28	IO/M	Chỉ thị BUS địa chỉ hoặc đang truy nhập bộ nhớ hoặc thiết bị I/O. Tín hiệu ở mức thấp khi truy nhập bộ nhớ, ở mức cao khi truy nhập I/O.
29	WR (Write)	Tín hiệu ra tích cực thấp, được dùng để chỉ thị dữ liệu trên BUS đang được ghi vào bộ nhớ hay cổng I/O. Chân này được dùng với chân 28 để thực hiện thao tác ghi. Chân 32 $\overline{RD}$ trong chế độ tối đa cũng tương tự như chế độ tối thiểu.
30	HLDA (Hold Acknowledge)	Tín hiệu ra tích cực cao. Sau khi có tín hiệu vào chân HOLD, CPU sẽ đáp lại bằng HLDA để thông báo cho bộ điều khiển DMA biết để sử dụng BUS.
31	HOLD (Hold)	Tín hiệu vào tích cực cao từ bộ điều khiển DMA xác nhận có thiết bị ngoài yêu cầu tham nhập vào I/O và bộ nhớ để CPU nhường quyền kiểm soát BUS cục bộ.
34	SSO (Status line)	Là tín hiệu ra được dùng đồng thời với IO/M và DT/R để giải mã trạng thái của chu kỳ BUS hiện tại.

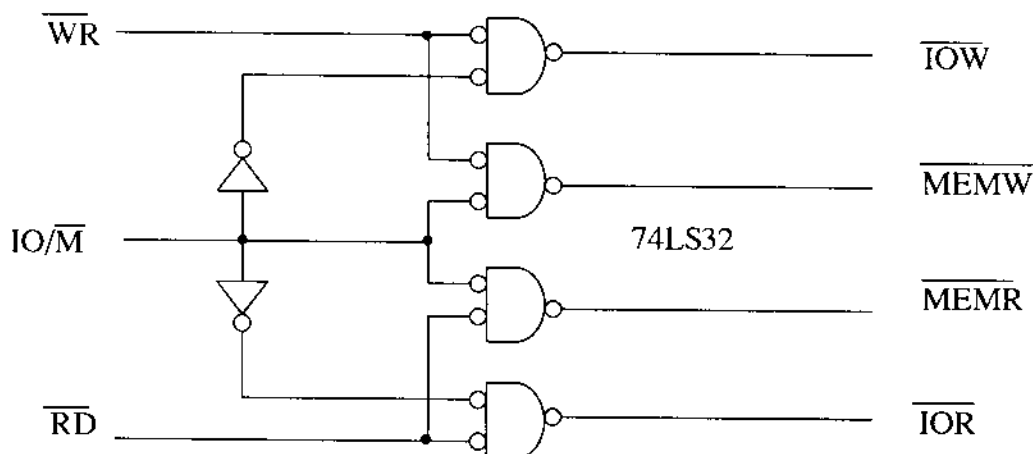
2.5. Tổ chức BUS điều khiển của 8088 ở chế độ tối thiểu

Đối với 8088, một số tín hiệu điều khiển ở chế độ tối thiểu được tạo ra bằng các cổng logic, còn ở chế độ tối đa được cung cấp từ 8288. 8088 cung cấp 3 tín hiệu $\overline{RD}$, $\overline{WR}$ và $IO/\overline{M}$ để tạo ra 4 tín hiệu điều khiển phụ thuộc vào hoạt động của CPU. Tất cả các tín hiệu điều khiển $\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$ và $\overline{MEMW}$ cần có mức tích cực thấp vì chúng đi vào các chân tích cực thấp $\overline{RD}$, $\overline{WR}$ của chip nhớ hoặc ngoại vi. Hình sau mô tả cách tạo ra các tín hiệu này:



Hình 3.4: Tạo tín hiệu điều khiển $\overline{IOW}$, $\overline{MEMW}$, $\overline{MEMR}$ và $\overline{IOR}$

Có thể tạo ra tín hiệu điều khiển bằng mạch sau:



Hình 3.5: Tạo tín hiệu điều khiển bằng cổng logic NAND

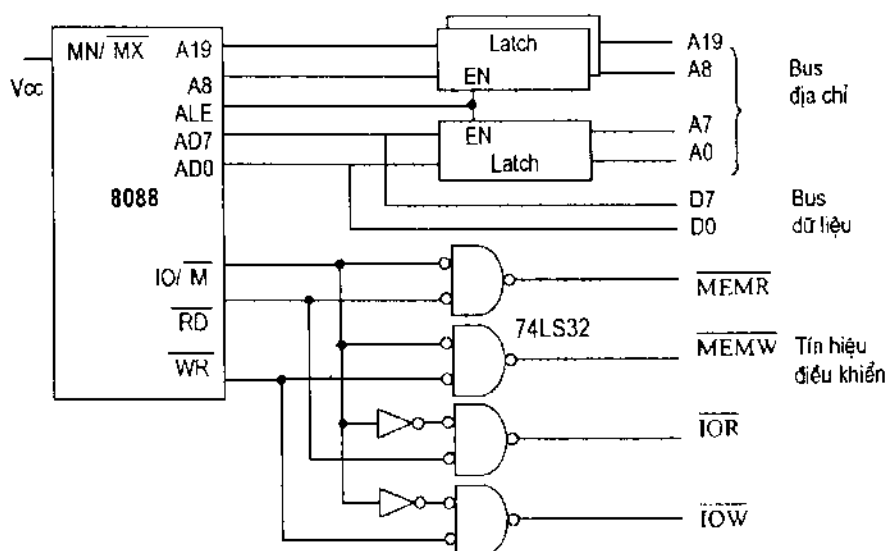
Hoạt động của mạch được mô tả bằng bảng trạng thái sau:

Bảng 3.6: Phát tín hiệu điều khiển BUS

$\overline{RD}$	$\overline{WR}$	$IO/\overline{M}$	Tín hiệu
0	1	0	$\overline{MEMR}$
1	0	0	$\overline{MEMW}$
0	1	1	$\overline{IOR}$
1	0	1	$\overline{IOW}$
0	0	X	Không có

2.6. BUS cơ sở của chế độ tối thiểu 8088

Mỗi máy tính đều có 3 BUS cơ sở để truyền tin: BUS địa chỉ, BUS dữ liệu và BUS điều khiển. 8088 có các chân AD_0 - AD_7 dùng chung cho cả BUS địa chỉ và dữ liệu nên các đường dây này cần phải được phân tách trước khi dùng. Như đã trình bày ở trên, các tín hiệu điều khiển $\overline{IOR}$, $\overline{IOW}$, $\overline{MEMR}$ và $\overline{MEMW}$ phải được tạo ra từ các chân $\overline{RD}$, $\overline{WR}$ và $IO/\overline{M}$ của 8088. Hình sau đây mô tả 8088 ở chế độ tối thiểu cùng các tín hiệu cơ sở (địa chỉ, dữ liệu và điều khiển).



Hình 3.6: BUS cơ sở của chế độ tối thiểu 8088

III. CÁC BỘ VI XỬ LÝ TIỀN TIẾN

1. Giới thiệu chung

Nói đến bộ vi xử lý tiên tiến thì có rất nhiều loại và của nhiều hãng khác nhau. Mỗi một hãng phát triển bộ vi xử lý của mình theo một hướng khác nhau nhưng đều tuân theo một nguyên tắc chung là nâng cấp bộ vi xử lý trước đó, đưa vào bộ vi xử lý các công nghệ mới tiên tiến hơn... để nâng cao hiệu quả hoạt động của bộ vi xử lý. Để khảo sát sự phát triển của các bộ vi xử lý tiên tiến ta hãy điểm qua các bộ vi xử lý của hãng Intel làm ví dụ.

Như các phần trên ta đã khảo sát, bộ vi xử lý 8088 của Intel (tương đương với bộ vi xử lý 8086, chỉ khác là bộ vi xử lý 8088 có 8 bit dữ liệu ngoài còn bộ vi xử lý 8086 có 16 bit dữ liệu ngoài). Bộ vi xử lý tiếp theo mà hãng Intel cho ra đời là bộ vi xử lý 80186. Về cơ bản bộ vi xử lý 80186 cũng giống như bộ vi xử lý 8086 (là bộ vi xử lý 16 bit) nhưng được mở rộng thêm khả năng làm việc bởi vì nó được cấy thêm các bộ phận sau: Bộ tạo xung đồng hồ, bộ điều khiển ngắt ưu tiên, bộ đếm thời gian lập trình được, bộ điều khiển việc thâm nhập trực tiếp vào bộ nhớ, bộ tạo thời gian đợi lập trình được và các mạch giải mã địa chỉ... Chính vì vậy mà bộ vi xử lý 80186 rất thích hợp cho các ứng dụng trong công nghiệp.

Tiếp theo là sự ra đời của bộ vi xử lý 80286, đó là bộ vi xử lý 16 bit, cũng là một nâng cấp của 8086, ra đời vào cùng một thời kỳ với 80186 nhưng được phát triển theo hướng khác. Thay vì các phối ghép ngoại vi được cấy thêm như ở 80186, 80286 có ở bên trong mạch quản lý bộ nhớ (Memory Management Unit, MMU) để làm việc với bộ nhớ ảo, với cơ chế bảo vệ bộ nhớ và khả năng địa chỉ hoá vật lý bộ nhớ có dung lượng đạt tới 16 Mbyte. Bộ vi xử lý 80286 là thành viên đầu tiên trong họ được thiết kế để có thể làm việc trong môi trường đa nhiệm hoặc nhiều người sử dụng.

Bộ vi xử lý 80386 ra đời là bước phát triển tiếp theo của bộ vi xử lý 80286. Đây là bộ vi xử lý 32 bit đầu tiên với BUS địa chỉ 32 bit, có khả năng địa chỉ 4 Gbyte bộ nhớ. Đơn vị số học và logic ALU và các thanh ghi thao tác với dữ liệu đều có độ dài 32 bit, làm cho tốc độ xử lý dữ liệu nhanh hơn. Các mạch quản lý và bảo vệ bộ nhớ của 80386 được cải tiến rất nhiều nên 80386 có thể hoạt động hữu hiệu hơn trong hệ làm việc đa nhiệm.

Bộ vi xử lý 80486 về cơ bản có quy mô như 80386 nhưng nó được tăng cường thêm ngay ở bên trong một bộ nhớ đệm cache (bộ nhớ ẩn) với dung lượng 8 Kbyte dùng chung cho lệnh và dữ liệu. 80486 còn được cấy thêm một bộ đồng xử lý toán học dấu phẩy động (Floating Point Unit, FPU). Khả năng

thao tác của 80486 vì thế cao hơn, còn có khả năng bảo vệ và quản lý bộ nhớ, khả năng phân biệt địa chỉ bộ nhớ thì vẫn giống như 80386.

Bộ vi xử lý Pentium là bộ vi xử lý đầu tiên có bus dữ liệu 64 bit, bus địa chỉ vẫn là 32 bit. So với 80486, Pentium được tăng cường thêm một bộ nhớ ẩn nữa với dung lượng 8 KB, nhờ vậy khi hoạt động nó dành riêng bộ nhớ ẩn 8 KB cho mã lệnh và một bộ nhớ ẩn 8 KB khác cho dữ liệu. Ngoài một bộ đồng xử lý toán học dấu phẩy động FPU có tốc độ cao hơn 10 lần so với bộ vi xử lý 80486, Pentium còn có hai bộ ALU 64 bit dành cho các phép toán với số nguyên (Integer Unit, IU), hai bộ IU này có khả năng làm việc song song. Tất cả các cải tiến này đã nâng cao đáng kể tốc độ hoạt động của hệ có Pentium so với các hệ vi xử lý khác của họ 80x86.

Cho đến ngày nay các bộ vi xử lý của hãng Intel và các hãng khác đã phát triển thêm một bước dài, nhưng về cấu trúc thì vẫn không có sự thay đổi nhiều. Việc nâng cao tốc độ xử lý của các bộ vi xử lý sau này được phát triển theo một hướng khác. Ví dụ như phát triển theo công nghệ micro, nano... các bộ vi xử lý phát triển theo công nghệ này sẽ được trình bày ở giáo trình và tài liệu tham khảo khác.

2. Sơ đồ cấu trúc chung

Các bộ vi xử lý tiên tiến về cơ bản đều có chung một sơ đồ cấu trúc như hình 3.7:

Trong đó:

Bus Interface Unit, BIU: Đơn vị giao diện BUS.

Prefetch Unit and Instruction Queue: Đơn vị nhận lệnh và hàng đợi lệnh.

Decoding Unit: Đơn vị giải mã.

Instruction Cache (Icache): Bộ nhớ đệm (ẩn) lệnh.

Data Cache (Dcache): Bộ nhớ đệm (ẩn) dữ liệu.

Branch Target Cache (BTC): Bộ nhớ đệm rẽ nhánh đích (lệnh), bộ nhớ cache rẽ nhánh lệnh.

Control Unit (CU): Đơn vị điều khiển.

Memory Management Unit (MMU): Đơn vị quản lý bộ nhớ.

Internal Bus: Bus nội.

Special Function Unit (SFU): Đơn vị (xử lý) chức năng đặc biệt.

Interger Register File (IRF): Tập thanh ghi thao tác với số nguyên.

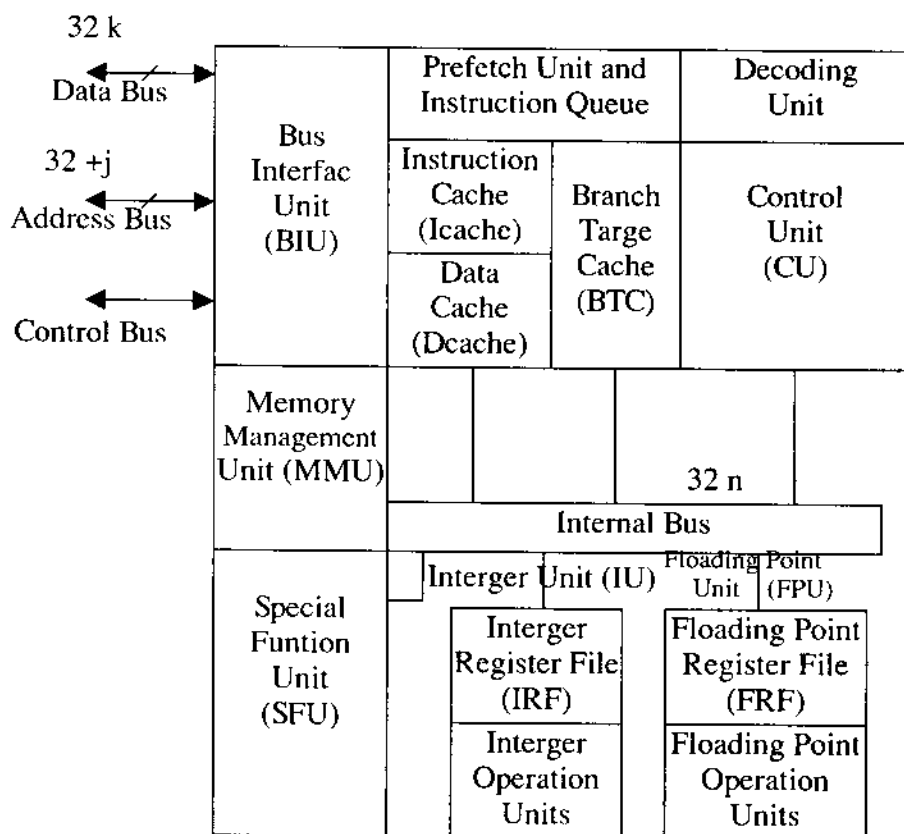
Floating Point Register File (FRF): Tập các thanh ghi thao tác với số dấu phẩy động.

Integer Operation Units: Đơn vị điều khiển thao tác với số nguyên.

Floating Point Operation Units: Đơn vị điều khiển thao tác với số dấu phẩy động.

Integer Unit (IU): Đơn vị (xử lý) số nguyên.

Floating Point Unit (FPU): Đơn vị (xử lý) dấu phẩy động.



Hình 3.7: Sơ đồ khối cấu trúc chung bộ vi xử lý tiên tiến

3. Các khối chức năng

3.1. Bus hệ thống

Gồm Bus dữ liệu, Bus địa chỉ và Bus điều khiển.

Bus dữ liệu có thể có 32 bit cho cả bên trong và bên ngoài chip, trong một vài hệ thống, bus dữ liệu là $32 \times 2 = 64$ bit, $32 \times 3 = 96$ bit hoặc $32 \times 4 = 128$ bit.

Bus địa chỉ trong hầu hết các hệ thống thường là 32 bit hoặc $32 + j$, với j là số nguyên. Với 32 bit địa chỉ thì bộ vi xử lý có khả năng quản lý được một không gian bộ nhớ là 4 Gbyte, tuy nhiên một số bộ vi xử lý được thiết kế đặc biệt để với 32 bit địa chỉ cũng có thể quản lý được một không gian bộ nhớ lớn hơn.

Bus điều khiển của các bộ vi xử lý tiên tiến sau này so với các bộ vi xử lý trước đó không có thay đổi nhiều.

3.2. Đơn vị giao diện Bus

Đơn vị giao diện bus (BIU) là bộ đệm giữa các khối bên trong của bộ vi xử lý và hệ thống bên ngoài. Giao diện Bus được kết nối với Bus hệ thống. Khối BIU được chia làm ba phần chính:

- Giao diện dữ liệu
- Giao diện địa chỉ
- Giao diện điều khiển

Đơn vị giao diện dữ liệu kết nối giữa bus dữ liệu hệ thống với các đơn vị bên trong của bộ vi xử lý. Dữ liệu được truyền giữa qua giao diện dữ liệu với các đơn vị bên trong bộ vi xử lý thông qua bus nội. Thường giao diện dữ liệu được kết nối trực tiếp với đơn vị nhận lệnh và hàng đợi lệnh, với các bộ nhớ đệm.

Giao diện địa chỉ gửi các địa chỉ được phát ra từ phía trong là địa chỉ của các lệnh hoặc địa chỉ của dữ liệu ra bus địa chỉ hệ thống. Địa chỉ được phát ra bởi đơn vị quản lý bộ nhớ MMU, MMU được kết nối trực tiếp với giao diện địa chỉ.

Giao diện điều khiển gửi và nhận các tín hiệu điều khiển và trạng thái từ bộ vi xử lý đến các hệ thống ngoại vi và từ các thiết bị ngoại vi đến bộ vi xử lý. Hầu hết các đường dây điều khiển của giao diện điều khiển thường được kết nối với đơn vị điều khiển CU, tuy nhiên CU cũng kết nối với các đơn vị khác. Các tín hiệu điều khiển được phát ra cho biết trạng thái hoạt động bên trong của bộ vi xử lý như kích hoạt các hoạt động đọc/ghi, chuyển đổi giữa không gian các loại địa chỉ khác nhau, chấp nhận các ngắt, yêu cầu chiếm bus và thực thi các nhiệm vụ khác. Các tín hiệu điều khiển đến cho biết trạng thái các thiết bị ngoại vi, cảnh báo cho bộ vi xử lý biết rằng hệ thống đang ở trạng thái cho phép hay đang ở trạng thái lỗi hệ thống, yêu cầu chiếm bus hoàn toàn, chiếm các bộ đệm bên trong và có thể phục vụ các mục đích khác.

3.3. Đơn vị nhận lệnh

Đơn vị nhận lệnh bao gồm các mạch logic cho phép nhận lệnh từ bộ nhớ đệm lệnh và sắp xếp chúng vào hàng đợi theo nguyên tắc vào trước ra trước

(FIFO). Một hàng đợi lệnh điển hình có thể có từ 8 đến 32 bytes. Sắp xếp các lệnh được chuyển tới đơn vị giải mã. Đơn vị giải mã thực hiện giải mã và lập tức chuyển các tín hiệu điều khiển tới đơn vị điều khiển CU. Ngày nay hầu hết các bộ vi xử lý tiên tiến đều xử lý lệnh theo kiểu song song (ống dẫn), điều đó có nghĩa là tại một thời điểm sẽ có nhiều hơn một lệnh được chuyển tới đơn vị giải mã.

3.4. Đơn vị chức năng đặc biệt SFU

Một vài bộ vi xử lý có thể có một đơn vị chức năng đặc biệt SFU kèm theo một đơn vị số nguyên IU và đơn vị dấu phẩy động FPU. Một SFU có thể là một trong các đơn vị sau đây:

- Đơn vị (xử lý) đồ họa.
- Đơn vị xử lý tín hiệu.
- Đơn vị xử lý ảnh.
- Đơn vị xử lý matrix và vector.

Một bộ vi xử lý có thể có nhiều hơn một đơn vị SFU. Công nghệ hiện nay cho phép tích hợp trên 3 triệu transistor trên một chip đơn, con số này có thể vượt quá 50 triệu vào những năm 2000. Chính vì vậy mà có thể cho phép tích hợp nhiều đơn vị trên cùng một chip. Nhiều đơn vị SFU được tích hợp trên cùng một chip sẽ tăng được hiệu suất xử lý, giảm được thời gian trễ là nhỏ nhất khi ghép nối giữa các đơn vị SFU với các đơn vị khác trong hệ thống và dẫn tới tăng tốc độ hoạt động của toàn hệ thống.

3.5. Bộ nhớ cache (ẩn, đệm)

Bộ nhớ cache là một bộ nhớ truy cập tốc độ cao được đặt giữa CPU và bộ nhớ chính. Với một lượng bộ nhớ cache cho phép có thể cải thiện được tốc độ của toàn bộ hệ thống một cách đáng kể vì nó cho phép CPU truy cập dữ liệu vào bộ nhớ chính nhanh hơn. Ngày nay, các bộ vi xử lý tiên tiến có tới 32 Kbyte hoặc nhiều hơn nữa bộ nhớ cache. Thực tế tất cả các bộ vi xử lý hiện đại đều thực hiện lệnh theo công nghệ pipeline, điều này có nghĩa là với n trạng thái lệnh song song thì bộ vi xử lý có thể xử lý được n trạng thái lệnh khác nhau tại cùng một thời điểm. Vì vậy, trong khi CPU có thể truy cập bộ nhớ để nhận lệnh thì nó có thể truy cập vào bộ nhớ để nhận dữ liệu hoặc là nhận lệnh khác trong cùng một khoảng thời gian, trong trường hợp này nếu có một bộ nhớ cache thì điều trên là không thể thực hiện được. Để khắc phục điều đó, các bộ vi xử lý tiên tiến đều trang bị hai bộ nhớ cache đó là bộ nhớ cache lệnh (Icache) và bộ nhớ cache dữ liệu (Dcache). Hầu hết các bộ vi xử lý đều có hai bộ nhớ cache có

kích thước bằng nhau, tuy nhiên nhiều trường hợp hai bộ nhớ cache này có kích thước khác nhau. Các bộ nhớ cache nhận thông tin từ bộ nhớ chính thông qua bus dữ liệu và đơn vị giao diện bus BIU dưới dạng một hàng gồm nhiều đường dây hoặc dưới dạng các khối, trong các bộ vi xử lý hiện đại kích thước đường dây có thể là 16 hay 32 byte. Bộ nhớ cache lệnh được kết nối trực tiếp với đơn vị nhận lệnh để có thể chuyển một hoặc nhiều hơn một lệnh trong một chu kỳ. Cả hai bộ nhớ cache đều được kết nối với bus nội bộ của bộ vi xử lý. Trong một số bộ vi xử lý, bộ nhớ cache dữ liệu được kết nối với các đơn vị điều khiển thông qua bus dữ liệu điều khiển mở rộng (Operation Data Bus, ODB). Bus ODB thường có độ rộng là 128 hoặc 256 bit để mang một số toán hạng cùng một lúc làm cho tốc độ của bộ vi xử lý được tăng lên đáng kể.

Nhiều bộ vi xử lý được thiết kế để tối ưu hoá hoạt động bằng cách trang bị một bộ nhớ đệm thứ hai gọi là secondary cache. Bộ nhớ cache trên chip được gọi là primary cache (bộ nhớ cache thứ nhất). Bộ nhớ cache thứ hai được đặt giữa bộ nhớ cache thứ nhất và bộ nhớ chính trong sơ đồ phân cấp bộ nhớ và nó nằm ngoài chip, bộ nhớ cache thứ hai có thể có kích thước lớn hơn bộ nhớ cache thứ nhất. Việc có mặt của bộ nhớ cache thứ hai làm cho CPU truy cập vào bộ nhớ chính nhanh hơn và truy cập được nhiều hơn. Trong một vài bộ vi xử lý còn trang bị một bộ nhớ cache thứ hai trên chip, bộ nhớ này thường được kết nối trực tiếp với bộ nhớ cache thứ hai (bên ngoài) để điều khiển và giao tiếp với đơn vị logic.

3.6. Bộ nhớ cache rẽ nhánh đích

Một số bộ vi xử lý tiên tiến thường kèm theo một bộ nhớ cache rẽ nhánh đích (BTC) trên chip. Trong một hệ thống xử lý song song kiểu ống dẫn pipeline, khi một lệnh nhánh được đưa vào thì các lệnh tiếp theo lệnh nhánh phải được lấy ra khỏi ống dẫn và lệnh đích (lệnh nhánh được thực hiện đầu tiên) phải được đưa vào ống dẫn. Nếu các lệnh đích phải nhận từ bộ nhớ thì nó sẽ phải được chèn thêm một khoảng thời gian trễ trong một chuỗi chu kỳ thực hiện lệnh. Tuy nhiên, với việc trang bị thêm một bộ nhớ cache rẽ nhánh đích thì các lệnh được đưa vào ống dẫn nhanh hơn lên, làm tăng tốc độ xử lý của toàn hệ thống. Trong một vài hệ thống, bộ nhớ cache rẽ nhánh đích chỉ chứa một địa chỉ lệnh đích duy nhất thay vì chứa địa chỉ của nhiều lệnh đích.

3.7. Đơn vị điều khiển CU

Đơn vị điều khiển CU có thể là phần cứng hoặc một vi chương trình. Trong các bộ vi xử lý trước đây được thiết kế với một tập lệnh đầy đủ CISC thì CU

thường là một vi chương trình, còn trong các bộ vi xử lý ngày nay được thiết kế với một tập lệnh rút gọn RISC thì CU thường là phần cứng. Điều này có nghĩa là các bộ vi xử lý ngày nay hoạt động với tốc độ nhanh hơn và chỉ với một chu kỳ chúng có thể thực hiện được tất cả hoặc hầu hết các lệnh.

3.8. Đơn vị số nguyên IU

Đơn vị số nguyên IU gồm có nhiều bộ cộng/trừ, nhân/chia được mắc song song với nhau và tập các thanh ghi (IRF) để làm việc với các số nguyên, các thanh ghi có 32 hoặc 64 bit (tùy thuộc vào hệ thống có 32 hay 64 bit). Các bộ vi xử lý với tập lệnh đầy đủ CISC thường có từ 8 đến 16 thanh ghi, còn các bộ vi xử lý với tập lệnh rút gọn RISC thường có ít nhất là 32 thanh ghi, thậm chí lên tới hơn 100 thanh ghi. Thông tin tới đơn vị điều khiển số nguyên đi theo hai hướng: Các lệnh đã được giải mã được đơn vị điều khiển CU hướng tới các đơn vị điều khiển thích hợp, lệnh số nguyên sẽ được gửi tới đơn vị điều khiển số nguyên, lệnh dấu phẩy động được gửi tới đơn vị điều khiển dấu phẩy động, còn các lệnh chức năng đặc biệt được gửi tới đơn vị chức năng đặc biệt. Trong khi đó, dữ liệu được đưa tới đơn vị điều khiển số nguyên từ bộ nhớ Dcache thông qua bus dữ liệu mở rộng ODB.

3.9. Đơn vị dấu phẩy động FPU

Thông tin được đưa đến đơn vị điều khiển số phẩy động cũng tương tự như đưa tới đơn vị điều khiển số nguyên. Đơn vị dấu phẩy động cũng có tập thanh ghi dùng để thao tác với các số dấu phẩy động, gồm có N thanh ghi. Bộ vi xử lý với tập lệnh đầy đủ CISC thường có 8 thanh ghi, còn các bộ vi xử lý với tập lệnh rút gọn thường có 32 thanh ghi. Thường các thanh ghi có độ dài là 32 bit, các thanh ghi này khi dùng để biểu diễn các số dấu phẩy động được gọi là biểu diễn số dấu phẩy động có độ chính xác đơn. Để tăng độ chính xác khi biểu diễn các số dấu phẩy động thì các bộ vi xử lý tiên tiến ngày nay thường được trang bị các thanh ghi với độ dài 64 bit.

3.10. Đơn vị quản lý bộ nhớ MMU

Đơn vị quản lý bộ nhớ có các chức năng sau:

- Dịch địa chỉ ảo (địa chỉ logic) sang địa chỉ vật lý (địa chỉ thực) và chuyển địa chỉ vật lý đến bộ nhớ cache hoặc tới thiết bị ngoại vi thông qua đơn vị giao diện BIU và bus địa chỉ.

- Thực hiện cơ chế phân trang bộ nhớ trong phương thức tổ chức quản lý bộ nhớ ảo.

- Thực hiện cơ chế phân đoạn bộ nhớ (nếu có thể) thông qua đơn vị phân đoạn.
- Thực hiện bảo vệ bộ nhớ, được thực hiện khi phân trang và phân đoạn bộ nhớ.
- Giám sát và quản lý các bộ đệm truy cập nhanh hoặc các bộ đệm cache dịch địa chỉ trong chế độ dịch địa chỉ ảo sang địa chỉ trang vật lý.

Câu hỏi ôn tập

1. Hãy cho biết cấu trúc của bộ vi xử lý 8088 bao gồm những khối và thành phần cơ bản nào? Trình bày tóm tắt chức năng và nhiệm vụ các thành phần đó.
2. Hãy cho biết bộ vi xử lý 8088 gồm có những thanh ghi nào? Các thanh ghi đó dùng để làm gì?
3. Hãy trình bày chức năng các chân của bộ vi xử lý 8088.
4. Cấu trúc chung của bộ vi xử lý tiên tiến gồm có những khối nào? Chức năng các khối đó là gì?
5. Hãy so sánh bộ vi xử lý tiên tiến với bộ vi xử lý 8088.

Chương 4

LẬP TRÌNH HỢP NGỮ VỚI BỘ VI XỬ LÝ 8088

Mục tiêu:

Giới thiệu các kiểu cấu trúc chương trình của một chương trình hợp ngữ, các bước hợp dịch và chạy một chương trình hợp ngữ đối với bộ vi xử lý 8088, từ đó học sinh hình thành được các bước lập trình và chạy chương trình với các bộ vi xử lý khác.

Giới thiệu các chế độ địa chỉ, các lệnh và các ngắt của bộ vi xử lý 8088 nhằm trang bị cho học sinh những kiến thức cơ bản nhất để tạo lập một chương trình hợp ngữ.

Nội dung chính:

I. Giới thiệu chung

1. Cấu trúc lệnh
2. Dữ liệu cho chương trình
3. Khai báo biến và hằng

II. Cấu trúc chương trình

1. Kiểu bộ nhớ
2. Cấu trúc chương trình
3. Các cấu trúc lập trình
4. Các bước tạo và dịch chương trình hợp ngữ

III. Các chế độ địa chỉ của 8088

1. Mã hoá lệnh
2. Các chế độ địa chỉ

IV. Tập lệnh của 8088

1. Các lệnh cơ bản
2. Các ngắt cơ bản

I. GIỚI THIỆU CHUNG

1. Cấu trúc lệnh

Để lập được một chương trình hợp ngữ trước tiên ta phải am hiểu cú pháp của ngôn ngữ đó. Một chương trình hợp ngữ mà ta viết ra nếu đúng về cú pháp sẽ được chương trình dịch hợp ngữ biên dịch ra mã máy, từ chương trình mã máy này ta có thể tạo ra các chương trình chạy (thực hiện) được ngay bằng cách dịch tiếp ra các tệp có đuôi EXE hoặc COM. Do đó, khi viết một chương trình hợp ngữ ta phải tuân thủ những quy tắc cú pháp nhất định để chương trình dịch MASM có thể hiểu và dịch được nó.

Một chương trình hợp ngữ bao gồm các dòng lệnh, một dòng lệnh có thể là một lệnh thật dưới dạng ký hiệu (symbolic), mà đôi khi còn được gọi là dạng gợi nhớ (memonic) của bộ vi xử lý hoặc một hướng dẫn cho chương trình dịch (assembler directive). Lệnh gợi nhớ sẽ được dịch ra mã máy, còn hướng dẫn cho chương trình dịch thì không được dịch, vì nó chỉ có tác dụng chỉ dẫn riêng cho chương trình dịch thực hiện công việc. Ta có thể viết các dòng lệnh này bằng chữ hoa hoặc chữ thường và chúng sẽ được coi là tương đương, vì đối với các dòng lệnh, chương trình không phân biệt kiểu chữ.

Một dòng lệnh của chương trình hợp ngữ có thể có những trường sau (không nhất thiết phải có đủ hết tất cả các trường):

Tên	Mã lệnh	Các toán hạng	Chú giải
-----	---------	---------------	----------

Ví dụ :

TIEP : MOV AH, [BX] [SI] ; nạp vào AH nội dung ô nhớ
; có địa chỉ DS : (BX + SI).

Trong ví dụ trên, tại trường tên có nhãn TIEP, tại trường mã lệnh ta có lệnh MOV, tại trường toán hạng ta có các thanh ghi AH, BX và phần chú giải gồm các dòng:

; nạp vào AH nội dung ô nhớ
; có địa chỉ DS : (BX + SI).

Một ví dụ khác là các dòng lệnh với các hướng dẫn cho chương trình dịch:

MAIN	PROC
------	------

Và

MAIN	ENDP
------	------

Trong ví dụ này: Ở trường tên ta có tên thủ tục là MAIN; ở trường nhãn lệnh ta có các lệnh giả PROC và ENDP, đây là các lệnh giả dùng để bắt đầu và kết thúc một thủ tục có tên là MAIN.

1.1. Trường tên

Trường tên chứa các nhãn tên biến hoặc tên thủ tục. Các tên và nhãn này sẽ được chương trình dịch gán bằng các địa chỉ cụ thể của ô nhớ. Tên và nhãn có thể có độ dài 1..31 ký tự, không được chứa dấu cách và không được bắt đầu bằng số. Các ký tự đặc biệt khác có thể dùng trong tên là ? . @ _ \$ % . Nếu dấu chấm (.) được dùng thì nó phải được đặt ở vị trí đầu tiên của tên. Một nhãn thường kết thúc bằng dấu hai chấm (:).

1.2. Trường mã lệnh

Trong trường mã lệnh nói chung sẽ có các lệnh thật hoặc lệnh giả. Đối với các lệnh thật thì trường này chứa các mã lệnh gọi nhớ. Mã lệnh này sẽ được chương trình dịch dịch ra mã máy. Đối với các hướng dẫn chương trình dịch thì trường này chứa các lệnh giả và sẽ không được dịch ra mã máy.

1.3. Trường toán hạng

Đối với một lệnh thật thì trường này chứa các toán hạng của lệnh. Tùy theo từng loại lệnh mà ta có thể có 0, 1 hoặc 2 toán hạng trong một lệnh. Trong trường hợp các lệnh với 1 toán hạng thông thường ta có toán hạng là đích hoặc nguồn, còn trong trường hợp lệnh với 2 toán hạng thì ta có 1 toán hạng là đích và 1 toán hạng là nguồn.

Đối với hướng dẫn chương trình dịch thì trường này chứa các thông tin khác nhau liên quan đến các lệnh giả của hướng dẫn.

1.4. Trường chú giải

Lời giải thích ở trường chú giải phải được bắt đầu bằng dấu chấm phẩy (;). Trường chú giải này được dành riêng cho người lập trình để ghi các lời giải thích cho các lệnh của chương trình với mục đích giúp cho người đọc chương trình dễ hiểu các thao tác của chương trình. Lời chú giải cũng có lợi ngay cho chính tác giả của nó vì sau một thời gian không xem đến chương trình thì mọi việc lại như mới. Khi đọc thấy dấu chấm phẩy, chương trình dịch bỏ qua không dịch từ phần này trở đi. Chính vì vậy, người ta cũng thường hay dùng dấu này để loại bỏ một dòng lệnh nào đó trong chương trình. Thông thường lời chú giải cần phải mang đủ thông tin để giải thích về thao tác của lệnh trong hoàn cảnh cụ thể và như thế thì mới có ích cho người đọc.

2. Dữ liệu chương trình

Dữ liệu của một chương trình hợp ngữ là rất đa dạng. Các dữ liệu có thể được cho dưới dạng số hệ hai, hệ mười, hệ mười sáu hoặc dưới dạng ký tự (cần chú ý là trên các máy IBM PC trong chương trình Debug, một công cụ tìm lỗi rất thông dụng cho các chương trình hợp ngữ đơn giản, dữ liệu bằng số được ngầm định phải ở hệ mười sáu).

Khi cung cấp số liệu cho chương trình, số cho ở hệ nào phải được kèm đuôi của hệ đó (trừ hệ mười thì không cần vì là trường hợp ngầm định của assembler). Riêng đối với số hệ mười sáu, nếu số đó bắt đầu bằng các chữ (a..f hoặc A..F) thì ta phải thêm 0 ở trước để chương trình dịch có thể hiểu được đó là một số hệ mười sáu chứ không phải là một tên hoặc một nhân.

Ví dụ các số viết đúng:

0011B	;Số hệ hai.
1234	;Số hệ mười.
0ABBAH	;Số hệ mười sáu.
1EF1H	;Số hệ mười sáu

Nếu dữ liệu là ký tự hoặc chuỗi ký tự thì chúng phải được đóng trong cặp dấu trích dẫn đơn hoặc kép, thí dụ như 'A' hay "abcd". Chương trình dịch sẽ dịch ký tự ra mã ASCII tương ứng của nó. Vì vậy, trong khi cung cấp dữ liệu kiểu ký tự cho chương trình ta có thể dùng bản thân ký tự được đóng trong dấu trích dẫn hoặc mã ASCII của nó. Ví dụ, ta có thể sử dụng dữ liệu ký tự là "0" hoặc mã ASCII tương ứng là 30H, ta có thể dùng '\$' hoặc 26 hoặc 34. . .

3. Khai báo biến và hằng

Biến trong chương trình hợp ngữ có vai trò như nó có ở trong ngôn ngữ bậc cao. Một biến phải được định kiểu dữ liệu là kiểu byte hay kiểu từ và sẽ được chương trình dịch gán cho một địa chỉ nhất định trong bộ nhớ. Để định nghĩa các kiểu dữ liệu khác nhau ta thường dùng các lệnh giả sau:

DB (define byte)	: Định nghĩa biến kiểu byte.
DW (define word)	: Định nghĩa biến kiểu từ.
DD (define double word)	: Định nghĩa biến kiểu từ kép.

3.1. Biến kiểu byte

Biến kiểu byte sẽ chiếm 1 byte trong bộ nhớ. Hướng dẫn chương trình dịch để định nghĩa biến kiểu byte có dạng tổng quát như sau:

Tên DB giá_trị_khởi_đầu

Ví dụ:

B1 DB 4

Ví dụ trên, định nghĩa biến byte có tên là B1 và dành 1 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 4.

Nếu trong lệnh trên ta dùng dấu ? thay vào vị trí của số 4 thì biến B1 sẽ được dành chỗ trong bộ nhớ nhưng không được gán giá trị khởi đầu. Cụ thể dòng lệnh giả:

B2 DB ?

Chỉ định nghĩa 1 biến byte có tên là B2 và dành cho nó một byte trong bộ nhớ.

Một trường hợp đặc biệt của biến byte là biến ký tự. Ta có thể có định nghĩa biến ký tự như sau:

C1 DB '\$'

C2 DB 34

3.2. Biến kiểu từ

Biến kiểu từ cũng được định nghĩa theo cách giống như biến byte. Hướng dẫn chương trình dịch để định nghĩa biến từ có dạng như sau:

Tên DW giá_trị_khởi_đầu

Ví dụ:

W1 DW 40

Ví dụ trên, định nghĩa biến từ có tên là W1 và dành cả 2 byte trong bộ nhớ cho nó để chứa giá trị khởi đầu bằng 40.

Chúng ta cũng có thể sử dụng dấu ? chỉ để định nghĩa và dành 2 byte trong bộ nhớ cho biến từ W2 mà không gán giá trị đầu cho nó bằng dòng lệnh sau:

W2 DW ?

3.3. Biến kiểu mảng

Biến kiểu mảng là biến hình thành từ một dãy liên tiếp các phần tử cùng loại byte hoặc từ. Khi định nghĩa biến kiểu mảng ta gán tên cho một dãy liên tiếp các byte hay từ trong bộ nhớ cùng với các giá trị ban đầu tương ứng.

Ví dụ:

M1 DB 4,5,6,7,8,9

Ví dụ trên, định nghĩa biến mảng có tên là M1 gồm 6 byte và dành cho nó trong bộ nhớ từ địa chỉ ứng với M1 để chứa các giá trị khởi đầu bằng

4,5,6,7,8,9. Phần tử đầu trong mảng là 4 và có địa chỉ trùng với địa chỉ của M1, phần tử thứ hai là 5 và có địa chỉ M1+1...

Khi chúng ta muốn khởi đầu các phần tử của mảng với cùng một giá trị chúng ta có thể dùng thêm toán tử DUP trong lệnh.

Ví dụ:

M2 DB 100 DUP (0)

M3 DB 100 DUP (?)

Ví dụ trên, định nghĩa một biến mảng tên là M2 gồm 100 byte, dành sẵn chỗ trong bộ nhớ cho nó để chứa 100 giá trị khởi đầu bằng 0 và một biến mảng khác tên là M3 gồm 100 byte, dành sẵn chỗ cho nó trong bộ nhớ để chứa 100 giá trị nhưng chưa được khởi đầu. Toán tử DUP có thể lồng nhau để định nghĩa ra 1 mảng.

Ví dụ, dòng lệnh:

M4 DB 4, 3, 2, 2 DUP (1, 2 DUP (5) , 6)

Sẽ định nghĩa ra một mảng M4 tương ứng với lệnh sau:

M4 DB 4,3,2,1,5,5,6,1,5,5,6

Một điều cần chú ý nữa là, đối với các bộ vi xử lý của Intel, nếu ta có một từ để trong bộ nhớ thì byte thấp của nó sẽ được để ở ô nhớ có địa chỉ thấp, byte cao sẽ được để ở ô nhớ có địa chỉ cao. Cách lưu giữ số liệu kiểu này được gọi là “quy ước đầu bé”. Còn một số các bộ vi xử lý của Motorola lại có cách cất số liệu theo thứ tự ngược lại hay còn được gọi là “quy ước đầu to”. Ví dụ, sau khi định nghĩa biến từ có tên là WORDA như sau:

WORDA DW OFFEEH

Thì ở trong bộ nhớ byte thấp (EEH) sẽ được để tại địa chỉ WORDA còn byte cao (FFH) sẽ được để tại địa chỉ tiếp theo, tức là tại WORDA+1.

3.4. Biến kiểu xâu ký tự

Biến kiểu xâu ký tự là một trường hợp đặc biệt của biến mảng trong đó các phần tử của mảng là các ký tự. Một xâu ký tự có thể được định nghĩa bằng các ký tự hoặc bằng mã ASCII của các ký tự đó.

Các ví dụ sau đều là các lệnh đúng và đều định nghĩa cùng một xâu ký tự nhưng gán nó cho các tên khác nhau:

STR1 DB 'string'

STR2 DB 73h, 74h, 72h, 69h, 6Eh 67h

STR2 DB 73h, 74h, 'r', 'i', 6Eh, 67h

3.5. Hàng có tên

Các hàng trong chương trình hợp ngữ thường được gán tên để làm cho chương trình dễ đọc hơn. Hàng có thể là kiểu số hay kiểu ký tự. Việc gán tên cho hàng được thực hiện nhờ lệnh giả EQU (equate) như sau:

CR EQU 0Dh ; CR là carriage return

LF EQU 0Ah ; LF là line feed

Trong ví dụ trên, lệnh giả EQU gán giá trị số 13 (mã ASCII của ký tự trở về đầu dòng) cho tên CR và 10 mã ASCII của ký tự thêm dòng mới cho tên LF. Hàng cũng có thể là một chuỗi ký tự. Trong ví dụ dưới đây, sau khi đã gán một chuỗi ký tự cho một tên:

CHAO EQU 'Hello'

Ta có thể sử dụng hàng này để định nghĩa một biến mảng khác:

MSG DB CHAO, '\$'

Vì lệnh giả EQU không dành chỗ của bộ nhớ cho tên của hàng nên ta có thể đặt nó khá tự do tại những chỗ thích hợp bên trong chương trình. Tuy nhiên, trong thực tế người ta thường đặt các định nghĩa này trong đoạn dữ liệu.

II. CẤU TRÚC CHƯƠNG TRÌNH

1. Kiểu bộ nhớ

Một chương trình mã máy trong bộ nhớ thường bao gồm các vùng nhớ khác nhau để chứa mã lệnh, chứa dữ liệu của chương trình và một vùng nhớ khác được dùng làm ngăn xếp phục vụ hoạt động của chương trình. Chương trình viết bằng hợp ngữ cũng phải có cấu trúc tương tự để khi được dịch nó sẽ tạo ra mã tương ứng với chương trình mã máy nói trên. Để tạo ra sườn của một chương trình hợp ngữ, chúng ta sẽ sử dụng cách định nghĩa đơn giản đối với mô hình bộ nhớ dành cho chương trình và đối với các thanh ghi đoạn, cách định nghĩa được phép từ phiên bản 5.0 của Microsoft Macro Assembler.

1.1. Khai báo quy mô sử dụng bộ nhớ

Kích thước của bộ nhớ dành cho đoạn mã và đoạn dữ liệu trong một chương trình được xác định nhờ hướng dẫn chương trình dịch Model như sau (hướng dẫn này phải được đặt trước các hướng dẫn khác trong chương trình hợp ngữ, nhưng sau hướng dẫn về loại CPU):

.MODEL Kiểu_kích_thước_bộ_nhớ

Có nhiều kiểu_kích_thước_bộ_nhớ cho các chương trình với đòi hỏi dung lượng bộ nhớ khác nhau. Thông thường các ứng dụng đòi hỏi mã chương trình dài nhất cũng chỉ cần chứa trong một đoạn (64KB), dữ liệu cho chương trình nhiều nhất nhưng chỉ cần chứa trong một đoạn, thích hợp nhất nên chọn gọn được trong một đoạn thì có thể chọn tiny (hẹp).

.Model Small

hoặc

.Model Tiny

Ngoài Kiểu_kích_thước_bộ_nhớ nhỏ hoặc hẹp nói trên, tùy theo nhu cầu cụ thể MASM còn cho phép sử dụng các kiểu_kích_thước_bộ_nhớ khác nhau như được liệt kê trong bảng sau:

Bảng 4.1: Các kiểu khai báo sử dụng bộ nhớ

Kiểu kích thước	Mô tả
Tiny (hẹp)	Mã lệnh và dữ liệu gói gọn trong một đoạn
Small (nhỏ)	Mã lệnh gói gọn trong một đoạn, dữ liệu nằm trong một đoạn
Medium (trung bình)	Mã lệnh không gói gọn trong một đoạn, dữ liệu nằm trong một đoạn
Compact (gọn)	Mã lệnh gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn
Large(lớn)	Mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, không có mảng nào lớn hơn 64KB
Huge (đồ sộ)	Mã lệnh không gói gọn trong một đoạn, dữ liệu không gói gọn trong một đoạn, các mảng có thể lớn hơn 64KB

1.2. Khai báo đoạn ngăn xếp

Việc khai báo đoạn ngăn xếp là cốt để dành ra một vùng bộ nhớ đủ lớn dùng làm ngăn xếp phục vụ cho hoạt động của chương trình khi có chương trình con. Việc khai báo thực hiện ở hướng dẫn chương trình dịch như sau:

. Stack Kích_thước

Kích_thước sẽ quyết định số byte dành cho ngăn xếp. Nếu ta không khai Kích_thước thì chương trình dịch sẽ tự động gán cho Kích_thước giá trị 1KB, đây là kích thước ngăn xếp quá lớn đối với một ứng dụng thông thường. Trong thực tế, các bài toán của ta thông thường với $100 + 256$ byte là đủ để làm ngăn xếp và ta có thể khai báo kích thước cho nó như sau:

```
.Stack          100
```

hoặc

```
.Stack          100H
```

1.3. Khai báo đoạn dữ liệu

Đoạn dữ liệu chứa toàn bộ các định nghĩa cho các biến của chương trình. Các hằng cũng nên được định nghĩa ở đây để đảm bảo tính hệ thống mặc dù ta có thể để chúng ở trong chương trình.

Việc khai báo dữ liệu được thực hiện nhờ hướng dẫn chương trình dịch .DATA, việc khai báo biến và hằng được thực hiện tiếp ngay sau đó bằng các lệnh giả thích hợp. Điều này được minh họa trong ví dụ sau:

```
.Data
```

```
MSG      DB    'Hello!$'
```

```
CR       DB    13
```

```
LF       EQU   10
```

1.4. Khai báo đoạn mã

Đoạn mã chứa mã lệnh của chương trình. Việc khai báo đoạn mã được thực hiện nhờ hướng dẫn chương trình dịch .Code như sau:

```
.CODE
```

Bên trong đoạn mã, các dòng lệnh phải được tổ chức một cách hợp lý, đúng ngữ pháp dưới dạng một chương trình chính (CTC) và nếu cần thiết thì kèm theo các chương trình con (ctc). Các chương trình con sẽ được gọi ra bằng các lệnh CALL có mặt bên trong chương trình chính.

Một thủ tục được định nghĩa nhờ các lệnh giả PROC và ENDP. Lệnh giả PROC được dùng để bắt đầu một thủ tục, còn lệnh giả ENDP được dùng để kết thúc nó. Như vậy, một chương trình chính có thể được định nghĩa bằng các lệnh giả PROC và ENDP theo mẫu sau:

```
Tên_CTC Proc
```

;các lệnh của thân chương trình chính

CALL Tên_etc ;gọi etc

Tên_CTC Endp

Giống như chương trình chính, một chương trình con cũng được định nghĩa dưới dạng một thủ tục nhờ các lệnh giả PROC và ENDP theo mẫu sau:

Tên_etc Proc

;các lệnh của thân chương trình con

:

RET

Tên_etc Endp

Trong các mẫu chương trình nói trên, ngoài các lệnh giả có tính nghi thức bắt buộc ta cần chú ý đến sự bố trí của lệnh gọi (CALL) trong chương trình chính và lệnh trở về (RET) trong chương trình con.

2. Cấu trúc chương trình

Cấu trúc của một chương trình hợp ngữ là khung hay khuôn dạng của chương trình viết bằng hợp ngữ. Cấu trúc của chương trình hợp ngữ có hai loại đó là cấu trúc mà khi chương trình dịch hợp ngữ dịch ra dưới dạng file chạy được là file có phần mở rộng là .EXE và .COM.

2.1. Cấu trúc chương trình .EXE

Từ các khai báo các đoạn của chương trình đã nói ở trên, ta có thể xây dựng một khung tổng quát cho các chương trình hợp ngữ với kiểu kích thước bộ nhớ nhỏ. Sau đây là một khung cho chương trình hợp ngữ để rồi sau khi được dịch (assembled) và nối (linker) trên máy IBM PC sẽ tạo ra một tệp chương trình chạy được ngay (executable) với đuôi .EXE.

.Model Small

.Stack 100

.Data

;các định nghĩa cho biến của hằng để tại đây.

.Code

MAIN proc

;khởi đầu cho DS

MOV AX,@Data

MOV DS, AX

Các lệnh của chương trình chính để tại đây

;trở về DOS dùng hàm 4CH của INT 21H

MOV AH, 4CH

INT 21H

MAIN Endp

;các chương trình con (nếu có) để tại đây

End MAIN

Trong khung chương trình trên, tại dòng cuối cùng của chương trình ta dùng hướng dẫn chương trình dịch END và tiếp theo là MAIN để kết thúc toàn bộ chương trình. Ta có nhận xét rằng MAIN là tên của chương trình chính, nhưng quan trọng hơn và về thực chất thì đó là nơi bắt đầu các lệnh của chương trình trong đoạn mã.

Khi một chương trình .EXE được nạp vào bộ nhớ, DOS sẽ tạo ra một mảng gồm 256 byte của đoạn mào đầu chương trình (program segment prefix, PSP) dùng để chứa các thông tin liên quan đến chương trình và đặt nó vào ngay phía trước phần bộ nhớ chứa mã lệnh của chương trình. DOS cũng đưa các thông số liên quan đến chương trình vào các thanh ghi DS và ES. Do vậy DS và ES không phải chứa giá trị địa chỉ của các loại dữ liệu cho chương trình. Để chương trình có thể chạy đúng ta phải có các lệnh sau để khởi động thanh ghi cho DS (hoặc cả ES nữa nếu cần).

MOV AX,@Data

MOV DS, AX

;MOV ES, AX ;nếu cần thì bỏ ';'

Trong đó @Data là tên của đoạn dữ liệu .Data định nghĩa bởi hướng dẫn chương trình dịch. Chương trình dịch sẽ dịch tên @Data thành giá trị số của đoạn dữ liệu. Ta phải dùng thanh ghi AX làm trung gian cho việc khởi đầu DS như trên là do bộ vi xử lý 8086/88, vì những lý do kỹ thuật, không cho phép chuyển giá trị số (chế độ địa chỉ tức thì) vào các thanh ghi đoạn. Thanh ghi AX cũng có thể được thay bằng các thanh ghi đa năng khác.

Sau đây là ví dụ của một chương trình hợp ngữ được viết để dịch ra chương trình với đuôi EXE. Khi cho chạy, chương trình này sẽ hiện lên màn hình lời chào 'Hello' nằm giữa 2 dòng trống cách đều các dòng mang dấu nhắc của DOS.

Chương trình Hello.EXE

.Model Small

.Stack 100

.Data

CRLF DB 13,10,'S'

MSG DB 'Hello!\$'

.Code

MAIN proc

; khởi đầu thanh ghi DS

MOV AX, @Data

MOV DS, AX

; về đầu dòng mới dùng hàm 9 của INT 21H

MOV AH, 9

LEA DX, CRLF

INT 21H

; hiển thị lời chào dùng hàm 9 của INT 21H

MOV AH, 9

LEA DX, MSG

INT 21H

; về đầu dòng mới dùng hàm 9 của INT 21H

MOV AH, 9

LEA DX, CRLF

INT 21H

; trở về DOS dùng hàm 4Ch của INT 21H

MOV AH, 4Ch

INT 21H

MAIN Endp

END MAIN

Trong ví dụ trên chúng ta đã sử dụng các dịch vụ có sẵn (các hàm 9 và 4Ch) của ngắt INT 21H của DOS trên máy IBM PC để hiển thị xâu ký tự và trở về DOS một cách thuận lợi.

2.2. Cấu trúc chương trình .COM

Nhìn vào chương trình hợp ngữ để dịch ra tệp chương trình đuôi .EXE ta thấy có mặt đầy đủ các đoạn. Trên máy tính IBM PC, ngoài tệp chương trình với đuôi .EXE, chúng ta còn có khả năng dịch chương trình hợp ngữ có kết cấu thích hợp ra một loại tệp chương trình chạy được kiểu khác với đuôi .COM. Đây là một chương trình ngắn gọn và đơn giản hơn nhiều so với tệp chương trình đuôi .EXE, trong đó các đoạn mã, đoạn dữ liệu và đoạn ngăn xếp được gộp lại trong một đoạn duy nhất là đoạn mã. Như vậy, nếu ta có các ứng dụng mà dữ liệu và mã chương trình không yêu cầu nhiều về không gian của bộ nhớ, ta có thể ghép luôn cả dữ liệu, mã chương trình và ngăn xếp chung vào trong cùng một đoạn mã rồi tạo ra tệp .COM. Với việc tạo ra tệp này không những tiết kiệm được thời gian và bộ nhớ khi cho chạy chương trình mà còn tiết kiệm được cả không gian nhớ khi phải lưu trữ nó trên ổ đĩa.

