NGUYỄN TĂNG CƯỜNG, PHAN QUỐC THẮNG

MICROCONTROLLER

CẤU TRÚC VÀ LẬP TRÌNH HỌ VI ĐIỀU KHIỂN 8051

NHÀ XUẤT BẢN
KHOA HỌC
VÀ KỸ THUẬT

NGUYỄN TĂNG CƯỜNG, PHAN QUỐC THẮNG

CẤU TRÚC VÀ LẬP TRÌNH HỌ VI ĐIỀU KHIỂN 8051

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
HÀ NỘI - 2004

MỤC LỤC

LỜI NÓI ĐẦU

Ngày nay, các bộ vi điều khiển đang có ứng dụng ngày càng rộng rãi và thâm nhập ngày càng nhiều trong các lĩnh vực kỹ thuật và đời sống xã hội. Hầu hết các thiết bị kỹ thuật từ phức tạp cho đến đơn giản như thiết bị điều khiển tự động, thiết bị văn phòng cho đến các thiết bị trong gia đình đều có dùng các bộ vi điều khiển. Mặc dù có nhiều ứng dụng thực tế như vậy nhưng những tài liệu hiện có chưa đáp ứng được nhu cầu về học tập, giảng dạy và nghiên cứu ứng dụng các bộ vi điều khiển đang ngày càng tăng hiện nay.

Xuất phát từ thực tế đó, mục đích chính của tài liệu muốn tập trung giới thiệu những kiến thức cơ bản nhất về cấu trúc và lập trình hệ vi điều khiển. Tài liệu đặc biệt chú trọng giới thiệu phần ứng dụng bao gồm tổ chức các hệ thực tiễn và phương pháp lập trình cho các hệ đó.

Tài liệu này là nằm trong loạt các tài liệu đã được Bộ môn Tự động và Kỹ thuật tính, Khoa Kỹ thuật Điều khiển ấn hành, bao gồm cấu trúc máy tính, cấu trúc và lập trình các hệ xử lý tín hiệu số, nay là cấu trúc và lập trình các hệ vi điều khiển.

Tài liệu được chia thành 15 chương và được sắp xếp như sau:

Chương 1 giới thiệu lịch sử phát triển, tổng quan chung về họ vi điều khiển 8051 và các thành viên như 8751, 89C51, DS5000 và 8031.

Chương 2 trình bày về kiến trúc của họ 8051 và những khái niệm cơ bản về lập trình hợp ngữ với 8051.

Chương 3 có chủ đề về tổ chức vòng lặp, lệnh nhảy và lệnh rẽ nhánh cùng một loạt các chương trình ví dụ.

Chương 4 dành cho tổ chức cổng vào ra của 8051 và cách lập trình. Chương này cho phép các sinh viên thực hiện các thử nghiệm đầu tiên về tổ chức giao diện vào/ra của 8051.

Chương 5 đề cập tới các chế độ định địa chỉ của 8051, cách truy cập dữ liệu và mã lệnh trong không gian nhớ của 8051.

Chương 6, chương 7 và chương 8 bàn về các lệnh số học, lệnh logic, lệnh xử lý bit và cách lập trình ứng dụng.

Chương 9 cung cấp các thông tin về bộ đếm/ bộ định thời và phương pháp lập

trình ứng dụng.

Chương 10 nghiên cứu truyền tin nối tiếp trên 8051, tổ chức giao diện với RS232 và truyền thông qua cổng COM của máy tính IBM PC.

Chương 11 bàn về một chủ đề rất quan trọng trong triển khai các ứng dụng của 8051, đó là tổ chức ngắt và cách lập trình với các ngắt.

Chương 12 và 13 bao gồm các ứng dụng thường gặp trong thực tế là tổ chức nối ghép với các thiết bị ngoại vi như với đèn LED, bộ biến đổi ADC, bộ cảm biến, bàn phím, động cơ bước và các bộ biến đổi DAC.

Hai chương cuối cùng là 14 và 15 đi sâu vào hai chủ đề quan trọng là tổ chức nối ghép 8031/51 với bộ nhớ ngoài và với vi mạch phục vụ cho vào ra 8255.

Cuốn sách được dùng làm giáo trình giảng dạy bậc đại học và sau đại học chuyên ngành điện, điện tử hoặc làm tài liệu tham khảo cho các nghiên cứu sinh và cho những ai quan tâm đến cấu trúc và lập trình cho các hệ vi điều khiển.

Cuốn sách được biên soạn bởi PGS. TS. Nguyễn Tăng Cường và TS. Phan Quốc Thắng, do PGS. TS. Nguyễn Tăng Cường chủ biên và được phân công như sau: PGS. TS. Nguyễn Tăng Cường viết từ chương 1 đến chương 7, các phần còn lại do TS. Phan Quốc Thắng đảm nhiệm.

Nhân dịp này, tập thể tác giả xin bày tỏ lời cảm ơn chân thành nhất đến những người đã có nhiều đóng góp trong quá trình hoàn thành tài liệu, đến các anh chị em Bộ môn Tự động và Kỹ thuật tính thuộc Khoa Kỹ thuật Điều khiển, Học viện Kỹ thuật Quân sự, đặc biệt phải kể đến sự hỗ trợ hiệu quả của TS. Trần Văn Hợp và kỹ thuật viên Lê Văn Minh.

Do kinh nghiệm và thời gian hạn chế, tài liệu này chắc chắn không thể tránh khỏi những thiếu sót. Rất mong nhận được các ý kiến đóng góp và xây dựng của bạn đọc gần xa. Ý kiến đóng góp xin gửi về địa chỉ: Bộ môn Tự động và Kỹ thuật tính, Khoa Kỹ thuật Điều khiển, Học viện Kỹ thuật Quân sự, 100 Hoàng Quốc Việt, Hà Nội; Điện thoại (04)7542281, email: tcuong@hn.vnn.vn.

Hà Nội, tháng 8 năm 2003

Tập thể tác giả

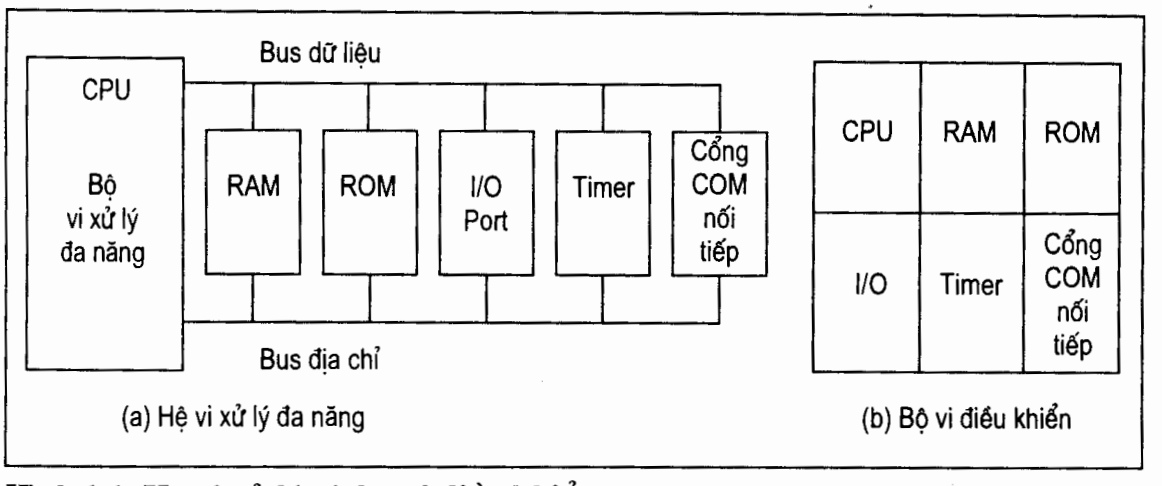
Chương 1
BỘ VI ĐIỀU KHIỂN 8051

1.1 BỘ VI ĐIỀU KHIỂN VÀ HỆ NHÚNG

Bộ vi điều khiển và bộ vi xử lý đa năng

Sự khác nhau giữa bộ vi điều khiển và bộ vi xử lý là gì? Bộ vi xử lý ở đây muốn nói đến các bộ vi xử lý đa năng như họ Intel ×86 (8086, 80286, 80386, 80486 và Pentium) hoặc họ Motorola 680×0(68000, 68010, 68020, 68030, 68040 v.v.). Các bộ vi xử lý này không có RAM, ROM và không có các cổng vào ra trên chip. Với lý do đó mà chúng được gọi chung là các bộ vi xử lý đa năng.

Một nhà thiết kế hệ thống sử dụng bộ vi xử lý đa năng, chẳng hạn như Pentium hay 68040 cần bổ sung thêm RAM, ROM, cổng vào ra và bộ định thời ngoài thì hệ thống mới hoạt động được. Dĩ nhiên, bổ sung RAM, ROM và cổng vào ra bên ngoài sẽ làm cho hệ thống cồng kềnh và đắt hơn, song có ưu điểm quan trọng là hết sức linh hoạt. Ví dụ, nhà thiết kế có thể tuỳ theo yêu cầu của từng ứng dụng mà quyết định số lượng RAM, ROM và các cổng vào ra phù hợp.



Hình 1.1. Hệ vi xử lý và bộ vi điều khiển

Đối với các bộ vi điều khiển thì vấn đề này lại khác. Bộ vi điều khiển có trên chip bộ vi xử lý (CPU), bộ nhớ RAM, ROM, cổng vào ra và bộ định thời. Nói cách khác, bộ xử lý, RAM, ROM, cổng vào ra và bộ định thời, tất cả cùng được nhúng trên chip vi điều khiển. Do vậy, người thiết kế không cần bổ sung thêm bộ nhớ ngoài, cổng vào ra hoặc bộ định thời để cho hệ thống hoạt động. Bộ vi điều

khiển với một dung lượng RAM, ROM trên chip và cổng vào - ra đã trở nên rất thích hợp trong nhiều ứng dụng yêu cầu giá thành hạ và không gian sử dụng hạn chế. Những ứng dụng, ví dụ như bộ điều khiển TV từ xa, thì không cần công suất tính toán lớn như của bộ vi xử lý 486, thậm chí không cần đến mức 8086. Một số ứng dụng lại đòi hỏi không gian nhỏ, công suất tiêu tthụ và giá thành thấp. Một số ứng dụng khác chỉ cần các thao tác vào - ra để đọc tín hiệu và tắt - mở những bit nhất định. Từ những yêu cầu đa dạng đó, một số nhà sản xuất đã đi xa hơn là tích hợp vào trong bộ vi điều khiển cả bộ chuyển đổi ADC và một số thiết bị ngoại vi khác.

Bảng 1.1. Một số sản phẩm nhúng sử dụng bộ vi điều khiển

Thiết bị gia đình	Đồ điện trong nhà Máy đàm thoại Máy điện thoại Hệ thống an toàn Bộ mở cửa nhà xe Máy trả lời Máy Fax	Máy tính gia đình Tivi Truyền hình cáp VCR Máy quay camera Điều khiển từ xa Trò chơi điện tử	Nhạc cụ điện tử Máy khâu Điều khiển ánh sáng Máy nhắn tin Máy chơi Poolball Đồ chơi Dụng cụ tập thể hình
Thiết bị văn phòng	Điện thoại Máy tính Hệ thống an toàn	Máy Fax Lò vi sóng Máy sao chụp	Máy in lazer Máy in màu Máy nhắn tin
Thiết bị tự động	Máy tính hành trình Điều khiển động cơ Túi đệm khí Thiết bị ABS	Đo lường Hệ thống bảo mật Điều khiển truyền tin Giải trí	Điều hoà nhiệt độ Điện thoại tổ ong Mở cửa không cần chìa khoá

Bộ vi điều khiển ở các hệ thống nhúng

Ở các tài liệu về bộ vi xử lý, chúng ta thường gặp khái niệm *hệ thống nhúng* (Embeded system). Thực tế, bộ vi xử lý và vi điều khiển được sử dụng rộng rãi trong nhiều sản phẩm nhúng. Sản phẩm nhúng dùng bộ vi xử lý (hoặc bộ vi điều khiển) để thực hiện một nhiệm vụ và chỉ một mà thôi. Máy in là ví dụ về một hệ nhúng, vì bộ xử lý chỉ được dùng cho công việc nhận và in dữ liệu. Đối với máy tính PC Pentium (hoặc PC tương thích IBM ×86 bất kỳ) thì khác. Máy tính PC có thể được dùng cho nhiều công việc khác nhau, như trạm dịch vụ in, máy trò chơi

điện tử, trạm dịch vụ mạng hoặc trạm đầu cuối Internet. Máy tính có thể chạy nhiều chương trình phần mềm khác nhau. Dĩ nhiên, máy tính PC thực hiện được nhiều công việc như vậy là do có bộ nhớ RAM đủ lớn và một hệ điều hành. Máy tính PC ×86 thường có hoặc được nối tới các sản phẩm nhúng khác nhau, chẳng hạn như bàn phím, máy in, modem, bộ điều khiển đĩa, thẻ âm, bộ điều khiển CD-ROM, chuột v.v. Mỗi thiết bị ngoại vi đều có một bộ vi điều khiển để thực hiện chỉ một công việc. Ví dụ, bên trong chuột có bộ vi điều khiển để thực thi công việc xác định vị trí chuột và gửi thông tin đến máy tính. Bảng 1.1 liệt kê một số sản phẩm nhúng.

Ứng dụng nhúng của PC×86

Thực tế, các bộ vi điều khiển là giải pháp thích hợp cho nhiều hệ thống nhúng. Tuy nhiên, trong nhiều trường hợp, bộ vi điều khiển không đủ khả năng đáp ứng nhiệm vụ đặt ra. Vì lý do đó mà những năm gần đây, nhiều nhà sản xuất bộ vi xử lý đa năng, chẳng hạn như Intel, Motorola, AMD (Advanced Micro Devices, Inc...) và Cyrix (hiện nay là một bộ phận của National Seniconductor, Inc) đã hướng tới bộ vi xử lý cao cấp cho thị trường nhúng. Intel, AMD và Cyrix sản xuất các bộ xử lý đa năng cho cả thị trường nhúng và thị trường máy tính PC để bàn, còn Motorola vẫn duy trì họ vi xử lý 68000 chủ yếu cho các hệ thống nhúng cao cấp và Apple hiện không còn dùng 680× cho máy tính Macintosh nữa. Trong những năm đầu thập kỷ 90 của thế kỷ XX, máy tính Apple bắt đầu sử dụng các bộ vi xử lý Power PC (như 603, 604, 620 v.v.) thay cho 680×0. Bộ vi xử lý Power PC là kết quả liên kết giữa IBM và Motorola, và nó được hướng cho thị trướng nhúng cao cấp cũng như cho cả thị trường máy tính PC. Cần lưu ý rằng, khi một bộ vi xử lý đa năng được hướng cho thị trường nhúng thì bộ vi xử lý đó cần tối ưu hoá cho các ứng dụng hệ nhúng. Vì lý do đó mà các bộ vi xử lý này thường được gọi là các bộ xử lý nhúng cao cấp. Do vậy, thuật ngữ bộ xử lý nhúng và bộ vi điều khiển thường được dùng lẫn.

Một trong những yêu cầu khắt khe của hệ nhúng là giảm thiểu công suất tiêu thụ và giảm không gian sử dụng. Bộ xử lý nhúng dựa trên ×86 và 680×0, kể cả khi bổ sung thêm cổng vào/ra, cổng COM và bộ nhớ ROM trên chip thì đều có công suất tiêu thụ thấp. Các bộ xử lý nhúng cao cấp có xu hướng tích hợp thêm chức năng trên chip CPU và cho phép người thiết kế chọn những đặc tính nào họ muốn sử dụng. Thiết kế hệ thống PC hiện cũng đang theo xu hướng này. Thông

thường, khi xây dựng bảng mạch chủ PC, ta cần có CPU, bộ chip-set chứa cổng vào/ra, bộ điều khiển cache, bộ nhớ Flash ROM có BIOS và cuối cùng là bộ nhớ cache thứ cấp. Những thiết kế mới, ví dụ, Cyrix tuyên bố rằng họ đang làm việc trên một chip chứa toàn bộ một máy tính PC ngoại trừ DRAM, hiện cũng đang khẩn trương đi vào sản xuất công nghiệp. Có lẽ trong một tương lai gần, chúng ta sẽ được nhìn thấy máy tính PC nằm gọn trên một chip.

Hiện nay, MS-DOS và Windows đang là chuẩn cho nhiều hệ máy tính và cả các hệ nhúng PC ×86 . Trong nhiều trường hợp, dùng máy tính PC ×86 cho các ứng dụng nhúng cao cấp là không tiết kiệm, song lại cho phép rút ngắn đáng kể thời gian phát triển do có sẵn thư viện phần mềm phong phú viết trên nền DOS và Windows. Đây là một vấn đề quan trọng cần lưu ý khi xây dựng hệ nhúng trong thực tế.

Lựa chọn bộ vi điều khiển

Có bốn họ vi điều khiển 8 bit chính, đó là: 6811 của Motorola, 8051 của Intel Z8 của Zilog và PIC 16× của Microchip Technology. Mỗi loại trên đều có một tập lệnh và thanh ghi riêng nên chúng không tương thích lẫn nhau. Cũng còn có các bộ vi điều khiển 16 bit và 32 bit. Vậy tiêu chuẩn để lựa chọn bộ vi điều khiển khi thiết kế là gì? Có ba tiêu chuẩn chính là: 1) Đáp ứng yêu cầu tính toán một cách hiệu quả và kinh tế. 2) Có sẵn các công cụ phát triển phần mềm, chẳng hạn như các trình biên dịch, trình hợp dịch và gỡ rối. 3) Nguồn cung cấp bộ vi điều khiển có nhiều và tin cậy.

Chúng ta sẽ nghiên cứu rõ hơn về các tiêu chuẩn này.

1. Tiêu chuẩn đầu tiên khi lựa chọn một bộ vi điều khiển, đó là phải đáp ứng yêu cầu về tính toán một cách hiệu quả và kinh tế. Do vậy, trước hết cần xem xét bộ vi điều khiển 8 bit, 16 bit hay 32 bit là thích hợp. Một số tham số kỹ thuật cần được cân nhắc khi lựa chọn là:
   a. Tốc độ: Tốc độ lớn nhất mà bộ vi điều khiển hỗ trợ là bao nhiêu.
   b. Kiểu đóng vỏ: Là kiểu 40 chân DIP, kiểu QFP hay là kiểu đóng vỏ khác (DIP - vỏ dạng 2 hàng chân. QFP là vỏ vuông dẹt)? Kiểu đóng vỏ quan trọng khi có yêu cầu về không gian, kiểu lắp ráp và tạo mẫu thử cho sản phẩm cuối cùng.
   c. Công suất tiêu thụ: Là một tiêu chuẩn cần đặc biệt lưu ý nếu sản phẩm dùng pin hoặc ắc quy.

d. Dung lượng bộ nhớ RAM và ROM trên chip.
e. Số chân vào/ra và bộ định thời trên chip.
f. Khả năng dễ dàng nâng cao hiệu suất cao hoặc giảm công suất tiêu thụ.
g. Giá thành trên một đơn vị khi mua số lượng lớn: Đây là vấn đề có ảnh hưởng đến giá thành cuối cùng của sản phẩm. Ví dụ, có bộ vi điều khiển giá 50 cent khi mua 100.000 bộ.

2. Tiêu chuẩn thứ hai khi lựa chọn bộ vi điều khiển là khả năng phát triển các sản phẩm như thế nào? Ví dụ, khả năng có sẵn các trình hợp dịch, gỡ rối, biên dịch ngôn ngữ C, mô phỏng, điều kiện hỗ trợ kỹ thuật cũng như khả năng sử dụng trong nhà và bên ngoài môi trường. Trong nhiều trường hợp, sự hỗ trợ của nhà cung cấp thứ ba cũng hết sức quan trọng.
3. Tiêu chuẩn thứ ba là khả năng sẵn sàng đáp ứng về số lượng ở hiện tại cũng như ở tương lai. Đối với một số nhà thiết kế, vấn đề này thậm chí còn quan trọng hơn cả hai tiêu chuẩn đầu tiên. Hiện nay, trong các họ vi điều khiển 8 bit hàng đầu thì 8051 có số lượng lớn nhất và có nhiều hãng cung cấp. Nhà cung cấp là nhà sản xuất bên cạnh nhà sáng chế bộ vi điều khiển. Đối với 8051 thì nhà sáng chế là Intel, nhưng hiện nay có rất nhiều hãng khác cùng sản xuất. Các hãng này gồm: Intel, Atmel, Philips/Signetics, AMD, Siemens, Matra và Dallas, Semiconductior.

Cũng nên lưu ý rằng, Motorola, Zilog và Microchip Technology đã dành một lượng dự trữ lớn để đảm bảo về mặt thời gian cho các sản phẩm. Trong những năm gần đây, các hãng trên cũng đã bắt đầu bán thư viện vi điều khiển cho ASIC.

1.2 TỔNG QUAN VỀ HỌ 8051

Tóm tắt lịch sử phát triển 8051

Năm 1981, hãng Intel cho ra mắt một bộ vi điều khiển được gọi là 8051. Bộ vi điều khiển này có 128 byte RAM, 4K byte ROM, hai bộ định thời, một cổng nối tiếp và bốn cổng 8 bit. Tất cả đều được tích hợp trên một chip. Lúc bấy giờ, bộ vi điều khiển như vậy được coi là một *"hệ thống trên chip"*. 8051 là bộ xử lý 8 bit, tức là CPU chỉ có thể làm việc với 8 bit dữ liệu. Dữ liệu lớn hơn 8 bit được chia thành các dữ liệu 8 bit để xử lý. 8051 có tất cả bốn cổng vào/ra, mỗi cổng rộng 8 bit, xem hình 1.2. 8051 có thể có một ROM trên chip cực đại là 64 K byte. Tuy nhiên, lúc đó các nhà sản xuất đã cho xuất xưởng chỉ 4K byte ROM trên chip. Chi tiết về vấn đề này sẽ được đề cập ở phần sau.

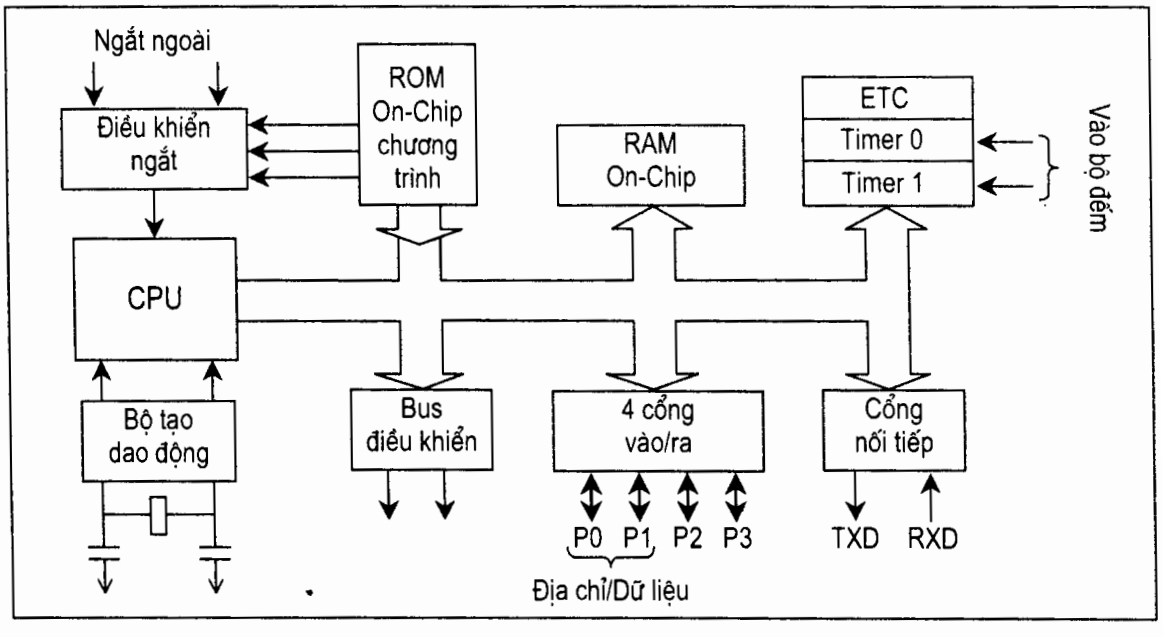
8051 đã trở nên phổ biến sau khi Intel cho phép các nhà sản xuất khác sản xuất và bán bất kỳ dạng biến thể nào của 8051 mà họ muốn với điều kiện họ phải để mã chương trình tương thích với 8051. Từ đó dẫn đến sự ra đời nhiều phiên bản của 8051 với các tốc độ khác nhau và dung lượng ROM trên chip khác nhau. Tuy nhiên, điều quan trọng là mặc dù có nhiều biến thể của 8051, như khác nhau về tốc độ và dung lượng nhớ ROM trên chip, nhưng tất cả các lệnh đều tương thích với 8051 ban đầu. Điều này có nghĩa là, nếu chương trình được viết cho một phiên bản 8051 nào đó thì cũng sẽ chạy được với mọi phiên bản khác không phụ thuộc vào hãng sản xuất.

Bảng 1.2. Thông số của 8051

Đặc tính	Số lượng
ROM	4K byte
RAM	128 byte
Bộ định thời	2
Chân vào/ra	32
Cổng nối tiếp	1
Nguồn ngắt	6

Bộ vi điều khiển 8051

Bộ vi điều khiển 8051 là thành viên đầu tiên của họ 8051. Hãng Intel ký hiệu là MCS51. Bảng 1.2 giới thiệu một số thông số kỹ thuật của 8051.



Hình 1.2. Sơ đồ khối bộ vi điều khiển 8051

Các thành viên khác của họ 8051

Có hai bộ vi điều khiển thành viên khác của họ 8051 là 8052 và 8031.

Bộ vi điều khiển 8052

8052 là một thành viên của họ 8051. 8052 có tất cả các thông số kỹ thuật của 8051, ngoài ra còn có thêm 128 byte RAM, 4K ROM và một bộ định thời nữa. Như vậy, 8052 có tổng cộng 256 byte RAM, 8K byte ROM (8051 có 4K byte ROM) và ba bộ định thời. Xem bảng 1.3.

Bảng1.3. Một số thông số chính các thành viên họ 8051

Đặc tính	8051	8052	8031
ROM trên chip (byte)	4K	8K	0K
RAM (byte)	128	256	128
Bộ định thời	2	3	2
Chân vào/ra	32	32	32
Cổng nối tiếp	1	1	1
Nguồn ngắt	6	8	6

Như thấy từ bảng 1.3, 8051 là một trường hợp riêng của 8052. Mọi chương trình viết cho 8051 đều chạy được trên 8052, nhưng điều ngược lại là không đúng.

Bộ vi điều khiển 8031

8031 là một thành viên khác của họ 8051. Chip này thường được coi là 8051 không có ROM trên chip. Để có thể dùng được chip này cần phải bổ sung ROM ngoài chứa chương trình cần thiết cho 8031. 8051 có chương trình ở ROM trên chip bị giới hạn đến 4K byte, còn ROM ngoài của 8031 thì có thể lên đến 64 kbyte. Tuy nhiên, để có thể truy cập hết bộ nhớ ROM ngoài cần dùng thêm hai cổng, do vậy chỉ còn lại hai cổng để sử dụng. Nhằm khắc phục vấn đề này, chúng ta có thể bổ sung thêm cổng vào/ra cho 8031. Chi tiết hơn Bạn có thể tham khảo chương 14.

Các phiên bản của 8051

8051 là thành viên phổ biến nhất của họ 8051, tuy nhiên chúng ta sẽ không thấy nguyên phần ký hiệu số "8051" trên chip. Sở dĩ như vậy là do 8051 có nhiều phiên bản, ví dụ với các kiểu bộ nhớ khác nhau như UV-PROM, Flash và NV-RAM và chúng đều được thể hiện ở ký hiệu linh kiện. Ví dụ, 8051 với bộ nhớ UV-PROM được ký hiệu là 8751. Phiên bản Flash ROM cũng được nhiều hãng sản xuất, chẳng hạn, của Atmel Corp có tên gọi là AT89C51. Còn phiên bản NV-RAM của Dalas Semi-Conductor thì gọi là DS5000. Ngoài ra còn có phiên bản

OTP (khả trình một lần) cũng được nhiều hãng sản xuất.

Bộ vi điều khiển 8751

Chip 8751 chỉ có 4K byte bộ nhớ UV-EPROM trên chip. Để sử dụng chip này cần có bộ đốt PROM và bộ xoá UV-EPROM. Do ROM trên chip của 8751 là UV-EPROM, nên cần phải mất 20 phút để xoá 8751 trước khi được lập trình. Vì đây là quá trình mất nhiều thời gian nên nhiều nhà sản xuất đã cho ra mắt phiên bản Flash Rom và UV-RAM. Ngoài ra còn có nhiều phiên bản với các tốc độ khác nhau.

Bộ vi điều khiển AT8951 của Atmel Corporation

AT8951 là phiên bản 8051 có ROM trên chip là bộ nhớ Flash. Phiên bản này rất thích hợp cho các ứng dụng nhanh vì bộ nhớ Flash có thể được xoá trong vài giây (chứ không phải 20 phút như 8751). Dĩ nhiên là để dùng AT8951 cần phải có một bộ đốt ROM hỗ trợ bộ nhớ Flash, song lại không cần bộ xoá ROM vì bộ nhớ Flash được xoá bằng bộ đốt PROM. Để tiện sử dụng, hiện nay Hãng Atmel đang nghiên cứu một phiên bản của AT 89C51 có thể được lập trình qua cổng COM của máy tính IBM PC và như vậy sẽ không cần bộ đốt PROM

Bảng 1.4. Các phiên bản của 8051 của Atmel (Flash ROM)

Ký hiệu*	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng vỏ
AT89C51	4K	128	32	2	6	5V	40
AT89LV51	4K	128	32	2	6	3V	40
AT89C1051	1K	64	15	1	3	3V	20
AT89C2051	2K	128	15	2	6	3V	20
AT89C52	8K	128	32	3	8	5V	40
AT89LV52	8K	128	32	3	8	3V	40

Ghi chú: * Chữ C trong ký hiệu AT89C51 là CMOS.

Thông số về kiểu đóng vỏ và tốc độ của bộ vi điều khiển cũng được thể hiện ở ký hiệu. Ví dụ, từ bảng 1.5, chữ "C" đứng trước số 51 ở ký hiệu AT 89C51 - 12PC là để chỉ công nghệ CMOS (tiêu thụ năng lượng thấp), "12" để chỉ tốc độ 12 MHZ và "P" là kiểu đóng vỏ DIP, và chữ "C" cuối cùng là ký hiệu cho thương mại (ngược với chữ "M" là quân sự). AT89C51 - 12PC rất thích hợp cho các thử nghiệm của học sinh, sinh viên.

Bảng 1.5. Các phiên bản 8051 với tốc độ khác nhau của Atmel

Ký hiệu	Tốc độ	Số chân	Đóng vỏ	Mục đích
AT89C51-12PC	42MHz	40	DTP	Thương mại

Bộ vi điều khiển DS5000 của Hãng Dallas Semiconductor

Một phiên bản phổ biến khác nữa của 8051 là DS5000 của Hãng Dallas Semiconductor. Bộ nhớ ROM trên chip của DS5000 là NV-RAM. DS5000 có khả năng nạp chương trình vào ROM trên chip trong khi nó vẫn ở trong hệ thống mà không cần phải lấy ra. Cách thực hiện là dùng qua cổng COM của máy tính IBM PC. Đây là một điểm mạnh rất được ưa chuộng. Ngoài ra, NV-RAM còn có ưu việt là cho phép thay đổi nội dung ROM theo từng byte. Nhắc lại là, bộ nhớ Flash và EPROM phải được xoá hết trước khi lập trình lại.

Bảng 1.6. Các phiên bản 8051 của Hãng Dallas Semiconductor

Mã linh kiện	ROM	RAM	Chân I/O	Timer	Ngắt	Vcc	Đóng vỏ
DS5000-8	8K	128	32	2	6	5V	40
DS5000-32	32K	128	32	2	6	5V	40
DS5000T-8*	8K	128	32	2	6	5V	40
DS5000T-8*	32K	128	32	2	6	5V	40

Ghi chú: * Chữ "T" sau ký hiệu "5000" là có đồng hồ thời gian thực.

Lưu ý đồng hồ thời gian thực RTC khác với bộ định thời Timer. RTC tạo và lưu giữ thời gian của ngày (giờ, phút, giây) và ngày tháng (ngày, tháng, năm) kể cả khi tắt nguồn.

Còn có nhiều phiên bản DS5000 với những tốc độ và kiểu đóng gói khác nhau như trình bày ở bảng 1.7. Ví dụ, DS5000-8-8 có 8K NV-RAM và tốc độ 8MHZ. Thông thường DS5000-8-12 hoặc DS5000T-8-12 là thích hợp cho các nghiên cứu, thử nghiệm của sinh viên.

Bảng 1.7. Các phiên bản của DS5000 với các tốc độ khác nhau

Mã linh kiện	NV- RAM	Tốc độ
DS5000-8-8	8K	8MHz
DS5000-8-12	8K	12MHz
DS5000-32-8	32K	8MHz
DS5000T-32-12	32K	8MHz
DS5000-32-12	32K	12MHz
DS5000-8-12	8K	12MHz (có RTC)

Phiên bản OTP của 8051

Phiên bản OTP (One Time Programmable) của 8051 là các chip 8051 có thể lập trình được một lần và được nhiều hãng sản xuất khác nhau cung cấp. Các phiên bản Flash và NV-RAM thường được dùng để phát triển sản phẩm mẫu. Khi sản phẩm mẫu được hoàn tất thì phiên bản OTP của 8051 được dùng để sản xuất hàng loạt vì giá thành trên một đơn vị sản phẩm sẽ rẻ hơn nhiều.

Họ 8051 của hãng Philips

Một nhà sản xuất quan trọng khác của họ 8051 là Philips Corporation. Quả thực, hãng này có một dải lựa chọn các bộ vi điều khiển họ 8051 rất rộng. Nhiều sản phẩm của hãng đã gộp luôn một số chức năng như bộ chuyển đổi ADC, DAC, cổng I/0 mở rộng, cả các phiên bản OTP và Flash.

Chương 2
LẬP TRÌNH HỢP NGỮ 8051

2.1 BÊN TRONG 8051

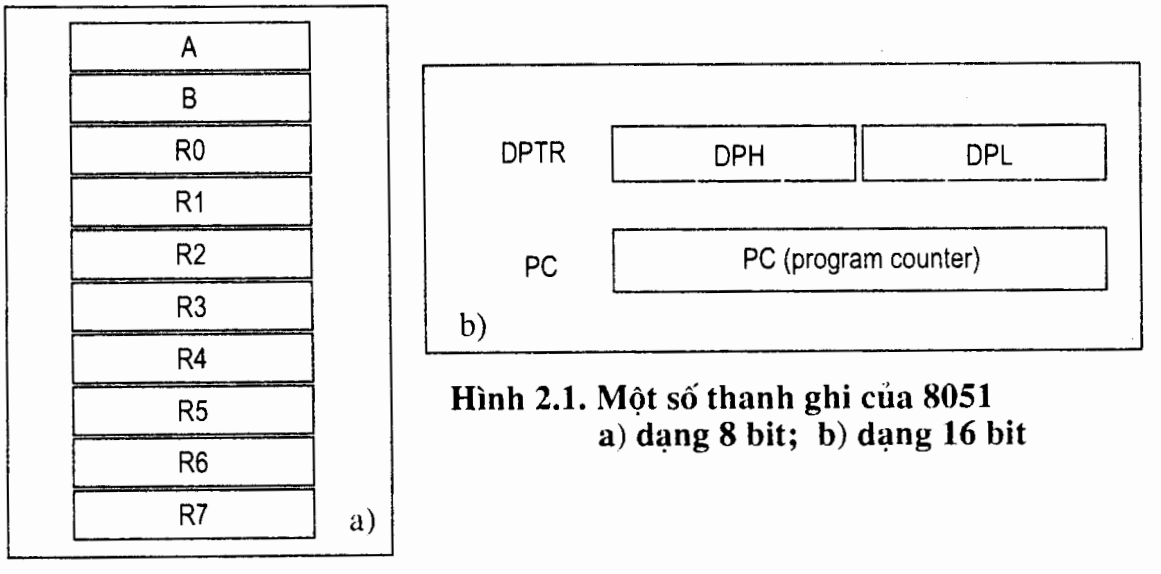
Ở phần này chúng ta nghiên cứu các thanh ghi cơ bản của 8051 và cách sử dụng chúng với một số lệnh đơn giản như MOV và ADD.

Các thanh ghi

Thanh ghi của 8051 được dùng để lưu tạm thời dữ liệu hoặc địa chỉ. Các thanh ghi này chủ yếu có kích thước 8 bit và 8051 cũng chỉ có một kiểu kiểu dữ liệu 8 bit. 8 bit của thanh ghi được sắp xếp như ở hình dưới, trong đó D7 là bit có trọng số cao nhất (MSB), còn D0 là bit có trọng số thấp nhất (LSB).

D7	D7	D5	D4	D3	D2	D1	D0

Vì chỉ có một kiểu dữ liệu 8 bit, nên mọi dữ liệu lớn hơn 8 bit trước khi xử lý đều được chia thành các phần 8 bit. 8051 có nhiều thanh ghi, nên ở đây chúng ta chỉ xem xét chủ yếu các thanh ghi đa năng. Bạn có thể tham khảo thêm phần phụ lục để có thông tin đầy đủ hơn về các thanh ghi của 8051.



Hình 2.1. Một số thanh ghi của 8051
a) dạng 8 bit; b) dạng 16 bit

Thanh ghi thường được sử dụng nhất là A (thanh ghi tích luỹ hay nói gọn hơn là thanh chứa), B, R0 ÷ R7, DPTR (con trỏ dữ liệu) và PC (bộ đếm chương trình). Trừ DPTR và PC là 16 bit, các thanh ghi còn lại đều 8 bit. Thanh ghi tích luỹ A

được sử dụng ở các phép toán số học và logic. Để minh hoạ, chúng ta sẽ tìm hiểu thông qua một số ví dụ với lệnh ADD và MOV.

Lệnh chuyển dữ liệu MOV

Lệnh MOV thực hiện sao dữ liệu từ vị trí này đến một ví trí khác. Cú pháp của lệnh như sau:

```
MOV  đích, nguồn        ;Sao dữ liệu nguồn vào đích
```

Lệnh yêu cầu CPU chuyển (trong thực tế là sao chép) toán hạng nguồn vào toán hạng đích. Ví dụ, "MOV A, R0" sao nội dung thanh ghi R0 vào thanh ghi A. Sau khi lệnh này được thực hiện thì thanh ghi A sẽ có giá trị giống như thanh ghi R0. Lệnh MOV không ảnh hưởng đến toán hạng nguồn. Ở đoạn chương trình dưới đây, đầu tiên là nạp thanh ghi A giá trị 55H, sau đó chuyển giá trị này vào các thanh ghi của CPU. Lưu ý dấu "#" là để thông báo rằng đó là một giá trị.

```
MOVA,#55H;   ;Nạp giá trị 55H vào thanh ghi A (A = 55H)
MOVR0,A      ;Sao nội dung A vào R0 (bây giờ R0=A)
MOVR1,A      ;Sao nội dung A vào R1 (bây giờ R1=R0=A)
MOVR2,A      ;Sao nội dung A vào R2 (bây giờ R2=R1=R0=A)
MOVR3,#95H   ;Nạp 95H vào thanh ghi R3 (R3=95H)
MOVA,R3      ;Sao nội dung R3 vào A (bây giờ A=95H)
```

Khi lập trình bộ vi điều khiển 8051 cần lưu ý các điểm sau:

1. Các giá trị có thể được nạp trực tiếp vào bất kỳ thanh ghi nào A, B, R0 - R7. Tuy nhiên, để thông báo đó là giá trị tức thời thì cần đặt ký hiệu "#" ở trước. Xem ví dụ sau:

```
MOV A,#23H      ;Nạp giá trị 23H vào thanh ghi A (A=23H)
MOV R0,#12H     ;Nạp giá trị 12H vào R0 (R0 = 12H)
MOV R1,#1FH     ;Nạp giá trị 1FH vào R1 (R1 = 1FH)
MOV R2,#2BH     ;Nạp giá trị 2BH vào R2 (R2 = 2BH)
MOV B,#3CH      ;Nạp giá trị 3CH vào B (B = 3CH)
MOV R7,#9DH     ;Nạp giá trị 9DH vào R7 (R7 = 9DH)
MOV R5,#0F9H    ;Nạp giá trị F9H vào R5 (R5 = F9H)
MOV R6,#12      ;Nạp trị 12d = 0CH vào R6(R6= 0CH)
```

Lưu ý ở lệnh "MOV R5, #0F9H" thì cần có số 0 đứng trước F và sau dấu # để báo rằng F là một số Hexa chứ không phải là một ký tự. Nói cách khác, lệnh "MOV R5, #F9H" sẽ gây lỗi.

2. Nếu các giá trị từ 0 đến F được chuyển vào một thanh ghi 8 bit, thì các bit còn lại được coi là 0. Ví dụ, lệnh "MOV, A #5" cho kết quả A=05, tức là A = 0000 0101 ở dạng nhị phân.

3. Chuyển một giá trị lớn hơn khả năng chứa của thanh ghi thì sẽ gây ra lỗi, ví dụ:

```
MOV A,#7F2H      ;Không hợp lệ vì 7F2H > FFH
MOV R2,456       ;Không hợp lệ vì 456 > 255 (FFH)
```

4. Để nạp một giá trị vào một thanh ghi thì cần phải gán dấu "#" trước giá trị đó. Nếu không có dấu "#" thì được hiểu là nạp giá trị lưu tại ô nhớ. Ví dụ, "MOV A, 17H" có nghĩa là nạp giá trị ở ngăn nhớ có địa chỉ 17H vào thanh ghi A và tại địa chỉ đó, dữ liệu có thể có bất kỳ giá trị nào từ 0 đến FFH. Còn để nạp giá trị 17H vào thanh ghi A thì cần phải có dấu "#" trước 17H, tức "MOV A, #17H". Và cũng cần lưu ý là nếu thiếu dấu "#" thì lệnh sẽ không gây lỗi vì hợp ngữ cho đó là một lệnh hợp lệ. Tuy nhiên, kết quả sẽ không đúng như ý muốn của người lập trình.

Lệnh cộng ADD

Lệnh cộng ADD có cú pháp như sau:

```
ADD A,nguồn      ;Cộng toán hạng nguồn vào thanh ghi A.
```

Lệnh cộng ADD yêu cầu CPU cộng toán hạng nguồn với thanh ghi A và lưu kết quả ở thanh ghi A. Ví dụ, cộng hai số 25H và 34H thì cần thực hiện theo trình tự sau:

```
MOV A,#25H       ;Nạp giá trị 25H vào A
MOV R2,#34H      ;Nạp giá trị 34H vào R2
ADD A,R2         ;Cộng R2 vào A và kết quả A=A+R2
```

Thực hiện chương trình trên ta được A = 59H (25H + 34H = 59H) và R2 = 34H, và nội dung R2 không thay đổi. Chương trình trên có thể viết theo nhiều cách khác nhau phụ thuộc vào thanh ghi được sử dụng. Dưới đây là một trong các cách viết:

```
MOV R5,#25H      ;Nạp giá trị 25H vào thanh ghi R5
MOV R7,#34H      ;Nạp giá trị 34H vào thanh ghi R7
MOV A,#0         ;Xoá thanh ghi A (A=0)
ADD A,R5         ;Cộng nội dung R5 vào A (A=A+R5)
ADD A,R7         ;Cộng R7 vào A (A=A+R7=25H+34H)
```

Chương trình này cho kết quả ở A Là 59H. Tuy nhiên, qua đoạn chương trình trên có thể đặt câu hỏi là có cần chuyển cả hai dữ liệu vào hai thanh ghi trước khi cộng chúng với nhau không? Câu trả lời là không cần. Hãy tìm hiểu đoạn chương trình sau:

```
MOV A,#25H   ;Nạp trị thứ nhất vào thanh ghi A (A=25H)
ADD A,#34H   ;Cộng giá trị thứ hai là 34H vào A (A=59H)
```

Trong ví dụ trên, 25H và 34H là các toán hạng tức thời. Ở lệnh ADD trong ví dụ này, toán hạng nguồn có thể là thanh ghi hoặc là dữ liệu tức thời, nhưng toán hạng đích thì luôn luôn là thanh ghi tích luỹ A. Như vậy, lệnh "ADD R2, #12H" là không hợp lệ, vì toán hạng đích không phải là thanh ghi A. Tương tự, lệnh "ADD R4, A" cũng không hợp lệ, vì thanh ghi A phải là toán hạng đích. Tóm lại, mọi phép toán số học của 8051 luôn dùng thanh ghi A làm toán hạng đích.

Có hai thanh ghi 16 bit của 8051 được dùng làm bộ đếm chương trình PC (Program Counter) và con trỏ dữ liệu DPTR (Data Pointer). Mục 2.3 giới thiệu cách sử dụng chúng. Thanh ghi DPTR được dùng để truy cập dữ liệu và sẽ được trình bày ở chương 5 khi đề cập tới các chế độ định địa chỉ.

2.2 GIỚI THIỆU VỀ LẬP TRÌNH HỢP NGỮ CỦA 8051

CPU chỉ có thể tính toán được trên các số nhị phân và với tốc độ rất cao. Tuy nhiên, đối với con người, nếu phải lập trình với các số nhị phân thì thật nhàm chán và chậm chạp. Chương trình chỉ gồm các số 0 và 1 là trình ngôn ngữ máy.

Thời kỳ đầu của máy tính, các lập trình viên phải viết chương trình dưới dạng ngôn ngữ máy. Mặc dù số thập lục phân (số Hexa) đã biểu diễn khá hiệu quả số nhị phân, song làm việc trên mã máy vẫn còn là công việc nặng nhọc đối với con người. Cuối cùng, hợp ngữ đã được xây dựng, trong đó có sử dụng các từ gợi nhớ và cùng với những đặc tính khác nữa, hợp ngữ đã giúp cho công việc lập trình được dễ dàng và ít lỗi hơn. Thuật ngữ *từ gợi nhớ* hay *từ gợi* (mnemonic) thường được sử dụng trong các tài liệu kỹ thuật máy tính để chỉ các mã lệnh và từ viết tắt tương đối dễ nhớ. Các chương trình hợp ngữ cần được dịch ra dạng mã máy nhờ một chương trình được gọi là trình hợp dịch. Hợp ngữ được coi là một ngôn ngữ bậc thấp vì nó có quan hệ trực tiếp với cấu trúc bên của CPU. Để lập trình hợp ngữ, lập trình viên cần nắm vững tất cả các thanh ghi của CPU, kích thước của chúng cũng như các chi tiết liên quan khác.

Ngày nay, Bạn có thể sử dụng các ngôn ngữ lập trình khác nhau, như Basic, Pascal, C, C++, Java và nhiều ngôn ngữ khác. Các ngôn ngữ này là những ngôn ngữ bậc cao vì lập trình viên không cần phải tương tác với các chi tiết bên trong CPU. Trình hợp dịch được dùng để dịch chương trình hợp ngữ ra mã máy (đôi khi cũng còn được gọi là mã đối tượng (Object Code) hay mã lệnh (Opcode)), còn các ngôn ngữ bậc cao được dịch thành mã máy bằng một chương trình gọi là trình biên dịch. Ví dụ, khi viết một chương trình bằng C, chúng ta cần sử dụng một trình biên dịch C để dịch chương trình về dạng mã máy. Tiếp theo, chúng ta sẽ tìm hiểu dạng thức hợp ngữ của 8051 và dùng trình hợp dịch để tạo ra một chương trình chạy được.

Cấu trúc của hợp ngữ

Một chương trình hợp ngữ bao gồm một chuỗi các dòng lệnh hợp ngữ. Một lệnh hợp ngữ có một từ gợi nhớ (mnemonic), và tuỳ theo từng lệnh mà sau đó có một hoặc hai toán hạng. Từ gợi nhớ là lệnh yêu cầu CPU thực hiện còn các toán hạng là các dữ liệu cần thao tác.

```
        ORG 0H          ;Bắt đầu (origin) tại ngăn nhớ 0
        MOV R5,#25H     ;Nạp 25H vào R5
        MOV R7,#34H     ;Nạp 34H vào R7
        MOV A,#0        ;Nạp 0 vào thanh ghi A
        ADD A,R5        ;Cộng nội dung R5 vào A (A=A+R5)
        ADD A,R7        ;Cộng nội dung R7 vào A (A=A+R7)
        ADD A,#12H      ;Cộng giá trị 12H vào A (A=A+12H)
HERE:   SJMP HERE       ;Ở lại trong vòng lặp
        END             ;Kết thúc tệp nguồn hợp ngữ
```

Chương trình 2.1. Ví dụ về một chương trình hợp ngữ

Chương trình 2.1 trên đây là một chuỗi các câu lệnh hay dòng lệnh, trong đó có các lệnh hợp ngữ như ADD và MOV và các chỉ dẫn. Lệnh hợp ngữ luôn yêu cầu CPU phải thực thi một nhiệm vụ, còn chỉ dẫn có nhiệm vụ hướng dẫn cho hợp ngữ chứ không yêu cầu CPU thực hiện hành động nào. Ở ví dụ trên thì ADD và MOV là lệnh, còn ORG và END là chỉ dẫn hợp ngữ. ORG hướng dẫn hợp ngữ bố trí mã lệnh bắt đầu từ ngăn nhớ 0, còn END là thông báo cho hợp ngữ biết chương trình được kết thúc. Như vậy, một chương trình hợp ngữ luôn được bắt đầu và kết thúc bằng các chỉ dẫn.

Nói chung, lệnh hợp ngữ có cấu trúc gồm 4 trường như sau:

```
[nhãn:]  [từ gợi nhớ]  [các toán hạng]  [;chú giải]
```

Dấu ngoặc vuông là tuỳ chọn, do vậy không phải dòng lệnh nào cũng có đủ 4 trường như trên. Có mấy điểm cần lưu ý như sau:

1. Trường nhãn cho phép chương trình tham chiếu đến một dòng lệnh bằng tên nhãn. Tên nhãn có độ dài tối đa được qui định. Cần kiểm tra độ dài tối đa cho phép khi viết chương trình hợp ngữ.

2. Từ gợi nhớ (lệnh) và các toán hạng là các trường kết hợp với nhau để thực hiện các nhiệm vụ của chương trình. Ở các lệnh:

```
ADD A,B
MOV A,#67
```

thì ADD và MOV là lệnh (từ gợi nhớ), còn "A, B" và "A, #67" là những toán hạng. Ở hai trường "từ gợi nhớ" và "toán hạng" có thể là các giả lệnh hoặc chỉ dẫn hợp ngữ. Nên nhớ là chỉ dẫn không tạo ra mã lệnh nào (mã máy) và chúng chỉ cần cho trình hợp dịch. Ngược lại, các lệnh thì được dịch ra mã máy (mã lệnh) để CPU thực hiện. Ở chương trình 2.1, ORG và END là các chỉ dẫn (đối với 8051 có thể được viết dưới dạng .ORG và .END). Mỗi hợp ngữ có thể có qui định cách viết riêng. Vấn đề này chúng ta sẽ xem xét kỹ hơn ở mục 2.5.