Để có thể dịch được ra chương trình đuôi .COM thì chương trình nguồn hợp ngữ phải được kết cấu sao cho thích hợp với mục đích này.

Sau đây là khung của một chương trình hợp ngữ để dịch được ra tệp chương trình đuôi .COM.

.Model Tiny

.Code

ORG 100h

START: JMP CONTINUE

;các định nghĩa cho biến và hằng để tại đây

CONTINUE:

Main Proc

; các lệnh của chương trình chính để tại đây

INT 20H ;trở về DOS

Main Endp

;các chương trình con (nếu có) để tại đây

END START

So sánh khung này với khung cho chương trình .EXE ta thấy trong khung này không có khai báo đoạn ngăn xếp và đoạn dữ liệu, còn khai báo quy mô sử dụng bộ nhớ là kiểu Tiny. Ở ngay đầu đoạn mã là lệnh giả ORG (origin: điểm xuất phát) và lệnh JMP (nhảy). Lệnh giả ORG 100H dùng để gán địa chỉ bắt đầu

cho chương trình tại 100H trong đoạn mã, chứa lại vùng nhớ với dung lượng 256 byte (từ địa chỉ 0 đến địa chỉ 255) cho đoạn mào đầu chương trình (PSP).

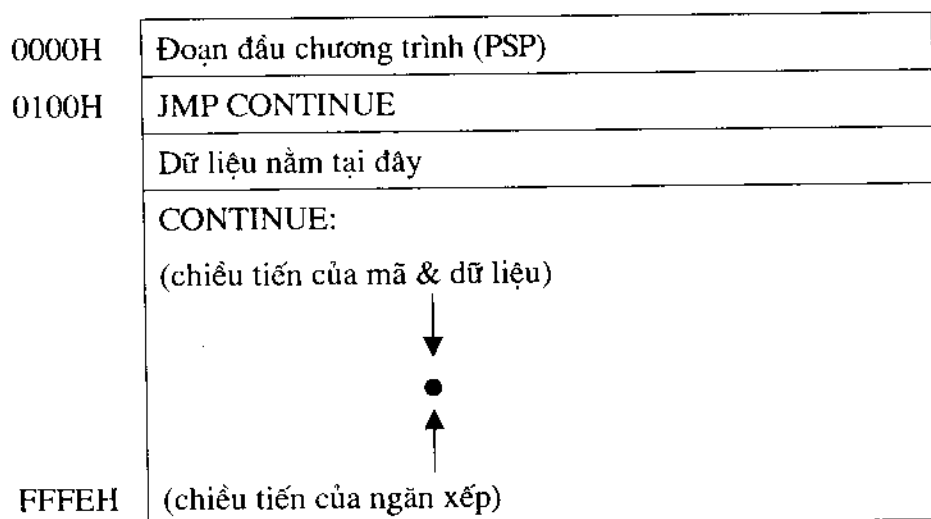
Lệnh JMP sau nhãn Sart dùng để nhảy qua phần bộ nhớ dành cho việc định nghĩa và khai báo dữ liệu (về nguyên tắc, dữ liệu có thể được đặt vào ở đầu hoặc ở cuối đoạn mã, nhưng ở đây ta đặt nó ở đầu đoạn mã để có thể áp dụng các định nghĩa đơn giản đã nói). Đích của lệnh nhảy là phần đầu của chương trình chính, một chương trình kiểu .COM được nạp vào và sắp xếp trong một đoạn mã của bộ nhớ được thể hiện ở hình “Tập chương trình .COM trong bộ nhớ”.

Ta thấy một chương trình .COM cũng được nạp vào bộ nhớ sau vùng PSP như chương trình đuôi .EXE. Ngăn xếp cho chương trình .COM được xếp đặt tại cuối đoạn mã, đỉnh của ngăn xếp lúc ban đầu là ô nhớ có địa chỉ là FFFEh. Trong trường hợp chương trình kiểu .COM này sẽ bị các hạn chế gây ra bởi:

- Dung lượng nhớ cực đại của một đoạn là 64 KB, tức là ta phải luôn chắc chắn được rằng, chương trình ứng dụng phải có số lượng byte của mã máy và dữ liệu cho chương trình không lớn lắm (nếu không nó sẽ làm cho cả nhóm này mở ra về phía địa chỉ cao của đoạn).

- Chương trình cũng chỉ được phép sử dụng ngăn xếp một cách hạn chế (nếu không điều này có thể làm cho đỉnh của nó trong khi hoạt động dâng lên nhiều phía về phía địa chỉ thấp của đoạn).

Địa chỉ lệch:



Hình 4.1: Tập chương trình .COM trong bộ nhớ

Tóm lại, chúng ta phải chắc chắn đảm bảo không thể xảy ra hiện tượng tràn vào nhau của các thông tin tại cùng ngăn xếp và thông tin tại vùng mã lệnh hoặc dữ liệu. Tuy nhiên, ta cũng không cần phải lo lắng quá đến vấn đề này, các chương trình kiểu .COM trong hầu hết các trường hợp trong thực tế vẫn có thể thỏa mãn được các yêu cầu của các bài toán mà ta muốn thử nghiệm.

Khi kết thúc chương trình kiểu .COM, để trở về DOS ta dùng ngắt INT 20H của DOS để làm cho chương trình gọn hơn. Tất nhiên ta cũng vẫn có thể dùng hàm 4Ch của ngắt INT 21H như đã dùng trong trường hợp chương trình để dịch ra tệp .EXE.

Để kết thúc toàn bộ chương trình ta dùng hướng dẫn chương trình dịch END đi kèm theo nhãn START. Ở đây nhãn SART tương ứng với địa chỉ lệnh đầu tiên của chương trình trong đoạn mã.

Sau đây là ví dụ của một chương trình hợp ngữ để dịch ra tệp chương trình chạy được với đuôi .COM.

.Model Tiny

.Code

ORG 100H

START: JMP CONTINUE

CRLF DB 13,10, 'S'

MSG DB 'Hello!S'

CONTINUE:

MAIN Proc

;về đầu dòng mới dùng hàm 9 của INT 21H

MOV AH,9

LEA DX, CRLF

INT 21H

; hiển thị lời chào

MOV AH,9

LEA DX, MSG

INT 21H

; về đầu dòng mới dùng hàm 9 của INT 21H

MOV AH,9

```
LEA DX,CRLG
INT 21H
; trở về DOS
INT20H
MAIN Endp
END SART
```

Chúng ta có thể nhận xét rằng, trong chương trình ta không cần đến các thao tác khởi đầu cho thanh ghi DS, như ta đã phải làm trong chương trình .EXE vì trong chương trình .COM không có đoạn dữ liệu nằm riêng rẽ.

3. Các cấu trúc lập trình

Khi tiến hành việc thiết kế hệ thống người ta thường dùng phương pháp thiết kế từ trên xuống. Phương pháp này vì thế cũng được áp dụng trong khi viết phần mềm cho một hệ thống nhằm giải quyết một nhiệm vụ nhất định nào đó.

Bản chất của phương pháp thiết kế này là: đầu tiên ta chia chương trình tổng thể thành các khối chức năng nhỏ hơn, các khối chức năng nhỏ này lại được chia tiếp thành các khối chức năng nhỏ hơn nữa, việc chia chức năng phải làm cho đến khi mỗi khối nhỏ này trở thành các khối chức năng đơn giản và dễ thực hiện.

Trong khi thực hiện các khối chức năng thành phần, thông thường người ta sử dụng các cấu trúc lập trình cơ bản để thực hiện các nhiệm vụ của khối đó. Điều này làm cho chương trình viết ra trở thành có cấu trúc với các ưu điểm chính là dễ phát triển, dễ hiệu chỉnh hoặc cải tiến và dễ lập tài liệu.

Để giải quyết các công việc khác nhau, thông thường trong khi viết chương trình ta chỉ cần đến 3 cấu trúc lập trình cơ bản sau:

- + Cấu trúc tuần tự.
- + Cấu trúc lựa chọn (IF-THEN-ELSE)
- + Cấu trúc lặp (WHILE-DO).

Thay đổi các cấu trúc này một chút ít, ta có thể tạo ra thêm 4 cấu trúc khác cũng rất có tác dụng khi viết chương trình:

- + Cấu trúc chọn kiểu IF-THEN.
- + Cấu trúc chọn kiểu CASE.
- + Cấu trúc lặp kiểu REPEAT-UNTIL
- + Cấu trúc lặp kiểu FOR-DO.

Đặc điểm chung của tất cả các cấu trúc lập trình cơ bản là tính chất cấu trúc: chỉ có một lối vào cấu trúc và một lối ra để ra khỏi cấu trúc đó.

Những cấu trúc lập trình kể trên là các cấu trúc mà ta đã làm quen ít nhiều khi viết chương trình ở ngôn ngữ bậc cao. Vấn đề với ta bây giờ là làm thế nào để thực hiện các cấu trúc lập trình này bằng hợp ngữ.

3.1. Cấu trúc tuần tự

Cấu trúc tuần tự là một cấu trúc thông dụng và đơn giản nhất. Trong cấu trúc này các lệnh được sắp xếp tuần tự, lệnh nọ tiếp theo lệnh kia. Sau khi thực hiện xong lệnh cuối cùng của cấu trúc thì công việc phải làm cũng được hoàn tất.

Ngữ pháp:

Lệnh 1

Lệnh 2

.....

Lệnh n

Ví dụ: Các thanh ghi CX và BX chứa các giá trị của biến c và b. Hãy tính giá trị của biểu thức $a = 2 \times (c+b)$ và chứa kết quả trong thanh ghi AX.

Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

XOR AX,AX ; tổng tại AX lúc đầu là 0.

ADD AX,BX ; cộng thêm b.

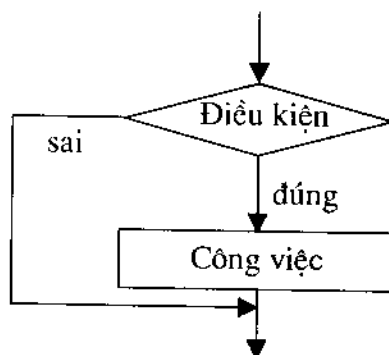
ADD AX,CX ; cộng thêm c.

SHL AX,1 ; nhân đôi kết quả trong AX.

RA: ;lối ra của cấu trúc

3.2. Cấu trúc IF-THEN

Ngữ pháp: IF Điều kiện THEN Công việc.



Hình 4.2: Cấu trúc IF-THEN

Từ ngữ pháp của cấu trúc IF-THEN ta thấy, nếu thoả mãn Điều kiện thì Côngviệc được thực hiện, nếu không Côngviệc sẽ bị bỏ qua. Điều này tương đương với việc dùng lệnh nhảy có điều kiện để bỏ qua một thao tác nào đó trong chương trình hợp ngữ.

Ví dụ:

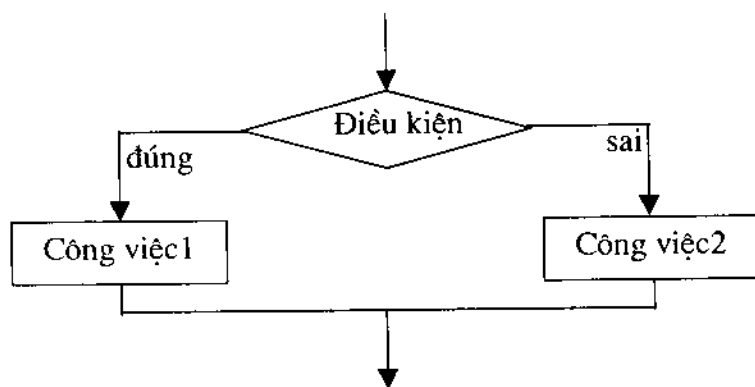
Gán cho BX giá trị tuyệt đối của AX.

Để thực hiện phép gán $BX \leftarrow [AX]$ ta có thể dùng các lệnh sau:

```
CMP AX,0          ;AX < 0 ?
JNL GAN           ; không gán luôn
NEG AX            ; đúng. Đảo dấu, rồi gán
GAN: MOV BX,AX    ; BX chứa [AX]
RA:               ; lối ra của cấu trúc.
```

3.3. Cấu trúc IF-THEN-ELSE

Ngữ pháp : IF Điều kiện THEN Côngviệc1 ELSE Côngviệc2



Hình 4.3: Cấu trúc IF-THEN-ELSE

Từ ngữ pháp của cấu trúc IF-THEN-ELSE ta thấy nếu thoả mãn Điều kiện thì Côngviệc1 được thực hiện, nếu không thì Côngviệc2 được thực hiện. Điều này tương đương với dùng lệnh nhảy có điều kiện và không điều kiện để nhảy đến các nhãn nào đó trong chương trình hợp ngữ.

Ví dụ:

Gán cho CL giá trị bit dấu của AX

Ta có thể thực hiện các công việc trên bằng mẫu chương trình sau:

```
OR AX,AX          ;AX > 0 ?
```

JNS DG	;đúng
MOV CL,1	;sai, cho CL←1 rồi
JMP RA	;đi ra.
DG:XOR CL,CL	;cho CL ←0
RA:	;lối ra của cấu trúc.

3.4. Cấu trúc CASE

Ngữ pháp: CASE biểu thức

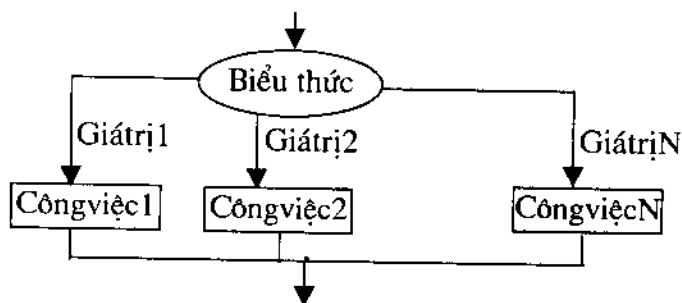
Giá trị1: Côngviệc1

Giá trị2: Côngviệc2

.....

Giá trịN: CôngviệcN

END CASE



Hình 4.4: Cấu trúc lệnh CASE

Từ ngữ pháp của cấu trúc ta thấy: nếu Biểuthức có Giá trị1 thì Côngviệc1 được thực hiện, nếu Biểuthức có Giá trị2 thì Côngviệc2 được thực hiện... điều này tương đương với việc dùng các lệnh nhảy có điều kiện và lệnh nhảy không điều kiện để nhảy đến các nhãn nào đó trong chương trình hợp ngữ. Cấu trúc CASE có thể thực hiện bằng các cấu trúc lựa chọn lồng nhau.

Ví dụ:

Dùng CX để thực hiện các giá trị khác nhau của AX theo quy tắc sau:

Nếu $AX < 0$ thì $CX = -1$,

Nếu $AX = 0$ thì $CX = 0$,

Nếu $AX > 0$ thì $CX = 1$.

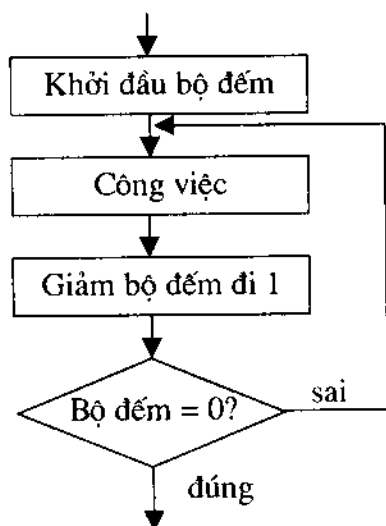
Ta có thể thực hiện công việc trên bằng chương trình sau:

CMP AX, 0 ; kiểm tra dấu của AX.

JL AM	; AX < 0.
JE KHONG	; AX = 0.
JG DUONG	; AX > 0.
AM : MOV CX, -1	
JMP RA	
DUONG: MOV CX, 1	
JMP RA	
KHONG: XOR CX, CX	
RA:	; lối ra của cấu trúc.

3.5. Cấu trúc lặp FOR - DO

Ngữ pháp: FOR Sốlầnlặp DO Côngviệc



Hình 4.5: Cấu trúc lặp FOR-DO

Từ ngữ pháp của cấu trúc FOR - DO ta thấy ở đây Côngviệc được thực hiện lặp đi lặp lại tất cả Sốlầnlặp lần. Điều này hoàn toàn tương đương với việc dùng lệnh LOOP trong hợp ngữ để lặp lại CX lần một Côngviệc nào đó, đương nhiên trước đó ta phải gán Sốlầnlặp cho thanh ghi CX.

Ví dụ: Hiển thị một dòng ký tự '\$' trên màn hình.

Một dòng trên màn hình máy tính IBM PC chứa được nhiều nhất là 80 ký tự. Ta sẽ sử dụng hàm 2H của ngắt 21H để hiển thị một ký tự. Ta phải lặp lại

công việc này 80 lần bằng lệnh LOOP. Muốn dùng lệnh này, ngay từ đầu ta phải nạp vào thanh ghi CX số lần hiển thị. Sau mỗi lần hiển thị, nội dung của CX được tự động giảm đi 1 do tác động của lệnh LOOP. Sau đây là chương trình thực hiện công việc trên.

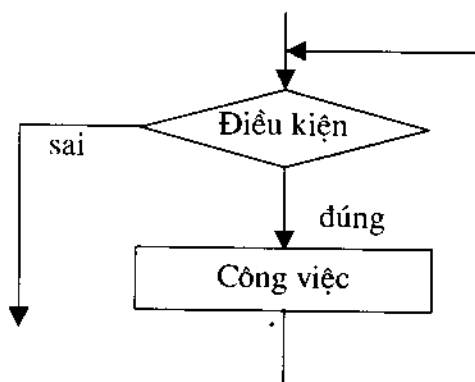
```

MOV CX, 89          ; số lần hiển thị trong CX
MOV AH, 2           ; AH chứa số hiệu hàm hiển thị
MOV DL, '$'         ; DL chứa ký tự cần hiển thị
HIEN: INT 21H        ; hiển thị
      LOOP HIEN      ; cả một dòng ký tự.
RA:                ; lối ra của cấu trúc.

```

3.6. Cấu trúc lặp WHILE - DO

Ngữ pháp: WHILE Điều kiện DO Côngviệc



Hình 4.6: Cấu trúc WHILE-DO

Từ ngữ pháp của cấu trúc WHILE - DO ta thấy: Điều kiện được kiểm tra đầu tiên. Côngviệc được lặp đi lặp lại chừng nào Điều kiện còn đúng. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng câu lệnh CMP để kiểm tra Điều kiện và sau đó dùng câu lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Ví dụ:

Đếm số ký tự đọc được từ bàn phím, khi gặp ký tự CR thì thôi.

Ta có thể thực hiện công việc trên bằng mẫu chương trình sau:

```

XOR CX,CX          ; tổng số ký tự đọc được lúc đầu là 0
MOV AH,1           ; hàm đọc ký tự từ bàn phím.
TIEP: INT 21H       ; đọc 1 ký tự, AL chứa mã ký tự.

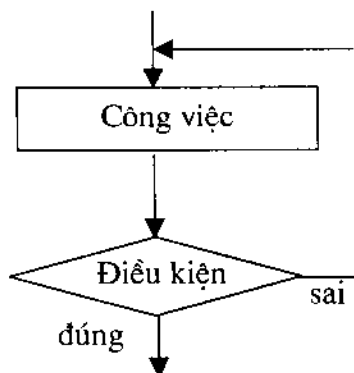
```

CMP AL,13	; đọc được CR?
JE RA	; đúng, ra.
INC CX	; sai, thêm 1 ký tự vào tổng.
JMP TIEP	; đọc tiếp.

RA: ; lỗi ra của cấu trúc.

3.7. Cấu trúc lặp REPEAT-UNTIL

Ngữ pháp: REPEAT Côngviệc UNTIL Điềukiện



Hình 4.7: Cấu trúc REPEAT-UNTIL

Từ ngữ pháp của cấu trúc REPEAT-UNTIL ta thấy: Côngviệc được thực hiện đầu tiên, điều đó có nghĩa là Côngviệc được lặp đi lặp lại cho tới khi Điềukiện được thoả mãn. Điều này trong hợp ngữ hoàn toàn tương đương với việc dùng lệnh CMP để kiểm tra Điềukiện và sau đó dùng lệnh nhảy có điều kiện để thoát khỏi vòng lặp.

Ví dụ:

Đọc ký tự từ bàn phím cho tới khi gặp '\$' thì thôi

Ví dụ này chỉ làm một phần công việc của ví dụ trước. Tại đây ta chỉ phải đọc các ký tự mà không phải đếm số ký tự đọc được.

Ta có thể thực hiện công việc trên bằng chương trình sau:

MOV AH,1	; hàm đọc ký tự từ bàn phím
TIEP: INT 21H	; đọc một ký tự.
CMP AL, '\$'	; đọc được dola ?
JNE TIEP	; chưa, đọc tiếp.
RA :	; lỗi, ra của cấu trúc.

4. Các bước tạo và dịch chương trình hợp ngữ

Như đã nói trong phần trước, máy IBM PC là phương tiện lý tưởng để chúng ta tạo ra và thử nghiệm các chương trình hợp ngữ 8088 (8086). Các bước để làm công việc này có thể liệt kê ra như sau:

Bước 1: Dùng các phần mềm soạn thảo văn bản (SK, Ncredit...) để tạo ra một tệp văn bản chương trình gốc bằng hợp ngữ. Tệp này phải được gắn đuôi .ASM.

Bước 2: Dùng chương trình dịch MASM để dịch tệp .ASM ra mã máy dưới dạng tệp .OBJ. Trong bước này, nếu trong chương trình có lỗi cú pháp thì ta phải quay lại bước 1 để sửa lại chương trình gốc.

Bước 3: Dùng chương trình LINK để nối một hay nhiều tệp OBJ lại với nhau thành một tệp chương trình chạy được với đuôi .EXE.

Bước 4: Nếu chương trình gốc viết ra là để dịch ra kiểu .COM thì ta phải dùng chương trình EXE2BIN (đọc là EXE to BIN) của DOS để dịch tiếp tệp .EXE ra tệp chương trình chạy được với đuôi .COM.

Bước 5: Cho chạy chương trình vừa dịch.

III. CÁC CHẾ ĐỘ ĐỊA CHỈ CỦA 8088

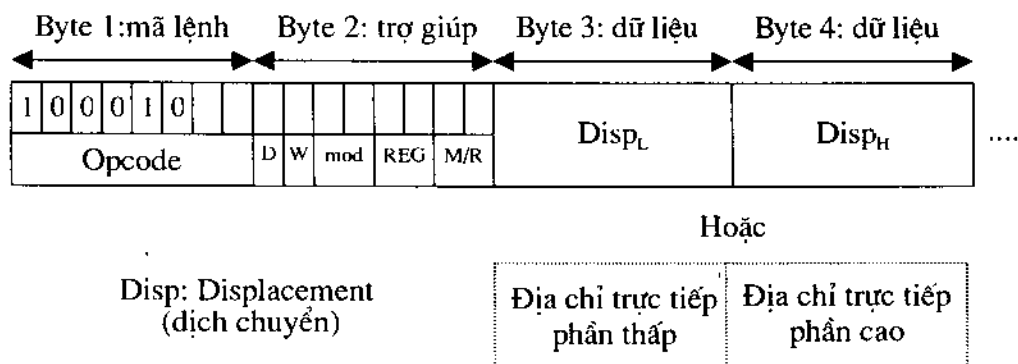
1. Mã hoá lệnh

Lệnh của bộ vi xử lý được ghi bằng các ký tự dưới dạng gọi nhớ để người sử dụng dễ nhận biết. Đối với bộ vi xử lý thì lệnh của nó được mã hoá dưới dạng các số 0 và 1 (được gọi là mã máy), vì đó là dạng biểu diễn thông tin duy nhất mà máy hiểu được. Vì lệnh của bộ vi xử lý được cho dưới dạng mã máy nên sau khi nhận lệnh, bộ vi xử lý phải thực hiện giải mã lệnh rồi mới thực hiện lệnh. Việc hiểu rõ bản chất cách ghi lệnh bằng số hệ hai cho bộ vi xử lý sẽ có lợi cho ta khi dịch bằng 'tay' một lệnh gọi nhớ khi ta làm việc với các bộ vi xử lý (việc này ít khi xảy ra vì ta thường làm việc với các bộ vi xử lý được trang bị chương trình dịch hợp ngữ).

Một lệnh có thể có độ dài một vài byte tùy theo bộ vi xử lý. Giả sử một bộ vi xử lý nào đó dùng 1 byte để chứa các mã lệnh (Opcode) của nó. Ta có thể tính được số lệnh lớn nhất mà 1 byte này có thể mã hoá được là 256 lệnh. Trong thực tế, việc ghi lệnh không phải hoàn toàn đơn giản như vậy. Việc mã hoá lệnh cho bộ vi xử lý là rất phức tạp và bị chi phối bởi nhiều yếu tố khác nữa.

Đối với bộ vi xử lý 8088, một lệnh có thể có độ dài từ 1 đến 6 byte. Ta sẽ lấy trường hợp lệnh MOV để giải thích các lệnh của 8088.

Lệnh MOV đích, nguồn dùng để chuyển dữ liệu giữa hai thanh ghi hoặc giữa ô nhớ và thanh ghi. Chỉ xét riêng với các thanh ghi của 8088, nếu ta lần lượt đặt các thanh ghi vào vị trí các toán hạng đích và nguồn ta thấy phải cần tới hàng trăm mã lệnh khác nhau để mã hoá tổ hợp các lệnh này. Hình sau đây biểu diễn dạng thức các byte dùng để mã hoá lệnh MOV.



Hình 4.8: Dạng thức các byte mã hoá lệnh của lệnh MOV

Ta thấy rằng, để mã hoá lệnh MOV ta cần ít nhất 2 byte, trong đó 6 bit của byte đầu dùng để chứa mã lệnh. Đối với các lệnh MOV, để chuyển dữ liệu kiểu:

Thanh ghi ↔ Thanh ghi (trừ thanh ghi đoạn) hoặc

Bộ nhớ ↔ Thanh ghi (trừ thanh ghi đoạn)

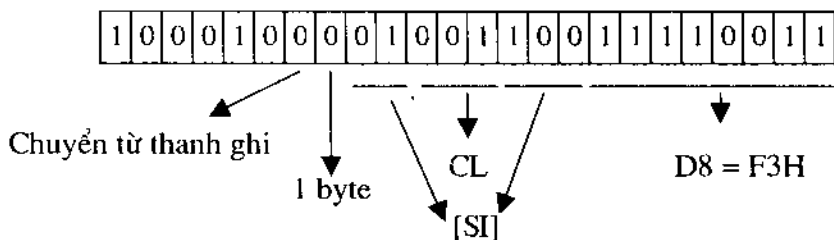
thì 6 bit đầu này luôn là 100010. Đối với các thanh ghi đoạn thì điều này lại khác.

Bit W dùng để chỉ ra rằng, một byte (W = 0) hoặc một từ (W = 1) sẽ được chuyển.

Trong các thao tác dữ liệu, một toán hạng luôn bắt buộc phải là thanh ghi. Bộ vi xử lý dùng 2 hoặc 3 bit để mã hoá các thanh ghi trong CPU như sau:

Thanh ghi		Mã	Thanh ghi đoạn	Mã
W = 1	W = 0			
AX	AL	000	CS	01
BX	BL	011	DS	11
CX	CL	001	ES	00
DX	DL	010	SS	10
SP	AH	100		
DI	BH	111		
BP	CH	101		
SI	DH	110		

- MOV 0F3H [SI], CL



2. Các chế độ địa chỉ

Chế độ địa chỉ (addressing mode) là cách để CPU tìm thấy toán hạng cho các lệnh của nó khi hoạt động. Một bộ vi xử lý có thể có nhiều chế độ địa chỉ, các chế độ địa chỉ này được xác định ngay từ khi chế tạo ra bộ vi xử lý và sau này không thể thay đổi được. Bộ vi xử lý 8088 có đến bảy chế độ địa chỉ như sau:

- Chế độ địa chỉ thanh ghi (Register addressing mode).
- Chế độ địa chỉ tức thì (Immediate addressing mode).
- Chế độ địa chỉ trực tiếp (Direct addressing mode).
- Chế độ địa chỉ gián tiếp qua thanh ghi (Register indirect addressing mode).
- Chế độ địa chỉ tương đối cơ sở (Based relative addressing mode).
- Chế độ địa chỉ tương đối chỉ số (Indirect relative addressing mode).
- Chế độ địa chỉ tương đối chỉ số cơ sở (Based indexed relative addressing mode).

Sau đây ta đi nghiên cứu chi tiết các chế độ địa chỉ này.

2.1. Chế độ địa chỉ thanh ghi

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

Ví dụ:

MOV BX, DX ; chuyển nội dung DX vào BX.

MOV DS, AX ; chuyển nội dung AX vào DS.

ADD AL, DL ; cộng nội dung AL và DL rồi đưa vào AL.

2.2. Chế độ địa chỉ tức thì

Trong chế độ địa chỉ này toán hạng đích là một thanh ghi hay một ô nhớ, còn toán hạng nguồn là một hằng số và ta có thể tìm thấy toán hạng này ở ngay

sau mã lệnh, (chính vì vậy chế độ địa chỉ này có tên là chế độ địa chỉ tức thì). Ta có thể dùng chế độ địa chỉ này để nạp dữ liệu cần thao tác vào bất kỳ thanh ghi nào (trừ các thanh ghi đoạn và thanh cờ) hoặc vào bất kỳ ô nhớ nào trong đoạn dữ liệu DS.

Ví dụ:

```
MOV    CL, 100      ; chuyển 100 vào CL
MOV    AX, 0FF0H    ; chuyển 0FF0H vào AX để rồi đưa
MOV    DS, AX        ; vào DS (vì không thể chuyển trực tiếp
                      ; vào thanh ghi đoạn)
MOV    [BX], 10      ; chuyển 10 vào ô nhớ tại địa chỉ DS: BX
```

Trong thí dụ, dòng lệnh cuối ta đã dùng thanh ghi để chỉ ra ô nhớ (toán hạng đích) sẽ nhận dữ liệu ở chế độ địa chỉ tức thì (toán hạng nguồn). Tại đây [BX] có nghĩa là ô nhớ có địa chỉ DS:BX.

2.3. Chế độ địa chỉ trực tiếp

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

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

Ví dụ:

```
MOV    AL, [1234H]   ; chuyển nội dung ô nhớ DS: 1234 vào AL
MOV    [4320H], CX    ; chuyển nội dung CX vào 2 ô nhớ
                      ; liên tiếp DS: 4320 và DS: 4321
```

2.4. Chế độ địa chỉ gián tiếp qua thanh ghi

Trong chế độ địa chỉ này, một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệch của ô nhớ chứa dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ (8088 không cho phép quy chiếu bộ nhớ 2 lần đối với một lệnh).

Ví dụ:

```
MOV    AL, [BX]      ; chuyển nội dung ô nhớ có địa chỉ DS:BX vào AL
MOV    [SI], CL       ; chuyển nội dung CL vào ô nhớ có địa chỉ DS: SI
MOV    [DI], AX       ; chuyển nội dung AX vào 2 ô nhớ liên tiếp có địa
                      ; chỉ DS: DI và DS: (DI + 1)
```

2.5. Chế độ địa chỉ tương đối cơ sở

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

Ví dụ:

MOV	CX, [BX] + 10	; chuyển nội dung 2 ô nhớ liên tiếp có địa chỉ ; DS : (BX + 10) và DS : (BX + 11) vào CX.
MOV	CX, [BX+10]	; 1 cách viết khác của lệnh trên.
MOV	CX, 10 [BX]	; 1 cách viết khác của lệnh đầu.
MOV	AL, [BP] + 5	; chuyển nội dung ô nhớ SS : (BP+5) vào AL
ADD	AL, Table [BX]	; cộng AL với nội dung ô nhớ do BX chỉ ra ; trong bảng table (bảng này nằm trong DS), ; kết quả đưa vào AL

Trong đó:

- 10.5. Table gọi là các dịch chuyển của các toán hạng tương ứng. 10 và 5 là các giá trị cụ thể, Table là tên mảng biểu diễn dịch chuyển của mảng (phần tử đầu tiên) so với địa chỉ của đoạn dữ liệu DS.

- (BX + 10) hoặc (BP + 5) gọi là địa chỉ hiệu dụng (effective address. EA, theo cách gọi của Intel).

- DS : (BX + 10) hoặc SS : (BP + 5) chính là logic tương ứng với một địa chỉ vật lý.

- Theo cách định nghĩa này thì địa chỉ hiệu dụng của một phần tử thứ BX nào đó (kể từ 0) trong mảng Table [BX] thuộc đoạn DS là $EA = Table + BX$ và của phần tử đầu tiên $EA = Table$.

2.6. Chế độ địa chỉ tương đối chỉ số

Trong chế độ địa chỉ này, các thanh ghi chỉ số như SI và DI và các hằng số biểu diễn các giá trị dịch chuyển (displacement values) được dùng để tính địa chỉ của toán hạng trong vùng nhớ DS.

Ví dụ:

MOV	AX, [SI] + 10	; chuyển nội dung 2 ô nhớ liên tiếp có địa chỉ ; DS : (SI + 10) và DS : (SI + 11) vào AX.
MOV	AX, [SI + 10]	; 1 cách viết khác của lệnh trên.
MOV	CL, [DI] + 5	; chuyển nội dung ô nhớ DS : (DI+5) vào CL

2.7. Chế độ địa chỉ tương đối chỉ số cơ sở

Kết hợp hai chế độ địa chỉ số và địa chỉ cơ sở ta có chế độ địa chỉ số cơ sở. Trong chế độ địa chỉ này ta dùng cả thanh ghi chỉ số để tính địa chỉ của toán hạng. Nếu ta dùng thêm cả thành phần biểu diễn sự dịch chuyển của địa chỉ thì ta có chế độ địa chỉ phức hợp nhất: chế độ địa chỉ tương đối chỉ số cơ sở. Ta có thể thấy chế độ địa chỉ này rất phù hợp cho việc địa chỉ hoá các mảng 2 chiều.

Ví dụ:

MOV AX, [BX] [SI]+8 ; chuyển nội dung 2 ô nhớ liên tiếp có địa chỉ ; DS : (BX+SI+8) và DS: (BX+SI+9) vào AX.

MOV AX, [BX+SI+8] ; 1 cách viết khác của lệnh trên.

MOV CL, [BP+DI+5] ; chuyển nội dung ô nhớ SS:(BP+DI+5) vào CL

- Tóm lại các chế độ địa chỉ của 8088 được tóm tắt theo bảng sau:

Bảng 4.3: Tóm tắt các chế độ địa chỉ của 8088

Chế độ địa chỉ	Toán hạng	Thanh ghi đoạn ngầm định
Thanh ghi	Reg	
Tức thì	Data	
Trực tiếp	[offset]	DS
Gián tiếp qua thanh ghi	[BX]	DS
	[SI]	DS
	[DI]	DS
Tương đối cơ sở	[BX] + Disp	DS
	[BP] + Disp	SS
Tương đối chỉ số	[DI] + Disp	DS
	[SI] + Disp	DS
Tương đối chỉ số cơ sở	[BX] + [DI] + Disp	DS
	[BX] + [SI] + Disp	DS
	[BP] + [DI] + Disp	SS
	[BP] + [SI] + Disp	SS

Trong đó:

Reg: Thanh ghi, Data: Dữ liệu tức thì, Disp: Dịch chuyển.

- Phương pháp bỏ ngầm định thanh ghi đoạn:

Các thanh ghi đoạn và thanh ghi lệch được ngầm định đi kèm với nhau theo từng cặp dùng để địa chỉ hoá các toán hạng trong các vùng khác nhau của bộ nhớ được tóm tắt bằng bảng sau:

Bảng 4.4: Các cặp thanh ghi đoạn và thanh ghi lệch ngầm định

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

Vì tính ngầm định của các thanh ghi đoạn nên trong các lệnh ta chỉ cần viết ra các thanh ghi lệch là đủ cơ sở để tính ra được địa chỉ của các toán hạng. Tuy nhiên, ngoài các tổ hợp ngầm định kể trên, 8088 còn cho phép ta làm việc với các tổ hợp khác của các thanh ghi đoạn và thanh ghi lệch. Muốn loại bỏ các tổ hợp ngầm định nói trên, trong khi viết lệnh ta phải ghi rõ thanh ghi đoạn sẽ dùng để tính địa chỉ và kèm theo dấu 2 chấm trước thanh ghi lệch. Cụm ký hiệu này được gọi là cụm tiếp đầu để loại bỏ thanh ghi đoạn ngầm định (Segment override prefix) và để đạt được việc loại bỏ này chỉ cần ghi rõ thanh ghi đoạn.

Ví dụ: Trong lệnh chuyển dữ liệu:

`MOV AL, [BX]`

thì địa chỉ vật lý của toán hạng để chuyển vào thanh ghi AL tương ứng với DS:BX, vì DS là thanh ghi đoạn ngầm định của vùng nhớ chứa toán hạng do BX chỉ ra. Nếu ta muốn thay đổi, không lấy toán hạng trong DS nữa, mà lại lấy toán hạng trong đoạn dữ liệu phụ ES để đưa vào AL, thì ta phải viết lệnh trên thành:

`MOV AL, ES:[BX]`

trong đó ta đã dùng cụm tiếp đầu ES: để loại bỏ thanh ghi đoạn ngầm định DS và để chỉ rõ là thanh ghi đoạn mới dùng trong lệnh này bây giờ là ES.

IV. TẬP LỆNH CỦA 8088

1. Các lệnh cơ bản

1.1. Các lệnh chuyển dữ liệu

* *MOV* **Đích, nguồn**

Lệnh này dùng để copy nội dung toán hạng nguồn sang toán hạng đích.

Đích và nguồn ở đây có thể thanh ghi 8 bit, 16 bit hay biến nhớ, nguồn còn có thể là dữ liệu trực tiếp. Tuy nhiên, đích và nguồn không thể đồng thời là biến nhớ.

Ví dụ:

MOV AX, BX ;copy nội dung thanh ghi BX 16 bit sang AX

MOV AH, BL ;copy nội dung thanh ghi BL 8 bit cho AH

MOV CH, BIEN ;copy biến nhớ BIEN 1 BYTE cho thanh ghi CH.

Đây là một lệnh đặc biệt thông dụng trong hợp ngữ, có thể nói một chương trình hợp ngữ nói chung gồm 80% là lệnh MOV.

*** MOVSB**

Lệnh này dùng để chuyển một byte dữ liệu từ địa chỉ DS:[SI] (địa chỉ đoạn nằm trong thanh ghi DS, địa chỉ offset trong thanh ghi SI), đến byte nhớ có địa chỉ ES: [DI].

Sau khi chuyển xong byte dữ liệu này các con trỏ DI, SI đều tăng lên một vị trí.

Lệnh này không ảnh hưởng đến cờ.

Chú ý các thanh ghi đoạn không thể gán giá trị một cách trực tiếp bằng lệnh MOV.

*** MOVSW**

Lệnh này có chức năng tương tự lệnh MOVSB nhưng ở đây là chuyển một WORD (2 byte) từ địa chỉ DS:[SI] tới địa chỉ ES:[DI].

*** LEA reg16, addr**

Đây là lệnh dùng để nạp giá trị cho thanh ghi 16 bit bằng địa chỉ offset của một biến nhớ nào đó.

Ví dụ:

*** LEA AX, bien1**

Sau khi thực hiện lệnh này thanh ghi AX được nhận giá trị là địa chỉ offset của biến nhớ có tên bien1.

Ví dụ:

LEA SI, bien2

LEA DI, bien5

MOVSW

Các lệnh trên để 1 word từ bien2 đến bien5.

*** LODSB**

Chuyển một byte dữ liệu tại vị trí con trỏ offset SI (địa chỉ đoạn chứa trong DS) vào thanh ghi AL sau đó con trỏ SI được tăng lên 1.

Ví dụ:

LEA SI, bien1

LODSB

Các lệnh trên để lấy byte nhớ tại bien1 vào thanh ghi AL, sau đó SI trỏ đến byte bên cạnh (vì được tự động tăng 1).

*** STOSB**

Chuyển một byte dữ liệu từ thanh ghi AL ra địa chỉ ES: [DI] sau đó con trỏ DI được tăng lên 1.

Ví dụ:

LEA DI, bien3

STOSB

Các lệnh trên để đưa ra giá trị trong thanh ghi AL ra bien3.

*** LDS reg16, mem**

Ví dụ :

LDS AX, bien3.

Lệnh trên sẽ chuyển 1 word tại biến nhớ bien3 vào thanh ghi AX, và word tiếp theo vào thanh ghi DS.

Các lệnh chuyển số liệu ở trên không ảnh hưởng đến cờ.

1.2. Các lệnh số học và logic

*** ADC Đích, nguồn**

Lệnh trên cộng toán hạng nguồn với toán hạng đích và với cờ nhớ Cy, kết quả lại cất ra toán hạng đích.

$(\text{Đích}) = (\text{Đích}) + (\text{Nguồn}) + \text{Cy}$

Ví dụ:

ADC AX, BX ; thực hiện phép cộng giữa hai thanh ghi
; kết quả cất ra AX

*** ADD Đích, nguồn**

Lệnh này có chức năng tương tự lệnh ADC nhưng không cộng với cờ nhớ.

*** SUB Đích, nguồn**

Lệnh này trừ toán hạng nguồn cho toán hạng đích, kết quả lại cất vào toán hạng đích.

$$(\text{Đích}) = (\text{Nguồn}) - (\text{Đích})$$

Ví dụ:

SUB AX, CX ; thực hiện việc trừ CX cho AX, kết quả cất vào AX

*** SBB Đích, nguồn**

Lệnh này chức năng tương tự lệnh SUB, nhưng ở đây trừ thêm cờ nhớ Cy.

*** MUL nguồn**

Lệnh này thực hiện phép nhân toán hạng nguồn với thanh ghi AL hay AX. Tùy theo toán hạng nguồn là Byte hay Word. Kết quả được cất ra AX hay DX:AX.

Ví dụ:

MUL BH

Lệnh này thực hiện phép nhân BH với AL, được kết quả cất ra AX.

MUL CX

Lệnh này nhân AX với CX, kết quả cất ra hai thanh ghi DX và AX.

*** DIV nguồn**

Lệnh này thực hiện phép chia với toán hạng nguồn là số chia có thể là byte hay word, số bị chia tương ứng nằm trong AX hay DX: AX khi số chia là word. Được thương cất vào AL hay AX nếu là phép chia cho word, số dư cất trong thanh ghi AH hoặc DX khi phép chia cho word.

Ví dụ:

DIV CH

Lệnh này thực hiện phép chia AX cho CH, được kết quả cất ra AL, số dư cất ra AH.

DIV BX

Lệnh này chia 2 word trong DX:AX cho BX, kết quả cất ra thanh ghi AX, số dư cất ra thanh ghi DX.

Ngoài ra còn có các lệnh IMUL, IDIV để nhân và chia các số có dấu.

*** INC Đích**

Lệnh này để tăng toán hạng đích lên 1, có thể là thanh ghi hay biến nhớ 8 bit hoặc 16 bit.

Ví dụ: INC BX

Để tăng thanh ghi BX lên 1.

INC CH

Để tăng thanh ghi CH lên 1

*** DEC Đích**

Lệnh này để giảm toán hạng đích đi 1, có thể là thanh ghi hay biến nhớ 8 bit hoặc 16 bit.

Ví dụ: DEC BX

Để giảm thanh ghi BX đi 1

DEC CH

Để giảm thanh ghi CH đi 1

*** CMP Đích, nguồn**

Lệnh này để so sánh toán hạng nguồn và toán hạng đích.

*** CLD**

Lệnh này để xoá cờ định hướng DF.

*** STD**

Lệnh này để thiết lập cờ định hướng DF.

*** AND Đích, nguồn**

Lệnh trên thực hiện phép AND logic giữa toán hạng nguồn và toán hạng đích, kết quả lại cất ra toán hạng đích.

(Đích) = (Đích) AND (Nguồn).

Ví dụ:

AND DX, CX ; thực hiện phép AND giữa DX, CX kết quả cất ra DX

AND CL, 45h ; thực hiện phép AND giữa CL và 45h kết quả cất ra CL.

*** OR Đích, nguồn**

Lệnh trên thực hiện phép OR logic giữa toán hạng nguồn và toán hạng đích, kết quả lại cất ra toán hạng đích.

(Đích) = (Đích) OR (Nguồn).

Ví dụ:

OR CH, BH ; thực hiện phép OR giữa CH, BH kết quả cất vào CH

OR BL, FAh ; thực hiện phép OR giữa BL với giá trị FAh.

*** XOR Đích, nguồn**

Lệnh trên thực hiện phép XOR logic giữa toán hạng nguồn và toán hạng đích, kết quả lại cất ra toán hạng đích.

(Đích) = (Đích) XOR (Nguồn).

Ví dụ:

XOR CH, BH

*** NOT Đích**

Lệnh này để lấy phủ định của toán hạng đích.

*** RCL Đích, 1 hoặc RCL Đích, CL**

Lệnh này để quay toán hạng đích sang trái 1 bit, hoặc CL bit, khi quay cờ CF tham gia vào vòng.

Ví dụ:

RCL AH, 1 ; quay các bit của AH sang trái 1 bit.

MOV CL, 04

RCL BH, CL ; quay các bit của BH sang trái 4 bit.

*** RCR Đích, 1 hoặc RCR Đích, CL**

Lệnh này để quay toán hạng đích sang phải 1 bit, hoặc CL bit, khi quay cờ CF tham gia vào vòng.

Ví dụ:

RCR BL, 1 ; quay các bit của BL sang phải 1 bit.

MOV CL, 05

RCR BL, CL ; quay các bit của BL sang phải 5 bit.

Các lệnh ROR; ROL cũng có cú pháp và chức năng quay tương tự nhưng không có sự tham gia của cờ CF trong vòng quay.

*** SHR đích, 1 hoặc SHR đích, CL**

Lệnh này dịch toán hạng đích sang phải 1 bit, hoặc CL bit.

Ví dụ:

SHR DX, 1; dịch các bit của DX sang phải 1 bit .

Ví dụ:

SHL CX, 1; dịch các bit của CX sang trái 1 bit

MOV CL, 03

SHL DX, CL; dịch các bit của DX sang trái 3 bit.

*** TEST Đích, nguồn**

Lệnh này thực hiện phép AND logic giữa toán hạng nguồn và toán hạng đích. Sau khi thực hiện phép tính này đích và nguồn không đổi giá trị nhưng kết quả ảnh hưởng đến các cờ.

*** STI**

Lệnh này để thiết lập IF lên 1.

*** CLI**

Lệnh này để xoá IF về 0.

1.3. Các lệnh rẽ nhánh

*** JMP Đích**

Đây là lệnh nhảy không điều kiện tới một đích nào đó, đích thường được chỉ ra bởi một nhãn.

Lưu ý: Chúng ta có thể đặt các nhãn ở bất cứ vị trí nào trong chương trình theo cú pháp: <tên nhãn:>

Ví dụ:

...

JMP CUOI ; nhảy tới thực hiện các lệnh tại nhãn CUOI

...

CUOI:

... Các lệnh tại nhãn CUOI

*** J <điều kiện> Đích**

Đây là lệnh nhảy tới nhãn đích theo điều kiện.

Ví dụ:

JZ END ; nhảy tới nhãn END, nếu cờ ZERO được lập.

*** LOOP Nhãn**

Lệnh này để nhảy chương trình về 1 nhãn nào đó nếu như CX còn khác 0, sau mỗi vòng nhảy về CX được giảm 1.

Ví dụ:

MOV CX, 100

LAP:

... Các lệnh trong vòng lặp

LOOP LAP

Với cấu trúc trên, các lệnh trong vòng lặp sẽ được lặp lại 100 lần.

*** LOOPE/LOOPZ**

Lệnh này để nhảy chương trình về một nhãn nào đó nếu như CX còn khác 0 và cờ ZF = 1, sau mỗi vòng nhảy về CX được giảm 1.

Ví dụ:

LOOPE HANOI ; lệnh này để lại quay về nhãn HANOI nếu CX chưa bằng 0 và cờ Equal đang được lập.

1.4. Các lệnh vào ra, vào ra ngăn xếp và gọi thủ tục

*** IN thanh chứa, cổng**

Lệnh này để nhập dữ liệu 1 byte hoặc word từ cổng I/O vào thanh chứa. Cổng ở đây là có thể là địa chỉ trực tiếp hay địa chỉ chứa trong thanh ghi DX.

Ví dụ:

IN AH, ACh; đọc 1 byte tại cổng ACh vào AH.

IN CX, DX; đọc 1 word từ cổng địa chỉ chứa trong DX vào CX.

*** OUT thanh chứa, cổng**

Lệnh này để đưa dữ liệu byte hay word từ thanh chứa ra cổng I/O. Cổng ở đây có thể là địa chỉ trực tiếp hay địa chỉ chứa trong thanh ghi DX. Ngoài ra có thể đưa giá trị trực tiếp ra cổng.

Ví dụ:

OUT AX, ABh; đưa giá AX ra cổng ABh

OUT 20, CCh ; đưa giá trị 20 ra cổng CCh

*** CALL <Tên thủ tục>**

Lệnh này để gọi một thủ tục đã được khai báo.

Ví dụ:

CALL Vidu ; Gọi thủ tục Vidu

; khai báo thủ tục vidu

Vidu PROC

... Các lệnh của thủ tục Vidu

*** RET**

Lệnh này để trở về chương trình chính khi kết thúc thủ tục.

RET

Vidu ENDP

*** *PUSH nguồn***

Cất toán hạng nguồn 2 byte vào đỉnh ngăn xếp, khi đó SP được giảm 2.

Ví dụ:

PUSH AX; cất AX ra ngăn xếp

PUSH SI

*** *PUSHF***

Cất thanh ghi cờ vào ngăn xếp, khi đó SP cũng được giảm 2.

*** *POP Đích***

Toán hạng đích được lấy giá trị là 2 byte đỉnh ngăn xếp, sau đó SP cũng được tăng 2.

*** *INT Số hiệu ngắt***

Lệnh này để gọi một chương trình ngắt.

Ví dụ :

INT 10h ; gọi ngắt màn hình có số hiệu 10h.

*** *IRET***

Để trở về chương trình gọi ngắt khi kết thúc một chương trình ngắt.

2. Các ngắt cơ bản

Với những người lập trình hệ thống thì việc sử dụng các dịch vụ ngắt là gần như không thể tránh khỏi trong mỗi chương trình, vì đây chính là các dịch vụ giúp chúng ta giao tiếp với các thiết bị trong hệ thống. Khi chúng ta thực hiện lệnh INT để gọi một số hiệu ngắt, thì tương ứng lúc đó có một chương trình con phục vụ ngắt sẽ được thực hiện. Nhiệm vụ của chương trình con phụ thuộc vào số hiệu ngắt và thực chất là phụ thuộc vào thiết bị phục vụ. Ngắt có thể là của DOS hoặc BIOS. Trước khi gọi một ngắt chúng ta phải xác định được là nhiệm vụ của ngắt và các tham số cần truyền cho ngắt đó, chủ yếu là thông qua các thanh ghi. Ví dụ ngoài việc xác định số hiệu ngắt cần gọi, trước khi thực hiện lệnh INT, chúng ta thường phải đặt số liệu hàm hay dịch vụ của ngắt cần phục vụ trong thanh ghi AH và đặt các tham số cần truyền trong các thanh ghi DX, CX, BX...

Cụ thể về các ngắt của DOS và BIOS xin mời các bạn tham khảo quyển “*Cẩm nang lập trình hệ thống T1, T2* của nhóm tác giả Nguyễn Mạnh Hùng, Nhà xuất bản Khoa học kỹ thuật”. Sau đây xin giới thiệu một số hàm của các ngắt cơ bản.

2.1. Ngắt DOS

Trong phần này tập trung giới thiệu ngắt 21H của DOS, còn các ngắt khác của DOS sẽ được giới thiệu các tài liệu chuyên ngành khác. Ngắt 21H của DOS có các hàm cơ bản như sau:

*** Hàm 4Ch**

Hàm này để kết thúc chương trình hiện tại và trả điều khiển cho chương trình gọi nó. Thông thường các chương trình được gọi từ DOS thì lệnh này để quay trở về DOS.

*** Hàm 2h: Hiển thị**

Đưa ký tự trong DL tới thiết bị ra chuẩn.

Vào: AH = 02h.

DL = ký tự

Ra: Không.

Ví dụ:

Các lệnh sau để đưa ký tự "A" ra màn hình.

MOV DL, 'A'

MOV AH, 02h

INT 21h

*** Hàm 9h: In chuỗi**

Đưa chuỗi ký tự ra tới thiết bị ra chuẩn.

Vào: AH = 09h.

DS:DX = Con trỏ đến chuỗi ký tự kết thúc bằng '\$'

Ra: Không.

Ví dụ:

Hiển thị dòng chữ 'Welcome to ASSEMBLER' ra màn hình tại toạ độ hàng cột là (10, 20), nhưng trước đó phải xoá màn hình. Nội dung chương trình như sau:

. Model Small

. Stack 100h

. Data

. Message dB 'Welcome to ASSEMBLER'

. Code

Main PROC

MOV AH, 06h; cuộn cửa sổ lên trên

MOV AL, 00 ; số dòng cuộn bằng 0 là xoá toàn bộ

MOV DH, 24

```

MOV DL, 79
MOV CH, 00
MOV CL, 00
MOV BH, 07h
INT 10h ; gọi hàm 06 ngắt 10h để xoá màn hình
MOV AH, 02h
MOV DH, 10 ; toạ độ hàng
MOV DL, 20 ; toạ độ cột
MOV BH, 00
INT 10h ; gọi hàm 02 ngắt 10h để định vị con trỏ
MOV AX, @DATA; lấy địa chỉ đoạn DATA vào AX
MOV DS, AX ; đưa địa chỉ đoạn vào DS
MOV AH, 09h ; hàm 09h
LEA DX, Message ; trỏ đến chuỗi ký tự
INT 21h
MOV AH, 4Ch ; quay về DOS
INT 21h
Main EndP
END Main

```

*** Hàm 3Ch: Tạo lập một file (CREAT)**

Tạo ra một file mới hoặc cắt bỏ file cũ cho có độ dài bằng 0 để chuẩn bị ghi.

Vào: AH = 3Ch

DS: DX = con trỏ trỏ đến một chuỗi ASCIIZ (kết thúc 0), để biểu diễn tiền file cần tác động.

CX = thuộc tính của file (CX = 0 là Normal).

Ra: AX = mã lỗi nếu cờ nhớ được thiết lập.

Thẻ 16 bit nếu cờ nhớ không được thiết lập.

*** Hàm 3Dh : Mở một file**

Mở một file cho trước.

Vào: AH = 3Dh

DS: DX = con trỏ, trỏ đến một tên đường dẫn ASCIIZ.

AI = mã truy nhập (thường là bằng 0 để đọc).

Ra: AX = mã lỗi nếu cờ nhớ được thiết lập.

Thẻ 16 bit nếu cờ nhớ không được thiết lập.

*** Hàm 3Eh : Đóng một thẻ file.**

Đóng một thẻ file cho trước.

Vào: AH = 3Eh

BX = thẻ file trả về bởi thao tác mở hay tạo lập file.

Ra: AX = mã lỗi nếu cờ nhớ được thiết lập.

Không có nghĩa nếu cờ nhớ không được thiết lập.

*** Hàm 3Fh: Đọc từ một file hay thiết bị.**

Chuyển một số cho trước các byte từ một file vào bộ đệm cho trước.

Vào: AH = 3Eh

BX = thẻ file

DS:DX = địa chỉ bộ đệm

CX = Số byte đọc

Ra: AX = số byte đọc được, là mã lỗi nếu cờ nhớ được thiết lập.

Ví dụ:

Hãy hiển thị 100 byte đầu tiên của file .BAT ra màn hình dưới các dạng ký tự.

Vậy với nhiệm vụ ở trên, chương trình phải thực hiện những thao tác sau:

- Mở file (sử dụng hàm 3Dh).
- Đọc dữ liệu từ file ra vùng đệm (sử dụng hàm 3Fh).
- Đóng file (sử dụng hàm 3Eh)
- Hiển thị dữ liệu trong vùng đệm (sử dụng hàm 02h).

Nội dung chương trình như sau:

. Model Samll

. Stack 100h

. Data

Filename dB 'C:\ .bat', 0

Baoloi dB 'Khong tim thay file C:\ .bat'0

. Code

Man PROC

MOV AX, @DATA; Trở DS tới đoạn DATA


```

MOV DS, AX
MOV AH, 3Dh ; Hàm mở file
MOV AL, 0
LEA DX, Filename; Trỏ DX tới tên file
INT 21h
JNC Moduoc ; Nếu mở được cờ nhớ không lập
LEA DX, Baoloi ; Báo lỗi nếu không mở được
MOV AH, 09h ; Sử dụng hàm 09h ngắt 21h
INT 21h ; Để hiển thị dòng thông báo
JMP CUOI

```

Moduoc: ; Sử dụng hàm 3Fh để đọc file

```

MOV BX, AX
MOV AX, @DATA
MOV DS, AX
MOV AH, 3Fh ; Hàm đọc file
MOV DX, buffers ; Trỏ DS:DX tới vùng đệm nhận dữ liệu
MOV CX, 100 ; Số bytes cần đọc
INT 21h
MOV AH, 3Eh ; Đóng file lại
INT 21h
MOV CX, 100 ; Số vòng lặp để hiển thị
LEA SI, buffers ; Trỏ tới vùng đệm
MOV AH, 02h ; Hàm để hiển thị ký tự

```

Hienthi:

```

LODSB ; Nạp byte từ đệm vào AL
MOV DL, AL
INT 21h ; Hàm 02 để hiển thị các ký tự trong đệm
LOOP hienthi ; Lặp lại hiển thị nếu chưa hết

```

CUOI:

```

MOV AH, 4Ch ; Quay về DOS
INT 21h

```

Main EndP

END Main

*** Hàm 40h: Viết vào một file hay một thiết bị.**

Chuyển một số cho trước các byte từ một bộ đếm vào một file cho trước.

Vào: AH = 40h

BX = thẻ file

DS:DX = địa chỉ của dữ liệu để viết

CX = số byte được viết

Ra: AX = số byte viết.

mã lỗi nếu cờ nhờ được thiết lập.

Ví dụ:

Hãy đọc Boot record từ ổ cứng của bạn và cất ra file Boot.dat. Biết rằng boot record nằm trong sector 1, mặt 1, xilanh 0.

Vậy với nhiệm vụ ở trên chương trình phải thực hiện những thao tác sau:

- Đọc dữ liệu từ sector ra vùng đệm (sử dụng hàm 02h ngắt 13h).

(xin mời xem trước hàm này ở ngắt đĩa).

- Tạo file mới (sử dụng hàm 3Ch).

- Ghi dữ liệu ra file (sử dụng hàm 40h)

- Đóng file (sử dụng hàm 3Eh).

Nội dung của chương trình như sau:

. Model Samll

. Stack 100h

. Data

Thongbao dB 'Nội dung Boot record như sau, nội dung', 10, 13
dB 'nay được cất ra file'

BOOT.DAT!', 10, 13, 10, 13, '\$'

Nhacno dB 10, 13, 10, ' Khi không boot bạn có thể', 10, 13
dB 'copy lại boot từ file'

BOOT.DAT!', 10, 13, 10, 13, '\$'

Filename dB 'C:\BOOT.DAT', 0

Buffers dB 512 DUP ('?').

. Code

Man PROC

```
MOV AX, @DATA
MOV ES, AX
LEA BX, Buffers
MOV AH, 02h      ; Hàm 02 ngắt 13h để đọc sector.
MOV AL, 01       ; Số sector cần đọc
MOV CH, 00       ; Số xi lanh
MOV CL, 01       ; Số hiệu đầu từ
MOV DH, 01       ; Số hiệu ổ đĩa cứng
INT 13h
MOA AX, @ DATA  ; Hiển thị thông báo
MOV DS, AX
LEA DX, Thongbao
MOV Ah, 09h
INT 21h
MOV CX, 512
LEA SI, Buffers
MOV AH, 02h      ; Hàm hiển thị
Hienthi:
LODSB
MOV DL, AL
INT 21h
LOOP Hienthi
                ; Cắt ra file C:\BOOT.DAT
MOV AH, 3Ch      ; Hàm này để tạo file mới
LEA DX, Filename
MOV CX, 00       ; Thuộc tính đọc/ghi thông thường
INT 21h
MOV BX, AX
MOV AH, 40h      ; Ghi dữ liệu ra file
```

```

LEA DX, Buffers
MOV CX, 512      ; số bytes cần ghi
INT 21h
MOV AX, @DATA
MOV DS, AX
LEA DX, nhacnho
MOV AH, 09h
      INT 21h
MOV AH, 4Ch      ; quay về DOS
INT 21h
Main EndP
END Main

```

2.2. Ngắt BIOS

Trong phần này tập trung nghiên cứu các ngắt BIOS thông dụng nhất, các ngắt của BIOS còn lại sẽ được giới thiệu trong các giáo trình chuyên ngành khác.

2.2.1. Ngắt 10h (Ngắt video, phục vụ các thao tác màn hình)

Ngắt 10h có các hàm cơ bản sau:

* *Hàm 1h*

Thay đổi kích thước con trỏ màn hình bằng cách đặt dòng quét bắt đầu và kết thúc của con trỏ.

Vào : AH = 1h
 CH = dòng quét đầu
 CL = dòng quét cuối

Ra: Không

Chúng ta biết rằng để hiển thị con trỏ trên màn hình thì máy tính phải quét nhiều dòng sáng liên tiếp nhau. Với các màn hình ngày nay thường có 16 dòng (0 ÷ 15). Giá trị của CH, CL trong hàm này để xác định các dòng được bật sáng để biểu diễn con trỏ, thường thì con trỏ được hiển thị là hai dòng sáng 6, 7. Nhưng bạn cũng có thể định nghĩa riêng theo ý của mình.

Ví dụ:

Viết chương trình hợp ngữ để tăng kích thước cực đại con trỏ màn hình.

Nội dung chương trình như sau:

. Model Small
. Stack 100h
. Code

Main PROC

MOV AH, 01
MOV CH, 00 ; Dòng quét đầu là 0.
MOV CL, 15 ; Dòng quét cuối 15 (lớn nhất)
INT 10h
MOV AH, 4Ch; hàm 4C ngắt 21h để quay về DOS
INT 21h

Main EndP

END Main

* Hàm 2h

Hàm này để định vị con trỏ màn hình đến một tọa độ nào đó.

Vào : AH = 2h
 DH = dòng
 BH = trang

Ra: Không

Ví dụ:

Viết chương trình để dịch chuyển con trỏ đến tốc độ 0,0 của màn hình. Nội dung của chương trình như sau:

. Model Small
. Stack 100
. Code

Main PROC

MOV AH, 02h
MOV DH, 00h; dòng 0
MOV DL, 00h; cột 0
MOV BH, 00h; trang 0
INT 10h
MOV AH, 4Ch

INT 21h

Main EndP

END Main

* *Hàm 6h*

Cuốn cả màn hình hay cửa sổ lên một số dòng xác định. Khi cuộn các dòng trong cửa sổ sẽ bị cuộn lên phía trên, các dòng phía dưới cũng sẽ bị bỏ trống.

Vào: AH = 6h

AL = số dòng cuộn (AL = 0 nghĩa là cuộn cả màn hình hay cửa sổ).

BH = thuộc tính của các dòng trống.

CH, CL = dòng, cột góc trái trên của cửa sổ cần cuộn.

DH, DL = dòng, cột góc phải dưới của cửa sổ cần cuộn.

Ra: Không

Ví dụ:

Viết chương trình để cuộn lên trên 3 dòng văn bản trong cửa sổ có toạ độ (hàng, cột) góc trái trên (05, 10) và toạ độ góc phải dưới (15, 50). Nội dung của chương trình như sau:

. Model Small

. Stack 100h

. Code

Main PROC

MOV AH, 06h

MOV AL, 3

MOV DH, 15

MOV DL, 50

MOV CH, 05

MOV CL, 10

MOV BH, 00h

INT 10h

MOV AH, 4Ch

INT 21h

Main EndP

END Main

** Hàm 7h*

Cuốn cả màn hình hay cửa sổ xuống một số dòng xác định.

Vào: AH = 7h

AL = số dòng cuộn (AL=0 nghĩa là cuộn cả màn hình hay cửa sổ).

BH = thuộc tính của các dòng trống.

CH, CL = dòng, cột góc trái trên của cửa sổ cần cuộn.

DH, DL = dòng, cột góc phải dưới của cửa sổ cần cuộn.

Ra: Không

Ví dụ:

Viết chương trình để cuộn xuống dưới 3 dòng văn bản trong cửa sổ có tọa độ (hàng, cột) góc trái trên (05, 10) và tọa độ góc phải dưới (15, 50). Nội dung của chương trình như sau:

. Model Small

. Stack 100h

. Code

Main PROC

MOV AH, 07h

MOV AL, 3

MOV DH, 15

MOV DL, 50

MOV CH, 05

MOV CL, 10

MOV BH, 00h

INT 10h

MOV AH, 4Ch

INT 21h

Main EndP

END Main

2.2.2. Ngắt 13h (Vào ra đĩa)

Ngắt 13h có các hàm cơ bản như sau:

** Hàm 2h: Đọc Sector.*

Đọc một hay nhiều sector.

Vào: AH = 02h
 CH = Xilanh
 CL = Sector
 DH = Đầu từ
 DL = ổ đĩa (0 - 7Fh = đĩa mềm, 80 - FFh = đĩa cứng)
 ES:BX = Địa chỉ segment: offset của bộ đệm

Ra: Nếu thành công

 CF = xoá

 AH = 0

 AL = số sector được đọc

Nếu không thành công:

 CF = thiết lập

 AH = mã lỗi

Ví dụ:

Hiển thị master boot của ổ cứng C ra màn hình, biết master boot này chứa trong sector 1, xilanh 0, mặt 0.

Nội dung của chương trình như sau:

. Model Small

. Stack 100h

. Data

 Thongbao dB 'Noidung Master boot nhu sau:', 10, 13, '\$'

 Buffers dB 512 DUP ('?'); vùng đệm chứa dữ liệu đọc được.

. Code

 Main PROC

 MOV AX, @DATA

 MOV ES, AX

 LEA BX, Buffers

 MOV AH, 02h ; hàm đọc sector

 MOV AL, 01 ; số sector cần đọc

 MOV CH, 00 ; số xi lanh

 MOV CL, 01 ; số hiệu sector bắt đầu

 MOV DH, 00 ; số hiệu đầu từ

MOV DL, 80h	; số hiệu ổ đĩa cứng C
INT 13h	; gọi ngắt
MOV AX, @DATA	; hiển thị thông báo
MOV DS, AX	
MOV DS, thongbao	
MOV AH, 09h	
INT 21h	
MOV CX, 512	; Số byte cần hiển thị
LEA SI, Buffers	; Trỏ tới bộ đệm cần hiển thị
MOV AH, 02h	; Hàm hiển thị

Hienthi:

LODSB	
MOV DL, AL	
INT 21h	
LOOP Hienthi	; Được lặp khi chưa hiển thị hết
MOV AH, 4Ch	; Quay về DOS
INT 21h	

Main EndP

END Main

** Hàm 3h: Ghi Sector.*

Viết một hay nhiều sector.

Vào:

AH = 3h
AL = Số sector
(BX = Thanh ghi màu thứ nhất)
CH = Xi lanh
CL = Sector
DH = Đầu từ
DL = Ổ đĩa (0 - 7Fh = đĩa mềm, 80 - FFh = đĩa cứng)
ES: BX = Địa chỉ segment: offset của bộ đệm

Ra: Nếu thành công:

CF = xoá
AH = 0

AL = số sector được viết

Nếu không thành công:

CF = thiết lập

AH = mã lỗi

Ví dụ:

Viết chương trình để copy nội dung của file boot.dat vào boot record trên ổ cứng. Biết boot record chứa trong sector 1, đầu từ 1, xilanh 0.

Nội dung của chương trình như sau:

. Model Small

. Stack 100h

. Data

Thongbao dB 'CHUONG TRINH KHOI PHUC BOOT', 10, 13

dB 'Boot rec duoc khoi phuc tu C:\boot.dat :', 10, 13, '\$'

Filename dB 'C:\boot.dat', 0

Baoloi dB 'Khong tim thay file C:\boot.dat ! \$'

Buffers dB 512 DUP ('?')

. Code

Main PROC

MOV AX, @DATA ; Hiển thị thông báo

MOV DS, AX

LEA DX, Thongbao

MOV Ah, 09h

INT 21h

; Mở file

MOV AH, 3Dh ; Mở file để đọc

MOV AL, 0

MOV DX, Filename

INT 21h

JNC Moduoc ; Mở được file

LEA DX, Baoloi ; Không mở được báo lỗi

MOV AH, 09h

INT 21h

JMP CUOI

Moduoc:

```
MOV BX, AX
MOV AX, @DATA
MOV DS, AX
MOV AH, 3Fh      ; Đọc dữ liệu từ file ra đệm buffers
MOV DX, buffers
MOV CX, 512      ; Số byte cần đọc
INT 21h
MOV AH, 3Eh      ; Dòng file
INT 21h
MOV AX, @DATA
MOV ES, AX
LEA BX, buffers
MOV AH, 03h      ; Hàm cất ra boot
MOV AL, 01       ; Số sector cần ghi
MOV CH, 00       ; Số xi lanh
MOV CL, 01       ; Số hiệu sector bắt đầu
MOV DH, 01       ; Số hiệu đầu từ
MOV DL, 80h      ; Số hiệu ổ đĩa C
INT 13h
```

CUOI:

```
MOV AH, 4Ch      ; Quay về DOS
INT 21h
```

Main EndP

END Main

2.2.3. Ngắt 16H (các thao tác với bàn phím)

Ngắt 16H có các hàm cơ bản như sau:

* *Hàm 0h: Đọc ký tự từ bàn phím.*

Vào: AH = 0h

Ra: AH = Mã Scan của ký tự đọc được
(xem khái niệm mã scan ở phần sau)

AL = Mã ASCII của ký tự đọc được

Ví dụ:

Hãy vào một ký tự từ bàn phím. Nếu ký tự đó là 'A', thì hiển thị 'Ban da chon chuc nang A!'. Nếu ký tự đó là 'B', thì hiển thị 'Ban da chon chuc nang B!'. Nếu không hiển thị 'Ban da chon khac chuc nang A, B!'.

Nội dung chương trình như sau:

. Model Small

. Stack 100h

. Data

Request dB 'Hay chon chuc nang (A/B)? \$'

Message1 dB 'Ban da chon chuc nang A ! \$'

Message2 dB 'Ban da chon chuc nang B ! \$'

Message3 dB 'Ban da chon chuc nang khac A,B ! \$'

. Code

Main PROC

MOV AX, @DATA

MOV DS, AX

MOV AH, 09h ; Hiển thị yêu cầu

LEA DX, Request

INT 21h

MOV AH, 00h ; Hàm để nhập một ký tự

INT 16h ; Gọi ngắt

CMP AL, 'A' ; So sánh với 'A'

JNZ KTB ; Nếu không bằng nhảy tới ký từ B, KTB

LEA DX, Message1; Nếu bằng trở DX đến dòng Message1

JMP CUOI ; Nhảy về CUOI

KTB:

CMP AL, 'B' ; So sánh với 'B'

JNZ NOAB ; Nếu không bằng nhảy đến NOAB

LEA DX, Message2; Nếu bằng B, trở DX đến dòng Massage2

JMP CUOI

NOAB:

LEA DX, Message3 ; Trỏ DX tới dòng Message3

CUOI:

MOV AX, @DATA

MOV DS, AX

MOV AH, 09h ; Hiển thị dòng nhắc mà DX đang trỏ đến

INT 21h

MOV AH, 4Ch ; Quay về DOS

INT 21h

Main EndP

END Main

** Hàm 2h: Lấy các cờ bàn phím.*

Vào: AH = 2h

Ra: AL = các cờ

Các bit cờ trong thanh ghi AL có ý nghĩa như sau:

Bit Nếu được thiết lập

7 Insert on

6 Caps Lock on

5 Num Lock on

4 Scroll Lock on

3 Phím Alt được nhấn

2 Phím Ctrl được nhấn

1 Phím Shift trái được nhấn

0 Phím Shift phải được nhấn

Ví dụ:

Hãy hiển thị trạng thái đèn CAPSLOCK. Nếu đèn này đang được bật hãy thông báo 'Đèn CAPSLOCK đang được bật', nếu đèn này không được bật hiển thị! 'Đèn CAPSLOCK không được bật!'. (Việc bật tắt phụ thuộc việc ấn phím CAPSLOCK trên bàn phím).

Nội dung chương trình như sau:

. Model Small

. Stack 100h

. Data

Thongbao dB 'Ban co the bat den CAPSLOCK de kiem tra! \$'

BatCapsLock dB 10, 13, 'Den CAPSLOCK dang duoc bat! \$'

TatCapsLock dB 10, 13, 'Den CAPSLOCK khong duoc bat! \$'

. Code

Main PROC

MOV AX, @DATA ; Hiển thị thông báo ban đầu

MOV DS, AX

MOV AH, 09h

LEA DX, Thongbao

INT 21h

MOV AH, 00h ; Chờ ấn phím

INT 16h

MOV AH, 02h ; Lấy cờ bàn phím vào AL

INT 16h

AND AL, 01000000b ; Xoá các bit cờ khác ngoại trừ bit cờ biểu diễn
; trạng thái CAPS LOCK

JZ TAT ; Nếu bằng 0, chứng tỏ bit cờ CAPSLOCK = 0

LEA DX, Batcapslock; Trỏ DX đến thông báo CAPSLOCK

; không được bật khi bit CAPSLOCK = 0

JMP CUOI

TAT:

LEA DX, TatCapsLock ; Trỏ DX đến thông báo CapsLock không được
; bật khi bit CAPSLOCK = 0

CUOI:

MOV AX, @DATA

MOV DS, AX

MOV AH, 09h ; Hiển thị thông báo

INT 21h

MOV AH, 4Ch ; Quay về DOS

INT 21h

Main EndP

END Main

Câu hỏi ôn tập

1. Trình bày cấu trúc của chương trình .EXE và .COM, giải thích cấu trúc đó, và so sánh ưu nhược điểm giữa hai loại cấu trúc.
2. Trình bày các bước tạo và chạy một chương trình hợp ngữ.
3. Chế độ địa chỉ là gì? Hãy trình bày các chế độ địa chỉ của bộ vi xử lý 8088.
4. Bộ vi xử lý 8088 có những loại lệnh nào? Hãy lấy một ví dụ lệnh cho mỗi nhóm lệnh.
5. Ngắt là gì? Ngắt để làm gì? Có bao nhiêu loại ngắt? Mỗi loại ngắt hãy lấy một ví dụ minh họa.

Chương 5

BỘ NHỚ

Mục tiêu:

Giới thiệu đặc điểm, cấu trúc và các loại bộ nhớ bán dẫn được sử dụng trong bộ vi xử lý và máy tính để học sinh có được những kiến thức cơ bản nhất về bộ nhớ.

Giới thiệu cách giải mã địa chỉ cho bộ nhớ, cách ghép nối bộ vi xử lý với bộ nhớ, cách ghép nối mở rộng bộ nhớ để học sinh có được kiến thức để có thể phân tích, sửa chữa và thực hiện ghép nối hệ thống.

Nội dung chính:

I. Khái quát chung về bộ nhớ

1. Tổ chức bộ nhớ
2. Quá trình đọc và ghi bộ nhớ
3. Phân loại bộ nhớ

II. Bộ nhớ ROM

1. Đặc điểm
2. Cấu trúc chung
3. Các loại bộ nhớ ROM

III. Bộ nhớ RAM

1. Đặc điểm
2. Cấu trúc chung
3. Bộ nhớ SRAM
4. Bộ nhớ DRAM
5. Các bộ nhớ DRAM tiên tiến

VI. Ghép nối bộ vi xử lý 8088 với bộ nhớ

1. Giải mã địa chỉ cho bộ nhớ
2. Ghép nối bộ vi xử lý 8088 với bộ nhớ
3. Mở rộng bộ nhớ

I. KHÁI QUÁT CHUNG VỀ BỘ NHỚ

1. Tổ chức bộ nhớ

Tổ chức các bộ nhớ được tuân theo nguyên tắc: Các phần tử nhớ cơ bản (nhỏ nhất) được mắc nối tiếp với nhau thành các dây nhớ được gọi là thanh ghi. Tiếp theo các thanh ghi lại được ghép với nhau tạo thành một ma trận nhớ, ma trận nhớ này muốn hoạt động được thì phải có các mạch điện phụ hỗ trợ cho nó và khi đó hình thành một bộ nhớ. Sau đây ta đi nghiên cứu các phần cơ bản này.

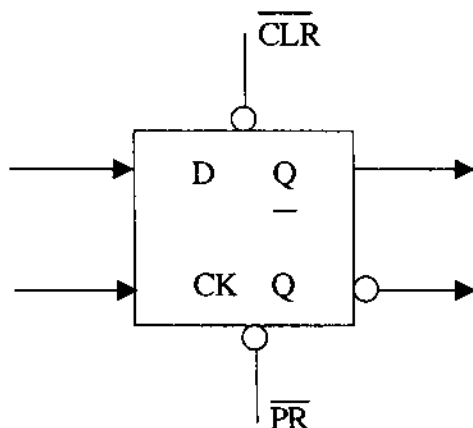
1.1. Mạch lật (F - F)

- Chức năng

Mạch lật là một mạch vi điện tử, đầu ra có 2 trạng thái ổn định tương ứng biểu diễn mức logic 0 hoặc mức logic 1. Trạng thái của mạch lật không thay đổi nếu nguồn nuôi không bị ngắt hay không có tín hiệu tác động từ bên ngoài. Loại mạch lật đơn giản nhất là loại mạch lật D - FF.

- Trạng thái của mạch lật D - FF

Trạng thái của mạch này được thể hiện bằng hàm logic sau đây:



Hình 5.1: Ký hiệu mạch lật kiểu D - FF

$$Q_{i+1} = D_i$$

Có nghĩa là dữ liệu đầu ra sẽ bằng đầu vào D khi có xung đồng hồ CK tác động (loại đồng bộ).

Ngoài ra các chân CLR, PR là các chân xoá và thiết lập 1 đầu ra.

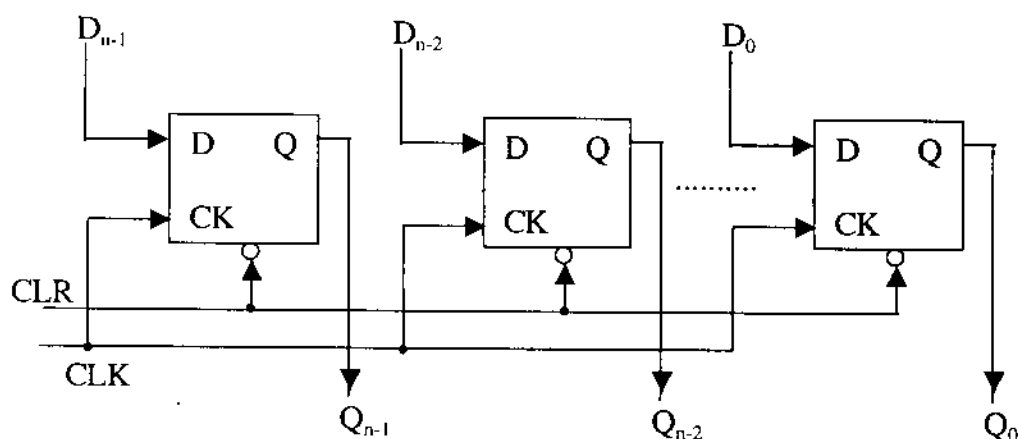
Bảng trạng thái hoạt động của mạch lật D - FF được thể hiện như bảng sau:

D_i	CK	$\overline{PR}$	$\overline{CLR}$	Q_{i+1}
X	X	0	1	1
X	X	1	0	0
X	0	1	1	D_i-1
1	1	1	1	1
0	1	1	1	0

Bảng 5.1: Bảng trạng thái hoạt động của DFF

1.2. Thanh ghi n bit

Là tập hợp của n ô nhớ được nối với nhau, mỗi ô nhớ có thể lưu trữ 1 bit thông tin (0,1). Thường các ô nhớ này được ghép song song với nhau.



Hình 5.2: Cấu trúc thanh ghi n bit sử dụng D - FF

1.3. Vi mạch nhớ

- Cấu trúc chung của một vi mạch nhớ

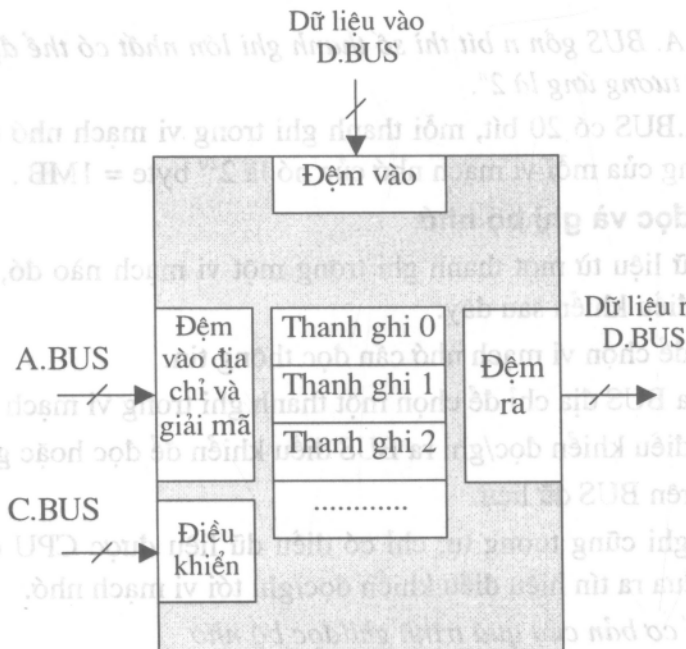
Chúng ta có thể hiểu một vi mạch nhớ là tập hợp nhiều thanh ghi. Nhưng các thanh ghi này phải được tổ chức sao cho chúng có các địa chỉ tách biệt và thời gian truy nhập đến chúng là nhanh nhất. Hình 5.3 mô tả cấu trúc chung của một vi mạch nhớ.

- Ý nghĩa các bộ phận

- A.BUS, (Address BUS): Đây là tập hợp của các đường dây để đưa tín hiệu vào vi mạch nhớ nhằm xác định địa chỉ thanh ghi cần truy nhập trong mạch nhớ đó.

- C.BUS, (Control BUS): Đây là tập hợp các đường dây để đưa các tín hiệu điều khiển đọc/ghi vào vi mạch nhớ.

- D.BUS, (Data BUS): Đây là tập hợp các đường dây để lưu chuyển dữ liệu giữa vi mạch nhớ và thế giới bên ngoài.



Hình 5.3: Sơ đồ khối cấu trúc của một vi mạch nhớ cơ bản

Chú ý: Trên các đường dây của A.BUS, C.BUS, D.BUS thông tin được truyền dưới dạng nhị phân (0,1) tương ứng với hai mức điện áp khác nhau.

- Các thanh ghi: Là nơi lưu trữ thông tin, số lượng các thanh ghi quyết định dung lượng của bộ nhớ. Mỗi thanh ghi trong vi mạch nhớ đều được xác định tại một địa chỉ riêng.

- Các đệm số liệu: Bao gồm có đệm vào và đệm ra, thông qua chúng dữ liệu có thể được ghi vào các thanh ghi (qua đệm vào) hoặc được đọc ra từ thanh ghi (qua đệm ra).

- Phần logic điều khiển: Đây là bộ phận không thể thiếu đối với các vi mạch nhớ, bộ phận này nhận các tín hiệu điều khiển từ bộ vi xử lý để điều khiển hoặc đọc/ghi của bộ nhớ.

- Đệm vào địa chỉ và giải mã: Tuy trong một vi mạch nhớ thường có rất nhiều thanh ghi, nhưng về nguyên tắc tại một thời điểm chỉ có thể truy nhập

(đọc/ghi) đến một thanh ghi nào đó mà địa chỉ của nó đã được xác định trên A. BUS. Trong vi mạch nhớ bao giờ cũng dành sẵn phần mạch logic để giải quyết vấn đề này đó là bộ phận đệm vào địa chỉ và giải mã.

Ví dụ A.BUS có 3 đường dây địa chỉ thì có chỉ có thể xác định $2^3 = 8$ thanh ghi khác nhau.

Tổng quát với A. BUS gồm n bit thì số thanh ghi lớn nhất có thể địa chỉ hoá trong vi mạch nhớ tương ứng là 2^n .

Ví dụ: Với A.BUS có 20 bit, mỗi thanh ghi trong vi mạch nhớ có 8 bit(1 byte) thì dung lượng của mỗi vi mạch nhớ của nó là 2^{20} byte = 1MB .

2. Quá trình đọc và ghi bộ nhớ

Khi cần đọc dữ liệu từ một thanh ghi trong một vi mạch nào đó, CPU cần có những tín hiệu điều khiển sau đây:

- Đưa tín hiệu để chọn vi mạch nhớ cần đọc thông tin.
- Đưa địa chỉ ra BUS địa chỉ để chọn một thanh ghi trong vi mạch nhớ đó.
- Đưa tín hiệu điều khiển đọc/ghi ra BUS điều khiển để đọc hoặc ghi.
- Đọc dữ liệu trên BUS dữ liệu.

Với quá trình ghi cũng tương tự, chỉ có điều dữ liệu được CPU đưa ra D. BUS trước khi nó đưa ra tín hiệu điều khiển đọc/ghi tới vi mạch nhớ.

** Các thông số cơ bản của quá trình ghi/đọc bộ nhớ*

- Thời gian địa chỉ hoá: Là thời gian kể từ lúc CPU đưa ra tín hiệu địa chỉ cho đến khi tín hiệu địa chỉ đã ổn định trên chân vi mạch.
- Thời gian ghi: Là thời gian kể từ lúc CPU phát tín hiệu điều khiển ghi tới vi mạch nhớ đến khi dữ liệu đã được ghi xong.
- Thời gian đọc: Là thời gian kể từ lúc CPU phát tín hiệu điều khiển đọc tới vi mạch nhớ đến khi dữ liệu đưa ra ngoài trên D. BUS.

- Thời gian thâm nhập: Đây là khoảng thời gian từ lúc CPU đưa ra địa chỉ đến khi hoàn thành việc đọc hoặc ghi lại dữ liệu trên D. BUS.

Dung lượng: Biểu diễn lượng thông tin của vi mạch nhớ có thể lưu trữ, phụ thuộc vào số lượng thanh ghi và độ dài các thanh ghi trong vi mạch nhớ.

3. Phân loại bộ nhớ

Thông thường ta có các cách phân loại bộ nhớ như sau:

** Phân loại bộ nhớ theo cách truy cập*

Phân loại bộ nhớ theo cách truy cập bộ nhớ, ta có các loại bộ nhớ sau:

- Bộ nhớ truy cập ngẫu nhiên: Đây là một loại bộ nhớ mà khi ta muốn truy cập đến một phần tử bất kỳ của bộ nhớ ta không cần phải lướt qua tất cả các phần tử nhớ đứng trước nó. Chính vì vậy mà thời gian truy cập đến các phần tử trong trường hợp này không phụ thuộc vào vị trí của các phần tử nhớ.

- Bộ nhớ truy cập tuần tự: Đây là loại bộ nhớ mà khi chúng ta muốn truy cập đến một phần tử bất kỳ trong bộ nhớ thì ta phải lướt qua tất cả các phần tử nhớ trước đó, ví dụ như một bộ nhớ băng từ.

** Phân loại theo khả năng ghi / đọc của bộ nhớ*

Tuỳ theo chức năng mà các bộ nhớ có thể có những khả năng ghi đọc thông tin khác nhau:

- Có những loại bộ nhớ chỉ có thể đọc thông tin từ chúng mà không thể ghi thông tin ra chúng (Read Only) thường gọi là ROM (Read Only Memory).

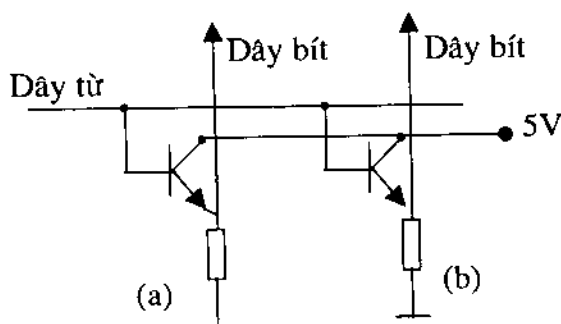
- Có những loại bộ nhớ vừa có thể đọc thông tin lại vừa có thể ghi thông tin ra chúng, thường gọi là RAM (Random Access Memory).

II. BỘ NHỚ ROM

1. Đặc điểm

- Đây là loại bộ nhớ mà các phần nhớ của nó có trạng thái cố định, vì vậy thông tin lưu giữ trong ROM cũng cố định và thậm chí không bị mất ngay cả khi mất điện. Việc ghi thông tin vào ROM cũng là việc thiết lập trạng thái cố định cho các phần tử nhớ, được thực hiện một lần tại nơi sản xuất. Do đó, người sử dụng không thể thay đổi những thông tin đã được ghi trong ROM.

Chính vì đặc điểm trên mà các phần tử nhớ trong ROM thường đơn giản hơn nhiều so với mạch lật, chúng chỉ có nhiệm vụ lưu giữ một trạng thái cố định 0 hoặc 1. Hình sau biểu diễn cấu tạo các phần tử ROM tiêu biểu.



Hình 5.4: Cấu tạo bit nhớ của bộ nhớ ROM

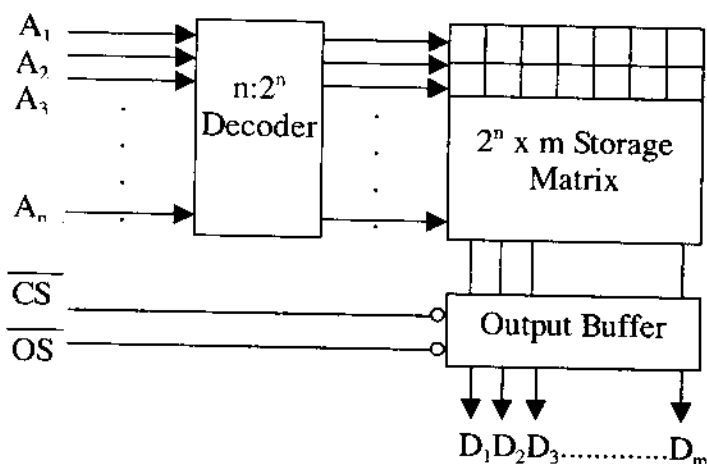
Với hình trên chúng ta thấy khi cho điện áp đủ lớn vào đường dây từ thì đầu ra của phần tử nhớ a (dây bít của a) luôn có điện áp cao biểu diễn mức 1, còn phần tử nhớ b do tranzitor của nó hở cực Emitter, nên điện áp ra (dây bít của b) luôn biểu diễn mức 0. Chúng ta thấy trạng thái của cả a và b sẽ luôn cố định ngay cả khi mất điện.

Chức năng của ROM để lưu giữ chương trình kiểm tra hệ thống khi khởi động (Post) và các vi chương trình điều khiển phần cứng.

2. Cấu trúc chung

Cấu trúc chung của các vi mạch ROM được biểu diễn trên hình sau:

Trong đó:



Hình 5.5: Cấu trúc chung của vi mạch nhớ ROM

- $A_1, A_2, \dots, A_n$ là các đường dây địa chỉ.
- $D_1, D_2, \dots, D_m$ là các đường dây dữ liệu.
- $\overline{CS}, \overline{OS}$: Là các đường dây nhận tín hiệu điều khiển. Trong đó $\overline{OS}$ (Output Select) để nhận tín hiệu chọn đầu ra. $\overline{CS}$ (Chip Select) để nhận tín hiệu chọn vi mạch.
- Bộ giải mã $n:2^n$ để chọn thanh ghi tương ứng m bít.
- Ma trận nhớ để lưu trữ các phần tử nhớ. Các phần tử nhớ được tổ chức thành 2^n thanh ghi, mỗi thanh ghi có độ dài m bít.
- Bộ đệm ra để đưa dữ liệu ra bên ngoài khi nhận được các tín hiệu điều khiển tương ứng $\overline{OS}, \overline{CS}$.

Nguyên lý hoạt động

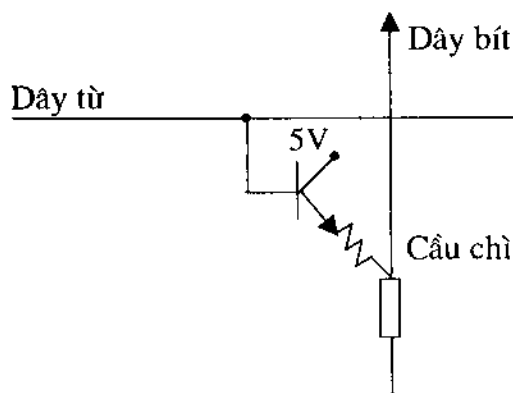
Để đọc thông tin trong vi mạch ROM, đầu tiên CPU đưa địa chỉ thanh ghi cần đọc ra A.BUS. Khi đó mạch giải mã chọn được thanh ghi có địa chỉ tương ứng và đưa dữ liệu trong đó ra bộ đệm ra.

Tiếp theo khi CPU thiết lập mức logic tích cực cho các tín hiệu $\overline{OS} = 0$, $\overline{CS} = 0$ thì bộ đệm được mở và dữ liệu sẽ xuất hiện trên các đường dây dữ liệu.

3. Các loại ROM

Một nhược điểm đối với bộ nhớ ROM đó là không thể ghi vào đó những thông tin chúng ta muốn, mà điều này phụ thuộc hoàn toàn vào nhà sản xuất. Để khắc phục nhược điểm này, tồn tại một số dạng khác của ROM sau đây:

- **PROM:** Đây là loại bộ nhớ mà khi sản xuất các phân tử nhớ của chúng được ghi cùng một giá trị 1 hoặc 0. Sau đó để ghi thông tin mong muốn, người ta chỉ việc dùng chương trình làm thay đổi trạng thái của các phân tử nhớ cần thiết và việc này thường được thực hiện bằng cách cho dòng điện đủ lớn chạy qua các ô nhớ này để làm các cầu chì đặc biệt tại các ô nhớ đó bị đứt.



Hình 5.6: Cấu trúc một bit nhớ của bộ nhớ PROM

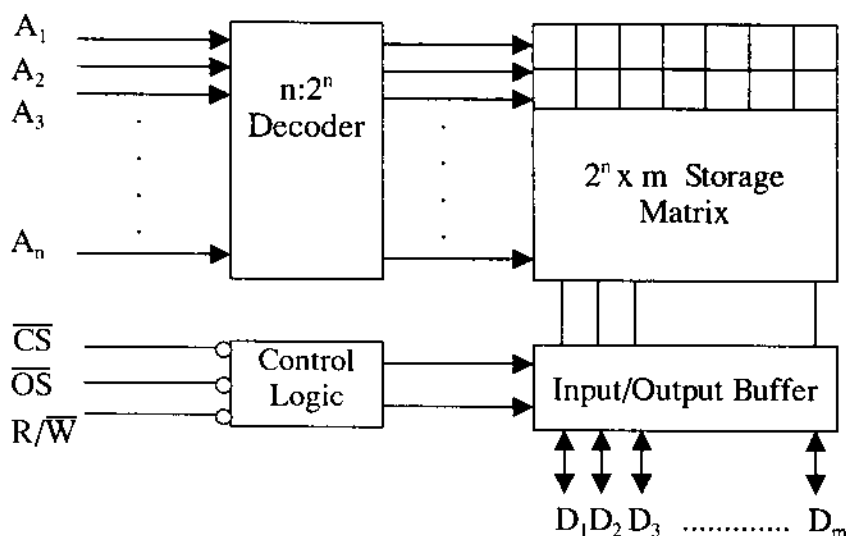
- **EPROM:** Với loại bộ nhớ PROM thì chúng ta chỉ có thể ghi những thông tin mong muốn 1 lần. Để khắc phục tình trạng này thì EPROM cho phép chúng ta ghi thông tin nhiều lần, tuy nhiên trước khi ghi những thông tin mới thì toàn bộ nội dung cũ của EPROM phải được xóa bằng tia cực tím. Chính vì vậy, mặt trên của các vi mạch loại này bao giờ cũng có một cửa sổ trong suốt cho phép tia sáng truyền qua.

- **EEPROM:** Chức năng của loại bộ nhớ này cũng tương tự như EPROM nhưng có điều thuận tiện hơn đó là nội dung của các ô nhớ có thể được xóa

bằng tín hiệu điện. Tuy nhiên một điều khó khăn đặt ra đó là phải sử dụng nhiều mức điện áp khác nhau để đọc/ghi thông tin.

Do EPROM và EEPROM có thể ghi thông tin, nên về cấu trúc của chúng có những điểm khác so với ROM, (Hình 5.7)

Qua hình 5.7 chúng ta thấy bộ đệm dữ liệu ở đây có thể là bộ đệm ghi vào (ghi dữ liệu), hoặc là bộ đệm ra (đọc dữ liệu). Ngoài ra, chúng ta còn thấy xuất hiện đường dây nhận tín hiệu điều khiển $R/\bar{W}$ để quyết định thao tác đọc hay ghi bộ nhớ.



Hình 5.7: Cấu trúc của bộ nhớ EPROM và EEPROM

III. BỘ NHỚ RAM

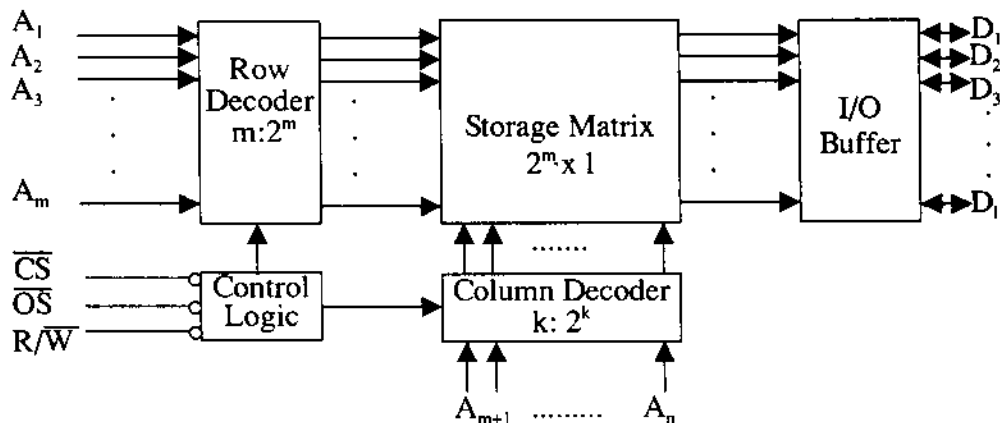
1. Đặc điểm

- Đây là loại bộ nhớ mà thông tin trong đó có thể thay đổi được và những thông tin này bị mất khi mất điện. Vì vậy, có thể nói đây là loại bộ nhớ có thể ghi/đọc (Random Access Memory).

- Hiện nay có 2 loại RAM được sử dụng rộng rãi đó là RAM tĩnh (SRAM - Static RAM) và RAM động (DRAM - Dynamic RAM).

2. Cấu trúc chung

Các bộ nhớ SRAM và DRAM đều có một sơ đồ cấu trúc chung như sau:



Hình 5.8: Sơ đồ khối cấu trúc bộ nhớ RAM

Trong đó:

- $A_1 \dots A_m$ là m đầu vào địa chỉ hàng để xác định 2^m thanh ghi trong bộ nhớ.
- Row Decoder là bộ giải mã hàng với m đầu vào và 2^m đầu ra.
- $\overline{CS}$, $\overline{OS}$, $R/\overline{W}$ là các tín hiệu điều khiển chọn vi mạch, chọn bộ đếm ra và điều khiển đọc/ghi.
- Control logic là bộ giải mã, đếm các tín hiệu điều khiển.
- Storage Matrix là ma trận nhớ gồm có 2^m thanh ghi theo hàng và mỗi thanh ghi có 1 bit nhớ.
- $A_{m+1} \dots A_n$ gồm k đường dây địa chỉ cột.
- Column Decoder là bộ giải mã địa chỉ cột với k đầu vào và 2^k đầu ra.
- I/O Buffer là bộ đệm vào/ra để lưu trữ tạm thời dữ liệu khi ghi/đọc.
- $D_1 \dots D_l$ gồm l bit dữ liệu được ghi/đọc vào/ra.

3. Bộ nhớ SRAM

RAM tĩnh là loại RAM dùng công nghệ TTL để thiết kế các phần tử nhớ của chúng. Các phần tử nhớ của nó thường được cấu trúc như dạng các mạch lật F - F. Chúng ta có thể thấy cấu trúc chung của một vi mạch SRAM được mô tả ở sơ đồ trên (Sơ đồ cấu trúc chung).

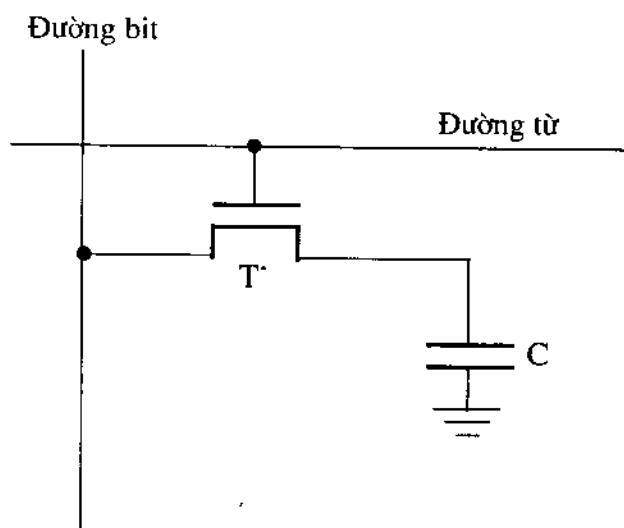
Qua hình trên chúng ta thấy về cấu trúc của SRAM cũng tương tự như EPROM, chỉ điều khác ở đây là có hai bộ giải mã địa chỉ hàng và cột. Như vậy để có thể chọn ra các bit của một thanh ghi trong ma trận nhớ khi ghi hoặc đọc thông tin phải qua hai bước giải mã. Bước giải mã hàng để xác định thanh ghi

cần thao tác nằm trên hàng nào trong ma trận nhớ, bước giải mã cột xác định thanh ghi đó nằm ở trên cột nào trong hàng đã chọn. Việc giải mã hai bước ở đây để tăng tốc độ và đơn giản hoá mạch giải mã. Nếu chỉ sử dụng một bộ giải mã thì nó rất phức tạp vì thường dung lượng của RAM lớn hơn rất nhiều so với bộ nhớ ROM.

Với cấu trúc như vậy, SRAM có đặc điểm là có tốc độ truy nhập rất nhanh. Tuy nhiên giá thành của SRAM khá cao và dung lượng bị hạn chế do cấu trúc của các phần tử khá cồng kềnh. Do đó trong máy tính chỉ sử dụng SRAM những nơi cần xử lý tốc độ cao. Thường là các bộ nhớ đệm nhanh (Cache).

4. Bộ nhớ DRAM

RAM động được thiết kế trên công nghệ C - MOS. Mỗi phần tử nhớ của nó được thiết kế dựa trên khả năng nạp điện của tụ điện và đặc điểm trở kháng của Tranzitor trường. Hình vẽ sau biểu diễn cấu trúc cơ bản một phần tử nhớ của DRAM.

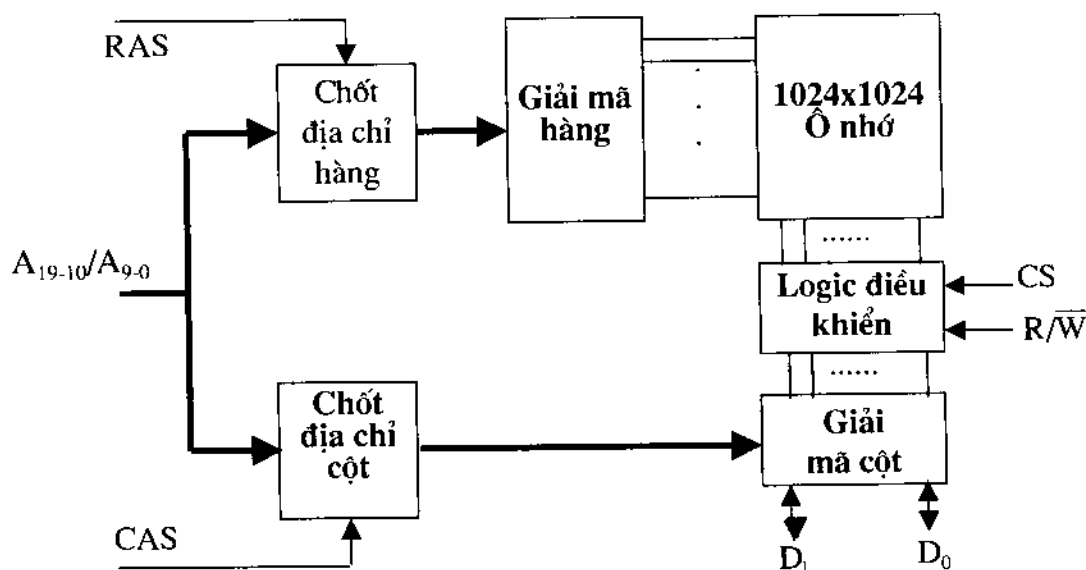


Hình 5.9: Cấu trúc một phần tử nhớ của bộ nhớ DRAM

Với cấu trúc trên, khi muốn ghi giá trị 1 vào phần tử nhớ này chỉ cần nâng cao điện áp các đường dây bit và từ thích hợp làm Tranzitor thông và tụ điện được nạp, khi đó phần tử nhớ này đã được ghi giá trị 1. Khi Tranzitor không mở thì trở kháng của nó rất lớn và tụ điện khó có thể phóng điện qua nó được và như vậy tụ điện vẫn giữ giá trị 1 (do điện áp cao). Tuy nhiên trở kháng của Tranzitor trường cũng không phải là ∞ , nên vẫn còn hiện tượng phóng điện của

tụ điện. Để đảm bảo thông tin không bị mất mát do tụ điện phóng điện, đi kèm với các vi mạch DRAM thường có các mạch làm tươi, nhiệm vụ của các mạch này là luôn hồi phục những thông tin đã lưu sau những chu kỳ nhất định.

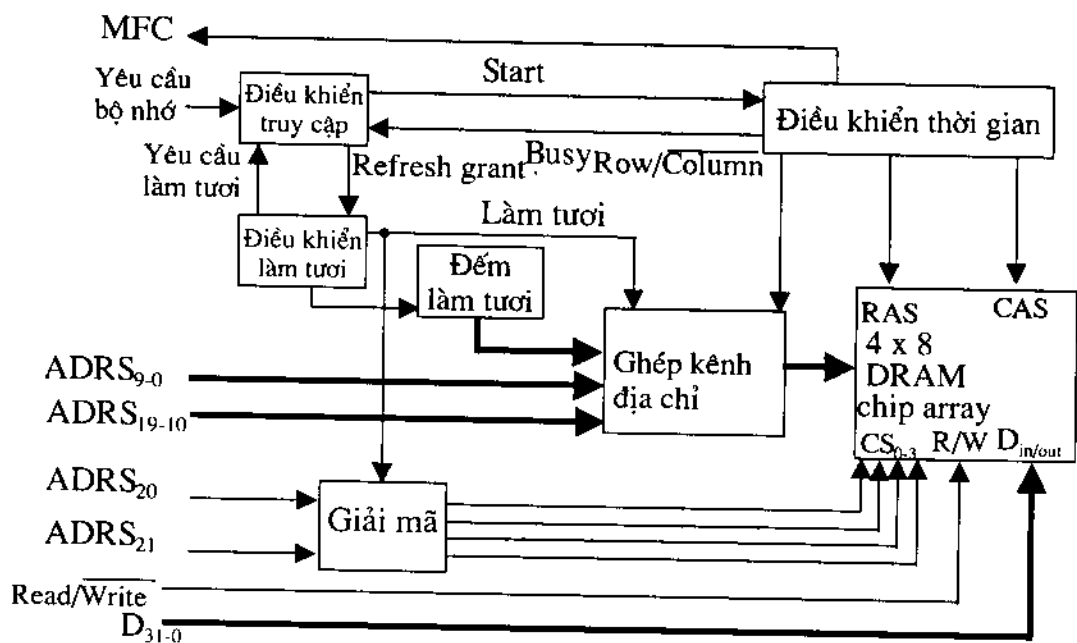
Do cấu trúc đơn giản của các phần tử mà DRAM thường có dung lượng rất lớn, điều này làm số đường dây địa chỉ phải rất nhiều. Để hạn chế số lượng chân địa chỉ của các vi mạch DRAM, thường người ta thiết kế các mạch logic để đưa các tín hiệu địa chỉ làm nhiều lần qua một số ít các chân địa chỉ. Chúng ta có thể hiểu rõ hơn trong hình sau miêu tả cấu trúc của một vi mạch RAM dung lượng 1Mx1.



Hình 5.10: Cấu trúc của vi mạch DRAM 1Mx1

Về mặt cấu trúc chúng ta thấy cũng tương tự SRAM, chỉ có điều các đường dây địa chỉ hàng và cột là chung nhau. Nhưng thường địa chỉ hàng được đưa vào chốt địa chỉ hàng trước nhờ tín hiệu RAS, tiếp theo đó địa chỉ cột được đưa vào chốt địa chỉ cột nhờ tín hiệu mở chốt CAS. Khi đã nhận đủ các tín hiệu địa chỉ thì hoạt động của DRAM cũng tương tự như SRAM.

Như phần trên đã nói đi kèm với các vi mạch DRAM thường phải có các mạch làm tươi. Chúng ta hãy xét sơ đồ khối của một Module nhớ DRAM dung lượng 4M x 32 được xây dựng từ các vi mạch DRAM dung lượng 1Mx4 như sau:



Hình 5.11: Bộ nhớ DRAM có điều khiển làm tươi bộ nhớ

Cách tổ chức các vi mạch nhớ để được một Module nhớ có kích thước mong muốn chúng ta sẽ có điều kiện xét ở phần sau, ở đây chúng ta chỉ quan tâm đến mạch làm tươi của DRAM hoạt động như thế nào.