3. Trường chú giải luôn được bắt đầu bằng dấu chấm phẩy ";". Chú giải có thể bắt đầu ở đầu dòng hoặc giữa dòng. Chú giải được trình hợp dịch bỏ qua, nhưng lại cần thiết cho các lập trình viên. Chú giải là phần tuỳ chọn, tuy vậy, lập trình viên nên dùng để mô tả chương trình nhằm giúp cho người khác đọc và hiểu chương trình dễ dàng hơn.

4. Về nhãn HERE của chương trình 2.1: sau mỗi nhãn tham chiếu đến một lệnh cần có dấu hai chấm ":". Ở lệnh nhảy ngắn SJMP, thì 8051 sẽ lặp vô hạn vòng lặp này. Nếu hệ thống có chương trình giám sát thì sẽ không cần đến vòng lặp này và có thể xoá khỏi chương trình.

2.3 HỢP DỊCH VÀ CHẠY CHƯƠNG TRÌNH 8051

Câu hỏi đặt ra, là chương trình được viết, hợp dịch và chạy như thế nào? Trình tự thực hiện có thể như sau:

1. Trước hết, cần sử dụng một trình soạn thảo để viết được chương trình, ví dụ như chương trình 2.1. Có nhiều trình soạn thảo tuyệt vời, trong số đó có thể kể

như EDIT của MS-DOS hoặc Notepad của Windows. Lưu ý là, trình soạn thảo phải tạo ra tệp mã ASCII. Đối với nhiều trình hợp dịch, tên tệp cần tuân theo quy ước của DOS, còn phần mở rộng của tệp nguồn phải là "asm" hay "src" tuỳ theo trình hợp dịch được sử dụng.

2. Tệp nguồn có phần mở rộng "asm" được tạo ra ở bước 1 sẽ được chương trình dịch hợp ngữ của 8051 hợp dịch. Trình hợp dịch sẽ dịch các lệnh của tệp tin nguồn ra mã máy và tạo ra tệp đối tượng, tệp liệt kê với phần mở rộng tương ứng là "obj" và "lst".

3. Bước thứ ba gọi là *liên kết*. Trình liên kết sẽ liên kết một hoặc nhiều tệp đối tượng để tạo ra tệp đối tượng tuyệt đối có phần mở rộng "abs".

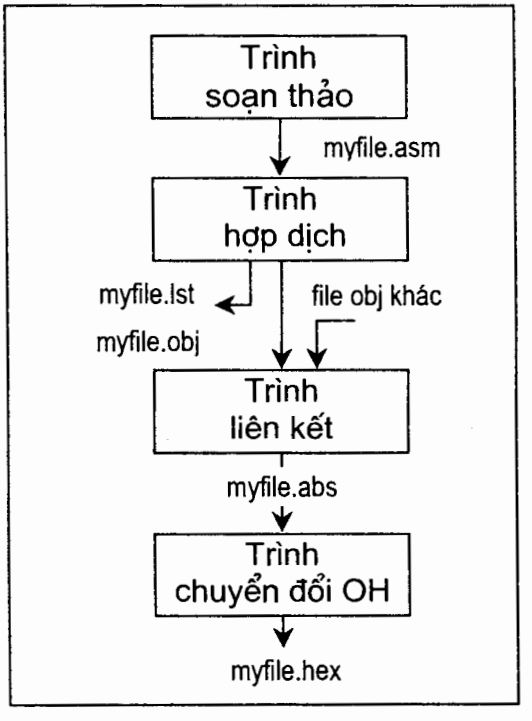
4. Tiếp theo, tệp "abs" được cấp cho chương trình chuyển tệp đối tượng về dạng số Hexa gọi là "OH" (Object to Hexa Converter) để tạo ra tệp mới có phần mở rộng là "Hexa" để có thể nạp vào ROM. Chương trình này có ở mọi bộ hợp dịch của 8051. Hiện nay, các trình hợp dịch dựa trên nền Windows đã kết hợp các bước 2 đến 4 vào một bước.

Tệp ".asm" và ".object"

Tệp ".asm" còn được gọi là tệp nguồn, và vì lý do đó mà một số trình hợp dịch dùng phần mở rộng là "src" từ chữ "source" là nguồn thay cho ".asm". Hãy kiểm tra trình hợp dịch 8051 đang sử dụng xem cần phần mở rộng nào? Như đã nói trước đây, từ tệp chương trình nguồn, trình trình hợp dịch của 8051 tạo ra cùng với tệp đối tượng .object, còn có tệp liệt kê "lst" (List file).

Tệp liệt kê ".lst"

Tệp liệt kê là tuỳ chọn. Tệp này hữu ích cho lập trình viên vì nó liệt kê tất cả các mã lệnh, địa chỉ cũng như các lỗi mà trình trình hợp dịch phát hiện được. Nhiều trình hợp dịch đặt ngầm định là không tạo



Hình 2.2. Các bước xây dựng chương trình hợp ngữ

ra tệp tin liệt kê. Lập trình viên sử dụng tệp liệt kê để xác định các lỗi cú pháp. Chỉ sau khi đã sửa hết các lỗi được đánh dấu trong tệp liệt kê, thì tệp đối tượng mới sẵn sàng làm đầu vào cho chương trình liên kết.

```
1 0000              ORG   0H        ;Bắt đầu ở địa chỉ 0
2 0000 7D25         MOV   R5,#25H   ;Nạp 25H vào R5
3 0002 7F34         MOV   R7,#34H   ;Nạp 34H vào R7
4 0004 7400         MOV   A,#0      ;Nạp 0 vào A (xoá A)
5 0006 2D           ADD   A,R5      ;Cộng R5 vào A (A=A+R5)
6 0007 2F           ADD   A,R7      ;Cộng R7 vào A (A=A+R7)
7 0008 2412         ADD   A,#12H    ;Cộng 12H vào A (A=A+12H)
8 00A  BCEF HERE:   SJMP  HERE      ;Ở lại vòng lặp
9 000C              END             ;Kết thúc tệp .asm
```

Chương trình 2.2. Tệp liệt kê

2.4 BỘ ĐẾM CHƯƠNG TRÌNH VÀ KHÔNG GIAN ROM CỦA 8051

Bộ đếm chương trình

Một thanh ghi quan trọng khác của 8051 là bộ đếm chương trình. Bộ đếm chương trình có nhiệm vụ trỏ đến địa chỉ của lệnh kế tiếp cần được thực hiện. Mỗi khi CPU nhận mã lệnh từ bộ nhớ ROM, thì bộ đếm chương trình tăng lên để trỏ đến lệnh kết tiếp. Bộ đếm chương trình của 8051 rộng 16 bit, điều đó có nghĩa là, 8051 có thể truy cập được địa chỉ chương trình từ 0000 đến FFFFH, tổng cộng là 64k byte. Tuy nhiên, không phải tất cả mọi thành viên của 8051 đều có đủ 64k byte ROM trên chip. Vấn đề tiếp theo là, khi 8051 được bật nguồn thì địa chỉ khởi đầu được bắt đầu từ đâu?

Địa chỉ bắt đầu khi 8051 được cấp nguồn

Mỗi họ vi điều khiển khi bật nguồn đều được bắt đầu từ những địa chỉ khác nhau. Đối với họ 8051 thì địa chỉ bắt đầu đều từ 0000. Bật nguồn có nghĩa là cấp điện áp V_{cc} đến chân RESET như sẽ trình bày ở chương 4. Nói cách khác, khi 8051 được cấp nguồn, thì bộ đếm chương trình có giá trị 0000. Điều này có nghĩa là nó sẽ thực hiện mã lệnh đầu tiên được lưu ở địa chỉ ROM 0000H. Tại sao lại ở vị trí 0000H của bộ nhớ ROM?. Đó là nhờ chỉ dẫn ORG ở chương trình nguồn như đã trình bày trước đây. Dưới đây là trình tự hoạt động của bộ đếm chương trình trong qúa trình nhận và thực thi một chương trình mẫu.

Mã chương trình ở ROM

Để hiểu rõ hơn về bộ đếm chương trình, chúng ta sẽ xem xét hoạt động của bộ đếm chương trình mỗi khi nhận và thực hiện lệnh. Trước hết, chúng ta khảo sát một lần nữa tệp liệt kê của chương trình mẫu lưu ở ROM. Như có thể thấy, mã lệnh và toán hạng của từng lệnh được liệt kê ở bên trái của tệp liệt kê.

```
0000                ORG   0H
0000  7D25           MOV   R5,#25H
0002  7F34           MOV   R7,#34H
0004  7400           MOV   A,#0
0006  2F             ADD   A,R5
0007  2F             ADD   A,R7
0008  2412           ADD   A,#12H
000A  80FE   HERE:   SJMP  HERE
000C                END
```

Chương trình 2.3. Tệp liệt kê

Bảng 2.1. Nội dung tệp liệt kê của chương trình 2.3

Địa chỉ ROM	Ngôn ngữ máy	Hợp ngữ
0000	7D25	MOV R5,#25H
0002	7F34	MOV R7,#34H
0004	7400	MOV A,#0
0006	2D	ADD A,R5
0007	2F	ADD A,R7
0008	2412	ADD A,#12H
000A	80EF	HERE: SJMP HERE

Sau khi chương trình được đốt vào ROM của họ 8051, như 8751, AT8951 hoặc DS5000 thì mã lệnh và toán hạng được đặt luôn bắt đầu từ địa chỉ 0000.

Nội dung ROM của chương trình 2.1 được giới thiệu ở bảng 2.2. Tại địa chỉ 0000 có mã 7D là mã lệnh chuyển một giá trị vào thanh ghi R5, còn địa chỉ 0001 chứa toán hạng (giá trị 25H) cần được chuyển vào R5. Do vậy, lệnh "MOV R5, #25H" có mã là "7D 25" trong đó 7D là mã lệnh, còn 25 là toán hạng. Tương tự, mã máy "7F 34" được ghi ở địa chỉ 0002 và 0003 biểu diễn mã lệnh và toán hạng của lệnh "MOV R7, #34H". Cũng như vậy, mã máy "74 00" tại địa chỉ 0004 và

0005 là biểu diễn lệnh "MOV A, #0". Ô nhớ 0006 có mã 2D là mã lệnh của "ADD A, R5", còn ô nhớ 0007 có nội dung 2F là mã lệnh của "ADD A, R7". Mã lệnh của lệnh "ADD A, #12H" được đặt ở ô nhớ 0008 và toán hạng 12H được đặt ở ô nhớ 0009. Ô nhớ 000A có mã lệnh của lệnh SJMP và địa chỉ đích của nó được đặt ở ô nhớ 000B. Lý do vì sao địa chỉ đích là FE sẽ được giải thích ở chương 3.

Bảng 2.2. Nội dung ROM

Địa chỉ	Mã lệnh
0000	7D
0001	25
0002	7F
0003	34
0004	74
0005	00
0006	2D
0007	2F
0008	24
0009	12
000A	80
000B	FE

Trình tự thực hiện chương trình

Giả sử chương trình trên đã được ghi vào ROM của 8051 (hoặc 8751, AT 8951 hay DS5000) thì trình tự các bước hoạt động của 8051 khi được cấp nguồn như sau:

1. Khi 8051 được bật nguồn, bộ đếm chương trình PC có nội dung 0000 và bắt đầu nạp mã lệnh đầu tiên từ vị trí nhớ 0000 của ROM chương trình. Đối với chương trình nêu trên, đó là mã 7D (chuyển một toán hạng vào R5). Khi thực hiện mã lệnh, CPU nhận giá trị 25 và chuyển vào R5. Đến đây việc thực hiện một lệnh được kết thúc. Sau đó, bộ đếm chương trình được tăng lên để trỏ đến ô nhớ 0002 (PC = 0002), tại đây có chứa mã 7F là mã của lệnh chuyển một toán hạng vào R7 "MOV R7, ...".

2. Khi thực hiện mã lệnh 7F thì giá trị 34H được chuyển vào R7 sau đó PC được tăng lên 0004.

3. Ô nhớ 0004 chứa mã lệnh của lệnh "MOV A, #0". Lệnh này được thực hiện và sau đó PC tăng lên 2 đơn vị: PC= 0006. Lưu ý là tất cả các lệnh trên đều là những lệnh 2 byte, tức là mỗi lệnh chiếm hai ô nhớ.

4. Với PC = 0006, bộ đếm chương trình trỏ đến lệnh kế tiếp là "ADD A, R5". Đây là lệnh một byte, sau khi thực hiện lệnh PC = 0007.

5. Ngăn nhớ 0007 chứa mã 2F là mã lệnh của "ADD A, R7". Đây cũng là lệnh một byte, sau khi thực hiện lệnh, PC được tăng lên 0008. Qúa trình này cứ tiếp tục cho đến khi tất cả mọi lệnh đều được nhận và thực hiện.

Do bộ đếm chương trình có ý nghĩa và cách thức làm việc như vậy, nên ở một số bộ vi xử lý, đặc biệt là dòng Intel ×86, bộ đếm chương trình còn được gọi là con trỏ lệnh IP (Instruction Pointer).

Bản đồ nhớ ROM của họ 8051

Như ta đã thấy ở chương trước, một số thành viên họ 8051 chỉ có 4k byte bộ nhớ ROM trên chip, ví dụ 8751, AT 8951, và một số khác như AT8951 có 8 k byte ROM, DS5000-32 của Dallas Semiconductor có 32k byte ROM. Dallas Semiconductor cũng có một họ 8051 với ROM trên chip là 64k byte. Điểm cần lưu ý là không có thành viên nào của họ 8051 có thể truy cập được trên 64k byte mã lệnh, vì bộ đếm chương trình của 8051 là 16 bit (dải địa chỉ từ 0000 đến FFFFH). Và điểm lưu ý thứ hai đó là, lệnh đầu tiên của ROM chương trình đều đặt ở 0000, còn lệnh cuối cùng phụ thuộc vào dung lượng ROM trên chip của mỗi thành viên họ 8051. Như vậy, 8751 và AT 8951 với 4k byte ROM thì dải địa chỉ sẽ từ 0000 đến 0FFFH. Do đó, ngăn nhớ đầu tiên có địa chỉ 0000 và ngăn nhớ cuối cùng có địa chỉ 0FFFH. Hãy xét ví dụ sau:

Ví dụ 2.1

Tìm địa chỉ bộ nhớ ROM của những thành viên họ 8051 sau đây.

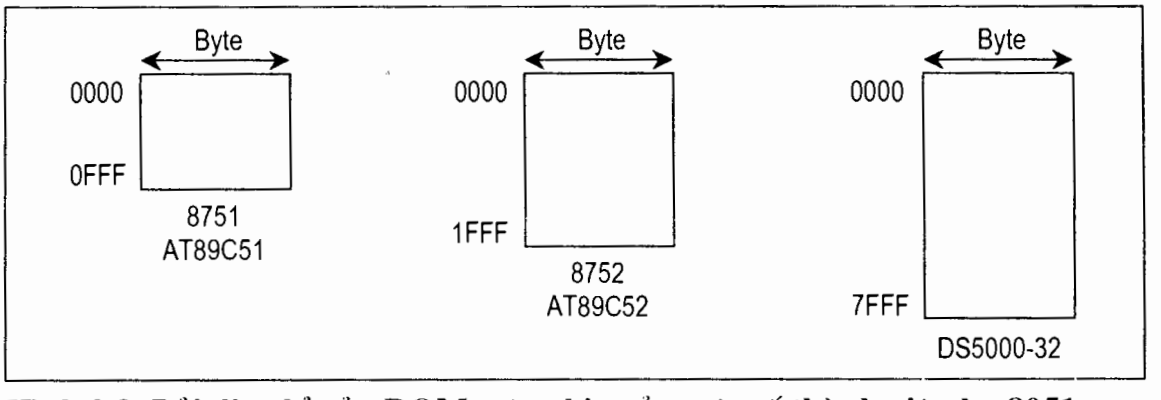
a) AT 8951 (hoặc 8751) với 4k byte.

b) DS 5000-32 với 32k byte.

Giải:

a) Với 4k byte không gian nhớ ROM trên chip, ta có 4096 byte bằng 1000H ở dạng Hexa (4×1024 = 4096D = 1000 H). Bộ nhớ này được sắp xếp trong các ngăn nhớ từ 0000 đến 0FFFFH. Lưu ý 0 luôn là ngăn nhớ đầu tiên.

b) Với 32k byte nhớ ta có 32.768 byte (32×1024). Chuyển 32.768 về số Hexa ta nhận được giá trị 8000H. Do vậy, không gian nhớ là dải từ 0000 đến 7FFFH.



Hình 2.3. Dải địa chỉ của ROM trên chip của một số thành viên họ 8051

2.5 KIỂU DỮ LIỆU VÀ CHỈ DẪN

Kiểu dữ liệu và chỉ dẫn của 8051

Bộ vi điều khiển chỉ có một kiểu dữ liệu, đó là kiểu 8 bit, và độ dài của thanh ghi cũng là 8 bit. Nhiệm vụ của lập trình viên đối với dữ liệu trên 8 bit là phân chia dữ liệu thành các phần 8 bit (từ 00 đến FFH hay từ 0 đến 255) để CPU xử lý. Ví dụ về xử lý dữ liệu lớn hơn 8 bit được trình bày ở chương 6. Dữ liệu của 8051 có thể là số âm hoặc số dương.

Chỉ dẫn định nghĩa byte DB

DB là chỉ dẫn định nghĩa dữ liệu 8 bit và được sử dụng hết sức rộng rãi khi lập trình hợp ngữ. 8 bit dữ liệu được định nghĩa có thể ở dạng thập phân, nhị phân, Hexa hay ASCII. Đối với dữ liệu thập phân thì có chữ "D" sau số thập phân, số nhị phân thì có chữ "B" và dữ liệu dạng Hexa thì có chữ "H". Và dù số được định nghĩa ở dạng nào, thì hợp ngữ luôn chuyển về dạng Hexa. Trường hợp mã sử dụng là ASCII thì đơn giản là đặt ký tự vào *'dấu nháy đơn'*. Trình hợp dịch sẽ gán mã ASCII số hoặc ký tự một cách tự động. DB là chỉ dẫn có thể sử dụng để định nghĩa chuỗi ký tự ASCII có nhiều hơn 2 ký tự. Do vậy, DB có thể được dùng để định nghĩa các dữ liệu ASCII. Dưới đây là một số ví dụ dùng định nghĩa DB:

```
         ORG    500H
DATA1:   DB     28              ;Số thập phân (=1CH)
DATA2:   DB     00110101B       ;Số nhị phân (=35H)
DATA3:   DB     39H             ;Số dạng Hexa
         ORG    510H
DATA4:   DB     "2591"          ;Các số ASCII
         ORG    518H
DATA5:   DB "My name is Binh"   ;Ký tự ASCII
```

Đối với chuỗi ASCII có thể sử dụng dấu nháy đơn hoặc nháy kép. Nếu dấu nháy đơn cũng được dùng để chỉ ký tự hay sở hữu cách của một số ngôn ngữ (như ở tiếng Anh chẳng hạn), thì định nghĩa ký tự ASCII lúc đó nên sử dụng dấu nháy kép, ví dụ, "Nhà O' Leary". Chỉ dẫn DB cũng còn được dùng để cấp phát bộ nhớ theo kích thước byte.

Một số chỉ dẫn thường gặp của trình hợp dịch

ORG (origin)

Chỉ dẫn ORG được dùng để báo địa chỉ bắt đầu của chương trình. Số sau ORG có thể ở dạng Hexa hoặc thập phân. Nếu có chữ H đằng sau số thì đó là ở

dạng Hexa, còn nếu không có thì ở dạng thập phân và trình hợp dịch sẽ chuyển thành số Hexa. Một số trình hợp dịch sử dụng dấu chấm đứng trước ".ORG" thay cho "ORG".

EQU (Equate)

Được dùng để định nghĩa một hằng số. Chỉ dẫn EQU không sử dụng ô nhớ để cất dữ liệu, mà thực hiện gán một hằng số cho nhãn sao cho khi nhãn xuất hiện trong chương trình thì giá trị hằng số sẽ được thay thế cho nhãn. Ở ví dụ sau dùng chỉ dẫn EQU để gán giá trị cho bộ đếm và sau đó nạp giá trị hằng cho thanh ghi RS.

```
COUNT  EQU 25
. . . . . .
MOV    R3,#count
```

Sau khi thực hiện lệnh "MOV R3, #COUNT" thì thanh ghi R3 sẽ có giá trị 25 (chú ý đến dấu #). Vậy ưu điểm của việc sử dụng chỉ dẫn EQU là gì? Giả sử có một hằng số (một giá trị cố định) được dùng ở nhiều chỗ khác nhau trong chương trình và lập trình viên muốn thay đổi giá trị đó. Nhờ sử dụng chỉ dẫn EQU, lập trình viên chỉ cần thay đổi giá trị một lần và trình hợp dịch sẽ tự động thay đổi giá trị tại mọi vị trí mà hằng xuất hiện.

END

Một chỉ dẫn quan trọng khác đó là END. Chỉ dẫn này báo cho trình hợp dịch biết kết thúc tệp nguồn (asm). Chỉ dẫn END là dòng cuối cùng của chương trình 8051, có nghĩa là ở mã nguồn thì mọi câu lệnh sau chỉ dẫn END đều bị trình hợp dịch bỏ qua. Một số trình hợp dịch sử dụng .END có dấu chấm đứng trước thay cho END.

Quy định tên nhãn

Với cách chọn tên nhãn có nghĩa, lập trình viên có thể làm cho chương trình trở nên sáng sủa và dễ bảo trì sau này. Có một số quy định mà các tên nhãn cần tuân theo. Thứ nhất là, tên nhãn phải là duy nhất. Các tên được sử dụng làm nhãn ở hợp ngữ gồm các chữ cái viết hoa, viết thường, các số từ 0 đến 9 và các dấu đặc biệt như: dấu hỏi (?), dấu gạch dưới (_), dấu đô la ($) và dấu chấm (.), dấu a còng (@). Ký tự đầu tiên của nhãn phải là một chữ cái và không thể là chữ số. Mỗi trình hợp dịch có một số từ dự trữ là các từ gợi nhớ cho các lệnh và không được dùng làm nhãn trong chương trình. Ví dụ như "MOV" và "ADD". Bên cạnh các

từ gợi nhớ còn có một số từ dự trữ khác. Nên kiểm tra bản liệt kê các từ dự phòng của hợp ngữ đang sử dụng.

2.6 CÁC BIT CỜ VÀ THANH GHI ĐẶC BỆT PSW CỦA 8051

Cũng như các bộ vi xử lý khác, 8051 có một thanh ghi cờ để báo các điều kiện số học. Thanh ghi cờ trong 8051 được gọi là thanh ghi từ trạng thái chương trình PSW.

Thanh ghi từ trạng thái chương trình PSW

Thanh ghi từ trạng thái chương trình PSW (Program Status Word), hay còn gọi là thanh ghi cờ là thanh ghi 8 bit. Thanh ghi PSW có 8 bit, nhưng chỉ 6 bit được 8051 sử dụng. Hai bit chưa dùng là các cờ mà người dùng có thể định nghĩa được. Bốn trong số các cờ là cờ điều kiện, nghĩa là, chúng báo một số điều kiện được thiết lập do kết quả thực hiện một lệnh. Bốn cờ này là: cờ nhớ CY (carry), cờ phụ AC (auxiliary cary), cờ chẵn lẻ P (parity) và cờ tràn OV (overflow).

Như giới thiệu ở hình 2.4, các bit PSW3 và PSW4 là RS0 và RS1 và được dùng để thay đổi các thanh ghi băng. PSW5 và PSW1 là các bit cờ trạng thái đa năng và lập trình viên có thể sử dụng cho mục đích của mình.

Dưới đây là giải thích sơ lược về 4 bit cờ của thanh ghi PSW.

Cờ nhớ CY

Cờ này được thiết lập mỗi khi có nhớ từ bit D7 và là kết quả của lệnh cộng hoặc trừ 8 bit. Có thể thiết lập trực tiếp cờ CY lên 1 hoặc xoá về 0 bằng lệnh "SETB C" và "CLR C", hay nói cách khác là "thiết lập cờ nhớ" và "xoá cờ nhớ".

Cờ nhớ phụ AC

Cờ này báo có nhớ từ bit D3 sang D4 ở phép cộng ADD hoặc trừ SUB. Cờ này được các lệnh số học mã BCD sử dụng.

Cờ bậc P

Cờ bậc (cờ chẵn lẻ) phản ánh số bit 1 trong thanh ghi A là chẵn hay lẻ. Nếu thanh ghi A chứa một số chẵn các bit 1 thì P = 0 còn chứa một số lẻ bit 1 thì P=1.

Cờ tràn OV

Cờ được thiết lập mỗi khi kết quả của phép tính số có dấu quá lớn làm cho bit cao bị tràn vào bit dấu. Nhìn chung, cờ nhớ được dùng để phát hiện lỗi trong các phép tính số học không dấu, còn cờ tràn được dùng để phát hiện lỗi trong các phép tính số học có dấu và được bàn chi tiết ở chương 6.

CY	AC	F0	RS1	RS0	OV	–	P

```
CY      PSW.7    ; Cờ nhớ
AC      PSW.6    ; Cờ phụ
--      PSW.5    ; Dành cho người dùng sử dụng mục đích chung
RS1     PSW.4    ; Bit = 1 chọn băng thanh ghi
RS0     PSW.3    ; Bit = 0 chọn băng thanh ghi
OV      PSW.2    ; Cờ tràn
--      PSW.1    ; Bit dành cho người dùng định nghĩa
P       PSW.0    ; Cờ chẵn, lẻ. Thiết lập/xoá bằng phần cứng
```

RS1	*RS0*	*Băng thanh ghi*	*Địa chỉ*
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình 2.4. Thanh ghi từ trạng thái PSW

Lệnh ADD và PSW

Bây giờ ta xem xét tác động của lệnh ADD lên các bit CY, AC và P của thanh ghi PSW. Dĩ nhiên, lệnh ADD không chỉ tác động lên ba cờ này mà cả cờ OV. Riêng cờ OV sẽ được nói đến ở chương 6 vì có liên quan đến phép tính số học của các số có dấu.

Các ví dụ từ 2.2 đến 2.4 minh hoạ tác động của lệnh ADD lên các bit cờ.

Bảng 2.3. Các lệnh ảnh hưởng tới bit cờ

Lệnh	CY	OV	AC	Lệnh	CY	OV	AC
ADD	X	X	X	SETB C	1		
ADDC	X	X	X	CLR C	0		
SUBB	X	X	X	CPL C	X		
MUL	0	X		ANL C,bit	X		
DIV	0	X		ANL C,/bit	X		
DA	X			ORL C,bit	X		
RRC	X			ORL C,/bit	X		
RLC	X			MOV C,bit	X		
SETB C	1			CJNE	X		

Ví dụ 2.2

Hãy xác định trạng thái các bit cờ CY, AC và P sau lệnh cộng 38H với 2FH:

```
MOV   A, #38H
ADD   A, #2FH          ; Sau khi cộng A = 67H, CY = 0
```

Giải:

	38	00111000
+	2F	00101111
	67	01100111

Cờ CY = 0 vì không có nhớ từ D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 1 vì thanh ghi A có 5 bit 1 (lẻ)

Ví dụ 2.3

Hãy xác định trạng thái các cờ CY, AC và P sau phép cộng 9CH với 64H.

Giải:

	9C	10011100
+	64	01100100
	100	00000000

Cờ CY = 1 vì có nhớ qua bit D7
Cờ AC = 1 vì có nhớ từ D3 sang D4
Cờ P = 0 vì thanh ghi A không có bit 1 nào (chẵn)

Ví dụ 2.4

Hãy xác định trạng thái các cờ CY, AC và P sau phép cộng 88H với 93H.

Giải:

	88	10001000
+	93	10010011
	11B	00011011

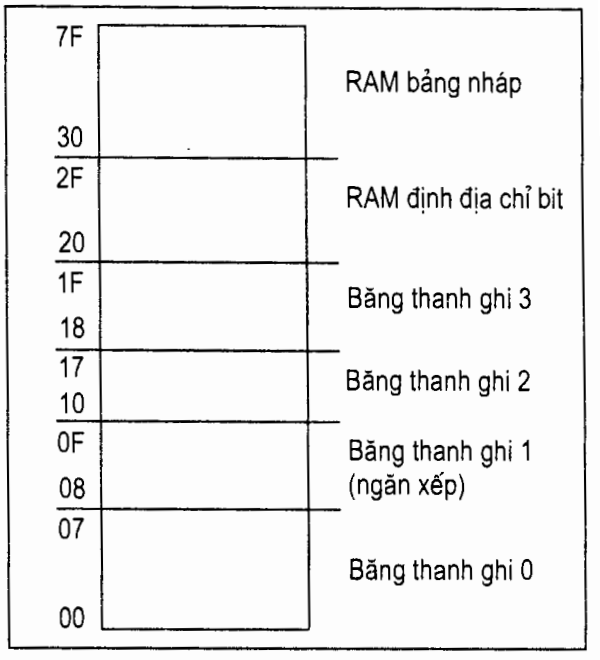
Cờ CY = 1 vì có nhớ từ bit D7
Cờ AC = 0 vì không có nhớ từ D3 sang D4
Cờ P = 0 vì số bit 1 trong A là 4 (chẵn)

2.7 BĂNG THANH GHI VÀ NGĂN XẾP CỦA 8051

8051 có tất cả 128 byte RAM. Trong mục này chúng ta xem xét phân bố của 128 byte RAM này và khảo sát công dụng của chúng khi sử dụng làm thanh ghi và ngăn xếp.

Tổ chức không gian bộ nhớ RAM của 8051

8051 có 128 byte RAM (một số thành viên, ví dụ 8052 có 256 byte RAM). 128 byte RAM bên trong 8051 được gán địa chỉ từ 00 đến 7FH. Như ta sẽ thấy ở chương 5, chúng có thể được truy cập trực tiếp như các ngăn nhớ. 128 byte RAM này được phân chia thành từng nhóm như sau:



Hình 2.5. Tổ chức RAM của 8051

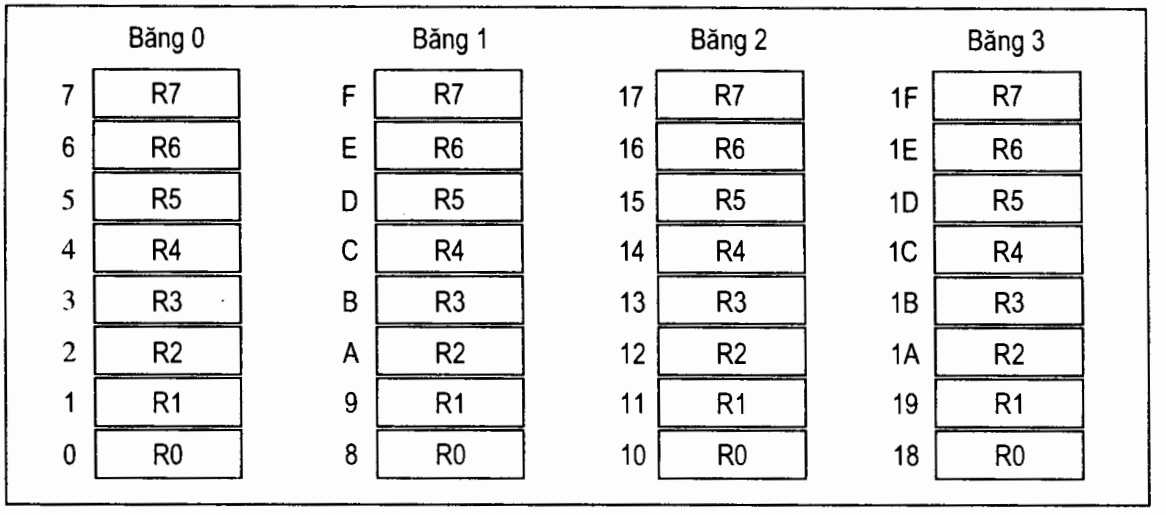
1. Từ ngăn nhớ 00 đến 1FH, tổng cộng 32 byte, được dành làm các băng thanh ghi và ngăn xếp.

2. Từ ngăn nhớ 20H đến 2FH, tổng cộng 16 byte, được dành làm bộ nhớ đọc/ghi định địa chỉ được theo bit. Chương 8 sẽ bàn chi tiết về bộ nhớ và các lệnh định địa chỉ bit.

3. Từ ngăn nhớ 30H đến 7FH, tổng cộng 80 byte, được dùng để lưu thông tin khi đọc và ghi, hay như vẫn thường gọi là bảng nháp. 80 byte RAM này thường được các lập trình viên sử dụng để lưu dữ liệu và tham số. Còn ở các chương sau, chúng ta sẽ sử dụng để lưu dữ liệu qua các cổng vào/ra vào CPU.

Băng thanh ghi của 8051

Như đã nói ở trên, tổng cộng 32 byte RAM được dành riêng cho các băng thanh ghi và ngăn xếp. 32 byte này được chia thành 4 băng thanh ghi, mỗi băng có 8 thanh ghi từ R0 đến R7. Các ngăn nhớ từ 0 đến 7 là băng 0 gồm 8 thanh ghi R0-R, trong đó R0 ở ngăn nhớ 0, R1 ở ngăn nhớ 1, R2 ở ngăn nhớ 2 v.v. Băng thanh ghi thứ hai (cũng R0 đến R7) bắt đầu từ ngăn nhớ 8 cho đến ngăn nhớ 0FH. Băng thứ ba bắt đầu từ ngăn nhớ 10H đến 17H và cuối cùng từ ngăn nhớ 18H đến 1FH là dùng cho băng thứ tư.



Hình 2.6. Các băng thanh ghi và địa chỉ RAM

Như có thể thấy ở hình 2.5, băng 1 dùng chính không gian RAM làm ngăn xếp. Đây là một vấn quan trọng trong lập trình 8051. Như vậy để bố trí ngăn xếp, chúng ta hoặc là sử dụng băng 1 hoặc phải dùng một không gian RAM khác.

Ví dụ 2.5

Xác định nội dung của các ngăn nhớ RAM sau khi thực hiện đoạn chương trình sau:

```
MOVR0, #99H          ; Nạp R0 giá trị 99H
MOVR1, #85H          ; Nạp R1 giá trị 85H
MOVR2, #3FH          ; Nạp R2 giá trị 3FH
MOVR7, #63H          ; Nạp R7giá trị 63H
MOVR5, #12H          ; Nạp R5 giá trị12H
```

Giải:

Sau khi thực hiện chương trình trên ta có:

Ngăn nhớ 0 của RAM có giá trị 99H
Ngăn nhớ 1 của RAM có giá trị 85H
Ngăn nhớ 2 của RAM có giá trị 3FH
Ngăn nhớ 7 của RAM có giá trị 63H
Ngăn nhớ 5 của RAM có giá trị 12H

Băng thanh ghi mặc định

Như đã biết, các ngăn nhớ từ 00 đến 1F được dành cho bốn băng thanh ghi, vậy băng thanh ghi nào cần được truy cập mỗi khi 8051 được cấp nguồn? Câu trả lời là băng thanh ghi 0. Đó là các ngăn nhớ RAM số 0, 1, 2, 3, 4, 5, 6 và 7 được truy cập với tên lập trình là R0, R1, R2, R3, R4, R5, R6 và R7. Dĩ nhiên sử dụng tên để truy cập ngăn nhớ thuận lợi hơn nhiều so với sử dụng trực tiếp các ngăn nhớ đó. Hãy xem ví dụ sau.

Ví dụ 2.6

Hãy viết lại chương trình ở ví dụ 2.5 sử dụng địa chỉ RAM (mà không dùng tên các thanh ghi).

Giải:

Đây được gọi là chế độ định địa chỉ trực tiếp và sử dụng địa chỉ ngăn nhớ RAM làm địa chỉ đích. Xem chi tiết ở chương 5 về chế độ định địa chỉ.

```
MOV    00,#99H      ;Nạp thanh ghi R0 giá trị 99H
MOV    01,#85H      ;Nạp thanh ghi R1 giá trị 85H
MOV    02,#3FH      ;Nạp thanh ghi R2 giá trị 3FH
MOV    07,#63H      ;Nạp thanh ghi R7giá trị 63H
MOV    05,#12H      ;Nạp thanh ghi R5 giá trị 12H
```

Chuyển băng thanh ghi

Như đã nói ở trên, băng thanh ghi 0 là mặc định khi 8051 được cấp nguồn. Chúng ta có thể chuyển sang các băng thanh ghi khác bằng cách sử dụng bit D3 và D4 của thanh ghi PSW như giới thiệu ở bảng 2.4.

Bit D3 và D4 của thanh ghi PSW còn được viết dưới dạng PSW.3 và PSW.4 và chúng có thể được truy cập bằng các lệnh định địa chỉ theo bit như SETB và CLR. Ví dụ, "SETB PSW.3" sẽ thiết lập PSW.3 và chọn băng thanh ghi 1. Chúng ta xem xét ví dụ 2.7 dưới đây.

Bảng 2.4. Bit chọn băng thanh ghi RS0 và RS1

	RS1 (PSW.4)	RS0 (PSW.3)
Băng 0	0	0
Băng 1	0	1
Băng 2	1	0
Băng 3	1	1

Ví dụ 2.7

Hãy xác định nội dung các ngăn nhớ RAM của đoạn chương trình sau:

```
SETB   PSW.4          ;Chọn băng thanh ghi 4
MOVR   0,#99H         ;Nạp thanh ghi R0 giá trị 99H
MOVR   1,#85H         ;Nạp thanh ghi R1 giá trị 85H
MOVR   2,#3FH         ;Nạp thanh ghi R2 giá trị 3FH
MOVR   7,#63H         ;Nạp thanh ghi R7giá trị 63H
MOVR   5,#12H         ;Nạp thanh ghi R5 giá trị12H
```

Giải:

Theo mặc định PSW.3 = 0 và PSW.4 = 0. Do vậy, lệnh "SETB PSW.4" sẽ bật bit RS1 = 1 và RS0 = 0, khi đó băng thanh ghi R0 đến R7 số 2 được chọn. Băng 2 sử dụng các ngăn nhớ từ 10H đến 17H. Nên sau khi thực hiện đoạn chương trình trên ta có nội dung các ngăn nhớ như sau:

Ngăn nhớ vị trí 10H có giá trị 99H
Ngăn nhớ vị trí 11H có giá trị 85H
Ngăn nhớ vị trí 12H có giá trị 3FH
Ngăn nhớ vị trí 17H có giá trị 63H
Ngăn nhớ vị trí 15H có giá trị 12H

Ngăn xếp của 8051

Ngăn xếp là một vùng bộ nhớ RAM được CPU sử dụng để lưu thông tin tạm thời. Thông tin này có thể là dữ liệu hoặc địa chỉ. CPU cần không gian lưu trữ này vì số các thanh ghi bị hạn chế.

Truy cập ngăn xếp

Nếu ngăn xếp là một vùng của bộ nhớ RAM thì phải có các thanh ghi trong CPU trỏ đến. Thanh ghi được dùng để trỏ đến ngăn xếp được gọi là thanh ghi con trỏ ngăn xếp SP (Stack Pointer). Con trỏ ngăn xếp của 8051 rộng 8 bit, tức là chỉ có thể trỏ được các địa chỉ từ 00 đến FFH.

Khi 8051 được cấp nguồn thì SP chứa giá trị 07, có nghĩa là ngăn nhớ 08 của RAM là ngăn nhớ đầu tiên được dùng làm ngăn xếp. Mỗi lần lưu thanh ghi vào ngăn xếp được gọi là cất và lệnh tương ứng là PUSH, còn ngược lại, mỗi lần nạp nội dung ngăn xếp trở lại thanh ghi được gọi lấy ra và lệnh tương ứng là POP. Để hiểu thêm về quá trình làm việc của ngăn xếp chúng ta sẽ xem xét các lệnh PUSH và POP dưới đây.

Cất thanh ghi vào ngăn xếp

Ở 8051 con trỏ ngăn xếp luôn trỏ đến ngăn nhớ sử dụng cuối cùng (gọi là đỉnh ngăn xếp). Khi cất dữ liệu vào ngăn xếp thì con trỏ ngăn xếp SP được tăng lên 1. Lưu ý rằng vấn đề này đối với các bộ vi xử lý có thể khác, ví dụ, đối với các bộ vi xử lý ×86, SP giảm xuống khi cất dữ liệu vào ngăn xếp. Xét ví dụ 2.8 dưới đây, ta thấy rằng mỗi khi lệnh PUSH được thực hiện thì nội dung của thanh ghi được cất vào ngăn xếp và SP được tăng lên 1. Dĩ nhiên điều đó là cho trường hợp dữ liệu 1 byte. Để cất giá trị thanh ghi vào ngăn xếp, lập trình viên cũng có thể sử dụng địa chỉ RAM của chúng . Ví dụ, lệnh "PUSH 1" là cất thanh ghi R1 vào ngăn xếp.

Ví dụ 2.8

Hãy xác định ngăn xếp và con trỏ ngăn xếp ở đoạn chương trình sau. Giả thiết vùng ngăn xếp là mặc định.

```
MOV   R6, #25H
MOV   R1, #12H
MOV   R4, #0F3H
PUSH  6
PUSH  1
PUSH  4
```

Giải:

	Sau PUSH 6	PUSH 1	PUSH 4
0B	0B	0B	0B
0A	0A	0A	0A F3
09	09	09 12	09 12
08	08 25	08 25	08 25
Bắt đầu SP=07	SP=08	SP=09	SP=0A

Lấy nội dung từ ngăn xếp

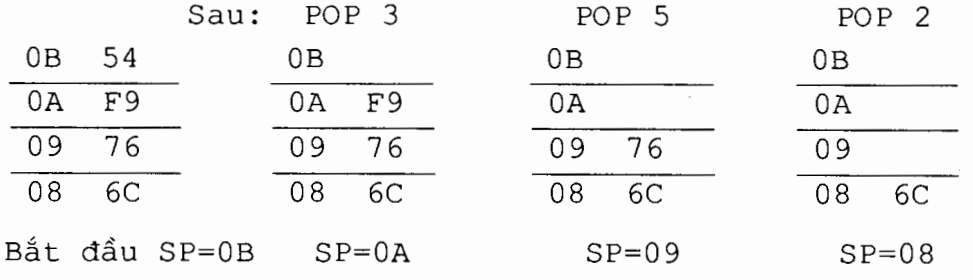
Lấy nội dung ra từ ngăn xếp trở lại thanh ghi là quá trình ngược với cất nội dung thanh ghi vào ngăn xếp. Mỗi lần lấy ra thì byte trên đỉnh ngăn xếp được sao vào thanh ghi được xác định trong lệnh và con trỏ ngăn xếp giảm xuống 1. Ví dụ 2.9 minh hoạ lệnh lấy nội dung ra khỏi ngăn xếp.

Ví dụ 2.9

Khảo sát ngăn xếp và xác định nội dung của các thanh ghi và SP sau khi thực hiện đoạn chương trình sau đây:

```
POP3      ;Lấy ngăn xếp trở lại R3
POP5      ;Lấy ngăn xếp trở lại R5
POP2      ;Lấy ngăn xếp trở lại R2
```

Giải:



Giới hạn trên của ngăn xếp

Như đã nói ở trên, các ngăn nhớ RAM từ 08 đến 1FH có thể được dùng làm ngăn xếp. Và khi đó, các ngăn nhớ RAM từ 20H đến 2FH dành cho bộ nhớ định địa chỉ bit và không thể dùng làm ngăn xếp. Nếu trong một chương trình cần có ngăn xếp lớn hơn 24 byte (08 đến 1FH = 24 byte) thì ta có thể đổi SP để trỏ đến các ngăn nhớ 30 đến 7FH. Điều này được thực hiển bởi lệnh "MOV SP, #XX".

Lệnh gọi CALL và ngăn xếp

Ngăn xếp, ngoài việc dùng để lưu cất nội dung các thanh ghi thì còn được CPU sử dụng để lưu cất tạm thời địa chỉ của lệnh đứng ngay sau lệnh CALL. Nhờ vậy mà CPU biết được địa chỉ quay về sau khi thực hiện xong chương trình con. Chi tiết về lệnh gọi CALL được trình bày ở chương 3.

Chương 3
VÒNG LẶP, LỆNH NHẢY VÀ LỆNH GỌI

3.1 VÒNG LẶP VÀ LỆNH NHẢY

Tạo vòng lặp ở 8051

Qúa trình lặp lại chuỗi lệnh với một số lần nhất định được gọi là vòng lặp. Vòng lặp thường hay được các bộ vi xử lý sử dụng. Đối với 8051, thực hiện vòng lặp là lệnh "`DJNZ thanh ghi, nhãn`", trong đó thanh ghi giảm đi một đơn vị và nếu khác 0 thì lệnh nhảy đến địa chỉ đích xác định bởi nhãn. Trước khi vòng lặp bắt đầu, số lần lặp sẽ được nạp vào thanh ghi bộ đếm. Chúng ta xem một số ví dụ sau:

Ví dụ 3.1

Viết chương trình để: a) xoá ACC và sau đó b) cộng 3 vào ACC 10 lần.

Giải:

```
      MOV    A,#0       ;Xoá ACC,  A = 0
      MOV    R2,#10     ;Nạp bộ đếm R2 = 10
BACK:ADD     A,#10      ;Cộng 03 vào ACC
      DJNZ   R2,AGAIN   ;Lặp cho đến khi R2 = 0 (10 lần)
      MOV    R5,A       ;Cất A vào thanh ghi R5
```

Ở chương trình trên, thanh ghi R2 được sử dụng làm bộ đếm. Bộ đếm lúc đầu được đặt bằng 10. Sau mỗi lần lặp, lệnh DJNZ giảm R2, nếu R2 khác 0 thì nó nhảy đến địa chỉ đích có nhãn "AGAIN". Quá trình lặp lại này tiếp tục cho đến khi R2 có giá trị bằng 0. Sau khi R2 = 0, chương trình thoát khỏi vòng lặp và thực hiện lệnh đứng ngay sau vòng lặp đó là "MOV R5, A".

Lưu ý rằng đối với lệnh DJNZ thì ở ví trí thanh ghi có thể là bất kỳ thanh ghi nào trong số R0 - R7. Bộ đếm cũng có thể là một ngăn nhớ ở RAM như sẽ thấy ở chương 5.

Ví dụ 3.2

Số lần cực đại mà vòng lặp ở ví dụ 3.1 có thể lặp lại là bao nhiêu?

Giải:

Vì thanh ghi R2 chứa số đếm và đó là thanh ghi 8 bit nên chỉ có thể chứa được giá trị cực đại là FFH hay 255. Do vậy số lần lặp lại cực đại mà vòng lặp ở ví dụ 3.1 có thể thực hiện là 256.

Vòng lặp lồng nhau

Như trình bày ở ví dụ 3.2, số đếm cực đại là 256. Vậy điều gì xảy ra nếu chúng ta muốn lặp lại nhiều hơn 256 lần? Để giải quyết vấn đề này, chúng ta có thể sử dụng một vòng lặp lồng bên trong một vòng lặp khác, gọi là vòng lặp lồng nhau. Như vậy, để tổ chức vòng lặp lồng nhau cần sử dụng 2 thanh ghi để lưu số đếm. Xem ví dụ 3.3 sau đây.

Ví dụ 3.3

Hãy viết một chương trình: a) nạp thanh ghi ACC với giá trị 55H và b) lấy bù ACC 700 lần.

Giải:

Vì 700 lớn hơn 256 (là số cực đại mà một thanh ghi có thể chứa được) nên ta phải dùng hai thanh ghi để chứa số đếm. Đoạn chương trình dưới đây trình bày cách sử dụng hai thanh ghi R2 và R3 để chứa số đếm.

```
      MOV   A,#55H    ;Nạp A = 55H
      MOV   R3,#10    ;Nạp R3 = 10 số đếm vòng lặp ngoài
NEXT: MOV   R2,#70    ;Nạp R2 = 70 số đếm vòng lặp trong
AGAIN:CPL   A         ;Bù thanh ghi A
      DJNZ  R2,AGAIN  ;Lặp lại 70 lần (vòng lặp trong)
      DJNZ  R3,NEXT
```

Trong chương trình này, thanh ghi R2 được dùng để chứa số đếm vòng lặp trong. Ở lệnh "DJNZ R2, AGAIN", mỗi khi R2 = 0 chương trình thực hiện luôn lệnh "JNZ R3, NEXT". Lệnh này buộc CPU nạp lại R2 giá trị số đếm 70 và vòng lặp trong bắt đầu lặp lại quá trình này và tiếp tục cho đến khi R3 trở về 0 và vòng lặp ngoài kết thúc.

Lệnh nhảy có điều kiện

Các lệnh nhảy có điều kiện của 8051 được tổng hợp ở bảng 3.1, còn chi tiết của từng lệnh được giới thiệu ở phụ lục A. Có thể thấy, ở bảng 3.1 một số lệnh như JZ (nhảy nếu A = 0) và JC (nhảy nếu có nhớ) chỉ thực hiện nhảy nếu có một điều kiện được thoả mãn. Tiếp theo, chúng ta sẽ xem xét cụ thể hơn một số lệnh nhảy có điều kiện thường gặp.

Lệnh JZ (nhảy nếu A = 0)

Ở lệnh này nội dung của thanh ghi A được kiểm tra. Nếu A=0 thì chương

trình nhảy đến địa chỉ đích. Ví dụ xét đoạn chương trình sau:

```
MOV    A,R0       ;Nạp giá trị của R0 vào A
JZ     OVER       ;Nhảy đến OVER nếu A = 0
MOV    A,R1       ;Nạp giá trị của R1 vào A
JZ     OVER       ;Nhảy đến OVER nếu A = 0
OVER ...
```

Ở đoạn chương trình trên, nếu R0 hoặc R1 có giá trị bằng 0 thì chương trình nhảy đến địa chỉ có nhãn OVER. Lưu ý là lệnh JZ chỉ có thể sử dụng cho đối với thanh ghi A. Chương trình chỉ tiến hành kiểm tra xem thanh ghi A có bằng 0 hay không, do đó lệnh này không áp dụng cho bất kỳ thanh ghi nào khác. Có thể thấy sử dụng lệnh này đơn giản hoá chương trình đáng kể bởi vì lập trình viên không cần phải thực hiện bất kỳ lệnh số học nào khác.

Bảng 3.1. Các lệnh nhảy có điều kiện

Lệnh	Ý nghĩa
JZ	Nhảy nếu A = 0
JNZ	Nhảy nếu A ≠ 0
DJNZ	Giảm và nhảy nếu A = 0
CJNE A, byte	Nhảy nếu A ≠ byte
CJNE re, # data	Nhảy nếu Byte ≠ data
JC	Nhảy nếu CY = 1
JNC	Nhảy nếu CY = 0
JB	Nhảy nếu bit = 1
JNB	Nhảy nếu bit = 0
JBC	Nhảy nếu bit = 1 và xoá nó

Ví dụ 3.4

Viết chương trình xác định xem R5 có chứa giá trị 0 không? Nếu không thì nạp vào R5 giá trị 55H.

Giải:

```
        MOV    A,R5       ;Sao nội dung R5 vào A
        JNZ    NEXT       ;Nhảy đến NEXT nếu A ≠ 0
        MOV    R5,#55H
NEXT:   ...
```

Lệnh JNC (nhảy nếu cờ nhớ CY = 0)

Ở lệnh này, bit cờ nhớ của thanh ghi cờ PSW được dùng làm điều kiện nhảy. Khi thực hiện lệnh "JNC nhãn", bộ xử lý kiểm tra cờ nhớ xem có được bật không (CY = 1). Nếu không được bật thì CPU sẽ thực hiện lệnh từ địa chỉ của nhãn. Nếu cờ CY = 1 thì chương trình không thực hiện lệnh nhảy mà thực hiện lệnh kế sau lệnh JNC.