Với cấu trúc như trên việc truy nhập đến các thanh ghi trong DRAM được thực hiện như sau:

Đầu tiên CPU đưa tín hiệu địa chỉ và các tín hiệu điều khiển đọc/ghi (Read/Write) và yêu cầu bộ nhớ (Memory request) với DRAM. Khi khối điều khiển truy nhập (Access Control) nhận được yêu cầu đó sẽ thiết lập tín hiệu Start lên 1. Khối điều khiển thời gian (Timing Control) ngay lập tức thiết lập đường dây báo bận (Busy) lên mức tích cực để cấm chu kỳ truy nhập bộ nhớ tiếp theo khi chu kỳ truy nhập bộ nhớ hiện hành chưa kết thúc. Sau đó khối điều khiển thời gian nạp các địa chỉ hàng và cột vào các vi mạch DRAM bằng cách đưa ra các tín hiệu cho phép chọn hàng, cột: RAS, CAS. Khi đã có các tín hiệu địa chỉ và các tín hiệu điều khiển đọc/ghi được đưa vào các DRAM thì sự truy nhập đến thanh ghi xác định nào đó sẽ được hoàn thành và những thông tin dữ liệu sẽ được giao tiếp qua các đường dữ liệu $D_0 - D_{31}$. Khi đó khối điều khiển thời gian sẽ đưa ra tín hiệu báo dữ liệu đã sẵn sàng thông qua đường dây MFC, kết thúc một chu kỳ bộ nhớ tín hiệu báo bận trên đường dây Busy được khử hoạt.

Hoạt động làm tươi bộ nhớ DRAM

Ở đây khối điều khiển làm tươi (Refresh Control) đều đặn phát ra những tín hiệu yêu cầu làm tươi (Refresh Control) theo kỳ tới khối điều khiển truy nhập. Khi đó khối điều khiển truy nhập hoạt động giống như một chu kỳ truy nhập bộ nhớ thông thường và chính những chu kỳ này để làm tươi bộ nhớ. Khối điều khiển truy nhập sẽ trả lời chấp nhận thao tác làm tươi thông qua đường dây Refresh Grant. Trong trường hợp khi có yêu cầu truy nhập bộ nhớ và yêu cầu làm tươi bộ nhớ cùng tác động đến khối điều khiển truy nhập thì khối này sẽ ưu tiên yêu cầu làm tươi thực hiện trước, để đảm bảo không bị mất thông tin trước khi truy nhập.

Khi khối điều khiển làm tươi nhận được tín hiệu chấp nhận từ khối điều khiển truy nhập nó sẽ đưa tín hiệu điều khiển tới khối tổ hợp địa chỉ (A. Multiplexer) thông qua đường dây Refresh, khi nhận được tín hiệu này khối tổ hợp sẽ lấy địa chỉ các thanh ghi thao tác từ khối bộ đếm làm tươi (Refresh Counter) chứ không lấy từ các chân địa chỉ thông thường và khi đó bộ đếm sẽ lần lượt đưa ra các địa chỉ của các hàng, cột để làm tươi toàn bộ các thanh ghi. Một vấn đề đặt ra là trong thời gian làm tươi phải đảm bảo tín hiệu điều khiển R/W không ở mức tích cực ghi vì nếu điều này xảy ra sẽ có hiện tượng thông tin không mong muốn sẽ ghi đè lên thông tin đang cần làm tươi.

Sử dụng RAM trong máy tính

Bất cứ máy tính nào cũng không thể thiếu RAM. Đa số các chương trình muốn chạy được đều phải được nạp vào RAM, kết quả xử lý thông tin của các chương trình cũng được cất ở RAM.

Nếu máy tính có dung lượng RAM càng lớn thì tốc độ xử lý thông tin càng nhanh. Trong máy tính, bộ nhớ chính (Main Memory) thường được sử dụng là DRAM do các vi mạch DRAM có dung lượng lớn, giá thành hạ, còn SRAM thường được sử dụng như những bộ đệm nhanh trong quá trình xử lý vì tốc độ của nó cao nhưng hạn chế về dung lượng và giá thành rất cao.

5. Các bộ nhớ DRAM tiên tiến

Việc trao đổi dữ liệu giữa bộ vi xử lý và bộ nhớ chính (RAM) có tốc độ không cao vì tốc độ truy cập của bộ nhớ chính thấp, mặc dù tốc độ của bộ vi xử lý là rất cao nên dẫn tới bộ vi xử lý phải đợi. Để khắc phục nhược điểm này một loạt các bộ nhớ tiên tiến ra đời nhằm thay thế cho bộ nhớ RAM truyền thống hoặc làm bộ nhớ đệm để ghép nối giữa bộ nhớ chính và bộ vi xử lý nhằm

nâng cao hiệu quả trao đổi dữ liệu giữa bộ vi xử lý và bộ nhớ chính. Các bộ nhớ tiên tiến gồm có.

- Synchronous RAM, SDRAM (RAM đồng bộ)

Loại DRAM được sử dụng rộng rãi là loại DRAM đồng bộ (SDRAM). Không giống như bộ nhớ DRAM truyền thống hoạt động theo phương thức không đồng bộ, bộ nhớ SDRAM trao đổi dữ liệu với bộ vi xử lý theo phương thức đồng bộ dưới sự đồng bộ của xung nhịp từ bộ vi xử lý. Chúng hoạt động với tốc độ tối đa cho phép và không có các trạng thái đợi.

Các bộ nhớ DRAM truyền thống khi nhận được các tín hiệu địa chỉ và điều khiển chuyển tới, có nghĩa là có nhu cầu đọc hoặc ghi dữ liệu vào một vùng nào đó của bộ nhớ thì các tín hiệu này bị trễ một khoảng thời gian, khoảng thời gian trễ này được gọi sẽ thực hiện nhiều là thời gian truy cập bộ nhớ. Trong khoảng thời gian trễ truy cập, DRAM công việc khác như làm tươi bộ nhớ, gửi dữ liệu và định hướng dữ liệu tới bộ đệm ra. Vì vậy mà việc truy cập bộ nhớ trở nên chậm.

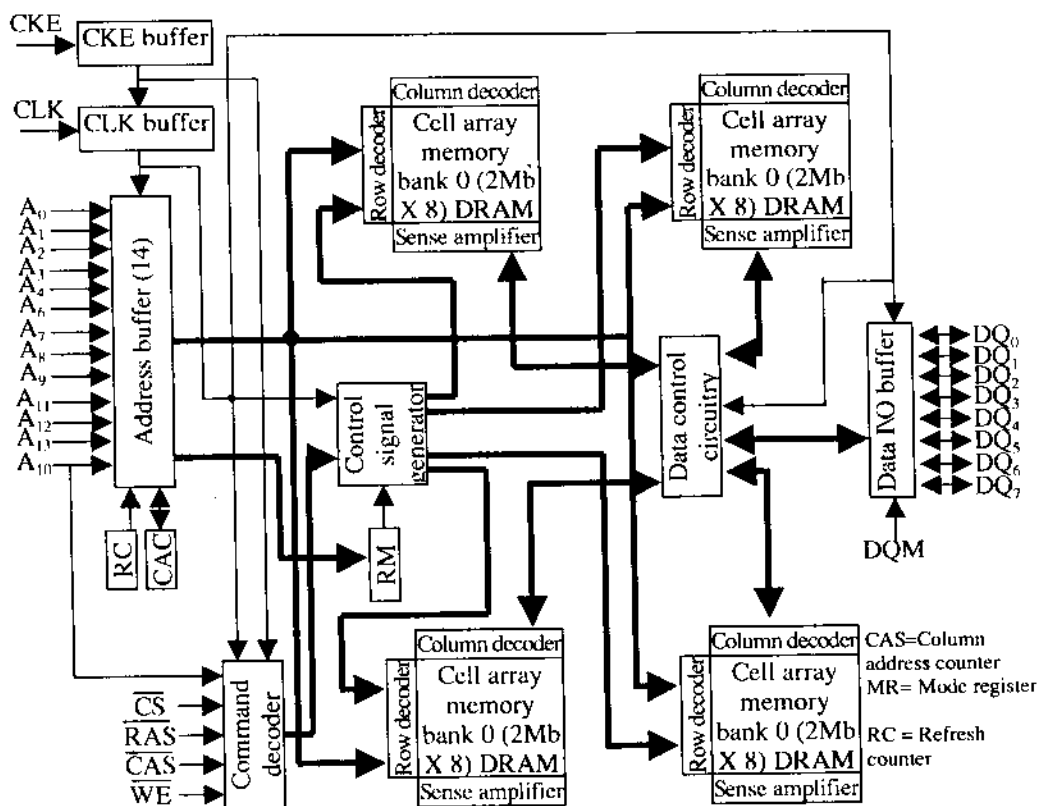
Với bộ nhớ SDRAM thì truy cập bộ nhớ theo phương thức đồng bộ. SDRAM chuyển dữ liệu vào ra dưới sự điều khiển của xung nhịp hệ thống. Khi bộ vi xử lý và các thiết bị chủ khác gửi ra lệnh điều khiển và địa chỉ tới bộ nhớ thì các tín hiệu này được bộ nhớ chốt lại và chỉ sau một số chu kỳ xung nhịp thì việc truy cập bộ nhớ của bộ vi xử lý và thiết bị chủ được đáp ứng. Sơ đồ bộ nhớ SDRAM được mô tả bằng hình vẽ 5.12.

- Cache DRAM, CDRAM

Bộ nhớ CDRAM được phát triển bởi Mitsubishi, nó chứa một bộ nhớ đệm SRAM nhỏ có dung lượng 16 Kb. Bộ nhớ SRAM trên CDRAM có thể được sử dụng vào hai mục đích:

- Thứ nhất, nó có thể được sử dụng như một bộ nhớ cache, gồm có 64 đường dây bit. Bộ nhớ cache trên CDRAM được sử dụng rất hiệu quả khi truy cập ngẫu nhiên vào bộ nhớ chính.

- Thứ hai, bộ nhớ SRAM trên CDRAM có thể được sử dụng như một bộ nhớ đệm hỗ trợ trong việc truy cập dữ liệu nối tiếp vào các khối dữ liệu. Ví dụ như làm tươi dữ liệu đưa lên màn hình.



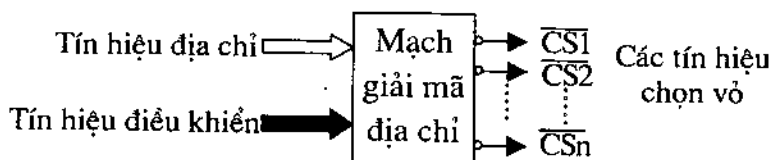
Hình 5.12: Cấu trúc bộ nhớ SDRAM

IV. GHEP NOI VI XU LY VOI BO NHÓ

1. Giai ma dia chi cho bo nhó

Mỗi mạch nhớ nối ghép với CPU cần phải được CPU quy chiếu tới một cách chính xác khi thực hiện các thao tác ghi/đọc. Điều đó có nghĩa là mỗi mạch nhớ phải được gán cho một vùng riêng biệt có địa chỉ xác định nằm trong không gian địa chỉ tổng thể của bộ nhớ. Việc gán địa chỉ cụ thể cho mạch nhớ được thực hiện nhờ một xung chọn vỏ lấy từ mạch giải mã địa chỉ. Việc phân định không gian địa chỉ tổng thể thành các vùng khác nhau để thực hiện những chức năng nhất định gọi là phân vùng bộ nhớ. Ví dụ, đối với CPU 8088 thì không gian địa chỉ tổng thể dành cho bộ nhớ là 1MB, trong đó vùng nhớ dung lượng 1KB kể từ địa chỉ thấp nhất 00000H nhất thiết phải được dành cho RAM (vì tại đây ta phải có chỗ để cho một bảng gồm 256 vector ngắt của 8088), vùng còn tại bộ nhớ có chứa địa chỉ FFFF0H thì lại nhất thiết phải được dành cho ROM hay EPROM (vì FFFF0H là địa chỉ khởi động của CPU).

Về nguyên tắc một bộ giải mã địa chỉ thường có cấu tạo như hình sau:



Hình 5.13: Mạch giải mã địa chỉ tổng quát

Đầu vào của bộ giải mã là các tín hiệu địa chỉ và tín hiệu điều khiển. Các tín hiệu địa chỉ gồm các bit địa chỉ có quan hệ nhất định với các tín hiệu chọn vỏ ở đầu ra. Tín hiệu điều khiển thường là tín hiệu IO/M dùng để phân biệt đối tượng mà CPU chọn làm việc là bộ nhớ hay thiết bị vào/ra. Mạch giải mã là một trong những khâu gây ra việc trễ thời gian của tín hiệu từ CPU tới bộ nhớ hoặc thiết bị ngoại vi mà trong khi chọn mạch nhớ/ngoại vi ta phải tính đến. Tùy theo quy mô của mạch giải mã mà ta có thể có ở đầu ra một hay nhiều tín hiệu chọn vỏ.

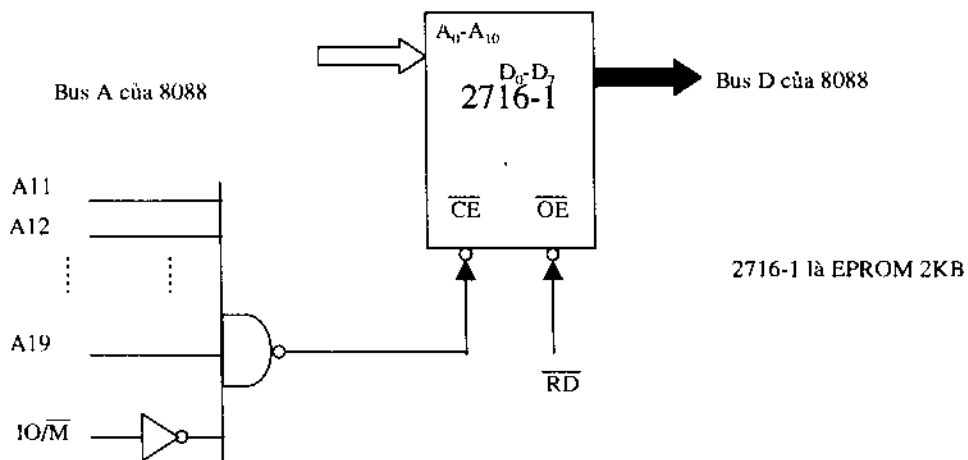
Giải mã đầy đủ cho một mạch nhớ đòi hỏi ta phải đưa đến đầu vào của mạch giải mã các tín hiệu địa chỉ sao cho tín hiệu ở đầu ra của nó chỉ chọn riêng mạch nhớ đã định. Trong trường hợp này ta phải dùng tổ hợp đầy đủ của các đầu vào địa chỉ tương ứng để chọn được mạch nhớ. Nếu ta bỏ bớt đi một bit địa chỉ nào đó thì đó là việc giải mã thiếu cho mạch nhớ, vì xung nhận được từ mạch giải mã ngoài việc chọn mạch nhớ ở vùng đã định sẽ có thể chọn ra các mạch nhớ ở vùng nhớ khác nữa.

Như vậy, giải mã thiếu thì tiết kiệm được linh kiện khi thực hiện bộ giải mã nhưng lại làm mất tính đơn trị của xung chọn thu được ở đầu ra.

Thông thường khi thiết kế mạch giải mã người ta hay tính đòi ra một chút để dự phòng, sao cho sau này nếu có sự thay đổi do phải tăng thêm dung lượng của bộ nhớ vẫn có thể sử dụng được mạch giải mã đã thiết kế.

Thực hiện mạch giải mã bằng các mạch NAND

Bằng các mạch kiểu mạch NAND ta có thể xây dựng được mạch giải mã địa chỉ đơn giản với số đầu ra hạn chế. Ta phải đưa đến đầu vào của mạch của nhiều lối vào một tổ hợp thích hợp của các bit địa chỉ để nhận được ở đầu ra của nó tín hiệu chọn vỏ cho mạch nhớ. Hình 5.14 giới thiệu một mạch giải mã dùng mạch NAND.



Hình 5.14: Mạch giải mã đơn giản dùng mạch NAND

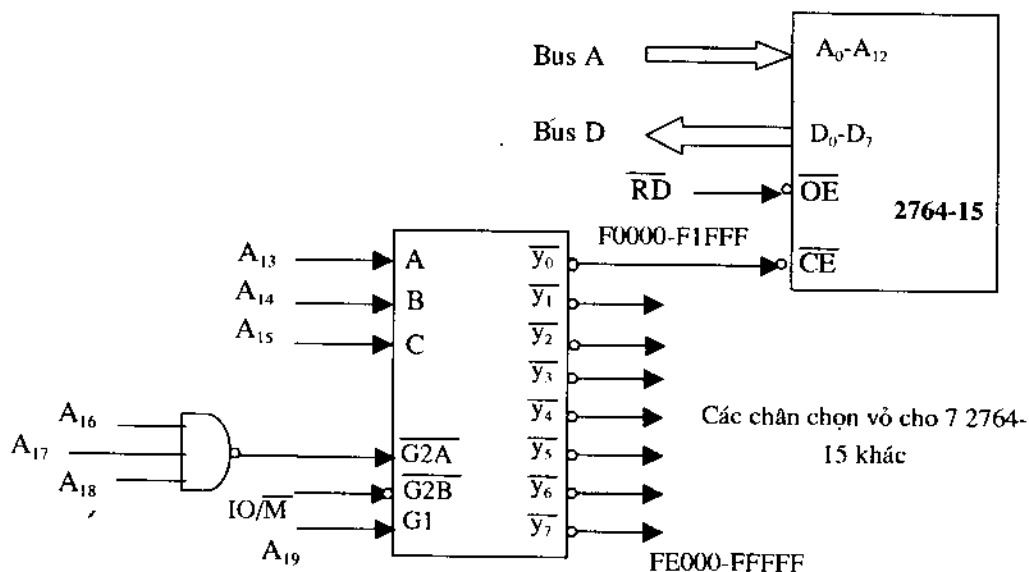
$A_{19}A_{18}A_{17}A_{16}$	$A_{15}A_{14}A_{13}A_{12}$	$A_{11}A_{10}A_9A_8$	$A_7A_6A_5A_4$	$A_3A_2A_1A_0$	
1 1 1 1	1 1 1 1	1 0 0 0	0 0 0 0	0 0 0 0	= FF800H
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	= FFFFFH

Phân địa chỉ không đổi Phân địa chỉ thay đổi

Trong mạch giải mã đơn giản cho EPROM này, xung chọn vỏ sẽ có tác động khi ta đọc bộ nhớ tại địa chỉ nằm trong khoảng FF800H-FFFFFH, tức là vùng địa chỉ có chứa địa chỉ khởi động của CPU 8088, 9 bit địa chỉ phần cao của bus địa chỉ ($A_{11} - A_{19}$), ở mức 1 sẽ phối hợp cùng xung $IO/\overline{M}$ (đã được đảo) để tạo ra xung chọn vùng nhớ 2KB đặt tại địa chỉ cao nhất trong không gian địa chỉ của 8088. Mỗi ô nhớ cụ thể trong 2KB của mạch nhớ EPROM-2716-1 sẽ do các bit thấp còn lại của bus địa chỉ ($A_0 - A_{10}$) chọn ra. Để kiểm chứng nhanh điều này ta cần nhớ rằng với bit địa chỉ A_{10} ta chọn ra được các mảng nhớ 1KB, và với bit A_{11} ta chọn ra được các mảng nhớ 2KB, các mảng nhớ này nằm rải rác trong không gian địa chỉ của bộ nhớ.

Thực hiện bộ giải mã dùng mạch giải mã kiểu 74LS138

Khi ta muốn có nhiều đầu ra chọn vỏ từ bộ giải mã mà vẫn dùng các mạch logic đơn giản thì thiết kế sẽ trở nên rất công kênh do số lượng các mạch cửa tăng lên. Trong trường hợp như vậy ta thường sử dụng các mạch giải mã có sẵn. Một trong các mạch giải mã hay được sử dụng là 74LS138.



Hình 5.15: Sơ đồ giải mã dùng 74LS138

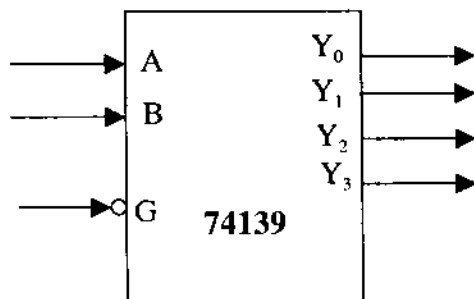
Trong đó, bảng nguyên lý hoạt động của vi mạch 74LS138 được mô tả như sau:

Bảng 5.2: Bảng nguyên lý hoạt động của vi mạch 74LS138

Các đầu vào					Các đầu ra								
Cho phép		Chọn											
G1	G2	C	B	A	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇	
X	H	X	X	X	H	H	H	H	H	H	H	H	
L	X	X	X	X	H	H	H	H	H	H	H	H	
H	L	L	L	L	L	H	H	H	H	H	H	H	
H	L	L	L	H	H	L	H	H	H	H	H	H	
H	L	L	H	L	H	H	L	H	H	H	H	H	
H	L	L	H	H	H	H	H	L	H	H	H	H	
H	L	H	L	L	H	H	H	H	L	H	H	H	
H	L	H	L	H	H	H	H	H	H	L	H	H	
H	L	H	H	L	H	H	H	H	H	H	L	H	
H	L	H	H	H	H	H	H	H	H	H	H	L	

Thực hiện bộ giải mã dùng mạch giải mã kiểu 74139

Ta có thể sử dụng mạch giải mã 74139 để thay cho mạch giải mã 74138 để giải mã cho địa chỉ bộ nhớ với dung lượng thấp. Sơ đồ chân và bảng nguyên lý hoạt động như sau:



Hình 5.16: Sơ đồ chân vi mạch giải mã 74139

Nguyên lý hoạt động của vi mạch 74139 được mô tả như bảng sau:

Bảng 5.3: Bảng nguyên lý hoạt động của vi mạch 74139

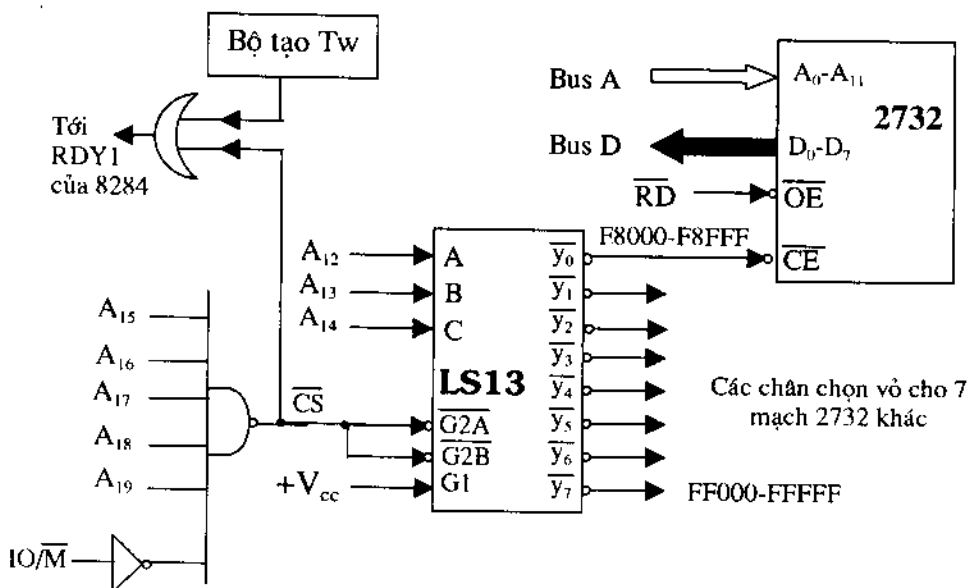
Các đầu vào			Các đầu ra			
Cho phép	Chọn					
$\overline{G}$	B	A	Y_0	Y_1	Y_2	Y_3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

2. Ghép nối bộ vi xử lý 8088 với bộ nhớ

Có thể nói một cách tổng quát rằng, nếu không có xung đột giữa tốc độ thâm nhập mạch nhớ và tốc độ CPU thì việc phối ghép CPU với bộ nhớ đơn giản chỉ là việc giải mã địa chỉ cho mạch nhớ. Trong phần lớn các trường hợp điều này có thể đúng cho các mạch nhớ RAM và các mạch nhớ EPROM có thời gian thâm nhập nhỏ hơn hoặc bằng 250ns, cách phối ghép CPU với các mạch này về cơ bản là giống nhau. Đối với các mạch nhớ EPROM như 2716,

2732... loại tốc độ 450ns khi thực hiện phối ghép với 8088-5MHz thì cần phải tính toán thận trọng hơn.

Ta biết rằng, muốn phối ghép được với CPU 8088-5MHz thì các mạch nhớ phải có thời gian thâm nhập dài nhất là cỡ 420ns, vì vậy nếu ta muốn ghép EPROM 2732 tốc độ 420ns vào bộ nhớ thì chắc chắn phải có cách để báo cho CPU xen thêm chu kỳ đợi. Trên hình là sơ đồ mạch phối ghép với EPROM 2732 có thêm mạch NAND để tạo tín hiệu cho phép mạch giải mã và tín hiệu yêu cầu đợi để đưa đến chân RDY1 của 8284. Bằng cách này mỗi khi CPU đọc EPROM 2732 tốc độ chậm thì một chu kỳ đợi sẽ tự động xen thêm.



Hình 5.17: Sơ đồ ghép nối bộ vi xử lý 8088 với bộ nhớ

Việc phối ghép SRAM với 8088 thường đơn giản hơn so với EPROM vì SRAM có tốc độ nhanh nên không cần mạch xen thêm chu kỳ đợi.

Trong hình ta có thể thấy việc ghép 16 KB SRAM với CPU có dùng một bộ giải mã kiểu 74LS138. Khối 16 KB SRAM này bắt đầu tại địa chỉ thấp nhất là 00000H, bao trùm vùng nhớ chứa bảng vectơ ngắt của CPU 8088.

3. Mở rộng bộ nhớ

3.1. Mục đích mở rộng bộ nhớ

Trên thị trường hiện nay xuất hiện rất nhiều loại IC nhớ, nhưng chúng thường có dung lượng nhỏ và vừa. Một nhiệm vụ đặt ra cho những người kỹ

thuật là phải tổ chức các IC nhớ này thành Module nhớ với dung lượng và độ dài thanh ghi lớn hơn theo yêu cầu.

Ví dụ 1: Có các IC nhớ dung lượng $2^n \times 4$ bit (gồm 2^n thanh ghi, mỗi thanh ghi độ dài 4 bit), hãy tổ chức thành Module nhớ có độ dài thanh ghi lớn (8 bit).

Ví dụ 2: Có các IC nhớ dung lượng $1M \times 4$ bit, hãy tổ chức thành Module nhớ có dung lượng $4M \times 8$ bit.

Nhiệm vụ đặt ra ở đây là tổ hợp IC nhớ có số lượng thanh ghi 2^{20} và độ dài mỗi thanh ghi 4 bit thành Module nhớ lớn có số thanh ghi 2^{22} và độ dài mỗi thanh ghi 8 bit. Với nhiệm vụ như vậy chúng ta vừa phải tăng số thanh ghi vừa phải tăng độ dài thanh ghi so với các IC nhớ đã có.

3.2. Quá trình mở rộng bộ nhớ

Để có thể xây dựng các Module nhớ theo yêu cầu dựa trên những IC nhớ đã có, quá trình tổ chức bộ nhớ được thực hiện qua những bước chính sau đây:

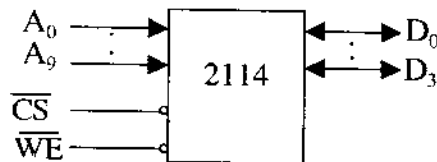
- Xác định yêu cầu của Module cần xây dựng: Về dung lượng, độ dài từ, vùng địa chỉ mà Module này chiếm chỗ.
- Xác định số lượng IC được có cần sử dụng.
- Thiết kế mạch logic để nối ghép các IC nhớ.

Trong các bước trên thì bước thiết kế mạch logic là phức tạp và quan trọng nhất, để đảm bảo những chỉ tiêu kỹ thuật của Module nhớ. Trong bước này có những nhiệm vụ sau đây phải thực hiện:

- + Ghép nối tiếp đầu ra các IC nhớ để tăng độ dài thanh ghi của Module nhớ cần thiết.
- + Ghép song song đầu ra các IC nhớ để tăng dung lượng (số dung lượng thanh ghi).
- + Thiết kế phần mạch logic để đảm bảo các thanh ghi trong Module nhớ có địa chỉ tách biệt, và nằm đúng trong vùng địa chỉ nào đó nếu có yêu cầu. Thông thường phần mạch logic này có các đầu vào là tập hợp một đường dây địa chỉ nào đó có đầu ra là các tín hiệu điều khiển chọn vỏ (CS) đưa tới chọn các IC nhớ tương ứng. Để đơn giản hoá phần mạch logic người ta sử dụng các mạch giải mã sẵn có.

3.3. Các ví dụ về mở rộng bộ nhớ

Ví dụ 1:

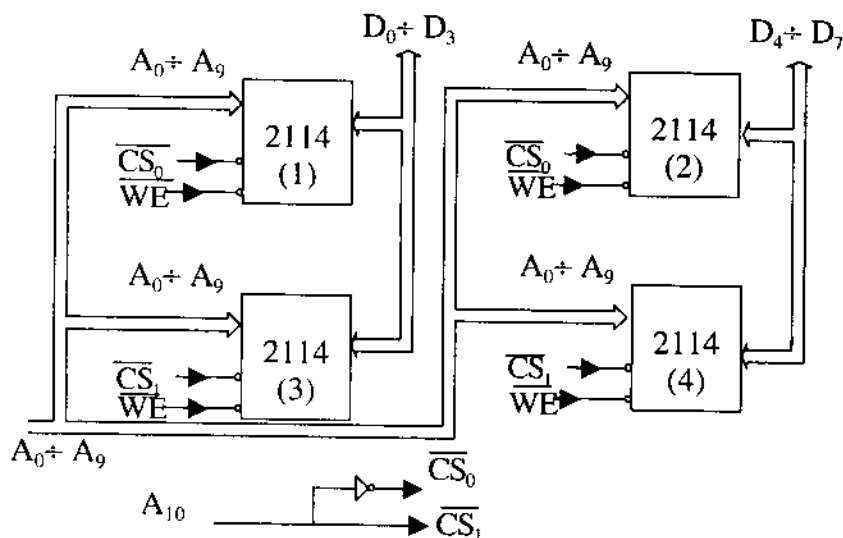


Hình 5.18: Sơ đồ chân vi mạch nhớ RAM 2114

Có các IC RAM 2114 dung lượng 1KB x 4 bit. Hãy xây dựng Module nhớ 2KB RAM.

Chúng ta thấy IC cần sử dụng là 4, vì để đạt được 1KB phải đấu đầu ra nối tiếp của 2 IC 2114. Và như vậy, để có 2KB phải đấu song song 2 nhóm trên tức là phải dùng 4IC 2114.

Để các thanh ghi của các Module tạo thành có các địa chỉ tách biệt thì phải có tín hiệu chọn vỏ thích hợp tác động vào các IC trên. Sau đây là sơ đồ nối ghép các IC này để có 2KB RAM.



Hình 5.19: Sơ đồ đấu nối 2KB RAM từ 4 IC RAM 2114

Với Module nhớ 2KB RAM ở trên chúng ta phải sử dụng 11 đường dây địa chỉ từ A_0 đến A_{10} (do $2^{11} = 2K$). Nhưng bản thân mỗi IC có dung lượng 1K nên chỉ cần 10 đường dây địa chỉ từ A_0 đến A_9 tác động vào các IC này, còn đường dây A_{10} được sử dụng để chọn cặp IC tương ứng, tránh tình trạng các thanh ghi trong các IC trùng địa chỉ. Với cách nối ở trên cặp IC1 và IC2 sẽ chứa các thanh ghi của Module nhớ trong vùng địa chỉ.

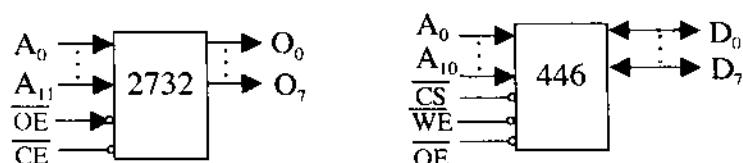
...00 0000 0000 $\Rightarrow$.. 01 1111 1111

Còn lại các cặp IC3, IC4 sẽ chứa các thanh ghi của Module nhớ trong vùng địa chỉ.

... 10 0000 0000 $\Rightarrow$... 11 1111 1111

Ví dụ 2:

Có các IC ROM 2732 dung lượng 4KB và các IC RAM μ PD 446 dung lượng 2KB. Hãy xây dựng vùng nhớ 4KB ROM và 4KB RAM.

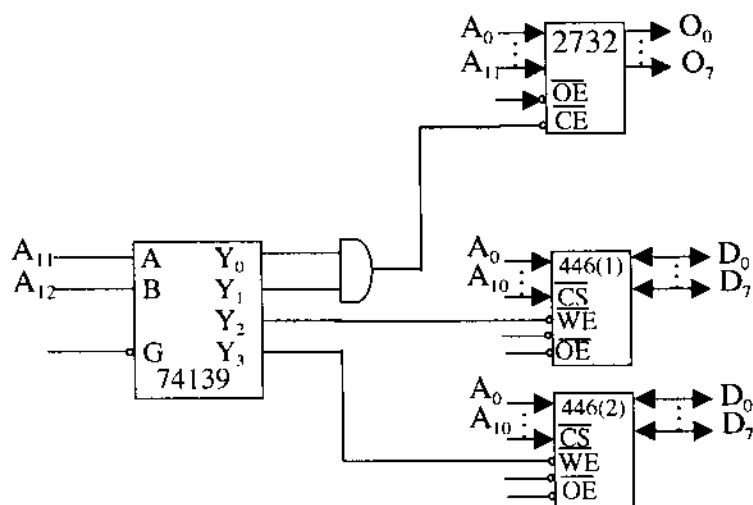


Hình 5.20: Sơ đồ chân các vi mạch ROM 2732 và RAM μ PD 446

Với yêu cầu đề bài thì chúng ta phải sử dụng 1 IC ROM và 2 IC RAM ở trên.

Do Module nhớ cần thiết kế có dung lượng 8KB nên phải sử dụng 13 đường dây địa chỉ từ A_0 đến A_{12} cho hệ thống (do $2^{13} = 8K$).

Với IC ROM có dung lượng 4KB nên có 12 đường dây địa chỉ được đưa vào từ A_0 đến A_{11} . Còn các IC RAM có dung lượng 2KB nên có 11 đường dây địa chỉ đi vào là A_0 đến A_{10} . Để tách biệt các thanh ghi của các IC nằm trong các vùng địa chỉ khác nhau sử dụng mạch giải mã 74139. Tín hiệu chọn lựa là các đường dây địa chỉ A_{11} , A_{12} đầu ra 74139 để chọn lựa các vi mạch như hình sau:



Hình 5.21: Sơ đồ bộ nhớ 4KB ROM và 4KB RAM

Với cách nối trên thì vùng địa chỉ của các thanh ghi trong IC ROM là:

... 00 00 0000 0000 $\Rightarrow$... 01 111 1111 1111 (bao gồm 4KB).

Vùng địa chỉ của IC là RAM 1 là:

...10 00 0000 0000 $\Rightarrow$...10 11 1111 1111 (bao gồm 2KB)

Vùng địa chỉ của IC RAM 2 là:

... 11 00 0000 0000 $\Rightarrow$...11 11 1111 1111 (Bao gồm 2KB).

Ví dụ 3:

Có các IC RAM μ PD 446 2KB hãy xây dựng Module nhớ RAM dung lượng 16KB. Nhưng Module này phải chứa các thanh ghi nằm trong vùng địa chỉ từ 4000h đến 7FFFh.

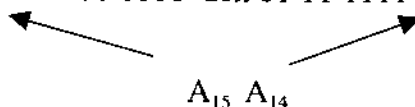
Với nhiệm vụ trên, trước hết chúng ta xác định được số IC RAM cần sử dụng là 8.

Số đường dây địa chỉ được sử dụng cho hệ thống là 14 (do $2^{14} = 16K$) từ A_0 đến A_{13} .

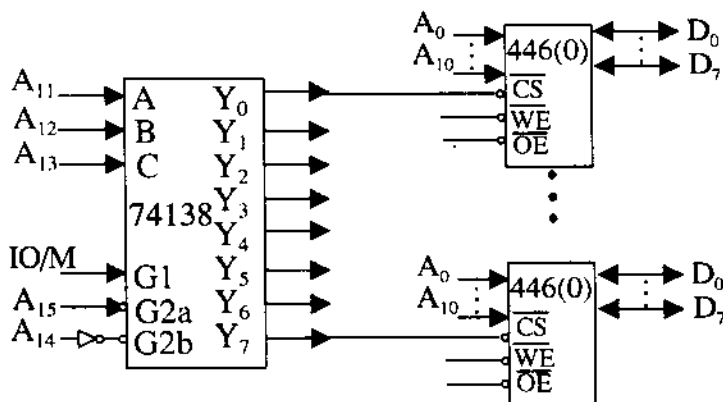
Tương tự như các ví dụ trên, chúng ta sử dụng mạch giải mã 74138 để phân 8 vùng địa chỉ riêng biệt, mỗi vùng 2KB cho mỗi IC. Một điều đáng chú ý ở đây là phải thiết kế sao cho các vùng địa chỉ trên bắt đầu từ địa chỉ 4000h và kết thúc tại địa chỉ 7FFFh.

Nếu biểu diễn vùng địa chỉ trên bằng hệ nhị phân ta có:

01 00 0000 0000 0000 đến 01 11 1111 1111 1111



Chúng ta thấy trong vùng địa chỉ này có các đường dây địa chỉ có giá trị không đổi đó là A_{15} luôn bằng 0 và A_{14} luôn bằng 1. Vì vậy Module nhớ nằm đúng trong phạm vi địa chỉ trên, chúng ta phải thiết kế sao cho mạch giải mã 74138 không đưa ra các tín hiệu chọn các IC nhớ A_{14} và A_{15} không nhận các giá trị tương ứng ở trên. Hình sau đây mô tả một cách thiết kế tiêu biểu.



Hình 5.22: Sơ đồ đấu nối bộ nhớ RAM 16KB bằng IC RAM μ PD 446 2KB

3.4. Một số Module nhớ trong thực tế

Trong thực tế các Module nhớ sử dụng làm bộ nhớ chính trong máy tính, thường được xây dựng từ các vi mạch DRAM. Có 2 loại đang được sử dụng rộng rãi đó là:

SIMM RAM: Đây là vi mạch RAM, mà các IC RAM được bố trí trên một hàng. Có thể cắm trực tiếp lên các Slot RAM trong máy tính. Dung lượng của nó tùy thuộc vào sự thiết kế, có thể là: 1M x 8, 4M x 8, 16M x 8 với những vi mạch này được giao tiếp với hệ thống bằng 32 chân. Một số loại SIMM RAM thế hệ mới có thể có dung lượng 1M x 32, 2M x 32, 4M x 32, 8M x 32 thì sử dụng 72 chân.

DIMM RAM: Đây là loại vi mạch RAM thế hệ mới, mà các IC RAM được bố trí làm 2 hàng trên 2 mặt để giao tiếp với hệ thống, thường dung lượng rất lớn.

Câu hỏi ôn tập

1. Tổ chức bộ nhớ là gì? Quá trình đọc và ghi bộ nhớ diễn ra như thế nào?
2. Bộ nhớ ROM là gì? Bộ nhớ ROM có những đặc điểm gì? Có bao nhiêu loại bộ nhớ ROM? Cho biết đặc điểm của từng loại.
3. Bộ nhớ RAM là gì? Bộ nhớ RAM có những đặc điểm gì? Có bao nhiêu loại bộ nhớ RAM? Cho biết đặc điểm của từng loại.
4. Hãy cho biết cấu trúc của bộ nhớ ROM, RAM. Nêu chức năng của từng khối và các đường dây tín hiệu.
5. Giải mã địa chỉ cho bộ nhớ là gì? Mở rộng bộ nhớ là gì? Hãy trình bày cách mở rộng bộ nhớ và lấy một ví dụ minh họa.

Chương 6

GHÉP NỐI VI XỬ LÝ VỚI THIẾT BỊ NGOẠI VI

Mục tiêu:

Giới thiệu các kiểu ghép nối bộ vi xử lý với thiết bị ngoại vi và các phương pháp điều khiển ghép nối vào ra nhằm trang bị cho học sinh những kiến thức cơ bản về cách trao đổi thông tin, dữ liệu giữa máy tính và các thiết bị ngoại vi.

Giới thiệu vi mạch ghép nối vào ra lập trình 8255A, là một vi mạch điều khiển ghép nối vào ra, có thể thay đổi được cách thức vào ra một cách linh hoạt, từ đó học sinh có thể sử dụng vi mạch ghép nối để ghép nối và điều khiển trao đổi dữ liệu giữa bộ vi xử lý, máy tính với các thiết bị ngoại vi trong thực tế.

Nội dung chính:

- I. Tổng quan về sự ghép nối
 1. Khái niệm về ghép nối
 2. Mục đích của ghép nối
 3. Các kiểu ghép nối vào ra
- II. Giải mã địa chỉ cho thiết bị vào ra
- III. Mạch ghép nối vào ra 8255A
- VI. Các phương pháp điều khiển vào ra
 1. Điều khiển bằng chương trình
 2. Điều khiển bằng ngắt vi xử lý
 3. Điều khiển bằng thâm nhập bộ nhớ trực tiếp

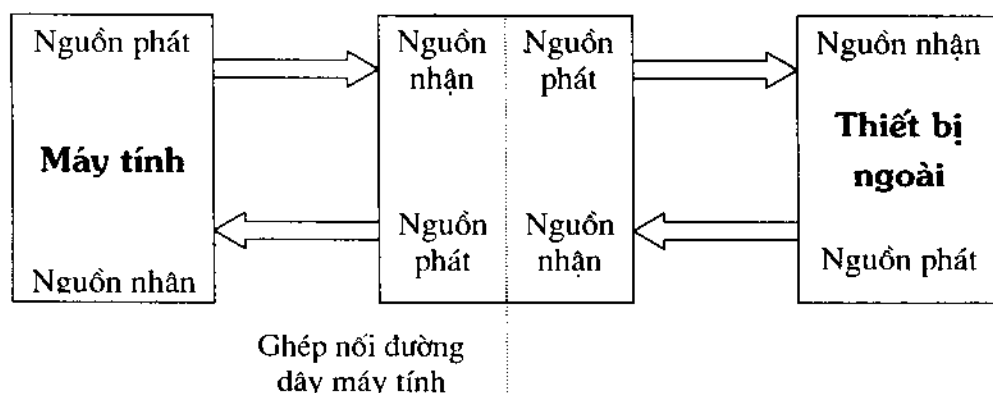
I. TỔNG QUAN VỀ SỰ GHÉP NỐI

1. Khái niệm về ghép nối

1.1. Khái niệm ghép nối

Ghép nối là sử dụng “khối ghép nối” hay còn gọi là “khối điều khiển ghép nối” để ghép giữa một máy tính với một thiết bị ngoại vi bên ngoài hoặc ghép

giữa các máy tính với nhau để trao đổi (truyền/nhận) tin giữa chúng. Ghép nối thường được thực hiện theo sơ đồ khối sau:



Hình 6.1: Sơ đồ khối ghép nối

1.1.1. Nhiệm vụ ghép nối

- Phối hợp được về mức và công suất tín hiệu:

Mức tín hiệu của máy tính thường là mức 0V và 5V, trong khi các thiết bị ngoại vi có nhiều mức khác nhau, ví dụ điện thoại có các mức $\pm 15V$, $\pm 48V$... Do đó cần phải biến đổi các mức trên cho phù hợp.

Công suất của đường dây máy tính thường nhỏ (cỡ vài chục mA) trong khi cần công suất lớn hơn cho thiết bị ngoại vi. Do đó, cần biến đổi công suất cho phù hợp như khuếch đại đường dây, khuếch đại công suất...

Thường sử dụng mạch ba trạng thái để đưa tin ra, nhận tin vào và với trở kháng cao khi không trao đổi tin để cô lập máy tính với ngoại vi để cho công suất tiêu thụ trên đường dây khi không trao đổi tin là bằng 0.

- Phối hợp về dạng tin:

Trao đổi tin của máy tính thường là song song, có thể là 8 bits, 16 bits, 32 bits... trong khi trao đổi tin của thiết bị ngoại vi có thể là song song hoặc nối tiếp và khi song song thường là 8 bits và 16 bits nên cần có sự phối hợp cho thích hợp.

- Phối hợp về tốc độ trao đổi tin:

Trong máy tính thường hoạt động với tốc độ cao, trong khi thiết bị ngoại vi thường hoạt động chậm hơn. Do đó, thiết bị ghép phải nhận tin nhanh từ máy tính để truyền cho thiết bị ngoại vi theo nhịp chậm để giải phóng cho máy tính làm nhiệm vụ khác. Quá trình truyền tin từ thiết bị ngoại vi tới máy tính cũng tương tự như vậy.

- Phối hợp về phương thức trao đổi tin:

Để đảm bảo trao đổi tin một cách tin cậy giữa máy tính và thiết bị ngoại vi, cần có khối ghép nối và cách trao đổi tin diễn ra theo một trình tự nhất định.

Nếu trao đổi tin do máy tính khởi xướng, tức máy tính chủ động đưa tin ra hay đọc tin vào thì quá trình diễn ra như sau:

+ Máy tính đưa lệnh điều khiển để khởi động thiết bị ngoại vi hay khởi động khối ghép nối.

+ Máy tính đọc trả lời sẵn sàng trao đổi hay trạng thái sẵn sàng của thiết bị ngoại vi. Nếu có trạng thái sẵn sàng mới trao đổi tin, nếu không sẵn sàng thì phải chờ và đọc lại trạng thái.

+ Máy tính trao đổi tin khi đọc thấy trạng thái sẵn sàng.

Nếu trao đổi tin do thiết bị ngoại vi khởi xướng. Để giảm thời gian chờ đợi trạng thái sẵn sàng của thiết bị ngoại vi, máy tính có thể khởi động thiết bị ngoại vi rồi thực hiện nhiệm vụ khác. Việc trao đổi tin diễn ra khi:

+ Thiết bị ngoại vi yêu cầu trao đổi tin vào bộ phận xử lý ngắt của khối ghép nối, để đưa yêu cầu ngắt chương trình cho máy tính.

+ Máy tính nhận yêu cầu, sửa soạn trao đổi và đưa ra tín hiệu xác nhận sẵn sàng trao đổi.

+ Khối ghép nối nhận và trao đổi tín hiệu xác nhận cho thiết bị ngoại vi.

+ Thiết bị ngoại vi trao đổi với khối ghép nối và khối ghép nối trao đổi tin với máy tính (nếu đưa tin vào).

+ Máy tính trao đổi tin với khối ghép nối và khối ghép nối trao đổi tin với thiết bị ngoại vi (nếu đưa tin ra).

1.1.2. Chức năng ghép nối

Tuỳ theo sự trao đổi giữa máy tính và thiết bị ngoại vi, khối ghép nối có thể có một hoặc nhiều các chức năng sau:

* Chức năng nhận tín hiệu (listener):

→ Nhận thông báo địa chỉ từ máy tính.

→ Nhận thông báo về trạng thái từ thiết bị ngoại vi.

→ Nhận lệnh điều khiển từ máy tính.

→ Nhận số liệu từ máy tính.

* Chức năng nguồn tín hiệu (talker):

- Phát tín hiệu địa chỉ cho khối chức năng của thiết bị ngoại vi.
- Phát lệnh cho thiết bị ngoại vi.
- Phát yêu cầu hay trạng thái của thiết bị ngoại vi cho máy tính.
- Phát số liệu cho thiết bị ngoại vi hay cho máy tính.

*** Chức năng điều khiển (Controller):**

Nếu khối ghép nối là chung cho nhiều thiết bị ngoại vi thì khối ghép nối sẽ đóng vai trò là khối điều khiển, có đồng thời cả hai nhiệm vụ nguồn nhận và nguồn phát lệnh, cụ thể là:

- Phát địa chỉ cho từng khối chức năng của thiết bị ngoại vi.
- Truyền lệnh cho từng khối chức năng hoặc nhiều khối.
- Nhận lệnh từ khối điều khiển khác.
- Yêu cầu trao đổi tin ở các khối chức năng, sắp xếp ưu tiên, rồi đưa ra yêu cầu vào máy tính.
- Phát nhịp thời gian cho các hành động khác nhau của các khối chức năng.

*** Chức năng phụ trợ khác:**

Ngoài các chức năng trên, khối ghép nối còn có các chức năng phụ trợ khác như sau:

- Yêu cầu phục vụ (Service Request - SR): yêu cầu máy tính trao đổi tin.
- Từ xa/cục bộ (Remote - Local): cho phép chuyển điều khiển thiết bị từ cơ cấu điều khiển bên trong (ở mặt trước của thiết bị) sang điều khiển từ xa.
- Xóa thiết bị (Clear - Device, DC): xác lập trạng thái ban đầu của thiết bị.
- Khởi phát thiết bị (Device Trigger, DT): khởi động thiết bị để thực hiện các hành động trong từng nhóm hay từng thiết bị riêng rẽ.

2. Mục đích của ghép nối

Mục đích của ghép nối máy tính với thiết bị ngoại vi hoặc giữa máy tính với máy tính là khi có nhu cầu:

*** Trao đổi tin với người điều hành thông qua thiết bị ngoại vi như bàn phím và màn hình.**

Khi người điều hành (người sử dụng) máy tính cần đưa lệnh (dưới dạng chữ) và số liệu (dưới dạng số) thông qua bàn phím. Khi người điều hành bấm vào các phím của bàn phím, thì những mã được tạo ra (mã ASCII) được truyền vào bộ nhớ của máy tính, đồng thời hiển thị lên màn hình các chữ và con số đã bấm. Trong thực tế một máy tính có thể được ghép nối với nhiều thiết

bị ngoại vi, nhiều thiết bị đầu cuối khác nhau thông qua một bàn phím và một màn hình được đặt ở một vị trí thuận tiện để một hay nhiều người điều hành trao đổi tin với máy tính với những mục đích khác nhau, ví dụ như: bán hàng, mua hàng, trao đổi thư từ, điều khiển các thiết bị...

* Trao đổi tin với các thiết bị thông dụng bên ngoài như: Máy đọc băng (từ, giấy), các bộ nhớ ngoài (băng từ, đĩa từ), máy in...

Các thiết bị ngoại vi thông dụng là các thiết bị tối thiểu thường dùng cho một hệ máy tính để nhận tin vào và đưa tin ra.

Các thiết bị đưa tin vào đó là:

- Máy đọc băng giấy: Là máy đọc tin đã lưu trữ trên băng giấy bị đục lỗ.

- Máy quét (Scanner) quang học: Là máy đọc tài liệu in theo phương pháp quét bằng một chùm sáng.

- Con chuột: Là thiết bị để người điều hành chọn thực đơn (danh sách chương trình), chọn lệnh theo chữ và hình vẽ trên màn hình.

Các thiết bị đưa tin ra đó là:

- Máy in chữ, số (đen trắng hoặc màu): Là thiết bị dùng để in chương trình (dạng chữ) và số liệu (dạng số) trên giấy.

- Máy đục băng giấy: Là thiết bị biểu diễn và lưu trữ tin (chữ và số) trên băng giấy dưới dạng các lỗ (cho tín hiệu 1) và không lỗ (cho tín hiệu 0).

Các bộ nhớ ngoài: Là bộ nhớ để lưu trữ tin mà máy tính có thể đưa tin ra để lưu trữ và đưa tin vào khi đọc.

* Trao đổi tin với các thiết bị ngoại vi khác:

Máy tính ngoài việc trao đổi tin với thiết bị ngoại vi thông dụng còn trao đổi tin với các thiết bị ngoại vi chuyên dụng khác ở chế độ đường dây ONLINE (nối mạch/trực tuyến).

- Trong hệ đo vật lý: Máy tính cần các tin vật lý như: nhiệt độ, áp suất, lực, dòng điện... dưới dạng tín hiệu điện từ các đầu dò bộ phát hiện (detector), cảm biến (sensor), bộ chuyển đổi (transducer). Ngoài ra, máy tính còn nhận các tin về trạng thái sẵn sàng hay bận của các thiết bị đo.

- Trong hệ đo - điều khiển: Máy tính nhận tin về số liệu đo, về trạng thái thiết bị đo. Đưa tin về sự chấp nhận trao đổi tin với thiết bị ngoại vi, về lệnh điều khiển các cơ cấu chấp hành của các động cơ servo, các van đóng mở, các thiết bị đóng mở, ngắt mạch điện... và các thông số kỹ thuật cho thiết bị.

* Trao đổi giữa các máy tính với nhau trong mạng:

Một máy tính trong mạng cần trao đổi tin với nhiều người sử dụng mạng, với nhiều máy tính, với nhiều thiết bị ngoại vi như các thiết bị đầu cuối, các bộ nhớ ngoài, các thiết bị lưu trữ và biểu diễn tin.

3. Các kiểu phối ghép vào ra

Có nhiều quan điểm phân loại, cách phân loại khác nhau về các kiểu ghép nối vào ra. Ví dụ như phân loại ghép nối theo cách trao đổi tin, phân loại ghép nối theo cấu trúc đường dây, phân loại ghép nối theo dạng tín vào/ra, phân loại ghép nối theo cấu trúc hệ trao đổi tin... Sau đây là một số kiểu ghép nối vào ra cơ bản:

3.1. Phân loại theo chế độ trao đổi tin

Có các kiểu ghép nối sau:

- Kiểu ghép nối theo chương trình: Là kiểu ghép nối có sự trao đổi tin theo những lệnh đưa số liệu vào (In) hay đưa số liệu ra (Out) của khối xử lý trung tâm (CPU) qua thanh chứa A (Accumulator) rồi mới đưa đến khối nhớ của máy tính. Kiểu ghép nối này còn được chia thành các kiểu ghép nối trực tiếp, hỏi vòng trạng thái và ngắt chương trình.

- Kiểu ghép nối truy nhập bộ nhớ trực tiếp (DMA): Là kiểu ghép nối mà khối DMAC có nhiệm vụ điều khiển sự trao đổi tin thay cho CPU, điều khiển trao đổi tin trực tiếp giữa thiết bị ngoại vi với khối nhớ, không qua thanh ghi chứa A của bộ vi xử lý.

3.2. Phân loại theo cách truyền tin

Gồm các kiểu ghép nối sau:

- Kiểu ghép nối song song: Là kiểu ghép nối mà trao đổi tin theo các đường dây song song giữa máy tính và thiết bị ngoại vi.

- Kiểu ghép nối song song - nối tiếp: Là kiểu ghép nối vừa song song với máy tính và vừa nối tiếp với thiết bị ngoại vi.

3.3. Phân loại theo dạng tín

Có các kiểu ghép nối sau:

- Kiểu ghép nối số: Là kiểu ghép nối mà dạng tín trao đổi dưới dạng tín hiệu số.

- Kiểu ghép nối tương tự - số: Là kiểu ghép nối khi nhận tín là tín hiệu tương tự rồi biến đổi thành tín hiệu số bằng bộ chuyển đổi từ tín hiệu tương tự sang tín hiệu số ADC.

- Kiểu ghép nối số - tương tự: Là kiểu ghép nối biến đổi tín hiệu dạng số thành dạng tương tự bằng bộ chuyển đổi từ tín hiệu số sang tín hiệu tương tự DAC.

3.4. Phân loại theo cấu trúc hệ trao đổi tin

Bao gồm các kiểu ghép nối sau:

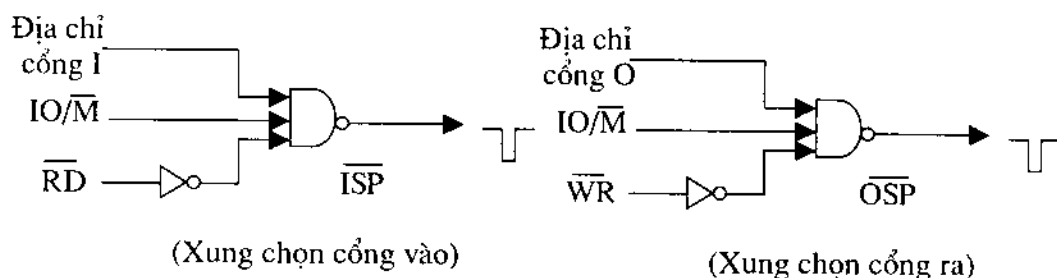
- Kiểu ghép nối một máy tính với một thiết bị ngoại vi đơn nhất.
- Kiểu ghép nối cho nhiều thiết bị ngoại vi mắc song song: Là kiểu ghép nối mà máy tính được ghép nối với nhiều thiết bị ngoại vi mắc song song qua khối phối hợp (Adapter) cho riêng từng thiết bị ngoại vi, khối phối hợp còn có tên là khối điều khiển (Controller).
- Kiểu ghép nối cho nhiều máy tính: Là kiểu ghép nối cho nhiều máy tính trong mạng để trao đổi tin giữa chúng.

II. GIẢI MÃ ĐỊA CHỈ CHO THIẾT BỊ VÀO RA

Việc giải mã địa chỉ cho thiết bị vào ra cũng gần giống như giải mã địa chỉ cho bộ nhớ. Sau đây ta sẽ đi tập trung nghiên cứu giải mã địa chỉ cho các cổng vào ra, còn các giải mã khác sẽ được xem xét trong các tài liệu chuyên ngành khác.

Thông thường các cổng vào ra có địa chỉ là 8 bit từ A_0 đến A_7 . Trong một số các hệ vi xử lý khác thì các cổng vào ra có địa chỉ là 16 bit từ A_0 đến A_{15} . Tùy theo độ dài toán hạng là 8 hay 16 bit, ta sẽ có một cổng vào ra 8 bit hay hai cổng vào ra 8 bit có địa chỉ liên nhau để tạo nên cổng vào ra có độ dài 'tù' tương ứng. Trong thực tế ít khi sử dụng hết 256 cổng vào ra khác nhau, nên ta chỉ xét các bộ giải mã địa chỉ 8 bit A_0-A_7 để tạo ra xung chọn các thiết bị.

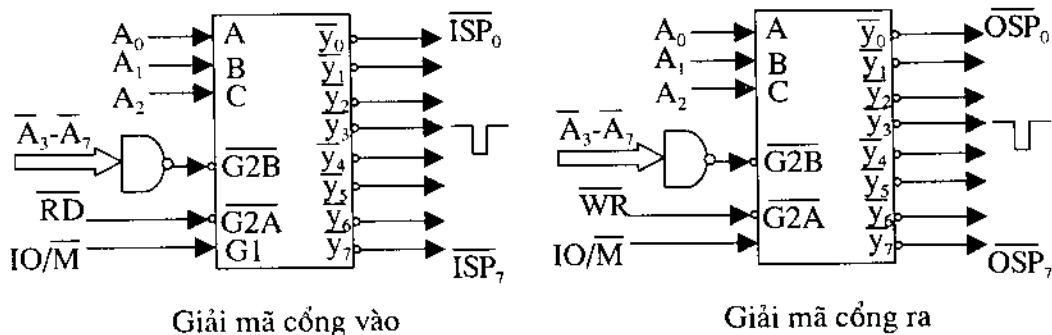
Các mạch giải mã đơn giản có thể được thực hiện bằng các cổng NAND như hình vẽ sau:



Hình 6.2: Mạch giải mã địa chỉ đơn giản cho thiết bị ngoại vi

Trong trường hợp cần có nhiều xung chọn ở đầu ra cho các cổng vào ra có địa chỉ liên tiếp, ta có thể dùng các mạch giải mã có sẵn kiểu 74LS138.

Ví dụ dùng hai cổng 74LS138 để giải mã cho 8 cổng vào và 8 cổng ra được ghép nối theo sơ đồ sau:

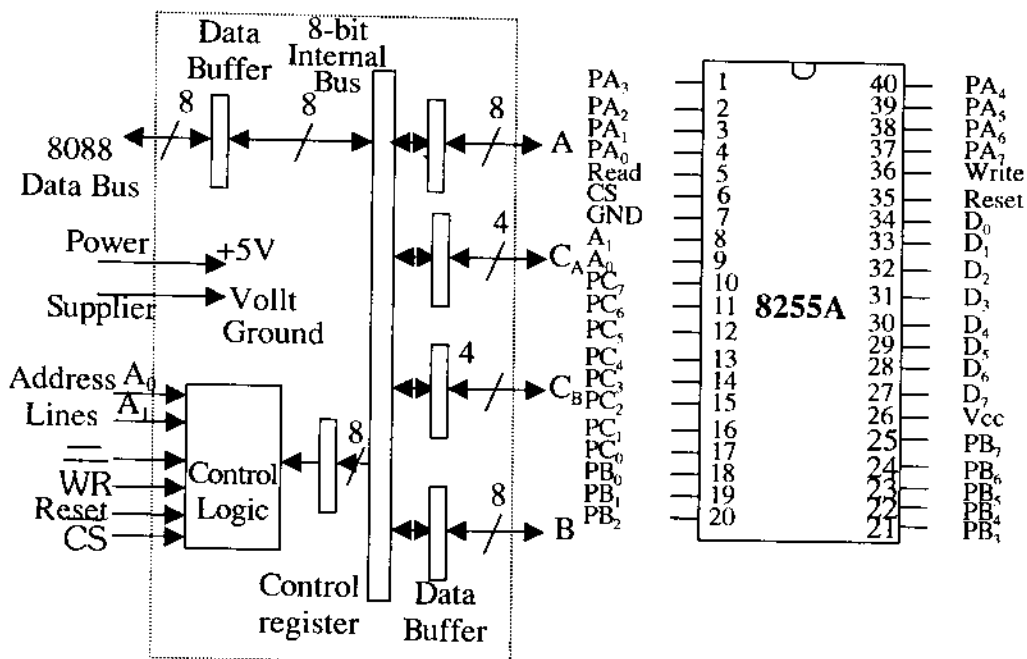


Hình 6.3: Các bộ giải mã cho các cổng vào và các cổng ra

III. MẠCH PHỐI GHÉP VÀO RA 8255A

1. Sơ đồ ghép nối

Đây là vi phối ghép vào ra lập trình được (Programmable Peripheral Interface, PPI). Do khả năng mềm dẻo trong các ứng dụng nên 8255A thường dùng để ghép nối vào ra với các bộ vi xử lý 8 - 16 bit. Trong vi mạch có 3 cổng 8 bit có thể lập trình để vào ra dữ liệu, trong đó cổng PA, PB có thể vào hoặc ra, cổng PC ngoài khả năng có thể tổ chức thành các cổng vào/ra, còn có thể điều chỉnh hoạt động các cổng PA, PB. Mỗi cổng có một thanh ghi 8 bit, ngoài ra có thanh ghi từ điều khiển CWR (Control Word Register) cũng có độ dài 8 bit. Sơ đồ khối và sơ đồ chức năng chân của vi mạch 8255A như hình 6.4



Hình 6.4: Sơ đồ khối và sơ đồ chân vi mạch 8255A

2. Hoạt động ghép nối

CPU nhìn 8255A như các thanh ghi mà địa chỉ của chúng xác định bởi A_0 - A_1 . CPU phối hợp giữa các địa chỉ này với các tín hiệu khác $\overline{CS}$, $\overline{WR}$, $\overline{RD}$ để thực hiện việc vào/ra và điều khiển các cổng của 8255A.

Bảng sau đây mô tả sự điều khiển của CPU tới 8255A.

Bảng 6.1: Bảng nguyên lý điều khiển của CPU đối với 8255A

Các tín hiệu vào					Hoạt động điều khiển
A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
X	X	X	X	1	D.BUS ở trạng thái trở kháng cao
0	0	0	1	0	Cổng PA vào dữ liệu
0	1	0	1	0	Cổng PB vào dữ liệu
1	0	0	1	0	Cổng PC vào dữ liệu
0	0	1	0	0	Cổng PA đưa ra dữ liệu
0	1	1	0	0	Cổng PB đưa ra dữ liệu
1	0	1	0	0	Cổng PC đưa ra dữ liệu
1	1	0	1	0	Cấm
1	1	1	0	0	Đưa dữ liệu từ D.BUS ra thanh ghi điều khiển

Việc CPU định giá trị cho thanh ghi điều khiển thực chất là việc định chế độ hoạt động của 8255A. Với 8255A có thể hoạt động ở 3 chế độ sau đây:

MODE 0: Vào ra cơ bản.

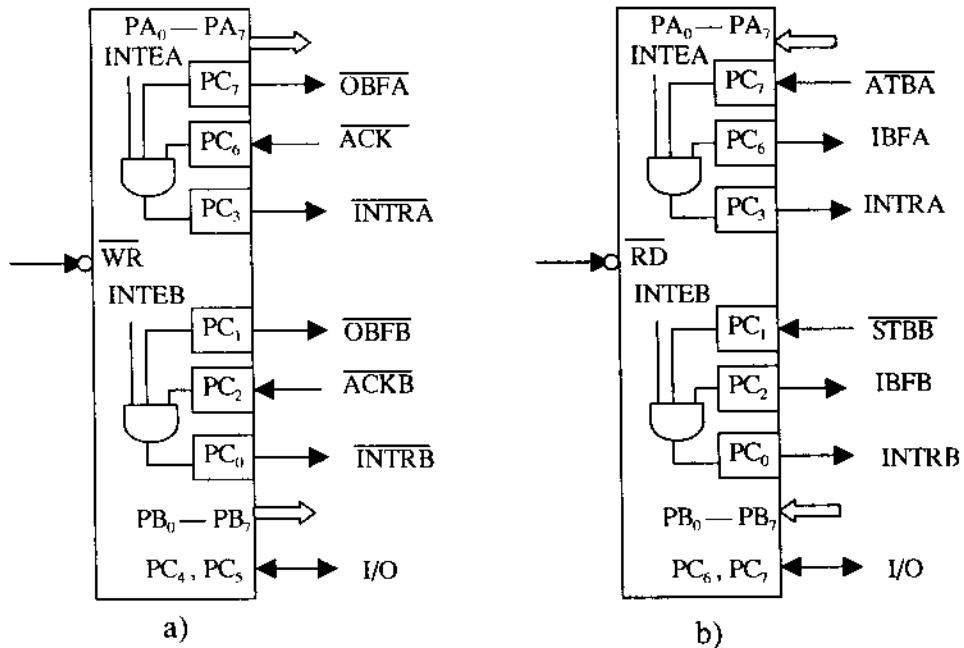
MODE 1: Vào ra với xung cho phép.

MODE 2: Vào ra hai chiều.

Hoạt động của 8255A ở 3 chế độ như sau:

MODE 0: Trong chế độ này phân ra làm hai cổng 8 bit PA, PB và hai cổng 4 bit PC_A , PC_B . Mỗi cổng này lại có thể lập trình để vào hay ra dữ liệu. Nếu là cổng ra thì phải có chốt, nếu là cổng vào thì không cần chốt.

MODE 1: 8255A có hai nhóm cổng A, B là (PA, PC_A), (PB, PC_B). Như vậy, mỗi nhóm này có một cổng 8 bit và một cổng 4 bit. Cổng 8 bit có thể điều khiển là vào hay ra còn cổng 4 bit để tạo tín hiệu liên lạc giữa hai nhóm cổng A, B.



Hình 6.5: 8255A ở chế độ 1 và các tín hiệu trạng thái

Ra dữ liệu trong mode 1: Hình 6.5a

Ở đây PA và PB cùng được định nghĩa là cổng ra và các tín hiệu míc nối tương đương nhau cho việc trao đổi dữ liệu. Ta chỉ cần nghiên cứu các tín hiệu cho PA, còn các tín hiệu cho PB tương tự.

$\overline{\text{OBFA}}$ (đệm ra của PA đầy). Tín hiệu báo hiệu cho thiết bị ngoại vi biết CPU đã ghi dữ liệu vào cổng để chuẩn bị đưa ra. Tín hiệu này thường được nối với $\overline{\text{STB}}$ của thiết bị nhận.

$\overline{\text{ACKA}}$ (Trả lời đã nhận được dữ liệu). Đây là tín hiệu của thiết bị ngoại vi cho biết là nó đã nhận được dữ liệu từ PA của 8255A.

INTRA (Yêu cầu ngắt từ PA). Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi, nó được dùng để phản ánh yêu cầu ngắt của PA tới CPU.

INTEA là tín hiệu của một mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt INTRA của PA, INTEA được lập/xoá thông qua bit PC_6 của PC.

Các tín hiệu đối thoại - trạng thái trên đều có thể lấy trực tiếp từ các chân tương ứng của vi mạch hoặc được đọc vào CPU thông qua việc đọc cổng PC. Các bit trạng thái đọc được của 8255A ở mode 1 như sau:

D_6		D_5	D_4	D_3		D_2	D_1	D_0
$\overline{\text{OBF}}$	INTEA	I/O	I/O	INTRA	INTEB	$\overline{\text{OBFb}}$	INTRB	

- Vào dữ liệu trong model: Hình 6.5b

Ở đây PA và PB cùng được định nghĩa là cổng vào và các tín hiệu móc nối tương đương nhau cho việc trao đổi dữ liệu. Ta chỉ cần nghiên cứu các tín hiệu cho PA, còn các tín hiệu cho PB tương tự.

$\overline{\text{STB}}$ (Cho phép chốt dữ liệu). Khi dữ liệu đã sẵn sàng để được đọc vào từ PA, thiết bị ngoại vi phải dùng $\overline{\text{STB}}$ để báo cho 8255A biết mà chốt dữ liệu.

IBF (Đệm vào đầy). Sau khi 8255A chốt được dữ liệu do thiết bị ngoại vi đưa đến nó đưa ra tín hiệu IBF để cho thiết bị ngoại vi biết là đã chốt xong.

INTRA (Yêu cầu ngắt từ cổng PA). Tín hiệu để báo cho CPU biết là đã có dữ liệu sẵn sàng để đọc từ PA. Đây là kết quả thu được từ quan hệ giữa các tín hiệu khác của 8255A trong quá trình đối thoại với thiết bị ngoại vi.

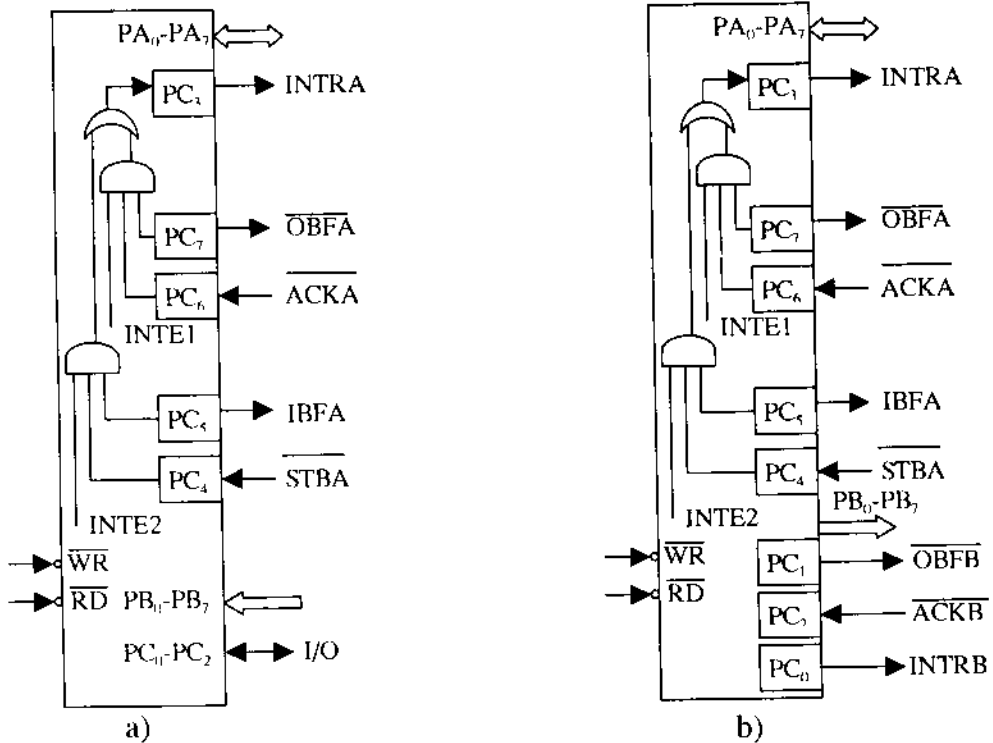
INTEA là tín hiệu của một mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt INTRA của PA. INTEA được lập/xoá thông qua bit PC_4 của PC.

Các tín hiệu đối thoại - trạng thái trên đều có thể lấy trực tiếp từ các chân tương ứng của vi mạch hoặc được đọc vào CPU thông qua việc đọc cổng PC. Các bit trạng thái đọc được của 8255A ở mode 1 như sau:

D ₆		D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
I/O	I/O	IBFA	INTE	INTR	INTEB	IBFB	INTRB

MODE 2: Các cổng được điều khiển riêng rẽ (vào ra hai chiều):

Trong chế độ này, chỉ riêng cổng PA được định nghĩa để làm việc như một cổng 2 chiều có các tín hiệu mức nổi do một số bit của PC đảm nhiệm, còn PB thì có thể làm việc ở mode 1 hoặc 0 tùy theo các bit điều khiển trong CWR. Các chân tín hiệu còn lại của PC có thể được định nghĩa để làm việc như các chân vào hoặc ra, hoặc phục vụ cho cổng PB.



PA: Mode 2
PB: Mode 1, vào
PC₀-PC₂: I/O tùy theo từ điều khiển

PA: Mode 2
PB: Mode 1, ra

Hình 6.6: 8255A ở chế độ bus hai chiều

Tín hiệu trạng thái như sau:

	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
a	$\overline{\text{OBF}}$	INTE1	IBFA	INTE2	INTRA	I/O	I/O	I/O

	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
	$\overline{\text{OBFA}}$	INTE1	IBFA	STBA	INTE2	INTEB	IBFB	INTRB

Một số tín hiệu móc nối đặc biệt gồm:

INTRA: Yêu cầu ngắt cho dữ liệu theo hai chiều vào và ra.

INTE1 và INTE2 là hai tín hiệu của hai mạch lật bên trong 8255A để cho phép/cấm yêu cầu ngắt của cổng PA. Các bit này được lập/xóa bởi các bit PC₆ và PC₄ của cổng PC.

Một số trong các tín hiệu đối thoại - trạng thái kể trên đều có thể lấy được trực tiếp từ các chân tương ứng của vi mạch hoặc có thể đọc được vào CPU từ cổng PC và cho phép điều khiển việc trao đổi dữ liệu bằng cách thăm dò các tín hiệu này.

Khi dùng 8255A trong chế độ bus hai chiều để trao đổi dữ liệu theo cách thăm dò ta phải kiểm tra xem bit $\overline{\text{OBF}}_A$ có bằng 1 (đệm ra rỗng) hay không trước khi dùng lệnh OUT để đưa dữ liệu ra cổng PA, hoặc kiểm tra xem bit IBFA có bằng 0 (đệm vào rỗng) hay không trước khi dùng lệnh IN để đọc dữ liệu vào từ cổng PA.

Dạng thức của thanh ghi từ điều khiển CWR có hai dạng:

	7	6	5	4	3	2	1	0
CWR	b ₇	b ₆	b ₅	b ₄	b ₃	b ₂	b ₁	b ₀

Thanh ghi này có hai dạng:

- Dạng 1: Định nghĩa cấu hình cho các cổng PA, PB, PC

Khi bit b₇ = 1, để định nghĩa cấu hình cho cổng PA, PB, PC

Với ý nghĩa các bit còn lại như sau:

b₀: Định cấu hình cho 4 bit thấp cổng PC:

1: In

0: Out

b₁: Định cấu hình cho cổng PB:

1: In

0: Out

b₂: Định chế độ cho cổng PB:

1: Mode 1

0: Mode 0

b₃: Định cấu hình cho 4 bit cao PC:

1: In

0: Out

b₄: Định cấu hình cho cổng PA

1: In

0: Out

b_6b_5 : Định chế độ cho cổng PA (các chế độ đã trình bày ở trên)

00: Mode 0

01: Mode 1

1x: Mode 2

- *Dạng 2: Lập xoá các bit cho cổng PC*

Khi bit $b_7 = 0$, để đặt giá trị cho các bit của PC.

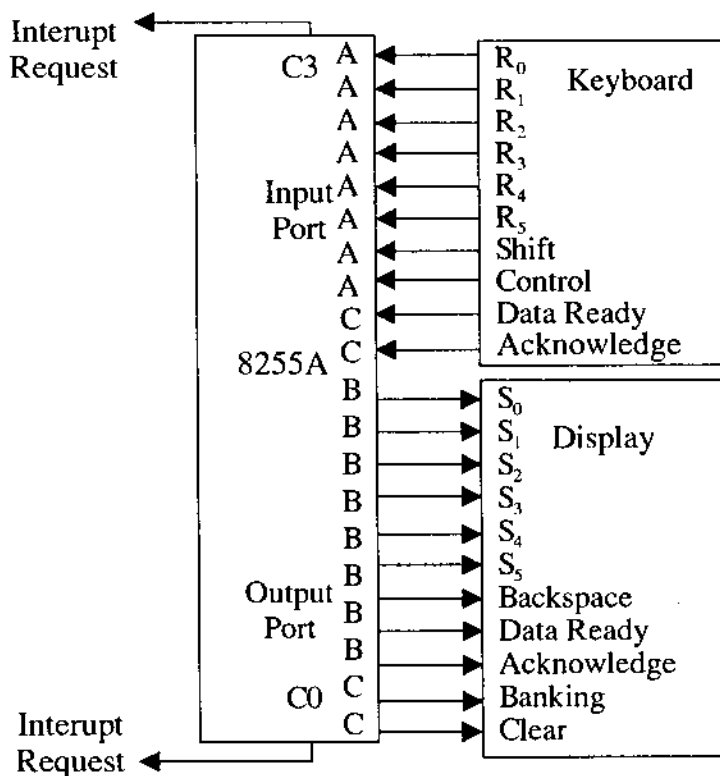
Ý nghĩa của bit như sau:

$b_3b_2b_1$: Địa chỉ bit cần đặt giá trị trong PC.

b_0 : giá trị cần đặt cho bit đó:

Ví dụ: $b_0=1$, $b_3b_2b_1=100$ để đặt bit thứ 4 của cổng C bằng 1.

3. Ứng dụng của 8255A



Hình 6.7: Sơ đồ ghép nối 8255A với bàn phím và màn hiển thị

Được sử dụng trong việc thiết kế các cổng I/O trong máy tính. Hình 6.7 là một ví dụ sử dụng 8255A như là một cổng vào và một cổng ra.

Trong sơ đồ trên 8255A sử dụng cổng PA để nhập vào dữ liệu từ bàn phím, các bit C_4 C_5 của các cổng PC phục vụ nối ghép các tín hiệu điều khiển. Cổng PB để đưa dữ liệu ra màn hình, các bit C_1 C_2 C_6 C_7 của cổng PC để phục vụ nối ghép.

IV. CÁC PHƯƠNG PHÁP ĐIỀU KHIỂN VÀO/RA

1. Điều khiển bằng chương trình (hỏi vòng)

Với phương pháp này CPU điều khiển các Module I/O bằng chương trình. Hay nói cách khác, phần mềm đóng vai trò chủ đạo trong sự ghép nối. Các chương trình này thường gồm các lệnh vào/ra dữ liệu với các cổng I/O có địa chỉ nhất định tùy thuộc vào sự ghép nối. Có hai cách chính để định địa chỉ cho các cổng I/O đó là:

** Địa chỉ I/O tách biệt*

Khi đó CPU dành riêng một vùng địa chỉ cho các cổng I/O. Với cách này để truy nhập đến các Module I/O thường phải đưa tín hiệu phân biệt trạng thái làm việc với bộ nhớ hay I/O (M/IO).

** Địa chỉ I/O theo ánh xạ bộ nhớ*

Với cách này địa chỉ của các cổng I/O cùng nằm chung trong vùng địa chỉ của bộ nhớ. Vì vậy, việc truy nhập đến các cổng I/O có thể thực hiện như việc truy nhập tới các thanh ghi của bộ nhớ.

Trong phương pháp điều khiển bằng chương trình thì có 2 kiểu ghép nối theo cấp độ phức tạp khác nhau đó là:

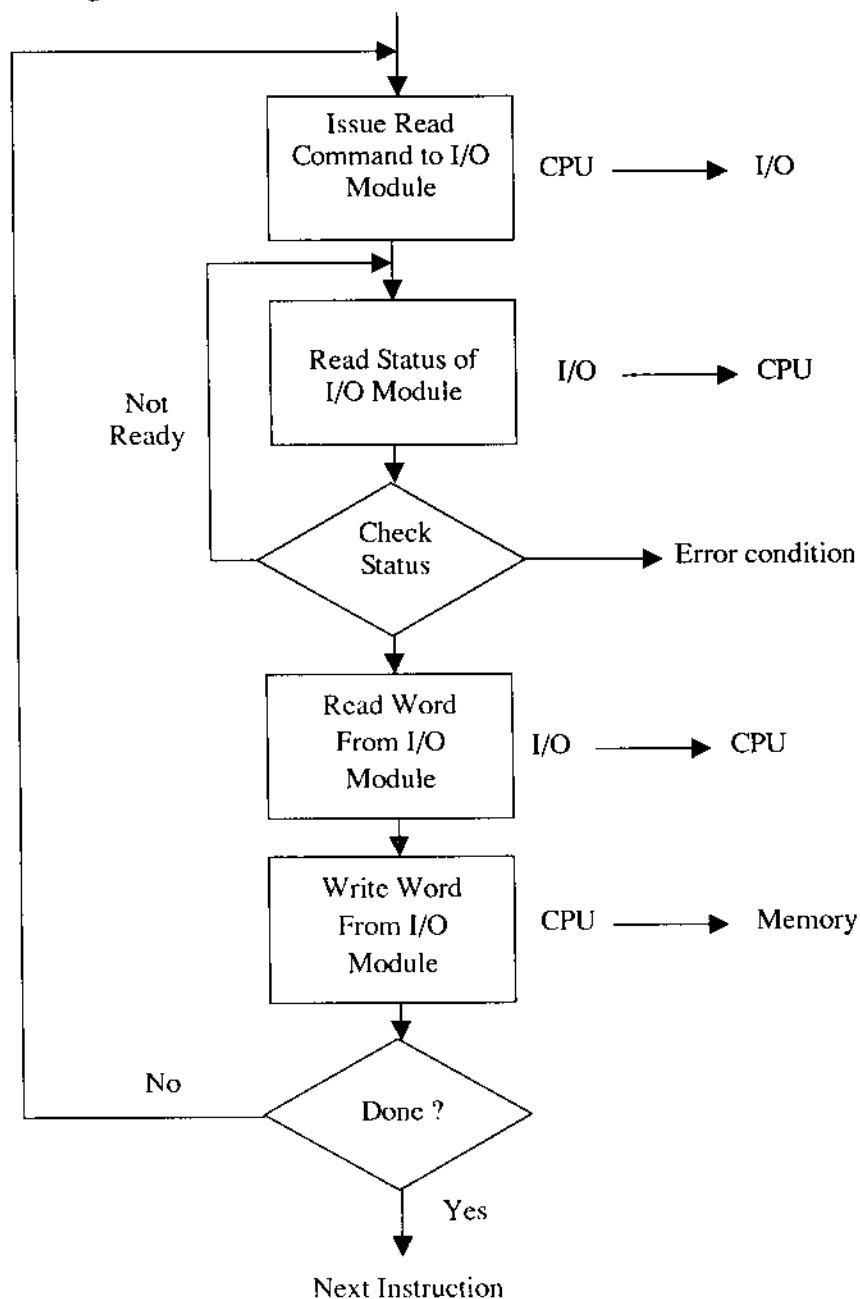
** Vào ra không điều kiện*

Với kiểu này chương trình chỉ thuần túy gồm các lệnh vào/ra dữ liệu với các cổng I/O trên các Module I/O, mà không cần biết trạng thái của các Module và các thiết bị I/O. Chương trình ở đây chủ yếu gồm các lệnh IN, OUT. Với kiểu điều khiển vào ra này có ưu điểm là chương trình điều khiển ngắn gọn, mạch điều khiển đơn giản, nhưng độ tin cậy lại không cao.

** Vào ra có điều kiện*

Với kiểu điều khiển vào ra này thì chương trình không chỉ thuần túy gồm các lệnh vào/ra dữ liệu với các cổng I/O trên các Module I/O, mà trước khi vào/ra với các thiết bị nó phải kiểm tra lần lượt (hỏi vòng) trạng thái làm việc

của các Module I/O và các thiết bị I/O. Chỉ khi các thiết bị ở trạng thái sẵn sàng thì sự vào/ra mới được thực hiện. Như vậy, với kiểu này chương trình dài, mạch phức tạp hơn, nhưng độ tin cậy cao. Lược đồ thuật toán sau miêu tả hoạt động chương trình khi đọc dữ liệu từ Module I/O vào bộ nhớ theo kiểu này.



Hình 6.8: Lược đồ thuật toán điều khiển vào ra bằng chương trình

Trong đó:

- Issue Read Command to I/O Module: Xuất lệnh đọc được tới khối I/O.
- Read Status of I/O Module: Đọc trạng thái của khối I/O.
- Check Status: Kiểm tra trạng thái.
- Read Word From I/O Module: Đọc từ khối I/O.
- Write Word From I/O Module: Ghi từ khối I/O.
- Done: Thực hiện.
- Error condition: Điều kiện lỗi.
- Not Ready: Trạng thái đang bận.
- Next Instruction: Lệnh tiếp theo.

Ví dụ: CPU thực hiện đọc cờ trạng thái của các thiết bị tại cổng 00. Nếu bit thứ i trên cờ bằng 1 thì một byte dữ liệu sẽ được vào từ cổng có địa chỉ i, dữ liệu vào được cất lần lượt ra một biến nhớ.

.Model Small

.Stack 100h

.Data

 buffers db 8 dup (?)

.Code

Main PROC

 MOV AX, @Data

 MOV ES, AX

 LEA DI, buffers

 MOV CX, 08; số lần kiểm tra

 IN BH, 00h

 MOV DX, 01h

LAP:

 ROR BH, 1; Quay phải để bit cuối ra cờ CF

 JNC khôngnhap; Cờ không lập, không nhập

 INT AL, DX; Nhập từ cổng tương ứng.

 STOSB ; Cất ra vùng nhớ.

```

Khongnhap:
INC DX
LOOP LAP
    MOV AH, 4Ch; Hàm 4C ngắt 21h để quay về DOS
    INT 21h
Main Endp
END Main

```

2. Điều khiển bằng ngắt vi xử lý

2.1. Khái niệm về phương pháp ngắt vi xử lý

Khi một thiết bị muốn nối ghép với CPU nó sẽ đưa ra một yêu cầu để ngắt CPU thông qua một trong những chân ngắt của CPU.

Sau khi nhận được yêu cầu này, nếu CPU chấp nhận, nó sẽ thực hiện nốt lệnh đang thực hiện, sau đó mới thực hiện những thao tác phục vụ cho quá trình ghép nối với thiết bị.

Trong phương pháp này chúng ta thấy rằng, CPU không mất thời gian để đi kiểm tra trạng thái sẵn sàng của các thiết bị, mà nó chỉ cần phục vụ cho những thiết bị có yêu cầu. Vì vậy tốc độ xử lý được nhanh hơn, chính xác hơn nhưng phần cứng phục vụ quá trình ghép nối phải phức tạp hơn. Một trong những công việc mà phần cứng phải làm ở đây đó là phải phân biệt ưu tiên giữa các ngắt, công việc này được thực hiện bằng mạch xử lý ngắt ưu tiên, mạch này có nhiệm vụ nhận các yêu cầu ngắt từ các thiết bị và chọn ra yêu cầu có mức ưu tiên cao nhất, sau đó báo hiệu cho CPU phục vụ thiết bị có yêu cầu ngắt tương ứng.

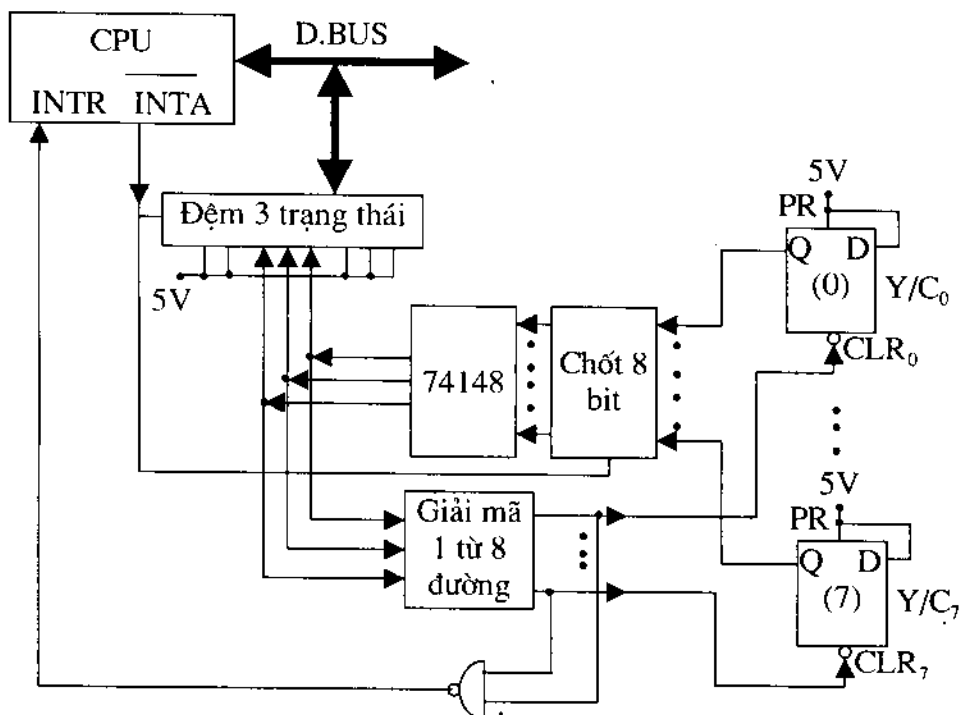
2.2. Những hoạt động của MP khi chấp nhận yêu cầu ngắt

- Cài trạng thái hiện hành ra ngăn xếp, để có thể quay trở lại chương trình chính sau khi đã phục vụ xong yêu cầu ngắt. Thường ở đây là việc cất các thanh ghi đặc biệt như bộ đếm chương trình, thanh ghi cờ.

- Thực hiện chương trình tương ứng phục vụ cho quá trình ghép nối giữa CPU với thiết bị đang có yêu cầu ngắt. Bằng cách gán bộ đếm chương trình đến địa chỉ chương trình này.

- Cuối cùng CPU phải khôi phục lại những thông tin đã cất giữ và quay trở về thực hiện lệnh tiếp theo trong chương trình chính.

2.3. Ví dụ về tổ chức nối ghép theo phương pháp ngắt MP



Hình 6.9: Sơ đồ ghép nối theo phương pháp ngắt CPU

Trên đây là sơ đồ gồm 8 thiết bị khác nhau được nối vào hệ thống, mỗi một thiết bị đều có một đường dây để gửi yêu cầu ngắt tới chốt và sau đó tới CPU thông qua mạch cổng.

Mạch giải mã ưu tiên 74148 nhận các tín hiệu yêu cầu ngắt từ các thiết bị thông qua các chốt, để định ra thiết bị nào được ưu tiên nhất sẽ được phục vụ ghép nối.

Mạch 'giải mã một từ 8 đường' có nhiệm vụ xoá yêu cầu từ thiết bị đã được phục vụ.

Mạch đếm 3 trạng thái dùng để giữ địa chỉ chương trình phục vụ ngắt.

Với cách thiết kế như trên thì nguyên lý hoạt động của sự nối ghép như sau:

Khi một hoặc nhiều thiết bị nào đó trong 8 thiết bị có yêu cầu phục vụ ngắt, các yêu cầu này một mặt được gửi tới chân ngắt INTR của CPU, một mặt lại qua chốt (Trigơ D) để tới thanh ghi chốt. Khi CPU chấp nhận yêu cầu ngắt từ chân INTR, nó sẽ đưa tín hiệu trả lời ngắt qua chân INTA cho phép mở

thanh ghi chốt. Mạch giải mã 74148 xác định được thiết bị ưu tiên nhất để phục vụ. Địa chỉ chương trình phục vụ ghép nối với thiết bị này được tạo ra trong thanh đếm 3 trạng thái. Tiếp theo CPU mở đếm ba trạng thái để đọc địa chỉ chương trình phục vụ ngắt tương ứng cũng bằng tín hiệu $\overline{INTA}$. Khi phục vụ xong yêu cầu của thiết bị nào đó thì yêu cầu của thiết bị đó sẽ được xoá trên Triggers tương ứng bằng các tín hiệu đầu ra của mạch 'giải mã 1 từ 8 đường' (74138). Tiếp theo các thiết bị khác lại có thể được phục vụ ngắt.

Trên thực tế người ta đã thiết kế và bán ra thị trường những mạch xử lý ngắt có sẵn khá hiện đại và tiện dụng. Một mạch tiêu biểu dạng này là mạch xử lý ngắt 8259 chúng ta có điều kiện xét sau.

3. Điều khiển bằng thâm nhập bộ nhớ trực tiếp DMA

3.1. Khái niệm

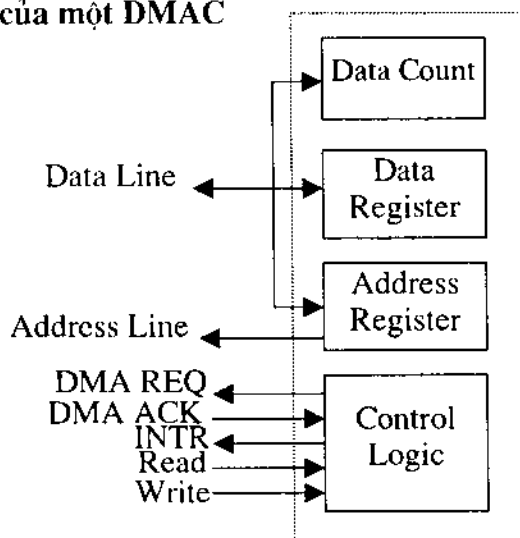
Đây là phương pháp sử dụng phần cứng phụ từ bên ngoài để điều khiển trực tiếp sự vào/ra giữa thiết bị ngoại vi và bộ nhớ mà không cần thông qua CPU. Khi đó CPU phải được treo khỏi Bus hệ thống.

3.2. Đặc điểm

- Khi sử dụng vi mạch điều khiển vào/ra trực tiếp (DMAC) thì tốc độ của quá trình chuyển dữ liệu được tăng lên rõ rệt. Bởi vì trong phương pháp này dữ liệu không phải truyền qua CPU mà được truyền trực tiếp giữa đích và nguồn.

- Phương pháp này đòi hỏi phải có thêm một phần cứng để điều khiển vào/ra. Do đó về phần cứng sẽ bị phức tạp.

3.3. Sơ đồ khối của một DMAC



Hình 6.10: Sơ đồ khối bộ điều khiển ngắt DMAC

- Thanh ghi dữ liệu: (Data Register): Để lưu các dữ liệu trung gian trước khi được truyền.

- Thanh ghi địa chỉ (Address Register): Thanh ghi này dùng để lưu giữ địa chỉ của ô nhớ tiếp theo trong sự truy xuất dữ liệu.

- Bộ đếm dữ liệu (Data Count): Đây là thanh ghi để lưu trữ số lượng dữ liệu cần phải truyền.

- Logic điều khiển (Control Logic): Đây là bộ phận điều khiển hoạt động chung của toàn vi mạch DMAC.

Chúng ta thấy có một số tín hiệu được móc nối giữa DMAC với bên ngoài:

* Các đường dây địa chỉ: Đây là các đường dây để xác định thanh ghi nào sẽ được truy nhập dữ liệu (có thể là đọc hoặc ghi dữ liệu).

* Các tín hiệu điều khiển: RD, WR đây là các tín hiệu được đưa đến các thiết bị ngoại vi hay bộ nhớ để cho phép ghi hoặc đọc thông tin.

* INTR: Tín hiệu ngắt CPU.

* Các tín hiệu hỏi đáp: Đó là các tín hiệu ACK, REQ yêu cầu CPU treo khỏi hệ thống và các tín hiệu trả lời.

* Các đường dây dữ liệu: Đây là các đường dây dùng để truyền dữ liệu trung gian giữa DMAC và thế giới bên ngoài như: Thiết bị I/O hoặc bộ nhớ.

3.4. Hoạt động của DMAC

Đầu tiên một thiết bị ngoại vi nào đó muốn trao đổi dữ liệu trực tiếp giữa chúng với bộ nhớ sẽ đưa yêu cầu tới DMAC, CPU sẽ ghi địa chỉ vùng nhớ cần chuyển và số từ cần chuyển vào kênh tương ứng của DMAC. Khi các thiết bị ngoại vi đã sẵn sàng làm việc thì DMAC gửi một tín hiệu yêu cầu treo tới CPU qua chân DMA REQ. Nếu CPU chấp nhận, nó sẽ gửi một tín hiệu trả lời qua chân DMA ACK tới DMAC và treo khỏi D.BUS, C. BUS, A.BUS. Khi DMAC nhận được tín hiệu trả lời này, nó sẽ bắt đầu điều khiển quá trình truyền dữ liệu. DMAC sẽ lần lượt đưa ra tín hiệu đọc thiết bị ngoại vi hay bộ nhớ để đọc được một dữ liệu vào thanh ghi dữ liệu, tiếp theo CPU lại đưa tín hiệu điều khiển để điều khiển ghi byte dữ liệu đã đọc được ra thiết bị ngoại vi hay bộ nhớ. Sau mỗi lần chuyển được một byte thì bộ đếm giảm 1 và thanh ghi địa chỉ được tăng 1. Quá trình cứ tiếp tục đến khi nào bộ đếm bằng 0, có nghĩa là đã truyền hết dữ liệu. Cuối cùng DMAC sẽ đưa ra một yêu cầu ngắt CPU (INTR) báo đã hoàn thành quá trình chuyển dữ liệu và trả lại điều khiển về cho CPU.

3.5. Các phương pháp DMA

- DMA theo khối: Với phương pháp này DMAC sẽ chuyển liên tục dữ liệu cho đến khi chuyển hết khối nhớ.

- DMA theo chu kỳ: Với phương pháp này DMAC thực hiện chuyển một số byte nhất định sau những chu kỳ thời gian nào đó, cho đến hết mà không chuyển liên tục. Với những phương pháp này không làm ngắt hoạt động CPU với BUS hệ thống trong một khoảng thời gian dài. Nhưng tốc độ DMA chậm hơn.

- Phương pháp “trong suốt”: Đây là phương pháp mà DMAC chỉ làm việc khi CPU không sử dụng Bus hệ thống. Do đó, phương pháp này không làm ảnh hưởng đến tốc độ làm việc của CPU, nhưng mạch sẽ rất phức tạp và tốc độ chuyển dữ liệu chậm.

Câu hỏi ôn tập

1. Ghép nối là gì? Ghép nối để làm gì? Khối ghép nối phải có những chức năng gì?
2. Giải mã địa chỉ cho thiết bị vào ra là gì? Hãy trình bày một ví dụ giải mã địa chỉ cho thiết bị vào ra.
3. Chức năng của vi mạch 8255 là gì? Vi mạch có những đặc điểm gì? Hãy tóm tắt nguyên lý hoạt động của vi mạch.
4. Có bao nhiêu phương pháp điều khiển ghép nối vào ra? Trình bày phương pháp điều khiển vào ra bằng ngắt vi xử lý.
5. Trình bày phương pháp điều khiển ghép nối vào ra bằng cách thâm nhập bộ nhớ trực tiếp.

Chương 7

CÁC MẠCH GHÉP NỐI PHỤ TRỢ KHÁC

Mục tiêu:

Giới thiệu một số vi mạch ghép nối phụ trợ cho các bộ vi xử lý, ví dụ như vi mạch điều khiển ngắt 8259, vi mạch điều khiển truy cập bộ nhớ trực tiếp 8257, vi mạch điều khiển ghép nối truyền dữ liệu 8251... để cho học sinh có được kiến thức một cách toàn diện về hệ thống vi xử lý và máy tính.

Giới thiệu một số ghép nối cơ bản nhằm cung cấp thêm kiến thức về ghép nối giữa bộ vi xử lý và máy tính với các thiết bị phụ trợ khác để trao đổi thông tin, dữ liệu, từ đó học sinh có thể nghiên cứu phát triển ghép nối bộ vi xử lý và máy tính với các thiết bị khác để mở rộng các ứng dụng trong thực tế.

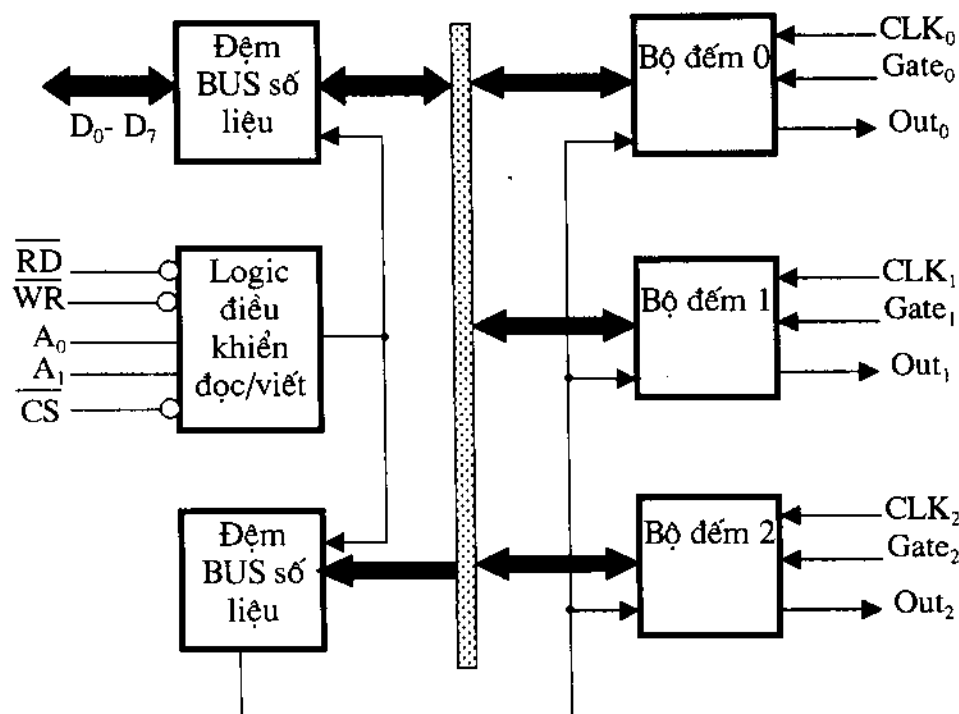
Nội dung chính:

- I. Vi mạch đếm khoảng thời gian lập trình 8253
- II. Vi mạch điều khiển thâm nhập bộ nhớ trực tiếp 8257
- III. Vi mạch điều khiển ngắt 8259
- VI. Vi mạch điều khiển ghép nối 8251
- V. Một số ghép nối cơ bản:
 1. Ghép nối bàn phím
 2. Ghép nối màn hình
 3. Ghép nối máy in
 4. Ghép nối ổ đĩa

I. VI MẠCH ĐẾM KHOẢNG THỜI GIAN LẬP TRÌNH 8253

1. Sơ đồ cấu trúc

Vi mạch 8253 đóng vai trò là bộ đếm khoảng thời gian lập trình được, có sơ đồ khối như sau:



Hình 7.1: Sơ đồ khối của vi mạch 8253

2. Nguyên lý hoạt động ghép nối

* CPU sử dụng các đường dây địa chỉ A_0 , A_1 để tác động đến các thanh ghi của 8253. Các đường dây địa chỉ còn lại thường để tổ hợp thành tín hiệu chọn vỏ CS.

Cụ thể địa chỉ của A_0, A_1 trở đến các thanh ghi được mô tả bằng bảng sau đây:

Bảng 7.1: Giải mã địa chỉ thanh ghi 8253

A_1	A_0	Để chọn ra
0	0	Bộ đếm 0
0	1	Bộ đếm 1
1	0	Bộ đếm 2
1	1	Thanh ghi điều khiển

* Dạng thức từ điều khiển

SC ₁	SC ₀	RL ₁	RL ₀	M ₂	M ₁	M ₀	BCD
-----------------	-----------------	-----------------	-----------------	----------------	----------------	----------------	-----

Ý nghĩa của các bit như sau:

- $SC_1 SC_0$: Tạo thành hai bit xác định bộ đếm nào được từ điều khiển này tác động.

00: Biểu diễn từ điều khiển này cho bộ đếm 0.

01: Biểu diễn từ điều khiển này cho bộ đếm 1.

10: Biểu diễn từ điều khiển này cho bộ đếm 2.

11: Đây là tổ hợp cấm.

- $RL_1 RL_0$: Với các chức năng điều khiển tương ứng giữa các giá trị như sau:

00: Chỉ cho phép đọc giá trị bộ đếm, trong khi đang đếm .

01: Chỉ cho phép đọc ghi BYTE thấp của bộ đếm.

10: Chỉ cho phép đọc ghi BYTE cao của bộ đếm.

11: Cho phép đọc ghi cả 2 BYTE của bộ đếm.

- $M_2 M_1 M_0$: Biểu diễn kiểu tạo xung ra của bộ đếm.

000: Kiểu 0.

001: Kiểu 1.

X10: Kiểu 2.

X11: Kiểu 3.

100: Kiểu 4.

101: Kiểu 5.

- BCD: Bit này điều khiển bộ đếm, đếm theo số nhị phân hay số BCD. Nếu bit này bằng 0 biểu diễn bộ đếm sẽ đếm theo số nhị phân 16 bit. Nếu bit này bằng 1 biểu diễn bộ đếm sẽ đếm theo số BCD (với số lớn nhất 9999).

CPU có thể tác động đến các thanh ghi của 8253 bằng các tín hiệu điều khiển và các tín hiệu địa chỉ được mô tả trong bảng sau:

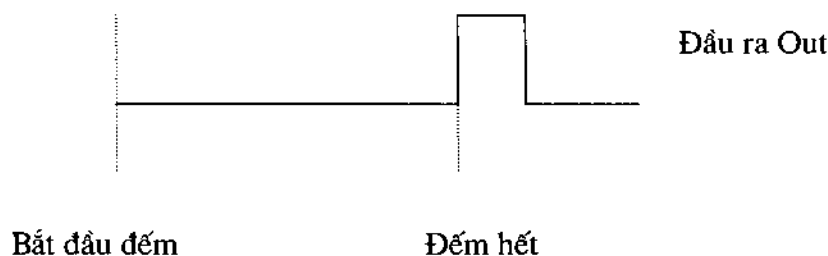
Bảng 7.2: Giải mã và điều khiển ghi đọc các thanh ghi - bộ đếm

Các tín hiệu vào					Hoạt động điều khiển
A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	
0	0	1	0	0	Ghi cơ số đếm vào bộ đếm 0
0	1	1	0	0	Ghi cơ số đếm vào bộ đếm 1
1	0	1	0	0	Ghi cơ số đếm vào bộ đếm 2

1	1	1	0	0	Ghi ra điều khiển (CWR)
0	0	0	0	0	Đọc bộ đếm 0
0	1	0	1	0	Đọc bộ đếm 1
1	0	0	1	0	Đọc bộ đếm 2
1	1	0	1	0	Tổ hợp cấm

Sau đây là các kiểu xung phát ra ở mỗi độ đếm tương ứng với các kiểu ở trên.

Kiểu 0: Tạo xung ngắt.

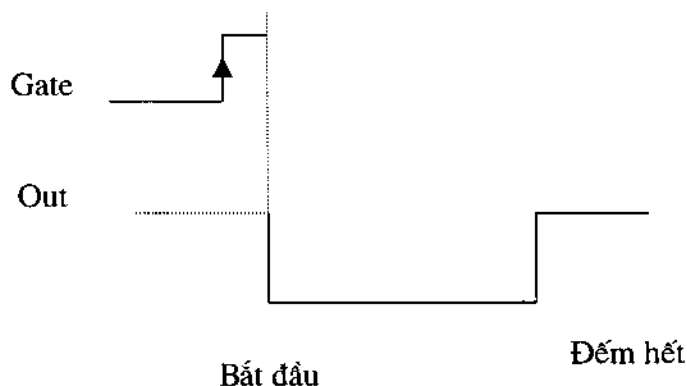


Với kiểu này khi đếm hết, xuất hiện một xung 1 ở đầu ra. Khi bắt đầu đếm đầu ra Out = 0.

Kiểu 1: Mạch đa hài đợi lập trình được.

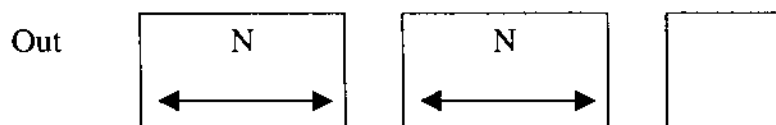
Khi cửa Gate có mức đột biến từ 0 lên 1, ngay sau 1 xung đầu ra Out về 0.

Khi đếm hết đầu ra Out lên 1.

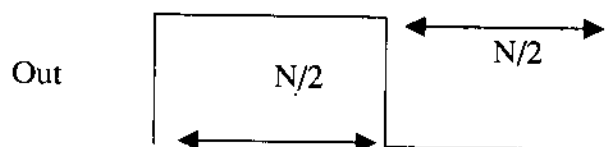


Kiểu 2: Bộ chia tần lập trình được.

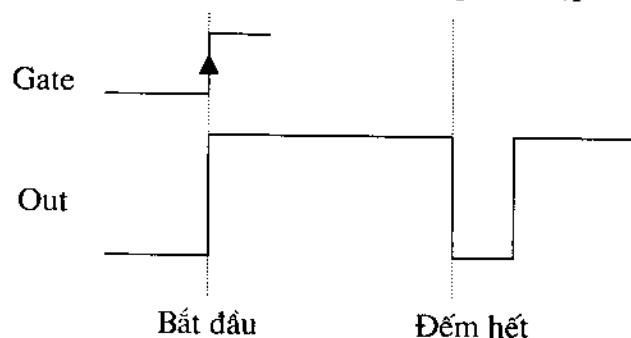
Sau mỗi lần đếm hết N xung, đầu ra xuất hiện một xung 0 lên một.