Cần lưu ý là cũng có lệnh "JC nhãn". Ở lệnh JC thì nếu CY = 1 chương trình nhảy đến địa chỉ đích là nhãn.

Ngoài ra, trong nhóm lệnh nhảy có điều kiện còn có lệnh JB (nhảy nếu bit có mức cao) và JNB (nhảy nếu bit có mức thấp). Các lệnh này được trình bày ở chương 4 và 8 khi nói về thao tác bit.

Ví dụ 3.5

Tìm tổng của 79H, F5H và E2H. Ghi byte thấp của tổng vào R0 và byte cao vào R5.

Giải:

```
        MOV    A,#0        ;Xoá thanh ghi A = 0
        MOV    R5, A       ;Xoá R5
        ADD    A,#79H      ;A = 0+79H=79H
        JNC    N-1         ;Nếu không có nhớ cộng kế tiếp
        INC    R5          ;Nếu CY = 1, tăng R5
N-1:    ADD    A,#0F5H     ;A = 79H+F5H = 6EH và CY = 1
        JNC    N-2         ;Nhảy nếu CY = 0
        INC    R5          ;Nếu CY = 1 tăng R5 (R5 = 1)
N-2:    ADD    A,#0E2H     ;A = GE + E2 = 50 và CY = 1
        JNC    OVER        ;Nhảy nếu CY = 0
        INC    R5          ;Nếu CY = 1 tăng R5
OVER:   MOV    R0,A        ;Bây giờ R0 = 50H và R5 = 02
```

Cần lưu ý rằng tất cả các lệnh nhảy có điều kiện đều là các lệnh nhảy ngắn, có nghĩa là địa chỉ của đích đều phải nằm trong khoảng -127 đến +127 byte, tức chứa đủ trong bộ đếm chương trình PC.

Lệnh nhảy không điều kiện

Lệnh nhảy không điều kiện là lệnh nhảy trong đó điều khiển được truyền không có điều kiện đến địa chỉ đích. Ở 8051 có hai lệnh nhảy không điều kiện, đó là: nhảy dài LJMP - và nhảy ngắn SJMP.

Lệnh nhảy dài LJMP (Long Jump)

Nhảy dài LJMP là lệnh 3 byte, trong đó byte đầu tiên là mã lệnh, hai byte còn lại là địa chỉ 16 bit của đích. Địa chỉ đích 02 byte cho phép lệnh có thể nhảy đến bất kỳ vị trí nhớ nào trong không gian nhớ 0000 - FFFFH.

Cần lưu ý rằng, bộ đếm chương trình của 8051 là 16 bit - nghĩa là ứng với

không gian địa chỉ 64k byte, tuy nhiên, không phải mọi thành viên họ 8051 đều có bộ nhớ chương trình ROM trên chip chiếm hết cả không gian nhớ này. 8051 đầu tiên chỉ có 4k byte ROM trên chip, do vậy mỗi byte bộ nhớ đều rất quý giá. Vì lý do đó mà có cả lệnh nhảy ngắn SJMP chỉ chiếm 2 byte và có cả lệnh nhảy dài LJMP dài 3 byte. Đây cũng là cách tiết kiệm bộ nhớ ở nhiều ứng dụng mà không gian bộ nhớ hạn hẹp.

Lệnh nhảy ngắn SJMP (Short Jump)

Trong 2 byte của lệnh nhảy ngắn thì byte đầu tiên là mã lệnh và byte thứ hai là địa chỉ tương đối của địa chỉ đích. Địa chỉ tương đối với phạm vi 00 - FFH được chia thành địa chỉ nhảy tiến và địa chỉ nhảy lùi, nghĩa là, phân ra từ -128 đến +127 byte của bộ nhớ tương đối so với địa chỉ hiện thời của bộ đếm chương trình. Nếu là lệnh nhảy tiến thì địa chỉ đích có thể nằm trong khoảng 127 byte từ giá trị hiện thời của bộ đếm chương trình. Nếu địa chỉ đích ở phía sau (nhảy lùi) thì có thể nằm trong khoảng -128 byte từ giá trị hiện hành của PC.

Xác định địa chỉ nhảy ngắn

SJMP là lệnh nhảy ngắn. Ngoài ra tất cả các lệnh nhảy có điều kiện như JNC, JZ và DJNZ cũng là những lệnh nhảy ngắn và có kích thước 2 byte. Ở những lệnh này, byte thứ nhất là mã lệnh, còn byte thứ hai là địa chỉ đích tương đối. Địa chỉ đích tương đối có nghĩa là so với giá trị của bộ đếm chương trình. Để xác định địa chỉ đích, byte thứ hai được cộng vào thanh ghi PC của lệnh đứng ngay sau lệnh nhảy. Để hiểu rõ hơn, chúng ta xem ví dụ sau:

Ví dụ 3.6

Sử dụng tệp tin liệt kê dưới đây hãy kiểm tra việc xác định địa chỉ nhảy về trước.

```
Dòng   PC     Mã lệnh          Lệnh   Toán hạng
01     0000                    ORG    0000
02     0000   7800             MOV    R0, #0
03     0002   7455             MOV    A, #55H
04     0004   6003             JZ     NEXT
05     0006   08               INC    R0
06     0007   04       AGAIN:  INC    A
07     0008   04               INC    A
08     0009   2477     NEXT:   ADD    A, #77h
```

```
09      000B    5005            JNC     OVER
10      000D    E4              CLR     A
11      000E    F8              MOV     R0, A
12      000F    F9              MOV     R1, A
13      0010    FA              MOV     R2, A
14      0011    FB              MOV     R3, A
15      0012    2B      OVER:   ADD     A, R3
16      0013    50F2            JNC     AGAIN
17      0015    80FE    HERE:   SJMP    HERE
18      0017                    END
```

Giải:

Trước hết nên lưu ý rằng các lệnh JZ và JNC đều là lệnh nhảy về trước. Địa chỉ đích của lệnh nhảy về trước được xác định bằng cách cộng giá trị PC của lệnh đi ngay sau lệnh nhảy vào byte thứ hai của lệnh nhảy ngắn. Ở dòng 4 lệnh "JZ NEXT" có mã lệnh 60 và toán hạng 03 tại địa chỉ 0004 và 0005. Ở đây 03 là địa chỉ tương đối - tương đối nghĩa là so với địa chỉ của lệnh kế tiếp là: "INC R0" và đó là 0006. Cộng 0006 vào 3 sẽ được địa chỉ đích của nhãn NEXT là 0009. Bằng cách tương tự, ở dòng 9 lệnh "JNC OVER" có mã lệnh và toán hạng là 50 và 05, trong đó 50 là mã lệnh còn 05 là địa chỉ tương đối. Do vậy, 05 được cộng với 000D là địa chỉ của lệnh "CLR A" đứng ngay sau lệnh "JNC OVER" và cho giá trị 12H chính là địa chỉ của nhãn OVER.

Ví dụ 3.7

Hãy xác định địa chỉ của các nhảy lùi ở ví dụ 3.6.

Giải:

Ở danh sách liệt kê chương trình, lệnh "JNC AGAIN" có mã lệnh là 50 và địa chỉ tương đối là F2H. Lấy địa chỉ tương đối F2H cộng với địa chỉ lệnh đứng sau lệnh nhảy là 15H, chúng ta có 15H + F2H = 07 (phần nhớ được bỏ đi). Để ý rằng 07 là địa chỉ nhãn AGAIN. Tương tự, lệnh "SJMP HERE" có mã lệnh 80 và địa chỉ tương đối FEH. Lấy địa chỉ tương đối FEH cộng với giá trị PC của lệnh kế tiếp là 0017H cho địa chỉ 0015H - chính là địa chỉ nhãn HERE (17H + FEH = 15H phần nhớ được bỏ đi). Lưu ý rằng FEH là -2 và 17h + (-2) = 15H. Về phép cộng số âm sẽ được bàn ở chương 6.

Xác định địa chỉ đích lệnh nhảy lùi

Ở trường hợp nhảy tiến, giá trị độ dời là một số dương trong khoảng từ 0 đến 127 (00 đến 7F ở dạng Hexa), còn ở trường hợp nhảy lùi, giá trị độ dời là một số âm nằm trong khoảng từ 0 đến -128 như sẽ giải thích ở ví dụ 3.7.

Cần lưu ý rằng, dù nhảy tới hay nhảy lùi thì đối với mọi lệnh nhảy ngắn bất kỳ, địa chỉ đích không bao giờ có thể ngoài vùng -128 đến +127 byte so với địa chỉ của lệnh đứng ngay sau lệnh SJMP. Nếu thực hiện lệnh nhảy ngoài phạm vi nêu trên thì trình hợp dịch cho ra thông báo lỗi.

3.2 LỆNH GỌI CALL

Một lệnh chuyển điều khiển khác là lệnh CALL dùng để gọi chương trình con. Chương trình con thường được sử dụng để thực thi các công việc được lặp lại thường xuyên. Nhờ cách tổ chức chương trình con mà chương trình vừa tiết kiệm không gian nhớ vừa trở nên có cấu trúc. Ở 8051 có hai lệnh gọi là: gọi dài LCALL và gọi tuyệt đối ACALL. Yếu tố quyết định sử dụng lệnh gọi nào phụ thuộc vào địa chỉ đích.

Lệnh gọi dài LCALL (Long Call)

Đây là lệnh 3 byte. Byte đầu tiên là mã lệnh, còn hai byte sau là địa chỉ của chương trình con đích. Như vậy LCALL có thể được dùng để gọi chương trình con ở bất kỳ vị trí nào trong phạm vi 64k byte không gian địa chỉ của 8051. Sau khi thực hiện xong một chương trình con, để 8051 biết được chỗ quay trở về thì địa chỉ của lệnh đứng ngay sau lệnh gọi LCALL sẽ được tự động cất vào ngăn xếp. Khi một chương trình con được gọi, điều khiển được chuyển đến chương trình con và bộ xử lý cất bộ đếm chương trình PC vào ngăn xếp và bắt đầu nạp lệnh và thực hiện lệnh của chương trình con. Sau khi thực hiện xong chương trình con, lệnh RET sẽ chuyển điều khiển về chương trình vừa gọi nó. Chương trình con luôn có lệnh cuối cùng là lệnh RET, xem ví dụ 3.8.

Ví dụ 3.8

Hãy viết chương trình liên tục đổi tất cả các bit của cổng P1 bằng cách luân phiên gửi đến cổng các giá trị 55H và AAH. Thiết lập một độ trễ thời gian giữa các lần xuất dữ liệu của cổng P1. Chương trình này sẽ được sử dụng để kiểm tra các cổng của 8051 ở chương tiếp theo.

Giải:

```
       ORG    0000
 BACK: MOVA,  #55H       ;Nạp A với giá trị 55H
       MOV    P1,A       ;Gửi 55H đến cổng P1
       LCALL  DELAY      ;Tạo trễ thời gian
       MOV    A,#0AAH    ;Nạp A với giá trị  AAH
       MOV    P1,A       ;Gửi AAH đến cổng P1
       LCALL  DELAY      ;Giữ chậm
       SJMP   BACK       ;Lặp lại vô tận
 ;--Đây là chương trình con tạo độ trễ thời gian
       ORG    300H       ;Đặt trình tạo trễ ở địa chỉ 300H
DELAY: MOV    R5,#00H    ;Nạp bộ đếm R5 = 255H (hay FFH)
AGAIN: DJNZ   R5,AGAIN   ;Tiếp tục cho đến khi R5 về 0
       RET               ;Trả điều khiển về nguồn gọi
       END               ;Kết thúc tệp tin
```

Từ ví dụ 3.8 cần lưu ý các điểm sau đây:

1. Khi thực hiện lệnh "LCALL DELAY" đầu tiên thì địa chỉ của lệnh ngay sau nó là "MOV A, #0AAH" được đẩy vào ngăn xếp và 8051 bắt đầu thực hiện các lệnh ở địa chỉ 300H.

2. Ở chương trình con DELAY, lúc đầu bộ đếm R5 được nạp giá trị 255 (R5 = FFH). Do vậy, vòng lặp được lặp lại 256 lần. Khi R5 trở về 0, lệnh thực hiện tiếp theo là RET, khi đó địa chỉ từ ngăn xếp được tải vào vào bộ đếm chương trình và lệnh sau lệnh CALL sẽ tiếp tục được thực hiện.

Khoảng thời gian trễ trong ví dụ 8.3 phụ thuộc vào tần số của 8051. Cách tính chính xác thời gian sẽ được giải thích ở chương 4. Tuy nhiên, ta có thể tăng thời gian trễ bằng cách sử dụng vòng lặp lồng nhau như ở ví dụ sau.

```
DELAY:                    ;Vòng lặp tạo trễ
        MOV    R4,#255    ;Nạp R4=255 (FFH dạng hexa)
NEXT:   MOV    R5,#255    ;Nạp R5=255 (FFH dạng hexa)
AGAIN:  DJNZ   R5,AGAIN   ;Lặp lại cho đến khi RT=0
        DJNZ   R4,NEXT    ;Giảm R4
                          ;Nạp tiếp R5 cho đến khi R4=0
           RET            ;Trở về (khi R4=0)
```

Lệnh CALL và vai trò của ngăn xếp

Ngăn xếp và con trỏ ngăn xếp sẽ được nghiên cứu ở chương cuối. Để hiểu được vai trò của ngăn xếp trong các bộ vi điều khiển, chúng ta sẽ khảo sát nội dung của ngăn xếp và con trỏ ngăn xếp qua ví dụ 3.9 dưới đây.

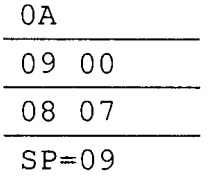
Ví dụ 3.9

Phân tích nội dung của ngăn xếp sau khi thực hiện lệnh LCALL đầu tiên ở ví dụ sau:

```
001 0000                ORG    0
002 0000 7455   BACK:   MOV    A,#55H;Nạp 55H vào A
003 0002 F590           MOV    P1,A   ;Gửi 55H tới cổng P1
004 0004 120300         LCALL  DELAY  ;Tạo trễ
005 0007 74AA           MOVA,  #0AAH  ;Nạp AAH vào A
006 0009 F590           MOV    P1,A   ;Gửi AAH tới cổng  P1
007 000B 120300         LCALL  DELAY  ;Tạo trễ
008 000E 80F0           SJMP   BACK   ;Tiếp tục
009 0010
010 0010              ;-- Đây là trình tạo trễ
011 0300                ORG    300H
012 0300         DELAY:
013 0300 7DFF           MOV    R5,#FFH   ;Nạp 255 vào R5
014 0302 DDFE   AGAIN:DJNZ    R5,AGAIN  ;Dừng ở đây
015 0304 22             RET              ;Trở về nguồn gọi
016 0305                END              ;Kết thúc
```

Giải:

Khi lệnh LCALL đầu tiên được thực hiện thì địa chỉ của lệnh "MOV A, #0AAH" đã được cất vào ngăn xếp. Byte thấp cất trước và byte cao cất sau. Lệnh cuối cùng của chương trình con là lệnh trở về RET, khi đó CPU lấy lại (POP) các byte trên đỉnh của ngăn xếp vào bộ đếm chương trình PC và tiếp tục thực hiện lệnh tại địa chỉ 07. Lược đồ bên cạnh chỉ ra trình tự nội dung của ngăn xếp sau lần gọi LCALL đầu tiên.



Sử dụng lệnh PUSH và POP ở chương trình con

Khi gọi một chương trình con thì ngăn xếp phải lưu được vị trí trở về của

CPU sau khi kết thúc chương trình con. Vì vậy, cần hết sức cẩn thận mỗi khi có thao tác với ngăn xếp. Có một nguyên tắc khi gọi bất kỳ chương trình con nào là số lần cất vào (PUSH) luôn bằng số lần lấy ra (POP). Nói cách khác, mỗi khi có một lệnh PUSH thì cũng cần phải có một lệnh POP, xem ví dụ 3.10.

Ví dụ 3.10

Phân tích nội dung ngăn xếp khi thực hiện lệnh LCALL đầu tiên ở đoạn chương trình sau.

```
01 0000                 ORG    0
02 0000 7455    BACK:   MOV    A,#55H    ;Nạp 55H vào A
03 0002 F590            MOV    P1,A      ;Gửi 55H ra cổng P1
04 0004 7C99            MOV    R4,#99H
05 0006 7D67            MOV    R5,#67H
06 0008 120300          LCALL  DELAY     ;Tạo trễ
07 000B 74AA            MOV    A,#0AAH   ;Nạp ÂH vào A
08 000D F590            MOV    P1,A      ;Gửi AAH ra cổng P1
09 000F 120300          LCALL  DELAY
10 0012 80EC            SJMP   BACK      ;Tiếp tục thực hiện
11 0014                 ;-- Đây là trình con tạo trễ DELAY
12 0300                 ORG    300H
13 0300 C004    DELAY:  PUSH   4         ;Đẩy R4 vào ngăn xếp
14 0302 C005            PUSH   5         ;Đẩy R5 vào ngăn xếp
15 0304 7CFF            MOV    R4,00FH   ;Gán R4 = FFH
16 0306 7DFF    NEXT:   MOV    R5,#00FH  ;Gán R5 = 255
17 0308 DDFE    AGAIN:  DJNZ   R5,AGAIN
18 030A DCFA            DJNZ   R4,NEXT
19 030C D005            POP    5  ;Kéo đỉnh ngăn xếp vào R5
20 030E D004            POP    4  ;Kéo đỉnh ngăn xếp vào R4
21 0310 22              RET       ;Trở về nguồn gọi
22 0311                 END       ;Kết thúc tệp tin hợp ngữ
```

Giải:

Trước hết nên lưu ý là đối với lệnh PUSH và POP chúng ta cần xác định địa chỉ trực tiếp của thanh ghi được cất vào và lấy ra từ ngăn xếp. Dưới đây là biểu diễn nội dung của ngăn xếp.

Sau LCALL đầu tiên		Sau PUSH 4		Sau PUSH 5	
0B		0B		0B 67	R5
0A		0A 99	R4	0A 99	R4
09 00	PCH	09 00	PCH	09 00	PCH
08 0B	PCL	08 0B	PCL	08 0B	PCL

Gọi chương trình con

Cách tổ chức của trình hợp ngữ thường có một chương trình chính và nhiều chương trình con được gọi từ chương trình chính đó. Điều này cho phép ta xây dựng chương trình con dưới dạng module riêng biệt. Mỗi module có thể được kiểm tra độc lập và sau đó được ghép cùng với chương trình chính. Cách tổ chức như vậy cũng rất có ý nghĩa khi xây dựng chương trình lớn vì các module có thể được giao cho nhiều lập trình viên khác nhau nhằm rút ngắn thời gian phát triển.

Cần nhắc lại rằng, trước khi sử dụng lệnh LCALL, địa chỉ đích của chương trình con có thể ở một vị trí nào đó trong không gian 64k byte của bộ nhớ 8051. Tuy nhiên, đối với các lệnh gọi khác thì có thể khác, ví dụ, trong trường hợp lệnh ACALL như giới thiệu ở dưới đây.

```
;Chương trình chính MAIN gọi chương trình con
      ORG   0
MAIN: LCALL SUBR-1
      LCALL SUBR-2
      LCALL SUBR-3

HERE: SJMP  MAIN
;--Kết thúc chương trình chính MAIN
;
SUBR_1:  ...
      ...
      RET
; -- Kết thúc chương trình con 1
; SUBR_2 ...
      ...
      RET
; -- Kết thúc chương trình con 2
; SUBR_3 ...
      ...
      RET
;--Kết thúc chương trình con 3
      END          ;Kết thúc trình hợp ngữ
```

Hình 3.1. Chương trình chính hợp ngữ của 8051 gọi các trình con

Lệnh gọi tuyệt đối ACALL (Absolute call)

Lệnh ACALL là lệnh 2 byte, khác với lệnh LCALL dài 3 byte. Do ACALL chỉ có 2 byte nên địa chỉ đích của chương trình con phải nằm trong khoảng 2 Kbyte địa chỉ vì chỉ có 11 bit của 2 byte được dùng để xác định địa chỉ. Nếu xét theo khía cạnh cất bộ đếm chương trình vào ngăn xếp hay sử dụng lệnh RET kết thúc chương trình con thì giữa ACALL và LCALL không có sự khác biệt nào. Chỉ có một điểm khác biệt đó là địa chỉ đích của lệnh LCALL có thể nằm bất cứ đâu trong phạm vi 64k byte không gian địa chỉ của 8051, trong khi đó địa chỉ của lệnh ACALL phải nằm trong khoảng 2 Kbyte. Thực tế, một số biến thể 8051 chỉ có 1 Kbyte ROM trên chip. Trong những trường hợp đó, sử dụng lệnh ACALL có thể tiết kiệm được một số byte bộ nhớ của không gian ROM chương trình so với lệnh LCALL.

Ví dụ 3.11

Một kỹ sư thiết kế phát triển ứng dụng của mình dựa trên chip vi điều khiển Atmel AT89C1051 với 1 Kbyte ROM Flash trên chip. Vậy trong trường hợp này nên sử dụng lệnh LCALL hay lệnh ACALL.

Giải:

Trong trường hợp này nếu sử dụng lệnh ACALL thì mỗi lần gọi lệnh sẽ tiết kiệm bộ nhớ hơn vì đó là lệnh 2 byte.

Dĩ nhiên, để viết chương trình có hiệu quả cần hiểu rõ từng lệnh được sử dụng. Sau đây là một ví dụ.

Ví dụ 3.12

Viết lại chương trình ở ví dụ 3.8 sử dụng lệnh lấy bù.

Giải:

```
        ORG    0
        MOV    A,#55H     ;Nạp giá trị 55H vào A
BACK:   MOV    P1,A       ;Xuất A ra cổng P1
        ACALL  DELAY      ;Tạo trễ
        CPL    A          ;Lấy bù thanh ghi A
        SJMP   BACK       ;Tiếp tục thực hiện vô hạn
```

```
; -- Đây là trình con tạo trễ DELAY
DELAY: MOV   R5,#0FFH  ;Nạp R5=255 (FFH) cho bộ đếm
AGAIN: DJNZ  R5,AGAIN  ;Dừng cho đến khi R5=0
       RET             ;Trở về
       END             ;Kết thúc
```

Lưu ý rằng, ở chương trình này thanh ghi A được nạp giá trị 55H. Lấy bù 55H được AAH, và lấy bù AAH lại cho 55H. Sở dĩ như vậy vì dưới dạng mã nhị phân "01010101" (tức 55H) và "10101010" tức AAH là bù của nhau.

3.3 TẠO VÀ TÍNH TOÁN THỜI GIAN GIỮ CHẬM

Chu kỳ máy

Để thực hiện một lệnh thì cần một số chu kỳ đồng hồ. Ở họ 8051, các chu kỳ đồng hồ này được gọi là các chu kỳ máy. Phụ lục A-2 cung cấp danh sách lệnh 8051 và các chu kỳ máy của chúng. Độ dài của chu kỳ máy của họ 8051 phụ thuộc vào tần số của bộ dao động thạch anh nối với hệ thống 8051. Bộ dao động thạch anh cùng với mạch điện trên chip cung cấp xung đồng hồ cho 8051. Tần số tinh thể thạch anh thường trong khoảng 4MHz đến 30 MHz phụ thuộc vào tốc độ chip và nhà sản xuất. Thường được sử dụng nhất là bộ dao động thạch anh tần số 10.0592MHz nhằm tương thích với cổng nối tiếp của IBM PC, xem chương 10. Đối với 8051, một chu kỳ máy gồm 12 chu kỳ dao động thạch anh. Do vậy, để tính chu kỳ máy đơn giản là xác định 12 chu kỳ dao động tinh thể thạch anh. Xem ví dụ 3.13.

Ví dụ 3.13

Hãy tìm chu kỳ máy trong 3 trường hợp tần số dao động thạch anh sau đây:
a) 11.0592MHz; b) 16MHz ; c) 20MHz.

Giải:

a) 11.0592/12 = 921.6kHz; Chu kỳ máy là 1/921.6kHz = 1.085μs .

b) 16MHz/12 = 1.333MHz; Chu kỳ máy MC = 1/1.333MHz = 0.75μs.

c) 20MHz/12 = 1.66MHz ⇒ MC = 1/1.66MHz = 0.60μs.

Ví dụ 3.14

Hệ thống 8051 có tần số dao động thạch anh là 11.0592MHz. Hãy tìm thời gian cần thiết để thực hiện các lệnh sau đây.

```
a) MOV R3,#55;      b) DEC R3;      c) DJNZ R2,đích
d) LJMP;            e) SJMP;        f) NOP
g) MUL AB.
```

Giải:

Chu kỳ máy của hệ 8051 có tần số đồng hồ 11.0592MHz là 1.085μs như đã tính ở ví dụ 3.13. Bảng A-1 ở phụ lục A giới thiệu số chu kỳ máy đối với các lệnh trên. Vậy ta có:

	Lệnh		*Số chu kỳ máy*	*Thời gian thực hiện*
a)	MOV	R3,#55	1	1×1.085 μs = 1.085 μs
b)	DEC	R3	1	1×1.085 μs = 1.085 μs
c)	DJNZ	R2,target	2	2×1.085 μs = 2.17 μs
d)	LJMP		2	2×1.085 μs = 2.17 μs
e)	SJMP		2	2×1.085 μs = 2.17 μs
f)	NOP		1	1×1.085 μs = 1.085 μs
g)	MUL	AB	4	4×1.085 μs = 4.34 μs

Tính thời gian trễ

Như đã trình bày ở trên, một chương trình con thực hiện giữ chậm gồm có hai phần: (1) thiết lập bộ đếm và (2) một vòng lặp. Hầu hết thời gian giữ chậm được thực hiện bởi thân vòng lặp như trình bày ở ví dụ 3.15.

Ví dụ 3.15

Hãy xác định thời gian giữ chậm ở chương trình sau, nếu tần số dao động thạch anh là 11.0592MHz.

```
        MOV   A,#55H
AGAIN:  MOV   P1,A
        ACALL DELAY
        CPLA
        SJMP  AGAIN; --Thời gian giữ chậm
DELAY:  MOV   R3,#200
HERE:   DJNZ  R3,HERE
```

```
      RET
```

Giải:

Từ bảng A-1 của phụ lục A, chúng ta có được số chu kỳ máy ứng với các lệnh của chương trình con giữ chậm như sau:

```
                          Chu kỳ máy
DELAY: MOV    R3,#200        1
HERE:  DJNZ   R3,HERE        2
       RET                   1
```

Như vậy, tổng thời gian giữ chậm là [(200 × 2) + 1 + 1] × 1.085 = 436.17μs.

Thông thường, thời gian giữ chậm được xác định dựa vào thời gian thực hiện các lệnh bên trong vòng lặp và bỏ qua các lệnh ở bên ngoài vòng lặp.

Ở ví dụ 3.15 giá trị lớn nhất mà R3 có thể lưu được là 255, do vậy một cách đơn giản để tạo thời gian trễ là sử dụng lệnh NOP (không thực hiện gì) đặt ở trong vòng lặp. Chúng ta xem ví dụ 3.16 dưới đây.

Ví dụ 3.16

Hãy xác định thời gian trễ của chương trình con sau. Giả thiết tần số dao động thạch anh là 11.0592MHz.

```
                          Số chu kỳ máy
DELAY:    MOV    R3,#250       1
HERE:     NOP                  1
          NOP                  1
          NOP                  1
          NOP                  1
          DJNZ   R3,HERE       2
          RET                  1
```

Giải:

Thời gian trễ bên trong vòng lặp HERE là [250 (1 + 1 + 1 + 1 + 1 + 2)] × 1.0851μs = 1627.5μs. Cộng thêm hai lệnh ngoài vòng lặp ta có 1627.5μs × 1.085μs = 1629.67μs.

Tạo trễ dùng vòng lặp lồng nhau

Một cách khác để tạo được độ trễ lớn là sử dụng vòng lặp lồng nhau, tức là dùng vòng lặp này lồng bên trong vòng lặp khác. Xem ví dụ 3.17 dưới đây.

Ví dụ 3.17

Cho chu kỳ máy là 1.085μs, hãy tính thời gian trễ ở chương trình con sau:

```
                              Chu kỳ máy
DELAY:
        MOV     R2,#200         1
AGAIN:  MOV     R3,#250         1
HERE:   NOP                     1
        NOP                     1
        DJNZ    R3,HERE         2
        DJNZ    R2,AGAIN        2
        RET                     1
```

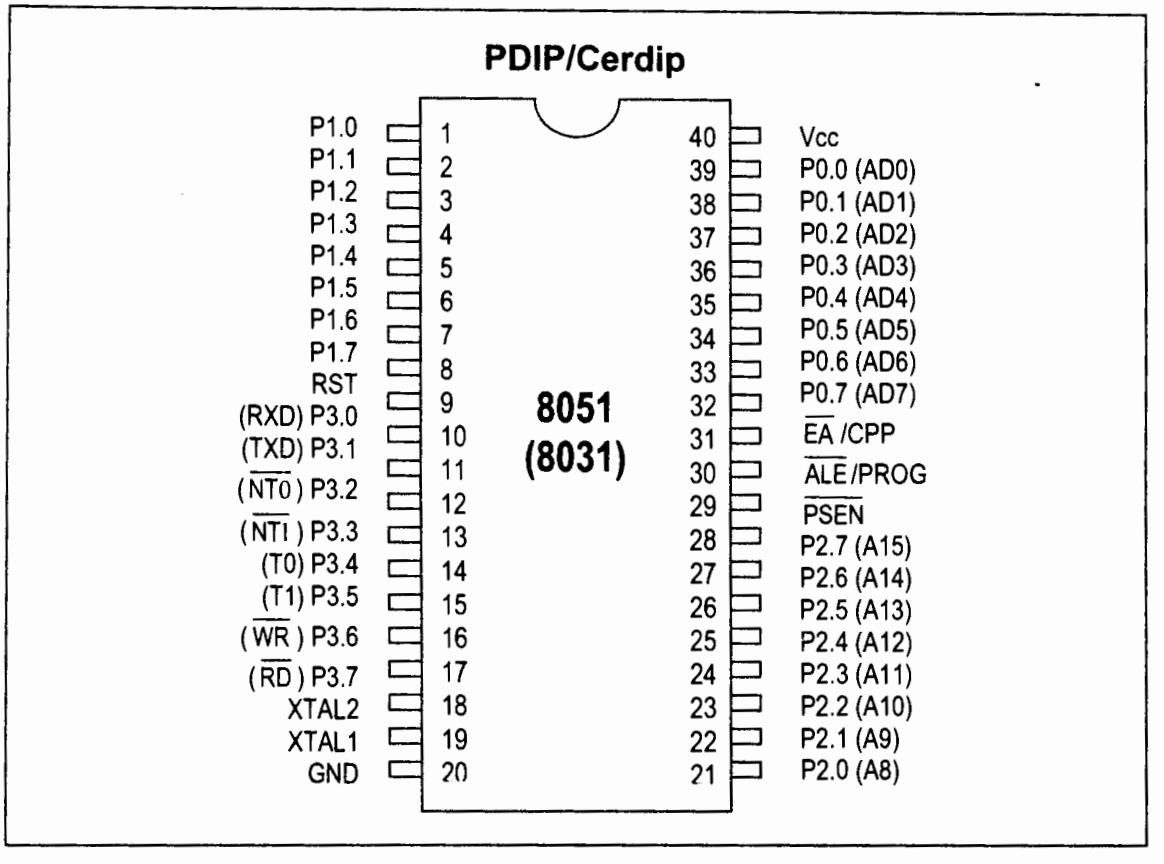
Giải:

Riêng vòng lặp HERE ta có (4 × 250) × 1.085μs = 1085μs. Vòng lặp AGAIN thực hiện lặp lại vòng lặp HERE 200 lần, do vậy thời gian trễ do vòng lặp HERE là 200 × 1085μs = 217000μs. Tuy nhiên, ở đây còn thêm thời gian trễ của lệnh "MOV R3, #250" và "DJNZ R2, AGAIN" ở đầu và cuối vòng lặp AGAIN tạo nên là (3 × 200 × 1.085μs) = 651μs. Do vậy, tổng thời gian trễ của cả chương trình DELAY là 217000 + 651 = 217651μs = 217,651 ms. Cũng nên lưu ý là, ở trường hợp vòng lặp lồng nhau cũng như ở các trường hợp vòng lặp giữ chậm khác, cách tính thời gian trễ như trên chỉ là gần đúng vì chưa tính đến các lệnh đầu và cuối của chương trình con.

Chương 4
CỔNG VÀO/RA VÀ LẬP TRÌNH

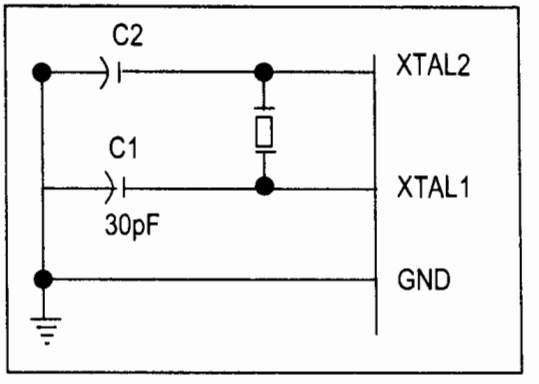
4.1 MÔ TẢ CHÂN CỦA 8051

Các thành viên của họ 8051, ví dụ 8751, 89C51, DS5000 đều có kiểu đóng vỏ khác nhau, chẳng hạn, dạng hai hàng chân DIP (Dual In - Line Pakage), dạng vỏ dẹt vuông QFP (Quad Flat Pakage) và dạng không có chân đỡ LLC (Leadless Chip Carrier), song chúng đều có 40 chân với các chức năng, như vào ra I/0, đọc $\overline{RD}$, ghi $\overline{WR}$, địa chỉ, dữ liệu và ngắt. Cần lưu ý là một số hãng sản xuất phiên bản 8051 có 20 chân với số cổng vào/ra ít hơn cho các ứng dụng yêu cầu thấp hơn. Tuy nhiên, vì hầu hết các nhà phát triển chính đều sử dụng chip đóng vỏ 40 chân kiểu hai hàng chân DIP nên chúng ta chỉ tập trung mô tả phiên bản này.

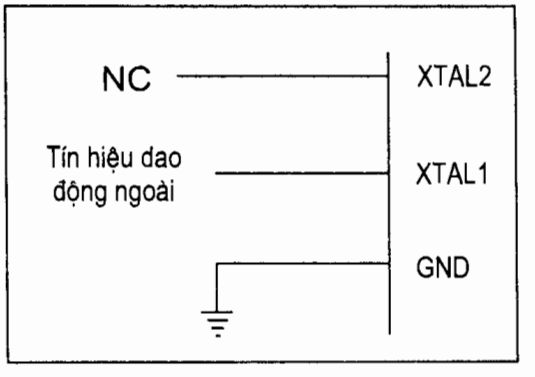


Hình 4.1. Bố trí chân của 8051

Hình 4.1 trình bày bố trí chân của 8051. Trong số 40 chân có 32 chân dành cho bốn cổng P0, P1, P2 và P3, mỗi cổng có 8 chân. Các chân còn lại dành cho nguồn V_{CC}, đất GND, các chân dao động XTAL1 và XTAL2, khởi động lại RST, cho phép chốt địa chỉ ALE, truy cập được địa chỉ ngoài $\overline{EA}$, cho phép cất chương trình $\overline{PSEN}$. Trong 8 chân này thì 6 chân V_{CC}, GND, XTAL1, XTAL2, RST và $\overline{EA}$ được các họ 8031 và 8051 sử dụng. Nói cách khác, các chân này cần được nối để cho hệ thống làm việc mà không phụ thuộc vào bộ vi điều khiển là 8051 hay 8031. Hai chân còn lại là $\overline{PSEN}$ và ALE được sử dụng chủ yếu trong các hệ thống dựa trên 8031.



Hình 4.2. a) Nối đồng hồ thạch anh



Hình 4.2. b) Nối đồng hồ ngoài

Nhóm chân nguồn, dao động và điều khiển

VCC **-** Chân số 40, cung cấp điện áp nguồn +5V cho chip.

GND **-** Chân số 20 là chân đất.

XTAL1 và XTAL2: 8051 có một bộ dao động trên chip nhưng vẫn cần có một bộ đồng hồ bên ngoài để kích hoạt. Bộ dao động thạch anh ngoài thường được nối tới các chân vào XTAL1 (chân 19) và XTAL2 (chân 18). Khi mắc dao động thạch anh, phải có hai tụ điện 30pF, một đầu mỗi tụ nối tới XTAL1 và XTAL2, còn đầu kia nối đất, như trình bày ở hình 4.2a.

Bảng 4.1. Giá trị một số thanh ghi sau RESET

Register	Reset Value
PC	0000
ACC	0000
B	0000
PSW	0000
SP	0000
DPTR	0007

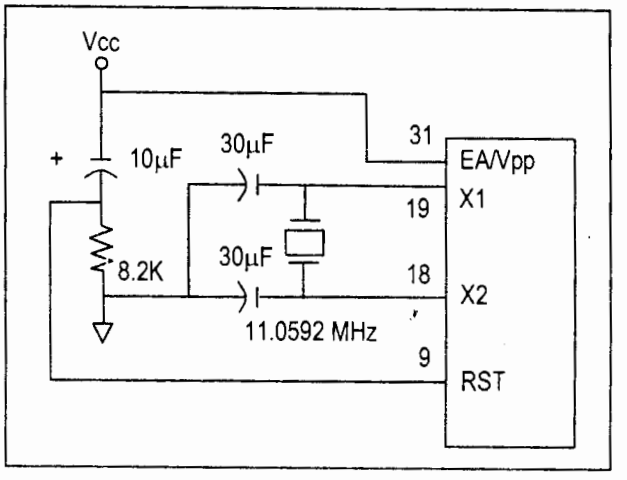
Cần lưu ý là họ 8051 có nhiều phiên bản tốc độ khác nhau. Tốc độ được hiểu là tần số cực đại của bộ dao động nối tới chân XTAL, ví dụ, chip 20 MHz, 12MHz hoặc thấp hơn. Giao động đồng hồ ngoài không nhất thiết là bộ dao động thạch anh mà cũng có thể dùng bộ dao động TTL. Khi đó bộ dao động được nối tới chân XTAL1, còn chân

XTAL2 để hở, như hình 4.2b.

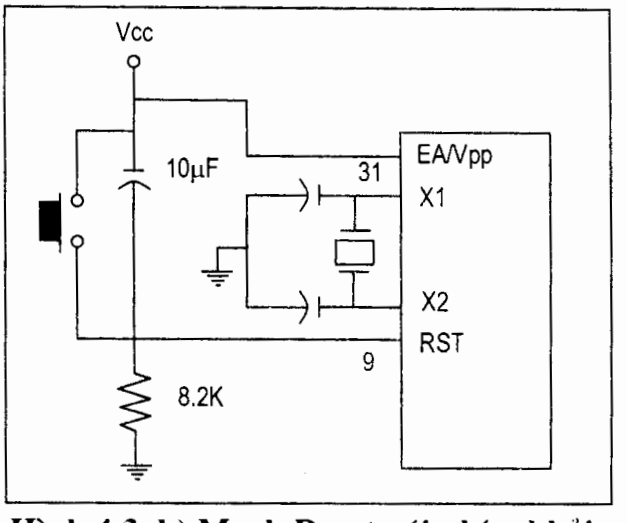
RST - Khởi động lại (RESET). Đó là chân vào, số 9, mức tích cực cao, bình thường ở mức thấp. Khi có xung cao đặt tới chân này thì bộ vi điều khiển sẽ kết thúc mọi hoạt động hiện tại và tiến hành khởi động lại. Quá trình xảy ra hoàn toàn tương tự như khi bật nguồn. Khi Reset, mọi giá trị trên các thanh ghi sẽ bị xoá. Bảng 4.1 giới thiệu giá trị các thanh ghi của 8051 khi thực hiện Reset.

Lưu ý rằng, khi Reset, giá trị của bộ đếm chương trình PC bằng 0, và như vậy CPU nhận mã lệnh đầu tiên tại địa chỉ 0000 của bộ nhớ ROM. Do đó, tại địa chỉ này phải có lệnh đầu tiên chương trình nguồn của ROM. Hình 4.3 trình bày hai cách nối chân RST với mạch Reset.

Để Reset có hiệu quả, chân RST cần duy trì trạng thái tích cực (mức cao) tối thiểu 2 chu kỳ máy.



Hình 4.3. a) Mạch nối chân RST



Hình 4.3. b) Mạch Reset với phím khởi động lại

Ở 8051 một chu kỳ máy bằng 12 chu kỳ dao động đồng hồ, như được đề cập ở chương 3.

$\overline{EA}$ - Truy cập bộ nhớ ngoài (External Access) là chân vào, số 31 trên vỏ kiểu DIP. Đối với các thành viên họ 8051 có ROM chương trình trên chip như 8751, 98C51 hoặc DS5000 thì chân $\overline{EA}$ được nối tới nguồn V_{CC}. Trường hợp không có ROM trên chip như 8031 và 8032 thì mã chương trình được lưu cất ở bộ nhớ ROM ngoài, khi đó chân $\overline{EA}$ được nối đất. Như vậy chân $\overline{EA}$ hoặc được nối với nguồn V_{CC} hoặc với đất GND chứ không bao giờ để hở.

Chương 14 sẽ giới thiệu phương pháp 8031 dùng chân này kết hợp với $\overline{PSEN}$ để truy cập các chương trình cất trên bộ nhớ ROM ở ngoài.

Ví dụ 4.1

Xác định chu kỳ máy với a) XTAL = 11.0592MHz b) XTAL = 16MHz.

Giải:

a. 11.0592MHz/12 = 921.6kHz.

Chu kỳ máy = 1/921.6kHz = 1.085μs.

b. 16MHz/12 = 1.333MHz

Chu kỳ máy = 1/1.333MHz = 0.75μs.

Trên đây là các chân mà mọi thành viên của họ 8051 đều có và đều được nối giống nhau. Dưới đây là một số chân sử dụng chủ yếu ở hệ thống 8031 và sẽ được trình bày chi tiết hơn ở chương 11.

$\overline{PSEN}$ - Là chân ra có chức năng cho phép cất chương trình (Program Store Enable). Ở hệ thống 8031, khi chương trình cất ở bộ nhớ ROM ngoài thì chân này được nối tới chân OE của ROM. Chi tiết được bàn ở chương 14.

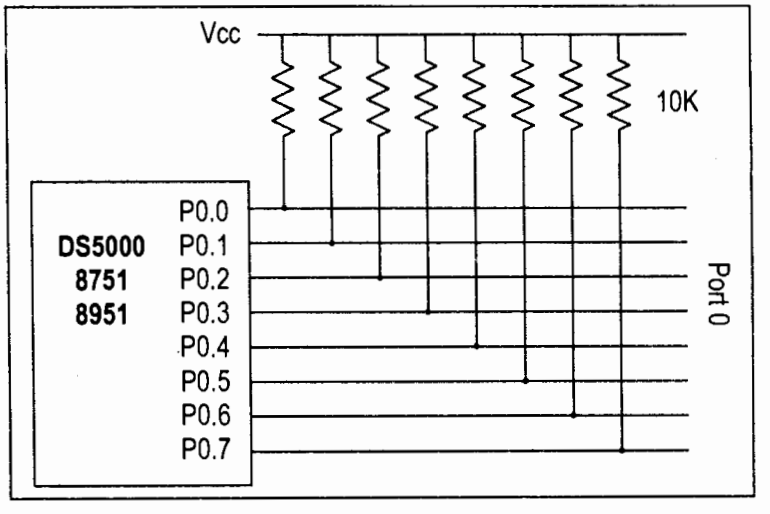
ALE - Cho phép chốt địa chỉ (Address Latch Enable) là chân ra có mức tích cực cao. Khi nối 8031 tới bộ nhớ ngoài thì cổng 0 cũng được cấp địa chỉ và dữ liệu. Hay nói cách khác, 8031 dồn địa chỉ và dữ liệu qua cổng 0 để tiết kiệm số chân. Chân ALE được sử dụng để phân kênh địa chỉ và dữ liệu bằng cách nối tới chân G của chip 74LS373.

Nhóm chân cổng Vào/Ra

Bốn cổng P0, P1, P2 và P3 đều có 8 chân và tạo thành cổng 8 bit. Tất cả các cổng khi Reset đều được cấu hình làm cổng ra. Để làm đầu vào thì cần được lập trình.

Cổng P0

Cổng P0 có 8 chân (từ chân 32 đến 39). Bình thường đây là cổng ra.



Hình 4.4. Mắc điện trở kéo cổng P0

Để có thể vừa làm đầu ra, vừa làm đầu vào thì mỗi chân phải được nối tới một điện trở kéo 10 kΩ bên ngoài. Sở dĩ như vậy là vì cổng P0 có dạng cực máng hở, đây là điểm khác với các cổng P1, P2 và P3. Khái niệm cực máng hở cũng tương tự như colector hở, tuy nhiên ở đây áp dụng cho các chip dạng MOS. Ở mọi hệ thống 8751, 89C51 hoặc DS5000 thì P0 luôn được nối với các điện trở kéo, xem hình 4.4, và như vậy ta có được cổng vào ra P0. Với các điện trở kéo ngoài, khi khởi động lại, cổng P0 được cấu hình làm cổng ra. Ví dụ, đoạn chương trình sau sẽ liên tục gửi ra cổng P0 các giá trị 55H và AAH.

```
        MOV    A,#554
BACK:   MOV    P0,A
        ACALL  DELAY
        CPL    A
        SJMP   BACK
```

P0 làm cổng vào

Khi có các điện trở nối tới cổng P0, để tạo thành cổng vào thì cần phải lập trình bằng cách ghi 1 tới tất cả các bit của cổng. Đoạn chương trình sau sẽ cấu hình P0 làm cổng vào theo cách vừa nêu, sau đó dữ liệu nhận được ở cổng này sẽ gửi đến P1.

```
        MOV    A,#FFH     ;Gán A = FF dạng Hexa
        MOV    P0, A      ;Đặt P0 làm cổng vào bằng cách
                          ;Ghi 1 vào tất cả các bit
BACK:   MOV    A, P0      ;Nhận dữ liệu từ P0
        MOV    P1, A      ;Gửi nó đến cổng 1
        SJMP   BACK       ;Lặp lại
```

Chuyển dữ liệu và địa chỉ qua cổng P0

Như trình bày ở hình 4.1, P0 ngoài chức năng chuyển dữ liệu còn được dùng để chuyển 8 bit địa chỉ AD0 - AD7. Khi nối 8051/31 tới bộ nhớ ngoài, thì cổng P0 cung cấp cả địa chỉ và dữ liệu bằng cách dồn kênh để tiết kiệm số chân. Chân ALE sẽ báo P0 có địa chỉ hay dữ liệu. Nếu ALE = 0 thì P0 cấp dữ liệu D0 - D7, còn nếu ALE=1 thì là địa chỉ.

Cổng P1

Cổng P1 cũng có 8 chân, từ chân 1 đến chân 8, và có thể sử dụng làm đầu

vào hoặc ra. Khác với cổng P0, cổng P1 không cần đến điện trở kéo vì nó đã có các điện trở kéo bên trong. Khi Reset , cổng P1 được cấu hình làm cổng ra. Ví dụ, đoạn chương trình sau sẽ gửi liên tục các giá trị 55H và AAH ra cổng P1.

```
        MOV     A, #55H
BACK:   MOV     P1, A
        ACAL    DELAY
        SJMP    BACK
```

Cổng P1 làm đầu vào

Để chuyển cổng P1 thành đầu vào cần lập trình bằng cách ghi 1 đến tất cả các bit của cổng. Chi tiết về vấn đề này được nêu ở phụ lục C.2. Trong đoạn chương trình sau, cổng P1 được cấu hình làm cổng vào bằng cách ghi 1 vào các bit của cổng, sau đó dữ liệu nhận được từ cổng này được cất vào R7, R6 và R5.

```
MOV   A,#0FFH   ;Nạp A = FF ở dạng hexa
MOV   P1,A      ;Tạo cổng P1 thành cổng đầu vào bằng
                ;cách ghi 1 vào các bit của cổng
MOV   A,P1      ;Nhận dữ liệu từ P1
MOV   R7,A      ;Cất nó vào thanh ghi R7
ACALL DELAY     ;Chờ
MOV   A,P1      ;Nhận dữ liệu khác từ P1
MOV   R6,A      ;Cất nó vào thanh ghi R6
ACALL DELAY     ;Chờ
MOV   A,P1      ;Nhận dữ liệu khác từ cổng P1
MOV   R5,A      ;Cất dữ liệu vào thanh ghi R5
```

Cổng P2

Cổng P2 cũng có 8 chân, từ chân 21 đến 28, và có thể được sử dụng làm đầu vào hoặc đầu ra. Cũng giống như cổng P1, cổng P2 không cần điện trở kéo vì bên trong đã có các điện trở kéo. Khi Reset, thì cổng P2 được cấu hình làm đầu ra. Ví dụ đoạn chương trình sau sẽ gửi liên tục ra cổng P2 các giá trị 55H và AAH.

```
      MOV   A, #55H
BACK: MOV   P2,A
      ACALL DELAY
      CPL   A
      SJMP  BACK
```

Cổng P2 làm đầu vào

Để P2 làm cổng vào thì cần được lập trình bằng cách ghi các số 1 tới tất cả các chân của cổng. Đoạn chương trình sau đây đầu tiên định cấu hình P2 làm cổng vào bằng cách ghi 1 đến tất cả các chân của cổng, sau đó dữ liệu nhận được từ P2 được gửi liên tục đến P1.

```
      MOV   A,0FFH   ;Gán A giá trị FF dạng Hexa
      MOV   P2,A     ;Đặt P2 làm cổng vào bằng cách
                     ;ghi 1 đến tất cả các chân của cổng
BACK: MOV   A,2      ;Nhận dữ liệu từ P2
      MOV   P1, A    ;Gửi dữ liệu đến P1
      SJMP  BACK     ;Lặp lại
```

Hai chức năng của cổng P2: Chuyển địa chỉ và dữ liệu

Ở các hệ thống 8751, 89C51 và DS5000, P2 được dùng làm cổng vào/ra. Tuy nhiên, ở hệ thống 8031 thì cổng P2 còn được sử dụng cùng với P0 để tạo ra địa chỉ 16 bit cho bộ nhớ ngoài. Vì 8031 có khả năng truy cập 64k byte bộ nhớ ngoài, nên cần có bus địa chỉ 16 bit. P0 cung cấp 8 bit địa chỉ thấp qua A0 - A7, còn lại 8 bit địa chỉ cao A8 - A15 do P2 cung cấp. Như vậy, khi 8031được nối tới bộ nhớ ngoài thì 8 bit P2 được dùng cho địa chỉ 16 bit và không thể dùng cho vào ra được.

Từ những trình bày trên đây, có thể thấy rằng, trong các hệ thống 8751, 89C51 hoặc DS5000 thì chúng ta có ba cổng P0, P1 và P2 cho các thao tác vào ra và như thế là có thể đủ cho hầu hết các ứng dụng. Còn lại cổng P3 được dùng cho ngắt và ta sẽ cùng tìm hiểu dưới đây.

Bảng 4.2. Các chức năng khác của cổng P3

Bit P3	Chức năng	Chân
P3.0	Nhận dữ liệu (RXD)	10
P3.1	Phát dữ liệu (TXD)	11
P3.2	Ngắt 0(INT0)	12
P3.3	Ngắt 1(INT1)	13
P3.4	Bộ định thời 0 (T0)	14
P3.5	1 Bộ định thời 1(T1)	15
P3.6	Ghi (WR)	16
P3.7	Đọc (RD)	17

Cổng P3

Cổng P3 chiếm 8 chân, từ chân 10 đến chân 17. Cổng này có thể được sử dụng làm đầu vào hoặc đầu ra. Cũng như P1 và P2, cổng P3 không cần điện trở kéo. Khi Reset, cổng P3 được cấu hình làm một

cổng ra, tuy nhiên đây không phải là ứng dụng chủ yếu. Cổng P3 có thêm một chức năng quan trọng khác là cung cấp một số tín hiệu đặc biệt, chẳng hạn như ngắt. Bảng 4.2 giới thiệu các chức năng khác của cổng P3. Bảng này được sử dụng cho cả 8051 và 8031.

Bit P3.0 và P3.1 được dùng để thu và phát dữ liệu trong truyền thông nối tiếp. Bit P3.2 và P3.3 được dành cho ngắt ngoài và sẽ được trình bày chi tiết ở chương 11. Bit P3.4 và P3.5 được dùng cho các bộ định thời 0 và 1 và chi tiết được trình bày ở chương 9. Cuối cùng bit P3.6 và P3.7 dùng để ghi, đọc các bộ nhớ ngoài. Trong các hệ thống 8751, 89C51 hoặc D35000, các chân P3.6 và P3.7 được dùng cho vào/ra, các chân còn lại của P3 được dùng với các chức năng khác nhau.

4.2 LẬP TRÌNH VÀO - RA, THAO TÁC BIT

Phương pháp truy cập toàn bộ 8 bit

Đoạn chương trình sau giới thiệu cách truy cập toàn bộ 8 bit của cổng P1 .

```
BACK:  MOV    A,#55H
       MOV    P1,A
       ACALL  DELAY
       MOV    A,#0AAH
       MOV    P1,A
       ACALL  DELAY
       SJMP   BACK
```

Đoạn chương trình trên liên tục thay đổi giá trị bit của P1. Chương trình dạng này cũng đã được biết ở phần trên. Bây giờ chúng ta viết lại đoạn chương trình trên gọn hơn bằng cách truy cập trực tiếp cổng mà không cần thông qua thanh ghi tích luỹ như sau:

```
BACK:  MOV    P1,#55H
       ACALL  DELAY
       MOV    P1,#00H
       CALL   DELAY
       SJMP   BACK
```

Cũng có thể trình bày đoạn chương trình trên bằng kỹ thuật đọc-sửa đổi- ghi như ở phần trình bày sau đây.

Đọc-Sửa đổi -Ghi (Read - Modify - Write)

Các cổng của 8051 có thể được truy cập bằng kỹ thuật được gọi là: Đọc-Sửa đổi-Ghi. Phương pháp này giảm thiểu được nhiều dòng lệnh nhờ kết hợp tất cả ba thao tác: 1) đọc cổng, 2) sửa đổi và 3) ghi ra cổng vào một lệnh đơn. Đoạn chương trình sau đầu tiên đặt 01010101 (nhị phân) vào cổng 1, sau đó lệnh "XLR P1, #0FFH" thực hiện phép logic XOR (OR loại trừ) trên cổng P1 với 1111 1111 (nhị phân) và cuối cùng ghi kết quả trở lại cổng P1.

Bảng 4.3. Các cổng định địa chỉ bit

P0	P1	P2	P3	Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

```
        MOV   P1,#55H      ;P1 = 01010101
AGAIN:  XLR   P1,#0FFH     ;P1 XOR 1111 1111
        ACALL DELAY
        SJMP  AGAIN
```

Lưu ý rằng lệnh XOR 55H với FFH cho kết quả là AAH. Tương tự như vậy, lệnh XOR AAH với FFH lại cho giá trị kết quả là 55H. Các lệnh logic được trình bày ở chương 7.

Định địa chỉ bit của các cổng

Nhiều ứng dụng chúng ta chỉ cần truy cập 1 hoặc 2 bit của cổng thay vì truy cập cả 8 bit. Một điểm mạnh của các cổng 8051 là có khả năng truy cập từng bit một mà không làm thay đổi các bit còn lại của cổng. Đoạn chương trình ở ví dụ sau liên tục đảo giá trị của bit P1.2.

```
BACK:   CPL   P1.2      ;Lấy bù bit P1.2
        ACALL DELAY
        SJMP  BACK
```

Một biến thể khác của đoạn chương trình trên như sau:

```
AGACN:  SETB  P1.2      ;Đặt bit P1.2 lên cao
        ACALL DELAY
        CLR   P1.2      ;Xoá bit P1.2 xuống thấp
        ACALL DELAY
        SJMP  AGAIN
```

Bảng 4.3 trình bày các bit các cổng vào ra của 8051. Xem ví dụ 4.2 về thao tác bit trên các bit vào/ra. Nên lưu ý là ở ví dụ 4.2, những bit không được dùng thì không bị thay đổi. Đây là khả năng định địa chỉ bit của các cổng vào/ra và là một trong những điểm rất mạnh của bộ vi điều khiển 8051.

Ví dụ 4.2

Hãy viết chương trình thực hiện các công việc sau:

a) Duy trì kiểm tra bit P1.2 cho đến khi bit này lên cao.

b) Khi P1.2 lên cao, hãy ghi giá trị 45H vào cổng P0.

c) Gửi một xung Cao-Xuống-Thấp (H-to-L) tới P2.3.

Giải:

```
        SET   P1.2          ;Tạo bit P1.2 là đầu vào
        MOV   A,#45H        ;Gán A = 45H
AGAIN:  JNB   P1.2,AGAIN    ;Thoát khi P1.2 = 1
        MOV   P0,A          ;Xuất A tới cổng P0
        SETB  P2.3          ;Đưa P2.3 lên cao
        CLR   P2.3          ;Xoá P2.3 để tạo xung H-to-L
```

Trong chương trình này lệnh "JNB P1.2, AGAIN" ở lại vòng lặp cho đến khi P1.2 chưa lên cao. Khi P1.2 lên cao, chương trình thoát khỏi vòng lặp, ghi giá trị 45H tới cổng P0 và tạo ra xung H-to-L nhờ chuỗi lệnh SETB và CLR.

Chương 5
CHẾ ĐỘ ĐỊNH ĐỊA CHỈ

CPU có thể truy cập dữ liệu theo nhiều cách khác nhau. Dữ liệu có thể ở trong một thanh ghi, ở trong bộ nhớ hoặc được cho dưới dạng một giá trị tức thời. Cách CPU truy cập dữ liệu được gọi là chế độ định địa chỉ. Chương này chúng ta tìm hiểu các chế độ định địa chỉ của 8051.

Các chế độ định địa chỉ được xác định từ khi thiết kế bộ vi xử lý và không thể bị thay đổi do lập trình viên. 8051 có tất cả 5 chế độ định địa chỉ như sau:

1. Tức thời
2. Thanh ghi
3. Trực tiếp
4. Gián tiếp qua thanh ghi
5. Chỉ số

5.1 CHẾ ĐỘ ĐỊNH ĐỊA CHỈ TỨC THỜI VÀ THANH GHI

Chế độ định địa chỉ tức thời

Ở chế độ định địa chỉ này, toán hạng nguồn là một hằng số và như tên gọi, toán hạng có ngay sau mã lệnh. Lưu ý rằng, trước dữ liệu tức thời cần có dấu "#". Chế độ định địa chỉ này có thể được dùng để nạp dữ liệu vào mọi thanh ghi, kể cả thanh ghi con trỏ dữ liệu DPTR. Ví dụ:

```
MOVA,#25H          ;Nạp giá trị 25H vào thanh ghi A
MOVR4,#62          ;Nạp giá trị  62 thập phân vào R4
MOVB,#40H          ;Nạp giá trị 40 H vào thanh ghi B
MOVDPTR,#4521H     ;Nạp 4512H vào con trỏ dữ liệu DPTR
```

DPTR là thanh ghi 16 bit, tuy vậy nó cũng có thể được truy cập như hai thanh ghi 8 bit DPH và DPL, trong đó DPH là byte cao và DPL là byte thấp. Xét đoạn chương trình sau:

```
MOVD   PTR,#2550H
MOV    A,#50H
MOV    DPH,#25H
```

Cũng nên lưu ý là lệnh dưới đây có thể tạo ra lỗi vì giá trị nạp vào DPTR lớn

hơn 16 bit:

```
MOV DPTR,#68975    ;Giá trị không hợp lệ ( >65535=FFFFH)
```

Có thể dùng chỉ dẫn EQU để truy cập dữ liệu tức thời như sau:

```
    COUNT  EQU 30
    ...    ...
    MOV    R4,#COUNT        ;R4=1E (30=1EH)
    MOV    DPTR,#MYDATA     ;DPTR=200H
    ORG    200H
MYDATA:    DB "Nha Trang"
```

Tất nhiên chúng ta cũng có thể sử dụng chế độ định địa chỉ tức thời để gửi dữ liệu đến các cổng của 8051.

Ví dụ, "MOV P1,#55H" là một lệnh hợp lệ.

Chế độ định địa chỉ thanh ghi

Chế độ định địa chỉ thanh ghi là sử dụng các thanh ghi để lưu dữ liệu cần thao tác. Chúng ta xem các ví dụ sau:

```
MOV A,R0   ;Sao nội dung thanh ghi RO vào thanh ghi A
MOV R2,A   ;Sao nội dung thanh ghi A vào thanh ghi R2
ADD A;R5   ;Cộng nội dung thanh ghi R5 vào thanh ghi A
ADD A,R7   ;Cộng nội dung thanh ghi  R7 vào thanh ghi A
MOV R6,A   ;Sao nội dung thanh ghi A vào thanh ghi R6
```

Nên lưu ý là thanh ghi nguồn và đích phải phù hợp về kích thước. Nói cách khác, lệnh " MOV DPTR, A" sẽ gây lỗi vì nguồn là thanh ghi 8 bit và đích lại là thanh ghi 16 bit. Xét đoạn chương trình sau:

```
MOV    DPTR,#25F5H
MOV    R7,DPL
MOV    R6,DPH
```

Để ý rằng, ta có thể chuyển dữ liệu giữa thanh ghi tích luỹ A và thanh ghi Rn (n có giá trị từ 0 đến 7) nhưng chuyển dữ liệu giữa các thanh ghi Rn thì không được phép. Ví dụ, lệnh "MOV R4, R7" là không hợp lệ.

Như vậy, ở hai chế độ định địa chỉ vừa xét ở trên, toán hạng hoặc ở trong thanh ghi hoặc là một giá trị trực tiếp trong lệnh. Tuy nhiên, trong nhiều trường hợp, dữ liệu lại nằm ở bộ nhớ RAM hoặc ROM. Sau đây, chúng ta xem xét cách truy cập dữ liệu ở những trường hợp này.

5.2 CÁC CHẾ ĐỘ ĐỊNH ĐỊA CHỈ TRUY CẬP BỘ NHỚ

Chế độ định địa chỉ trực tiếp

Như đã nói ở chương 2, trong 8051 có 128 byte bộ nhớ RAM. Bộ nhớ RAM được gán địa chỉ từ 00 đến FFH và được phân chia như sau:

- Các ngăn nhớ từ 00 đến 1FH được gán cho các băng thanh ghi và ngăn xếp.
- Các ngăn nhớ từ 20H đến 2FH được dành cho không gian định địa chỉ bit để lưu dữ liệu theo từng bit.
- Các ngăn nhớ từ 30H đến 7FH là không gian để lưu dữ liệu có kích thước 1 byte.

Chế độ định địa chỉ trực tiếp có thể truy cập toàn bộ không gian của bộ nhớ RAM. Tuy nhiên, chế độ này thường được dùng để truy cập các ngăn nhớ RAM từ 30H đến 7FH, vì thực tế đối với không gian nhớ dành cho băng thanh ghi thì đã được truy cập bằng tên thanh ghi như R0 - R7. Ở chế độ định địa chỉ trực tiếp, địa chỉ ngăn nhớ RAM chứa dữ liệu là toán hạng của lệnh. Dễ thấy sự khác nhau với chế độ định địa chỉ tức thì, ở đó dữ liệu (chứ không phải là địa chỉ) tự nó là toán hạng của lệnh. Dấu "#" là dấu hiệu phân biệt giữa hai chế độ định địa chỉ. Xét các ví dụ dưới đây và lưu ý rằng các lệnh không có dấu (#):

```
MOV R0,40     ;Sao nội dung ngăn nhớ 40H của RAM vào R0
MOV 56H,A     ;Sao thanh ghi A vào ngăn nhớ RAM 56H
MOV R4,7FH    ;Chuyển nội dung ngăn nhớ 7FH vào R4
```

Như đã nói ở phần trên, các ngăn nhớ từ 0 đến 7 của RAM được cấp cho băng 0 của các thanh ghi R0 - R7. Các thanh ghi này có thể được truy cập theo hai cách như sau:

```
MOV A,4       ;Lệnh này và lệnh (MOV A,R4)là như nhau
MOV A,R4      ;Sao nội dung thanh ghi R4 vào A
MOV A,7       ;Lệnh này và lệnh (MOV A,R7)là như nhau
MOV A,R7      ;Sao nội dung thanh ghi R7 vào A
```

Để rõ hơn về vai trò của dấu (#) trong các lệnh của 8051, chúng ta sẽ xem xét các đoạn chương trình sau đây:

```
MOV R2,#05    ;Gán R2=05
MOV A,2       ;Sao nội dung thanh ghi R2 vào A (A=R2=5)
MOV B,2   ;Sao nội dung thanh ghi R2 vào B (B=R2=5)
```

```
MOV 7,2   ;Sao R2 vào R7
          ;vì lệnh "MOV R7,R2" là không hợp lệ.
```

Dễ thấy sử dụng tên R0 - R7 để truy cập bộ nhớ thì rõ ràng hơn là dùng địa chỉ bộ nhớ. Tuy nhiên, đối với các ngăn nhớ từ 30H đến 7FH thì không thể truy cập theo cách như vậy được mà phải dùng địa chỉ của chúng, vì đơn giản ở vùng này chúng không có tên.

Nhóm thanh ghi SFR và địa chỉ của chúng

Trong số các thanh ghi chúng ta tìm hiểu trên đây, ví dụ R0 - R7, đều là một phần trong 128 byte của bộ nhớ RAM. Thế còn các thanh ghi như A, B, PSW và DPTR liệu có địa chỉ không? Câu trả lời là có. Các thanh ghi A, B, PSW và DPTR thuộc nhóm thường được gọi là nhóm thanh ghi đặc biệt SFR (Special Function Registers). Có nhiều thanh ghi với chức năng đặc biệt và chúng được sử dụng rộng rãi mà chúng ta sẽ có dịp đề cập đến ở các chương sau. Các thanh ghi SFR có thể truy cập theo tên (và như thế dễ dàng hơn nhiều) hoặc theo địa chỉ. Ví dụ, địa chỉ của thanh ghi A là E0H và thanh ghi B là F0H như giới thiệu ở bảng 5.1. Sau đây là ví dụ các cặp lệnh có cùng chức năng:

```
MOV   0E0H,#55H    ;Nạp 55H vào thanh ghi A(A=55H)
MOV   A,#55H       ;
MOV   0F0H,#25H    ;Nạp 25H vào thanh ghi B(B=25)
MOV   3,#25H       ;
MOV   0E0H,R2      ;Sao nội dung thanh ghi R2 vào A
MOV   A,R2         ;
MOV   0F0H,R0      ;Sao nội dung thanh ghi R0 vào B
MOV   B,R0         ;
```

Bảng 5.1 dưới đây liệt kê các thanh ghi đặc biệt SFR của 8051 và địa chỉ của chúng. Cần phải lưu ý đến hai điểm sau về địa chỉ của SFR:

1. Các thanh ghi SFR có địa chỉ nằm trong khoảng từ 80H đến FFH, vì các địa chỉ từ 00 đến 7FH là địa chỉ bộ nhớ RAM trong của 8051.

2. Không phải tất cả mọi địa chỉ từ 80H đến FFH đều dành cho SFH sử dụng. Những ngăn nhớ trong khoảng từ 80H đến FFH chưa dùng là dự trữ nhưng lập trình viên 8051 không được sử dụng.

Bảng 5.1. Địa chỉ của thanh ghi chức năng đặc biệt SFR

Lệnh	Tên	Địa chỉ
ACC*	Thanh ghi tích luỹ (thanh ghi tổng) A	0E0H
B*	Thanh ghi B	0F0H
PSW*	Từ trạng thái chương trình	0D0H
SP	Con trỏ ngăn xếp	81H
DPTR	Con trỏ dữ liệu hai byte	
DPL	Byte thấp của DPTR	82H
DPH	Byte cao của DPTR	83H
P0*	Cổng 0	80H
P1*	Cổng 1	90H
P2*	Cổng 2	0A0H
P3*	Cổng 3	0B0H
IP*	Điều khiển ưu tiên ngắt	0B8H
IE*	Điều khiển cho phép ngắt	0A8H
TMOD	Điều khiển chế độ bộ đếm/bộ định thời	89H
TCON*	Điều khiển bộ đếm/bộ định thời	88H
T2CON*	Điều khiển bộ đếm/bộ định thời 2	0C8H
T2MOD	Điều khiển chế độ bộ đếm/bộ định thời 2	0C9H
TH0	Byte cao của bộ đếm/bộ định thời 0	8CH
TL0	Byte thấp của bộ đếm/bộ định thời 0	8AH
TH1	Byte cao của bộ đếm/bộ định thời 1	8DH
TL1	Byte thấp của bộ đếm/bộ định thời 1	8BH
TH2	Byte cao của bộ đếm/bộ định thời 2	0CDH
TL2	Byte thấp của bộ đếm/bộ định thời 2	0CCH
RCAP2H	Byte cao của thanh ghi bộ đếm/bộ định thời 2	0CBH
RCAP2L	Byte thấp của thanh ghi bộ đếm/bộ định thời 2	0CAH
SCON*	Điều khiển nối tiếp	98H
SBUF	Bộ đệm dữ liệu nối tiếp	99H
PCON	Điều khiển công suất	87H

Ghi chú: * Các thanh ghi có thể định địa chỉ bit.

Nên lưu ý ở chế độ định địa chỉ trực tiếp là địa chỉ truy cập được đến từng byte, trong khoảng 00 - FFH. Như vậy, nếu sử dụng chế độ định địa chỉ này thì Bạn có thể truy cập được các ngăn nhớ RAM và các thanh ghi bên trong 8051.

Ví dụ 5.1

Viết chương trình để gửi 55H đến cổng P1 và P2 sử dụng hoặc
a) Tên các cổng
b) Hoặc địa chỉ các cổng

Giải:

```
a)  MOV    A,#55H     ;A=55H
    MOV    P1,A       ;P1=55H
    MOV    P2,A       ;P2=55H
```

Từ bảng 5.1 ta có địa chỉ cổng P1 là 80H và P2 là A0H

```
    MOV    A,#55H     ;A=55H
    MOV    80H,A      ;P1=55H
    MOV    0A0H A     ;P2=55H
```

Ngăn xếp và chế độ định địa chỉ trực tiếp

Một ứng dụng quan trọng của chế độ định địa chỉ trực tiếp là ngăn xếp. Trong họ 8051, chỉ có chế độ định địa chỉ trực tiếp là được phép cất và lấy dữ liệu từ ngăn xếp. Như vậy, lệnh "PUSH A" là không hợp lệ. Lệnh trên cần viết dưới dạng sau "PUSH 0E0H", trong đó 0E0H là địa chỉ của thanh ghi A. Tương tự như vậy, để cất thanh ghi R3 băng 0 vào ngăn xếp, ta cần viết là "PUSH 03". Tương tự, đối với lệnh POP cũng cần sử dụng chế độ định địa chỉ trực tiếp.

Ví dụ 5.2

Viết đoạn chương trình cất thanh ghi R5, R6 và A vào ngăn xếp và sau đó lấy ra và cho vào các thanh ghi R2, R3 và B tương ứng.

Giải:

```
    PUSH 05       ;Đẩy R5 vào ngăn xếp
    PUSH 06       ;Đẩy R6 vào ngăn xếp
    PUSH 0E0H     ;Đẩy thanhghi A vào ngăn xếp
    POP  0F0H     ;Kéo đỉnh ngăn xếp cho vào thanh ghi B
                  ;Bây giờ B=A
    POP 02        ;Kéo đỉnh ngăn xếp cho vào thanh ghi R2
                  ;Bây giờ R2=R6
    POP 03        ;Kéo đỉnh ngăn xếp cho vào thanh ghi
                  ;Bây giờ R3=R5
```

Chế độ định địa chỉ gián tiếp thanh ghi

Ở chế độ này, thanh ghi được dùng để trỏ đến dữ liệu có trong bộ nhớ. Nếu dữ liệu có trên chip CPU thì chỉ các thanh ghi R0 và R1 mới được sử dụng, và như vậy cũng có nghĩa không thể dùng các thanh ghi R2 - R7 để trỏ đến địa chỉ của toán hạng ở chế độ định địa chỉ này. Nếu R0 và R1 được dùng làm con trỏ, nghĩa là chúng lưu địa chỉ của các ngăn nhớ RAM thì trước các thanh ghi cần đặt dấu "@" như ví dụ sau:

```
MOV A,@R0     ;Chuyển ngăn nhớ RAM có địa chỉ ở R0 vào A
MOV @R1,B     ;Chuyển B vào ngăn nhớ RAM có địa chỉ ở R1
```

Lưu ý rằng, ở đây R0 cũng như R1 đều có dấu "@" đứng trước. Nếu không có dấu "@" thì đó là lệnh chuyển nội dung thanh ghi R0 và R1 chứ không phải dữ liệu ngăn nhớ có địa chỉ trong R0 và R1.

Ví dụ 5.3

Viết chương trình để sao giá trị 55H vào các ngăn nhớ RAM tại địa chỉ từ 40H đến 44H sử dụng:

a) Chế độ định địa chỉ trực tiếp.

b) Chế độ định địa chỉ gián tiếp thanh ghi không dùng vòng lặp.

c) Chế độ b có dùng vòng lặp.

Giải:

a)

```
MOV    A,#55H     ;Nạp A giá trị 55H
MOV    40H,A      ;Sao A vào ngăn nhớ RAM  40H
MOV    41H,A      ;Sao A vào ngăn nhớ RAM  41H
MOV    42H,A      ;Sao A vào ngăn nhớ RAM  42H
MOV    43H,A      ;Sao A vào ngăn nhớ RAM  43H
MOV    44H,A      ;Sao A vào ngăn nhớ RAM  44H
```

b)

```
MOV    A,# 55H    ;Nạp vào A giá trị 55H
MOV    R0,#40H    ;Nạp con trỏ R0=40H
MOV    @R0,A      ;Sao A vào ngăn nhớ RAM do R0 trỏ đến
INC    R0         ;Tăng con trỏ. Bây giờ R0=41H
MOV    @R0,A      ;Sao A vào ngăn nhớ RAM do R0 trỏ đến
```

```
   INC    R0          ;Tăng con trỏ. Bây giờ R0=42H
   MOV    @R0,A       ;Sao A vào ngăn nhớ RAM do R0 trỏ đến
   INC    R0          ;Tăng con trỏ. Bây giờ R0=43H
   MOV    @R0,A       ;Sao A vào ngăn nhớ RAM do R0 trỏ đến
   INC    R0          ;Tăng con trỏ. Bây gờ R0=44H
   MOV    @R0,A
c)
   MOV    A,#55H      ;Nạp vào A giá trị 55H
   MOV    R0,#40H     ;Nạp con trỏ. Địa chỉ RAM R0=40H
   MOV    R2,#05      ;Nạp bộ đếm R2=5
AGAIN:
   MOV    @R0,A       ;Sao A vào ngăn nhớ RAM do R0 chỉ đến
   INC                ;Tăng con trỏ R0
   DJNZ   R2,AGAIN    ;Lặp lại cho đến khi bộ đếm=0.
```

Ưu điểm của chế độ định địa chỉ gián tiếp thanh ghi

Một trong những ưu điểm của chế độ định địa chỉ gián tiếp thanh ghi là cho phép truy cập dữ liệu linh hoạt hơn so với chế độ định địa chỉ trực tiếp. Ví dụ 5.3 giới thiệu đoạn chương trình sao giá trị 55H vào các vị trí ngăn nhớ của RAM từ 40H đến 44H . Lưu ý rằng lời giải b) có hai lệnh được lặp lại một số lần. Như vậy có thể tạo ra một vòng lặp hai lệnh này như ở lời giải c). Lời giải c) rõ ràng hiệu quả hơn cả và chỉ có thể sử dụng ở chế độ định địa chỉ gián tiếp qua thanh ghi. Vòng lặp không dùng được ở chế độ định địa chỉ trực tiếp. Đây là sự khác nhau chủ yếu giữa định địa chỉ trực tiếp và gián tiếp.

Ví dụ 5.4

Hãy viết chương trình xoá 16 ngăn nhớ RAM bắt đầu từ địa chỉ 60H.

Giải:

```
      CLR    A           ;Xoá A=0
      MOV    R1,#60H     ;Nạp con trỏ.R1=60H
      MOV    R7,#16H     ;Nạp bộ đếm,R7=16(10 H dạng hexa)
AGAIN:
      MOV    @R1,A       ;Xoá ngăn nhớ RAM do R1 trỏ đến
      INC    R1          ;Tăng R1
      DJNZ   R7,AGAIN    ;Lặp lại cho đến khi bộ đếm=0
```

Ví dụ 5.5 minh hoạ cách sử dụng R0 và R1 ở chế độ định địa chỉ gián tiếp thanh ghi khi truyền khối.

Ví dụ 5.5

Hãy viết chương trình để sao một khối 10 byte dữ liệu từ vị trí ngăn nhớ RAM bắt đầu từ 35H vào các vị trí ngăn nhớ RAM bắt đầu từ 60H

Giải:

```
      MOV   R0,#35H      ;Con trỏ nguồn
      MOV   R1,#60H      ;Con trỏ đích
      MOV   R3,#10       ;Bộ đếm
BACK: MOV   A,@R0        ;Lấy 1 byte từ nguồn
      MOV   @R1,A        ;Sao chép đến đích
      INC   R0           ;Tăng con trỏ nguồn
      INC   R1           ;Tăng con trỏ đích
      DJNZ  R3,BACK      ;Lặp cho đến khi hết 10 byte
```

Hạn chế của chế độ định địa chỉ gián tiếp thanh ghi ở 8051

Như đã nói ở phần trước, R0 và R1 là các thanh ghi duy nhất được dùng để làm con trỏ ở chế độ định địa chỉ gián tiếp thanh ghi. R0 và R1 là các thanh ghi 8 bit, nên chúng chỉ cho phép truy cập đến các ngăn nhớ RAM trong, từ địa chỉ 30H đến 7FH và các thanh ghi SFR. Trong thực tế, có nhiều trường hợp cần truy cập dữ liệu được cất ở RAM ngoài hoặc ở không gian ROM trên chip. Trong những trường hợp đó chúng ta cần sử dụng thanh ghi 16 bit là DPTR.

Chế độ định địa chỉ chỉ số và truy cập bộ nhớ ROM trên chip

Chế độ định địa chỉ chỉ số được sử dụng rộng rãi khi truy cập các phần tử dữ liệu của bảng trong không gian ROM chương trình của 8051. Lệnh được dùng cho mục đích này là "MOVC A, @ A + DPTR". Thanh ghi 16 bit DPTR và thanh ghi A được dùng để tạo ra địa chỉ của phần tử dữ liệu được lưu trong ROM trên chip. Do các phần tử dữ liệu được cất trong không gian ROM chương trình trên chip 8051, nên cần dùng lệnh MOVC thay cho lệnh MOV (chữ "C" ở cuối lệnh có nghĩa là mã lệnh "Code"). Ở lệnh này, nội dung của A được cộng với nội dung thanh ghi 16 bit DPTR để tạo ra địa chỉ 16 bit. Xét ví dụ sau.

Ví dụ 5.6

Giả sử chữ "NAM" được lưu trong ROM có địa chỉ bắt đầu tại 200H và chương trình được ghi vào ROM bắt đầu từ địa chỉ 0. Hãy phân tích cách chương trình hoạt động và xác định xem chữ "NAM" sau khi chạy chương trình này được cất vào đâu?

Giải:

```
MYDATA:
 DB    0000H          ;Bắt đầu đốt ROM tại địa chỉ 00H
 MOV   DPTR,#200H     ;Địa chỉ bảng tham chiếu DPTR=200H
 CLA   A              ;Xoá thanh ghi A (A=0)
 MOVC  A,@A+DPTR      ;Lấy ký tự không gian nhớ chương trình
 MOV   R0,A           ;Cất vào trong R0
 INC   DPTR           ;DPTR=201, trỏ đến ký tự kế tiếp
 CLR   A              ;Xoá thanh ghi A
 MOVC  A,@A+DPTR      ;Lấy ký tự kế tiếp
 MOV   R1,A           ;Cất vào R1
 INC   DPTR           ;DPTR=202 con trỏ trỏ đến ký tự sau đó
 CLR   A              ;Xoá thanh ghi A
 MOVC  A@A+DPTR       ;Nhận ký tự kế tiếp
 MOV   R2,A           ;Cất vào R2
HERE: SJMP   HERE     ;Dừng lại ở đây.
;Dữ liệu được đốt ở không gian mã lệnh tại địa chỉ  200H
       ORG 200H
       END            ;Kết thúc chương trình
```

Ở chương trình trên, các ngăn nhớ ROM chương trình 200H - 2002H có các nội dung sau:

200 = ('N'); 201= ('A') và 202 = ('M')

Chúng ta bắt đầu với DPTR = 200H và A = 0. Lệnh "MOVC A, @ A + DPTR chuyển nội dung của ô nhớ 200H trong ROM (200H + 0 = 200H) vào A. Thanh ghi A chứa giá trị 4EH là mã ASC của ký tự "N". Ký tự này được cất vào R0. Kế đó, DPTR được tăng lên tạo thành DPTR = 201H. A lại được xoá về 0 để lấy nội dung của vị trí nhớ kế tiếp trong ROM là 201H chứa ký tự "A". Sau khi chương trình này chạy ta có R0 = 4EH, R1 = 41H và R2 = 4FH là các mã ASCII của các ký tự "N", "A" và "M".

Ví dụ 5.7

Giả sử không gian ROM bắt đầu từ địa chỉ 250H có từ "Viet Nam", hãy viết chương trình để chuyển các byte vào các vị trí ngăn nhớ RAM bắt đầu từ địa chỉ 40H.

Giải:

```
;(a)  Phương pháp sử dụng một bộ đếm
  ORG   000
  MOV   DPTR,#MYDATA    ;Nạp con trỏ ROM
  MOV   R0,#40H         ;Nạp con trỏ RAM
  MOV   R2,#7           ;Nạp bộ đếm
BACK:   CLR A           ;Xoá thanh ghi A
  MOVC  A,@A+DPTR       ;Chuyển dữ liệu từ không gian mã
  MOV   R0,A            ;Cất vào ngăn nhớ RAM
  INC   DPTR            ;Tăng con trỏ ROM
  INC   R0              ;Tăng con trỏ  RAM
  DJNZ  R2,BACK         ;Lặp lại cho đếnkhi bộ đếm=0
HERE:   SJM HERE
;không gian mã của ROM trên chip dùng để cất dữ liệu
  ORG   250H
MYDATA:DB  "Viet Nam"
  END
;(b) phương pháp sử dụng ký tự null để kết thúc chuỗi
  ORG   000
  MOV   DPTR,#MYDATA    ;Nạp con trỏ ROM
  MOV   R0,#40          ;Nạp con trỏ RAM
BACK:   CLR    A S      ;Xoá thanh ghi A(A=0)
  MOVC  A,@A+DPTR       ;Chuyển dữ liệu từ không gian mã
  JZ    HERE            ;Thoát ra nếu có ký tự Null
  MOV   DPTR,#MYDATA    ;Cất vào ngăn nhớ của RAM
  INC   @R0,A           ;Tăng con trỏ ROM
   INC  R0              ;Tăng con trỏ RAM
   SJM  BACK            ;Lặp lại
```

```
HERE:   SJMP HERE
;Không gian mã của ROM trên chip dùng để cất dữ liệu
   ORG  250H
MYADTA:DB "Viet Nam",0;Ký tự Null để kết thúc chuỗi END
   END
```

Lưu ý đến cách ta sử dụng lệnh JZ để phát hiện ký tự trống, 0 và kết thúc chuỗi.

Bảng tham chiếu và sử dụng chế độ định địa chỉ chỉ số

Khái niệm bảng tham chiếu được dùng rộng rãi trong lập trình các bộ vi xử lý. Bảng cho phép truy cập các phần tử của một bảng thường xuyên được sử dụng với số thao tác ít nhất. Ví dụ, giả sử có một ứng dụng cần x^2 giá trị trong phạm vi 0 đến 9. Ta có thể sử dụng bảng tham chiếu thay cho việc tính toán. Hãy xem ví dụ sau.

Ví dụ 5.8

Hãy viết một chương trình để lấy x giá trị cổng P1 và liên tục gửi giá trị x2 tới cổng P2.

Giải:

```
       ORG    000
       MOV    DPTR,#300 H  ;Nạp địa chỉ bảng xắp xếp
       MOV    A,#0FFH      ;Nạp A giá trị FFH
       MOV    P1,A         ;Đặt cổng P1 là đầu vào
BACK:  MOVA,  P1           ;Lấy giá trị X từ P1
       MOVC   A,@A+DPTR    ;Lấy giá trị X từ bảng
       MOV    P2, A        ;Xuất nó ra cổng P2
       SJMP   BACK         ;Lặp lại
       ORG    300H
XSQR-TABLE:
       DB     0,1,4,9,16,25,36,49,64,81
       END
```

Lưu ý lệnh đầu tiên có thể thay bằng lệnh "`MOV DPTR,#XSQR-TABLE`".

Ví dụ 5.9

Từ hình 4.8 hãy trả lời các câu hỏi sau:

a) Xác định nội dung các vị trí 300 - 309H của ROM.

b) Vị trí nào của ROM chứa 6^2 và giá trị đó là bao nhiêu?

c) Giả sử P1 có giá trị là 9 thì giá trị P2 là bao nhiêu (ở dạng nhị phân)?

Giải:

a) Giá trị ở các ngăn nhớ 300H-309H của ROM là:

```
300=(00)    301=(01)    302=(04)    303=(09)
304=(10)    4×4=16=10 in hexa
305=(19)    5×5=25=19 in hexa
30 =(24)    6×6=36=24H
307=(31)    308=(40)                309=(51)
```

b) 306H và giá trị là 24H.

c) 01010001B là giá trị nhị phân của 51H và 81D (9^2=81).

Ngoài việc sử dụng DPTR để truy cập không gian bộ nhớ ROM chương trình thì nó còn có thể được dùng để truy cập bộ nhớ ngoài của 8051, xem chương 14.

Một thanh ghi khác nữa được sử dụng ở chế độ định địa chỉ chỉ số là bộ đếm chương trình, xem phụ lục A.

Trong nhiều ví dụ nêu trên đây, chúng ta thường dùng lệnh khá quen thuộc là lệnh MOV, tuy nhiên cũng có thể sử dụng bất kỳ lệnh nào khác nếu vẫn hỗ trợ cho chế độ định địa chỉ. Ví dụ, lệnh "ADD A, @ " sẽ cộng nội dung ngăn nhớ do R0 trỏ 00 đến vào nội dung của thanh ghi A.

Chương 6
LỆNH SỐ HỌC

6.1 CỘNG VÀ TRỪ CÁC SỐ KHÔNG DẤU

Số không dấu được định nghĩa là dữ liệu mà tất cả mọi bit đều được dùng để biểu diễn giá trị và không có bit dành cho dấu âm hoặc dương. Như vậy, đối với dữ liệu 8 bit, toán hạng chỉ có thể nằm giữa 00 và FFH (0 đến 255 hệ thập phân).

Cộng các số không dấu

Ở 8051, để cộng hai toán hạng cần sử dụng thanh ghi A. Cú pháp lệnh cộng ADD có dạng sau:

```
ADD A,nguồn      ;A = A + nguồn
```

Lệnh ADD được dùng để cộng hai toán hạng. Toán hạng đích luôn là thanh ghi A, còn toán hạng nguồn có thể là thanh ghi, dữ liệu tức thời hoặc bộ nhớ. Cần nhớ rằng, đối với hợp ngữ thì phép toán số học giữa hai dữ liệu bộ nhớ với nhau là không hợp lệ. Lệnh cộng có thể làm thay đổi các bit AF, CF hoặc PF của thanh ghi cờ. Tác động của lệnh ADD lên cờ tràn sẽ được tìm hiểu ở mục 6.3. Chúng ta xem xét ví dụ 6.1 dưới đây:

Ví dụ 6.1

Hãy làm rõ các lệnh sau ảnh hưởng lên cờ nào?

```
MOV    A,#0F5H        ; A=F5H
ADD    A,#0BH         ; A=F5+0B=00
```

Giải:

```
      F5H              1111  0101
+     0BH        +     0000  1011
      100H             0000  0000
```

Sau phép cộng, thanh ghi A chứa giá trị 00 và trạng thái các cờ sẽ như sau:
CY = 1 vì có phép nhớ từ D7.
PF = 1 vì số các số 1 là 0 (một số chẵn) cờ PF được thiết lập lên 1.
AC =1 vì có nhớ từ bit D3 sang D4.

Cộng các byte

Ở chương 2 đã trình bày phép cộng 5 byte dữ liệu. Giá trị tổng được giữ lại nhỏ hơn FFH tức nhỏ hơn giá trị cực đại 8 bit. Để tính tổng các toán hạng thì cờ nhớ phải được kiểm tra sau mỗi lần cộng một toán hạng. Ví dụ sau dùng R7 để tích luỹ giá trị nhớ mỗi khi toán hạng được cộng vào A.

Ví dụ 6.2

Giả sử các ngăn nhớ 40 - 44 của RAM có giá trị ở dạng Hexa như sau: 40 = (7D); 41 = (EB); 42 = (C5); 43 = (5B) và 44 = (30). Hãy viết chương trình tính tổng các giá trị trên. Kết quả byte thấp cất ở thanh ghi A và byte cao cất ở R7 (các giá trị trên đều được cho ở dạng Hexa).

Giải:

```
        MOV    R0,#40H       ;Nạp con trỏ
        MOV    R2,#5         ;Nạp bộ đệm
        CLR    A             ;Xoá thanh ghi A
        MOV    R7,A          ;Xoá thanh ghi R7
AGAIN:  ADD    A,@R0         ;Cộng byte con trỏ đến theo R0
        JNC    NEXT          ;Nếu CY=0 không tích luỹ cờ nhớ
        INC    R7            ;Bám theo số lần nhớ
NEXT:   INC    R0            ;Tăng con trỏ
        DJNZ   R2,AGAIN      ;Lặp lại cho đến khi R0=0
```

Phân tích ví dụ 6.2

Chúng ta sẽ bàn kỹ hơn ba lần lặp lại của vòng lặp. Phần còn lại dành cho người đọc tự kiểm tra.

1. Ở lần lặp đầu tiên, 7DH được cộng vào A với CY = 0 và R7 = 00 và bộ đếm R2 = 04.

2. Ở lần lặp lại thứ hai, EBH được cộng vào A và kết quả trong A là 68H còn CY = 1. Vì cờ nhớ xuất hiện, R7 được tăng lên. Lúc này bộ đếm R2 = 03.

3. Ở lần lặp lại thứ ba, C5H được cộng vào A nên A = 2DH và cờ nhớ lại tiếp tục nhớ, do vậy R7 lại được tăng lên và bộ đếm R2 = 02.

Ở phần cuối, khi vòng lặp kết thúc, giá trị tổng được lưu ở thanh ghi A và R7, trong đó A chứa byte thấp và R7 chứa byte cao.

Cộng có nhớ và cộng các số 16 bit

Khi cộng hai toán hạng 16 bit thì phải lưu ý đến giá trị nhớ từ byte thấp đến byte cao. Trong những trường hợp đó cần dùng lệnh ADDC (cộng có nhớ). Ví dụ, cộng hai số sau: 3CE7H + 3B8DH.

```
    3C   E7
+   3B   8D
   ------
    78   74
```

Khi cộng byte thứ nhất (E7 + 8D = 74, CY = 1), cờ nhớ được truyền lên byte cao tạo ra kết quả 3C + 3B + 1 = 78. Dưới đây là chương trình thực hiện các bước nêu trên.

Ví dụ 6.3

Hãy viết chương trình cộng hai số 16 bit là 3CE7H và 3B8DH. Cất kết quả vào R7 (byte cao) và R6 (byte thấp).

Giải:

```
CLR             ;Xoá cờ CY=0
MOV   A,#0E7H   ;Nạp byte thấp vào A → A=E7H
ADD   A,#8DH    ;Cộng byte thấp vào A → a=74H và CY=1
MOV   R6,A      ;Lưu byte thấp của tổng vào R6
MOV   A,#3CH    ;Nạp byte cao vào A → A=3CH
ADDC  A,#3BG    ;Cộng byte cao có nhớ vào A → A=78H
MOV   R7,A      ;Lưu byte cao của tổng vào R7
```

Số BCD

Số BCD là số thập phân được mã hoá theo nhị phân (Binary Coded Decimal). Đây là dạng số rất quen thuộc bởi vì trong thực tế chúng ta thường sử dụng các số từ 0 đến 9 chứ không phải số nhị phân hay số Hexa. Biểu diễn nhị phân của các số từ 0 đến 9 được gọi là BCD, xem hình 6.1. Trong các tài liệu về máy tính thường có hai khái niệm về các số BCD là: BCD dạng nén và không nén.

Số	*BCD*
1	0000
2	0001
3	0010
4	0011
5	0100
6	0101
7	0111
8	1000
9	1001

Hình 6.1. Mã BCD

Số BCD dạng không nén

Số BCD không nén có 4 bit thấp biểu diễn số BCD còn 4 bit cao bằng 0. Ví dụ "00001001" và "0000

0101" là những số BCD không nén của số 9 và số 5. Để biểu diễn số BCD không nén cần dùng một byte bộ nhớ hay một thanh ghi 8 bit.

Số BCD nén

Đối với số BCD nén thì một byte chứa được 2 số BCD, một số ở 4 bit thấp và số thứ 2 nằm ở 4 bit cao. Ví dụ "0101 1001" là số BCD nén của số 59H. Như vậy chỉ cần 1 byte bộ nhớ là có thể lưu được hai toán hạng BCD. Đây là lý do tại sao sử dụng số BCD nén thì hiệu quả lưu dữ liệu tăng gấp đôi.

Có một vấn đề khi cộng các số BCD cần được khắc phục. Đó là sau khi cộng các số BCD nén thì kết quả không còn là số BCD nữa. Ví dụ:

```
MOV     A,#17H
ADD     A,#28H
```

Cộng hai số này cho kết quả là 0011 1111B (3FH) không còn là số BCD. Một số BCD chỉ nằm trong giải 0000 đến 1001 (từ số 0 đến số 9), nói cách khác, cộng hai số BCD phải cho kết quả là số BCD. Kết quả trên đáng lẽ phải là 17 + 28 = 45 (0100 0101). Để giải quyết vấn đề này, lập trình viên phải cộng 6 (0110) vào số thấp 3F + 06 = 45H. Trường hợp tương tự cũng có thể xảy ra ở số cao, ví dụ, khi cộng hai số 52H + 87H = D94. Tương tự, để khắc phục vấn đề này cần cộng thêm 6 vào số cao (D9H + 60H = 139). Tình trạng này rất hay xảy ra và mọi bộ xử lý đều có một lệnh chuyên để xử lý vấn đề này. Ở 8051, đó là lệnh "DAA".

Lệnh DA

Lệnh điều chỉnh thập phân của phép cộng DA (Decimal Adjust for Addition) ở 8051 dùng để hiệu chỉnh sự sai lệch như trình bày trên đây khi cộng các số BCD. Lệnh DA sẽ cộng 6 vào 4 bit thấp hoặc 4 bit cao khi cần. Còn bình thường lệnh để nguyên kết quả tìm được. Hãy xem ví dụ sau.

```
MOV     A,#47H      ;A=47H là toán hạng BCD đầu tiên
MOV     B,#25H      ;B=25H là toán hạng BCD thứ hai
ADD     A,B         ;Cộng các số hexa(nhị phân)A=6CH
DA      A           ;Điều chỉnh phép cộng BCD(A=72H)
```

Sau khi chương trình được thực hiện, thanh ghi A sẽ chứa 72h (47 + 25 = 72). Lệnh "DA" chỉ làm việc với thanh ghi A. Cũng cần nhấn mạnh rằng, lệnh DA sẽ được sử dụng sau phép cộng các số BCD và các số BCD không thể lớn hơn 9. Như vậy, ở số BCD các số A - F là không được phép. Còn một lưu ý nữa là DA được dùng sau phép cộng ADD, mà không bao giờ dùng sau lệnh tăng INC.

Tóm tắt về hoạt động của lệnh DA

Sau lệnh ADD hoặc ADDC,

1. Nếu 4 bit thấp lớn hơn 9 hoặc cờ AC = 1 thì lệnh sẽ cộng 0110 vào 4 bit thấp.
2. Nếu 4 bit cao lớn hơn 9 hoặc cờ CY = 1 thì lệnh sẽ cộng 0110 vào 4 bit cao.

Trong thực tế, cờ AC chỉ dùng để phục vụ cho phép cộng và hiệu chỉnh các số BCD. Ví dụ, cộng 29H với 18H cho kết quả 41H là sai. Để sửa lại, lệnh DA sẽ cộng 6 vào 4 bit thấp (vì AC = 1).

	Hexa		***BCD***		
	29H		0010	1001	
+	18H	+	0001	1000	
	41H		0100	0001	AC=1
+	6	+		0110	
	47H		0100	0111	

Ví dụ 6.4

Giả sử 5 dữ liệu BCD được lưu trong RAM tại địa chỉ bắt đầu từ 40H như sau: 40 = (71), 41 = (11), 42 = (65), 43 = (59) và 44 = (37). Hãy viết chương trình tính tổng của tất cả 5 số trên và kết quả phải là dạng BCD.

Giải:

```
        MOV   R0,#40H       ;Nạp con trỏ
        MOV   R2,#5         ;Nạp bộ đếm
        CLR   A             ;Xoá thanh ghi A
        MOV   R7,A          ;Xoá thanhg ghi R7
AGAIN:  ADD   A,@R0         ;Cộng byte con trỏ chỉ bởi R0
        DA    A             ;Điều chỉnh về dạng BCD đúng
        JNC   NEXT          ;Nếu CY=0 không tích luỹ cờ nhớ
        JNC   R7            ;Tăng R7 bám theo số lần nhớ
NEXT:   INC   R0            ;Dịch con trỏ lên ô nhớ kế tiếp
        DJNZ  R2,AGAIN      ;Lặp lại cho đến khi R2=0
```

Trừ các số không dấu

Cú pháp

```
SUBB A,nguồn      ;A=A-nguồn-CY
```

Ở các bộ xử lý thông thường có hai lệnh thực hiện phép tính trừ, đó là phép trừ SUB và trừ có mượn SUBB (Substract With Borrow). Đối với 8051 thì chỉ có

một lệnh SUBB duy nhất. Để có thể thực hiện phép tính trừ SUB cần phân biệt hai trường hợp của lệnh SUBB là CY = 0 và CY = 1. Lưu ý ở đây ta dùng cờ CY để mượn.

Lệnh SUBB khi CY=0

Để thực hiện phép trừ, tất các bộ vi xử lý đều dùng số bù 2. Bộ 8051 được thiết kế một mạch cộng để thực hiện lệnh trừ. Giả sử 8051 sử dụng mạch cộng để thực hiện lệnh trừ và CY = 0 trước khi thực hiện lệnh thì tóm tắt các bước CPU tiến hành lệnh SUBB với trường hợp các số không dấu như sau:

1. Lấy bù 2 của số trừ (toán hạng nguồn)
2. Cộng số vừa tìm được với số bị trừ (A)
3. Đảo có nhớ

Kết thúc ba bước sẽ cho kết quả của phép trừ và các cờ cũng được thiết lập. Chúng ta xem ví dụ minh hoạ sau:

Ví dụ 6.5

Làm rõ các bước thực hiện trong chương trình sau:

```
CLR    C              ;Tạo CY=0
MOV    A,#3FH         ;Nạp 3FH vào A (A=3FH)
MOV    R3,#23H        ;Nạp 23H vào R3 (R3=23H)
SUBB   A,R3           ;Trừ A cho R3 đặt kết quả vào A
```

Giải:

```
A=  3F    0011  1111      0011  1111
R3= 23    0010  0011  +   1101  1101   bù2 (bước 1)
    1C                  1 0001  1100 - 1C  (bước 2)
                        0  CF=0            (bước 3)
```

Các cờ sẽ được thiết lập như sau: CY = 0, AC = 0 và lập trình viên phải biết được trạng thái các cờ nhớ để xác định kết quả là âm hay dương.

Sau khi thực hiện lệnh SUBB, nếu CY = 0 thì kết quả là số dương, còn nếu CY = 1 thì kết quả là số âm và đích có giá trị bù 2 của kết quả. Kết quả thường được để ở dạng bù 2, song cũng có thể sử dụng lệnh lấy bù CPL và tăng INC để chuyển kết quả về dạng thông thường. Lệnh CPL tiến hành lấy bù 1, sau đó kết quả được tăng lên 1 (INC) và trở thành dạng bù 2. Xem ví dụ 6.6.

Ví dụ 6.6

Phân tích chương trình sau:

```
        CLR    C
        MOV    A,#4CH          ;Nạp A giá trị 4CH(A=4CH)
        SUBB   A,#6EH          ;Trừ A cho 6EH
        JNC    NEXT            ;Nếu CY=0 nhảy đến đích NEXT
        CPL    A               ;Nếu CY=1 thực hiện bù 1
        INC    A               ;Tăng 1 để có bù 2
NEXT:   MOV    R1,A            ;Lưu A vào R1
```

Giải:

Các bước thực hiện lệnh "SUBB A, 6EH" như sau:

```
   4C    0100 1100                  0100 1100
 - 6E    0110 1110 → lấy bù 2 =     1001 0010   (bước 1)
 - 22                             0 1101 1110   (bước 2)
```

Cờ CY=1, kết quả là số âm ở dạng bù 2.

Lệnh SUBB khi CY = 1

Lệnh này được dùng khi trừ các số nhiều byte và có mượn. Nếu CY = 1 trước khi thực hiện lệnh SUBB thì kết quả cũng sẽ được trừ đi 1. Xem ví dụ sau đây.

Ví dụ 6.7

Phân tích chương trình sau:

```
CLR    C               ;CY=0
MOV    A,#62           ;A=62H
SUB    BA,#96H         ;62H-96H=CCH with CY=1
MOV    R7,A            ;Save the result
MOV    A,#27H          ;A=27H
SUB    BA,#12H         ;27H-12H-1=14H
MOV    R6,A            ;Save the result
```

Giải:

Sau khi SUBB, A = 62H - 96H = CCH và cờ nhớ được lập báo rằng có mượn. Vì CY = 1 nên khi SUBB được thực hiện lần thứ 2 thì a = 27H - 12H - 1= 14H. Do vậy, ta có kết quả 2762H - 1296H = 14CCH.

6.2 NHÂN VÀ CHIA CÁC SỐ KHÔNG DẤU

Khi nhân và chia hai số ở 8051 cần phải sử dụng hai thanh ghi A và B vì các lệnh nhân và chia chỉ làm việc với những thanh ghi này.