Kiểu 3: Tạo ra một chuỗi xung vuông:

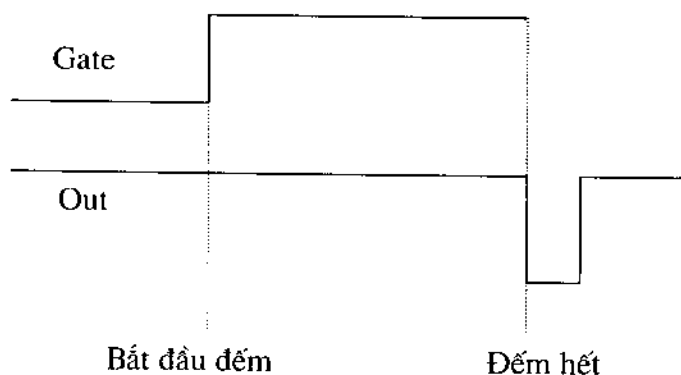


Kiểu 4: Chế độ tạo xung cho phép STB bằng cách lập trình:



Kiểu 5: Chế độ tạo xung cho phép STB bằng mạch.

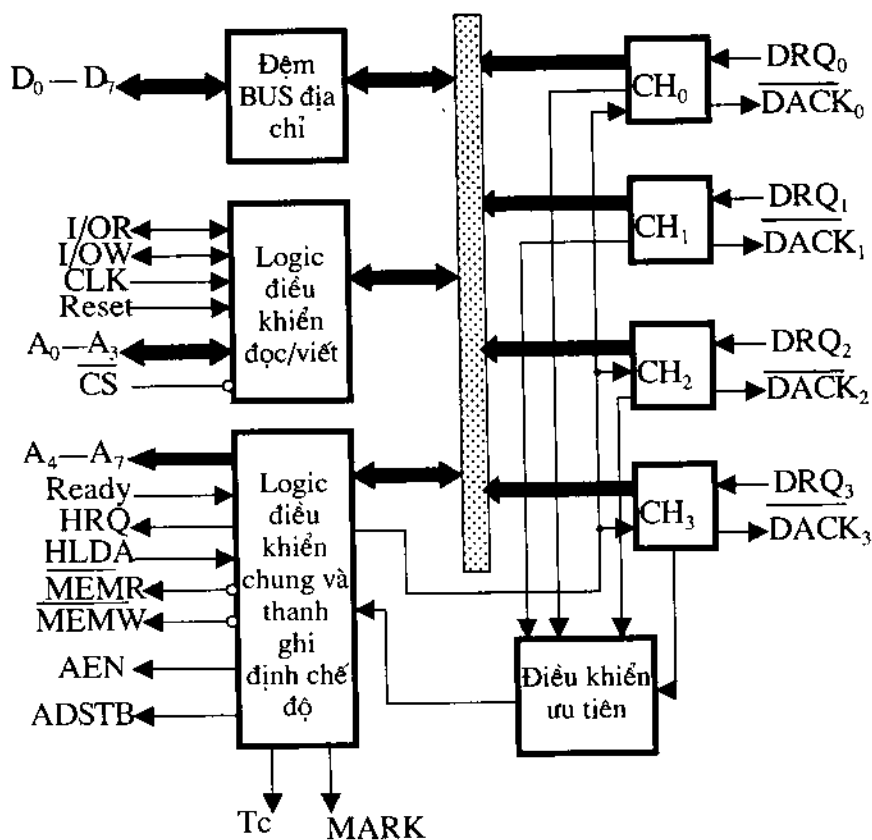
Bộ đếm bắt đầu đếm khi Gate từ 0 lên 1. Khi đếm hết xuất hiện một xung 0 ở đầu ra.



II. VI MẠCH ĐIỀU KHIỂN THÂM NHẬP BỘ NHỚ TRỰC TIẾP 8257

1. Sơ đồ cấu trúc

Vi mạch 8257 đóng vai trò là bộ điều khiển thâm nhập bộ nhớ trực tiếp DMAC, sơ đồ khối của vi mạch như sau:



Hình 7.2: Sơ đồ khối cấu trúc của 8257

ADSTB: Cho phép chốt địa chỉ có trên D_0 đến D_7 của DMAC, vì vậy thường phải có thêm thanh ghi chốt (8212).

AEN: Là chân cho phép thả nổi D.BUS, C.BUS, A.BUS của các khối nối với CPU nhưng không thực hiện DMA.

Tc: Đây là chân phát xung báo hiệu đã chuyển hết số lượng byte và kênh tương ứng đó sẽ bị cấm (Terminal Count).

Mark: Đầu ra bộ đếm mã đếm Module 128 (cơ số đếm 128)... Mỗi khi có 128 chu kỳ của DMA, thì có một xung ra ở chân này → Đếm xung này có thể xác định số chu kỳ DMA đã thực hiện.

2. Nguyên lý hoạt động ghép nối

Thông thường 8257 kết hợp 8212 trở thành mạch điều khiển DMA cho 4 kênh mỗi kênh có khả năng chuyển 16 Kbyte một lúc (2^{14}).

Khi mạch này nhận được yêu cầu cần chuyển số liệu theo kiểu DMA (thông qua DRQ), nó sẽ hoạt động như sau:

- Đầu tiên đưa yêu cầu sử dụng C.BUS hệ thống hay các BUS khác thông qua HRQ tới CPU.

- Nếu MP chấp nhận yêu cầu treo → Mạch DMAC báo cho kênh ưu tiên cao nhất được phép thực hiện chu trình DMA bằng tín hiệu DACK(i) (Kênh i được ưu tiên nhất).

- Địa chỉ thấp sẽ được đưa ra A_0 đến A_7 của DMAC 8257.

- Byte địa chỉ phần cao được đưa ra chân số liệu D_0 đến D_7 của 8257 thông qua sự có mặt 8212 đưa BUS địa chỉ → Byte thấp, cao → A.BUS của hệ thống.

- Sau đó 8257 tạo ra xung đọc viết RD, WR đối với thiết bị I/O hay để chuyển số liệu, khi chuyển xong số liệu đầu ra TC có 1 xung thông báo quá trình chuyển dữ liệu đã hoàn thành.

Có 3 kiểu DMA thực hiện bởi mạch này:

- Đọc bằng DMA: Số liệu đi từ bộ nhớ → I/O.

- Ghi bằng DMA: Số liệu đi từ I/O → M.

- Chế độ kiểm tra: Không có xung đọc viết RD, WR, không có số liệu truyền trên BUS, ở đây chỉ để kiểm tra.

Ba kiểu làm việc trên của 8257 được chọn do hai bit của ($b_{15} b_{14}$) của thanh ghi cơ số đếm MSB (phần cao):

Bảng 7.3: Các kiểu hoạt động của 8257

Kiểu MDA	$b_{15}b_{14}$
Kiểm tra chu kỳ DMA	00
Chu kỳ DMA ghi	01
Chu kỳ DMA đọc	00
Cấm	11

- Bằng bit A_3 có thể điều khiển chọn ra thanh ghi trong DMAC.

- Mạch 8257 có mạch lật FF (F/L)... mà nội dung “ F/L” được gán 1 khi hoàn thành việc ghi hoặc đọc xong 2 byte của thanh ghi 16 trong 8257.

Bảng 7.4: Chức năng đọc ghi các thanh ghi của 8257

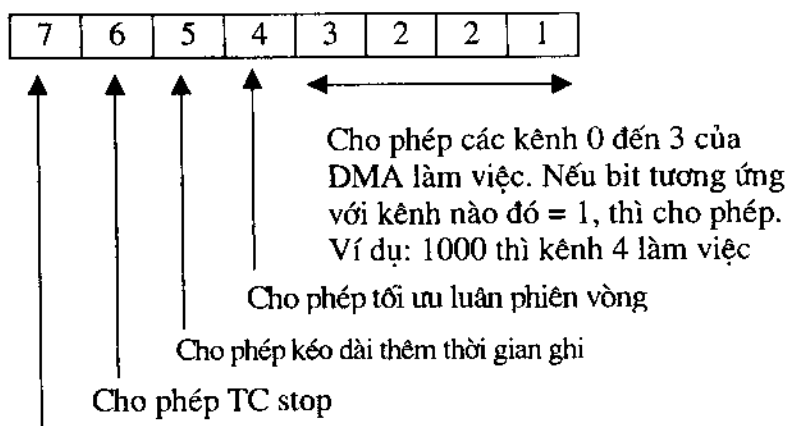
Các thanh ghi bên trong 8257	Byte	Địa chỉ	(F/L)
		A ₃ A ₂ A ₁ A ₀	
a. Thanh ghi địa chỉ của DMA ₀	LSB	0 0 0 0	0
	MSB	0 0 0 0	1
- Thanh ghi cơ số đếm DMA ₀	LSB	0 0 0 1	0
	MSB	0 0 0 1	1
b. Thanh ghi địa chỉ của DMA ₁	LSB	0 0 1 0	0
	MSB	0 0 1 0	1
- Thanh ghi cơ số đếm DMA ₁	LSB	0 0 1 1	0
	MSB	0 0 1 1	1
c. Định chế độ		1 0 0 0	0 (Ghi)
d. Thanh ghi trạng thái		1 0 0 0	0

Ta thấy A₁A₂ chỉ ra chọn một kênh nào trong 4 kênh của DMA.

A₀ phân biệt thanh ghi địa chỉ hay cơ số trong một kênh.

A₃ = 1 Chọn thanh ghi định chế độ.

* *Dạng thức thanh ghi định chế độ*



Cho phép Autoload hay tự nạp điều khiển

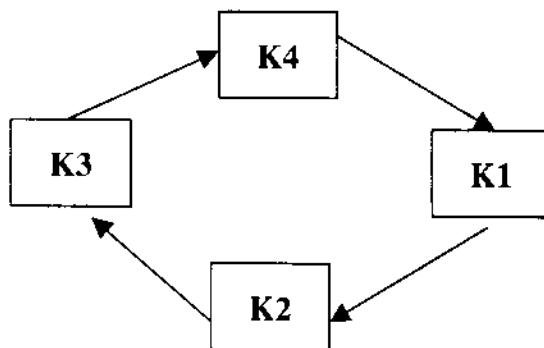
Hình 7.3: Dạng thức thanh ghi định chế độ của 8257

Sau khi Reset thanh ghi này bị xoá, nghĩa là tất cả các kênh DMA bị cấm hết. Sau khi đưa địa chỉ và cơ số đếm, thanh ghi này được nạp chế độ mong muốn.

Bit 4: (Cho phép ưu tiên vòng, luân phiên)

Nếu là 0: Ưu tiên thường: Kênh số 0 rồi đến số 1-> 3

Nếu là 1: Có ưu tiên vòng. Sự ưu tiên được luân phiên cho các kênh như hình sau:



Hình 7.4: Lưu đồ ưu tiên vòng

Bit 5: Cho phép nới rộng thời gian ghi: vì có 1 số thiết bị nhớ bị chậm khi ghi vào, nhưng không phải đến mức phải dừng nhiều trạng thái chờ của 8257, lập trình để kéo dài một chút.

Bit 6: (TC Stop) khi bit 6 =1. Khi đó nếu có xung tại chân TC thì kênh DMA vừa hoạt động sẽ bị cấm.

Bit 7: (Autoload)

Bit 7 = 1 cho phép kênh 2 được dùng lặp lại hay mảng số liệu từ kênh 3 mà không cần thời gian gián đoạn để lập trình nạp dữ liệu. Với nguyên tắc sau:

Nguyên tắc: Các thanh ghi địa chỉ khởi đầu, cơ số đếm,... được ghi bình thường cho kênh số 2, còn các thanh ghi số 3 ghi những số liệu mà kênh 2 chuẩn bị làm việc sau khi mảng kênh 2 chuyển xong có TC báo, tất cả các giá trị từ kênh 3 chuyển sang kênh 2 để làm việc tiếp.

* Dạng thức thanh ghi trạng thái

7	6	5	4	3	2	1	0
0	0	0	Flag	TC3	TC2	TC1	TC0

Trong đó:

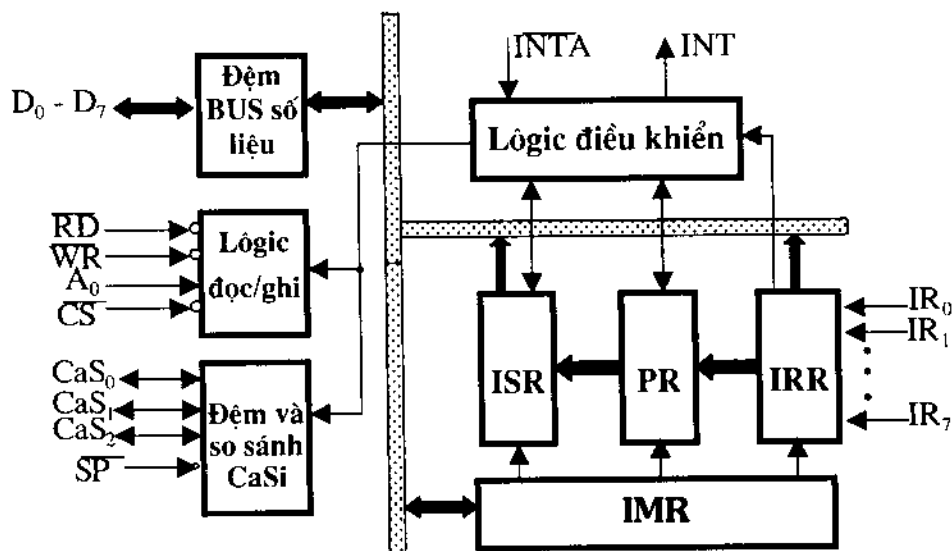
- Flag: Là bit cập nhật. Bằng 1 khi 8257 ở trạng thái cập nhật (hay ở chu kỳ cập nhật). Bằng 0 khi bị Reset hoặc khi xoá Autoload (b7 của bit tương ứng).
- Các bit TC: Thông báo trạng thái TC. Nếu bằng 1 là đã đạt TC. Nếu bằng 0 là sau khi Reset hay sau khi đọc TC.

Chu kỳ cập nhật: Là thời gian mà kênh 2 cập nhật thông tin từ kênh 3, trước khi thực hiện việc chuyển dữ liệu. Lúc đó cờ cập nhật bit 4 của thanh ghi trạng thái được gán bởi 1. Khi chuyển xong bit 4 bị xoá về 0.

III. VI MẠCH ĐIỀU KHIỂN NGẮT 8259

1. Sơ đồ cấu trúc

Vi mạch 8259 có chức năng điều khiển ngắt lập trình được, sơ đồ cấu trúc của vi mạch như sau:



Hình 7.5: Sơ đồ khối cấu trúc của 8259

Trong sơ đồ này, chức năng các khối như sau:

IRR: Đây là khối nhớ tất cả các mức ngắt của các yêu cầu ngắt. Thông thường IR_7 là mức ưu tiên thấp nhất, IR_0 có mức ưu tiên cao nhất.

ISR: Đây là thanh ghi, ghi nhớ các yêu cầu ngắt đang được phục vụ.

PR: Là khối logic xác định yêu cầu có mức ưu tiên nhất được phục vụ (Phân xử lý ưu tiên các yêu cầu ngắt ở IRR). Ngắt có mức ưu tiên cao nhất được chọn đưa sang ISR bằng việc cho bit tương ứng lên 1 khi có INTA.

IMR: Là thanh ghi mà các bit của nó là các mặt nạ để che yêu cầu ngắt tại các chân ngắt tương ứng.

Khối đệm và so sánh CaS_i : CaS_i là 3 bit địa chỉ của hệ 8259 (phân biệt các 8259 nếu có nhiều) trong đó có một 8259 là chủ các 8259 khác là phụ, nếu là chủ CaS_i là ra, còn nếu là phụ thì CaS_i vào.

SP: Báo 8259 làm việc là chủ hay thợ. Nếu đưa vào đó là 1, 8259 là chủ. Nếu đưa 0 vào chân này, 8259 là thợ. Khi 8259 là chủ thì các đầu CaS_i của nó để xác định các 8259 thợ. Khi 8259 chủ đã chọn ra 8259 thợ, thì 8259 thợ sẽ đưa ra địa chỉ chương trình phục vụ ngắt.

Để lập trình cho 8259 làm việc: Có thể tác động vào 2 loại từ điều khiển của 8259 đó là ICW_i và OCW_i .

ICW_i là từ lệnh khởi đầu, OCW_i là từ lệnh thao tác.

ICW_i có thể gồm một chuỗi 2, hoặc 3 từ lệnh dùng để khởi đầu 8259.

OCW_i là từ lệnh để xác định chế độ ngắt của 8259.

2. Nguyên lý hoạt động ghép nối

- Các yêu cầu ngắt IR_i nếu = 1; làm các bit tương ứng của thanh ghi (IRR_i) = 1.
- 8259 sẽ nhận được yêu cầu ngắt giải quyết ưu tiên sau đó mới gửi yêu cầu ngắt giải quyết ưu tiên $INT \rightarrow MP$.

- CPU nhận yêu cầu ngắt INT , nếu ngắt cho phép đưa ra xung $\overline{INTA_1}$ đầu tiên.
- 8259 nhận được $INTA_1$ sẽ ghi nhận yêu cầu ngắt ưu tiên nhất đã chọn (trong 8 khối). Khi đó bit tương ứng được ghi 1 (ISR_i ưu tiên nhất) = 1 đồng thời (IRR_i ưu tiên cao nhất) = 0. Vì để xoá yêu cầu do đã nhớ sang ISR_i . Mặt khác 8259 đưa ra D.BUS mã lệnh "CD" (mã lệnh của lệnh gọi).

- Lúc đó MP (hay CPU) đọc "CD" biết đó là lệnh gọi; đưa ra liên tiếp hai xung $\overline{INTA}$ để đọc nốt 2 byte (địa chỉ) của lệnh gọi.

$INTA_2$: Để đọc địa chỉ phần cấp do 8259 chủ hoặc thợ đưa ra (Nếu không có thợ; chủ đưa ra, nếu có thợ, thợ đưa ra).

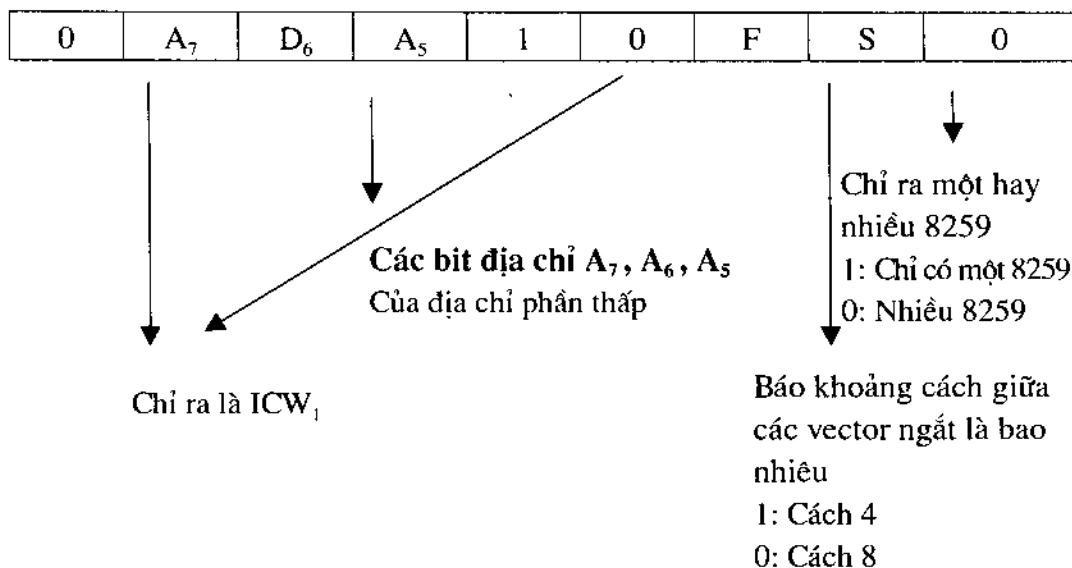
$INTA_3$: Để đọc địa chỉ phần cao.

- Bit (ISR_i cao nhất) = 1 cho tới khi kết thúc chương trình phục vụ ngắt. trước lúc này CPU gửi đến 8259 lệnh EOI để kết thúc ngắt.

Các từ lệnh cho 8259.

* ICW_i : Bao gồm các từ lệnh.

- Từ lệnh ICW_1 :



- Từ lệnh ICW_2 :

1	A ₁₅							A ₈
---	-----------------	--	--	--	--	--	--	----------------

- A₁₅ ÷ A₈ là các bit địa chỉ phân cao.

- Bit 1 tận cùng bên trái báo rằng đây là ICW_2 .

Phản ứng của 8259 khi cho ICW_1 :

Khi đưa vào 8259 ICW_1 thì:

- Các mạch cạnh (ghi nhận) IR_i sẽ bị xoá để chờ đột biến đầu vào nếu có ở $IR_i \rightarrow$ chờ yêu cầu mới.

- IMR bị xoá.

- $IR_7; \dots$ gán mức ưu tiên thấp nhất (mức IR_0 cao nhất).

Véc tơ ngắt:

A ₁₅							A ₈
-----------------	--	--	--	--	--	--	----------------

Đây là các bit địa chỉ cao do ICW_2 cung cấp:

A ₇							A ₀
----------------	--	--	--	--	--	--	----------------

Các bit A_7, A_6, A_5 do D_7, D_6, D_5 và F của ICW_1 chỉ ra.

Các bit A_4, A_3, A_2, A_1, A_0 do 8259 tổ chức.

Việc tổ chức 8259 đối với $(\text{Adr})_i$ có hai dạng khác nhau tùy thuộc (F) và IR_i nào.

IR_i	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
IR_7	A_7	A_6	A_5	1	1	1	0	0
IR_6				1	1	0		
IR_5								
IR_4								
IR_3								
IR_2								
IR_1								
IR_0	A_7	A_6	A_5	0	0	0	0	0
$F = 1$								

IR_i	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
IR_7	A_7	A_6	1	1	1	0	0	0
IR_6			1	1	0			
IR_5								
IR_4								
IR_3								
IR_2								
IR_1								
IR_0	A_7	A_6	0	0	0	0	0	0
$F = 0$								

$D_4 D_3 D_2$ quyết định i :

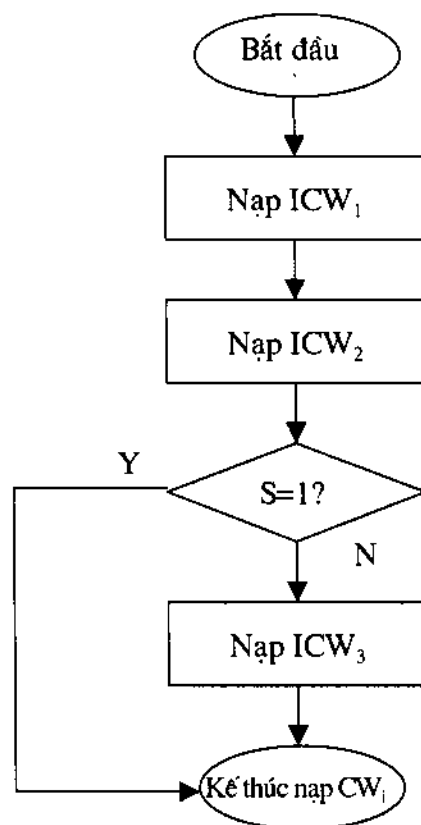
Ta thấy ứng với $F = 1 \rightarrow$ Bit D_2 có trọng số là 4 $\rightarrow$ các véc tơ cách nhau là 4.

$D_5 D_4 D_3$ là i tương tự các véc tơ cách 8.

- Từ lệnh ICW_3 :

Nạp vào thanh ghi thợ. Tùy thuộc vào bit S , $S = (ICW_1)_i = ?$

Để làm việc ở chế độ này định trình như sau:



Hình 7.6: Lưu đồ nạp các thanh ghi ICW_i

- Khi $SP = 1$, có ICW₃ cho 8259 chủ.

1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
---	-------	-------	-------	-------	-------	-------	-------	-------

S_0 đến S_7 báo tương ứng đầu vào thứ i có thợ không

1: có thợ

0: không có thợ

- Khi $SP = 0$: ICW₃ cho 8259 thợ

1	0	0	0	0	0	ID ₂	ID ₁	ID ₀
---	---	---	---	---	---	-----------------	-----------------	-----------------

ID₂ ID₁ ID₀: Mã BCD phân biệt thợ thứ i nối vào IR_i

Ví dụ : 000 thợ 0 nối vào IR_0

* **Từ lệnh OCW:** Đưa vào sau ICW.

- OCW_1 : Lập trình cho thanh ghi IMR:

1	M_7	M_6	M_5	M_4	M_3	M_2	M_1	M_0
---	-------	-------	-------	-------	-------	-------	-------	-------

Đây là mặt nạ ngắt cho các ngắt tương ứng.

Nếu bằng 1: Có mặt nạ che yêu cầu ngắt Iri (tương ứng M_i).

Nếu bằng 0: Không có mặt nạ che yêu cầu ngắt Iri (tương ứng M_i).

- OCW_2 : Lập trình cho thanh ghi IMR:

0	R	SEOI	EOI	0	0	L2	L1	L0
---	---	------	-----	---	---	----	----	----

3 giá trị 0 cố định ở đây chỉ ra đây là ICW_2 .

$L_2 L_1 L_0$: Đây là mã BCD để mã hoá bit thứ i của ISR_i

- Bị xoá khi $EOI = 1$.

- Bị đưa về mức ưu tiên thấp nhất đang được phục vụ bị xoá 0.

SEOI: Kết thúc ngắt đặc biệt.

1: Dừng tổ hợp $L_2 L_1 L_0$ chỉ ra mức ưu tiên sẽ bị đưa về mức thấp nhất.

R: Cho biết làm việc luân phiên không.

1: Không làm việc luân phiên.

0: Không luân phiên.

- OCW_3 :

0	X	ESMM	SMM	0	1	P	ERIS	RIS
---	---	------	-----	---	---	---	------	-----

3 giá trị 001 chỉ ra đây là OCW_3 .

ERIS RIS: Cho phép chọn ra IRR, ISR để đọc cho lần đọc sau:

10: Cho phép chọn IRR

11: Cho phép chọn ISR

ESMM SMM: Chế độ mặt nạ đặc biệt

10: Xóa mặt nạ đặc biệt

11: Lập mặt nạ đặc biệt

*** Các chế độ làm việc**

Đầu tiên đưa ra các từ điều khiển để định nghĩa hoặc có tác dụng nhất định đối với các chế độ làm việc sau:

a. Chế độ ưu tiên cố định

Đặc điểm sau khi khởi đầu (ICW) thì 8259 được định nghĩa là chế độ ưu tiên cố định (không cần OCW nào nữa)

- Cố định IR_0 mức ưu tiên cao nhất, IR_7 mức ưu tiên thấp nhất

- Cơ chế chế độ này: Khi có yêu cầu ngắt được nhận biết thì yêu cầu ngắt có ưu tiên cao nhất sẽ được xác định và vec tơ tương ứng đưa ra D BUS. Bit ISR_i tương ứng sẽ được gán bởi 1 và ISR_i này bằng 1 kéo dài tới khi CPU đưa ra lệnh EOI trước lúc trở về chương trình phục vụ ngắt.

- Khi (ISR_i) = 1 thì các ngắt có ưu tiên thấp hơn đều bị cấm. Nếu $ISR_i = 1$ với mức ưu tiên cao hơn, có thể ngắt chương trình đang được phục vụ.

b. Chế độ luân phiên (Ưu tiên luân phiên)

Đặc điểm: Chế độ này cho phép thay đổi các mức ưu tiên theo kiểu luân phiên hoặc kiểu chỉ đích danh.

Sự luân phiên: Ở chế độ luân phiên thực hiện thay đổi mức ưu tiên bằng cách quay (khi có lệnh quay hay thay đổi luân phiên).

Lúc đó mức ISR_i có mức ưu tiên cao nhất sẽ bị xoá về 0 và mức ưu tiên cao nhất được gán cho yêu cầu ngắt tiếp theo.

Chỉ đích danh: Bằng chương trình có thể thay đổi mức ưu tiên bằng cách chỉ ra mã BCD ứng với mức ưu tiên thấp nhất. Khi đó mức ưu tiên bên trên phần tử chỉ ra là thấp nhất, lại có ưu tiên cao nhất.

c. Chế độ mặt nạ

- Mặt nạ thường: OCW_1 thao túng IMR (có thể cho mặt nạ các yêu cầu). Nếu yêu cầu nào được nhận biết bằng xung INTA thì tất cả các mức ngắt thấp hơn đều bị cấm cho tới khi có EOI.

- Mặt nạ đặc biệt: Cho phép các mức ưu tiên thấp hơn không bị che khi một ngắt đang được phục vụ. Muốn đặt chế độ này thì phải sử dụng OCW_3 , để lập chế độ mặt nạ đặc biệt khi:

$$ESMM = 1$$

$$SMM = 1$$

Và việc mức ưu tiên thấp hơn được cho phép sẽ tồn tại cho tới khi bằng OCW_3 chúng ta xoá chế độ mặt nạ đặc biệt bằng giá trị:

ESMM = 1

SMM = 0

d. Chế độ thăm dò (Polling)

Có thể kiểm tra yêu cầu ngắt bằng cách polling (hỏi vòng) trạng thái có yêu cầu ngắt của 8259. Để hỏi vòng phải thực hiện các thao tác sau:

- Cấm ngắt đối với CPU.
- Viết từ lệnh OCW₃ với P = 1, mỗi khi cần hỏi vòng 8259.
- Đọc từ trạng thái hỏi vòng ngắt do 8259 đưa ra trên D.BUS từ trạng thái có dạng thức sau:

I	x	x	x	x	W ₂	W ₁	W ₀
---	---	---	---	---	----------------	----------------	----------------

Trong đó:

I = 1: có yêu cầu ngắt

I = 0: không có yêu cầu ngắt

W₂, W₁, W₀ là mã BCD của mức ưu tiên cao nhất yêu cầu ngắt

* Đọc trạng thái của 8259:

Một số trạng thái vào của các thanh ghi bên trong có thể đọc bằng cách kết hợp RD với OCW₃.

IMR: Khi A₀ = 1

ISR: Khi OCW₃ có ERIS = 1, RIS = 1

IRR: Khi ERIS = 1, RIS = 0

Polling: Khi OCW₃ có P = 1

* Nối tầng các 8259:

Có thể nối ghép nhiều 8259 với nhau để có thể xử lý nhiều ngắt một lúc. Ví dụ: Một 8259 chủ và 8259 thợ có thể xử lý được 64 mức ưu tiên khác nhau. Trong đó các chân INT của 8259 thợ thứ i, mắc vào chân IR_i của 8259 chủ. Khi đó tất cả CaSi của thợ cùng nối vào CaSi của chủ.

* Nguyên lý hoạt động

Khi yêu cầu ngắt từ 8259 thợ đưa tới 8259 chủ được chấp nhận thì khi có xung INTA đầu tiên (INTA₁) 8259 chủ đưa ra D.BUS lệnh "CD" tạo điều kiện cho 8259 thợ tương ứng đưa ra địa chỉ chương trình phục vụ ngắt trong khoảng

thời gian tồn tại hai xung $INTA_2$ và $INTA_3$ mà CPU phát ra. Cuối cùng MP nhận địa chỉ này từ D.MUS và phục vụ chương trình ngắt tương ứng.

CaSi của chủ đưa ra địa chỉ của thợ trong suốt quá trình có xung INTA.

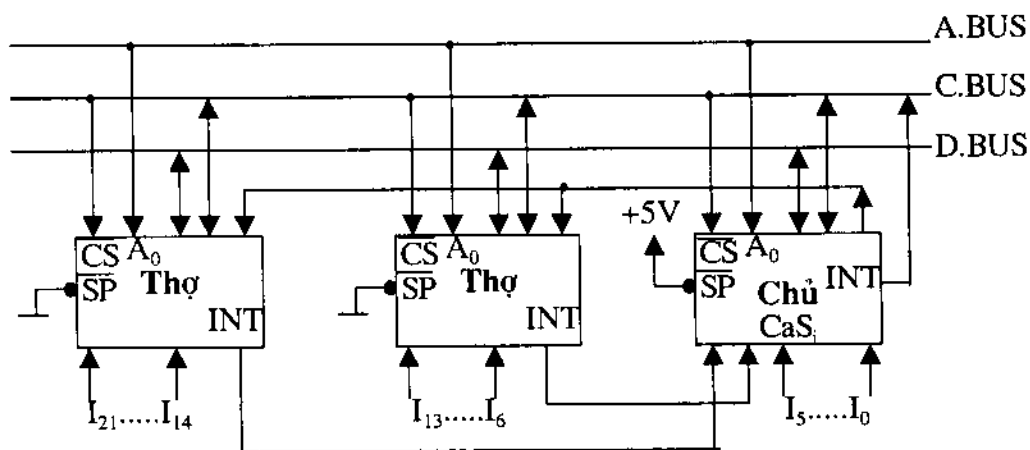
Mỗi 8259 trong hệ đều phải khởi đầu và có thể làm việc ở các chế độ khác.

Mỗi lệnh EOI phải được đưa ra hai lần: một lần cho 8259 chủ và một lần cho 8259 thợ tương ứng.

Mỗi xung CS của 8259 tùy thuộc vào từng mạch và từng 8259.

Ví dụ:

Sử dụng 3 mạch 8259 một chủ, 2 thợ phục vụ 22 yêu cầu ngắt khác nhau. Trong đó có 16 yêu cầu được phục vụ bởi hai 8259 thợ và 6 yêu cầu còn lại được phục vụ bởi 8259 chủ.

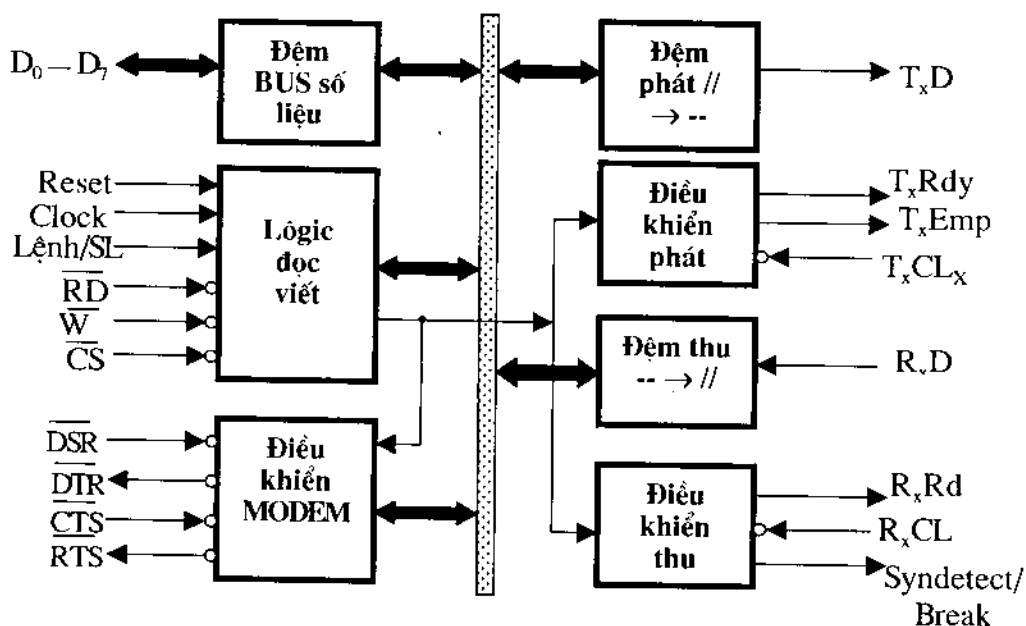


Hình 7.7: Sơ đồ ghép nối các 8259

IV. VI MẠCH ĐIỀU KHIỂN GHÉP NỐI 8251

1. Sơ đồ cấu trúc

Vi mạch 8251 có chức năng điều khiển ghép nối giữa CPU và các đường thu/phát dữ liệu nối tiếp.



Hình 7.8: Sơ đồ khối cấu trúc vi mạch 8251

2. Nguyên lý hoạt động ghép nối

Mạch này phục vụ cho việc nối ghép giữa CPU với các đường truyền tin nối tiếp, đây có thể là bộ nhận hoặc phát thông tin đồng bộ hay dị bộ.

Nếu truyền tin trong chế độ dị bộ tốc độ có thể đạt được 50Bd đến 2400Bd. Khi truyền đồng bộ tốc độ có thể đạt được là: 3800Bd - 9600Bd.

C/D chân điều khiển số liệu thường nối với A_0 của CPU. Sau đây là tín hiệu chọn các thanh ghi từ CPU.

Bảng 7.5: Giải mã điều khiển 8251

A_0	$\overline{RD}$	$\overline{WR}$	Chọn ra
0	0	1	Đệm số liệu phần thu
0	1	0	Đệm số liệu phần phát
1	0	1	Chọn thanh ghi trạng thái
1	1	0	Chọn thanh ghi điều khiển

Khi đưa các xung đồng hồ T_xCLK , R_xCLK thì tần số bằng bội 1, 16, 61 tốc độ truyền.

Chân $T_x\text{Empty}$: Báo tất cả các ký tự được nhận từ CPU đã phát hết.

Chân $T_x\text{Rdy}$: Báo đệm phát sẵn sàng nhận ký tự khác để phát.

Chân $R_x\text{Rdy}$: Báo đệm phân thu có ký tự để CPU đọc vào.

SynDetect/BreakDetect: Khi chân $R_x\text{Rdy}$ giữ nguyên trong khoảng thời gian ứng với 2 ký tự, thì tại chân này phát xung báo gián đoạn (trong quá trình truyền dữ liệu). Chân này có nhiệm vụ phát xung khi nhận được ký tự đồng bộ, trong khi truyền đồng bộ.

Sau khi Reset đồng thời phải đưa vào thanh ghi điều khiển hai từ điều khiển một lúc, 1 từ là từ chế độ, từ thứ 2 là từ lệnh kế tiếp nhau. Và phải có liên lạc giữa CPU và 8251 để xem nó có được phép làm việc hay không. Sự nối ghép này có thể theo hai phương pháp sau:

- Phương pháp ngắt: $T_x\text{Rdy}$, $R_x\text{Rdy}$ nối với INTR của CPU.
- Phương thức hỏi vòng (polling): Xét các chân của mạch này $T_x\text{Rdy}$, $R_x\text{Rdy}$ hoặc đọc thanh ghi trạng thái của mạch (dễ dàng hơn).

Dạng thức từ chế độ

8 bit của từ chế độ có ý nghĩa như sau:

B_1B_0 : Quyết định tốc độ truyền, vì hai bit này quy định một hệ số. Mà tốc độ dữ liệu truyền được tính bằng tần số xung đồng hồ chia hệ số này.

00: hệ số 1

01: hệ số 16

11: hệ số 64

B_3B_2 : Cho biết độ dài ký tự cần truyền.

00: độ dài 5 bit

01: độ dài 6 bit

10: độ dài 7 bit

11: độ dài 8 bit

B_4 : Cho phép kiểm tra parity hay không

1: cho phép kiểm tra parity

0: không cho phép kiểm tra parity

B_5 : Quyết định việc tạo parity chẵn hay lẻ.

1: tạo parity chẵn

0: tạo parity lẻ

B_7B_6 : Cặp bit này có ý nghĩa tùy thuộc 8251 làm việc ở chế độ nào, đồng bộ hay dị bộ.

Nếu ở chế độ đồng bộ:

Bit B_6 : Để phát hiện đồng bộ ngoài

Nếu là 1 đã phát hiện được xung SynDetect

Nếu là 0 phát xung SynDetect

Bit B_7 : Báo số ký tự dùng để đồng bộ

Nếu bằng 1: có 1 ký tự đồng bộ

Nếu bằng 0: có 2 ký tự đồng bộ

Nếu ở chế độ không đồng bộ thì B_6B_7 hợp thành 2 bit có ý nghĩa như sau:

Nếu là 00: là tổ hợp cấm

Nếu bằng 01: báo có 1 bit STOP

Nếu bằng 10: báo độ dài 1,5 bit STOP

Nếu bằng 11: báo độ dài 2 bit STOP

Dạng thức từ lệnh

Gồm 8 bit với ý nghĩa như sau:

B_0 : Cho phép hoặc không cho phép phát:

1: Cho phép phát.

0: Không cho phép phát.

B_1 : Nếu bằng 1 thì xoá chân DTR về 0.

B_2 : Cho phép thu hay không ?

1: Cho phép.

0: Không cho phép.

B_3 : Nếu bằng 1, thì xoá chân T_xD về 0.

B_4 : Để xoá các cờ lỗi.

Nếu bằng 1 xoá tất cả các cờ lỗi.

B_5 : Nếu bằng 1 thì xoá chân RST về 0.

B_6 : Nếu bằng 1 thì xoá trạng thái nội bộ 8251, như được Reset từ CPU.

B_7 : Nếu bằng 1 cho phép tìm đồng đội khi thấy.

Dạng thức từ trạng thái

DSR	SYNDE	FE	OE	PE	TxEmpty	RxRdy	TxRdy
-----	-------	----	----	----	---------	-------	-------

Chúng ta thấy một số bit có ý nghĩa giống như các chân tín hiệu tương ứng. Chỉ có một số chân khác như sau:

PE: Phát hiện lỗi parity

1: Báo cáo lỗi

0: Không có lỗi

OE: Báo tràn nếu bit này bằng 1

FE: Báo lỗi khung truyền

1: Khi không tìm thấy bit STOP

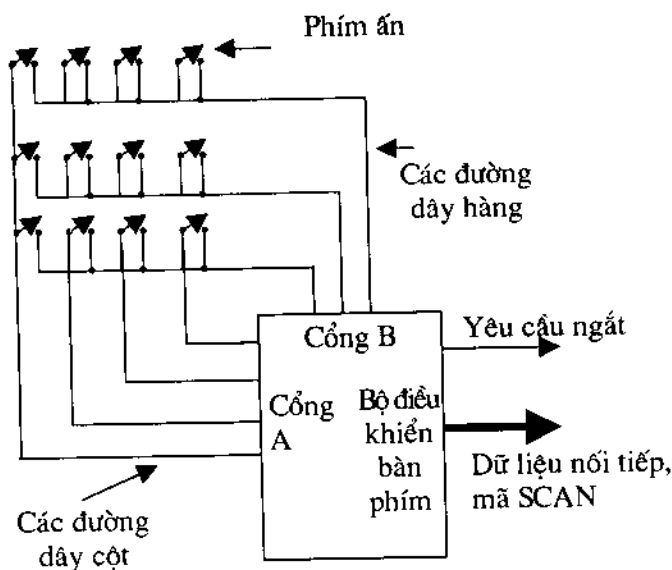
0: Khi có ER bằng 1

Ứng dụng: Có thể sử dụng mạch này để nhận tín hiệu từ bàn phím. Vì dữ liệu từ bàn phím là nối tiếp phải biến đổi thành tín hiệu song song trước khi có thể xử lý bằng CPU.

V. MỘT SỐ GHÉP NỐI CƠ BẢN

1. Ghép nối với bàn phím

Ngày nay bàn phím vẫn là một thiết bị vào chuẩn, mà mọi máy tính đều phải sử dụng. Trên đó có nhiều phím ấn, mỗi phím có một chức năng nhất định như để vào kí tự, phím điều khiển, phím con trỏ. Sau đây chúng ta sẽ tìm hiểu hoạt động của bàn phím trong sự nối ghép với máy tính.



Hình 7.9: Sơ đồ ghép nối bàn phím

Trên hình chúng ta thấy mỗi phím được nằm trên một hàng và một cột nhất định. Các đường dây hàng và cột được nối với các cổng của một vi mạch LSI.

Mạch này có chức năng là phải đưa ra các tín hiệu quét các hàng và cột thông qua các cổng của nó để phát hiện phím ấn. Khi một phím được ấn, thông tin mà vi mạch nhận được là tọa độ hàng, cột của phím ấn. Với những thông tin này mạch LSI phải tạo ra một mã đặc trưng cho phím ấn, mã này gọi là mã SCAN, mã này được cất ra bộ đệm của bản thân bàn phím.

Để phân biệt thời điểm ấn và nhả phím, vi mạch điều khiển bàn phím thường tạo ra hai mã SCAN khác nhau tại hai thời điểm. Khi ấn phím tạo ra mã MAKE, khi nhả phím tạo ra mã BREAK. Khi đã có mã trong vùng đệm, vi mạch điều khiển gửi yêu cầu ngắt tới CPU. Khi CPU nhận được ngắt bàn phím nó phải gọi chương trình phục vụ ngắt (09H) để lấy mã SCAN từ bàn phím thông qua cổng, biến đổi thành mã ASCII và cất ra bộ đệm bàn phím trong bộ nhớ chính.

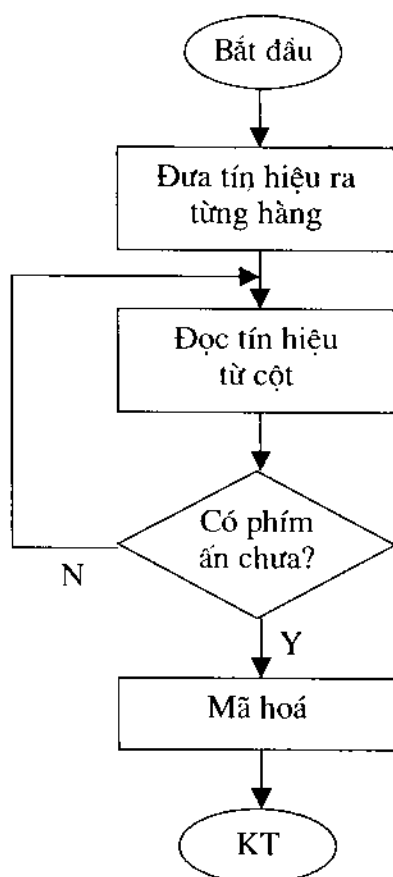
Thông thường bộ đệm của bàn phím lưu trữ được 20 ký tự, nhưng đó cũng không phải nhỏ quá vì khả năng nhận phím của CPU rất nhanh so với tốc độ gõ phím của người sử dụng. Dữ liệu lấy được từ bàn phím thông qua đường nối tiếp, nên chỉ có một đường dây dữ liệu trong cáp bàn phím.

Kết cấu điểm tiếp xúc tại các phím cũng khác nhau tùy thuộc loại bàn phím. Có thể là tiếp xúc loại điện dung (khi ấn phím làm thay đổi điện dung một tụ điện nào đó), hay loại công tắc (khi ấn phím đường giây hàng được nối thông với đường giây cột). Nhưng xét về mặt bản chất là khi ấn phím phải phát sinh ra một tín hiệu để bộ điều khiển bàn phím phát hiện ra. Vi mạch điều khiển bàn phím thông dụng IC 8048 của hãng Intel.

Hình 7.10 là lược đồ chương trình đọc phím mà bộ điều khiển bàn phím phải thực hiện:

Để tìm ra được một phím ấn, bộ điều khiển đầu tiên phải đưa ra tín hiệu chọn một hàng, sau đó nó sẽ kiểm tra tín hiệu nhận được từ các cột để phát hiện phím ấn trên một cột nào đó, trong hàng đã chọn. Khi phát hiện được phím ấn nó phải mã hoá phím ấn thành mã SCAN tương ứng.

Bộ mã SCAN của các bàn phím khác nhau có thể khác nhau.



Hình 7.10: Lược đồ chương trình đọc bàn phím

2. Ghép nối với màn hình

Từ khi máy tính ra đời cho đến nay màn hình là thiết bị ra không thể thiếu đối với mỗi máy tính. Trước khi tìm hiểu về sự nối ghép của màn hình với hệ thống, chúng ta xét qua sự hiển thị hình ảnh trên màn hình điện tử.

Để hiển thị hình ảnh lên màn hình người ta dùng một tia điện tử quét lên các điểm của một màn phospho. Tuỳ thuộc vào sự thay đổi cường độ tia điện tử mà xuất hiện các điểm sáng tối trên màn hình và do đó xuất hiện hình ảnh trên màn hình. Chính vì vậy, để hiển thị hình ảnh lên màn hình tia điện tử phải chịu tác động của nhiều tín hiệu điều khiển như: Tín hiệu quét dòng để kéo tia điện tử quét theo chiều ngang theo từng dòng quét, tín hiệu quét màn hình để kéo tia điện tử quét theo chiều dọc theo từng màn hình, tín hiệu Video để biểu diễn hình ảnh.

Thông thường tần số quét dòng là 15750Hz. Như vậy chu kỳ quét một dòng là $63.5\mu s$, trong đó thời gian quét thuận là $50\mu s$, thời gian quét ngược là $13.5\mu s$. Tần số quét màn là 60Hz. Như vậy một chu kỳ quét màn là 16.7ms, trong đó thời gian quét thuận là 15.45ms, thời gian quét ngược là 1.25ms. Trong đó thời gian quét ngược dòng hay màn người ta phải đập tia quét ngược để tránh nhiễu do chúng gây ra.

Thông thường màn hình được điều khiển ở hai chế độ khác nhau đó là:

- Chế độ văn bản

Trong chế độ này màn hình chỉ có thể hiển thị được các ký tự. Với những màn hình ngày nay có thể hiển thị 25 dòng 80 cột ký tự trên một màn hình. Khi muốn hiển thị một ký tự, tia điện tử phải tác động vào một ma trận các điểm tương ứng trên màn hình. Làm các điểm trong đó sáng tối để biểu diễn được ký tự tương ứng. Thường ma trận điểm hiển thị có kích thước 5x7 nhưng để ngăn cách giữa các ký tự trên các hàng các cột nên kích thước ma trận điểm thường là 7x9.

Đặc điểm:

Trong chế độ văn bản thông tin lưu trữ cho một màn hình rất ít, vì để lưu trữ thông tin mỗi ký tự chỉ cần lưu trữ ASCII của nó (1byte)

Như vậy, để lưu trữ thông tin của một màn hình chỉ cần 25x80 2000 bytes. Với chế độ này chúng ta không thể hiển thị các hình ảnh đồ họa phức tạp.

- Chế độ đồ họa

Đây là chế độ mà màn hình có thể hiển thị các hình ảnh đồ họa phức tạp vì khi đó máy tính có thể điều khiển độc lập từng điểm sáng trên màn hình. Tùy theo từng hệ thống mà độ phân giải về màu và nét là khác nhau, thông thường hiện nay các máy tính có thể hiển thị được 16 hoặc 256 màu với độ phân giải 640x480 điểm hoặc 1024x768 điểm.

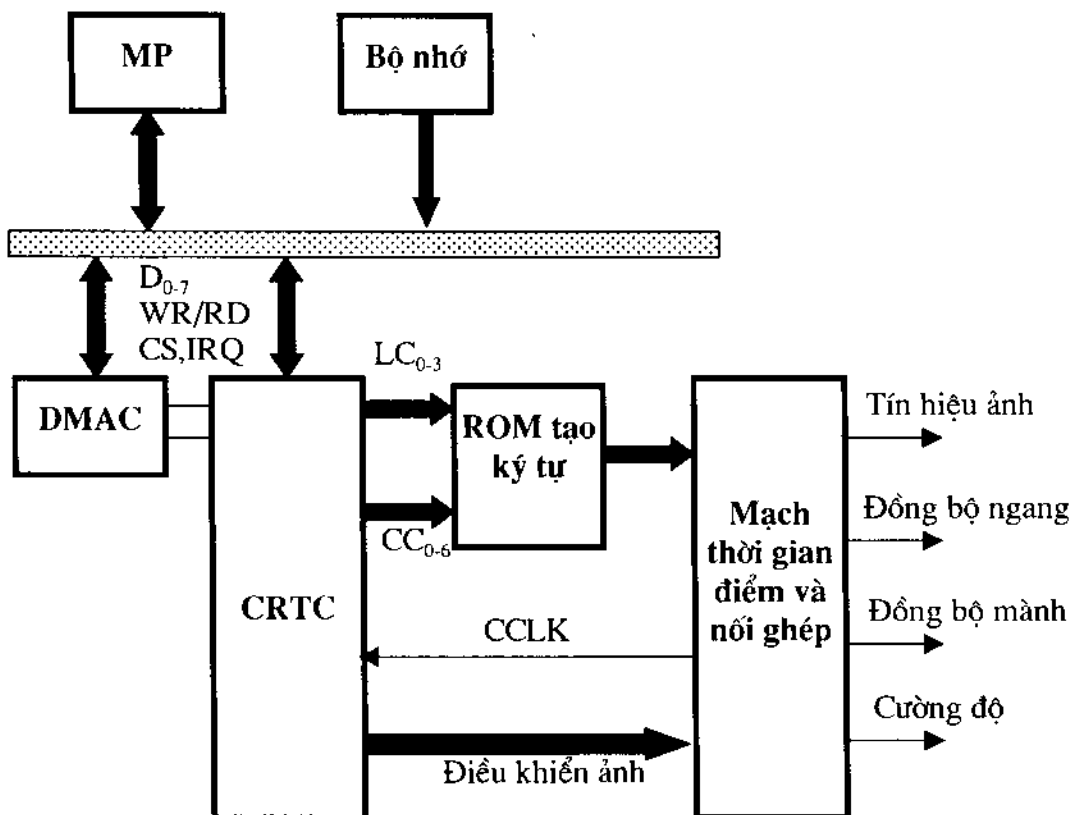
Đặc điểm:

Trong chế độ này thông tin lưu trữ về một màn hình rất lớn. Ví dụ để lưu trữ thông tin về một màn hình độ phân giải 640x480 16 màu cần vùng nhớ dung lượng 640x48x4 (bít). Chúng ta thấy lớn hơn rất nhiều so với chế độ văn bản.

Trong chế độ này chúng ta có thể hiển thị được những hình ảnh đồ họa phức tạp với màu sắc phong phú.

- Sự nối ghép giữa màn hình và CPU

Sơ đồ nối ghép màn hình



Hình 7.11: Sơ đồ khối ghép nối màn hình

Chức năng các khối trong sơ đồ sau:

Bộ nhớ: Để lưu trữ thông tin về hình ảnh được hiển thị trên màn hình, thường gọi là VIDEO RAM.

DMAC: Bộ điều khiển vào ra trực tiếp (đã xét ở phần trước), để chuyển thông tin hình ảnh từ bộ nhớ vào đệm của CRTC, trước khi nó được hiển thị.

CRTC: (CRT Controller) bộ điều khiển màn hình. Đây là bộ phận quan trọng nhất, để điều khiển sự phối ghép giữa hệ thống và màn hình thông qua VIDEO CARD.