Nhân hai số không dấu

Bộ vi điều khiển chỉ hỗ trợ phép nhân byte với byte. Các byte được giả thiết là dữ liệu không dấu. Cấu trúc lệnh như sau:

```
MUL  AB   ;A×B và kết quả 16 bit được đặt trong A và B
```

Khi nhân byte với byte thì một toán hạng ở thanh ghi A còn toán hạng kia ở thanh ghi B. Kết quả của phép nhân được lưu ở A và B, trong đó byte thấp ở A, còn byte cao ở B. Ví dụ dưới đây trình bày phép nhân 25H với 65H. Kết quả 16 bit được đặt ở A và B.

```
MOV    A,#25H     ;Nạp vào A giá trị 25H
MOV    B,65H      ;Nạp vào B giá trị 65H
MUL    AB         ;25H*65H=E99 với B=0EH và A=99H
```

Bảng 6.1. Tóm tắt phép nhân hai số không dấu (MULAB)

Nhân	Toán hạng 1	Toán hạng 2	Kết quả
Byte*Byte	A	B	A = byte thấp, B = byte cao

Chia hai số không dấu

8051 cũng chỉ hỗ trợ phép chia hai số không dấu byte cho byte. Cú pháp như sau:

```
DIV AB    ;Chia A cho B
```

Khi chia một byte cho một byte thì tử số (số bị chia) phải ở trong thanh ghi A còn mẫu số (số chia) phải ở trong thanh ghi B. Kết quả của phép chia DIV được đặt trong A, còn số dư được đặt trong B. Xét ví dụ dưới đây:

```
MOV    A,#95      ;Nạp số bị chia vào A=95
MOV    B,#10      ;Nạp số chia vào B=10
DIV    AB         ;A=09(thương số);B=05(số dư)
```

Đối với lệnh "DIV AB" cần lưu ý các điểm sau:

1. Lệnh này luôn xoá các cờ CY = 0 và OV = 0 nếu mẫu số khác 0.
2. Nếu mẫu số bằng 0 (B = 0) thì OV =1 báo lỗi và CY = 0.

Thực tế, ở tất cả mọi bộ vi xử lý khi chia một số cho 0 thì bộ vi xử lý sẽ báo kết quả không xác định. Đối với 8051 đó là cờ OV được thiết lập lên 1.

Bảng 6.2. Tóm tắt phép chia không dấu (DIV AB)

Phép chia	Tử số	Mẫu số	Thương số	Số dư
Byte cho Byte	A	B	A	B

Ví dụ ứng dụng lệnh chia

Trong thực tế, chúng ta thường gặp một hệ đo, trong đó bộ chuyển đổi tín hiệu tương tự sang tín hiệu số ADC được nối tới một cổng của 8051. Đầu vào của ADC là giá trị nhiệt độ hay áp suất, còn đầu ra là dữ liệu 8 bit ở dạng Hexa trong dải 00 - FFH. Dữ liệu Hexa này cần được chuyển đổi về dạng thập phân. Để thực hiện chuyển đổi, dữ liệu được chia lặp nhiều lần cho 10 và lưu lại số dư, như trình bày ở ví dụ sau.

Ví dụ 6.8

a) Viết chương trình để nhận dữ liệu dạng Hexa trong phạm vi 00 - FFH từ cổng 1 và chuyển đổi về dạng thập phân. Lưu các số vào các thanh ghi R7, R6 và R5 trong đó số có nghĩa nhỏ nhất là R7.

b) Phân tích chương trình với giả thiết P1 có giá trị dữ liệu FDH.

Giải:

a)

```
MOV    A,#0FFH
MOV    P1,A          ;Tạo P1 là cổng đầu vào
MOV    A,P1          ;Đọc dữ liệu từ P1
MOV    B,#10         ;B=0A Hexa(0 thập phân)
DIV    AB            ;Chia cho 10
MOV    R7,B          ;Cất số thấp
MOV    B,#10
DIV    AB            ;Chia 10 lần nữa
MOV    R6,B          ;Cất số tiếp theo
MOV    R5,A          ;Cất số cuối cùng
```

b) Để chuyển đổi số nhị phân hay Hexa về số thập phân ta thực hiện chia lặp cho 10 liên tục cho đến khi thương số nhỏ hơn 10. Sau mỗi lần chia, cần lưu lại số dư. Trường hợp số nhị phân 8 bit FDH (253D) trình tự tiến hành như sau:

	Thương số	Số dư	
FD/0A	19	3	Số thấp
19/0A	2	5	Số giữa
		2	Số cao

Do vậy, ta có FDH = 253D. Để hiển thị dữ liệu này thì cần phải chuyển về mã ASCII và sẽ được đề cập đến ở chương sau.

6.3 SỐ CÓ DẤU VÀ CÁC PHÉP TÍNH SỐ HỌC

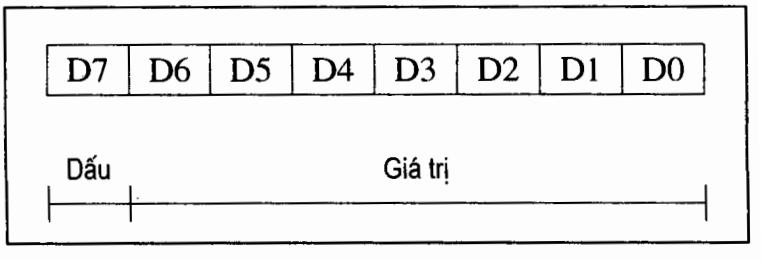
Tất cả những dữ liệu được xét cho đến bây giờ đều là số không dấu, có nghĩa là toàn bộ 8 bit đều được dùng biểu diễn độ lớn của dữ liệu. Có nhiều ứng dụng yêu cầu dữ liệu có dấu. Ở phần này chúng ta sẽ tìm hiểu các lệnh liên quan đến các số có dấu.

Khái niệm số có dấu trong máy tính

Các số được dùng trong thực tế có thể là âm hoặc dương. Ví dụ 5 độ dưới 0^0C được biểu diễn là -5^0C và 20 độ trên 0^0C được biểu diễn là $+20^0$C. Máy tính cũng cần đáp ứng được yêu cầu này. Cách biểu diễn số âm và dương có dấu trong máy tính như sau: Bit cao nhất MSB được dùng cho bit dấu (+) hoặc (-), các bit còn lại được dùng biểu diễn độ lớn. Bit dấu có giá trị 0 đối với các số dương và 1 đối với các số âm. Một byte có dấu được biểu diễn như trên hình 6.2.

Toán hạng 8 bit có dấu

Ở toán hạng dạng byte có dấu thì bit cao nhất MSB là D7 được dùng để biểu diễn dấu, 7 bit còn lại từ D6 - D0 dùng để biểu diễn độ lớn của số đó. Nếu D7 = 0 thì đó là toán hạng dương và nếu D7 = 1 thì đó là toán hạng âm.



Hình 6.2. Toán hạng 8 bit có dấu

Số dương có dấu

Dải giá trị của số dương được biểu diễn như trên hình 6.2 là từ 0 đến +127. Nếu muốn biểu diễn số lớn hơn thì cần sử dụng toán hạng 16 bit. Vì 8051 không hỗ trợ dữ liệu 16 bit nên ta không bàn đến ở đây.

Số âm có dấu

Đối với số âm thì D7 = 1, còn độ lớn được biểu diễn ở dạng số bù 2. Nhiệm vụ chuyển đổi sang số bù 2 do trình hợp dịch đảm nhiệm, song đối với chúng ta điều quan trọng là phải hiểu được việc chuyển đổi đó diễn ra như thế nào. Để chuyển đổi về dạng số âm (số bù 2), các bước tiến hành như sau:

1. Viết độ lớn của số ở dạng nhị phân 8 bit (không dấu).
2. Đảo ngược tất cả các bit.
3. Cộng số vừa xác định với 1.

Ví dụ 6.9

Trình bày cách 8051 biểu diễn số - 5.

Giải:

Các bước thực hiện như sau:

```
1. 0000 0101      Biểu diễn số 5 ở dạng 8 bit nhị phân
2. 1111 1010      Đảo các bit
3. 1111 1011      Cộng 1(thành số FB ở dạng Hexa)
```

Do vậy, số FBH là biểu diễn số có dấu dạng bù 2 của số - 5.

Ví dụ 6.10

Trình bày cách 8051 biểu diễn số - 34H.

Giải:

Hãy quan sát các bước sau:

```
1. 0011 0200      Số 34 được cho ở dạng nhị phân
2. 1100 1011      Đảo các bit
3. 1100 1100      Cộng 1(thành số CC ở dạng Hexa)
```

Vậy số CCH là biểu diễn dạng bù 2 có dấu của số - 34H.

Ví dụ 6.11

Trình bày cách 8051 biểu diễn số - 128:

Giải:

Quan sát các bước sau:

```
1. 1000 0000      Số 128 ở dạng nhị phân 8 bit
2. 0111 1111      Đảo bit
3. 1000 0000      Cộng 1(trở thành số 80 dạng Hexa)
```

Vậy - 128 = 80H là biểu diễn số có dấu dạng bù 2 của - 128.

Từ các ví dụ trên đây có thể dễ dàng thấy rằng, dải của các số âm có dấu 8 bit là từ - 1 đến - 128. Dưới đây là liệt kê các số có dấu 8 bit:

Số thập phân	*Số nhị phân*	*Số Hexa*
-128	1000 0000	80
-127	1000 0001	81
-126	1000 0010	82
...	....	...
-2	1111 1110	FE
-1	1111 1111	FF
0	0000 0000	00
+1	0000 0001	01
+2	0000 0010	02
...	....	...
-127	0111 1111	FE

Tràn ở các phép toán số có dấu

Khi sử dụng các số có dấu xuất hiện một vấn đề buộc phải xử lý, đó là *tràn*. Khi tràn, 8051 chỉ báo có lỗi bằng cách thiết lập cờ tràn OV, nhưng để tránh nhầm lẫn, lập trình viên cần hiểu rõ hơn về trường hợp này. Vậy tràn là gì? Nếu kết quả của một phép toán trên các số có dấu mà lớn quá kích thước thanh ghi thì xuất hiện tràn và lập trình viên cần được cảnh báo. Xét ví dụ 6.12 dưới đây.

Ví dụ 6.12

Khảo sát đoạn chương trình sau và phân tích kết quả.

```
MOV   A,#+96    ;A=0110  0000(A=60H)
MOV   R1,#+70   ;R1=0100 0110(R1=46H)
ADD   A, R1     ;A=1010  0110=A6H=-90D  Sai !!!
```

Giải:

```
      +96   0110  0000
  +   +70   0100  0110
  +   166   1010  0110   và OV=1
```

CPU cho kết quả là -90 và đó là kết quả sai. CPU bật cờ OV = 1 để báo tràn số.

Ở ví dụ 6.12 thì +96 được cộng với +70 và kết quả CPU tính được là -90. Tại sao như vậy? Lý do là kết quả của (+96) + (+70) = 172 vượt quá kích thước thanh

ghi A. Thanh ghi A có 8 bit và chỉ chứa được số lớn nhất là + 127. Vì thế, cờ tràn OV được thiết kế nhằm mục đích báo cho lập trình viên biết kết quả của phép toán số có dấu vừa tính là sai.

Thiết lập cờ tràn OV

Ở các phép toán với số có dấu 8 bit thì cờ OV được bật lên 1 khi xuất hiện một trong hai điều kiện sau:

1. Có nhớ từ D6 sang D7 nhưng không có nhớ ra từ D7 (cờ CY = 0).
2. Có nhớ từ D7 (cờ CY = 1) nhưng không có nhớ từ D6 sang D7.

Nói cách khác, cờ tràn OV được bật lên 1 nếu có nhớ từ D6 sang D7 hoặc từ D7 nhưng không đồng thời xảy ra cả hai. Như vậy, nếu có nhớ cả từ D6 sang D7 và từ D7 ra thì cờ OV = 0. Ở ví dụ 6.12 vì chỉ có nhớ từ D7 ra nên cờ OV = 1. Ví dụ 6.13, ví dụ 6.14 và 6.15 minh hoạ thêm về sử dụng cờ tràn trong các phép tính số học với số có dấu.

Ví dụ 6.13

Hãy quan sát đoạn chương trình sau và để ý đến vai trò của cờ OV.

```
MOV    A,#-128    ;A=1000   0000(A=80H)
MOV    R4,#-2     ;R4=1111  (R4=FEH)
ADD    A,R4       ;A=0111   1110(A=7EH=+126,invalid)
```

Giải:

```
   -128      1000   0000
+   - 2      1111   1110
  -130      0111   1110   và OV = 1
```

CPU cho kết quả + 126 là sai (cờ OV = 1).

Ví dụ 6.14

Hãy quan sát đoạn chương trình sau và lưu ý cờ OV.

```
MOV    A,#-2      ;A=1111   1110(A=FEH)
MOV    R1,#-5     ;R1=1111  1011(R1=FBH)
ADD    A,R1       ;A=1111   1001(A=F9H=-7,orrect,OV=0)
```

Giải:

```
    -2       1111   1110
+  -5        1111   1011
    -7       1111   1001   và OV=0
```

CPU cho kết quả - 7 là đúng (cờ OV = 0).

Ví dụ 6.15

Xác định vai trò của cờ OV qua đoạn chương trình sau.

```
MOV    A,#+7      ;A= 0000  0111(A=07H)
MOV    R1,#+18    ;R1=0001  0010(R1=12H)
ADD    A,R1       ;A=1111   1001(A=19H=-25,đúng,OV=0)
```

Giải:

CPU cho kết quả - 25 là đúng (cờ OV = 0).

```
    7     0000   0111
+  18     0001   0010
   25     0001   1001   và OV=0
```

Từ các ví dụ trên đây chúng ta có thể kết luận rằng, trong mọi phép cộng số có dấu, cờ OV luôn luôn báo kết quả là đúng hoặc sai. Nếu cờ OV = 1 thì kết quả là sai, còn nếu OV = 0 thì kết quả là đúng. Cũng cần nhấn mạnh lại rằng, ở phép cộng các số không dấu, lập trình viên cần theo dõi trạng thái cờ nhớ CY, còn ở phép cộng các số có dấu thì đó là cờ tràn OV. Đối với 8051, các lệnh như JNC và JC cho phép chương trình rẽ nhánh ngay sau phép cộng các số không dấu, như trình bày ở mục 6.1, song ở trường hợp cờ tràn OV thì không như vậy. Tuy nhiên, vẫn có thể thực hiện được điều này bằng lệnh "JB PSW.2" hoặc "JNB PSW.2" vì PSW là thanh ghi cờ có thể định địa chỉ bit.

Chương 7
LỆNH LOGIC

7.1 LỆNH LOGIC VÀ SO SÁNH

Lệnh AND (VÀ)

Cú pháp

```
ANL đích,nguồn      ;đích=đích AND nguồn
```

Lệnh thực hiện phép AND logic trên hai toán hạng đích và nguồn và đặt kết quả vào đích. Đích thường là thanh ghi tích luỹ. Toán hạng nguồn có thể là thanh ghi, bộ nhớ hoặc giá trị cho sẵn. Xem phụ lục A1 để biết thêm về các chế độ định địa chỉ của lệnh này. ANL là lệnh với toán hạng dạng byte và không có tác động lên các cờ. Lệnh này thường được dùng để che (xoá về 0) một số bit của toán hạng. Xem ví dụ 7.1.

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

Ví dụ 7.1

Xác định kết quả của các lệnh sau:

```
MOV   A,#35H    ;Gán A=35H
ANL   A,#0FH    ;Thực hiện AND logic A với 0FH(A=05)
```

Giải:

```
35H0   0 1 1 0 1 0 1
0FH0   0 0 0 1 1 1 1
05H0   0 0 0 0 1 0 1      35H AND 0FH=05H
```

Lệnh OR (Hoặc)

Cú pháp

```
ORL đích,nguồn  ;Đích=đích OR nguồn
```

Lệnh thực hiện OR logic toán hạng đích với toán hạng nguồn, kết quả cất vào đích. Phép OR có thể được dùng để thiết lập những bit nhất định của một toán hạng. Toán hạng đích thường là thanh ghi

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

tích luỹ, toán hạng nguồn có thể là thanh ghi, bộ nhớ hoặc giá trị cho sẵn. Tham khảo phụ lục A để biết thêm về chế độ định địa chỉ của lệnh này. ORL là lệnh dùng cho các toán hạng dạng byte và sẽ không có tác động đến bất kỳ cờ nào. Xem ví dụ sau:

Ví dụ 7.2

Xác định kết quả của đoạn chương trình sau:

```
MOV    A,#04          ;A=04
MOV    A,#68H         ;A=6C
```

Giải:

```
04H0000    0100
68H0110    1000
6CH0110    1100      04  OR  68=6CH
```

Lệnh XOR (OR loại trừ)

Cú pháp

```
XRL đích,nguồn  ;đích=đích OR loại trừ nguồn
```

Lệnh thực hiện phép XOR trên hai toán hạng và đặt kết quả vào đích. Đích thường là thanh ghi tích luỹ. Toán hạng nguồn có thể là thanh ghi, bộ nhớ hoặc giá trị cho sẵn. Xem phụ lục A.1 để biết thêm về chế độ định địa chỉ của lệnh này. XRL là lệnh dùng với toán hạng dạng byte và sẽ không có tác động đến bất kỳ cờ nào. Xét ví dụ các ví dụ sau.

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

Ví dụ 7.3

Trình bày kết quả của đoạn chương trình sau:

```
MOV    A,#54H
XRL    A,#78H
```

Giải:

```
54H0    1   0   1   0   1   0   0
78H0    1   1   1   1   0   0   0
2CH0    0   1   0   1   1   0   1      54H XOR 78H=2CH
```

Ví dụ 7.4

Lệnh XRL có thể dùng để xoá nội dung của một thanh ghi bằng cách XOR với chính nó. Làm rõ lệnh "XRL A, A" xoá nội dung của thanh ghi A như thế nào? giả thiết AH = 45H.

Giải:

```
45H     01000101
45H     01000101
00      00000000      54H XOR 78H=2CH
```

Lệnh XRL cũng có thể được dùng để kiểm tra hai thanh ghi có giá trị giống nhau hay không? Lệnh "XRL A, R1" thực hiện OR loại trừ thanh ghi A với thanh ghi R1 và đặt kết quả vào A. Nếu cả hai thanh ghi có cùng giá trị thì cho kết quả ở A là 0. Sau đó có thể dùng lệnh nhảy JZ để rẽ nhánh thực hiện chương trình dựa vào kết quả vừa tính. Xem ví dụ sau.

Ví dụ 7.5

Đọc và kiểm tra cổng P1 xem có chứa giá trị 45H không? Nếu có thì gửi 99H đến cổng P2, nếu không có thoát khỏi đoạn chương trình.

Giải:

```
  MOV  P2,#00        ;Xóa P2
  MOV  P1,#0FFH      ;Lấy P1 là cổng đầu vào
  MOV  R3,#45H       ;R3=45H
  MOV  A,P1          ;Đọc P1
  XRL  A,R3
  JNZ  EXIT          ;Nhảy nếu A có giá trị khác 0
  MOV  P2,#99H
EXIT: ...
```

Một ứng dụng thường gặp khác của lệnh XRL là đảo một bit nào đó. Ví dụ, đảo bit 2 và giữ nguyên các bit còn lại của thanh ghi A, chúng ta có thể sử dụng lệnh sau.

```
XRLA, #04H   ; A XOR 0000 0100
```

Lệnh CPL A (lấy bù thanh ghi tích luỹ)

Lệnh thực hiện lấy bù nội dung của thanh ghi tích luỹ A. Phép bù là phép đảo các bit 0 thành 1 và 1 thành 0. Phép đảo này còn được gọi là phép bù 1.

```
MOV A,#55H
CPL A          ;Bây giờ A=AAH
               ;Vì đảo 0101 0101(55H)được 1010 1010(AAH)
```

Để lấy bù 2, đơn giản là cộng thêm 1 vào kết quả bù 1. Ở 8051 không có lệnh bù 2. Lưu ý rằng khi lấy bù một byte thì dữ liệu phải ở trong thanh ghi A. Lệnh CPL không hỗ trợ chế độ định địa chỉ nào. Xem ví dụ sau.

Ví dụ 7.6

Tìm giá trị bù 2 của 85H.

Giải:

```
MOV A,#85H       ;Nạp 85H vào A(85H=1000 0101)
MOV A            ;Lấy bù 1 của A(kết quả=0111 1010)
ADD A,#1         ;Cộng 1 vào A được bù 2
                 ;A=0111 1011(7BH)
```

Lệnh so sánh

8051 có một lệnh cho phép so sánh hai toán hạng với nhau. Cú pháp của lệnh như sau:

```
CJNE  Đích,nguồn,địa chỉ tương đối
```

Ở 8051 lệnh so sánh được kết hợp với lệnh nhảy thành một lệnh đó là CJNE (Compare and Jump if Not Equal). Lệnh CJNE so sánh hai toán hạng nguồn và đích và nhảy đến địa chỉ tương đối nếu hai toán hạng không bằng nhau. Ngoài ra, lệnh còn thay đổi cờ nhớ CY để báo nếu toán hạng đích lớn hơn hay nhỏ hơn. Và điều quan trọng là các cờ khác vẫn giữ nguyên không thay đổi. Ví dụ, sau khi thực hiện lệnh "CJNE A, #67H, NEXT" thì thanh ghi A vẫn có giá trị ban đầu (tức giá trị trước lệnh CJNE). Lệnh này so sánh nội dung thanh ghi A với giá trị 67H và nhảy đến địa chỉ đích NEXT nếu thanh ghi A có giá trị khác 67H.

Ví dụ 7.7

Xét đoạn chương trình dưới đây và xác định:

a) Chương trình có nhảy đến NEXT không?

b) A có giá trị bao nhiêu sau lệnh CJNE?

```
        MOV     A,#55H
        CJNE    A,#99H,NEXT
        ...
NEXT:   ...
```

Giải:

a) Có, vì 55H và 99H không bằng nhau.
b) A = 55H là giá trị trước khi thực hiện CJNE.

Ở lệnh CJNE thì toán hạng đích có thể là thanh ghi tích luỹ hoặc trong một các thanh ghi Rn. Toán hạng nguồn có thể là thanh ghi, bộ nhớ hoặc giá trị cho sẵn. Hãy xem phụ lục A để biết thêm chi tiết về các chế độ định địa chỉ của lệnh. Lệnh này chỉ tác động tới cờ nhớ CY và giá trị thay đổi được giới thiệu ở bảng 7.1. Dưới đây là ví dụ minh hoạ sử dụng lệnh so sánh.

Bảng 7.1. Thiết lập cờ CY ở lệnh CJNE

So sánh	Cờ nhớ CY
Đích > Nguồn	CY = 0
Đích < Nguồn	CY = 1

```
            CJNE  R5,#80,NOT-EQUAL  ;Kiểm tra R5=80?
            ...                     ;Nếu R5=80
NOT-EQUAL:  JNC   NEXT              ;Nhảy đến NEXT nếu R5>80
            ...
NEXT:       ...
```

Cũng nên lưu ý là cờ nhớ CY luôn được kiểm tra song chỉ sau khi đã xác định được các toán hạng không bằng nhau. Xem các ví dụ sau.

Ví dụ 7.8

Hãy viết đoạn chương trình xác định xem thanh ghi A có chứa giá trị 99H không? Nếu có thì nạp R1 = FFH còn nếu không thì R1 = 0.

Giải:

```
  MOV   R1,#0     ;Xoá R1
  CJNE  A,#99H    ;Nếu A không bằng 99H thì nhảy đến NEXT
  MOV   R1,#0FFH  ;Nếu bằng nhau thì gán R1=0FFH
NEXT:  ...        ;Nếu không bằng nhau thì gán R1=0
OVER:  ...
```

Ví dụ 7.9

Giả sử có một cảm biến nhiệt được nối tới cổng đầu vào P1. Hãy viết chương trình đọc nhiệt độ và so sánh với giá trị 75. Dựa vào kết quả kiểm tra để đặt giá trị nhiệt độ vào các thanh ghi như sau:

Nếu T = 75 thì A = 75
Nếu T < 75 thì R1 = T
Nếu T > 75 thì R2 = T

Giải:

```
        MOV    P1,0FFH        ;Tạo P1 làm cổng đầu vào
        MOV    A,P1           ;Đọc cổng P1,nhiệt độ
        CJNE   A,#75,OVER     ;Nhảy đến OVER nếu A≠75
        SJMP   EXIT           ;A=75 thoát
OVER:   JNCNEXT               ;Nếu CY=0 thì A>75 nhảy đến
        MOV    R1,A           ;Nếu CY=1thì A<75 lưu vào R1
        SJMP   EXIT           ;và thoát
NEXT:   MOV    R2,A           ;A>75 lưu nó vào R2
EXIT:   ...
```

Lệnh so sánh về bản chất là một phép tính trừ song không làm thay đổi giá trị của các toán hạng. Các cờ được thay đổi tuỳ thuộc vào kết quả thực hiện lệnh trừ SUBB. Cũng cần nhắc lại rằng, kết quả so sánh ở lệnh CJNE không có tác động lên các toán hạng mà chỉ tác động lên cờ CY.

Ví dụ 7.10

Viết chương trình liên tục kiểm tra cổng P1 khi có giá trị khác với 63H. Nếu P1 = 63H thì không kiểm tra P1 nữa.

Giải:

```
        MOV    P1,#0FFH       ;Chọn P1 làm cổng đầu vào
HERE:   MOV    A,P1           ;Lấy nội dung của P1
        CJNE   A,#63,HERE     ;Duy trì kiểm tra trừ khi P=63H
```

Ví dụ 7.11

Giả sử các ngăn nhớ của RAM ở địa chỉ 40H - 44H chứa nhiệt độ của 5 ngày như được chỉ ra dưới đây. Hãy tìm xem có giá trị nào bằng 65 không?

Nếu có giá trị 65 hãy đặt địa chỉ ngăn nhớ vào R4, nếu không thì đặt R4 = 0.

```
40H=(76); 41H=(79); 42H=(69); 43H=(65); 44H=(64)
```

Giải:

```
      MOV   R4,#0          ;Xoá R4 = 0
      MOV   R0,#40H        ;Nạp con trỏ
      MOV   R2,#05         ;Nạp bộ đếm
      MOV   A,#65          ;Gán giá trị cần tìm vào A
BACK: CJNE  A,@R0,NEXT     ;So sánh dữ liệu RAM với 65
      MOV   R4,R0          ;Nếu là 65,lưu địa chỉ vào R4
      SJMP  EXIT           ;Thoát
NEXT: INC   R0             ;Nếu không tăng bộ đếm
      DJNZ  R2,BACK        ;Kiểm tra cho đến khi bộ đếm = 0
EXIT: ...
```

7.2 LỆNH QUAY VÀO TRÁO ĐỔI

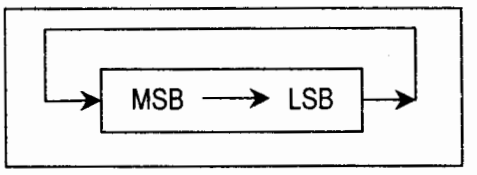
Trong rất nhiều ứng dụng cần phải thực hiện phép quay bit của một toán hạng. Ở 8051 có các lệnh chuyên cho mục đích này là R1, RR, RLC và RRC. Chúng cho phép quay các bit của thanh ghi tích luỹ sang trái hoặc phải. Đối với 8051, để quay một byte thì toán hạng phải ở trong thanh ghi A. Có hai kiểu quay là: quay đơn giản các bit của thanh ghi A và quay có nhớ.

Quay các bit của thanh ghi A sang trái hoặc phải

Quay phải

```
RR A  ;Quay các bit thanh ghi A sang phải
```

Ở phép quay phải, 8 bit của thanh ghi tích luỹ được quay sang phải 1 bit và bit D0 rời từ vị trí bit thấp nhất và chuyển sang bit cao nhất D7. Xem đoạn chương trình dưới đây:

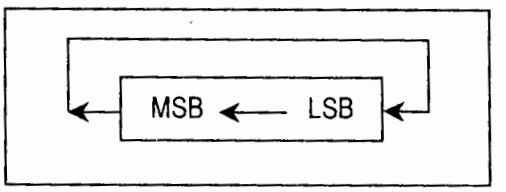



```
MOV   A,#36H        ;A=0011 0110
RR    A             ;A=0001 1011
RR    A             ;A=1000 1101
RR    A             ;A=1100 0110
RR    A             ;A=0110 0011
```

Quay trái

```
RL    A  ;Quay trái các bit của thanh ghi A
```

Ở phép quay trái, 8 bit của thanh ghi A được quay sang trái 1 bit và bit D7 rời khỏi vị trí bit cao nhất chuyển sang vị trí bit thấp nhất D0. Xem đoạn mã chương trình sau:




```
MOV   A,#72H     ;A=0111 0010
RL    A          ;A=1110 0100
RL    A          ;A=1100 1001
```

Lưu ý rằng các lệnh RR và RL không tác động lên cờ nào.

Quay có nhớ

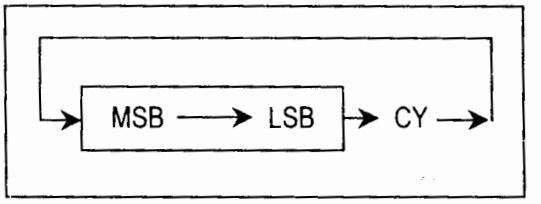
8051 còn có 2 lệnh quay nữa là quay phải có nhớ và quay trái có nhớ.

RRCA và RLC A

Quay phải có nhớ

```
RRC   A
```

Khi quay phải có nhớ thì các bit của thanh ghi A được quay từ trái sang phải 1 bit, bit thấp nhất được đưa vào cờ nhớ CY và sau đó cờ CY được đưa vào vị trí bit cao nhất. Nói cách khác, ở lệnh "RRC A" thì LSB được chuyển vào CY và CY được chuyển vào MSB. Thực tế, cờ nhớ CY làm việc như là một bit của thanh ghi A, do vậy thanh ghi A đường như có 9 bit.

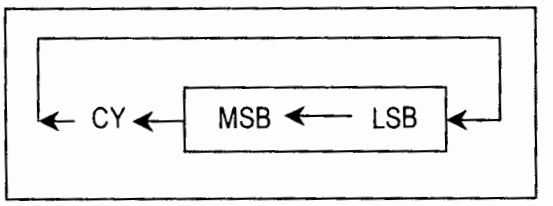



```
CLR   C          ;make CY=0
MOV   A#26H      ;A=0010 0110
RRC   A          ;A=0001 0011  CY=0
RRC   A          ;A=0000 1001  CY=1
RCC   A          ;A=1000 0100  CY=1
```

Quay trái có nhớ

```
RLC   A
```

Khi thực hiện lệnh RLC A, các bit được dịch phải một bit và đẩy bit MSB vào cờ nhớ CY, sau đó cờ CY được

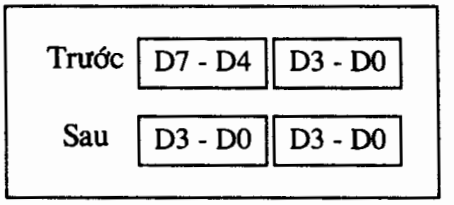


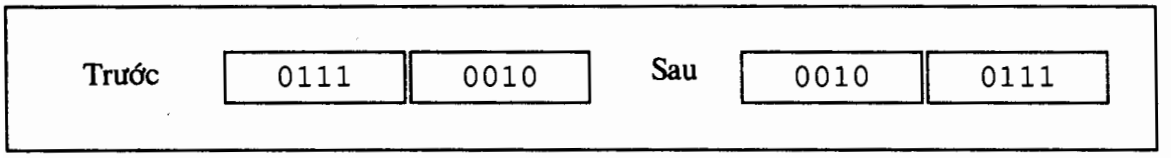
chuyển vào bit LSB. Nói cách khác, ở lệnh RLC thì bit MSB được chuyển vào CY và CY được chuyển vào LSB. Hãy xem đoạn chương trình sau.

```
SETB   C          ;Đặt CY=1
MOV    A#15H      ;A=0001 0101
RRC    A          ;A=0101 1011  CY=0
RRC    A          ;A=0101 0110  CY=0
RCC    A          ;A=1010 1100  CY=0
RCC    A          ;A=1000 1000  CY=1
```

Lệnh tráo đổi thanh ghi A: SWAP A

Một lệnh hữu ích khác nữa là lệnh tráo đổi SWAP. Lệnh này chỉ làm việc ở thanh ghi A theo cách: nửa phần cao của byte và nửa phần thấp của byte được tráo đổi cho nhau. Như vậy, 4 bit cao được chuyển thành 4 bit thấp và 4 bit thấp trở thành 4 bit cao.





Ví dụ 7.12

a) Hãy tìm nội dung của thanh ghi A ở đoạn chương trình sau.

b) Trường hợp không có lệnh SWAP thì cần phải làm thế nào để tráo đổi những bit này? Hãy viết đoạn chương trình thực hiện quá trình đó.

Giải:

a)

```
   MOV    A,#72H     ;A=72H
   SWAP   A          ;A=27H
```

b)

```
   MOV    A,#72H     ;A=0111 0010
   RL     A          ;A=1110 0100
   RL     A          ;A=1100 1001
   RL     A          ;A=0010 0111
```

Ví dụ 7.13

Viết chương trình để tìm tổng số các số 1 trong một byte đã cho.

Giải:

```
      MOV    R1,#0          ;Chọn R1 lưu số các số 1
      MOV    R7,#8          ;Đặt bộ đếm = 8 để quay 8 lần
      MOV    A,#97H         ;Tìm các số 1 trong byte 97H
AGAIN:RLC A                 ;Quay trái có nhớ một lần
      JNC    NEXT           ;Kiểm tra cờ CY
      INC    R1             ;Nếu CY=1 thì cộng 1 vào bộ đếm
NEXT: DJNZ R7,AGAIN         ;Lặp lại quá trình 8 lần
```

Để truyền nối tiếp byte dữ liệu thì cần chuyển byte dữ liệu đó từ dạng song song sang nối tiếp bằng các lệnh quay như sau:

```
RRC    A              ;Bit thứ nhất đưa vào cờ CY
MOV    P1.3,C         ;Xuất CY như một bit dữ liệu
RRC    A              ;Bit thứ hai đưa vào CY
MOV    P1.3,C         ;Xuất CY ra như một bit dữ liệu
RRC    A              ;
MOV    P1.3,C         ;
...
```

Đoạn chương trình trên đây là một phương pháp được sử dụng rộng rãi trong truyền dữ liệu tới các bộ nhớ nối tiếp như EEPROM.

7.3 CHƯƠNG TRÌNH ỨNG DỤNG MÃ BCD VÀ ASCII

Số mã BCD đã được giới thiệu ở chương 6. Chúng ta đều biết, trong các bộ vi điều khiển thế hệ mới đều có một đồng hồ thời gian thực RTC để duy trì thời gian và ngày tháng kể cả khi bị tắt nguồn. Các bộ vi điều khiển này cung cấp thời gian và ngày tháng dưới dạng số BCD. Tuy nhiên, để hiển thị được thì chúng phải chuyển về mã ASCII. Ở phần này chúng ta ứng dụng lệnh quay và lệnh logic trong việc chuyển đổi mã BCD và ASCII.

Bảng 7.2. Mã ASCII của các số từ 0- 9

Phím	Mã ASCII (hexa)	Mã nhị phân	Mã BCD không nén
0	30	011 0000	0000 0000
1	31	011 0001	0000 0001

2	32	011 0010	0000 0010
3	33	011 0011	0000 0011
4	34	011 0100	0000 0100
5	35	011 0101	0000 0101
6	36	001 0110	0000 0110
7	37	001 0111	0000 0111
8	38	001 1000	0000 1000
9	39	001 1001	0000 1001

Mã ASCII

Trên bàn phím ASCII, khi nhấn phím "0" thì dãy mã "011 0001" (30H) được đưa tới máy tính. Tương tự như vậy 31H (011 0001) ứng với phím "1" v.v. như chỉ ra trong bảng 7.2.

Cần nhớ rằng, mã ASCII là chuẩn ở Mỹ (và nhiều quốc gia khác), song các số mã BCD được dùng chung cho mọi quốc gia. Vì bàn phím, máy in và màn hình đều sử dụng mã ASCII nên cần phải thực hiện đổi chuyển giữa mã ASCII về mã BCD và ngược lại.

Chuyển đổi mã BCD nén về ASCII

Các bộ vi điều khiển DS5000T đều có đồng hồ thời gian thực RTC. Đồng hồ này cung cấp liên tục thời gian trong ngày (giờ, phút và giây) và lịch (năm, tháng, ngày) mà không quan tâm đến nguồn tắt hay bật. Tuy nhiên, dữ liệu này được cho dưới dạng mã BCD nén. Để hiển thị dữ liệu này trên màn hình LCD hoặc in ra máy in thì cần được chuyển về dạng mã ASCII.

Để chuyển đổi mã BCD nén về mã ASCII, trước hết cần được chuyển về mã BCD không nén rồi sau đó mới chuyển sang mã ASCII. Dưới đây minh hoạ việc chuyển đổi từ mã BCD nén về mã ASCII.

Mã BCD nén	*Mã BCD không nén*	*Mã ASCII*
29H	02H & 09H	32H & 39H
0010 1001	0000 0010 & 0000 1001	0011 0010 & 0011 1001

Chuyển đổi mã ASCII về mã BCD nén

Để chuyển đổi mã ASCII về BCD nén, trước hết mã ASCII cần được chuyển về mã BCD không nén và sau đó được kết hợp để tạo ra mã BCD nén. Ví dụ số 4 và số 7 thì bàn phím nhận được 34 và 37. Mục tiêu là tạo ra số 47H hay "0100

0111" là mã BCD nén. Quá trình này như sau:

Phím	*Mã ASCII*	*Mã BCD không nén*	*Mã BCD nén*
4	34	0000 0100	
7	37	0000 0111	0100 0111 hay 47H

```
MOV   A,#'4'    ;Gán mã ASCII của số 4 cho A (A=34H)
MOV   R1,#'7'   ;Gán mã ASCII của số 4 cho R1 (R1=37H)
ANL   A,#0FH    ;Che nửa byte cao A(A=04)
ANL   R1,#0FH   ;Che nửa byte cao của R1(R1=07)
SWAP  A         ;A=40H
ORL   A,R1      ;A=47H,mã BCD nén
```

Kết quả của đoạn chương trình trên cho số BCD dạng nén ở thanh ghi A. Cũng nên nói thêm rằng, có một số lệnh đòi hỏi toán hạng phải là số BCD nén. Ví dụ, như ở chương 6 đã biết, đó là lệnh "DA A".

Ví dụ 7.14

Giả sử thanh ghi A có số BCD nén. Hãy viết chương trình để chuyển số BCD thành hai số ASCII và đặt chúng vào R2 và R6.

Giải:

```
MOV   A,#29H      ;Gán A=29, mã BCD nén
MOV   R2,A        ;Giữ một bản sao của BCD trong R2
ANL   A,#0FH      ;Che phần nửa cao của A (A=09)
ORL   A,#30H      ;Chuyển thành mã ASCII A=39H
MOV   R6,A        ;Lưu vào R6(R6=39H - ký tự ASCII)
MOV   A,R2        ;Lấy giá trị ban đầu của A(A= 29H)
ANL   A,#0F0H     ;Che nửa byte phần thấp của A(A=20)
RR    A           ;Quay phải
RR    A           ;Quay phải
RR    A           ;Quay phải
RR    A           ;Quay phải(A=02)
ORL   A,#30H      ;Chuyển thành mã ASCII(A=32H)
MOV   R2,A        ;Lưu ký tự ASCII vào R2
```

Dĩ nhiên, ở ví dụ trên đây, chúng ta có thể thay 4 lệnh RR quay phải bằng một lệnh trao đổi WAPA.

Chương 8
LỆNH XỬ LÝ BIT VÀ LẬP TRÌNH

8.1 LẬP TRÌNH VỚI CÁC LỆNH XỬ LÝ BIT

Ở hầu hết các bộ vi xử lý, dữ liệu được truy cập theo từng byte, có nghĩa là, nội dung của thanh ghi, bộ nhớ RAM hay cổng đều phải được truy cập từng byte một. Nói cách khác, lượng dữ liệu tối thiểu có thể được truy cập là một byte. Ví dụ, bộ vi xử lý Pentium có cổng vào/ra được tổ chức theo byte, khi đó, để thay đổi một bit thì trước hết cần truy cập toàn bộ 8 bit. Song trong thực tế, có nhiều ứng dụng chỉ cần thay đổi giá trị của một bit, chẳng hạn như bật hoặc tắt một thiết bị. Do vậy, khả năng định địa chỉ đến từng bit của 8051 rất thích hợp cho các ứng dụng dạng này. 8051 có thể truy cập đến từng bit thay vì phải truy cập cả byte làm cho nó trở thành một trong những bộ vi điều khiển 8 bit mạnh nhất trên thị trường. Vậy những bộ phận nào của CPU, RAM, thanh ghi, cổng I/O hoặc ROM có thể định địa chỉ bit được?. ROM chỉ đơn giản lưu mã chương trình thực thi, nên nó không cần định địa chỉ bit. Hầu hết tất cả các lệnh đều được định hướng theo byte, tuy nhiên ở 8051, có nhiều vị trí RAM trong, một số thanh ghi và tất cả các cổng I/O lại có thể định địa chỉ theo bit. Dưới đây chúng ta sẽ tìm hiểu kỹ hơn về vấn đề này.

Các lệnh xử lý bit

Các lệnh xử lý bit được cho ở bảng 8.1. Một số lệnh một bit khác chỉ liên quan đến cờ nhớ CY (Cary Flag) sẽ được đề cập ở mục sau.

Bảng 8.1. Các lệnh xử lý bit của 8051

Lệnh	Chức năng
`SETB bit`	Thiết lập bit (bit = 1)
`CLR bit`	Xoá bit về 0 (bit = 0)
`CPL bit`	Bù bit (bit = NOT bit)
`JB bit, đích`	Nhảy về đích nếu bit = 1
`JNB bit, đích`	Nhảy về đích nếu bit = 0
`JBC bit, đích`	Nhảy về đích nếu bit = 1 và sau đó xoá bit

Cổng I/O và định địa chỉ bit

Bộ vi điều khiển 8051 có bốn cổng I/O 8 bit là P0, P1, P2 và P3. Chúng ta có thể truy cập toàn bộ 8 bit hoặc bit bất kỳ mà không làm thay đổi các bit còn lại. Nếu muốn truy cập từng bit của cổng, có thể sử dụng lệnh "SETB X, Y" với X là số của cổng 0, 1, 2 hoặc 3, còn Y là vị trí bit từ 0 đến 7. Ví dụ "SETB P1.5" là thiết lập lên 1 bit số 5 của cổng 1. Cần nhớ rằng, D0 là bit có nghĩa thấp nhất LSB và D7 là bit có nghĩa là cao nhất MSB. Xem ví dụ 8.1.

Ví dụ 8.1

Viết các chương trình sau:

a) Tạo sóng vuông (hàm xung vuông) với độ lấp đầy xung 50% ở bit 0 cổng 1.

b) Tạo một hàm xung vuông với độ đầy xung 66% ở bit 3 cổng 1.

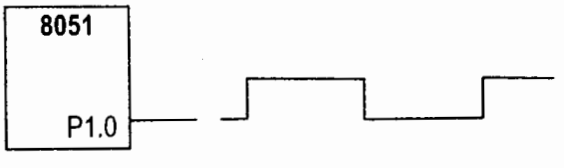
Giải:

a) Hàm xung vuông với độ đầy xung 50% có nghĩa là trạng thái "bật" và "tắt" (hoặc phần cao và thấp của xung) có cùng độ dài. Do vậy ta thay đổi mức của bit P1.0 với cùng một thời gian giữ chậm.

```
HERE:  SETB   P1.0   ;Thiết lập bit 0 cổng 1 lên 1
       LCALL  DELAY  ;Gọi trình con giữ chậm DELAY
       CLR    P1.0   ;Xoá P1.0=0
       SJMP HERE     ;Tiếp tục thực hiện.
```

Cũng có thể viết chương trình này theo cách khác:

```
HERE:  CPL    P1.0   ;Lấy bù bit 0 của cổng 1.
       LCALL DELAY   ;Gọi chương trình con giữ chậm DELAY
       SJMP   HERE   ;Tiếp tục thực hiện nó.
```



b) Hàm xung vuông với độ đầy xung 66% có nghĩa là trạng thái "bật" có thời gian dài gấp đôi trạng thái "tắt".

```
BACK:  SETB   P1.3      ;Thiết lập bit 3 cổng 1 lên 1
       LCALL DELAY      ;Gọi chương trình con DELAY
       LCALL DELAY      ;Gọi chương trình DELAY lần nữa
```

```
CLR    P1.3       ;Xoá bit 3 của cổng 1
LCALL  DELAY      ;Gọi chương trình con DELAY
SJMP   BACK       ;Tiếp tục thực hiện
```

Lưu ý rằng, lệnh "SETB P1.0" khi hợp dịch sẽ trở thành "SETB 90H" vì P1.0 có địa chỉ RAM là 90h. Từ hình 8.1 ta thấy rằng, địa chỉ bit của P0 là từ 80H đến 87H, của P1 là từ 90H đến 97H v.v. Hình 8.1 cũng chỉ ra tất cả các thanh ghi có khả năng định địa chỉ bit.

Bảng 8.2. Định địa chỉ bit của các cổng

P0	P1	P2	P3	Bit
P0.0	P1.0	P2.0	P3.0	D0
P0.1	P1.1	P2.1	P3.1	D1
P0.2	P1.2	P2.2	P3.2	D2
P0.3	P1.3	P2.3	P3.3	D3
P0.4	P1.4	P2.4	P3.4	D4
P0.5	P1.5	P2.5	P3.5	D5
P0.6	P1.6	P2.6	P3.6	D6
P0.7	P1.7	P2.7	P3.7	D7

Ví dụ 8.2

Sử dụng hình 8.1 hãy xác định các lệnh dưới đây sẽ tác động lên bit nào của SFR.

```
a)SETB 86H;      b)CLR 87H;       c)SETB 92H
d)SETB DA7H;     e)CLR 0F2H;      f)SETB 0E7H
```

Giải:

```
a)SETB 86H là SETB P0.6.
b)CLR  87H là CLR P0.7.
c)SETB 92H là SETB P1.2.
d)SETB 0A7H là SETB P2.7.
e)CLR  0F2H là CLR D2 của thanh ghi B.
f)SETB 0E7H là SETB ACC.7(bit D7 của thanh ghi A).
```

Kiểm tra bit vào

Lệnh JNB (nhảy nếu bit = 0) và JB (nhảy nếu bit = 1) cũng là những phép thao tác bit được sử dụng rộng rãi. Các lệnh này cho phép kiểm tra bit và thực

hiện lệnh tiếp theo phụ thuộc vào giá trị của bit đó.

Ví dụ 8.3

Giả sử bit P2.3 là một đầu vào và biểu diễn nhiệt độ của một lò sấy. Nếu bit có giá trị 1 có nghĩa là lò quá nóng. Hãy liên tục kiểm tra bit này, nếu nhiệt độ quá cao thì hãy gửi một xung Cao-xuống-Thấp (High-to-Low) đến cổng P1.5 để bật còi báo hiệu.

Giải:

```
HERE: JNB    P2.3,HERE    ;Duy trì kiểm tra bit
      SETB   P1.5         ;Thiết lập P1.5=1
      CLR    P1.5         ;Chuyển xung từ Cao-xuống-Thấp
```

Ví dụ 8.4

Hãy viết chương trình kiểm tra xem thanh ghi tích luỹ có chứa một số chẵn không? Nếu có thì chia cho 2, nếu không thì tăng lên 1 để chẵn hoá số đó rồi chia nó cho 2.

Giải:

```
      MOV    B,#2          ;Gán B=2
      JNB    ACC 0,YES     ;D0 của thanh ghi A có bằng 0?
      JNC    A             ;Nếu có thì nhảy về YES
YES:  DIV    AB            ;Chia A/B
```

Ví dụ 8.5

Hãy viết đoạn chương trình để kiểm tra xem bit 0 và 5 của thanh ghi B có giá trị cao không? Nếu không phải thì đặt chúng lên 1 và lưu vào thanh ghi R0.

Giải:

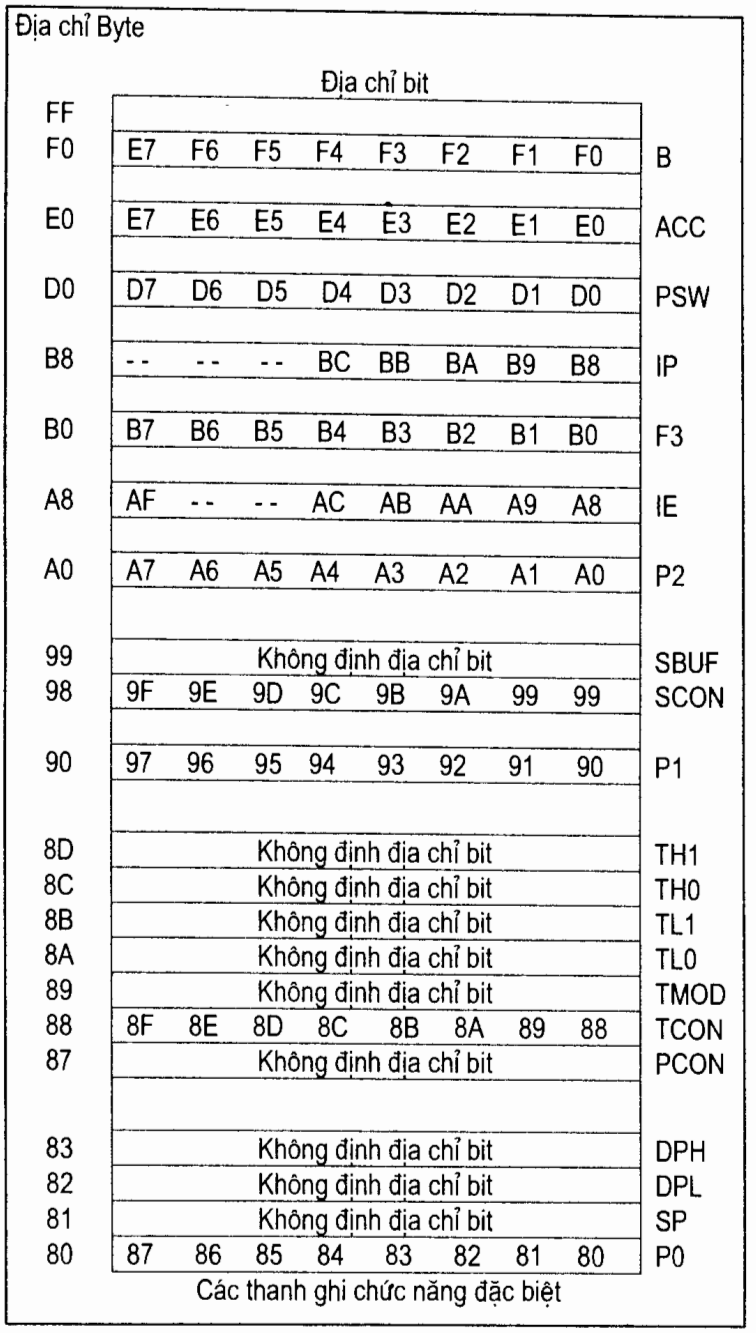
```
      JNB    0F0H,NEXT-1      ;Nhảy về NEXT-1 nếu B.0=0
      SETB   0F0H             ;Đặt B.0=1
NEXT-1:JNB   0F5H,NEXT-2      ;Nhảy về NEXT-2 nếu B.5=0
      SETB   0F5H             ;Đặt B.5=1
NEXT-2:MOV   R0,B             ;Cất B vào thanh ghi R0
```

Các lệnh JNB và JB có thể được dùng cho các bit bất kỳ của cổng I/O 0, 1, 2

và 3 vì tất cả các cổng này đều có khả năng định địa chỉ bit. Tuy nhiên, cổng 3 hầu như chỉ để dùng cho các tín hiệu ngắt, truyền thông nối tiếp và không dùng cho I/O. Vấn đề này sẽ được nghiên cứu ở chương 10 và 11.

Các thanh ghi định địa chỉ bit

Tất cả các cổng I/O đều có khả năng định địa chỉ bit, tuy nhiên đối với các thanh ghi thì khác. Từ hình 8.1 có thể thấy, chỉ các thanh ghi: B, PSW, IP, IE, ACC, SCON và TCON là có thể định địa chỉ bit. Ở đây chúng ta sẽ tập trung vào các thanh ghi quen thuộc là A, B và PSW, còn các thanh ghi khác sẽ được đề cập ở các chương sau. Như hình 8.1 giới thiệu, địa chỉ bit 80H-87H được gán cho cổng P0, địa chỉ bit 88-8FH được gán cho thanh ghi TCON, cuối cùng, địa chỉ bit F0-F7H được gán cho thanh ghi B. Chúng ta tìm hiểu khả năng định địa chỉ bit của các thanh ghi này qua các ví dụ 8.4 và 8.5.



Hình 8.1. Địa chỉ byte và bit của bộ nhớ RAM các thanh ghi chức năng đặc biệt

CY	AC	- -	RS1	RS0	OV	- -	P

RS1	*RS0*	*Băng thanh ghi*	*Địa chỉ*
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình 8.2. Các bit của thanh ghi PSW

Như đã nói ở chương 2, thanh ghi PSW có hai bit dùng để chọn băng thanh ghi. Khi RESET thì chọn ngầm định băng 0, tuy nhiên chúng ta có thể chọn băng bất kỳ bằng cách sử dụng khả năng định địa chỉ bit của PSW.

Ví dụ 8.6

Hãy viết chương trình để lưu thanh ghi tích luỹ vào R7 của băng 2.

Giải:

```
CLR    PSW.3
SETB   PSW.4
MOV    R7.A
```

Ví dụ 8.7

Có hai lệnh JNC và JC chuyên để kiểm tra bit cờ nhớ CY, song lại không có lệnh nào để kiểm tra bit cờ tràn OV. Hãy viết đoạn chương trình kiểm tra cờ tràn OV.

Giải:

Cờ OV là bit PSW.2 của thanh ghi PSW. PSW là thanh ghi có thể định địa chỉ bit, do vậy ta có thể sử dụng lệnh sau để kiểm tra cờ OV:

```
JB PSW.2,TARGET    ;Nhảy về TARGET nếu OV=1
```

Vùng nhớ RAM định địa chỉ bit

Trong số 128 byte RAM trong của 8051 thì chỉ có 16 byte là có thể định địa chỉ bit được. Phần còn lại được định địa chỉ byte. Vùng RAM có thể định địa chỉ bit là từ 20H đến 2FH. Như vậy, 16 byte này cho phép định địa chỉ cho 128 bit, vì

16 × 8 = 128. Chúng được đánh địa chỉ từ 0 đến 127D, hoặc từ 00 đến 7FH. Địa chỉ bit từ 0 đến 7 dùng cho byte đầu tiên ngăn nhớ 20H của RAM, các bit từ 8 đến 0FH là địa chỉ bit của byte thứ hai ngăn nhớ 21H của RAM.v.v. Byte cuối cùng 2FH có địa chỉ bit từ 78H đến 7FH (xem hình 8.3). Lưu ý rằng các vị trí RAM trong từ 20H đến 2FH vừa có thể định địa chỉ byte vừa có thể định địa chỉ bit.

Địa chỉ byte	Địa chỉ bit							
7F 30	Bộ nhớ RAM đa năng							
2F	7F	7E	7D	7C	7A	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1C	1E	1C	1B	1A	19	18
22	27	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F 18	Băng 3							
17 10	Băng 2							
0F 08	Băng 1							
07 00	Băng thanh ghi ngầm định R0-R7							

Hình 8.3. 128 byte của RAM trong

Từ hình 8.3 và 8.1 có thể thấy rằng, các địa chỉ bit 00 - 7FH thuộc về địa chỉ byte của RAM từ 20H - 2FH và các địa chỉ bit từ 80 đến F7H thuộc các thanh ghi đặc biệt SFR, các cổng P0, P1, v.v.

Ví dụ 8.8

Hãy kiểm tra xem các bit sau đây thuộc byte nào? Hãy cho địa chỉ của byte RAM ở dạng Hexa.

```
a) SETB    42H    ;Thiết lập bit 42H về 1
b) CLR     67H    ;Xoá bit 67
c) CLR     0FH    ;Xoá bit OFH
d) SETB    28H    ;Lập bit 28H lên 1
```

```
e)CLR      12      ;Xoá bit 12(thập phân)
f)SETB     05
```

Giải:

```
a)Địa chỉ bit 42H của RAM thuộc bit D2 vị trí RAM 28H
b)Địa chỉ bit 67H của RAM thuộc bit D7 vị trí RAM 2CH
c)Địa chỉ bit 0FH của RAM thuộc bit D7 vị trí RAM 21H
d)Địa chỉ bit 28H của RAM thuộc bit D0 vị trí RAM 25H
e)Địa chỉ bit 12H của RAM thuộc bit D4 vị trí RAM 21H
f)Địa chỉ bit 05H của RAM thuộc bit D5 vị trí RAM 20H
```

Ví dụ 8.9

Trạng thái của các bit P1.2 và P1.3 của cổng vào/ra P1 phải được lưu cất trước khi bị thay đổi. Hãy viết chương trình để lưu trạng thái của P1.2 vào bit 06 và trạng thái P1.3 vào vị bit 07.

Giải:

```
      CLR  06          ;Xoá địa chỉ bit 06
      CLR  07          ;Xoá địa chỉ bit 07
      JNB  P1.2,OVER   ;Nhảy về OVER nếu P1.2=0
      SETB 06          ;Thiết lập bit 6 lên 1 nếu P1.2=1
OVER: JNB  P1.3,NEXT   ;Nhảy về NEXT nếu P1.3=0
      SETB 07          ;Thiết lập bit 7 lên 1 nếu P1.3=1
NEXT: ....
```

8.2 LỆNH THAO TÁC CỜ NHỚ CY

Như đã thấy ở trên, lệnh logic và số học có thể tác động lên cờ nhớ CY, tuy nhiên, đối với 8051 còn có một số lệnh khác cũng có thể thao tác trực tiếp cờ nhớ CY. Các lệnh này được trình bày ở bảng 8.3.

Trong số các lệnh được nêu ở bảng thì chúng ta đã làm quen với lệnh JNC, CLR và SETB. Dưới đây chúng ta tiếp tục tìm hiểu về một số lệnh khác.

Bảng 8.3. Các lệnh thao tác cờ nhớ CY

Lệnh		Chức năng
SETB	C	Đặt bit nhớ CY=1
CLR	C	Xoá bit nhớ CY=0

CPL	C	Bù bit nhớ
MOV	b,C	Sảo chép trạng thái bit nhớ vào bit b=CY
MOV	C,b	Sao chép bit b vào trạng thái bit nhớ CY=b
JNC	đích	Nhảy tới đích nếu CY=0
JC	đích	Nhảy tới đích nếu CY=1
ANL	C.bit	Thực hiện phép AND với bit b và lưu vào CY
ANL	C./bit	Thực hiện phép AND với bit đảo,lưu vào CY
ORL	C.bit	Thực hiện phép OR với bit và lưu vào CY
ORL	C./bit	Thực hiện phép OR với bit đảo,lưu vào CY

Ví dụ 8.10

Hãy viết chương trình lưu cất trạng thái của các bit P1.2 và P1.3 vào vị trí RAM 6 và 7 tương ứng.

Giải:

```
MOV  C,P1.2  ;Lưu trạng thái P1.2 vào CY
MOV  06,C    ;Lưu trạng thái CY vào bit 6 của RAM
MOV  C,P1.3  ;Lưu trạng thái P1.2 vào CY
MOV  07,C    ;Lưu trạng thái CY vào vị trí RAM 07
```

Ví dụ 8.11

Giả sử bit nhớ 12H trong RAM giữ trạng thái có điện thoại gọi đến hay không. Nếu bit ở trạng thái cao có nghĩa là có một cuộc gọi mới. Hãy viết chương trình để hiển thị "có cuộc gọi" trên màn hình LCD nếu bit 12H có giá trị cao. Nếu bit này có giá trị thấp thì LCD hiển thị "không có cuộc gọi".

Giải:

```
   MOV   C,12H          ;Sao bit 12H của RAM vào CY
   JNC   NO             ;Kiểm tra trạng thái cao cờ CY
   MOV   DPTR,#400H     ;Nạp địa chỉ "có cuộc gọi" nếu cao
   LCAL  DISPLAY        ;Hiển thị thông báo
   SJMP  NEXT           ;Thoát
NO:MOV   DSTR,#420H     ;Nạp địa chỉ "Không có cuộc gọi"
   LCAL  DISPLAY        ;Hiển thị thông báo
   EXIT                 ;Thoát
```

```
;---- Dữ liệu hiển thị ở LCD
          ORG    400H
YES-MG:   DB     "Có cuộc gọi"
          ORG    420H
NO-MG:    DB     "Không có cuộc gọi"
```

Ví dụ 8.12

Giả sử bit P2.2 được dùng để kiểm tra đèn bên ngoài còn bit P2.5 dùng để kiểm tra đèn bên trong của một toà nhà. Hãy viết chương trình để bật đèn ngoài và tắt đèn trong nhà.

Giải:

```
SET    BC          ; Đặt CY=1
ORL    C,P2.2,C    ; Thực hiện phép OR với CY
MOV    P2.2,C      ; Bật đèn nếu chưa bật
CLR    C           ; Xoá CY=0
ANL    C,P2.5      ; CY=(P2.5 AND CY)
MOV    P2.5,C      ; Tắt đèn nếu chưa được tắt
```

8.3 ĐỌC CÁC CHÂN ĐẦU VÀO VÀ CHỐT CỔNG

Người ta phân biệt 2 tình huống đọc trạng thái cổng, đó là:

- Đọc trạng thái của chân vào.
- Đọc chốt trong của cổng ra.

Các lệnh đọc cổng vào

Như đã nói ở chương 4, để đặt một bit bất kỳ của cổng 8051 làm đầu vào, chúng ta phải ghi mức logic cao vào bit đó. Sau khi đã định cấu hình các bit của cổng là đầu vào, ta có thể sử dụng những lệnh nhất định để nhận dữ liệu từ bên ngoài trên các chân vào. Bảng 8.4 là những lệnh nói trên.

Bảng 8.4. Các lệnh đọc cổng vào

Lệnh	Ví dụ	Mô tả
MOV A,PX	MOV A,P2	Chuyển dữ liệu ở chân P2 vào ACC
JNB PX.Y,...	JNB P2.1,đích	Nhảy tới đích nếu, chân P2.1 = 0
JB PX.Y,...	JB P1.3,đích	Nhảy đích nếu, chân P1.3 = 1
MOV C,PX.Y	MOV C,P2.4	Sao trạng thái chân P2.4 vào CY

Đọc chốt cổng ra

Bảng 8.5 liệt kê một số lệnh đọc nội dung của chốt cổng trong. Ví dụ, xét lệnh "ANL P1, A". Trình tự thực hiện lệnh này như sau:

1. Đọc chốt trong của cổng và chuyển dữ liệu đó vào trong CPU.
2. Dữ liệu này được AND với nội dung của thanh ghi A.
3. Kết quả được ghi ngược lại ra chốt cổng.
4. Dữ liệu tại chân cổng được thay đổi và có cùng giá trị như chốt cổng.

Từ những bàn luận trên ta thấy rằng, các lệnh đọc chốt cổng thường đọc một giá trị, thực hiện một phép tính (và có thể thay đổi nó) sau đó ghi ngược lại ra chốt cổng. Điều này thường được gọi "*Đọc-sửa-ghi*", ("Read-Modify-Write"). Bảng 8.5 liệt kê các lệnh đọc-sửa-ghi sử dụng cổng làm toán hạng đích, hay nói cách khác, chúng chỉ được dùng cho các cổng được cấu hình thành cổng ra.

Bảng 8.5. Các lệnh đọc chốt (đọc-sửa-ghi)

Lệnh	Ví dụ
ANL PX	ANL P1, A
ORL PX	ORL P2, A
XRL PX	XRL P0, A
JBC PX.Y,đích	JBC P1.1, đích
CPL PX	CPL P1.2
INC PX	INC P1
DEC PX	DEC P2
DJNZ PX.Y,đích	DJN2 P1,đích
MOV PX.Y,C	MOV P1.2, C
CLR PX.Y	CLR P2.3
SETB PX.Y	SETB P2.3

Ghi chú: Để có thể tiến hành nối phần cứng với hệ 8051, Bạn cần nghiên cứu kỹ cấu trúc cổng vào ra của 8051, vì nếu thực hiện sai hoặc nối chân không đúng có thể làm hỏng các cổng của 8051.

Chương 9
BỘ ĐẾM/BỘ ĐỊNH THỜI VÀ LẬP TRÌNH

8051 có hai bộ định thời/bộ đếm. Chúng có thể được dùng làm bộ định thời để tạo trễ thời gian hoặc làm bộ đếm để đếm các sự kiện xảy ra bên ngoài bộ vi điều khiển.

9.1 LẬP TRÌNH CÁC BỘ ĐỊNH THỜI CỦA 8051

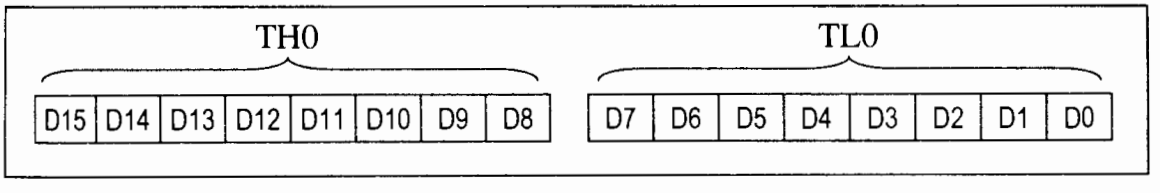
8051 có hai bộ định thời là Timer 0 và Timer 1. Ở phần này chúng ta tìm hiểu các thanh ghi và phương pháp lập trình để tạo ra các độ trễ thời gian.

Các thanh ghi cơ sở của bộ định thời

Cả hai bộ định thời Timer 0 và Timer 1 đều có độ dài 16 bit. Do 8051 có cấu trúc 8 bit, nên mỗi bộ định thời được truy cập dưới dạng hai thanh ghi độc lập là byte thấp và byte cao. Chúng ta sẽ nghiên cứu từng thanh ghi.

Thanh ghi của bộ Timer 0

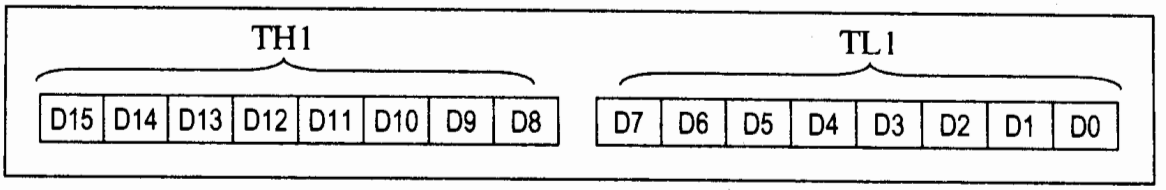
Thanh ghi 16 bit của bộ Timer 0 được truy cập theo 2 byte là byte thấp và byte cao. Thanh ghi byte thấp được gọi là TL0 (Timer 0 Low byte) và thanh ghi byte cao là TH0 (Timer 0 high byte). Các thanh ghi này có thể được truy cập như mọi thanh ghi khác, chẳng hạn như A, B, R0, R1, R2 v.v. Ví dụ, lệnh "MOV TL0, #4FH" là chuyển giá trị 4FH vào TL0 - byte thấp của bộ định thời 0. Các thanh ghi này cũng có thể được đọc như các thanh ghi khác. Ví dụ "MOV R5, TH0" là lưu byte cao TH0 của Timer 0 vào R5.



Hình 9.1. Thanh ghi của bộ định thời Timer 0

Thanh ghi của bộ Timer 1

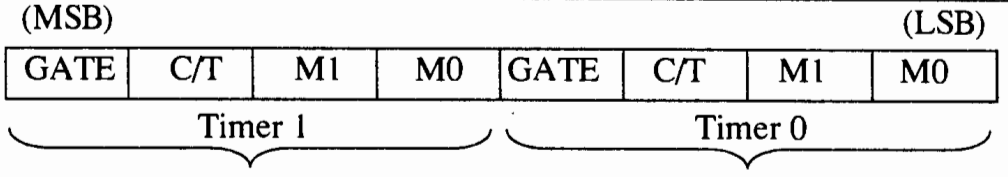
Bộ định thời gian Timer 1 cũng dài 16 bit và thanh ghi 16 bit cũng được chia ra thành hai byte là TL1 và TH1. Các thanh ghi này được truy cập và được đọc giống như các thanh ghi của bộ Timer 0 ở trên.



Hình 9.2. Thanh ghi của bộ định thời Timer 1

Thanh ghi chế độ của bộ định thời TMOD

Cả hai bộ định thời Timer 0 và Timer 1 đều dùng chung một thanh ghi được gọi là TMOD để thiết lập các chế độ làm việc khác nhau của bộ định thời. Thanh ghi TMOD là thanh ghi 8 bit gồm có 4 bit thấp dành cho bộ Timer 0 và 4 bit cao dành cho Timer 1. Trong đó hai bit thấp của chúng dùng để thiết lập chế độ của bộ định thời, còn 2 bit cao dùng để xác định phép toán. Dưới đây chúng ta sẽ tìm hiểu về các phép toán.



GATE Điều khiển cổng khi được thiết lập. Bộ định thời/bộ đếm được mở chỉ khi chân INTx cao và chân điều khiển TRx được lập.
Nếu GATE được xoá, bộ định thời được mở khi TRx được lập

C/T Bộ định thời hoặc bộ đếm bị xoá khi hoạt động (đầu vào từ đồng hồ hệ thống trong). Thiết lập hoạt động bộ đếm (đầu vào từ chân vào Tx)

M1 Chế độ bit 1

M0 Chế độ bit 0

M1	*M0*	*Mode*	*Chế độ hoạt động*
0	0	0	Chế độ bộ định thời 13 bit Bộ định thời/bộ đếm 8 bit, định tỷ lệ trước 5 bit
0	1	1	Chế độ bộ định thời 16 bit, không định tỷ lệ trước
1	0	2	Chế độ 8 bit tự nạp lại Bộ đếm/bộ định thời 8 bit tự nạp lại. THx lưu giá trị sẽ tự nạp vào TLx mỗi khi tràn
1	1	3	Chế độ bộ định thời chia tách

Hình 9.3. Thanh ghi TMOD

Bit M1, M0

M0, M1 là các bit chế độ dùng để chọn chế độ 0, 1, và 2 của các bộ Timer 0, Timer 1. Chế độ 0 là bộ định thời 13 bit, chế độ 1 là bộ định thời 16 bit và chế độ 2 là bộ định thời 8 bit. Chúng ta chỉ tập trung vào các chế độ được sử dụng rộng rãi là chế độ 1 và 2. Các chế độ được thiết lập theo trạng thái của M1 và M0 như sau:

Bit C/T (đồng hồ/bộ định thời)

Là bit của thanh ghi TMOD dùng để xác định bộ định thời được sử dụng làm bộ tạo trễ hay bộ đếm sự kiện. Nếu bit C/T = 0 thì đó là bộ tạo trễ. Nguồn đồng hồ cho chế độ trễ thời gian là tần số thạch anh của 8051. Về sử dụng bộ định thời làm bộ đếm sự kiện sẽ được thảo luận ở phần sau.

Ví dụ 9.1

Hãy xác định chế độ và bộ định thời của các trường hợp sau:

```
a)MOV TMOD,#01H;     b)MOV TMOD,#20H;   c)MOV TMDO,#12H..
```

Giải:

Chuyển số Hexa sang số nhị phân và đối chiếu với hình 9.3 chúng ta có:

```
a)TMOD=00000001,chế độ 1 bộ định thời Timer 0 được chọn.
b)TMOD=00100000,chế độ 2 bộ định thời Timer 1 được chọn.
c)TMOD=00010010,chế độ 2 bộ định thời Timer 0 và chế độ 1
của Timer 1 được chọn.
```

Nguồn xung đồng hồ của bộ định thời

Như đã biết, bộ định thời luôn cần có xung đồng hồ để giữ nhịp. Vậy nguồn xung đồng hồ cho các bộ định thời của 8051 là gì? Nếu C/T = 0 thì tần số thạch anh trên chip 8051 được làm nguồn cho đồng hồ của bộ định thời. Điều đó có nghĩa là giá trị của tần số thạch anh của 8051 quyết định tốc độ đồng hồ của các bộ định thời 8051. Tần số của bộ định thời luôn bằng 1/12 tần số của thạch anh trên chip 8051. Xem ví dụ 9.2.

Ví dụ 9.2

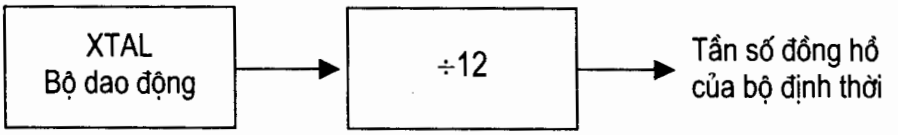
Hãy tìm tần số đồng hồ và chu kỳ của bộ định thời của các hệ thống xây dựng trên 8051 với tần số thạch anh như sau:

```
a) 12MHz
```

b) 16MHz

c) 11,0592MHz

Giải:



a) $\frac{1}{12} \times 12\text{MHz} = 1\text{MHz}$ và $T = \frac{1}{1/1\text{MHz}} = 1\mu s$

b) $\frac{1}{12} \times 16\text{MHz} = 1{,}111\text{Mz}$ và $T = \frac{1}{1{,}333\text{MHz}} = 0{,}75\mu s$

c) $\frac{1}{12} \times 11{,}0592\text{MHz} = 921{,}6\text{kHz}$ và $T = \frac{1}{0{,}9216\text{MHz}} = 1{,}085\mu s$

Các hệ thống xây dựng trên 8051 thường có tần số thạch anh từ 10 đến 40MHz, song ở đây, chúng ta chỉ quan tâm đến tần số thạch anh 11,0592MHz. Sở dĩ có số lẻ như vậy là do liên quan đến tốc độ baud truyền thông nối tiếp của 8051. Tần số XTAL = 11,0592MHz cho phép hệ 8051 truyền thông không có lỗi với IBM PC. Vấn đề này chúng ta sẽ tìm hiểu ở chương 10.

GATE

Một bit khác của thanh ghi TMOD là bit cổng GATE. Như ở hình 9.3 có thể thấy, cả hai bộ định thời Timer 0 và Timer 1 đều có bit GATE. Vậy chức năng của bit GATE là gì? Mỗi bộ định thời đều có cách khởi động và dừng khác nhau. Một số bộ định thời thực hiện điều này bằng phần mềm, một số khác bằng phần cứng và một số khác nữa thì kết hợp cả phần cứng lẫn phần mềm. Chính bộ định thời của 8051 dùng phương pháp kết hợp. Khởi động và dừng bộ định thời được thực hiện bằng phần mềm nhờ các bit khởi động bộ định thời TR (Timer Start) là TR0 và TR1. Lệnh thực hiện khởi động và dừng Timer 1 tương ứng là "SETB TR1" và "CLR TR1" còn đối với Timer 0 là "SETB TR0" và "CLR TR0". Các lệnh này thực hiện khởi động và dừng các bộ định thời khi bit GATE = 0. Nếu dùng phần cứng từ bên ngoài để khởi động và dừng bộ định thời thì cần đặt bit GATE = 1. Tuy nhiên, để tránh nhầm lẫn, ở đây chúng ta giả thiết GATE = 0, có nghĩa là không sử dụng chế độ khởi động và dừng bộ định thời bằng phần cứng mà bằng phần mềm. Cách viết lệnh chung cho hai bộ Timer là "SETB TRx" và

"CLR TRx". Việc sử dụng phần cứng ngoài để khởi động và dừng bộ định thời chúng ta sẽ đề cập tới ở chương 11 khi bàn về các ngắt.

Ví dụ 9.3

Tìm giá trị của TMOD nếu muốn lập trình bộ Timer 0 ở chế độ 2 sử dụng thạch anh XTAL 8051 làm nguồn đồng hồ và sử dụng lệnh để khởi động và dừng bộ định thời.

Giải:

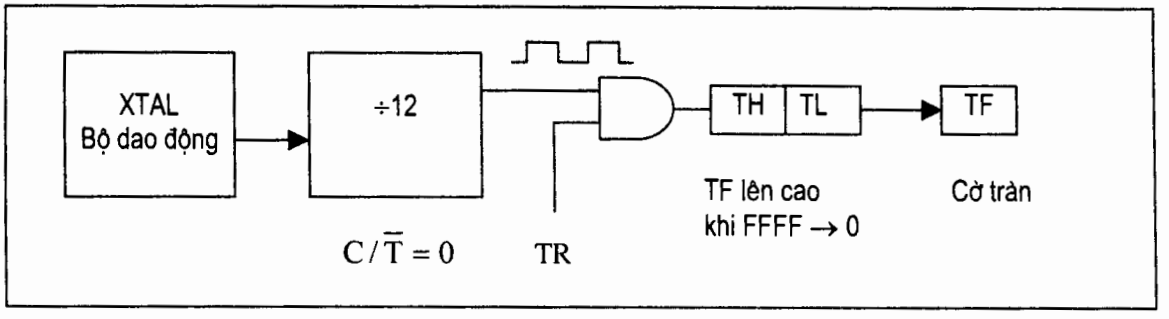
TMOD = 0000 0010: Timer 0, chế độ 2
C/T = 0: Dùng đồng hồ từ nguồn XTAL
GATE = 0 Dùng phần mềm để khởi động và dừng bộ định thời.

Chế độ 1 và chế độ 2 là hai chế độ được sử dụng khá phổ biến, vì vậy ở đây chúng ta sẽ nghiên cứu chi tiết hơn.

Lập trình chế độ 1 (Mode1)

Dưới đây là những đặc trưng và những phép toán của Mode1:

1. Đó là bộ định thời 16 bit, do vậy các giá trị trong khoảng từ 0000 đến FFFFH có thể được nạp vào thanh ghi TL và TH của bộ định thời.
2. Sau khi TL và TH được nạp giá trị ban đầu 16 bit thì bộ định thời được khởi động. Thực hiện điều này nhờ lệnh "SETB TR0" cho Timer 0 và "SETB TR1" cho Timer 1.
3. Bộ định thời sau khi được khởi động thì bắt đầu tiến hành đếm tăng. Bộ định thời đếm lên cho đến khi đạt được giới hạn FFFFH. Khi đó bộ định thời sẽ quay vòng từ FFFFH về 0000 và bật cờ bộ định thời TF (Timer Flag) lên mức cao. Cờ bộ định thời này có thể kiểm tra được. Khi cờ bộ định thời được thiết lập thì để dừng bộ định thời bằng phần mềm có thể sử dụng lệnh "CLR TR0" cho Timer 0 hoặc "CLR TR1" cho Timer 1. Cũng nên nhắc lại là mỗi bộ định thời đều có cờ TF riêng: TF0 của Timer 0 và TF1 của Timer 1.
4. Bộ định thời sau khi đạt giá trị giới hạn thì thực hiện quay vòng về 0. Để lặp lại quá trình đếm của bộ định thời, các thanh ghi TH và TL phải được nạp lại giá trị ban đầu và cờ TF cần được xoá về 0.



Các bước lập trình ở Mode 1

Để tạo ra độ trễ thời gian khi dùng chế độ 1 của bộ định thời thì cần phải thực hiện các bước dưới đây:

1. Nạp giá trị cho thanh ghi TMOD xác định bộ định thời nào (Timer 0 hay Timer 1) và chế độ nào được chọn.
2. Nạp giá trị đếm ban đầu cho các thanh ghi TL và TH.
3. Khởi động bộ định thời.
4. Kiểm tra trạng thái bật của cờ bộ định thời TF bằng lệnh "JNB TFx, đích". Thoát vòng lặp khi TF được bật lên cao.
5. Dừng bộ định thời.
6. Xoá cờ TF cho vòng kế tiếp.
7. Quay trở lại bước 2 để nạp lại TL và TH.

Để tính toàn thời gian trễ chính xác và tần số sóng vuông được tạo ra trên chân P1.5 thì ta cần biết tần số XTAL (xem ví dụ 9.5).

Từ ví dụ 9.6 có thể lập công thức tính độ trễ sử dụng chế độ 1 (16 bit) của bộ định thời ở tần số thạch anh XTAL = 11, 0592MHz (xem hình 9.4). Có thể dùng máy tính ở thư mục Accessory của Microsoft Windows để tìm các giá trị TH và TL: Máy tính này hỗ trợ phép tính theo số thập phân, nhị phân và thập lục.

a) Tính theo số Hexa	**b) Tính theo số thập phân**
(FFFF-YYXX+1)x1,085μs trong đó YYXX là giá trị khởi tạo của TH,TL tương ứng. Lưu ý rằng các giá trị YYXX là số Hexa.	Đổi giá trị YYXX của TH,TL về số thập phân NNNNN, sau đó tính (65536-NNNNN)x1,085 μs

Hình 9.4. Tính thời gian trễ ở tần số XTAL = 11, 0592MHz

Ví dụ 9.4

Chương trình dưới đây tạo ra sóng vuông với độ đầy xung 50% trên chân P1.5. Sử dụng bộ định thời Timer 0 để tạo trễ thời gian. Hãy phân tích chương trình này.

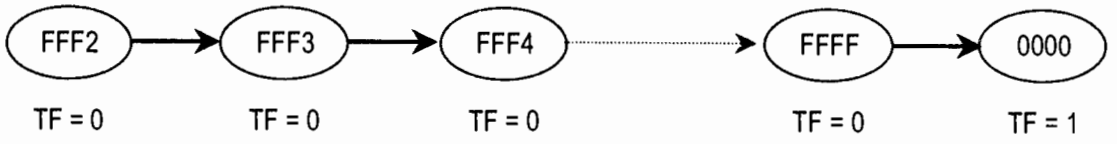
```
       MOV    TMOD,#01      ;Chọn Timer 0 chế độ 1(16 bit)
HERE:  MOV    TL0,#0F2H     ;TL0=F2H,byte thấp
       MOV    TH0,#0FFH     ;TH0=FFH,byte cao
       CPL    P1.5          ;Sử dụng chân P1.5
       ACALL  DELAY
       SJMP   HERE          ;Nạp lại TH,TL
;--- tạo trễ dùng Timer 0
DELAY:
       SETB   TR0           ;Khởi động Timer 0
AGAIN: JNB    TF0,AGAIN     ;Kiểm tra cờ bộ định thời
                            ;cho đến khi quay vòng
       CLR    TR0           ;Dừng Timer 0
       CLR    TF0           ;Xoá cờ bộ định thời 0
       RET
```

Giải:

Các bước được thực hiện ở chương trình trên như sau:

1. Nạp TMOD.
2. Nạp giá trị FFF2H vào TH0 - TL0.
3. Chọn chân P1.5 để tạo xung.
4. Gọi chương trình con tạo trễ DELAY.
5. Trong chương trình con DELAY, dùng lệnh "SETB TR0" để khởi động bộ định thời Timer 0.
6. Bộ Timer 0 thực hiện đếm tăng theo xung đồng hồ từ máy phát thạch anh. Bộ định thời đếm tăng qua các trạng thái FFF3, FFF4 ... cho đến khi đạt giá trị FFFFH và thêm 1 xung nữa là nó quay về 0 và bật cờ bộ định thời TF0 = 1. Tại thời điểm này thì lệnh JNB sẽ thoát khỏi vòng lặp và rẽ nhánh xuống thực hiện lệnh sau JNB.
7. Bộ Timer 0 được dừng bởi lệnh "CLR TR0". Chương trình con DELAY kết thúc và quá trình được lặp lại.

Lưu ý rằng để lặp lại quá trình trên cần nạp lại các thanh ghi TH, TL và khởi động lại bộ định thời.



Ví dụ 9.5

Hãy tính thời gian trễ do chương trình con DELAY tạo ra ở ví dụ 9.4 với giả thiết tần số XTAL = 11,0592MHz.

Giải:

Bộ định thời làm việc ở tần số đồng hồ bằng 1/12 tần số XTAL, do vậy ta có $\frac{11,0592}{12} = 0,9216\text{MHz}$ là tần số của bộ định thời. Kết quả là mỗi nhịp xung đồng hồ có chu kỳ $T = \frac{1}{0,9216\text{MHz}} = 1,085\mu s$. Như vậy, bộ Timer 0 tiến hành đếm tăng sau mỗi 1,085μs và độ trễ được xác định bằng *số đếm* ×1,085μs.

Số đếm bằng FFFFH - FFF2H = 0DH (=13D). Tuy nhiên, ta phải cộng 1 vào 13 vì cần thêm một nhịp đồng hồ để thực hiện quay vòng từ FFFFH về 0 và bật cờ TF. Do vậy, ta có 14 × 1,085μs = 15,19μs cho nửa chu kỳ, nên cả chu kỳ là T = 2 × 15,19μs = 30, 38μs là thời gian trễ do bộ định thời tạo ra.

Ví dụ 9.6

Hãy tính tần số của xung vuông được tạo ra trên chân P1.5 ở ví dụ 9.5.

Giải:

Ở ví dụ 9.5, khi tính thời gian trễ, chúng ta không xét đến thời gian thực hiện các lệnh khác của vòng lặp. Để tính chính xác hơn, cần bổ sung thêm các chu kỳ thời gian của các lệnh trong vòng lặp bằng cách sử dụng bảng phụ lục chu kỳ máy A-1.

```
                                    Số chu kỳ máy
HERE:  MOV    TL0, #0F2H            2
       MOV    TH0, #0FFH            2
```

```
          CPL    P1-5                1
          ACALL  DELAY               2
          SJMP   HERE                2
;--- Tạo trễ bằng timer 0
   DELAY:
          SETB   TR0                 1
   AGAIN: JNB    TF0,AGAIN           1
          CLR    TR0                 1
          CLR    TF0                 1
          RET                        1
                        Tổng cộng    27
```

T = (2 × 27 × 1.085µs và F = 17067.75Hz).

Tổng số chu kỳ đã bổ xung là x7 nên chu kỳ thời gian trễ là T = 2 × 27 × 1.085µs = 58,59µs và tần số là F = 17067,75Hz.

Ví dụ 9.7

Tìm độ trễ do Timer 0 tạo ra ở đoạn chương trình sau, dùng cả hai phương pháp ở hình 9.4 và bỏ qua thời gian thực hiện các lệnh khác của vòng lặp.

```
       CLR    P2.3          ;Xoá P2.3
       MOV    TMOD,#01      ;Chọn Timer 0,chế độ 1(16 bit)
HERE:  MOV    TL0,#3EH      ;TL0=3EH,byte thấp
       MOV    TH0,#0B8G     ;TH0=B8H,byte cao
       SETB   P2.3          ;Bật P2.3 lên cao
       SETB   TR0           ;Khởi động Timer 0
AGAIN: JNB    TF0,AGAIN     ;Hiển thị cờ bộ định thời TF0
       CLR    TR0           ;Dừng bộ định thời
       CLR    TF0           ;Xoá cờ bộ định thời vòng sau
       CLR    P2.3
```

Giải:

a) Độ trễ được tạo ra trong đoạn chương trình trên là:

(FFFF - B83E + 1) = 47C2H = 18370D

Nên độ trễ là: 18370 × 1,085µs = 19, 93145µs.

b) Vì TH - TL = B83EH = 47166D, nên ta có 65536 - 47166 = 18370.

Như vậy, bộ định thời thực hiện đếm từ B83EH đến FFFF và được cộng

thêm 1 để quay vòng về 0, tổng cộng tất cả là 18370 chu kỳ đồng hồ. Do vậy ta có độ rộng xung là 18370 × 1,085μs = 19,93145ms.

Ví dụ 9.8

Thay đổi lại giá trị của TH và TL ở ví dụ 9.7 để nhận được độ trễ thời gian lớn nhất. Hãy tính độ trễ theo miligiây. Cần xét thêm thời gian thực hiện các lệnh vòng lặp.

Giải:

Để có trễ lớn nhất, ta cần đặt TH và TL bằng 0. Như vậy, bộ định thời đếm được từ 0000 đến FFFFH và sau đó quay qua về 0.

```
        CLR    P2.3          ;Xoá P2.3
        MOV    TMOD,#01      ;Chọn Timer 0,chế độ 1(16 bit)
HERE:   MOV    TL0,#0        ;Đặt TL0=0,byte thấp
        MOV    TH0,#0        ;Đặt TH0=0,byte cao
        SETB   P2.3          ;Bật P2.3 lên cao
        SETB   TR0           ;Khởi động bộ Timer 0
AGAIN:  JNB TF0,AGAIN        ;Hiển thị cờ bộ định thời TF0
        CLR    TR0           ;Dừng bộ định thời.
        CLR    TF0           ;Xoá cờ TF0
        CLR    P2.3
```

Cho TH=0 và TL = 0 thì bộ định thời đếm tăng từ 0000 đến FFFFH, sau đó quay về 0 và bật cờ bộ định thời TF. Kết quả đếm dược 65536 giá trị. Do vậy, *Độ trễ* = (65536 - 0) × 1.085μs = 71.1065μs.

Ở ví dụ 9.7 và 9.8 không cần nạp lại giá trị TH và TL vì chúng ta đang làm việc với xung đơn. Ở ví dụ 9.9 cần nạp lại các thanh ghi đó.

Ví dụ 9.9

Chương trình dưới đây tạo một sóng vuông trên chân P1.5 bằng cách dùng bộ Timer 1 để tạo ra độ trễ thời gian. Hãy tìm tần số của sóng vuông nếu tần số XTAL = 11.0592 MHz bỏ qua thời gian thực hiện các lệnh khác của vòng lặp.

```
        MOV    TMOD,#01H     ;Chọn Timer 0,chế độ 1(16 bit)
```

```
HERE:  MOV    TL1,#34H      ;Đặt byte thấp TL1=34H
       MOV    TH1,#76H      ;Đặt byte cao TH1=76H
                            ;(giá trị bộ định thời là 7634H)
       SETB   TR1           ;Khởi động Timer 1
AGAIN: JNB    TF1,BACK      ;Lặp cho đến khi quay vòng
       CLR    TR1           ;Dừng bộ định thời
       CPL    P1.5          ;Lấy bù chân P1.5 để tạo xung
       CLR    TF            ;Xoá cờ bộ định thời
       SJMP   AGAIN         ;Nạp lại bộ định thời do chế độ
                            ;1 không tự động nạp lại.
```

Giải:

Trong chương trình trên, cần lưu ý đến đích của lệnh SJMP. Ở chế độ 1, sau mỗi vòng lặp cần nạp lại thanh ghi TH và TL để có được sóng liên tục. Dưới đây là kết quả tính toán:

Ta có FFFFH - 7634H = 89CBH + 1 = 89CCH và 89CCH = 35276 là số lần đếm xung đồng hồ, nên độ trễ là 35276 × 1.085μs = 38274ms và tần số là $\frac{1}{38274}$(Hz) = 26127Hz.

Xác định giá trị cần nạp vào bộ định thời

Giả sử chúng ta đã biết thời gian trễ cần có, vậy giá trị nạp cho các thanh ghi TH và TL sẽ là bao nhiêu? Chúng ta xét ví dụ sau với giả thiết tần số dao động thạch anh của hệ 8051 là 11. 0592MHz.

Từ ví dụ 9.10 có thể thấy các bước tìm giá trị của thanh ghi TH và TL như sau:

1. Chia thời gian trễ cho 1.085μs.
2. Tính 65536 - n, trong đó n là giá trị thập phân nhận được ở bước 1.
3. Chuyển kết quả của bước 2 sang số Hexa, trong đó số yyxx là giá trị Hexa ban đầu cần nạp vào các thanh ghi bộ định thời.
4. Đặt TL = xx và TH = yy.

Ví dụ 9.10

Giả sử tần số XTAL = 11.0592MHz. Hãy xác định giá trị nạp vào các thanh ghi TH và TL nếu cần trễ thời gian là 5μs. Hãy viết chương trình cho bộ Timer 0 để tạo ra xung trên chân P2.3 có độ rộng 5μs.

Giải:

Vì tần số XTAL = 11.0592MHz nên bộ đếm tăng giá trị sau mỗi chu kỳ 1.085µs. Để có độ rộng xung 5ms thì số nhịp đồng hồ là: n = 5ms/1.085µs = 4608. Giá trị cần nạp vào TL và TH được tính theo cách: lấy 65536 trừ đi 4608 bằng 60928, rồi đổi số này sang số Hexa được EE00H. Do vậy, giá trị nạp vào TH là EE và TL là 00.

```
        CLR    P2.3          ;Xoá bit P2.3
        MOV    TMOD,#01      ;Chọn Timer 0,chế độ 1(16 bit)
HERE:   MOV    TL0,#0        ;Nạp TL=00
        MOV    TH0,#EEH      ;Nạp TH=EEH
        SETB   P2.3          ;Bật P2.3 lên cao
        SETB   TR0           ;Khởi động bộ định thời Timer 0
AGAIN:JNB      TF0,AGAIN     ;Lặp cho đến khi TF0 quay về 0
        CLR    TR0           ;Dừng bộ định thời
        CLR    TF0           ;Xoá cờ TF0 cho vòng sau
```

Ví dụ 9.11

Giả sử tần số XTAL là 11,0592MHz, hãy viết chương trình tạo ra một sóng vuông tần số 2kHz trên chân P2.5.

Giải:

Ví dụ này tương tự với ví dụ 9.10 song khác về chân. Các bước thực hiện như sau:

a) $T = \frac{1}{f} = \frac{1}{2kHz} = 500\mu s$ là chu kỳ của sóng vuông.

b) Khoảng thời gian cao và thấp là 0.5T bằng 250µs.

c) Số nhịp cần trong thời gian đó là $\frac{250\mu s}{1,085\mu s} = 230$ và giá trị cần nạp vào các thanh ghi là 65536 - 230 = 65306D = FF1AH.

d) Như vậy, giá trị nạp vào TL là 1AH và TH là FFH.

Chương trình được viết như sau:

```
        MOV    TMOD,#10H     ;Chọn Timer 0,chế độ 1 (16 bit)
AGAIN: MOV     TL1,#1AH      ;Gán giá trị byte thấp TL1=1AH
```

```
      MOV   TH1,#0FFH    ;Gán giá trị byte cao TH1=FFH
      SETB  TR1          ;Khởi động Timer 1
BACK: JNB   TF1,BACK     ;Giữ cho đến khi Timer về 0
      CLR   TR1          ;Dừng bộ định thời
      CPL   P1.5         ;Bù bit P1.5
      CLR   TF1          ;Xoá cờ TF1
      SJMP  AGAIN        ;Nạp lại bộ định thời
```

Ví dụ 9.12

Giả sử XTAL=11.0592 Mhz. Hãy viết chương trình tạo sóng vuông tần số 50 Hz ở chân P2.3.

Giải:

Các bước thực hiện như sau:

a) Tính chu kỳ sóng vuông: $T = \frac{1}{50Hz} = 20\,ms$.

b) Tính thời gian nửa chu kỳ cho phần cao: 0.5T=10 ms.

c) Tính số nhịp đồng hồ: $n = \frac{10\,ms}{1{,}085\mu s} = 9216$.

d) Tính giá trị cần nạp vào TH và TL: 65536 - 9216 = 56320 chuyển về dạng Hexa là DC00H, như vậy TH = DCH và TL = 00H.

```
      MOV   TMOD,#10H    ;Chọn Timer 0,chế độ 1(16 bit)
AGAIN:MOV   TL1,#00      ;Gán giá trị byte thấp TL1=00
      MOV   TH1,#0DHCH   ;Gán giá trị byte cao TH1=DC
      SETB  TR1          ;Khởi động Timer 1
BACK: JNB   TF1,BACK     ;Lặp cho đến khi quay về 0
      CLR   TR1          ;Dừng bộ định thời
      CPL   P2.3         ;Bù bit P1.5
      CLRTF1             ;Xoá cờ TF1
      SJMP AGAIN         ;Nạp lại bộ định thời
```

Tạo độ trễ thời gian lớn

Từ các ví dụ trên có thể thấy, thời gian trễ tạo ra phụ thuộc vào hai yếu tố:

- Tần số thạch anh XTAL.
- Thanh ghi 16 bit của bộ định thời ở chế độ 1.

Cả hai yếu tố này nằm ngoài khả năng điều chỉnh của lập trình viên 8051. Ví

dụ, độ trễ thời gian lớn nhất có thể đạt được khi đặt cả TH và TL bằng 0. Nhưng như vậy là chưa đủ nếu cần có thời gian trễ lớn hơn. Trước khi xem xét ví dụ minh hoạ nội dung này, chúng ta hãy làm quen với việc sử dụng Calculator của Windows.

Dùng Calculator của Windows để tính TH và TL

Calculator (bàn tính con) của Windows có ngay trong phần mềm Windows rất dễ sử dụng để tính các giá trị cho TH và TL. Giả sử tìm giá trị cho TH và TL với độ trễ thời gian lớn là 35.000 nhịp đồng hồ với chu kỳ 1,085μs, ta thực hiện các bước như sau:

1. Chọn máy tính Calculator từ Windows và đặt chế độ tính về số thập phân Decimal.
2. Nhập từ bàn phím số 35.000.
3. Chuyển về chế độ Hexa, Calculator cho giá trị 88B8H.
4. Chọn +/- để nhận số đổi dấu - 35.000 dạng thập phân và chuyển về dạng Hexa là 7748H.
5. Hai số Hexa cuối là cho TL = 48 và hai số Hexa tiếp theo là cho TH = 77. Ta bỏ qua các số F ở phía bên trái trên Calculator vì số cần có chỉ 16 bit.

Ví dụ 9.13

Hãy kiểm tra chương trình sau và tìm độ trễ thời gian tính theo giây, bỏ qua thời gian thực hiện các lệnh trong vòng lặp.

```
        MOV    TMOD,#10H     ;Chọn Timer 1,chế độ 1(16 bit)
AGAIN:  MOV    R3,#200       ;Chọn bộ đếm độ giữ chậm lớn
        MOV    TL1,#08       ;Nạp byte thấp TL1=08
        MOV    TH1,#08       ;Nạp byte cao TH1=01
        SETB   TR1           ;Khởi động Timer 1
BACK:   JNB    TF1,BACK      ;Giữ cho đến khi quay về 0
        CLR    TR1           ;Dừng bộ định thời.
        CLR    TF1           ;Xoá cờ bộ định thời TF1
        DJNZ   R3,AGAIN      ;Nếu R3≠0 thì nạp lại
```

Giải:

TH-TL=0108H=264D và 65536-264=65272. Do vậy, 65272x1,085 μs = 70,820 ms, và 200x70,820 ms = 14,164024 s.

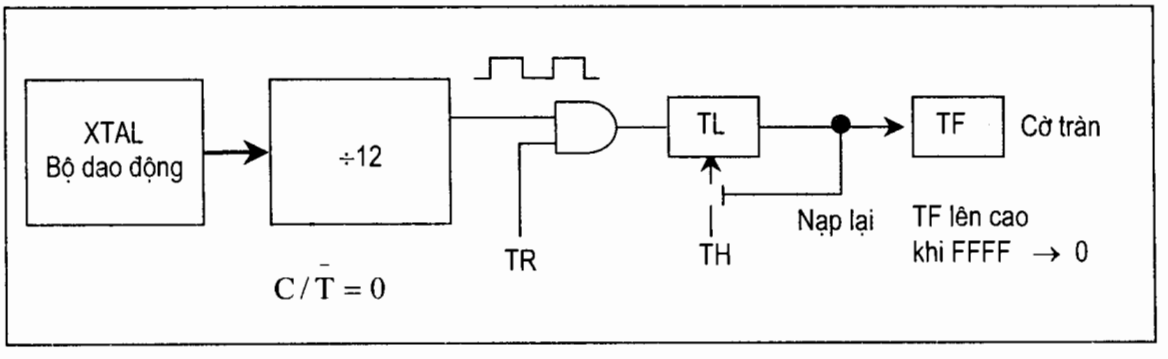
Chế độ 0

Chế độ 0 hoàn toàn giống chế độ 1, chỉ có điểm khác đó là bộ định thời 16 bit được thay bằng 13 bit. Bộ đếm 13 bit có thể lưu được trong TH - TL các giá trị từ 0000 đến 1FFFF. Do vậy, khi bộ định thời đạt được giá trị cực đại là 1FFFH thì nó sẽ quay trở về 0000 và cờ TF được bật.

Lập trình chế độ 2

Đặc trưng và các phép tính của chế độ 2:

1. Đó là một bộ định thời 8 bit, do vậy chỉ cho phép các giá trị từ 00 đến FFH được nạp vào thanh ghi TH của bộ định thời.
2. Sau khi TH được nạp giá trị 8 bit thì 8051 sao nội dung đó vào TL và bộ định thời được khởi động. Lệnh thực hiện khởi động là "SETB TR0" đối với Timer 0 và "SETB TR1" đối với Timer 1 giống như ở chế độ 1.
3. Bộ định thời sau khi được khởi động thì bắt đầu đếm tăng bằng cách tăng thanh ghi TL. Bộ định thời đếm cho đến khi đạt giá trị giới hạn là FFH. Khi quay vòng từ FFH trở về 00 thì cờ bộ định thời TF được thiết lập. Cụ thể, đó là hai cờ TF0 và TF1 tương ứng với bộ định thời được sử dụng là Timer 0 hay Timer 1.



4. Khi thanh ghi TL quay từ FFH trở về 00, cờ TF được bật lên 1 thì thanh ghi TL được tự động nạp lại với giá trị ban đầu được giữ ở thanh ghi TH. Để lặp lại quá trình, đơn giản là chỉ việc xoá cờ TF và để bộ định thời tự làm việc mà không cần lập trình viên can thiệp hay nạp lại gía trị ban đầu. Vì lý do đó, chế độ 2 được gọi là chế độ tự nạp, và khác với chế độ 1, ở đó lập trình viên cần nạp lại các thanh ghi TH và TL.

Nên nhắc lại là, chế độ 2 là bộ định thời 8 bit. Tuy nhiên, chế độ này có khả năng tự nạp lại. Khi tự nạp lại thì TH không thay đổi mà vẫn giữ nguyên giá trị ban đầu, còn TL được nạp lại giá trị lấy từ TH. Chế độ này có nhiều ứng dụng, trong đó có cả thiết lập tần số baud trong truyền thông nối tiếp như ta sẽ đề cập đến ở chương 10.

Các bước lập trình cho chế độ 2

Các bước tạo thời gian trễ sử dụng chế độ 2 của bộ định thời như sau:

1. Nạp thanh ghi giá trị TMOD để báo bộ định thời nào (Timer 0 hay Timer 1) và chế độ làm việc nào được chọn.
2. Nạp vào thanh ghi TH giá trị đếm ban đầu.
3. Khởi động bộ định thời.
4. Duy trì kiểm tra cờ bộ định thời TF bằng lệnh "JNB TFx, đích" để xem cờ đã được bật chưa. Thoát khỏi vòng lặp khi TF lên cao.
5. Xoá cờ TF.
6. Quay trở lại bước 4 vì chế độ 2 là chế độ tự nạp lại.