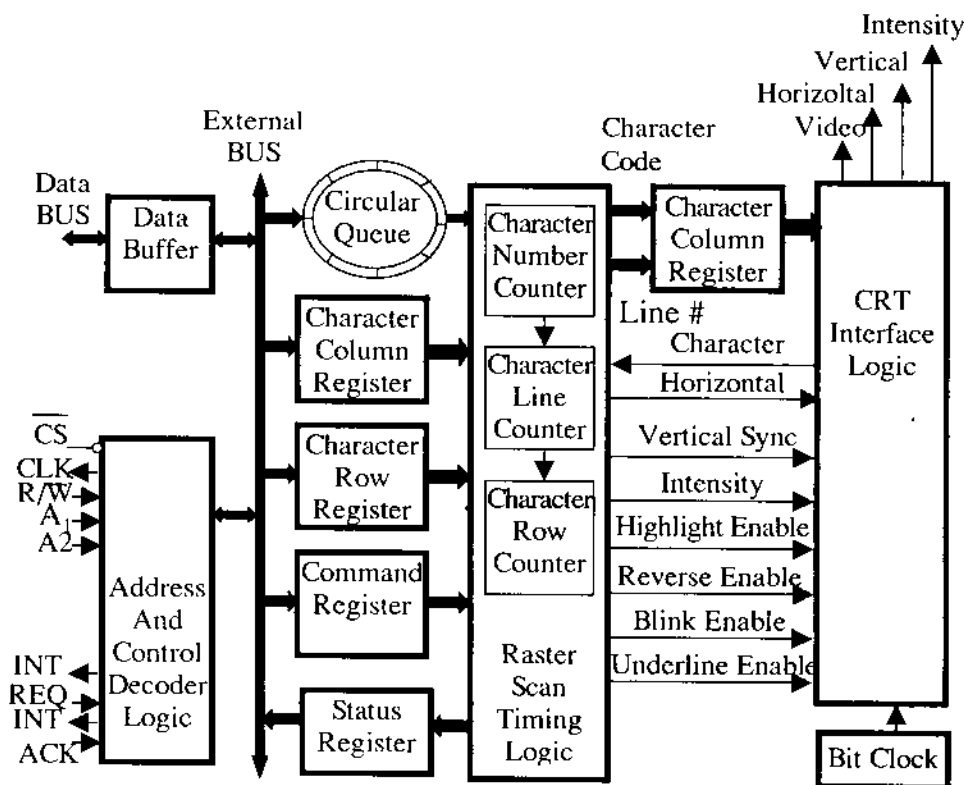
ROM tạo ký tự: Trong chế độ văn bản các ký tự hiển thị khi tia điện tử quét lên một ma trận điểm trên màn hình làm cho các điểm trong ma trận này sáng tối tùy theo. Như vậy để mỗi ma trận điểm có thể hiển thị các ký tự khác nhau phải có các mẫu điểm sáng tối tương ứng khác nhau. Mục đích của ROM

tạo ký tự ở đây là đưa ra mẫu bit theo từng dòng của ma trận điểm tương ứng với mã ký tự cần hiển thị. Sau đó các mẫu bit này được qua bộ nối ghép biến đổi thành các mẫu điểm dạng nối tiếp và điều khiển tia điện tử hiển thị.

Mạch thời gian điểm và ghép nối:

Mạch này để ghép nối trực tiếp với màn hình, từ đây các tín hiệu ảnh, cường độ, đồng bộ dòng, đồng bộ màn hình được đưa tới CRT.

Để hiểu rõ hơn sự ghép nối, chúng ta xét sơ đồ cấu trúc của 1 CRTC như hình sau:



Hình 7.12: Sơ đồ cấu trúc của CRTC

Chức năng các khối trong sơ đồ trên như sau:

- Khối logic giải mã điều khiển và địa chỉ: (Address and controll Decoder Logic). Khối này để thực hiện nhiệm vụ giao tiếp với hệ thống như CPU, DMAC.

Chính vì vậy giao diện của khối này với bên ngoài gồm các tín hiệu: chọn vô CS, yêu cầu ngắt INT REQ, nhận trả lời ngắt INT ACK, yêu cầu DMA DMA REQ, nhận trả lời DMA DMA ACK.

- Đệm dữ liệu: (Data Buffer): Là giao diện của CRTC với D.BUS
- Hàng đợi vòng: (Circular Queue) Đây là nơi để lưu trữ hàng kí tự trước khi hiển thị ra màn hình. Thường nó có thể lưu trữ 2 hàng ký tự. Khi một hàng đang được hiển thị thì hàng tiếp theo đã được nạp, điều này khiến hoạt động hiển thị nhịp nhàng và nhanh hơn.
- Các thanh ghi hàng, cột ký tự: (Character Row/Column Register). Để lưu trữ vị trí các ký tự cần hiển thị.
- Thanh ghi lệnh: (Command Register). Để nhận lệnh từ hệ thống.
- Thanh ghi trạng thái: (Status Register). Để lưu trữ thông tin trạng thái của CRTC.
- Bộ đếm ký tự: (Character Number Counter). Để trở đến ký tự cần hiển thị trong hàng.
- Bộ đếm dòng ký tự: (Character Line Counter). Để xác định thứ tự dòng quét của tia điện tử trên các ký tự.
- Bộ đếm hàng ký tự: (Character Row Counter). Để trở đến các hàng ký tự.
- Khối logic đồng bộ và quét: (Raster Scan Timing Logic). Để điều khiển việc quét và đồng bộ sự hiển thị các ký tự trên màn hình. Từ đây thường có các ký tự điều khiển ảnh tới bộ nối ghép như: đồng bộ dòng, đồng bộ màn hình, cho phép Highlight, tín hiệu video đảo, cho phép gạch chân,...

Hoạt động ghép nối

- Trong chế độ văn bản

Để hiển thị một dòng ký tự ra màn hình hoạt động của hệ thống được thực hiện các bước như sau:

- Đầu tiên CRTC yêu cầu DMAC chuyển một hàng ký tự cần hiển thị dưới dạng mã ASCII từ Video RAM vào hàng đợi vòng của nó.
- Tại mỗi thời điểm: Bộ đếm ký tự của CRTC xác định một ký tự trong hàng cần hiển thị, bộ đếm dòng quét xác định dòng điện tử quét hiện hành cho ký tự tương ứng. Khi đó mã ASCII của ký tự và số hiệu dòng quét, tạo ra các bit địa chỉ cao và các bit địa chỉ thấp tác động vào ROM tạo ký tự. Đầu ra của ROM tạo ký tự là một từ gồm các bit biểu diễn mẫu điểm trên dòng tương ứng của ký tự cần hiển thị.
- Do bộ đếm ký tự có tần số thường gấp 80 lần tần số bộ đếm dòng (bằng số ký tự trên dòng), nên khi bộ đếm dòng chưa thay đổi thì bộ đếm ký tự lại chuyển sang ký tự khác và như vậy ở thời điểm tiếp theo, mẫu bit tương ứng

của ký tự tiếp theo trên cùng dòng quét lại được đưa ra khỏi ROM tạo ký tự. Quá trình được tiếp diễn cho tới khi bộ đếm ký tự đếm hết các ký tự trên một dòng thì bộ đếm dòng mới được tăng lên 1. Khi các dòng quét trên một hàng ký tự được hoàn thành thì một hàng ký tự được xuất hiện trên màn hình.

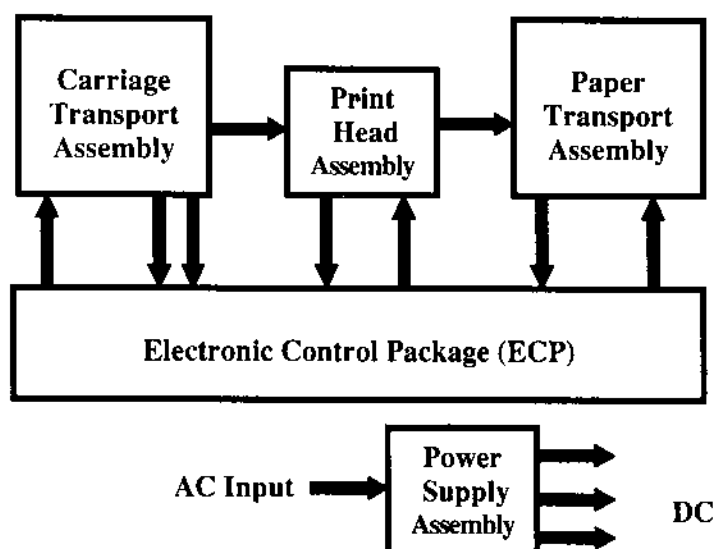
Đầu ra ROM tạo ký tự là tín hiệu song song qua bộ nối ghép được biến đổi thành tín hiệu nối tiếp sau đó được đưa tới màn hình. Nhiệm vụ của CRTC luôn phải làm tươi màn hình bằng cách viết đều đặn thông tin trong Video RAM ra màn hình với tần số 50, 60Hz để tránh rung hình.

- Trong chế độ đồ họa

Trong chế độ đồ họa sự ghép nối đơn giản hơn vì không phải qua bộ tạo ký tự. Vì thông tin trong Video RAM đã là thông tin của các điểm sáng.

3. Ghép nối với máy in

Máy in cũng là một thiết bị ra rất quan trọng trong hệ thống máy tính. Ngày nay có rất nhiều loại máy in như: máy in kim, máy in Laser, máy in phun... Nhưng về mặt nguyên tắc các máy in đều bao gồm các khối chức năng được mô tả như sau:



Hình 7.13: Sơ đồ khối máy in

Chức năng các khối trong sơ đồ:

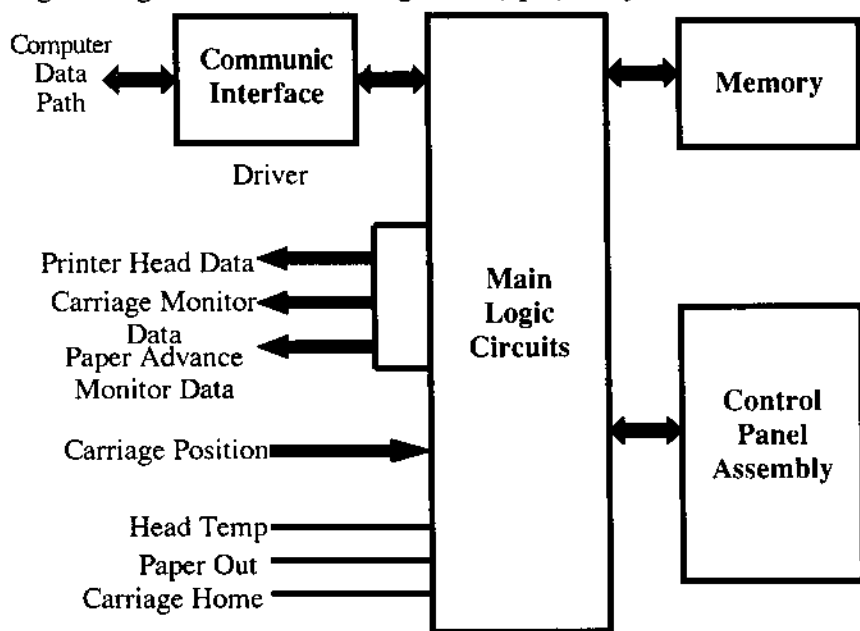
- Paper Transport: Đây là một bộ nhận để nạp giấy. Nhiệm vụ của nó là phải lấy từng tờ giấy ra khỏi khay đựng, kéo tờ giấy lướt qua đầu in và cuộn cùng đưa tờ giấy đã được in ra ngoài.

- **Print heads:** Đây là các thiết bị để in những văn bản hay hình ảnh trên bề mặt giấy. Có rất nhiều công nghệ in được sử dụng ở đây tùy thuộc loại máy in như: Máy in kim (sử dụng các đầu kim), máy in phun (sử dụng những súng phun mực), máy in Laser (dùng tia laser).

- **Carriage Transport:** Đây là bộ phận để dịch chuyển bộ phận in khi in. Ví dụ khi máy in kim thì bộ phận này phải dịch chuyển đầu kim từ trái qua phải và ngược lại trên mỗi dòng in...

- **Power Supply:** Đây là bộ phận cung cấp nguồn. Mục đích để biến đổi điện áp lưới xoay chiều thành một số mức điện áp một chiều cung cấp cho các bộ phận điện tử hoạt động. Đầu ra của nó thường là các điện áp một chiều 5V, 12V, 24V đặc biệt với máy in tĩnh điện, điện áp cung cấp tới 6000V hoặc hơn.

- **Electronic Control Package:** Đây là khối điều khiển điện tử có vai trò chủ chốt điều khiển hoạt động chung của các bộ phận trong máy in và để giao tiếp với hệ thống bên ngoài. Cấu trúc chung của bộ phận này như hình sau:



Hình 7.14: Sơ đồ khối khối điều khiển trong máy in

Chức năng các khối trong sơ đồ

- **Communic Interface:** Bộ phận này để giao tiếp giữa khối điều khiển với máy tính. Bao gồm dữ liệu, các tín hiệu điều khiển từ CPU, các tín hiệu móc nối.

- Memory: Đây là vùng nhớ để lưu trữ những thông tin nhận được từ máy tính trước khi in. Chính vì vậy, với những máy in có bộ phận nhớ lớn, thao tác in rất nhanh vì những thông tin cần in được nạp hết ra bộ nhớ của bản thân máy in và lúc đó máy tính có thể chạy chương trình khác, trong khi máy in đang in.

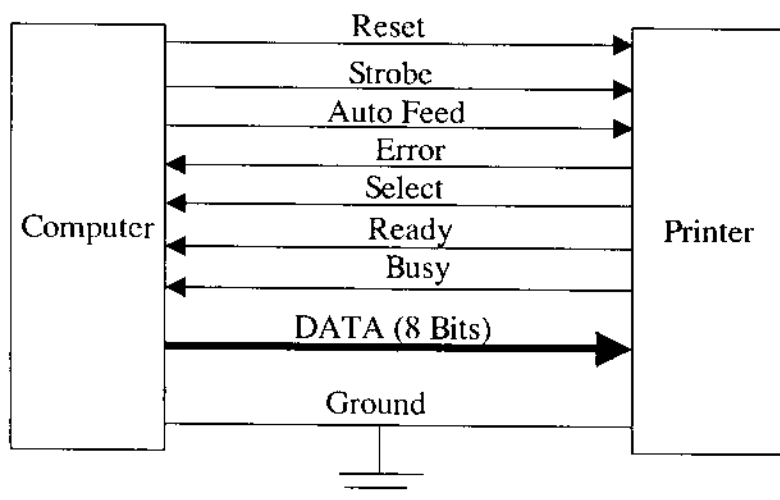
- Control Panel: Đây là bảng điều khiển của máy in, giúp người sử dụng có những thao tác trực tiếp với máy in: Bật nguồn, kéo giấy, chọn font chữ...

- Main Logic Circuits: Đây là bộ phận đóng vai trò điều khiển chung hoạt động của khối điều khiển. Bộ phận này nhận thông tin điều khiển từ CPU, từ các nút điều khiển, từ trạng thái các bộ phận in. Sau đó đưa ra các tín hiệu điều khiển tới các bộ phận tương ứng.

Sự ghép nối giữa máy in với hệ thống

Sự ghép nối của các máy in với hệ thống có thể được thực hiện trên hai cổng: cổng song song và cổng nối tiếp.

- Nối ghép qua cổng song song

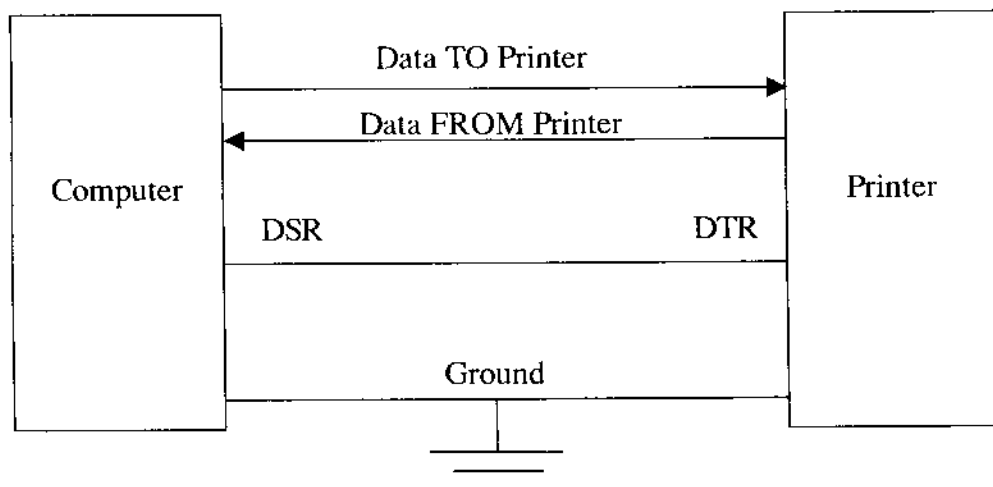


Hình 7.15: Sơ đồ ghép nối máy in qua cổng song song

Qua sơ đồ chúng ta thấy dữ liệu được đưa qua cổng song song 8 bit tới máy in. Tuy nhiên, để hệ thống hoạt động tin cậy, có nhiều tín hiệu được móc nối giữa máy in và máy tính. Máy tính đưa tới máy in các tín hiệu điều khiển (Reset, Strobe, Auto Feed). Máy in hồi tiếp lại máy tính những thông tin trạng thái (Error, Select, Ready, Busy).

Với cách nối như vậy tốc độ truyền dữ liệu tới máy in rất nhanh, có thể đạt tới 8000 bits/s. Nhưng với cách này mà nối với máy in ở xa thì rất phức tạp vì số lượng dây nhiều và tốn kém.

- Nối ghép qua cổng nối tiếp



Hình 7.16: Sơ đồ ghép nối máy in qua cổng nối tiếp

Trên sơ đồ chúng ta thấy có hai đường tín hiệu nối ghép giữa máy tính và máy in. Một để truyền dữ liệu từ máy tính tới máy in. Một để truyền dữ liệu từ máy in về máy tính. Như vậy dữ liệu ở đây được liên kết theo hai chiều (bidirectional data link). Tại mỗi thời điểm trên mỗi đường dây, chỉ một bit thông tin có thể được truyền, như vậy để gửi một kí tự từ máy tính sang máy in phải mất 8 lần truyền.

Ngoài ra để có thể đồng bộ được quá trình truyền dữ liệu, hệ thống phải truyền thêm các bit đồng bộ trước và sau các bit dữ liệu và để sửa lỗi thường có thêm bit chẵn lẻ parity. Các bit dữ liệu và các bit phụ này tạo ra các khung truyền trên đường truyền nối tiếp mà máy in phải nhận biết được. Để tạo sự móc nối giữa máy tính và máy in theo cách này có hai phương pháp:

4. Ghép nối với ổ đĩa

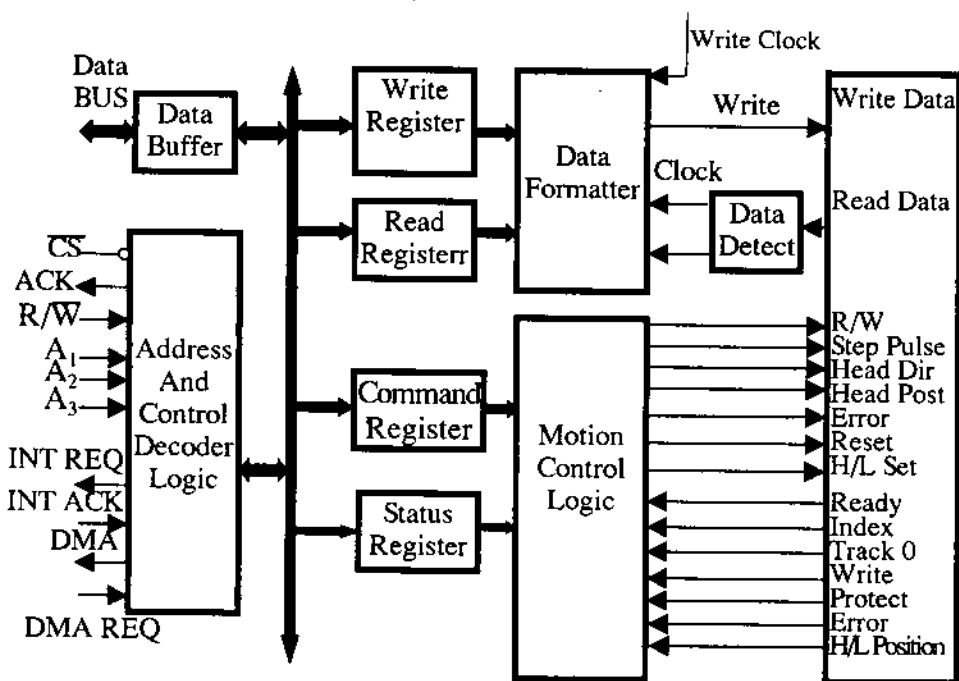
Ổ đĩa là một thiết bị vào ra khá thông dụng của các máy tính ngày nay. Thực tế xuất hiện một số loại như: ổ đĩa mềm, ổ đĩa cứng, ổ đĩa CD-ROM... Trong phần này ta sẽ nghiên cứu ghép nối với ổ đĩa mềm, còn các ghép nối với ổ đĩa khác sẽ được nghiên cứu ở các tài liệu chuyên ngành khác.

Ghép nối ổ đĩa mềm

Đĩa mềm thường là một tấm nhựa trên đó phủ lớp từ tính. Thông tin được ghi thành các rãnh đồng tâm (track), trên mỗi rãnh thông tin lại được tổ chức thành các cung gọi là các sector. Ngoài cùng có một lớp vỏ bảo vệ. Khi cần đọc/ghi đến một sector trên đĩa, đầu từ đọc/ghi sẽ được dịch đến rãnh tương ứng sau đó đĩa được quay tròn để sector cần truy cập lướt qua đầu từ.

Đĩa mềm thường có hai loại: Loại kích thước 5.25" dung lượng 1.2 MB và loại kích thước 3.5" dung lượng 1.44 MB.

Để đọc thông tin từ đĩa mềm phải có ổ đĩa mềm. Trong đó có các đầu từ đọc/ghi và các bộ phận khác như: Motor làm quay đĩa, động cơ bước làm dịch chuyển đầu từ, giao diện nối ghép... Sự nối ghép của bộ điều khiển đĩa của ổ đĩa mềm với hệ thống được mô tả ở hình sau.



Hình 7.17: Sơ đồ ghép nối ổ đĩa mềm

- Khối logic giải mã điều khiển và địa chỉ (Address and Control Decoder Logic). Khối này để thực hiện nhiệm vụ giao tiếp với hệ thống như một CPU, DMAC. Chính vì vậy mà giao diện của khối này với bên ngoài gồm các tín hiệu: Chọn vỏ CS, yêu cầu ngắt INT REQ, nhận trả lời ngắt INT ACK, yêu cầu DMA DMA REQ, nhận trả lời DMA DMA ACK.

- Đệm dữ liệu: (Data Buffer) Là giao diện của bộ điều khiển đĩa với D.BUS.

- Thanh ghi viết (Write Register): Thanh ghi này để nhận dữ liệu từ đệm dữ liệu trước khi ghi ra đĩa.

- Thanh ghi lệnh (Command Register): Để nhận lệnh từ hệ thống.

- Thanh ghi đọc (Read Register): Thanh ghi này để lưu trữ dữ liệu đọc được từ đĩa trước khi đưa ra đệm.

- Thanh ghi trạng thái (Status Register): Để lưu trữ thông tin trạng thái của bộ điều khiển.

- Khối định dạng dữ liệu (Data Formatter): Để chuyển đổi dữ liệu từ song song thành nối tiếp và thêm vào đó những tín hiệu đồng hồ, khi ghi thông tin lên đĩa. Để biến đổi dữ liệu nối tiếp đọc được từ đĩa và tách bỏ xung đồng hồ, khi đọc thông tin từ đĩa.

- Khối logic điều khiển dịch chuyển (Motion Control Logic): Để nhận lệnh từ CPU và thông tin trạng thái của ổ đĩa, từ đó đưa ra các tín hiệu điều khiển tương ứng như: Điều khiển vận tốc motor, vị trí đầu từ, reset lỗi, R/W...

- Khối ổ đĩa (Floppy Disk Drive Unit): Khối này chủ yếu gồm các bộ phận cơ khí của ổ đĩa và phần mạch điện tử điều khiển trực tiếp sự đọc/ghi đĩa.

Hoạt động của ổ đĩa chủ yếu gồm hai thao tác sau

- Đọc thông tin: Thông tin được đọc từ đĩa dạng nối tiếp qua bộ định dạng dữ liệu được chuyển thành dạng song song, sau đó được cất vào thanh ghi đọc. Từ đây thông tin qua đệm dữ liệu tới D.BUS.

- Ghi thông tin: Thông tin từ D.BUS qua đệm dữ liệu tới thanh ghi viết, qua định dạng dữ liệu chuyển thành thông tin nối tiếp, và sau đó được ghi lên bề mặt đĩa.

Khi hoạt động ngoài các tín hiệu dữ liệu được lưu chuyển như ở trên, còn có rất nhiều tín hiệu điều khiển nối ghép khác được thực hiện.

Để điều khiển hoạt động của bộ nối ghép, CPU có thể ghi lệnh vào thanh ghi lệnh và có thể đọc trạng thái của nó qua thanh ghi trạng thái. Ngoài ra bộ điều khiển ổ đĩa có thể được nối ghép với DMAC, giúp cho việc chuyển dữ liệu được nhanh hơn.

Câu hỏi ôn tập

1. Vi mạch 8253 có chức năng gì? Trình bày cấu trúc và hoạt động của vi mạch.
2. Vi mạch 8257 có chức năng gì? Trình bày cấu trúc và hoạt động của vi mạch.
3. Vi mạch 8259 có chức năng gì? Trình bày cấu trúc và hoạt động của vi mạch.
4. Vi mạch 8251 có chức năng gì? Trình bày cấu trúc và hoạt động của vi mạch.
5. Trình bày ghép nối và điều khiển ghép nối giữa bộ vi xử lý với các thiết bị bàn phím, màn hình, máy in hoặc ổ đĩa.

Chương 8

VI ĐIỀU KHIỂN

Mục tiêu:

Giới thiệu về cấu trúc, nguyên lý hoạt động của họ vi điều khiển 8051 là một hệ vi xử lý được ứng dụng rất nhiều trong các ứng dụng công nghiệp để học sinh có được kiến thức để khai thác các bộ vi xử lý, vi điều khiển khác.

Giới thiệu tập lệnh và các giao tiếp ứng dụng của bộ vi điều khiển 8051 với thiết bị ngoại vi để cho học sinh hình thành được các khả năng ứng dụng của các bộ vi xử lý và vi điều khiển trong thực tế.

Nội dung chính:

- I. Khái quát chung về vi điều khiển
- II. Cấu trúc phần cứng của họ vi điều khiển 8051
 1. Sơ đồ khối của vi điều khiển 8051
 2. Hoạt động của vi điều khiển 8051
 3. Ghép nối 8051 với ngoại vi
- III. Giới thiệu bộ lệnh của vi điều khiển 8051
 1. Các kiểu định địa chỉ
 2. Các loại lệnh

I. KHÁI QUÁT CHUNG VỀ VI ĐIỀU KHIỂN

Vào năm 1971 tập đoàn Intel đã giới thiệu 8080, bộ vi xử lý (micro-processor) thành công đầu tiên. Sau đó không lâu, Motorola, RCA, kế đến là MOS Technology và Zilog đã giới thiệu các bộ vi xử lý tương tự: 6800, 1801, 6502 và Z80. Bản thân các vi mạch (IC: integrated circuit) này tuy không có nhiều hiệu quả sử dụng nhưng khi là một phần của máy tính đơn board (single-board computer), chúng trở thành thành phần trung tâm trong các sản phẩm có ích dùng để nghiên cứu và thiết kế. Các máy tính đơn board này, trong đó có D2 của Motorola, KIM-1 của MOS Technology và SDK-85 của Intel là đáng

ghi nhớ nhất, đã nhanh chóng xâm nhập vào các phòng thí nghiệm thiết kế của trường trung học, trường đại học và các công ty điện tử.

Vào năm 1976 Intel giới thiệu bộ vi điều khiển (microcontroller) 8748, một chip tương tự như các bộ vi xử lý và là chip đầu tiên trong họ vi điều khiển MCS-48. 8748 là một vi mạch chứa trên 17000 transistor bao gồm một CPU, 1 Kbyte EPROM, 64 byte RAM, 27 chân xuất nhập và một bộ định thời 8 bit. IC này và các IC khác tiếp theo của họ MCS-48 đã nhanh chóng trở thành chuẩn công nghiệp trong các ứng dụng hướng điều khiển (control-oriented application). Việc thay thế các thành phần cơ điện trong các sản phẩm như các máy giặt và các bộ điều khiển đèn giao thông là một ứng dụng phổ biến ban đầu. Các sản phẩm khác mà trong đó bộ vi điều khiển được tìm thấy bao gồm xe ô tô, thiết bị công nghiệp, các sản phẩm tiêu dùng và các ngoại vi của máy tính (bàn phím của IBM-PC là một ví dụ sử dụng họ vi điều khiển trong các thiết kế tối thiểu thành phần).

Độ phức tạp, kích thước và khả năng của các bộ vi điều khiển được tăng thêm một bậc quan trọng vào năm 1980 khi Intel công bố chip 8051, bộ vi điều khiển đầu tiên của họ vi điều khiển MCS-51. So với 8048, chip 8051 chứa trên 60000 transistor bao gồm 4 Kbyte ROM, 128 byte RAM, 32 đường xuất nhập, 1 port nối tiếp và 2 bộ định thời 16 bit, một số lượng mạch đáng chú ý trong một IC đơn. Các thành viên mới được thêm vào cho họ MCS-51 và các biến thể ngày nay gần như có gấp đôi các đặc trưng này. Tập đoàn Siemens, nguồn sản xuất thứ hai các bộ vi điều khiển thuộc họ MCS-51 cung cấp chip SAB80518, một cải tiến của 8051 chứa trong một vỏ 68 chân, có 6 port xuất nhập 8 bit, 13 nguồn tạo ra ngắt và một bộ biến đổi A/D 8-bit với 8 kênh ngõ vào. Họ 8051 là một trong những bộ vi điều khiển 8bit mạnh và linh hoạt nhất, đã trở thành bộ vi điều khiển hàng đầu trong những năm gần đây.

II. CẤU TRÚC PHẦN CỨNG CỦA HỌ ĐIỀU KHIỂN 8051

1. Sơ đồ khối

MCS-51 là họ vi điều khiển của Intel. Các nhà sản xuất IC khác như Siemens, Advanced Micro Devices, Fujitsu và Philips được cấp phép làm các nhà cung cấp thứ hai cho các chip của họ MCS-51.

Vi mạch tổng quát của họ MCS-51 là chip 8051, linh kiện đầu tiên của họ này được đưa ra thị trường. Chip 8051 có các đặc trưng được tóm tắt như sau:

- 4KB ROM.
- 128 byte RAM.
- 4 port xuất nhập (I/O port) 8-bit.
- 2 bộ định thời 16-bit.
- Mạch giao tiếp nối tiếp.
- Không gian nhớ chương trình (mã) ngoài 64K.
- Không gian nhớ dữ liệu ngoài 64K.
- Bộ xử lý bit (thao tác trên các bit riêng rẽ).
- 210 vị trí nhớ được định địa chỉ, mỗi vị trí 1 bit.
- Nhân/chia trong 4 μ s.

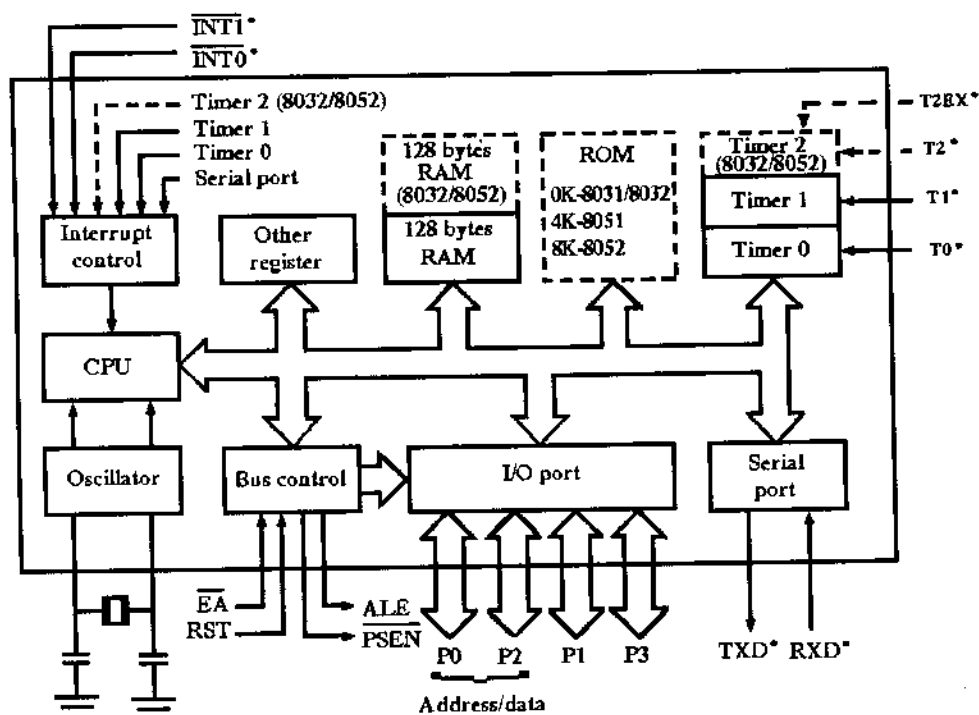
Các thành viên khác của họ MCS -51 có các tổ hợp ROM (EPROM), RAM trên chip khác nhau hoặc có thêm bộ định thời thứ ba. Mỗi một IC của họ MCS -51 cũng có phiên bản CMOS công suất thấp.

Bảng 8.1: So sánh các chip của họ MCS-51

Chip	Bộ nhớ chương trình trên chip	Bộ nhớ dữ liệu trên chip	Các bộ định thời
8051	4 K ROM	128 byte	2
8031	0 K	128 byte	2
8751	4 K EPROM	128 byte	2
8052	8 KROM	256 byte	3
8032	0 K	256 byte	3
8752	8 K EPROM	256 byte	3

Thuật ngữ “8051” được dùng để chỉ rộng rãi các chip của họ MCS -51.

Khi việc thảo luận tập trung vào một cải tiến từ chip 8051 cơ bản, chip cải tiến được chỉ rõ ràng. Các đặc trưng vừa nêu trên được trình bày trong sơ đồ khối sau:



Hình 8.1: Sơ đồ khối của chip 8051

Trong đó:

Interrupt control: điều khiển ngắt.

Other registers: các thanh ghi khác.

128 byte RAM: RAM 128 byte.

Timer 2, 1, 0: bộ định thời 2, 1, 0.

CPU: đơn vị điều khiển trung tâm.

Oscillator: mạch dao động.

Bus control: điều khiển bus.

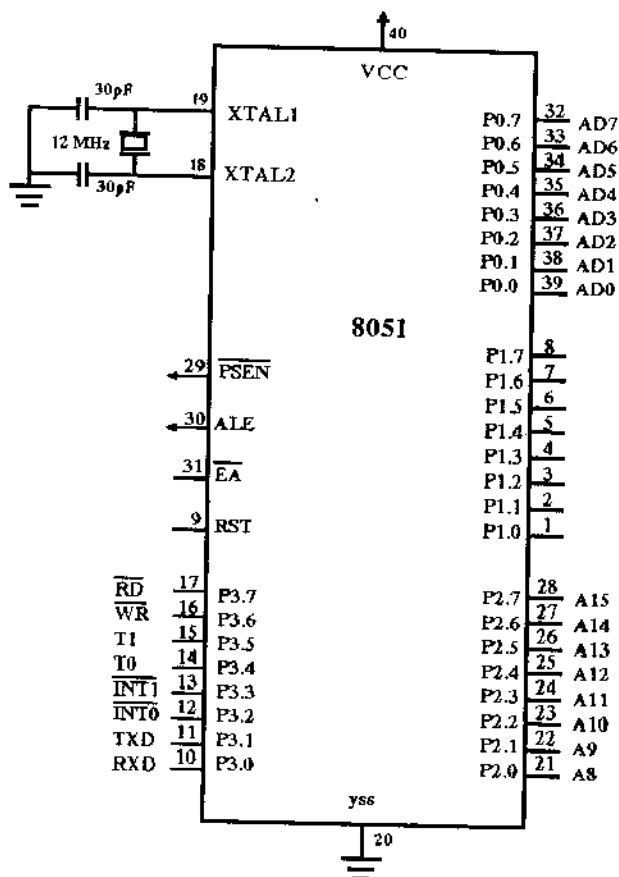
I/O port: các port xuất/nhập.

Serial port: port nối tiếp.

Address/data: địa chỉ/dữ liệu.

2. Hoạt động của vi điều khiển 8051

Sơ đồ chân của chip 8051. Mô tả tóm tắt chức năng của từng chân như sau:



Hình 8.2: Sơ đồ chân của 8051

Như ta thấy 32 trong số 40 chân của 8051 có công dụng xuất/nhập, tuy nhiên 24 trong 32 đường này có 2 mục đích (công dụng) (26/32 đối với 8032/8052). Mỗi một đường có thể hoạt động như một đường địa chỉ/dữ liệu của bus địa chỉ/dữ liệu đa hợp.

32 chân nêu trên hình thành 4 port 8-bit. Với các thiết kế yêu cầu một mức tối thiểu bộ nhớ ngoài hoặc các thành phần bên ngoài khác, ta có thể sử dụng các port này làm nhiệm vụ xuất/nhập. 8 đường cho mỗi port có thể được xử lý như một đơn vị giao tiếp với các thiết bị song song như máy in, bộ biến đổi D-A, vv... hoặc mỗi đường có thể hoạt động độc lập giao tiếp với một thiết bị đơn bit như chuyển mạch, LED, BJT, FET, cuộn dây, động cơ, loa,...

2.1. Port 0

Port 0 (các chân từ 32 đến 39 trên 8051) có 2 công dụng. Trong các thiết kế có tối thiểu thành phần, port 0 được sử dụng làm nhiệm vụ xuất/nhập. Trong

các thiết kế lớn hơn có bộ nhớ ngoài, port 0 trở thành địa chỉ bus dữ liệu đa hợp (byte thấp của bus địa chỉ nếu là địa chỉ).

2.2. Port 1

Port 1 chỉ có một công dụng là xuất/nhập (các chân từ 1 đến 8 trên 8051). Các chân của port 1 được ký hiệu là P1.0, P1.1, ..., P1.7 và được dùng để giao tiếp với các thiết bị bên ngoài khi có yêu cầu. Không có chức năng nào khác nữa gán cho các chân của port 1, nghĩa là chúng chỉ sử dụng được giao tiếp với các thiết bị ngoại vi. (Ngoại lệ với 8032/8052, ta có thể sử dụng P1.0 và P1.1 hoặc làm các đường xuất/nhập hoặc làm các ngõ vào cho mạch định thời thứ ba).

2.3. Port 2

Port 2 (các chân từ 21 đến 28 trên 8051) có hai công dụng, hoặc làm nhiệm vụ xuất/nhập hoặc là byte địa chỉ 16-bit cho các thiết kế có bộ nhớ chương trình ngoài hoặc các thiết kế có nhiều hơn 256 byte bộ nhớ dữ liệu ngoài.

2.4. Port 3

Port 3 (các chân từ 10 đến 17 trên 8051) có 2 công dụng. Khi không hoạt động xuất/nhập, các chân của port 3 có nhiều chức năng riêng (mỗi chân có chức năng riêng liên quan đến các đặc trưng cụ thể của 8051). Bảng dưới đây cho ta chức năng của các chân của port 3 và 2 chân P1.0, P1.1 của port 1.

Bảng 8.2: Chức năng của các chân port 3 và port 1

Bit	Tên	Địa chỉ bit	Chức năng
P3.0	RxD	BOH	Chân nhận dữ liệu của port nối tiếp
P3.1	TxD	B1H	Chân phát dữ liệu của port nối tiếp
P3.2	$\overline{\text{INT0}}$	B2H	Ngõ vào ngắt ngoài 0
P3.1	$\overline{\text{INT1}}$	B3H	Ngõ vào ngắt ngoài 1
P3.4	T0	B4H	Ngõ vào của bộ định thời/dếm 0
P3.5	T1	B5H	Ngõ vào của bộ định thời/dếm 1
P3.6	$\overline{\text{WR}}$	B6H	Điều khiển ghi bộ nhớ dữ liệu ngoài
P3.7	$\overline{\text{RD}}$	B7H	Điều khiển đọc bộ nhớ dữ liệu ngoài
P1.0	T2	90H	Ngõ vào của bộ định thời/dếm 2
P1.1	T2EX	91H	Nạp lại/thu nhận của bộ định thời 2

2.5. Chân cho phép bộ nhớ chương trình $\overline{\text{PSEN}}$

8051 cung cấp 4 tín hiệu điều khiển bus. Tín hiệu cho phép bộ nhớ chương trình $\overline{\text{PSEN}}$ (program store enable) là tín hiệu xuất trên chân 29. Đây là tín hiệu điều khiển cho phép ta truy xuất bộ nhớ chương trình ngoài. Chân này thường nối với chân cho phép xuất OE (output enable) của EPROM (hoặc ROM) để cho phép đọc các byte lệnh.

Tín hiệu $\overline{\text{PSEN}}$ ở logic 0 trong suốt thời gian tìm nạp lệnh. Các mã nhị phân của chương trình hay opcode (mã thao tác) được đọc từ EPROM, qua bus dữ liệu và được chốt vào thanh ghi lệnh IR của 8051 để được giải mã.

Khi thực thi một chương trình chứa ở ROM nội, $\overline{\text{PSEN}}$ được duy trì ở logic không tích cực (logic 1).

2.6. Chân cho phép chốt địa chỉ ALE

8051 sử dụng chân 30, chân xuất tín hiệu cho phép chốt địa chỉ (address latch enable) để giải đa hợp (phân kênh, demultiplexing) bus dữ liệu và bus địa chỉ. Khi port 0 được sử dụng làm bus địa chỉ/dữ liệu đa hợp, chân ALE xuất tín hiệu để chốt địa chỉ (byte thấp của địa chỉ 16-bit) vào một thanh ghi ngoài trong suốt 1/2 đầu của chu kỳ bộ nhớ (memory cycle). Sau khi điều này đã được thực hiện, các chân của port 0 sẽ xuất/nhập dữ liệu hợp lệ trong suốt 1/2 thứ hai của chu kỳ bộ nhớ.

Tín hiệu ALE có tần số bằng 1/6 tần số của mạch dao động bên trong chip vì điều khiển và có thể được dùng làm xung clock cho phần còn lại của hệ thống. Nếu mạch dao động có tần số 12MHz, tín hiệu ALE có tần số 2MHz. Ngoại lệ duy nhất là trong thời gian thực thi lệnh MOVX, một xung ALE sẽ bị bỏ qua. Chân ALE còn được dùng để nhận xung ngõ vào lập trình cho EPROM trên chip đối với các phiên bản của 8051 có EPROM này.

2.7. Chân truy xuất ngoài $\overline{\text{EA}}$

Ngõ vào này (chân 31) có thể được nối với 5V (logic 1) hoặc với GND (logic 0). Nếu chân này nối với 5V, 8051/8052 thực thi chương trình trong ROM nội (chương trình nhỏ hơn 4K/8K). Nếu chân này nối với GND (và chân $\overline{\text{PSEN}}$ cũng ở logic 0), chương trình cần thực thi chứa ở bộ nhớ ngoài. Đối với 8031/8032 chân $\overline{\text{EA}}$ phải ở logic 0 vì chúng không có bộ nhớ chương trình trên chip. Nếu chân $\overline{\text{EA}}$ ở logic 0 đối với 8051/8052, ROM nội bên trong chip được vô hiệu hoá và chương trình cần thực thi chứa ở EPROM bên ngoài.

Các phiên bản EPROM của 8051 còn sử dụng chân $\overline{EA}$ làm chân nhận điện áp cấp điện 21V (V_{pp}) cho việc lập trình EPROM nội (nạp EPROM).

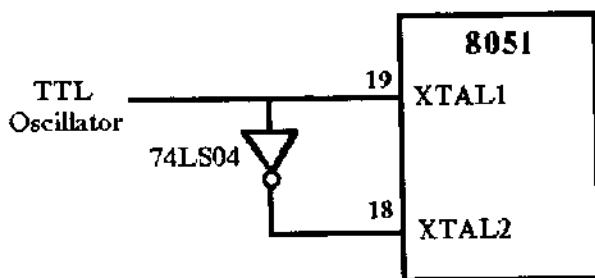
2.8. Chân RESET (RST)

Ngõ vào RST (chân 9) là ngõ vào xoá chính (master reset) của 8051 dùng để thiết lập lại trạng thái ban đầu cho hệ thống hay gọi tắt là reset hệ thống.

Khi ngõ vào này được treo ở logic 1 tối thiểu hai chu kỳ máy, các thanh ghi bên trong của 8051 được nạp các giá trị thích hợp cho việc khởi động lại hệ thống.

2.9. Các chân XTAL1 và XTAL2

Mạch dao động bên trong chip 8051 được ghép với thạch anh bên ngoài ở 2 chân XTAL1 và XTAL2 (chân 18 và 19) cùng với các tụ ổn định. Tần số danh định của thạch anh là 12MHz cho hầu hết các chip của họ MCS -51 (80C31BH-1 sử dụng thạch anh 16MHz bên trong, mạch dao động trong chip không cần thạch anh bên ngoài). Một nguồn xung clock TTL có thể được nối với các chân XTAL1 và XTAL2 để đồng bộ từ bên ngoài.



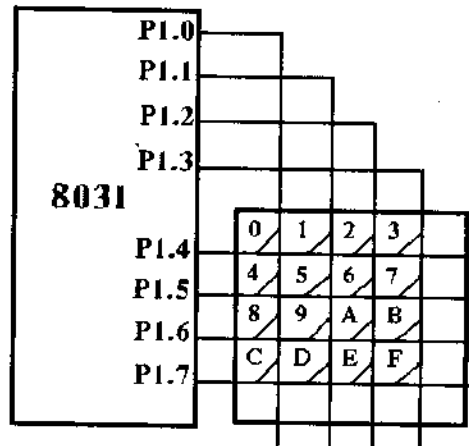
Hình 8.3: 8051 ghép với mạch dao động TTL bên ngoài

3. Ghép nối vi điều khiển 8051 với ngoại vi

3.1. Giao tiếp với bàn phím số hex

Giao tiếp với bàn phím thường được cần đến đối với các thiết kế dựa trên bộ vi điều khiển. Nhập từ bàn phím và xuất ra LED là sự lựa chọn kinh tế để giao tiếp với người sử dụng và thường thích hợp với các ứng dụng phức tạp. Ví dụ bao gồm việc giao tiếp người sử dụng với lò vi ba hoặc máy đổi tiền tự động.

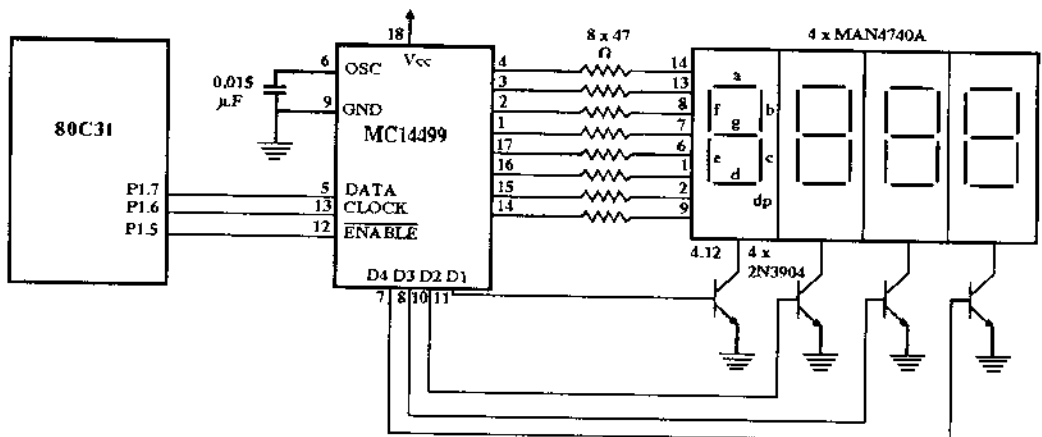
Hình 8.4 trình bày cách giao tiếp giữa port 1 và bàn phím số hex. Bàn phím có 16 bit được sắp xếp thành 4 hàng và 4 cột. Các đường hàng được nối với các bit từ 4 đến 7 còn các đường cột được nối với các bit từ 0 đến 3 của port 1.



Hình 8.4: Giao tiếp với bàn phím số hex

3.2. Giao tiếp với các đèn 7 đoạn

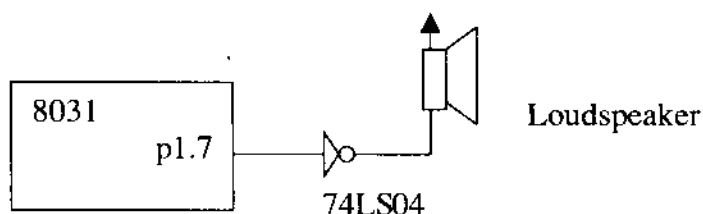
Một mạch ví dụ giao tiếp với bốn đèn LED 7 đoạn được sử dụng ba trong số các đường xuất nhập của 8051. Trung tâm của thiết kế này là vi mạch giải mã và kích đèn 7 đoạn MC14499 của Motorola, vì mạch này chứa bên trong nhiều mạch cần thiết cho việc kích 4 đèn 7 đoạn. Các thành phần được thêm vào chỉ là tụ định thời $0,015\mu\text{F}$, 7 điện trở giới hạn dòng 47Ω và 4 transistor 2N3904.



Hình 8.5: Giao tiếp với MC14499 và 4 đèn 7 thanh

3.3. Giao tiếp với loa

Giao tiếp vi điều khiển với loa được thực hiện bằng sơ đồ sau:



Hình 8.6 : Giao tiếp với loa

Các loa nhỏ, chẳng hạn như các loa trong máy tính cá nhân hoặc đồ chơi trẻ em, có thể được kích từ một cổng logic. Một đầu cuộn dây của loa ghép với +5V, đầu còn lại ghép với ngõ ra của cổng đảo 74LS04. Cổng đảo được cần đến do cổng này có khả năng cấp và hút dòng cao hơn các chân port của 8031.

3.4. Giao tiếp với RAM không mất nội dung

RAM không mất nội dung NVRAM (Non Volatile RAM) là bộ nhớ bán dẫn duy trì được nội dung khi ta không cung cấp điện cho RAM. NVRAM kết hợp cả hai: các phần tử của RAM tĩnh và các phần tử của ROM lập trình được và xoá được (EEPROM). Mỗi một bit của RAM tĩnh được phủ một bit của EEPROM. Dữ liệu có thể truyền qua lại giữa hai bit nhớ và như vậy giữa hai bộ nhớ.

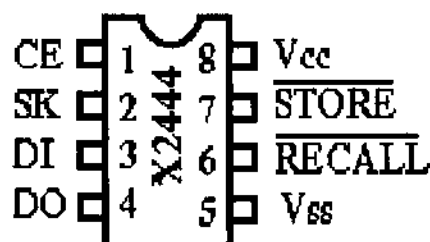
NVRAM chiếm một vị trí quan trọng trong các ứng dụng của bộ vi xử lý và bộ vi điều khiển. NVRAM được sử dụng để lưu các dữ liệu và các tham số được cài đặt, các dữ liệu và các thông số này thỉnh thoảng được thay đổi bởi người sử dụng nhưng phải được duy trì khi không được cung cấp điện.

Nhiều thiết kế của thiết bị hiển thị video chuẩn (VDT) không sử dụng chuyển mạch DIP mà sử dụng NVRAM để lưu giữ các thông tin được cài đặt như là tốc độ baud, cho phép hoặc không cho phép bit chẵn lẻ, kiểm tra chẵn hay lẻ... Mỗi khi VDT được cấp điện, các tham số sẽ được gọi từ NVRAM và hệ thống được khởi động tương ứng một cách thích hợp. Khi một tham số được thay đổi từ người sử dụng (thông qua bàn phím), giá trị mới được lưu vào trong NVRAM.

Ví dụ, bộ nhớ NVRAM được sử dụng ở đây là X2444 của Xicor. X2444 chứa 256 bit RAM tĩnh được phủ bởi 256 bit EEPROM. Các dữ liệu được truyền qua lại giữa hai bit nhớ hoặc bằng các lệnh được gửi đi từ bộ vi xử lý trên mạch giao tiếp nối tiếp hoặc bằng cách sử dụng hai ngõ vào STORE và RECALL. Các dữ liệu không bị mất được lưu trong EEPROM trong khi dữ liệu đọc lập được truy xuất và cập nhật trong RAM. Các đặc trưng của X2444 như sau:

- * Lý tưởng khi sử dụng máy tính đơn chip:
 - Định thời tĩnh.
 - Giao tiếp I/O tối thiểu.
 - Tương thích với port nối tiếp.
 - Dễ dàng giao tiếp với các port của các bộ vi điều khiển.
 - Các mạch hỗ trợ tối thiểu.
- * Đặc trưng không mất thông tin được điều khiển bởi phần cứng và phần mềm.
- Bảo vệ bộ nhớ tối đa.
- * Tương thích TTL.
- * Tổ chức 16x16.
- * Công suất tiêu tán nhiệt thấp.
 - Dòng tích cực: điển hình là 15 mA.
 - Dòng lưu trữ: điển hình là 8 mA.
 - Dòng chờ: điển hình là 6 mA.
 - Dòng nghỉ: điển hình là 5 mA.

Sơ đồ chân và thông số kỹ thuật của X2444 được mô tả ở hình sau đây:

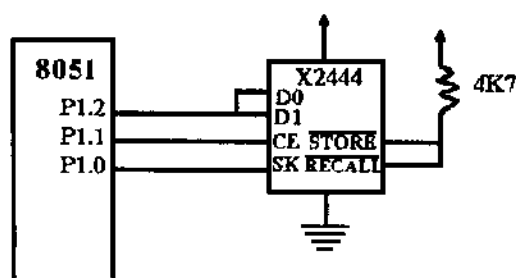


CE	Chip Enable
SK	Serial Clock
DI	Serial Data In
DO	Serial Data Out
<u>RECALL</u>	Recall
<u>STORE</u>	Store
Vcc	+5V
Vss	Ground

Hình 8.7: Các thông số kỹ thuật của X2444

Sơ đồ giao tiếp giữa bộ vi điều khiển 8051 với bộ nhớ NVRAM X2444 được thể hiện ở hình 8.8.

Trong trường hợp này các đường lệnh STORE và RECALL không được sử dụng. Các chế độ khác của thao tác được đưa vào bằng cách gửi đến X2444 các lệnh ở dạng nối tiếp thông qua các chân port của 8051. Có ba đường được sử dụng là:



Hình 8.8: Giao tiếp với bộ nhớ X2444

- P1.0 - SK (Serial clock: xung nhịp nối tiếp)
- P1.1 - CE (Chip Enable: cho phép chip)
- P1.2 - DI/DO (Data input/output: dữ liệu nhập/xuất)

Các lệnh được chuyển đến X2444 bằng cách cho CE ở mức cao và kể đến dịch 8 bit của opcode bằng xung clock và X2444 qua các đường SK và DI/DO. Các opcode sau đây cần cho ví dụ này:

Lệnh	Opcode	Mô tả
RCL	85H	Gọi lại dữ liệu trong EEPROM vào RAM
WREN	84H	Thiết lập chốt cho phép ghi
STORE	81H	Lưu dữ liệu trong RAM vào EEPROM
WRITE	1AAAA011B	Ghi dữ liệu vào RAM ở địa chỉ AAAA
READ	1AAAA111B	Đọc dữ liệu từ RAM ở địa chỉ AAAA

III. GIỚI THIỆU BỘ LỆNH CỦA VI ĐIỀU KHIỂN 8051

1. Các kiểu định địa chỉ

Khi một lệnh được thực thi và lệnh này cần dữ liệu, một câu hỏi được đặt ra là:

“Dữ liệu chứa ở đâu”. Câu trả lời cho câu hỏi này tạo ra các kiểu định địa chỉ của 8051. Có nhiều kiểu định địa chỉ do vậy có nhiều câu trả lời cho câu hỏi nêu trên, chẳng hạn như: trong byte thứ 2 của 1 lệnh, trong thanh ghi R4,

trong địa chỉ trực tiếp hoặc có thể trong bộ nhớ ngoài ở địa chỉ chứa trong con trỏ dữ liệu.