Ví dụ 9.14 minh hoạ các bước thực hiện ở chế độ 2. Để có được độ trễ lớn chúng ta có thể dùng nhiều thanh ghi như được trình bày ở ví dụ 9.15.

Ví dụ 9.14

Giả sử tần số XTAL = 11.0592MHz. Hãy tìm :

a) Tần số của sóng vuông được tạo ra trên chân P1.0 trong chương trình sau.

b) Tần số nhỏ nhất có thể có được bằng chương trình này và giá trị TH cần có là bao nhiêu?

```
        MOV    TMOD,#20H     ;Chọn Timer 1/chế độ 2
        MOV    TH1,#5        ;TH1=5
        SETB   TR1           ;Khởi động Timer 1
BACK:   JNB    TF1,BACK      ;giữ cho đến khi về 0
        CPL    P1.0          ;Dừng bộ định thời
        CLR    TF1           ;Xoá cờ bộ định thời TF1
        SJMP   BACK          ;Chế độ 2 tự động nạp lại.
```

Giải:

a) Trước hết cần lưu ý đến đích của lệnh SJMP. Ở chế độ 2 ta không cần phải nạp lại TH vì đó là chế độ tự nạp. Lấy (256 - 05) ×1.085μs = 251×1.085μs = 272.33μs là phần cao của xung. Cả chu kỳ xung là T = 544.66μs và tần số là $\frac{1}{T}$ = 1,83597 kHz.

b) Để có được tần số nhỏ nhất, tức chu kỳ T lớn nhất, thì TH = 00. Khi đó T = 2 × 256 × 1.085 μs = 555.52μs và tần số nhỏ nhất sẽ là 1/T=1,8 kHz.

Ví dụ 9.15

Hãy tìm tần số của xung vuông được tạo ra trên chân P1.0.

Giải:

```
       MOV    TMOD,#2H      ;Timer 0,chế độ 1 (8 bit, tự nạp)
AGAIN: MOV    TH0 #0        ;Nạp TH0=0
       MOV    R5,#250       ;Đếm cho độ trễ lớn
       ACALL  DELAY
       CPL    P1.0
       SJMP   AGAIN
DELAY: SETB   TR0           ;Khởi động Timer 0
BACK:  JNB    TF1,BACK      ;Giữ cho đến khi quay về 0
       CLR    TR0           ;Dừng Timer 0.
       CLR    TF0           ;Xoá cờ TF0 cho vòng sau.
       DJNZ   R5,DELAY
       RET
```

T=2×(250×256×1.085μs)=1.38.88ms và =72 Hz.

Ví dụ 9.16

Giả sử chế độ 2 được chọn. Hãy xác định giá trị (dạng Hexa) cần nạp vào TH cho các trường hợp sau:

```
a)MOV  TH1,#200;     b)MOV  TH0,#-60
c)MOV  TH1,#-3;      d)MOV  TH1,#-12
e)MOV  TH0,#-48
```

Giải:

Chúng ta có thể dùng Calculator của Windows để kiểm tra kết quả. Đặt Calculator ở chế độ Decimal và nhập số 200. Sau đó chọn Hexa, ấn +/- để nhận giá trị của TH. Nên lưu ý là chúng ta chỉ sử dụng hai chữ số và bỏ qua các số bên trái vì dữ liệu đúng chỉ có 8 bit. Kết quả ta nhận được như sau:

Số thập phân	*Số bù hai (giá trị TH)*
-200	38H
-60	C4H
-3	FDH
-12	F4H
-48	D0H

Trình hợp dịch và các số âm

Ở chế độ 2 bộ định thời là 8 bit, nên có thể để cho trình hợp dịch tính giá trị TH. Ví dụ, ở lệnh "MOV TH0, # - 100" thì trình hợp dịch sẽ tính 100D = 9CH và gán TH = 9CH. Cách này làm cho công việc của lập trình viên nhẹ nhàng hơn.

Ví dụ 9.17

Hãy xác định:

a) Tần số sóng vuông được tạo ra trong đoạn chương trình dưới đây.

b) Độ đầy xung của sóng này.

```
       MOV    TMOD,#2H      ;Chọn bộ Timer 0/chế độ 2
       MOV    TH0,#-150     ;Nạp TH0=6AH là bù 2 của -150
       SETB   TR1           ;Khởi động Timer 1
AGAIN: SETB   P1.3          ;P1.3=1
       ACALL  DELAY
       ACALL  P1.3          ;P1.3=0
       ACALL  DELAY
       SJMP   AGAIN
       SETB   TR0           ;Khởi động Timer 0
BACK:  JNB    TF0,BACK      ;Giữ cho đến khi quay về 0
       CLR    TR0           ;Dừng Timer 0
       CLR    TF0           ;Xoá cờ TF cho vòng sau.
       RET
```

Giải:

Để xác định giá trị nạp cho TH ở chế độ 2 thì nên để trình hợp dịch thực hiện việc chuyển đổi số âm được nhập vào, và nhờ vậy công việc tính toán của lập trình viên có nhẹ đi. Vì số xung đồng hồ được sử dụng là 150, nên thời gian trễ của chương trình con DELAY là 150 × 1.085μs và tần số là f=1/T=2,048 kHz.

Trong nhiều ví dụ tính thời gian trễ nêu trên, chúng ta đã bỏ qua thời gian thực hiện các lệnh trong vòng lặp. Dĩ nhiên, để tính toán chính xác thì những thời gian trên cần được xét đến.

Chúng ta đã nghiên cứu bộ định thời của 8051 dùng làm bộ tạo thời gian trễ. Tuy nhiên, bộ định thời còn có một ứng dụng thú vị nữa, đó là làm bộ đếm sự kiện. Chúng ta sẽ tiếp tục tìm hiểu nội dung này.

9.2 LẬP TRÌNH CHO BỘ ĐẾM

Bộ định thời của 8051 ngoài việc dùng làm bộ tạo trễ thời gian còn có thể được dùng làm bộ đếm các sự kiện bên ngoài 8051. Phương pháp lập trình bộ định thời được giới thiệu ở mục trước cũng được áp dụng cho bộ đếm sự kiện, ngoại trừ nguồn tần số. Khi bộ định thời/bộ đếm là bộ định thời thì nguồn tần số là tần số thạch anh của 8051, còn nếu làm bộ đếm, thì nguồn xung để tăng nội dung các thanh ghi TH và TL được lấy từ bên ngoài 8051. Nói chung, các thanh ghi TMOD, TH, TL, cũng như các chế độ làm việc của bộ định thời và bộ đếm đều như nhau.

Bit C/T của thanh ghi TMOD

Như giới thiệu ở trên, bit C/T của thanh ghi TMOD có nhiệm xác định nguồn xung đồng hồ cấp cho bộ định thời. Nếu bit C/T = 0 thì bộ định thời nhận xung đồng hồ từ bộ dao động thạch anh của 8051. Ngược lại, nếu C/T = 1 thì bộ định thời được sử dụng làm bộ đếm và nhận các xung đồng hồ từ nguồn bên ngoài của 8051. Do vậy, với bit C/T = 1 thì bộ đếm thực hiện đếm tăng, khi các xung được đưa đến chân 14 và 15. Các chân này có tên là T0 (đầu vào của bộ định thời Timer 0), T1 (đầu vào của bộ Timer 1) và cả hai chân này thuộc về cổng P3. Đối với Timer 0, khi C/T = 1 thì chân P3.4 cấp xung đồng hồ và bộ đếm sẽ tăng trạng thái mỗi khi có xung đồng hồ đến chân này. Tương tự, đối với Timer 1 thì khi C/T = 1, xung đồng hồ đến P3.5 sẽ làm tăng bộ đếm lên 1.

Bảng 9.1. Chân cổng P3 được dùng cho Timer 0 và Timer 1

Chân	Chân cổng	Chức năng	Mô tả
14	P3.4	T0	Đầu vào ngoài của bộ đếm 0
15	P3.5	T1	Đầu vào ngoài của bộ đếm 1

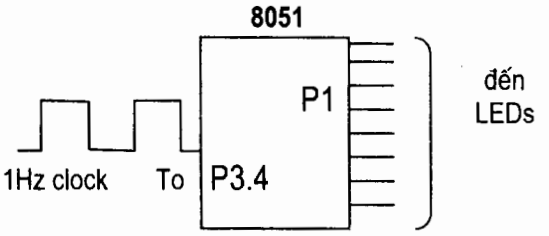
Ví dụ 9.18

Giả sử xung đồng hồ được cấp tới chân T1. Hãy viết chương trình cho bộ đếm 1 ở chế độ 2 để đếm các xung và hiển thị trạng thái của số đếm TL1 trên cổng P2.

Giải:

```
        MOV    TMOD,#01100000B ;Chọn bộ đếm 1, chế độ 2, bit
                               ;C/T=1 xung ngoài
```

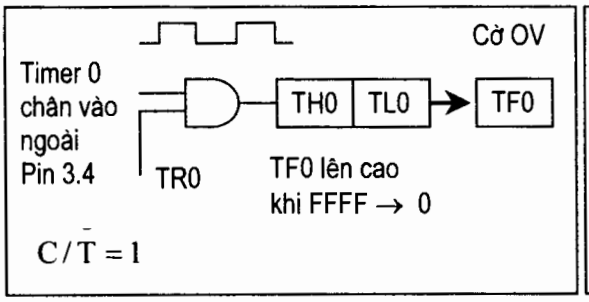
```
        MOV    TH1,#0          ;Xoá TH1
        SETB   P3.5            ;Lấy đầu vào T1
AGAIN:  SETB   TR1             ;Khởi động bộ đếm
BACK:   MOV    A,TL1           ;Lấy bản sao số đếm TL1
        MOV    P2,A            ;Đưa TL1 hiển thị ra cổng P2
        JNB    TF1,Back        ;Duy trì nó nếu TF=0
        CLR    TR1             ;Dừng bộ đếm
        CLR    TF1             ;Xoá cờ TF
        SJMP   AGAIN           ;Tiếp tục thực hiện
```



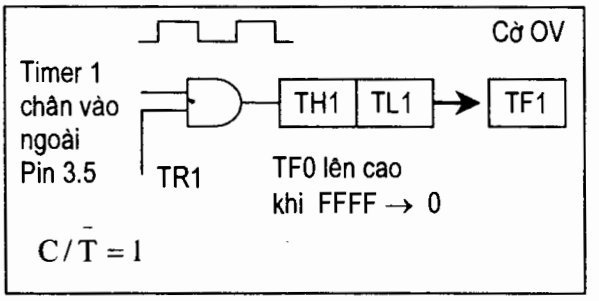
Để ý lệnh "SETB P3.5" ở chương trình trên. Vì các cổng được thiết lập làm đầu ra mỗi khi 8051 được cấp nguồn, do đó, để P3.5 trở thành đầu vào thì cần chuyển nó lên mức cao. Nói cách khác, ta cần định lại cấu hình cho chân T1 (P3.5) để trở thành chân vào.

Ở ví dụ 9.18 chúng ta sử dụng bộ Timer 1 làm bộ đếm sự kiện để đếm tăng mỗi khi có xung đồng hồ cấp đến chân P3.5. Các xung đồng hồ này có thể biểu diễn số người đi qua cổng hoặc số vòng quay hoặc bất kỳ sự kiện nào có thể chuyển đổi thành xung.

Ở ví dụ 9.19, thanh ghi TL được chuyển đổi về mã ASCII để hiển thị trên màn hình LCD.



Hình 9.5. Timer 0, chân vào ngoài (Mode 1)



Hình 9.6. Timer 1, chân vào ngoài (Mode 1)

Ví dụ 9.19

Giả sử một dãy xung có tần số 1 Hz được nối tới chân đầu vào P3.4. Hãy viết chương trình hiển thị bộ đếm 0 trên màn LCD. Giá trị ban đầu của TH0 là - 60.

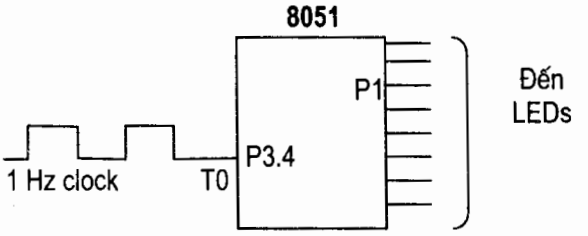
Giải:

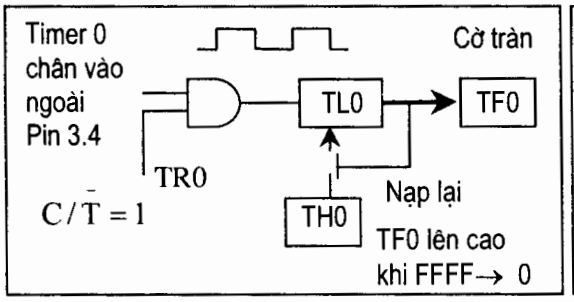
Để hiển thị số đếm TL trên LCD ta phải thực hiện chuyển đổi dữ liệu 8 bit nhị phân về mã ASCII.

```
      ACALL  LCD-SET UP       ;Khởi tạo CLD
      MOV    TMOD,#000110B    ;Chọn Timer 0,Mode2,bit C/T=1
      MOV    TH0,#-60         ;Đếm 60 xung
      SETB   P3.4             ;Lấy T0 làm đầu vào
AGAIN:SETB   TR0              ;Bắt đầu đếm TL0
BACK: MOV    A,TL0            ;Sao lại TL0
      ACALL  CONV             ;Chuyển đổi ở R2,R3,R4
      ACALL  DISLAY           ;Hiển thị trên LCD
      JNB    TF0,BACK         ;Thực hiện vòng lặp nếu TF=0
      CLR    TR0              ;Dừng bộ đếm 0
      CLR    TF0              ;Xoá cờ TF0=0
      SJMP   AGAIN            ;Tiếp tục
; chuyển đổi mã nhị phân 8 bit về mà SCII
;khi trở về, R4,R3,R2 có mã ASCII
CONV: MOV    B,#10            ;Chia cho 10
      DIV    AB
      MOV    R2,B             ;Lưu giữ số thấp
      MOV    B,#10            ;Chia cho 10 một lần nữa
      DIV    AB
      ORL    A,#30H           ;Đổi về ASCII
      MOV    R4,A             ;Lưu MSD
      MOV    AB
      ORL    A,#30H           ;Đổi số thứ hai về ASCII
      MOVR3, A
      MOV    A, R2
      ORLA, #30H              ;Đổi số thứ ba về ASCII
      MOVR2, A                ;Lưu số ASCII vào R2
      RET
```

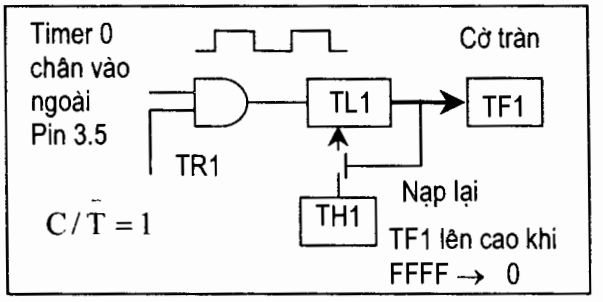
Dùng tần số 60Hz ta có thể tạo ra đồng hồ chỉ thời gian giây, phút, giờ.

Lưu ý rằng, vòng đầu tiên được bắt đầu từ 0 vì khi RESET thì TL0 = 0. Để giải quyết vấn đề này cần nạp TL0 = - 60 từ đầu chương trình.





Hình 9.7. Timer 0, chân vào ngoài (Mode 2)



Hình 9.8. Timer 1, chân vào ngoài (Mode 2)

Một ví dụ ứng dụng khác của bộ định thời gian với bit C/T = 1 là nạp một sóng vuông ngoài với tần số 60Hz vào bộ định thời. Chương trình sẽ tạo ra các đơn vị thời gian chuẩn theo giây, phút, giờ và hiển thị lên màn LCD. Kết quả sẽ cho một đồng hồ thời gian số rất hấp dẫn, tuy nhiên độ chính xác chỉ ở mức độ.

Để kết thúc phần này, cần nhắc lại hai vấn đề quan trọng:

1. Liệu Bạn có nghĩ rằng, dùng lệnh "JNB TFx, đích" để thường xuyên kiểm tra mức cao của cờ TF là lãng phí thời gian của bộ vi điều khiển không? Đúng như vậy. Tuy nhiên, chúng ta cũng có giải pháp cho vấn đề này đó là sử dụng ngắt. Nếu sử dụng ngắt, bộ vi điều khiển có thể thực hiện mọi công việc khác và khi cờ TF được bật, ngắt có nhiệm vụ báo cho bộ xử lý biết cờ đã chuyển trạng thái. Đây là một đặc điểm quan trọng và là một điểm mạnh của 8051 (mà chúng ta sẽ bàn ở chương 11).
2. Câu hỏi tiếp theo là các thanh ghi TR0 và TR1 thuộc về đâu? Chúng thuộc về một thanh ghi gọi là TCON sẽ được đề cập đến ngay sau đây [TCON - là thanh ghi điều khiển bộ đếm (bộ định thời)].

Bảng 9.2. Các lệnh tương đương đối với thanh ghi điều khiển bộ định thời

Đối với Timer 0	
	SETB TR0 = SETB TCON.4
	CLR TR0 = CLR TCON.4
	SETB TF = SETB TCON.5
	CLR TF0 = CLR TCON.5
Đối với Timer 1	
	SETB TR1 = SETB TCON.6
	CLR TR1 = CLR TCON.6
	SETB TF1 = SETB TCON.7
	CLR TF1 = CLR TCON.7

TCON: Timer/Counter Control Register

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

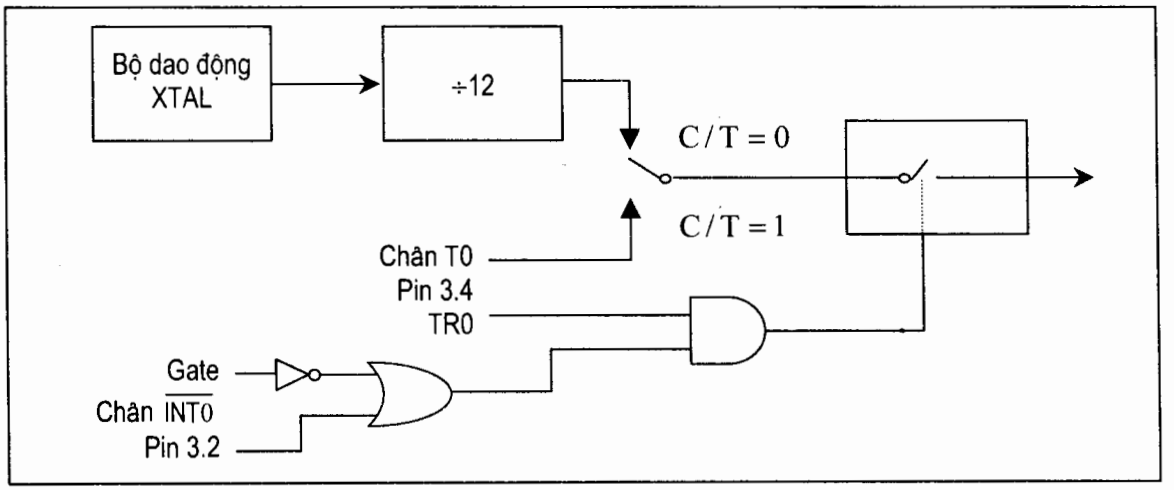
Thanh ghi TCON

Ở các ví dụ trên đây, chúng ta đã biết công dụng của các cờ TR0 và TR1 để bật/tắt các bộ định thời. Các bit này thuộc thanh ghi điều khiển bộ định thời TCON (Timer Control). Đây là thanh ghi 8 bit. Như bảng 9.2 giới thiệu, bốn bit cao được dùng để lưu các bit TF và TR cho cả Timer 0 và Timer 1. Còn bốn bit thấp được thiết lập dành cho điều khiển các bit ngắt mà ta sẽ bàn ở chương 11. Cũng cần lưu ý rằng, thanh ghi TCON là thanh ghi có thể định địa chỉ bit được. Nên hoàn toàn có thể thay các lệnh như "SETB TR1" và "CLR TR1" bằng các lệnh tương ứng như "SET TCON.6" và "CLR TCON.6" (xem bảng 9.2).

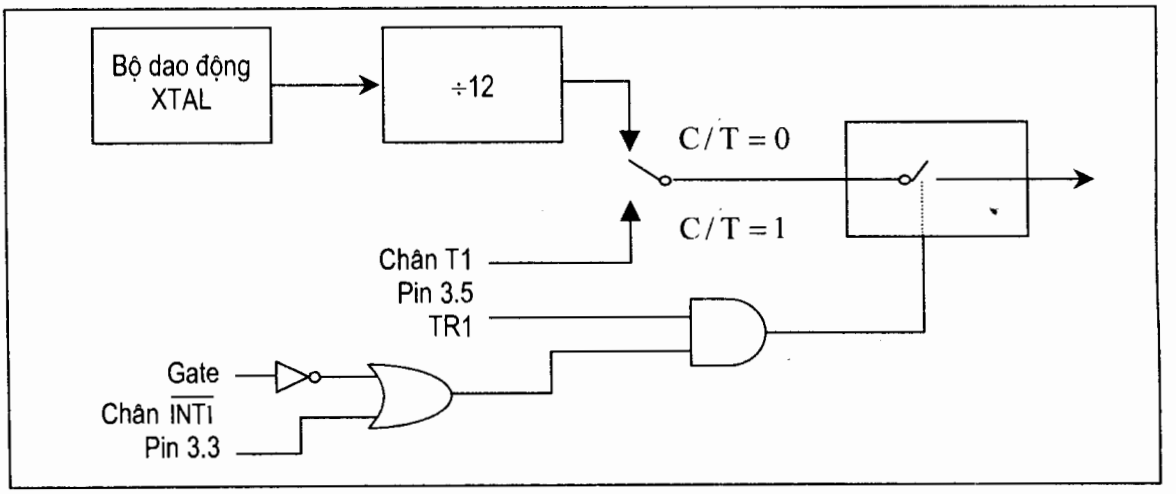
Trường hợp bit GATE = 1 (của TMOD)

Trước khi kết thúc chương, chúng ta cần làm rõ thêm về trường hợp bit GATE = 1 trong thanh ghi TMOD. Tất cả những gì được tìm hiểu trong chương này đều với giả thiết GATE = 0. Khi GATE = 0 thì các bộ định thời Timer 0 và Timer 1được khởi động bằng các lệnh tương ứng "SETB TR0" và "SETB TR1". Vậy nếu GATE = 1 thì sao? Như có thể thấy trên hình 9.9 và 9.10, nếu GATE = 1

thì việc khởi động và dừng bộ định thời Timer 0 và Timer 1được thực hiện từ bên ngoài qua chân P2.3 và P3.3 tương ứng. Tuy nhiên, nếu TRx được bật lên bằng lệnh "SETB TRx" thì cũng cho phép khởi động và dừng bộ định thời từ bên ngoài tại bất kỳ thời điểm nào thông qua công tắc chuyển mạch đơn giản. Phương pháp dùng phần cứng để dừng và khởi động bộ định thời có rất nhiều ứng dụng. Ví dụ, hệ 8051 làm thiết bị báo động theo từng giây sử dụng bộ Timer 0. Bộ Timer 0 được bật bằng phần mềm nhờ lệnh "SETB TR0" và nằm ngoài sự kiểm soát của người dùng. Tuy nhiên, nếu nối một công tắc chuyển mạch tới chân P2.3 thì ta có thể dừng và khởi động bộ định thời và như vậy có thể tắt báo động.



Hình 9.9. Bộ định thời/bộ đếm 0



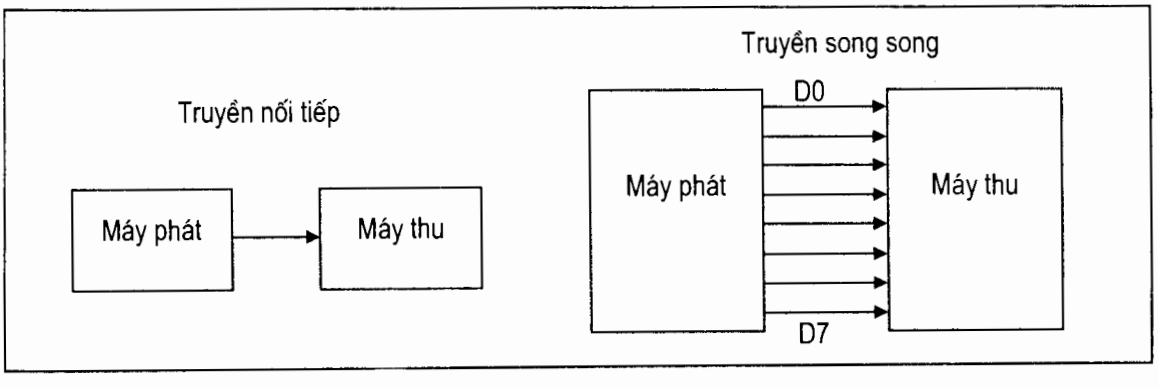
Hình 9.10. Bộ định thời/bộ đếm 1

Chương 10
TRUYỀN TIN NỐI TIẾP VỚI 8051

Máy tính truyền dữ liệu theo hai phương pháp: song song và nối tiếp. Truyền dữ liệu song song thường sử dụng 8 hoặc nhiều đường dây dẫn để truyền dữ liệu đến thiết bị ở cách xa một vài mét. Ví dụ của truyền dữ liệu song song là máy in và ổ đĩa cứng. Phương pháp này cho phép truyền dữ liệu với tốc độ cao nhờ dùng nhiều dây dẫn để truyền dữ liệu đồng thời, nhưng khoảng cách truyền thì bị hạn chế. Để truyền dữ liệu đi xa thì cần sử dụng phương pháp truyền nối tiếp. Phương pháp này truyền dữ liệu theo từng bit một.

10.1 CƠ SỞ CỦA TRUYỀN TIN NỐI TIẾP

Khi bộ vi xử lý truyền tin với thế giới bên ngoài thì nó cấp dữ liệu dưới dạng từng byte (8 bit) một. Trong một số trường hợp, chẳng hạn như máy in, thì thông tin được lấy từ bus dữ liệu 8 bit của máy tính và gửi tới bus dữ liệu 8 bit của máy in. Phương pháp này chỉ thực hiện được khi đường cáp không quá dài vì nếu cáp dài quá sẽ làm suy giảm thậm chí làm méo tín hiệu. Ngoài ra, đường 8 bit dữ liệu giá thành cao. Vì những lý do đó, trong trường hợp hai hệ thống ở cách xa nhau hàng trăm đến hàng triệu kilômet thì người ta sử dụng truyền thông nối tiếp. Hình 10.1 so sánh hai phương pháp truyền tin nói trên.



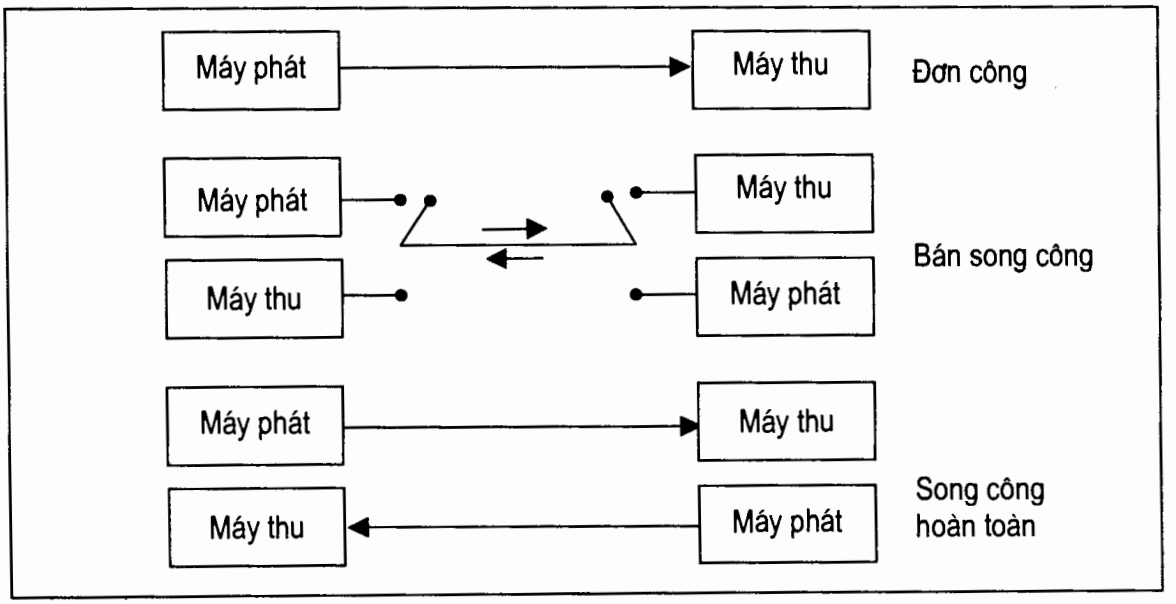
Hình 10.1. Truyền dữ liệu song song và nối tiếp

Để truyền tin nối tiếp, người ta sử dụng một đường dữ liệu thay cho bus dữ liệu 8 bit của truyền tin song song, nhờ vậy không chỉ làm cho giá thành hạ hơn nhiều mà còn mở ra một khả năng để hai máy tính ở cách rất xa nhau vẫn có thể

truyền thông với nhau qua đường điện thoại.

Để tổ chức truyền tin nối tiếp, trước hết byte dữ liệu được chuyển thành các bit nối tiếp nhờ thanh ghi dịch *vào-song song-ra-nối tiếp*. Tiếp theo, dữ liệu được truyền quan một đường dữ liệu đơn. Như vậy, ở đầu thu cũng phải có một thanh ghi dịch *vào-nối tiếp-ra-song song* để nhận dữ liệu nối tiếp và sau đó gói chúng thành từng byte một. Tất nhiên, nếu dữ liệu được truyền qua đường thoại thì cần được chuyển đổi từ các số 0 và 1 sang âm thanh ở dạng sóng hình sin. Việc chuyển đổi này do một thiết bị có tên gọi là *bộ điều chế/giải điều chế Modem* (Modulator/demodulator) thực hiện.

Khi cự ly truyền ngắn thì tín hiệu số có thể được truyền, như giới thiệu ở trên, bằng một dây dẫn và không cần điều chế. Đây cũng chính là phương pháp mà bàn phím IBM-PC vẫn sử dụng để truyền dữ liệu đến bảng mạch chính. Tuy nhiên, để truyền dữ liệu đi xa qua các đường truyền chẳng hạn như đường điện thoại thì truyền dữ liệu nối tiếp yêu cầu một modem để điều chế (chuyển các số 0 và 1 về tín hiệu âm thanh) và sau đó giải điều chế (chuyển tín hiệu âm thanh về các số 0 và 1).



Hình 10.2. Các chế độ thu phát dữ liệu

Truyền tin nối tiếp có hai phương pháp: đồng bộ và dị bộ. Phương pháp đồng bộ chuyển mỗi lần một khối dữ liệu (các ký tự), còn phương pháp dị bộ chỉ truyền từng byte một. Có thể viết phần mềm để sử dụng một trong hai phương pháp

truyền này. Tuy nhiên, chương trình máy tính dạng này thường rất dài và buồn tẻ. Vì lý do đó mà nhiều nhà sản xuất đã cho ra thị trường các loại IC chuyên dụng phục vụ cho truyền dữ liệu nối tiếp. Những IC này dùng làm các bộ thu - phát dị bộ tổng hợp UART (Universal Asynchronous Synchronous Receiver Transmitter) và các bộ thu - phát đồng - dị bộ tổng hợp USART (Universal Synchronous-Asynchronous Receiver Transmitter). Bộ vi điều khiển 8051 được xây dựng sẵn một bộ UART và sẽ được bàn kỹ ở mục sau.

Truyền dữ liệu bán song công và song công

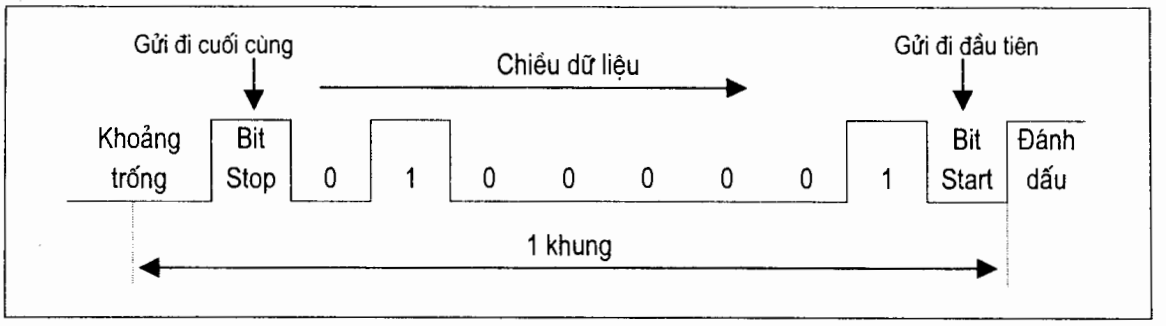
Nếu trong quá trình truyền, dữ liệu vừa có thể phát vừa có thể thu thì gọi là truyền *song công*. Trái với truyền *đơn công* (như ở máy in chẳng hạn) máy tính chỉ phát dữ liệu. Truyền song công có hai dạng: song công hoàn toàn (còn gọi là song công) hay bán song công tuỳ thuộc việc truyền dữ liệu có đồng thời hay không. Nếu tại mỗi thời điểm dữ liệu chỉ truyền một chiều thì được gọi là *bán song công*. Còn trường hợp dữ liệu truyền được cả hai chiều đồng thời thì gọi là *song công hoàn toàn*. Như vậy, ở chế độ song công hoàn toàn, để truyền được 2 chiều đồng thời cần có 2 đường dữ liệu, một đường để phát, còn đường kia để thu.

Truyền thông nối tiếp không đồng bộ và định khung dữ liệu

Dữ liệu tới đầu thu của đường truyền nối tiếp là tín hiệu 0 và 1. Việc xác định nội dung của dữ liệu sẽ khó khăn nếu giữa đầu phát và đầu thu không có một quy tắc thống nhất còn được gọi là *giao thức* (Protocol) như: dữ liệu được sắp xếp như thế nào, có bao nhiêu bit tạo thành một ký tự, khi nào bắt đầu và khi nào kết thúc dữ liệu, ...

Bit khởi động và bit dừng

Truyền tin nối tiếp không đồng bộ được sử dụng rộng rãi trong truyền ký tự, còn truyền dữ liệu định hướng khối sử dụng phương pháp đồng bộ. Ở phương pháp không đồng bộ, mỗi ký tự được bố trí vào giữa bit khởi động và bit dừng. Người ta gọi cách thức này là *định khung*. Như vậy, đối với truyền tin không đồng bộ, để định khung, dữ liệu ký tự được nén giữa bit khởi động và bit dừng. Bit khởi động luôn chỉ có một bit, còn bit dừng có thể có 1 hoặc 2 bit. Bit khởi động luôn có giá trị 0 (mức thấp), còn bit dừng thì có giá trị 1 (cao). Ví dụ, xem hình 10.3, trong đó ký tự "A" ASCII có mã nhị phân là 0100 0001, được định khung giữa 1 bit khởi động và 2 bit dừng. Chú ý là bit trọng số thấp LSB được gửi đi trước.



Hình 10.3. Định khung ký tự "A" (mã ASCII - 41 h)

Ở hình 10.3, khi không truyền thì tín hiệu là 1, gọi là *dấu (mark)*. Tín hiệu 0 được gọi là *khoảng trống (space)*. Lưu ý là trình tự truyền bắt đầu với bit khởi động, tiếp theo là bit D0, bit LSB, tiếp nữa là các bit còn lại cho đến bit có trọng số lớn nhất MSB là D7, và cuối cùng là bit dừng để báo rằng đã kết thúc ký tự "A".

Trong truyền tin nối tiếp không đồng bộ, các thiết bị ngoại vi và modem được lập trình để có độ dài dữ liệu là 7 bit hoặc 8 bit, đó là chưa kể các bit dừng có thể chiếm 1 hoặc 2 bit. Trước đây, ký tự ASCII là 7 bit, hiện nay, ký tự ASCII mở rộng có 8 bit. Trong một số hệ thống trước đây, do thiết bị thu không nhanh nên cần sử dụng 2 bit dừng để thiết bị có đủ thời gian cho truyền byte tiếp theo. Tuy nhiên, trong máy tính PC hiện đại phổ biến sử dụng một bit dừng.

Giả sử chúng ta truyền một tệp văn bản ký tự ASCII sử dụng 1 bit dừng thì ta có tổng cộng 10 bit cho mỗi ký tự, gồm: 8 bit ký tự ASCII chuẩn, 1 bit khởi động và 1 bit dừng. Do đó, cứ 8 bit ký tự thì có thêm 2 bit, chiếm 25% tổng phí.

Ở một số hệ thống, nhằm tăng khả năng bảo toàn của dữ liệu, người ta còn thêm vào khung dữ liệu 1 bit bậc (hay còn gọi là bit chẵn lẻ). Bit bậc có thể là bậc chẵn hoặc lẻ. Bit bậc lẻ có tổng các số 1 của các bit dữ liệu cùng với bit bậc là lẻ. Tương tự, bit bậc chẵn là khi tổng các số 1 của các bit dữ liệu và bit bậc là chẵn. Ví dụ, với ký tự "A", mã ASCII là 0100 0001 thì bit bậc chẵn sẽ có giá trị là 0 vì tổng các số 1 là 2. Chip UART cho phép lập trình bit bậc với các tuỳ chọn chẵn, lẻ hoặc không có bit bậc. Trong hệ thống có bit bậc, thì bit này được gửi đi sau bit MSB và trước bit dừng.

Tốc độ truyền dữ liệu

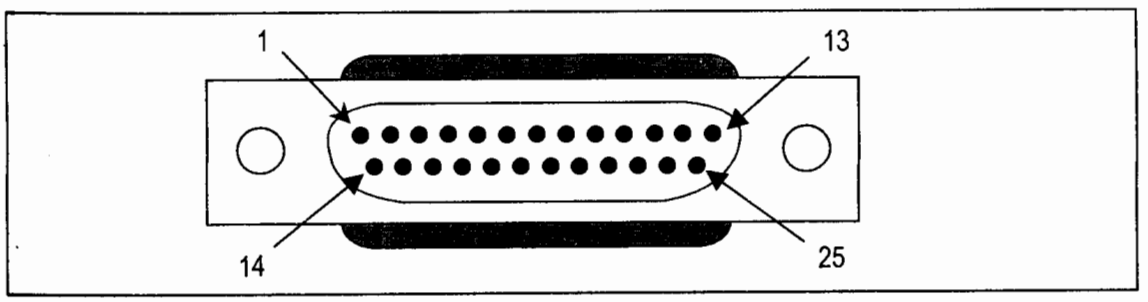
Tốc độ truyền tin nối tiếp được tính bằng bit/giây *bps* (Bit Per Second). Một thuật ngữ khác cũng thường được sử dụng là *baud*. Tuy nhiên, khái niệm bps và

baud không hoàn toàn giống nhau. Baud là đơn vị đo dùng cho modem và được định nghĩa là số lần thay đổi tín hiệu trong một giây. Đối với modem, mỗi lần thay đổi tín hiệu có thể truyền được nhiều bit dữ liệu. Còn đối với đường truyền thì tốc độ baud và bps là một. Do đó, trong tài liệu này chúng ta không phân biệt hai thuật ngữ trên.

Tốc độ truyền dữ liệu của từng máy tính phụ thuộc vào cổng truyền tin của hệ máy đó. Ví dụ, các máy tính PC/XT của IBM trước kia có thể truyền dữ liệu với tốc độ từ 100 đến 9600 bps. Tuy nhiên, trong những năm gần đây, tốc độ của các máy PC dựa trên Pentium đã đạt tốc độ truyền dữ liệu rất cao, lên tới 56 Kbps. Cần phải nói thêm rằng, đối với truyền dữ liệu nối tiếp không đồng bộ thì nói chung tốc độ bị hạn chế đến 100.000 bps.

Chuẩn RS232

Để bảo đảm sự tương thích giữa các thiết bị truyền dữ liệu nối tiếp do các hãng khác nhau sản xuất, năm 1960 Hiệp hội Công nghiệp Điện tử EIA đã xây dựng một chuẩn giao diện được gọi là RS232. Năm 1963, chuẩn này được cải tiến và gọi là RS232A. RS232B và RS232C vào những năm 1965 và 1969. Tài liệu này chỉ đề cập đến RS232. Ngày nay, RS232 là chuẩn giao diện I/O được sử dụng rộng rãi nhất . Tuy nhiên, do chuẩn này ra đời đã khá lâu, trước khi có họ mạch vi điện tử TTL, vì vậy các mức điện áp vào/ra của nó không tương thích với TTL. Ở RS232, mức 1 tương ứng từ -3V đến -25V, còn mức 0 tương ứng từ +3V đến +25V, khoảng từ -3V đến +3V không xác định. Do đó, để nối RS232 với máy tính đều phải qua bộ biến đổi điện áp như MAX232 để chuyển mức logic TTL sang mức điện áp của RS232 và ngược lại. Nhìn chung, các chip IC MAX232 được dùng để điều khiển đường truyền. Kết nối RS232 với MAX232 được thảo luận ở mục 10.2.



Hình 10.4. Ổ cắm RS232 DB-25

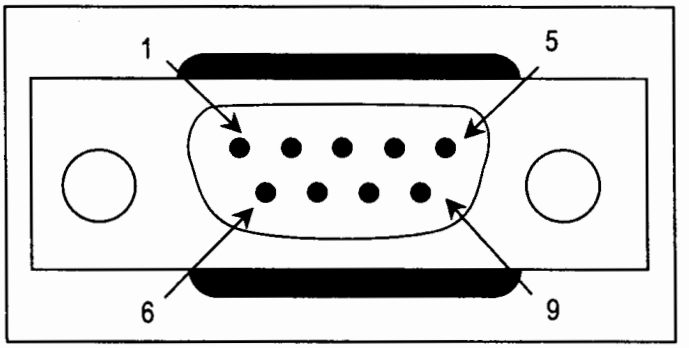
Bố trí chân của RS232

Bảng 10.1 giới thiệu sơ đồ bố trí chân của RS232 dạng 25 chân và có tên gọi là DB - 25. Để phân biệt, người ta dùng ký hiệu DB - 25P để chỉ đầu đực (Plug-cắm vào) và DB - 25S để chỉ đầu cái (Socket-ổ cắm).

Bảng 10.1. Bố trí chân DB-25P (đầu đực) của RS232

Chân	Tên gọi	Chân	Tên gọi
1	Protective ground	14	2nd transmitted data
2	Transmitted data (TxD)	15	Transmitted signal element timing
3	Received data (RxD)	16	2nd received data
4	Request to send ($\overline{\text{RTS}}$)	17	Receive signal element timing
5	Clear to send ($\overline{\text{CTS}}$)	18	Unassigned
6	Data set ready ($\overline{\text{DSR}}$)	19	2nd Request to send
7	Signal ground (GND)	20	Data terminal ready ($\overline{\text{DTR}}$)
8	Data carrier detect ($\overline{\text{DCD}}$)	21	Signal quality detector
9/10	Reserved for data set testing	22	Ring indicator (RI)
11	Unassigned	23	Data signal rate select
12	2nd clear to send	24	Transmitted signal element timing
13	2nd clear to send	25	Unassigned

Không phải tất cả các chân của DB-25 đều được sử dụng, do vậy IBM đưa ra phiên bản chuẩn vào/ra nối tiếp chỉ sử dụng có 9 chân gọi là DB - 9, như trình bày ở bảng 10.2 và hình 10.5.



Hình 10.5. Sơ đồ đầu nối DB - 9 của RS232

Phân loại truyền dữ liệu

Người ta phân biệt thiết bị truyền thông dữ liệu thành thiết bị đầu cuối dữ liệu DTE (Data Terminal Equipment) hoặc thiết bị truyền thông dữ liệu DCE (Data Communication Equipment). DTE chủ yếu là máy tính và các thiết bị đầu cuối gửi và nhận dữ liệu, còn DCE là thiết bị truyền thông, chẳng hạn như các modem chịu trách nhiệm về truyền dữ liệu. Lưu ý rằng tất cả mọi định nghĩa về chức năng các chân RS232 ở các bảng 10.1 và 10.2 đều nhìn từ góc độ của DTE.

Kết nối đơn giản nhất giữa một PC và bộ vi điều khiển yêu cầu tối thiểu những chân sau: TxD, RxD và đất - như trình bày ở hình 10.6. Để ý rằng trên hình này thì các chân TxD và RxD được đổi cho nhau.

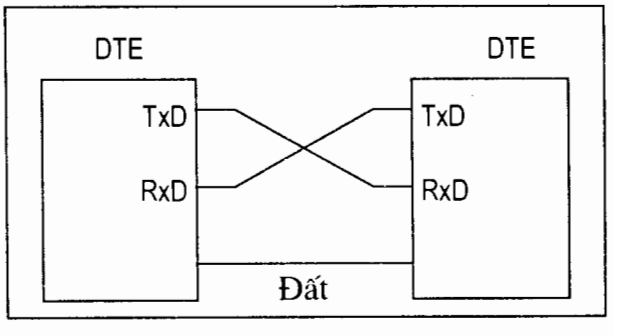
Bảng 10.2. Tín hiệu các chân đầu nối DB - 9 trên máy tính IBM PC

Chân	Mô tả	Ý nghĩa
1	Data carrier detect (DCD)	Tách tín hiệu mang dữ liệu
2	Received data (RxD)	Dữ liệu được nhận
3	Transmitted data (TxD)	Dữ liệu được gửi
4	Data terminal ready (DTR)	Đầu cuối dữ liệu sẵn sàng
5	Signal ground (GND)	Đất của tín hiệu
6	Data set ready (DSR)	Dữ liệu sẵn sàng
7	Request to send (RTS)	Yêu cầu gửi
8	Clear to send (CTS)	Xoá để gửi
9	Ring indicator (RL)	Báo chuông

Kiểm tra tín hiệu bắt tay của RS232

Để bảo đảm truyền dữ liệu nhanh và tin cậy giữa hai thiết bị thì việc truyền dữ liệu phải được phối hợp tốt. Chẳng hạn như trong trường hợp của máy in, do thực tế là trong truyền tin nối tiếp, thiết bị thu có thể không đủ chỗ để chứa dữ liệu, khi đó phải có cách để báo cho bên phát dừng gửi dữ liệu. Có một số chân của RS232 được dùng làm các tín hiệu bắt tay. Phần giới thiệu dưới đây có thể xem như để tham khảo và có thể được bỏ qua vì chúng không được chip UART của 8051hỗ trợ.

1. DTR (data terminal ready) - thiết bị đầu cuối sẵn sàng. Sau khi được cấp nguồn, và sau khi kết thúc quá trình tự kiểm tra, thiết bị đầu cuối (hoặc cổng COM máy tính) gửi tín hiệu DTR để báo rằng thiết bị đã sẵn sàng để truyền tin. Nếu có lỗi ở cổng COM, tín hiệu này sẽ không được kích hoạt. Đây là tín hiệu tích cực thấp và có thể được dùng để thông báo cho modem biết rằng máy tính đang hoạt động. Đây là chân ra từ



Hình 10.6. Kết nối modem rỗng

DTE (cổng COM máy tính) và đưa vào modem.

2. DSR (data set ready) - dữ liệu sẵn sàng. Khi thiết bị truyền thông DCE (ví dụ modem) được cấp nguồn và thực hiện xong quá trình tự kiểm tra, tín hiệu DSR được sử dụng để xác nhận dữ liệu đã sẵn sàng để truyền tin. Như vậy DSR là chân ra từ modem (DCE) và vào máy tính (DTE). Tín hiệu này có mức tích cực thấp. Nếu vì một lý do nào đó modem không thể kết nối được vào mạng điện thoại thì tín hiệu này sẽ không được kích hoạt để báo cho máy tính (hoặc thiết bị đầu cuối) rằng nó không thể nhận hoặc gửi dữ liệu.
3. RTS (request to send) - yêu cầu gửi dữ liệu. Tín hiệu RTS được sử dụng để báo cho modem biết thiết bị DTE (ví dụ máy tính) có một byte cần gửi. RTS là đầu ra tích cực thấp từ thiết bị DTE và vào thiết bị DCE.
4. CTS (clear to send) -tín hiệu thông. Để trả lời tín hiệu RTS, và khi đã sẵn sàng, modem gửi tín hiệu CTS đến thiết bị DTE (máy tính chẳng hạn) báo máy tính biết modem đang sẵn sàng nhận dữ liệu. CTS là tín hiệu vào thiết bị DTE và được DTE sử dụng để bắt đầu truyền dữ liệu.
5. CD (carrier detect) - dò sóng mang, hoặc DCD (data carrier detect) - dò sóng mang dữ liệu. Modem gửi tín hiệu DCD để báo cho thiết bị DTE (PC) rằng đã phát hiện được sóng mang và sự kết nối giữa thiết bị DTE với modem đã được thiết lập. DCD là tín hiệu ra từ modem và đưa vào DTE (PC).
6. RI (ring indicator) - chỉ thị chuông. Đây là tín hiệu ra từ modem (DCE) đưa vào PC (DTE) để báo rằng điện thoại đang réo chuông. Tín hiệu này tắt hoặc mở đồng bộ với tiếng chuông. Trong 6 tín hiệu bắt tay, đây là tín hiệu ít được sử dụng nhất.

Từ những trình bày trên, có thể tóm tắt quá trình truyền tin giữa PC và modem như sau: khi có các tín hiệu DTR và DSR của PC và modem, chúng báo hiệu là PC và modem làm việc tốt, các tín hiệu RTS và CTS điều khiển luồng dữ liệu. Khi PC muốn gửi dữ liệu thì nó gửi tín hiệu RTS cho modem và để trả lời lại, nếu modem đã sẵn sàng (có chỗ chứa dữ liệu) để nhận dữ liệu từ PC thì nó gửi lại cho PC tín hiệu CTS.

Nếu modem bận thì nó không phát tín hiệu CTS, PC sẽ huỷ tín hiệu DTR và sẽ thử lặp lại quá trình này. Tín hiệu CTS và RTS còn gọi là tín hiệu điều khiển luồng phần cứng. Còn nếu thiếu chỗ chứa dữ liệu thì modem không kích hoạt CTS và PC thôi không yêu cầu DTR và tiến hành thử lại lần nữa. Các tín hiệu RTS và CTS cũng được coi là tín hiệu luồng điều khiển phần cứng.

Đến đây, chúng ta tạm kết thúc phần giới thiệu 9 chân quan trọng nhất của các tín hiệu bắt tay RS232 cùng với các tín hiệu TxD, RxD và đất (Ground). Tín hiệu đất Ground còn được ký hiệu là SG (Signal Ground).

Cổng COM của máy IBM PC và tương thích

Máy tính IBM PC và tương thích dựa trên các bộ vi xử lý ×86 (8086, 286, 384, 486 và Pentium) thường có hai cổng COM. Cả hai cổng COM đều có đầu nối RS232. Nhiều máy tính PC sử dụng mỗi đầu nối một kiểu ổ cắm DB-25 và DB-9. Trong những năm gần đây, cổng COM 1 được dùng cho chuột và COM 2 - cho các thiết bị chẳng hạn như Modem. Chúng ta có thể nối cổng nối tiếp của 8051 đến cổng COM 2 của một máy tính PC để thực hiện một số thử nghiệm về truyền thông nối tiếp.