Các kiểu định địa chỉ là phần cần thiết cho toàn bộ tập lệnh cả mỗi một bộ vi xử lý, bộ vi điều khiển. Các kiểu định địa chỉ cho phép ta xác định rõ nguồn và đích của dữ liệu theo nhiều cách khác nhau phụ thuộc vào tình huống lập trình. Trong phần này chúng ta sẽ khảo sát tất cả các kiểu định địa chỉ của 8051 và nêu ví dụ cho từng kiểu. Có 8 kiểu định địa chỉ:

1.1. Định địa chỉ thanh ghi (Register Addressing)

Người lập trình trên 8051 có thể truy xuất 8 thanh ghi từ R_0 đến R_7 . Các lệnh sử dụng kiểu định địa chỉ thanh ghi được mã hoá bằng cách dùng 3 bit thấp nhất của opcode (của lệnh) để chỉ ra một thanh ghi bên trong không gian địa chỉ logic này. Vậy thì một mã chức năng và địa chỉ toán hạng có thể kết hợp để hình thành một lệnh ngắn.

Hợp ngữ của 8051 chỉ ra kiểu định địa chỉ thanh ghi bằng ký hiệu R_n , trong đó n có giá trị từ 0 đến 7.

Ví dụ: Để cộng nội dung của thanh ghi R_7 với thanh chứa A, ta dùng lệnh sau:

`ADD A, R7`

Và lệnh này có opcode là 00101111B. Năm bit cao 00101 cho biết đây là lệnh cộng và 3 bit thấp 111 chỉ ra thanh ghi R_7 .

Có 4 dãy thanh ghi làm việc nhưng ở một thời điểm chỉ có một dãy tích cực. Các dãy thanh ghi chiếm 32 byte đầu tiên của RAM dữ liệu trên chip (địa chỉ từ 00H đến 1FH) và ta dùng các bit 4 và 3 của từ trạng thái chương trình PSW để chỉ ra dãy thanh ghi tích cực. Một reset bằng phần cứng cho phép dãy 0 tích cực còn các dãy khác được chọn bằng cách sửa đổi các bit 4 và 3 của PSW sao cho phù hợp.

Ví dụ: lệnh,

`MOV PSW,#00011000B`

Sẽ tích cực dãy thanh ghi 3 bằng cách đặt các bit chọn dãy thanh ghi ($RS1$ và $RS0$) trong PSW lên 1 (các bit này ở vị trí 3 và 4). Một số lệnh đặc biệt liên quan đến 1 thanh ghi xác định nào đó như là thanh chứa A, con trỏ dữ liệu... không cần các bit địa chỉ, bản thân opcode của lệnh đã chỉ ra thanh ghi cần thiết. Các lệnh đặc biệt liên quan đến thanh ghi này tham chiếu đến thanh chứa bằng ký hiệu "A", con trỏ dữ liệu bằng ký hiệu "DPTR", bộ đếm chương trình bằng ký hiệu "PC", cờ nhớ bằng ký hiệu "C" và cặp thanh ghi AB bằng ký hiệu "AB". Ví dụ:

INC DPTR

Là lệnh 1 byte, lệnh này cộng 1 vào nội dung của con trỏ dữ liệu 16 bit.

1.2. Định địa chỉ trực tiếp (Direct Addressing)

Kiểu định địa chỉ trực tiếp được sử dụng để truy xuất các biến nhớ hoặc thanh ghi trên chip. Một byte thêm vào tiếp theo opcode dùng để xác định địa chỉ.

Phụ thuộc vào bit có giá trị vị trí (hay trọng số) cao của địa chỉ trực tiếp, một trong hai không gian nhớ trên chip được chọn. Khi bit 7 bằng 0, địa chỉ trực tiếp ở trong tầm từ 0 đến 127 (00H-7FH) và 128 byte thấp trên của RAM trên chip được tham chiếu. Tất cả các cổng xuất nhập và các thanh ghi chức năng đặc biệt, điều khiển, trạng thái được gán địa chỉ trong tầm từ 128 đến 255 (80H-FFH). Khi byte địa chỉ theo sau opcode có nội dung nằm trong giới hạn này (với bit 7 bằng 1), thanh ghi chức năng đặc biệt được truy xuất. Ví dụ port 0 và port 1 được gán địa chỉ trực tiếp là 80H và 90H.

Ta không nhất thiết phải biết địa chỉ của các thanh ghi này, trình dịch hợp ngữ cho phép ta sử dụng mã gọi nhớ viết tắt dễ hiểu như là “P0” cho cổng 0, “TMOD” cho thanh ghi chế độ định thời (time mode register)...Lệnh sau đây là một ví dụ cho kiểu định địa chỉ trực tiếp:

```
MOV P1, A
```

Lệnh này chuyển nội dung của thanh chứa A vào port 1. Địa chỉ trực tiếp của port 1 là 90H được xác định bởi trình dịch hợp ngữ và trình dịch này đặt 90H vào byte 2 của lệnh. Nguồn của dữ liệu, thanh chứa, được xác định rõ ràng trong opcode.

1.3. Định địa chỉ gián tiếp (Indirect Addressing)

Làm cách nào để nhận biết 1 biến khi địa chỉ của biến đã được xác định, được tính toán hoặc được sửa đổi trong khi một chương trình đang chạy? Tình huống này được phát sinh khi ta quản lý các vị trí nhớ liên tiếp, các điểm nhập được định chỉ số trong các bảng chứa trong RAM, các số chính xác hoặc các chuỗi ký tự. Các kiểu định địa chỉ thanh ghi hoặc trực tiếp không sử dụng được cho các tình huống này, do vậy ta cần có các địa chỉ toán hạng được biết trong thời gian thích hợp dịch.

Giải pháp của 8051 là dùng kiểu định địa chỉ gián tiếp. Các thanh ghi R0 và R1 có thể hoạt động như là các con trỏ và nội dung của chúng chỉ ra địa chỉ trong RAM, nơi mà dữ liệu được đọc hay được ghi. Bit có ý nghĩa thấp nhất của opcode (của lệnh) xác định thanh ghi nào (R0 hay R1) được sử dụng làm

con trỏ. Trong hợp ngữ của 8051, kiểu định địa chỉ gián tiếp được nhận biết nhờ vào ký tự @ đặt trước R0 hoặc R1. Lấy ví dụ nếu R1 chứa 40H và địa chỉ 40H của bộ nhớ nội chứa 55H, lệnh:

```
MOV A, @R1
```

Nạp 55H cho thanh chứa A.

Ta cần đến kiểu định địa chỉ gián tiếp khi ta duyệt các vị trí liên tiếp trong bộ nhớ. Ví dụ sau thực hiện việc tuần tự xoá RAM nội từ địa chỉ 60H đến 7FH:

```
MOV R0, #60H
```

```
LOOP: MOV @R0, #0
```

```
INC R0
```

```
CJNE R0, #80H, LOOP
```

(tiếp tục)

Lệnh đầu tiên khởi động R0 với nội dung là 60H, địa chỉ bắt đầu của khối nhớ trong RAM; lệnh thứ hai sử dụng kiểu định địa chỉ gián tiếp để nạp 00H cho vị trí được trỏ bởi R0; lệnh thứ ba tăng con trỏ (R0) để trỏ đến địa chỉ tiếp theo và lệnh cuối cùng kiểm tra con trỏ xem đã kết thúc khối nhớ chưa. Việc kiểm tra sử dụng hằng số 80H thay vì là 7FH vì lệnh tăng xuất hiện sau lệnh di chuyển. Điều này đảm bảo vị trí nhớ sau cùng (7FH) được ghi 00H trước khi kết thúc.

1.4. Định địa chỉ tức thời (Immediate Addressing)

Khi toán hạng nguồn là một hằng số thay vì là một biến (nghĩa là lệnh sử dụng 1 giá trị đã biết trước ở thời gian hợp dịch), hằng số này có thể đưa vào lệnh và đây là byte dữ liệu tức thời (byte thêm vào này có giá trị biết trước).

Trong hợp ngữ, các toán hạng tức thời được nhận biết nhờ vào ký tự # đặt trước chúng. Toán hạng này có thể là một hằng số học, một biến hoặc một biểu thức số học sử dụng hằng số, các ký hiệu và các toán tử. Trình hợp dịch hợp ngữ tính giá trị và thay thế dữ liệu tức thời vào trong lệnh. Ví dụ lệnh:

```
MOV A, #12
```

Nạp giá trị 12 (0CH) vào thanh chứa A.

Tất cả các lệnh sử dụng kiểu định địa chỉ tức thời đều sử dụng hằng dữ liệu 8 bit làm dữ liệu tức thời. Có một ngoại lệ khi ta khởi động con trỏ dữ liệu 16 bit DPTR, hằng địa chỉ 16 bit được cần đến. Ví dụ:

```
MOV DPTR, #8000H
```

Là một lệnh 3 byte, lệnh này nạp hằng địa chỉ 8000H vào con trỏ dữ liệu DPTR.

1.5. Định địa chỉ tương đối (Relative Addressing)

Kiểu định địa chỉ tương đối chỉ được sử dụng cho các lệnh nhảy. Một địa chỉ tương đối (hay còn gọi là offset) là một giá trị 8 bit có dấu. Giá trị này được cộng với bộ đếm chương trình để tạo ra địa chỉ của lệnh tiếp theo cần được thực thi. Do ta sử dụng một offset 8 bit có dấu, tầm nhảy được giới hạn là -128 byte đến 127 byte. Byte địa chỉ tương đối là byte thêm vào tiếp theo byte opcode của lệnh.

Nhờ vào phép cộng, bộ đếm chương trình được tăng đến địa chỉ theo sau lệnh nhảy; vậy thì địa chỉ mới liên quan đến lệnh kế tiếp, không liên quan đến địa chỉ của lệnh nhảy. Thông thường chi tiết này không liên quan đến người lập trình do bởi các đích nhảy thường được xác định bằng các nhãn và trình dịch hợp ngữ sẽ xác định offset tương đối tương ứng. Ví dụ nhãn THERE đặt trước lệnh ở địa chỉ 1040H, lệnh:

JMP THERE

Ở trong bộ nhớ tại địa chỉ 1000H và 1001H, trình dịch hợp ngữ sẽ gán offset tương đối là 3EH cho byte 2 của lệnh ($1002H + 3EH = 1040H$).

Định địa chỉ tương đối có điểm lợi là cung cấp cho ta mã không phụ thuộc vào vị trí (vì các địa chỉ tuyệt đối không được dùng), nhưng lại có điều bất lợi là các đích nhảy bị giới hạn trong tầm.

1.6. Định địa chỉ tuyệt đối (Absolute Addressing)

Kiểu định địa chỉ tuyệt đối chỉ được sử dụng với các lệnh ACALL và AJMP. Đây là 2 lệnh 2 byte cho phép rẽ nhánh chương trình trong trang 2K hiện hành của bộ nhớ chương trình bằng cách cung cấp 11 bit thấp của địa chỉ đích, trong đó 3 bit cao (A8-A10) đưa vào opcode và 8 bit thấp (A0-A7) thành lập byte thứ 2 của lệnh.

5 bit cao của địa chỉ đích là 5 bit cao hiện hành trong bộ đếm chương trình, do vậy lệnh theo sau lệnh rẽ nhánh và đích của lệnh rẽ nhánh phải ở trong cùng 1 trang 2K, bởi vì A11-A15 không thay đổi. Ví dụ nếu nhãn THERE đặt trước lệnh ở địa chỉ 0F64H, lệnh:

AJMP THERE

Ở trong bộ nhớ tại địa chỉ 0900H và 0901H, trình dịch hợp ngữ sẽ mã hoá lệnh như sau:

11100001 - byte 1 (A10 - A8 + opcode)

01000110 - byte 2 (A7 - A0)

Các bit được gạch chân là 11 bit thấp của địa chỉ đích, 0F64H = 0000111101110B. Năm bit cao ở trong bộ đếm chương trình sẽ không thay đổi

khi lệnh trên được thực thi. Lưu ý là lệnh AJMP và đích nhảy đến đều ở trong 1 trang 2K giới hạn bởi 0800H - 0FFFH và do vậy 5 bit địa chỉ cao như nhau.

Địa chỉ tuyệt đối có điều lợi là lệnh ngắn (2-byte) nhưng có điều bất lợi là đích bị giới hạn tầm địa chỉ và cung cấp cho ta mã phụ thuộc vào vị trí.

1.7. Định địa chỉ dài (Long Addressing)

Kiểu định địa chỉ dài chỉ được dùng cho các lệnh LCALL và LJMP. Các lệnh 3-byte này chứa địa chỉ đích 16 - bit (2 byte: byte 2 và byte 3) của lệnh.

Lợi ích của kiểu định địa chỉ này là sử dụng hết toàn bộ không gian nhớ chương trình 64K, nhưng lại có điểm bất lợi là lệnh dài đến 3- byte và phụ thuộc vào vị trí.

Việc phụ thuộc vào vị trí được xem là bất lợi bởi chương trình không thể được thực thi ở một địa chỉ khác. Ví dụ nếu chương trình bắt đầu ở 2000H và có một lệnh như là LJMP 2040H, chương trình nhảy đến địa chỉ 2040H và đây không phải là vị trí đúng khi chương trình đã được di chuyển.

1.8. Định địa chỉ chỉ số (Indexed Addressing)

Kiểu định địa chỉ chỉ số sử dụng một thanh ghi nền (hoặc bộ đếm chương trình hoặc con trỏ dữ liệu) và một offset (thanh chứa A) tạo thành dạng địa chỉ hiệu dụng cho lệnh JMP hoặc MOVC. Trong nhiều ứng dụng, các bảng nhảy hoặc các bảng tìm kiếm được tạo ra dễ dàng bằng cách sử dụng kiểu định địa chỉ chỉ số. Ví dụ lệnh MOVC A, @A + <base-reg> và JMP @A+DPTR.

2. Các loại lệnh

Các lệnh của 8051 được chia làm 5 nhóm :

- Nhóm lệnh số học.
- Nhóm lệnh logic.
- Nhóm lệnh di chuyển dữ liệu.
- Nhóm lệnh xử lý bit.
- Nhóm lệnh rẽ nhánh.

Trong một lệnh 8051 thường có các ký hiệu sau:

Rn: Định địa chỉ thanh ghi sử dụng R0 - R7

Direct: Địa chỉ 8 bit trong KAM nội (OOH-FFFH)

@ Ri: Định địa chỉ gián tiếp sử dụng thanh ghi R0 hoặc R1

Source: Toán hạng nguồn, có thể là Rn hoặc Direct hoặc @ Ri

Dest: Toán hạng đích, có thể là Rn hoặc @ Ri

#data: Hàm số 8 bit chứa trong lệnh

#data16: hằng số 16 bit

bit: địa chỉ trực tiếp (8 bit) của 1 bit

rel: offset 8 bit có dấu

addr 16: địa chỉ 16 bit

Sau đây sẽ mô tả các nhóm lệnh 8051.

2.1. Nhóm lệnh số học

Nhóm lệnh số học gồm có các lệnh sau đây:

- * ADD A, Source : lệnh cộng toán hạng nguồn với A
- * ADD A, #data
- * ADDC A, Source : lệnh cộng toán hạng nguồn với A và cờ nhớ
- * ADDC A, #data
- * SUBB A, Source : lệnh trừ A bởi toán hạng nguồn và số mượn (cờ nhớ)
- * SUBB A, # data
- * INC A : lệnh tăng
- * INC Source
- * DEC A: lệnh giảm
- * DEC source
- * INC DPTR : lệnh tăng DPTR
- * MUL AB : lệnh nhân A với B
- * DIV AB : lệnh chia A với B
- * DA A : lệnh hiệu chỉnh thập phân thanh ghi A

Các lệnh số học có thể có 4 khả năng định địa chỉ:

Ví dụ:

- | | | |
|-----|---------|--------------------------|
| ADD | A, 7FH | [định địa chỉ trực tiếp] |
| ADD | A, @R0 | [định địa chỉ gián tiếp] |
| ADD | A, R7 | [định địa chỉ thanh ghi] |
| ADD | A, #35H | [định địa chỉ tức thời] |

Tất cả các lệnh số học được thực thi trong một chu kỳ máy, ngoại trừ lệnh INC DPTR được thực thi trong hai chu kỳ máy, các lệnh MUL AB và DIV AB được thực thi trong 4 chu kỳ máy. Một chu kỳ máy dài 1 microsec (nếu 8051 hoạt động với xung clock 12MHz).

8051 cung cấp kiểu định địa chỉ linh hoạt cho không gian nhớ nội. Một vị trí bất kỳ đều có thể tăng hay giảm bằng cách dùng kiểu định địa chỉ trực tiếp mà không cần qua trung gian thanh chứa A. Ví dụ nếu vị trí 7FH của RAM nội chứa giá trị 40H thì lệnh INC 7FH

Tăng giá trị địa chỉ 7FH thành 41H và đặt kết quả tại 7FH.

8051 cũng có lệnh INC thao tác trên con trỏ dữ liệu 16 bit. Do con trỏ dữ liệu chứa 16-bit địa chỉ của bộ nhớ ngoài, việc tăng nội dung con trỏ này là một đặc trưng thường sử dụng. Trong 8051 không có lệnh giảm nội dung con trỏ dữ liệu và ta phải dùng một chuỗi lệnh sau thay cho điều thiếu sót này:

DEC DPL ; Giảm byte thấp của DPTR

MOV R7, DPL ; Cất vào R7

CJNE R7, #0FFH, SKIP ; So sánh với byte cao của DPTR

SKIP: (tiếp tục) ; Không bằng: Bỏ qua

Các byte thấp và byte cao của DPTR phải được giảm riêng rẽ trong đó byte cao (DPH) chỉ được giảm 1 nếu byte thấp DPL trấn từ 60H qua FFH.

Lệnh MUL AB nhân dữ liệu 8 bit chứa trong thanh ghi B với nội dung thanh chứa A và đặt kết quả 16 bit trong cặp thanh ghi AB (thanh ghi A chứa byte cao, thanh chứa B chứa byte thấp).

Lệnh DIV AB chia nội dung thanh ghi A cho dữ liệu chứa trong thanh ghi B, thương số 8 bit cất trong thanh chứa A và số dư 8 bit cất trong thanh ghi B. Ví dụ thanh chứa A là 25 (19H) và thanh ghi B chứa 6 (60H) thì lệnh:

DIV AB

Là chia 25 cho 6, kết quả 4 cất trong thanh chứa A và số dư 1 cất trong thanh ghi B.

Với số BCD, các lệnh ADD và ADDC phải được tiếp theo bởi lệnh DA A để đảm bảo rằng kết quả ở trong tầm số BCD. Lưu ý là lệnh hiệu đính DA A sẽ không biến đổi số nhị phân thành BCD mà chỉ tạo ra một kết quả hợp lệ và ta thường gọi là hiệu đính trong phép cộng 2 số BCD.

Ví dụ nếu thanh chứa A chứa giá trị BCD là 59 (59H) thì chuỗi lệnh sau:

ADD A, #1

DA A

Trước tiên cộng 1 với nội dung thanh chứa A, kết quả là 5AH. Kết quả này được lệnh thứ hai hiệu đính thành giá trị BCD hợp lệ là 60 (60H) và cất vào thanh chứa A.

2.2. Nhóm lệnh logic

Nhóm lệnh logic gồm có các lệnh sau:

- * ANL A, source : Lệnh nhân logic, AND
- * ANL A, #data
- * ANL direct, A
- * ANL direct, #data
- * ORL A, source : Lệnh cộng logic, OR
- * ORL A, #data
- * ORL direct, A
- * ORL direct, #data
- * XRL A, source : Lệnh cộng module, XOR
- * XRL A, #data
- * XRL direct, A
- * XRL direct, #data
- * CLR A : Lệnh xoá A
- * CPL A : Lệnh lấy bù A
- * RL A : Lệnh quay trái A
- * RLC A : Lệnh quay trái A và cờ nhớ
- * RR A : Lệnh quay phải A
- * RRC A : Lệnh quay phải A và cờ nhớ
- * SWAP A : Lệnh hoán đổi 2 nibble (hai nửa 4 bit)

Nhóm lệnh logic của 8051 thực hiện các phép toán logic (AND, OR, XOR và NOT) trên các byte dữ liệu và thực hiện trên từng bit có cùng giá trị vị trí (trọng số). Nếu thanh chứa A có giá trị 0011101B, lệnh AND logic byte sau đây:

ANL A, #0101001B

Sẽ tạo ra kết quả là 00010001B cất trong thanh chứa A. Điều này được minh họa như sau:

	01010011	(dữ liệu tức thời)
AND	00110101	(giá trị ban đầu chứa trong A)
	00010001	(kết quả chứa trong A)

Các kiểu định địa chỉ cho các lệnh logic cũng giống như các kiểu định địa chỉ cho các lệnh số học, lệnh AND logic có thể có dạng sau:

ANL	A, 55H	(định địa chỉ trực tiếp)
ANL	A, @R0	(định địa chỉ gián tiếp)
ANL	A, R6	(định địa chỉ thanh ghi)
ANL	A, #33H	(định địa chỉ tức thì)

Tất cả các lệnh logic sử dụng thanh chứa A để lưu một toán hạng sẽ được thực thi trong 1 chu kỳ máy, ngược lại nếu sử dụng thanh ghi khác hoặc byte nhớ thay cho thanh chứa A, lệnh phải được thực thi trong 2 chu kỳ máy. Các phép toán logic có thể được thực hiện trên một byte bất kỳ trong bộ nhớ dữ liệu nội mà không cần qua trung gian thanh chứa A. Lệnh "XRL direct, #data" giúp ta nhanh chóng và dễ dàng đảo mức logic các bit của port.

Ví dụ : XRL P1, #0FFH

Thực hiện một thao tác đọc-sửa-ghi 8 bit của port1 được đọc, sau đó từng bit được XOR với các bit tương ứng (cùng vị trí) của dữ liệu tức thời. Vì 8 bit dữ liệu tức thời đều bằng 1, kết quả từng bit của port1 được lấy bù, kết quả này ghi trở lại port1 (vì $A \text{ XOR } 1 = \bar{A}$).

Các lệnh quay (RL và RR $\bar{A}$) dịch thanh ghi A qua trái hoặc qua phải 1 bit. Với lệnh quay trái (RL A), bit có giá trị vị trí lớn nhất MSB được đưa vào vị trí có giá trị thấp nhất LSB. Với lệnh quay phải (RR A), bit có giá trị thấp nhất LSB được đưa vào vị trí có giá trị lớn nhất MSB. Các lệnh RLC A và RRC A là các lệnh quay 9 bit sử dụng thanh chứa A và cờ nhớ CY trong thanh ghi PSW. Ví dụ cờ nhớ CY chứa 1 và thanh chứa A chứa 00H, thì lệnh sau:

RRC A

Sẽ cho kết quả là: cờ nhớ CY chứa 0 và thanh chứa A có nội dung là 80H. Điều này có nghĩa là cờ nhớ CY được đưa đến ACC.7 và ACC.0 được đưa đến cờ nhớ.

Lệnh SWAP A trao đổi nửa thấp 4 bit với nửa cao 4 bit trong thanh chứa A với nhau. Lệnh này thường dùng trong các phép toán số BCD. Ví dụ nếu thanh chứa A chứa một số nhị phân đã biết và có giá trị nhỏ hơn $100_{(10)}$, ta có thể biến đổi số nhị phân này thành số BCD bằng các dòng lệnh sau:

MOV B, #10

DIV AB

SWAP A

ADD A, B

Việc cho một số 10 trong hai lệnh đầu tiên tạo ra digit chục trong nửa thấp của thanh chứa A và digit đơn vị trong thanh chứa B. Lệnh SWAP và ADD di chuyển digit chục đến nửa cao của thanh chứa A và digit đơn vị vào nửa thấp của thanh chứa này.

2.3. Nhóm lệnh di chuyển dữ liệu

Các lệnh di chuyển dữ liệu bao gồm các lệnh sau:

- * MOV A, source : Lệnh chuyển toán hạng nguồn đến đích
- * MOV A, #data
- * MOV dest, A
- * MOV dest, source
- * MOV dest, #data
- * MOV DPTR, #data16
- * MOVC A,@A+DPTR :Lệnh di chuyển từ bộ nhớ chương trình
- * MOVC A,@A+PC
- * MOVX A,@Ri :Lệnh di chuyển từ bộ nhớ dữ liệu
- * MOVX A,@DPTR
- * MOVX @Ri,A
- * MOVX @DPTR, A
- * PUSH direct : Lệnh cất vào stack
- * POP direct : Lệnh lấy ra từ stack
- * XCH A, source : Lệnh trao đổi các byte
- * XCHD A, @Ri : Lệnh trao đổi các digit thấp

• Trong RAM nội

Các lệnh di chuyển dữ liệu bên trong không gian nhớ nội được thực thi trong 1 hay 2 chu kỳ máy. Dạng của lệnh như sau:

MOV <destination>,<source>

Lệnh trên cho phép dữ liệu được di chuyển giữa các vị trí của RAM nội hoặc các thanh ghi chức năng đặc biệt SFR mà không cần qua trung gian thanh chứa A. Cần nhớ là 128 byte cao của RAM dữ liệu (đối với 8032/8052) chỉ được truy xuất bằng kiểu định địa chỉ gián tiếp và các thanh ghi chức năng đặc biệt chỉ được truy xuất bằng kiểu định địa chỉ trực tiếp.

Một đặc trưng làm cho cấu trúc của MCS-51 khác với cấu trúc của hầu hết các bộ vi xử lý là vùng stack thường trú trong RAM trên chip (RAM nội) và tăng dần về phía trên của bộ nhớ, phía các địa chỉ cao hơn. Lệnh PUSH trước tiên tăng con trỏ stack (SP) rồi sao chép byte vào trong stack. Các lệnh PUSH và POP sử dụng kiểu định địa chỉ trực tiếp để nhận biết byte được cất hoặc được phục hồi nhưng bản thân stack được truy xuất bởi kiểu định địa chỉ gián tiếp sử dụng con trỏ stack SP.

Điều này có nghĩa là vùng stack có thể sử dụng 128 byte cao của bộ nhớ RAM nội trên 8032/8052. 128 byte cao này không có trong 8031/8051. Với các bộ vi điều khiển này, nếu nội dung của SP vượt quá 7FH (127) (nghĩa là nội dung của SP lớn hơn 7FH), các byte được PUSH sẽ bị mất còn các byte được POP không được xác định.

Các lệnh chuyển dữ liệu còn bao gồm lệnh MOV 16-bit dùng để khởi động con trỏ dữ liệu DPTR cho mục đích tìm kiếm các bảng trong bộ nhớ chương trình hoặc cho mục đích truy xuất bộ nhớ dữ liệu ngoài 16 bit.

Lệnh hoán đổi nội dung XCH được sử dụng để hoán đổi nội dung của thanh chứa A với nội dung của byte được chỉ ra trong lệnh. Dạng lệnh như sau:

XCH A,<source>.

Lệnh trên là cho thanh chứa A và byte định địa chỉ trao đổi dữ liệu với nhau. Việc trao đổi một digit được sử dụng lệnh có dạng:

XCHD A, @Ri

Cũng hoạt động tương tự, tuy nhiên chỉ có các nửa thấp của các byte được trao đổi với nhau. Ví dụ nếu thanh chứa A chứa F3H, R1 chứa 40H và tại địa chỉ 40H trong RAM nội chứa 5BH, lệnh:

XCHD A, @R1

Cho kết quả là A chứa FBH và tại địa chỉ 40H trong RAM nội chứa 53H.

• Trong RAM ngoài

Với các lệnh mà việc di chuyển dữ liệu cho phép dữ liệu được di chuyển giữa RAM nội với RAM ngoài, ta phải sử dụng kiểu định địa chỉ gián tiếp. Các địa chỉ gián tiếp được xác định bằng cách dùng địa chỉ 1 byte (như @ Ri, trong đó Ri là R0 hoặc R1 của dãy thanh ghi được chọn) hoặc địa chỉ 2 byte (như @DPTR). Điểm bất lợi khi dùng địa chỉ 16 bit là tất cả 8 bit của port 2 phải được dùng như byte cao của bus địa chỉ và điều này sẽ không cho ta sử dụng cổng 2 làm cổng xuất/nhập. Ngược lại các địa chỉ 8 bit cho phép ta truy xuất đến một vài KB của RAM mà không cần sử dụng toàn bộ port 2.

Tất cả các lệnh di chuyển dữ liệu hoạt động trên bộ nhớ ngoài được thực thi trong 2 chu kỳ máy và sử dụng thanh chứa làm toán hạng nguồn hoặc đích. Các tín hiệu dùng để truy xuất RAM ngoài (RD đảo và WR đảo) chỉ tích cực trong khi lệnh MOVX được thực thi. Bình thường các tín hiệu này không tích cực (mức cao) và nếu bộ nhớ ngoài không được sử dụng, các đường RD (đảo) và WR (đảo) có chức năng như các đường xuất/nhập.

- **Các bảng tìm kiếm.**

Có hai lệnh di chuyển dữ liệu dành cho việc đọc các bảng tìm kiếm trong bộ nhớ chương trình. Do bởi các lệnh này truy xuất bộ nhớ chương trình, các bảng tìm kiếm chỉ có thể được đọc và không được cập nhật. Mã gọi nhớ của lệnh là MOVC (move constant: di chuyển hằng). MOVC sử dụng hoặc bộ đếm chương trình hoặc con trỏ dữ liệu làm thanh ghi nền và thanh chứa A chứa địa chỉ offset. Lệnh sau:

```
MOVC A,@A+DPTR
```

Có thể truy xuất một bảng 256 điểm nhập được đánh số từ 0 đến 255. Số của điểm nhập yêu cầu được nạp cho thanh chứa A và con trỏ dữ liệu được khởi động để chứa địa chỉ đầu bảng. Lệnh sau:

```
MOVC A,@A+PC
```

Cũng hoạt động tương tự, ngoại trừ ở đây bộ đếm chương trình được dùng để chứa địa chỉ nền và bảng được truy xuất nhờ vào một chương trình con. Trước tiên số của điểm nhập yêu cầu được nạp cho thanh chứa A, sau đó chương trình con được gọi. Chuỗi lệnh cho phép khởi động và gọi có thể là:

```
MOV A, ENTRY-NUMBER
```

```
CALL LOOK-UP
```

```
.  
.   
.
```

```
LOOK-UP:      INC
```

```
MOV A, @A+PC
```

```
RET
```

```
TABLE: DB data,data,data...
```

Bảng được định nghĩa ngay sau lệnh RET trong chương trình. Lệnh tăng được cần đến do PC trở tới lệnh RET khi lệnh MOVC được thực thi. Việc tăng nội dung thanh chứa sẽ bỏ qua lệnh RET.

2.4. Nhóm lệnh xử lý bit

Nhóm các lệnh thao tác trên bit bao gồm các lệnh sau:

* CLR	C	:Lệnh xoá bit
* CLR	bit	
* SETB	C	:Lệnh set bit bằng 1
* SETB	bit	
* CPL	C	:Lệnh lấy bù bit
* CPL	bit	
* ANL	C, bit	:Lệnh AND với bit C
* ANL	C, /bit	:Lệnh AND NOT bit với C
* ORL	C, bit	:Lệnh OR với bit C
* ORL	C, /bit	:Lệnh OR NOT bit với C
* MOV	C, bit	:Lệnh di chuyển bit đến bit
* MOV	bit, C	
* JC	rel	:Lệnh nhảy đến C bằng 1
* JNC	rel	:Lệnh nhảy đến C bằng 0
* JB	bit, rel	:Lệnh nhảy nếu bit bằng 1
* JNB	bit, rel	:Lệnh nhảy nếu bit bằng 0
* JBC	bit, rel	:Lệnh nhảy nếu bit bằng 1 rồi xoá bit

Bộ xử lý của 8051 chứa 1 bộ xử lý logic trên bit cho phép ta thực hiện các phép toán đơn bit. RAM nội chứa 128 bit được định địa chỉ và không gian SFR hỗ trợ thêm đến 128 bit được định địa chỉ. Tất cả các đường port đều có địa chỉ bit và mỗi đường có thể được xử lý như là một port đơn bit riêng rẽ. Các lệnh truy xuất các bit này không chỉ là các lệnh rẽ nhánh có điều kiện mà còn là các lệnh di chuyển, set, xoá, lấy bù, OR và AND. Các thao tác trên bit như vậy (một trong các đặc trưng mạnh của MCS-51) không dễ dàng có được trong các cấu trúc khác, các cấu trúc sử dụng thao tác hướng byte.

Mọi thao tác truy xuất bit đều sử dụng kiểu định địa chỉ trực tiếp với các địa chỉ từ 00H đến 7FH trong 128 vị trí thấp, và từ địa chỉ 80H đến FFH trong không gian SFR. Các địa chỉ bit ở 128 vị trí thấp thuộc các byte có địa chỉ từ 20H đến 2FH được đánh số liên tục từ bit 0 của byte ở địa chỉ 20H (bit 00H) đến bit 7 của byte ở địa chỉ 2FH (bit 7FH).

Các bit có thể được set và xoá bằng 1 lệnh. Điều khiển đơn bit được dùng cho nhiều thiết bị xuất/nhập, bao gồm xuất ra rơ le, động cơ, cuộn dây, các LED, mạch còi báo động, loa hoặc nhập từ các chuyển mạch hoặc các bộ chỉ thị trạng thái. Nếu một còi báo động nối-với bit 7 của port1, ta có thể tác động mạch còi bằng cách set bit của port:

```
SETB    P1.7
```

Hoặc tắt còi bằng cách xoá bit của port:

```
CLR      P1.7
```

Trình dịch hợp ngữ sẽ biến đổi ký hiệu P1.7 thành địa chỉ bit là 97H. Ví dụ sau cho phép ta di chuyển 1 cờ vào một chân port:

```
MOV      C, FLAG
```

```
MOV      P1.0, C
```

Trong ví dụ trên FLAG là tên của một bit được định nghĩa chỉ trong 128 vị trí thấp hoặc trong không gian SFR. Một đường xuất/nhập (ở ví dụ trên là bit 0 của port1) được set hoặc xoá phụ thuộc vào bit cờ có giá trị 1 hay 0. Bit nhớ trong PSW được dùng như một thanh chứa đơn bit của bộ xử lý logic trên bit, các lệnh đơn bit liên quan đến bit nhớ ký hiệu là C là các lệnh đặc biệt liên quan đến cờ nhớ (như là CLR C). Bit nhớ cũng có một địa chỉ trực tiếp vì bit này được lưu trữ trong thanh ghi PSW, thanh ghi này được định địa chỉ từng bit.

Cũng giống như các thanh ghi khác được định địa chỉ từng bit trong không gian SFR, các bit của PSW có mã gọi nhớ mà trình dịch hợp ngữ sẽ chấp nhận thay cho địa chỉ bit. Mã gọi nhớ của cờ nhớ là CY được định nghĩa thay cho địa chỉ bit 0D7H. Ta hãy khảo sát hai lệnh sau:

```
CLR      C
```

```
CLR      CY
```

Cả hai đều có công dụng, tuy nhiên dạng lệnh trước là lệnh 1 byte trong khi dạng lệnh sau là lệnh 2 byte. Trong dạng lệnh sau byte thứ 2 là địa chỉ trực tiếp của bit được xác định - cờ nhớ.

Các lệnh logic trên bit bao gồm cả lệnh ANL và ORL nhưng không bao gồm lệnh XRL. Nếu ta cần XOR 2 bit, BIT1 và BIT2, và kết quả cất trong cờ nhớ, các lệnh sau được sử dụng:

```
MOV      C, BIT1
```

```
JNB      BIT2, SKIP
```

```
CPL      C
```

SKIP:

Trước tiên BIT1 được nạp cho cờ nhớ. Nếu BIT2 = 0, thì C đã chứa kết quả (nghĩa là BIT1 XOR BIT2 nếu BIT2 = 0). Nếu BIT2 = 1, C chứa kết quả là bù của cờ nhớ. Việc lấy bù C hoàn tất phép toán XOR.

Chương trình trong ví dụ trên sử dụng lệnh JNB, một trong chuỗi lệnh kiểm tra bit. Các lệnh này sẽ nhảy đến bit được định địa chỉ được set bằng 1 (như lệnh JC, JB, JNB). Trong ví dụ trên nếu BIT2 = 0, lệnh CPL C được bỏ qua. Lệnh JBC (nhảy đến bit được set, sau đó xoá bit) thực hiện việc nhảy nếu bit được định địa chỉ được set và cùng xoá bit, vậy thì một cờ nhớ có thể được kiểm tra và xoá bằng một lệnh.

Tất cả các bit của PSW đều có thể định địa chỉ trực tiếp, do vậy bit chẩn lẻ hoặc các cờ đa mục đích cũng hợp lệ đối với các lệnh kiểm tra bit.

2.5. Nhóm lệnh rẽ nhánh

Nhóm các lệnh rẽ nhánh bao gồm các lệnh sau:

* ACALL addr11	:Lệnh gọi chương trình con
* LCALL addr16	
* RET	:Lệnh quay về từ chương trình con
* RETI	:Lệnh quay về từ trình phục vụ ngắt
* AJMP addr11	:Lệnh nhảy
* LJMP addr16	
* SJMP rel	
* JMP @A+DPTR	
* JZ rel	:Lệnh nhảy nếu A bằng 0
* JNZ rel	:Lệnh nhảy nếu A khác 0
* CJNE A, direct, rel	:Lệnh so sánh và nhảy
* CJNE A, #data, rel	
* CJNE Rn, #data, rel	
* CJNE @Ri, #data, rel	
* DJNZ Rn, rel	:Lệnh giảm và nhảy nếu khác 0
* DJNZ direct, rel	
* NOP	:Lệnh không làm gì

Trong tập lệnh của 8051 có nhiều lệnh điều khiển luồng chương trình, bao gồm các lệnh gọi thủ tục và quay về từ một thủ tục, rẽ nhánh có điều kiện hoặc

không có điều kiện. Các khả năng này được cải tiến hơn nữa bởi 3 kiểu định địa chỉ cho các lệnh rẽ nhánh chương trình. Có 3 biến thể của lệnh nhảy: SJMP, LJMP và AJMP (sử dụng định địa chỉ tương đối, dài và tuyệt đối).

Trình dịch hợp ngữ của Intel (ASM51) cho phép sử dụng mã gọi nhớ JMP nếu người lập trình không quan tâm đến các biến thể. Trình dịch hợp ngữ của các công ty khác có thể không hỗ trợ đặc tính này. JMP tổng quát dịch thành AJMP nếu đích nhảy đến không chứa tham chiếu thuận và ở trong một trang 2K. Ngược lại, JMP được dịch thành LJMP. Lệnh CALL cũng hoạt động theo cách này.

Lệnh SJMP xác định địa chỉ đích là offset tương đối như đã bàn đến trước đây khi đề cập đến các kiểu định địa chỉ. Vì lệnh dài 2 byte (bao gồm một opcode cộng với một offset tương đối 8 bit), khoảng cách nhảy được giới hạn từ -128 đến +127 byte so với địa chỉ của lệnh theo sau lệnh SJMP.

Lệnh SJMP xác định địa chỉ đích là hằng số 16 bit. Vì lệnh dài 3 byte (bao gồm 1 opcode cộng với hai byte địa chỉ), địa chỉ đích có thể ở bất cứ đâu trong không gian nhớ chương trình 64K.

Lệnh AJMP xác định địa chỉ đích là một hằng số 11 bit. Cũng như lệnh SJMP, lệnh AJMP cũng dài 2 byte nhưng được mã hoá khác. Byte opcode sẽ chứa 3 trong 11 bit địa chỉ và 2 byte chứa 8 bit thấp của địa chỉ đích. Khi lệnh được thực thi, 11 bit này thay chỗ cho 11 bit thấp trong PC còn 5 bit cao của PC vẫn giữ nguyên. Địa chỉ đích do vậy phải ở trong cùng một trang 2K với lệnh theo sau lệnh AJMP. Do đó có không gian nhớ chương trình là 64K, ta có 32 trang và mỗi trang bắt đầu ở địa chỉ là biên của 2K (như là 0000H, 0800H, 1000H, 1800H...cho đến F800H).

Trong mọi trường hợp người lập trình xác định địa chỉ đích cho trình dịch hợp ngữ theo cách thông thường như là nhân hoặc là 1 hằng số 16 bit. Trình dịch hợp ngữ sẽ đặt địa chỉ đích theo đúng khuôn dạng của từng lệnh. Nếu khuôn dạng yêu cầu bởi lệnh không được hỗ trợ khoảng cách dùng để xác định địa chỉ đích, một thông báo "địa chỉ đích ngoài tầm" sẽ được đưa ra.

*** Các bảng nhảy**

Lệnh JMP @A+DPTR hỗ trợ các thao tác nhảy phụ thuộc vào trường hợp cụ thể cho các bảng nhảy. Địa chỉ đích được tính ở thời điểm thực thi lệnh là tổng của nội dung thanh ghi DPTR 16 bit với nội dung của thanh chứa A. DPTR được nạp địa chỉ của bảng nhảy và thanh chứa A đóng vai trò của một

thanh ghi chỉ số. Ví dụ nếu có 5 trường hợp được yêu cầu, một giá trị 0 đến 4 được nạp cho thanh chứa A và một nhảy cho trường hợp tương thích được thực hiện như sau:

```
MOV DPTR, #JUMP_TABLE
MOV A, INDEX_NUMBER
RL A
JMP @A+DPTR
```

Lệnh RL A ở trên biến đổi chỉ số (0 → 4) thành một số chẵn trong tầm từ 0 cho đến 8 vì mỗi điểm nhập trong bảng nhảy là 1 địa chỉ 2 byte:

```
JUMP_TABLE:  AJMP      CASE0
               CASE1
               CASE2
               CASE3
```

*** Chương trình con và ngắt**

Có 2 biến thể cho lệnh CALL: ACALL và LCALL sử dụng kiểu định địa chỉ tuyệt đối và dài. Cũng như lệnh JMP, mã gọi nhớ CALL có thể được sử dụng với trình dịch hợp ngữ của Intel nếu người lập trình không quan tâm đến cách định địa chỉ. Cả 2 lệnh trên đều cất nội dung của bộ đếm chương trình vào stack và nạp cho bộ đếm chương trình địa chỉ đã được xác định trong lệnh. Lưu ý là PC sẽ chứa địa chỉ của lệnh theo sau lệnh CALL khi nội dung thanh ghi này được cất vào stack. Khi nạp vào stack, byte thấp nạp trước và byte cao nạp sau. Các byte được lấy ra từ stack theo trình tự ngược lại. Lấy ví dụ nếu lệnh CALL được chứa trong bộ nhớ chương trình ở các địa chỉ là 1000H-1002H và con trỏ stack chứa 20H, lệnh LCALL sẽ:

- Nạp địa chỉ quy về 1003H và stack (đặt 03H tại địa chỉ 21H và 10H tại địa chỉ 22H).
- Nội dung của con trỏ stack bây giờ là 22H.
- Nhảy đến chương trình con bằng cách nạp cho PC địa chỉ chứa trong byte 2 và byte 3 của lệnh.

Các lệnh LCALL và ACALL cũng có các hạn chế trên địa chỉ đích như các lệnh LJMP và AJMP. Các thủ tục cần được kết thúc bằng lệnh RET, lệnh này trả việc thực thi chương trình trở về lệnh theo sau lệnh CALL. Không có điều gì bí ẩn về cách mà lệnh RET trả điều khiển về cho chương trình chính. Đơn

giản là lệnh này lấy lại (POP) 2 byte sau cùng ra khỏi stack và nạp chúng cho bộ đếm chương trình. Một quy luật chủ yếu cho việc lập trình với các thủ tục là chúng luôn luôn được gọi bởi lệnh CALL và luôn luôn trả điều khiển về chương trình gọi bởi lệnh RET. Các thao tác nhảy vào hoặc nhảy ra khỏi một thủ tục bằng một cách khác nào đó thường làm rối vùng stack và làm cho chương trình bị dừng.

Lệnh RETI trả điều khiển về chương trình gọi từ một trình phục vụ ngắt (ISR: interrupt service routine). Điểm khác nhau giữa RET và RETI là RETI báo hiệu cho hệ thống điều khiển ngắt rằng quá trình xử lý ngắt đã xong. Nếu không có một ngắt nào duy trì trong thời gian RETI được thực thi, RETI hoạt động giống RET.

*** Nhảy có điều kiện**

8051 cung cấp cho ta nhiều lệnh nhảy có điều kiện. Tất cả các lệnh này xác định địa chỉ đích bằng kiểu định địa chỉ tương đối và cùng bị giới hạn ở khoảng cách nhảy từ - 128 byte đến + 127 byte kể từ lệnh theo sau lệnh nhảy có điều kiện. Tuy nhiên cần lưu ý là người lập trình sẽ xác định địa chỉ đích theo cùng cách với các lệnh nhảy khác bằng cách dùng nhãn hoặc bằng số 16 bit. Trình dịch hợp ngữ sẽ thực hiện các việc còn lại. Không có bit 0 trong PSW. Các lệnh JZ và JNZ kiểm tra dữ liệu trong thanh chứa cho điều kiện này.

Lệnh DJNZ (giảm và nhảy nếu khác không) dành cho điều khiển lặp vòng. Để thực thi một vòng lặp N lần, ta nạp một byte số lần đếm N cho một thanh ghi và kết thúc vòng lặp với DJNZ trở tới điểm bắt đầu vòng lặp.

MOV R7, #10

LOOP: (bắt đầu vòng lặp)

(kết thúc vòng lặp)

DJNZ R7, LOOP

(tiếp tục)

Lệnh CJNE (so sánh và nhảy nếu không bằng) cũng dành cho việc điều khiển vòng lặp. Hai byte được xác định trong trường toán hạng của lệnh và việc nhảy chỉ được thực thi nếu 2 byte khác 0. Ví dụ, nếu một ký tự vừa được đọc vào thanh chứa A từ port nối tiếp và ta muốn nhảy đến một lệnh được nhận biết bởi nhãn TERMINATE nếu ký tự đó là control-C (03H), các dòng lệnh sau được sử dụng:

CJNE A, #03H, SKIP

SJMP REMINATE

SKIP;

Vì thao tác nhảy chỉ xuất hiện nếu thanh chứa A chứa mã của control-C, một nhãn SKIP (bỏ qua) được dùng để bỏ qua việc kết thúc lệnh nhảy ngoại trừ khi mã yêu cầu được đọc. Lệnh trên còn được ứng dụng trong các phép so sánh lớn hơn hay nhỏ hơn. Hai byte trong trường toán hạng là các số nguyên không dấu. Nếu byte đầu nhỏ hơn byte thứ hai, cờ nhớ được set lên 1. Nếu byte đầu lớn hơn hoặc bằng byte thứ hai, cờ nhớ được xoá. Ví dụ ta muốn nhảy đến BIG nếu giá trị trong thanh chứa A lớn hơn hoặc bằng 20H, các giá trị sau được dùng:

CJNE A, #20H, \$+3

JNC BIG

Đích nhảy cho lệnh CJNE được xác định là \$+3. Dấu \$ là một ký hiệu của trình dịch hợp ngữ biểu thị địa chỉ của lệnh hiện hành. Vì CJNE là lệnh 3 byte, \$+3 là địa chỉ lệnh tiếp theo, JNC. Mặt khác, lệnh CJNE theo sau bởi lệnh JNC không quan tâm đến kết quả so sánh. Mục đích duy nhất của việc so sánh là để set hay xóa cờ nhớ và lệnh JNC quyết định nhảy hay không nhảy. Ví dụ này là một thể hiện trong đó 8051 tiếp cận với một tình huống lập trình tổng quát một cách vụng về hơn so với hầu hết các bộ vi xử lý. Việc sử dụng các macro cho phép ta có các chuỗi lệnh mạnh được cấu trúc và thực thi bằng cách dùng một mã lệnh gọi nhớ duy nhất.

Câu hỏi ôn tập

1. Cấu trúc của bộ vi điều khiển 8051 gồm có những khối cơ bản nào? Các khối trong cấu trúc và các tín hiệu điều khiển có chức năng gì?
2. Trình bày nguyên lý hoạt động của bộ vi điều khiển 8051.
3. Bộ vi điều khiển 8051 có thể ghép nối với các thiết bị ngoại vi nào? Phân tích một ghép nối cụ thể.
4. Bộ vi điều khiển 8051 có bao nhiêu chế độ địa chỉ? Các chế độ địa chỉ đó là gì?
5. Bộ vi điều khiển 8051 có bao nhiêu loại lệnh, mỗi loại hãy lấy một ví dụ minh họa.

TÀI LIỆU THAM KHẢO

1. *Máy tính cấu trúc và lập trình T1, T2*, Nguyễn Tăng Cường, Nhà xuất bản Khoa học kỹ thuật, 2004.
2. *Kỹ thuật ghép nối máy tính*, Nguyễn Mạnh Giang, Nhà xuất bản Khoa học kỹ thuật, 2004.
3. *Cẩm nang lập trình hệ thống T1, T2*, Nguyễn Mạnh Hùng, Nhà xuất bản Khoa học kỹ thuật.
4. *Kỹ thuật vi xử lý*, Văn Thế Minh, Nhà xuất bản Giáo dục, 1997.
5. *Kỹ thuật vi xử lý và lập trình assembly cho hệ vi xử lý*, Đỗ Xuân Tiến, Nhà xuất bản Khoa học kỹ thuật, 2002.
6. *Cấu trúc máy vi tính*, Trần Quang Vinh, Nhà xuất bản Khoa học kỹ thuật, 1997.
7. *Hệ vi điều khiển 8051*, Tống Văn On, Hoàng Đức Hải, Nhà xuất bản Lao động - Xã hội, 2001.
8. *Kỹ thuật vi điều khiển với AVR*, Ngô Diên Tập, Nhà xuất bản Khoa học kỹ thuật, 2003.
9. *Intel Processors*, Glencoe, Roy W. Goody, 1993.
10. *Computer organization and architecture*, William Stallings, Prentice - Hall International, 1996.

MỤC LỤC

<i>Lời giới thiệu</i>	3
<i>Lời nói đầu</i>	5
<i>Bài mở đầu</i>	7
Chương 1. KHÁI QUÁT CHUNG VỀ VI XỬ LÝ	9
I. Lịch sử phát triển của vi xử lý.....	9
II. Cấu trúc chung của một máy vi tính.....	12
Chương 2. BIỂU DIỄN THÔNG TIN TRONG MÁY VI TÍNH	16
I. Các dạng dữ liệu cơ bản.....	16
II. Biểu diễn số trong máy vi tính.....	18
Chương 3. BỘ VI XỬ LÝ 8088 VÀ CÁC BỘ VI XỬ LÝ TIỀN TIẾN	34
I. Bộ vi xử lý 8088.....	34
II. Nguyên tắc hoạt động của 8088	42
III. Các bộ vi xử lý tiên tiến	50
Chương 4. LẬP TRÌNH HỢP NGỮ VỚI BỘ VI XỬ LÝ 8088	58
I. Giới thiệu chung.....	59
II. Cấu trúc chương trình	64
III. Các chế độ địa chỉ của 8088.....	80
IV. Tập lệnh của 8088.....	87
Chương 5. BỘ NHỚ	114
I. Khái quát chung về bộ nhớ	115
II. Bộ nhớ ROM	119
III. Bộ nhớ RAM	122
IV. Ghép nối vi xử lý 8088 với bộ nhớ	129
Chương 6. GHÉP NỐI VI XỬ LÝ VỚI THIẾT BỊ NGOẠI VI	140
I. Tổng quan về sự ghép nối	140
II. Giải mã địa chỉ cho thiết bị vào ra.....	146

III. Mạch phối ghép vào/ra 8255A	147
IV. Các phương pháp điều khiển vào/ra	154
Chương 7. CÁC MẠCH GHÉP NỐI PHỤ TRỢ KHÁC	162
I. Vi mạch đếm khoảng thời gian lập trình 8253	162
II. Vi mạch điều khiển thâm nhập bộ nhớ trực tiếp 8257.....	166
III. Vi mạch điều khiển ngắt 8259	171
IV. Vi mạch điều khiển ghép nối 8251	179
V. Một số ghép nối cơ bản	183
Chương 8. VI ĐIỀU KHIỂN	196
I. Khái quát chung về vi điều khiển	196
II. Cấu trúc phần cứng của họ vi điều khiển 8051.....	197
III. Giới thiệu bộ lệnh của vi điều khiển 8051	207
<i>Tài liệu tham khảo</i>	227

NHÀ XUẤT BẢN HÀ NỘI
4 - TỐNG DUY TÂN, QUẬN HOÀN KIẾM, HÀ NỘI
ĐT: (04) 8252916, 8257063 - FAX: (04) 9289143

GIÁO TRÌNH
KỸ THUẬT VI XỬ LÝ
NHÀ XUẤT BẢN HÀ NỘI - 2006

Chịu trách nhiệm xuất bản

NGUYỄN KHÁC OÁNH

Biên tập

PHẠM QUỐC TUẤN

Bìa

TRẦN QUANG

Kỹ thuật vi tính

MINH ĐỖ

Sửa bản in

PHẠM QUỐC TUẤN

LÊ XUÂN THỌ

In 870 cuốn, khổ 17x24cm tại Công ty cổ phần in 15. Quyết định xuất bản số: 154 - 2006/CXB/585GT - 15/HN cấp ngày 06/12/2006. In xong và nộp lưu chiểu quý I/2007.

BỘ GIÁO TRÌNH XUẤT BẢN NĂM 2006
KHOẢNG TRƯỜNG TRUNG HỌC ĐIỆN TỬ - ĐIỆN LẠNH

1. LÝ THUYẾT MẠNG VÀ HỆ THỐNG ĐIỀU HÒA MẠNG
2. THIẾT KẾ HỆ THỐNG LẠNH
3. THỰC HÀNH BẢO TRÌ HỆ THỐNG MÁY TÍNH
4. THỰC HÀNH SỬA CHỮA MÁY LẠNH
5. ĐỒNG BỘ VÀ BÁO HIỆU TRONG MẠNG VIỄN THÔNG
6. TỔ CHỨC MẠNG VÀ DỊCH VỤ VIỄN THÔNG
7. THIẾT BỊ ĐẦU CUỐI
8. NHIỆT KỸ THUẬT
9. MÀN HÌNH VÀ CARD ĐIỀU KHIỂN MÀN HÌNH
10. ĐO LƯỜNG KỸ THUẬT LẠNH
11. THỰC HÀNH KỸ THUẬT SỐ
12. THỰC HÀNH ĐIỆN TỬ CÔNG NGHIỆP
13. CẤU TRÚC MÁY TÍNH
14. LÝ THUYẾT BẢO TRÌ HỆ THỐNG MÁY TÍNH
15. KỸ THUẬT VI XỬ LÝ
16. KỸ THUẬT SỐ VÀ MẠCH LOGIC
17. THÔNG TIN QUANG
18. HỆ ĐIỀU HÀNH LINUX
19. THỰC HÀNH MẠNG LAN
20. KỸ THUẬT ĐIỀU HÒA KHÔNG KHÍ
21. THỰC HÀNH GIA CÔNG LẮP ĐẶT ĐƯỜNG ỐNG
22. MÁY VÀ THIẾT BỊ LẠNH
23. THỰC HÀNH MÀN HÌNH MÁY TÍNH
24. THỰC HÀNH VIỄN THÔNG CÔNG NGHỆ MẠNG



GT KT vi xử lý



1011080000115

31,500

Giá: 31.500đ