Với kiến thức nền về truyền thông nối tiếp trình bày ở trên, đến đây chúng ta có thể sẵn sàng để làm việc với 8051.

10.2 NỐI GHÉP 8051 VỚI RS232

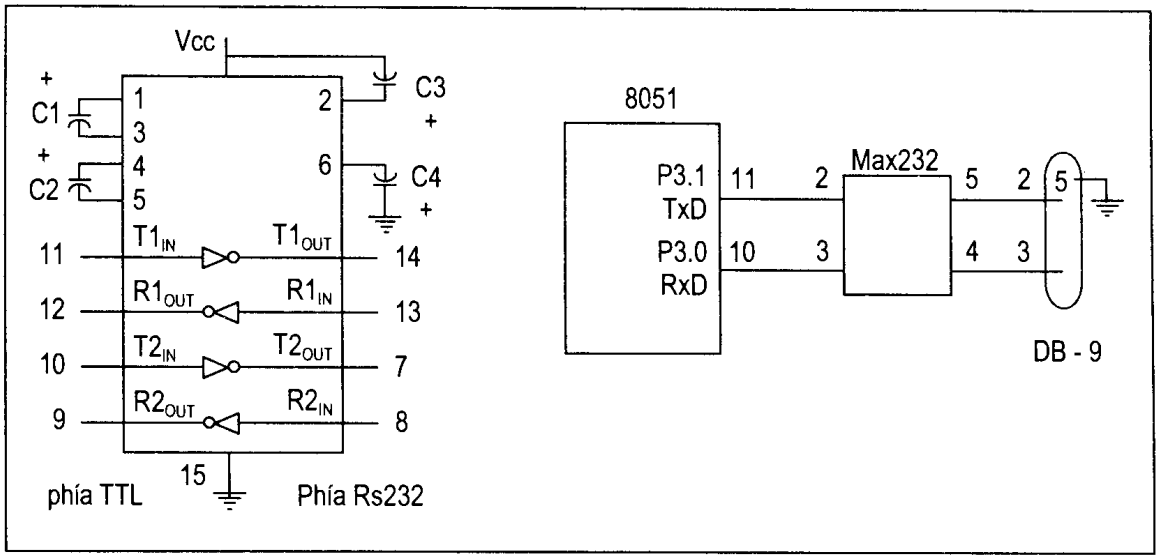
Như đã nói ở phần 10.1, chuẩn RS232 không tương thích với mức logic TTL, nên cần bổ sung thêm một bộ điều khiển đường truyền, chẳng hạn như chip MAX232 để chuyển đổi các mức điện áp RS232 về các mức TTL và ngược lại. Nội dung chính của phần này là bàn về nối ghép 8051 với đầu nối RS232 thông qua chip MAX232.

Chân RxD và TxD của 8051

8051 có hai chân được dùng chuyên cho truyền và nhận dữ liệu nối tiếp. Hai chân này là RxD và TxD và là một phần của cổng P3 (đó là P3.0 và P3.1 tương ứng). P3.0 là chân số 10 của 8051, còn P3.1 là chân số 11. Các chân này tương thích với mức logic TTL. Do vậy cần có thêm một bộ điều khiển đường truyền để chúng tương thích với RS232. Bộ điều khiển như vậy có thể là chip MAX232.

Bộ điều khiển đường truyền MAX232

Vì RS232 không tương thích với các bộ vi xử lý và vi điều khiển hiện nay nên ta cần một bộ điều khiển đường truyền (bộ chuyển đổi điện áp) để chuyển đổi các tín hiệu RS232 về các mức điện áp TTL được các chân TxD và RxD của 8051 chấp nhận. Một ví dụ của bộ chuyển đổi như vậy là chip MAX232 của hãng Maxim. Bộ MAX232 chuyển đổi từ các mức điện áp RS232 về mức TTL và ngược lại. Một điểm mạnh khác của chip MAX232 đó là dùng điện áp nguồn +5v, cùng với điện áp nguồn của 8051.



Hình 10.7. a) Sơ đồ cấu trúc của MAX232;
b) Sơ đồ nối ghép modem rỗng MAX232 với 8051

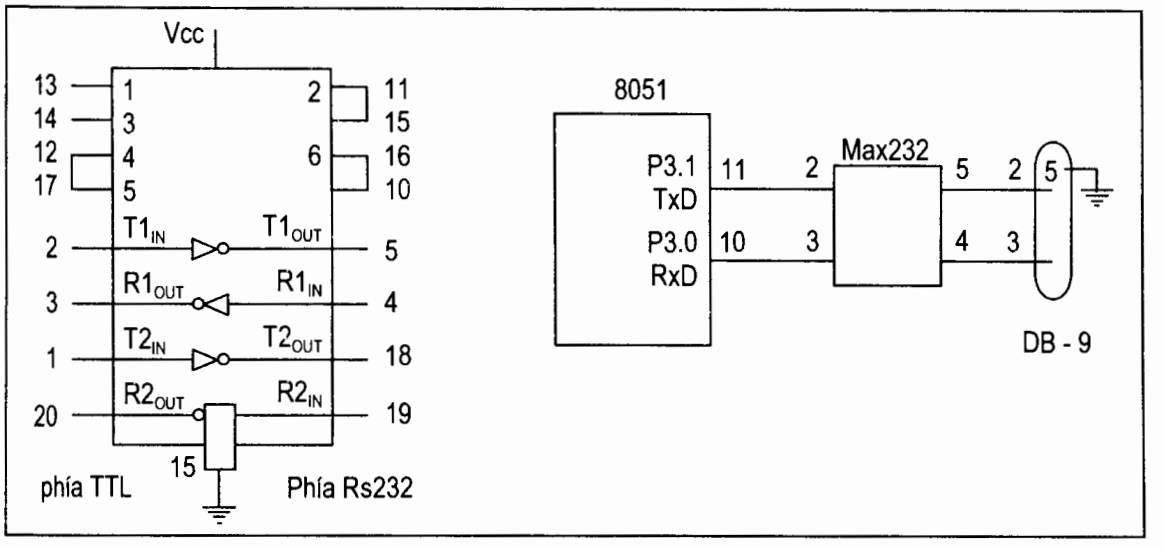
MAX232 có hai bộ điều khiển đường truyền là nhận và truyền dữ liệu như giới thiệu ở hình 10.7. Các bộ điều khiển đường truyền dùng cho TxD được gọi là T1 và T2. Trong nhiều ứng dụng thì chỉ có một cặp được dùng. Ví dụ T1 và R1 được dùng với nhau cho trường hợp TxD và RxD của 8051, còn cặp R2 và T2 thì không dùng đến. Để ý rằng, bộ điều khiển T1 của MAX232 có gán $T1_{in}$ và $T1_{out}$ trên các chân số 11 và 1 tương ứng. Chân $T1_{in}$ là ở phía TTL và được nối tới chân RxD của bộ vi điều khiển, còn $T1_{out}$ là ở phía RS232 được nối tới chân RxD của đầu nối DB của RS232. Bộ điều khiển đường R1 cũng có gán $R1_{in}$ và $R1_{out}$ trên các chân số 13 và 12 tương ứng. Chân $R1_{in}$ (chân số 13) là ở phía RS232 được nối tới chân TxD đầu nối DB của RS232 và chân $R1_{out}$ (chân số 12) là ở phía TTL và được nối tới chân RxD của bộ vi điều khiển, xem hình 10.7. Để ý rằng nối ghép modem rỗng là nối ghép mà chân TxD bên phát được nối với RxD của bên thu và ngược lại.

MAX232 cần có 4 tụ điện giá trị từ 1 đến 22μF. Giá trị thường dùng là 22μF.

Bộ điều khiển MAX233

Để tiết kiệm không gian trên bảng mạch, nhiều nhà thiết kế sử dụng chip MAX233 của hãng Maxim. Bộ điều khiển MAX233 có các chức năng như MAX232 song không cần các tụ điện. Tuy nhiên, chip MAX233 đắt hơn đáng kể

so với MAX233 và không tương thích với MAX232 (bố trí chân khác nhau). Như vậy Bạn không thể rút chip MAX232 ra khỏi bảng mạch và thay vào đó MAX233. Hãy xem hình 10.8 và thấy MAX233 không dùng tụ điện.



Hình 10.8. a) Sơ đồ cấu trúc của MAX233;
b) Nối ghép modem rỗng MAX232 với 8051

10.3 LẬP TRÌNH TRUYỀN THÔNG NỐI TIẾP CHO 8051

Trong phần này chúng ta sẽ nghiên cứu về các thanh ghi truyền thông nối tiếp của 8051 và cách lập trình để phát và thu dữ liệu. Máy tính IBM PC và tương thích truyền thông với các hệ 8051 được sử dụng khá phổ biến. Ở đây chúng ta tập trung chủ yếu vào truyền thông nối tiếp của 8051 với cổng COM của PC. Để thực hiện truyền dữ liệu không có lỗi giữa máy tính PC và hệ 8051 thì tốc độ baud của hệ 8051 phải phù hợp với tốc độ baud của cổng COM máy tính PC như được cho ở bảng 10.3. Bạn có thể kiểm tra tốc độ baud này bằng cách vào Windows Terminal và bấm chuột lên tuỳ chọn Communication Settings. Trong Window95 và cao hơn, Bạn có thể sử dụng chức năng Hyperterrminal. Hàm Hyperterminal hỗ trợ tốc độ baud cao hơn nhiều so với các tốc độ cho trong bảng 10.3.

Bảng 10.3. Tốc độ baud của máy tính PC

100
150
300
600
1200
2400
4800
9600
19200

Ví dụ 10.1

Cho tần số XTAL là 11.0592MHz. Hãy xác định giá trị TH1 để có tốc độ baud là:

a) 9600; b) 2400; c) 1200.

Giải:

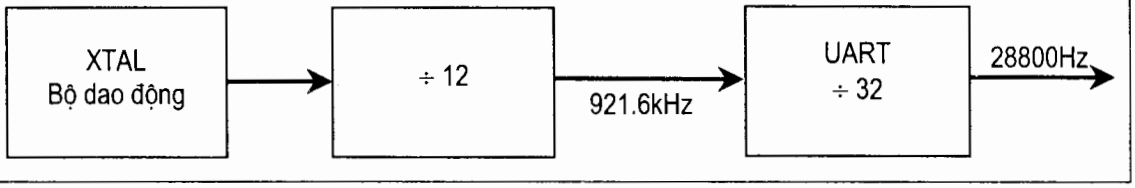
Với tần số XTAL là 11.0592MHz thì tần số chu trình máy của 8051 là 11.0592MHz : 12 = 921.6kHz, lấy 921.6kHz/32 = 28.800Hz là tần số UART cấp tới bộ định thời Timer 1 để thiết lập tốc độ.

a) 28.800/3 = 9600 trong đó - 3 = FD được nạp vào TH1.

b) 28.800/12 = 2400 trong đó - 12 = F4 được nạp vào TH1.

c) 28.800/24 = 1200 trong đó - 24 = F8 được nạp vào TH1.

Lưu ý rằng việc chia 1/12 của tần số thạch anh cho 32 là giá trị mặc định khi kích hoạt chân RESET của 8051. Chúng ta có thể thay đổi giá trị cài đặt mặc định này và sẽ tìm hiểu ở cuối chương.



Tốc độ baud của 8051

8051 phát và thu dữ liệu nối tiếp theo nhiều tốc độ khác nhau. Tốc độ truyền có thể lập trình được thông qua bộ định thời Timer 1. Trước khi tìm hiểu vấn đề này, chúng ta sẽ xem xét quan hệ giữa tần số thạch anh và tốc độ baud của 8051.

Như đã đề cấp trước đây, tần số chu trình máy của 8051 là tần số thạch anh chia cho 12. Nếu XTAL = 11.0592MHz thì tần số chu trình là 921.6kHz (11.0592MHz : 12 = 921.6kHz). Mạch UART truyền nối tiếp của 8051 lại chia tần số chu trình máy cho 32 một lần nữa trước khi đưa đến Timer 1 để tạo ra tốc độ baud. Do vậy, 921.6kHz : 32 = 28.800Hz là tần số sẽ được dùng trong phần này để xác định tốc độ baud. Muốn Timer 1 đặt tốc độ baud thì nó phải được lập trình về mode 2, đó là chế độ thanh ghi 8 bit tự động nạp lại. Để có tốc độ baud tương thích với PC ta phải nạp TH1 theo các giá trị cho trong bảng 10.3. Ví dụ 10.1 trình bày cách kiểm tra giá trị dữ liệu cho trong bảng 10.3.

Bảng 10.4. Giá trị thanh ghi TH1 của Timer1 với các tốc độ baud khác nhau

Tốc độ baud	TH1 (thập phân)	TH1 (số Hexa)
9600	-3	ED
4800	-6	FA
2400	-12	F4
1200	-24	E8

Thanh ghi SBUF

SBUF là thanh ghi 8 bit được dùng cho truyền thông nối tiếp của 8051. Để byte dữ liệu được truyền qua đường TxD thì cần đặt dữ liệu trong thanh ghi SBUF. Tương tự, SBUF lưu một byte dữ liệu khi nó được nhận qua đường RxD của 8051. SBUF có thể được mọi thanh ghi của 8051 truy cập. Xem ví dụ sau:

```
MOV    SBUF,#"D" ;Nạp 44H (mã ACSII của "D" vào SBUF
MOV    SBUF,A    ;Sao thanh ghi A vào SBUF
MOV    A,SBUF    ;Sao SBUF vào thanh ghi A
```

Khi byte dữ liệu được ghi vào thanh ghi SBUF thì byte sẽ được định khung với bit Start và Stop và được truyền nối tiếp quan chân TxD. Tương tự như vậy, khi các bit được nhận nối tiếp từ RxD thì 8051 mở khung, tức loại trừ các bit Start và Stop để lấy ra một byte từ dữ liệu nhận được và đặt vào thanh ghi SBUF.

Thanh ghi điều khiển nối tiếp SCON

SCON là thanh ghi 8 bit được dùng cho một số công việc, trong đó có lập trình bit khởi động Start, bit dừng Stop và các bit dữ liệu khi định khung dữ liệu.

Dưới đây là mô tả các bit của SCON:

SM0, SM1

Đây là các bit D7 và D6 của thanh ghi SCON. Các bit này được dùng để xác định chế độ định khung dữ liệu bằng cách xác định số bit của một ký tự và các bit Start và Stop. Các tổ hợp của chúng là:

SM0	*SM1*	
0	0	Chế độ nối tiếp 0
0	1	Chế độ nối tiếp 1, 8 bit dữ liệu, Start, Stop
1	0	Chế độ nối tiếp 2
1	1	Chế độ nối tiếp 3

Trong bốn chế độ trên, chúng ta chỉ quan tâm đến chế độ 1, các chế độ còn lại được giải thích ở phụ lục A3. Ở chế độ 1, dữ liệu được định khung gồm 8 bit dữ liệu, 1 bit Start, 1 bit Stop để tương thích với cổng COM của IBM PC và các máy tính tương thích khác. Đáng chú ý hơn là chế độ nối tiếp 1 cho phép tốc độ baud thay đổi và do Timer 1 của 8051 thiết lập. Như vậy, ở chế độ nối tiếp 1, mỗi ký tự gồm có 10 bit được truyền, trong đó đầu tiên là bit Start, sau đó là 8 bit dữ liệu và cuối cùng là bit Stop.

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

SM0	SCON.7	Số xác định chế độ làm việc cổng nối tiếp
SM1	SCON.6	Số xác định chế độ làm việc cổng nối tiếp
SM2	SCON.5	Dùng cho truyền thông giữa các bộ vi xử lý (SM2 = 0)
REN	SCON.4	Bật/xoá bằng phần mềm để cho phép/ không cho thu
TB8	SCON.3	Không sử dụng rộng rãi
RB8	SCON.2	Không sử dụng rộng rãi
T1	SCON.1	Cờ ngắt truyền. Đặt bằng phần cứng khi bắt đầu bit Stop ở Mode 1. Xoá bằng phần mềm
R1	SCON.0	Cờ ngắt thu. Đặt bằng phần cứng khi bắt đầu bit Stop ở Mode 1. Xoá bằng phần mềm.

Ghi chú: Các bit TM2, TB8 và RB8 đặt về 0.

Hình 10.9. Thanh ghi điều khiển cổng nối tiếp SCON (định địa chỉ bit)

SM2

SM2 là bit D5 của thanh ghi SCON. Bit này dùng trong hệ đa xử lý của 8051 và nằm ngoài phạm vi của chương này. Trong các ứng dụng được nêu ở tài liệu này cần đặt SM2 = 0 vì chúng ta không dùng 8051 trong hệ đa xử lý.

REN

REN là bit cho phép thu (Receive Enable), bit D4 của thanh ghi SCON. Bit này còn được viết là SCON.4 vì SCON là thanh ghi có thể định địa chỉ bit. Khi bit REN ở mức cao thì nó cho phép 8051 thu dữ liệu trên chân RxD. Như vậy, nếu muốn 8051 vừa phát vừa thu dữ liệu thì bit REN phải được đặt lên 1. Nếu đặt REN=0 thì bộ thu bị khoá. Có thể dùng lệnh "SETB SCON.4" và "CLR SCON.4" để đặt REN = 1 hay REN = 0 tương ứng. Nên lưu ý là các lệnh này sử dụng khả

năng định địa chỉ bit của thanh ghi SCON. Bit này có thể dùng để khoá việc thu dữ liệu nối tiếp và có thể nói, đây là một bit hết sức quan trọng của thanh ghi SCON.

TB8 và RB8

TB8 (Transfer Bit 8) là bit D3 của thanh ghi SCON. Bit được dùng cho chế độ nối tiếp 2 và 3. Trong các ứng dụng được nêu ở đây, bit này không được sử dụng, vì vậy cần đặt TB8=0.

RB8 (Receive Bit 8) là bit D2 của thanh ghi SCON. Ở chế độ nối tiếp 1, bit này nhận một bản sao của bit Stop khi một dữ liệu 8 bit được nhận. Cũng như bit TB8, bit này hiếm khi được dùng. Trong các ứng dụng được nêu, cần đặt RB8 = 0 vì không sử dụng chế độ nối tiếp 2 và 3.

TI và RI

Ngắt phát TI (Transmit Interrupt) và ngắt thu RI (Transmit Receive) là các bit D1 và D0 của thanh ghi SCON. Đây là những bit cờ rất quan trọng của thanh ghi SCON. Khi 8051 kết thúc phát một ký tự 8 bit thì cờ TI được bật để báo rằng bộ vi điều khiển sẵn sàng phát byte tiếp theo. Bit TI được bật khi bắt đầu bit Stop.

Khi 8051 nhận được dữ liệu nối tiếp qua chân RxD thì tiến hành tách các bit Start và Stop để lấy ra 8 bit dữ liệu và đặt vào thanh ghi SBUF. Sau khi quá trình này hoàn tất, cờ RI được bật để báo rằng bộ vi điều khiển đã nhận xong một byte và cần phải được cất đi nếu không sẽ bị mất. Cờ RI được bật khi đang tách bit Stop. Chúng ta sẽ hiểu hơn cách sử dụng của các bit TI và RI qua một số ví dụ tiếp theo.

Lập trình 8051 truyền dữ liệu nối tiếp

Trình tự các bước khi lập trình 8051 truyền các byte ký tự nối tiếp như sau:

1. Nạp giá trị 20H vào thanh ghi TMOD báo sử dụng Timer 1 ở chế độ 2 (8 bit tự nạp) để thiết lập chế độ baud.
2. Nạp giá trị cho trong bảng 10.4 vào thanh ghi TH1 để thiết lập chế độ baud truyền dữ liệu nối tiếp (với giả thiết tần số XTAL = 11.0592MHz).
3. Nạp giá trị 50H vào thanh ghi SCON báo chế độ nối tiếp 1 để định khung 8 bit dữ liệu, 1 bit Start và 1 bit Stop.
4. Bật TR1 = 1 khởi động Timer 1.
5. Xoá bit TI bằng lệnh "CLR TI"
6. Ghi byte ký tự cần truyền vào SBUF.

7. Kiểm tra bit cờ TI bằng lệnh "JNB TI, xx" để báo hoàn tất việc chuyển ký tự.
8. Trở về bước 5 để truyền ký tự tiếp theo.

Ví dụ 10.2 sau đây giới thiệu chương trình truyền nối tiếp với tốc độ 4800 baud. Ví dụ 10.3 trình bày cách truyền liên tục chữ "YES".

Ví dụ 10.2

Hãy viết chương trình cho 8051 để truyền nối tiếp liên tục một ký tự "A" với tốc độ 4800 baud.

Giải:

```
        MOV    TMOD,#20H ;Chọn Timer1,chế độ 2(tự nạp lại)
        MOV    TH1,#-6   ;Chọn tốc độ 4800 baud
        MOV    SCON,#A"  ;8 bit,1 bit Stop, cho phép thu
        SETB   TR1       ;Khởi động Timer1
AGAIN:  MOV    SBUF,#"A" ;Truyền ký tự "A"
HERE:   JNB    TI,HERE   ;Chờ đến bit cuối cùng
        CLR    TI        ;Xoá bit TI cho ký tự kế tiếp
        SJMP   AGAIN     ;Tiếp tục gửi lại chữ A
```

Ví dụ 10.3

Hãy viết chương trình để truyền nối tiếp liên tục chữ "YES" với tốc độ 9600 baud (8 bit dữ liệu, 1 bit Stop).

Giải:

```
        MOV    TMOD,#20H    ;Chọn bộ Timer1, chế độ 2
        MOV    TH1,#-3      ;Chọn tốc độ 9600 baud
        MOV    SCON,#50H    ;8 bit DL,1 Stop, cho phép thu
        SETB   TR1          ;Khởi động Timer1
AGAIN:  MOV    A,# "Y"      ;Truyền ký tự "Y"
        ACALL  TRANS
        MOV    A,# "E"      ;Truyền ký tự "E"
        ACALL  TRANS
        MOV    A,# "S"      ;Truyền ký tự "S"
        ACALL  TRANS
        SJMP   AGAIN        ;Tiếp tục
;Chương trình con truyền dữ liệu nối tiếp.
```

```
TRANS: MOV   SBUF,A        ;Nạp SBUF
HERE:  JNB   TI,HERE       ;Chờ đến khi truyền bit cuối
       CLR   TI            ;Chờ sẵn cho byte kế tiếp
       RET
```

Vai trò của cờ TI

Để làm rõ vai trò của cờ ngắt TI, chúng ta xem xét trình tự thực hiện của 8051 khi truyền một ký tự qua đường TxD:

1. Ghi byte ký tự cần truyền vào SBUF.
2. Truyền bit Start.
3. Truyền 8 bit ký tự lượt từng bit một.
4. Truyền bit Stop. Trong khi truyền bit Stop thì 8051 bật cờ TI (TI = 1) để báo rằng ký tự đã được truyền xong và bộ vi điều khiển sẵn sàng để truyền ký tự tiếp .
5. Thông qua việc kiểm tra cờ TI có thể kiểm soát việc nạp dữ liệu vào thanh ghi SBUF. Nếu nạp một byte vào SBUF trước ghi cờ TI được bật thì phần dữ liệu của byte trước chưa truyền hết sẽ bị mất. Nói cách khác là 8051 bật cờ TI khi đã truyền xong một byte và sẵn sàng để truyền byte tiếp theo.
6. Sau khi một byte mới được nạp vào SBUF thì cờ TI bị xoá về 0 nhờ lệnh "CLR TI" để báo byte mới có thể được truyền.

Từ phần trình bày trên đây, có thể thấy rằng, thông qua việc kiểm tra bit cờ ngắt TI có thể biết được 8051 đã sẵn sàng để truyền một byte. Cần nhấn mạnh ở đây là bit cờ TI được 8051 bật khi đã hoàn tất việc truyền một byte dữ liệu, còn xoá cờ thì do lập trình viên thực hiện bằng lệnh "CLR TI". Cũng nên lưu ý là, nếu ghi một byte vào thanh ghi SBUF trước khi cờ TI được bật thì sẽ có nguy cơ mất phần dữ liệu chưa kịp truyền. Bit cờ TI có thể kiểm tra bằng lệnh "JNB TI ..." hoặc sử dụng ngắt như sẽ được đề cập tới trong chương 11.

Lập trình 8051 nhận dữ liệu nối tiếp

Lập trình 8051 để nhận các byte ký tự nối tiếp cần thực hiện các bước sau:

1. Nạp giá trị 20H vào thanh ghi TMOD báo sử dụng bộ Timer1, chế độ 2 (8 bit, tự động nạp lại) để thiết lập tốc độ baud.
2. Nạp giá trị cho trong bảng 10.4 vào TH1 để tạo ra tốc độ baud với giả thiết XTAL = 10.0592MHz.
3. Nạp giá trị 50H vào thanh ghi SCON báo sử dụng Mode 1 truyền nối tiếp là

định khung 8 bit dữ liệu, 1 bit Start và 1 bit Stop.

4. Bật TR1 = 1 để khởi động Timer 1.
5. Xoá cờ ngắt RI bằng lệnh "CLR RI".
6. Bit cờ RI được hiển thị bằng lệnh "JNB RI, xx" để xem toàn bộ ký tự đã được nhận chưa.
7. Khi RI được thiết lập thì trong SBUF đã có 1 byte. Các nội dung của nó được cất lưu vào một nơi an toàn.
8. Quay trở về bước 5 để nhận một ký tự tiếp theo.

Ví dụ 10.4

Hãy lập trình cho 8051 để nhận các byte dữ liệu nối tiếp và đặt chúng vào cổng P1. Đặt tốc độ baud là 4800, 8 bit dữ liệu và 1 bit Stop.

Giải:

```
        MOV   TMOD,#20H ;Chọn Timer1,chế độ 2(tự nạp lại)
        MOV   TH1,#-6   ;Chọn tốc độ 4800 baud
        MOV   SCON,#50H ;Chọn khung dữ liệu 8 bit Stop
        SETB  TR1       ;Khởi động bộ Timer1
HERE:   JNBR1,HERE      ;Đợi nhận hết toàn bộ ký tự
        MOV   A,SBUF    ;Lưu cất ký tự vào thanh A
        MOV   P1,A      ;Gửi ra cổng P.1
        CLR   RI        ;Sẵn sàng nhận byte kế tiếp
        SJMP  HERE      ;Tiếp tục nhận dữ liệu
```

Ví dụ 10.5

Giả sử cổng nối tiếp của 8051 được nối vào cổng COM của máy tính IBM PC và chương trình để gửi và nhận dữ liệu nối tiếp ở máy tính PC là Terminal.Exe. Cổng P1 và P2 của 8051 được nối tương ứng tới đèn LED và các công tắc chuyển mạch. Hãy viết một chương trrình cho 8051:

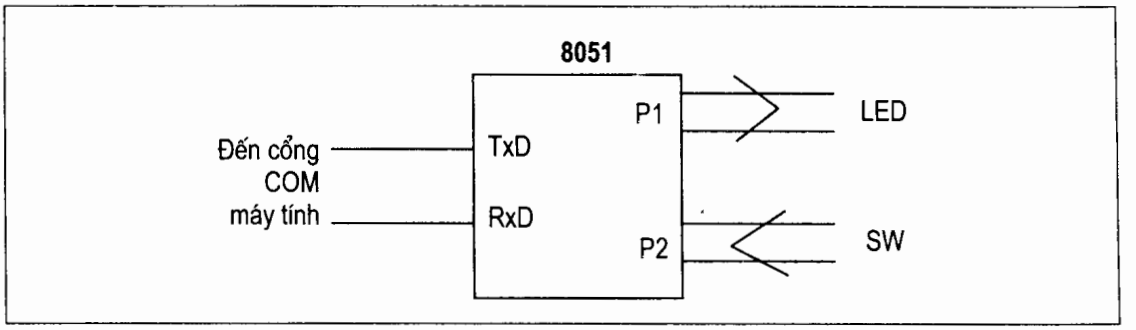
a) Gửi thông báo "Đã sẵn sàng" tới máy tính PC.

b) Nhận bất kỳ dữ liệu gì được PC gửi đến và chuyển đến đèn LED đang được nối đến các chân của cổng P1.

c) Nhận dữ liệu trên các chuyển mạch được nối tới P2 và gửi nối tiếp tới máy tính PC. a) được thực hiện một lần, nhưng b) và c) chạy liên tục với tốc độ 4800 baud.

Giải:

```
      ORG    0
      MOV    P2,0FFH       ;Lấy cổng P2 làm cổng vào
      MOV    TMOD,#20H     ;Chọn Timer1,Mode 2 (tự nạp lại)
      MOV    TH1,#0FAH     ;Chọn tốc độ 4800 baud
      MOV    SCON,#50H     ;Định khung 8 bit dữ liệu,
                           ;1 Stop,cho phép REN
      SETB   TR1           ;Khởi động bộ Timer1
      MOV    DPTR,#MYDATA  ;Nạp con trỏ đến thông báo
H-1:  CLR    A
      MOVC   A,'A+DPTR     ;Lấy ký tự
      JZ     DPTR          ;Nếu ký tự cuối cùng muốn gửi ra
      ACALL  SEND          ;Nếu chưa thì gọi trình con SEND
      INC    DPTR          ;Chạy tiếp
      SJMP   H-1           ;Quay lại vòng lặp
B-1:  MOV    A,P2          ;Đọc dữ liệu trên cổng P2
      ACALL  RECV          ;Truyền nối tiếp
      ACALL  RECV          ;Nhận dữ liệu nối tiếp
      MOV    F1,A          ;Hiển thị ra đèn LED
      SJMP   B - 1         ;Ở lại vòng lặp vô hạn
;--- Truyền dữ liệu nối tiếp. ACC có dữ liệu
SEND: MOV    SBUF,A        ;Nạp dữ liệu
 H-2: JNB    TI,H-2        ;Ở lại vòng lặp vô hạn
      CLR    TI            ;Truyền dữ liệu nối tiếp
      RET
;--- Nhận dữ liệu nối tiếp vào ACC
RECV: JNB    RI,RECV       ;Nạp dữ liệu
      MOV    A,SBUF        ;ở lại cho đến khi gửi xong
      CLR    RI            ;Sẵn sàng cho ký tự mới
      RET
;--- Thông báo
MYDATA:
      DB "Đã sẵn sàng",0
      END
```



Vai trò của cờ RT

Các bước cần thực hiện khi 8051 nhận các bit dữ liệu qua chân RxD như sau:

1. Nhận bit Start báo rằng bit sau đó là bit dữ liệu đầu tiên cần nhận.
2. Ký tự 8 bit được nhận lần lượt từng bit một. Khi bit cuối cùng được nhận thì hình thành một byte và được đặt vào trong SBUF.
3. Khi nhận được bit Stop thì 8051 bật RI = 1 để báo rằng toàn bộ ký tự đã được nhận xong và cần được cất đi trước khi có byte mới nhận về sẽ ghi đè lên.
4. Nhờ kiểm tra trạng thái bật lên 1 của bit cờ RI mà có thể biết một ký tự đã được nhận và đang nằm trong SBUF. Cần sao lại nội dung SBUF vào nơi an toàn ở một số thanh ghi hay bộ nhớ trước khi có thể bị mất.
5. Sau khi SBUF được ghi vào nơi an toàn xong thì cờ RI được xoá về 0 bằng lệnh "CLR RI" để cho phép chuyển vào SBUF các ký tự vừa nhận được tiếp theo. Nếu không thực hiện được bước này thì ký tự vừa nhận được sẽ bị mất.

Từ những mô tả trên đây, chúng ta có thể nhận thấy rằng, thông qua việc kiểm tra cờ RI, chúng ta có thể biết được 8051 đã nhận được một byte ký tự hay chưa. Nếu không sao được nội dung của thanh ghi SBUF vào nơi an toàn thì có nguy cơ mất ký tự vừa nhận được. Và cũng cần nhấn mạnh rằng cờ RI được 8051 bật lên, nhưng để xoá thì do lập trình viên thực hiện bằng lệnh "CLR RI". Bit cờ RI có thể được kiểm tra bởi lệnh "JNB RI, xx" hoặc bằng ngắt mà chúng ta sẽ bàn đến ở chương 11.

Nhân đôi tốc độ baud của 8051

Có hai cách để tăng tốc độ baud truyền dữ liệu của 8051:

1. Sử dụng tần số thạch anh cao hơn.
2. Thay đổi một bit trong thanh ghi điều khiển công suất PCON (Power Control).

D7							D0
SMOD	–	–	–	GF0	GF0	PD	IDL

Phương pháp 1 không khả thi trong nhiều trường hợp vì tần số thạch anh của hệ thống là cố định. Nhưng lý do không khả thi chủ yếu là vì tần số thạch anh mới không tương thích với tốc độ baud của cổng COM nối tiếp của máy tính IBM PC. Do vậy, chúng ta sẽ tập trung tìm hiểu phương pháp 2. Có một giải pháp có thể nhân đôi tần số baud bằng phần mềm với tần số thạch anh không đổi nhờ sử dụng thanh ghi 8 bit PCON. Trong 8 bit của PCON có một số bit không được dùng và một số bit khác được dùng để điều khiển công suất của 8051. Bit dành cho truyền thông là bit chế độ nối tiếp SMOD D7 (Serial Mode). Khi 8051 được bật nguồn thì bit D7 của thanh ghi PCON ở mức thấp 0. Chúng ta có thể đặt bit này lên 1 bằng phần mềm và do vậy nhân đôi được tốc độ baud. Thứ tự các lệnh được sử dụng để thiết lập bit D7 lên cao như sau (vì PCON không phải là thanh ghi định địa chỉ bit).

```
MOV    A,PCON    ;Đặt bản sao của PCON vào ACC
SETB   ACC.7     ;Đặt D7 của ACC lên 1
MOV    PCON,A    ;Bây giờ SMOD=1, các bit khác không đổi
```

Để hiểu tại sao tốc độ baud được tăng gấp đôi, chúng ta xem xét vai trò của bit SMOD khi có giá trị 0 và 1.

SMOD = 0

Khi SMOD = 0 thì tần số của bộ Timer 1 dùng để thiết lập tốc độ baud được xác định bằng 1/12 tần số thạch anh chia cho 32. Trường hợp XTAL = 11.0592MHz thì ta có: Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{32} = 28.800\text{Hz}$ vì SMOD = 0.

Đây là tần số Timer 1 sử dụng để đặt tốc độ baud và cũng là giá trị được tính ở tất cả các ví dụ từ trước đến giờ vì đó là giá trị mặc định của 8051 khi bật nguồn. Tốc độ baud khi SMOD = 0 được cho ở bảng 10.4.

SMOD = 1

Với tần số thạch anh cố định ta có thể nhân đôi tốc độ baud bằng cách đặt bit SMOD = 1. Khi bit D7 của PCON (bit SMOD) được đưa lên 1 thì 1/12 tần số XTAL được chia cho 16 (thay vì chia cho 32 như trường hợp SMOD = 0) và đây

là tần số được Timer 1 dùng để thiết lập tốc độ baud. Trong trường hợp XTAL = 11.0592MHz ta có:

Tần số chu trình máy $= \frac{11.0592\text{MHz}}{12} = 921.6\text{kHz}$ và $\frac{921.6\text{kHz}}{16} = 57.600\text{kHz}$ vì SMOD = 1.

Bảng 10.5 là các giá trị cần nạp vào TH1 và trong cả hai trường hợp giá trị này như nhau, tuy nhiên tốc độ baud đã được tăng gấp đôi khi SMOD=1.

Bảng 10.5. So sánh tốc độ baud khi SMOD=0 và SMOD=1

TH1 (thập phân)	TH1 (Hexa)	Tốc độ baud	
		SMOD = 0	SMOD = 1
-3	FD	9,600	19,200
-6	FA	4,800	9,600
-12	F4	2,400	4,800
-24	F8	1,200	2,400

Ví dụ 10.6

Giả sử tần số XTAL = 11.0592MHz. Hãy xác định:

a) Chương trình dưới đây có nhiệm vụ gì?

b) Hãy tính tần số Timer 1 sử dụng để đặt tốc độ baud?

c) Hãy tìm tốc độ baud truyền dữ liệu.

```
        MOV   A,PCON        ;Sao thanh ghi PCON vào ACC
        SETB  ACC.7         ;Đặt D7=0
        MOV   PCON,A        ;Đặt SMOD=1 nhân đôi tần số baud
        MOV   TMOD,#20H     ;Chọn Timer1,Mode2,tự nạp lại
        MOV   TH1,-3        ;Tốc độ baud 19200
                            ;(57600/3=19200) vì SMOD=1
        MOV   SCON,#50H     ;Khung 8 bit,1 Stop,cho phép RI
        SETB  TR1           ;Khởi động Timer1
        MOV   A,#"B"        ;Truyền ký tự B
A-1:    CLR   TI            ;Đảm bảo TI=0
        MOV   SBUF,A        ;Truyền
H-1:    JNB   TI,H-1        ;Chờ đến bit cuối được gửi đi
        SJMP  A-1           ;Tiếp tục gửi"B"
```

Giải:

a) Chương trình này truyền liên tục mã ASCII chữ B (ở dạng nhị phân là 0100 0010).

b) Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có:

11.0592MHz/12 = 921.6kHz là tần số chu trình máy.

921.6kHz/16 = 57.6kHz là tần số được Timer1 sử dụng để đặt tốc độ baud.

c) 57.6kHz/3 = 19.200 là tốc độ cần tìm.

Ví dụ 10.7

Cho tần số XTAL = 11.0592MHz, hãy tìm giá trị TH1 (ở dạng thập phân và hexa) để đặt tốc độ baud cho các trường hợp sau.

a) 9600.

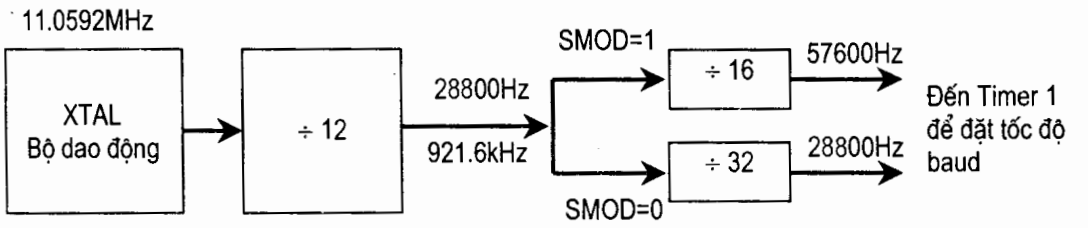
b) 4800 nếu SMOD = 1.

Giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer 1 là 57.6kHz.

a) 57.600/9600 = 6 do vậy TH1 = - 6 hay TH1 = FAH.

b) 57.600/4800 = 12 do vậy TH1 = - 12 hay TH1 = F4H.



Ví dụ 10.8

Hãy tìm tốc độ baud nếu TH1=-2, SMOD=1 và tần số XTAL = 11.0592MHz. Tốc độ này có được máy tính IBM PC và tương thích hỗ trợ không?

Giải:

Với tần số XTAL = 11.0592MHz và SMOD = 1 ta có tần số cấp cho Timer1 là 57.6kHz. Tốc độ baud là 57.600kHz/2 = 28.800. Tốc độ này không được máy tính IBM PC và tương thích hỗ trợ. Tuy nhiên, PC có thể được lập trình để truyền dữ liệu với tốc độ này. Phần mềm của nhiều modem cũng như

Hyperterminal của Windows 95 có thể hỗ trợ tốc độ này và một số tốc độ khác nữa.

Truyền dữ liệu dùng phương pháp ngắt

Cũng phải thấy rằng, thật lãng phí thời gian khi để các bộ vi điều khiển liên tục kiểm tra trạng thái và bật các cờ TI và RI. Do vậy, để tăng hiệu suất của 8051 ta có thể lập trình các cổng truyền thông nối tiếp bằng cách sử dụng ngắt. Đây chính là nội dung của chương 11 tiếp sau đây.

Chương 11
NGẮT VÀ LẬP TRÌNH

Ngắt là sự đáp ứng các sự kiện bên trong hoặc bên ngoài nhằm thông báo cho bộ vi điều khiển biết thiết bị đang cần được phục vụ. Trong chương này chúng ta tìm hiểu về ngắt và cách lập trình.

11.1 NGẮT CỦA 8051

Phương pháp ngắt và phương pháp thăm dò

Một bộ vi điều khiển có thể phục vụ một số thiết bị. Có hai phương pháp phục vụ thiết bị, đó là sử dụng ngắt và phương pháp thăm dò (polling).

Ở phương pháp ngắt, mỗi khi có một thiết bị cần được phục vụ thì thiết bị sẽ báo cho bộ vi điều khiển bằng cách gửi một tín hiệu ngắt. Khi nhận được tín hiệu này, bộ vi điều khiển ngừng mọi công việc đang thực hiện để chuyển sang phục vụ thiết bị. Chương trình đi cùng với ngắt được gọi là *trình phục vụ ngắt* ISR (Interrupt Service Routine) hay còn gọi là *bộ quản ngắt* (Interrupt handler).

Đối với phương pháp thăm dò, bộ vi điều khiển liên tục kiểm tra tình trạng của thiết bị và khi điều kiện được đáp ứng thì tiến hành phục vụ thiết bị. Sau đó, bộ vi điều khiển chuyển sang kiểm tra trạng thái của thiết bị tiếp theo cho đến khi tất cả đều được phục vụ.

Điểm mạnh của phương pháp ngắt là bộ vi điều khiển có thể phục vụ được nhiều thiết bị, nhưng dĩ nhiên là không cùng một thời điểm. Mỗi thiết bị có thể được bộ vi điều khiển phục vụ dựa theo mức ưu tiên được gán. Ở phương pháp thăm dò thì không thể gán mức ưu tiên cho thiết bị được vì bộ vi điều khiển tiến hành kiểm tra các thiết bị theo kiểu hỏi vòng. Ngoài ra, phương pháp ngắt cho phép bộ vi điều khiển che hoặc bỏ qua một yêu cầu phục vụ của thiết bị, điều mà phương pháp thăm dò không thực hiện được. Song, lý do chính mà phương pháp ngắt được ưa chuộng hơn là vì, phương pháp thăm dò lãng phí đáng kể thời gian của bộ vi điều khiển do phải hỏi dò từng thiết bị, kể cả khi chúng không cần phục vụ. Để làm rõ hơn vấn đề này, chúng ta xem lại ví dụ được nêu ở chương 9, trong đó có lệnh "JNB TF, đích" đợi cho đến khi bộ định thời quay trở về 0. Ở ví dụ này, trong khi chờ đợi bật cờ TF thì bộ vi điều khiển không thể làm được công việc gì khác, do vậy bị lãng phí thời gian. Cũng với bộ định thời này, nếu dùng

phương pháp ngắt, bộ vi điều khiển có thể thực hiện một số công việc nào đó cho đến khi cờ TF bật lên, bộ vi điều khiển sẽ bị ngắt cho dù đang làm công việc gì.

Trình phục vụ ngắt

Mỗi ngắt luôn có một trình phục vụ ngắt. Khi một ngắt được kích hoạt thì bộ vi điều khiển chạy trình phục vụ ngắt. Trình phục vụ ngắt của mỗi ngắt có một vị trí cố định trong bộ nhớ để giữ địa chỉ ISR. Tập hợp các ô nhớ lưu giữ địa chỉ của tất cả các ISR được gọi là bảng vector ngắt (xem bảng 11.1).

Trình tự thực hiện ngắt

Khi một ngắt được kích hoạt, trình tự thực hiện của bộ vi điều khiển như sau:

1. Kết thúc lệnh hiện tại và lưu địa chỉ của lệnh kế tiếp (PC) vào ngăn xếp.
2. Lưu lại trạng thái hiện hành của tất cả các ngắt vào bên trong (nghĩa là không lưu vào ngăn xếp).
3. Nhảy đến một vị trí cố định trong bộ nhớ được gọi là bảng vector ngắt, nơi lưu giữ địa chỉ của trình phục vụ ngắt.
4. Nhận địa chỉ ISR từ bảng vector ngắt rồi nhảy tới địa chỉ đó và bắt đầu thực hiện trình phục vụ ngắt cho đến lệnh cuối cùng của ISR là RETI (trở về từ ngắt).
5. Kết thúc trình phục vụ ngắt, bộ vi điều khiển gặp lệnh RETI và trở về nơi nó đã bị ngắt. Trước hết, hai byte của đỉnh ngăn xếp được nạp vào bộ đếm chương trình PC, tiếp theo, bộ vi điều khiển bắt đầu thực hiện lệnh tại địa chỉ đó.

Lưu ý vai trò quan trọng của ngăn xếp ở bước 5. Vì lý do này mà chúng ta cần phải cẩn thận khi thao tác với ngăn xếp khi lưu địa chỉ của ISR. Không riêng gì đối với ISR, mà đối với bất kỳ chương trình con nào được gọi, số lần đẩy vào ngăn xếp (PUSH) và số lần lấy ra (POP) phải bằng nhau.

Sáu ngắt của 8051

Thực tế ở 8051 chỉ có 5 ngắt dành cho người dùng, song nhiều tài liệu kỹ thuật của các nhà sản xuất vẫn nói rằng có 6 ngắt vì họ tính cả lệnh RESET. Bố trí sáu ngắt của 8051 như sau:

1. RESET: Khi chân RESET được kích hoạt thì 8051 nhảy về địa chỉ 0000. Đây là địa chỉ bật lại nguồn đã được bàn ở chương 4.
2. Hai ngắt dành cho bộ định thời Timer 0 và Timer 1. Địa chỉ ở bảng vector ngắt của hai ngắt này tương ứng với Timer 0 và Timer 1 là 000B4 và 001B4.
3. Hai ngắt phần cứng dành cho các thiết bị bên ngoài nối tới chân 12 (P3.2) và 13 (P3.3) của cổng P3 là INT0 và INT1 tương ứng. Các ngắt ngoài cũng còn được

gọi là EX1 và EX2. Vị trí nhớ trong bảng vector ngắt của hai ngắt INT0 và INT1 này là 0003H và 0013H.

4. Truyền thông nối tiếp có một ngắt cho cả thu lẫn phát. Địa chỉ của ngắt này trong bảng vector ngắt là 0023H.

Bảng 11.1. Bảng vector ngắt của 8051

Ngắt	Địa chỉ ROM (Hexa)	Chân
RESET	0000	9
Ngắt phần cứng ngoài (INT0)	0003	12 (P3.2)
Ngắt bộ Timer 0 (TF0)	000B	
Ngắt phần cứng ngoài 1 (INT1)	0013	13 (P3.3)
Ngắt bộ Timer 1 (TF1)	001B	
Ngắt COM nối tiếp (RI và TI)	0023	

```
   ORG    0      ;địa chỉ bắt đầu ở ROM khi khởi động lại
   LJMP   MAIN   ;bỏ qua bảng vector ngắt
;--- chương trình khởi động lại
   ORG30H
MAIN:
   . . .
   END
```

Hình 11.1. Định lại địa chỉ khi bật nguồn ở 8051

Chú ý rằng, trong bảng 11.1 có số lượng byte hạn chế dành riêng cho từng ngắt. Ví dụ, đối với ngắt INT0 (ngắt cứng ngoài số 0) thì được dành tổng cộng là 8 byte từ địa chỉ 0003H đến 000AH. Tương tự, 8 byte từ địa chỉ 000BH đến 0012H là dành cho ngắt bộ định thời Timer 0 là TF0. Nếu trình phục vụ ngắt của một ngắt đủ ngắn để vừa trong không gian nhớ được thì trình đó được đặt trong bảng vector ngắt, còn nếu không vừa thì ở bảng vector ngắt sẽ được đặt lệnh LJMP để trỏ đến địa chỉ của ISR. Ở trường hợp này, các byte rỗi còn lại của ngắt sẽ không dùng đến. Dưới đây là một số ví dụ minh hoạ về lập trình ngắt.

Từ bảng 11.1 có thể thấy rằng, chỉ có 3 byte không gian bộ nhớ ROM được dành cho RESET. Đó là địa chỉ 0, 1 và 2. Địa chỉ 3 thuộc về ngắt phần cứng ngoài - ngắt 0. Vì lý do đó, trong chương trình cần đặt lệnh LJMP làm lệnh đầu tiên để hướng bộ xử lý ra ngoài bảng vectơr ngắt như chỉ ra ở hình 11.1.

Cho phép ngắt và cấm ngắt

Khi Reset thì tất cả mọi ngắt đều bị cấm (bị che), có nghĩa là không có ngắt nào được bộ vi điều khiển đáp ứng nếu chúng được kích hoạt. Các ngắt phải được cho phép bằng phần mềm để bộ vi điều khiển có thể đáp ứng được. Có một thanh ghi được gọi là cho phép ngắt IE (Interrupt Enable) chịu trách nhiệm về việc cho phép (không che) và cấm (che) các ngắt. Hình 11.2 giới thiệu thanh ghi IE, lưu ý rằng IE là thanh ghi có thể định địa chỉ bit.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Nếu EA = 0 thì không ngắt nào được báo nhận. Nếu EA = 1 thì từng nguồn ngắt sẽ được mở hoặc cấm bằng cách bật hoặc xoá bit cho phép tương ứng.
- -	IE.6	Dự phòng cho tương lai. *
ET2	IE.5	Cho phép hoặc cấm ngắt tràn hoặc thu của Timer 2 (8952)
ES	IE.4	Cho phép hoặc cấm ngắt cổng nối tiếp
ET1	IE.3	Cho phép hoặc cấm ngắt tràn của Timer 1
EX1	IE.2	Cho phép hoặc cấm ngắt ngoài 1
ET0	IE.1	Cho phép hoặc cấm ngắt tràn của Timer 0
EX0	IE.0	Cho phép hoặc cấm ngắt ngoài 0

* Các bit này có thể dùng ở các bộ vi điều khiển có đặc tính mới trong tương lai.

Hình 11.2. Thanh ghi cho phép ngắt IE

Ở hình 11.2, bit D7 của thanh ghi IE được gọi là bit cho phép tất cả các ngắt EA (Enable All). Để thanh ghi làm việc, bit này cần được thiết lập lên 1. Bit D6 không sử dụng. Bit D5 là dành cho 8052, còn bit D4 dùng cho ngắt nối tiếp v.v.

Các bước thực hiện khi cho phép một ngắt

Để cho phép một ngắt, trình tự thực hiện các bước như sau:

1. Bit D7 của thanh ghi IE là EA phải được bật lên cao cho phép các bit còn lại của thanh ghi có hiệu lực.
2. Nếu EA = 1 thì tất cả mọi ngắt đều được phép và sẽ được đáp ứng nếu các bit tương ứng của ngắt này trong IE có mức cao. Nếu EA = 0 thì không có ngắt nào được đáp ứng, cho dù bit tương ứng trong IE có giá trị cao.

Để hiểu rõ hơn vấn đề này, chúng ta xem ví dụ 11.11 sau:

Ví dụ 11.1

Hãy viết các lệnh thực hiện:

a) Cho phép ngắt nối tiếp, ngắt Timer 0 và ngắt phần cứng ngoài 1 (EX1).

b) Cấm (che) ngắt Timer 0.

c) Cấm tất cả mọi ngắt chỉ bằng một lệnh duy nhất.

Giải:

```
a)MOV  IE,#10010110B   ;Cho phép ngắt nối tiếp, Timer 0
                       ;và cho phép ngắt phần cứng ngoài
```

Vì IE là thanh ghi định địa chỉ bit, nên có thể sử dụng các lệnh sau đây để truy cập đến các bit riêng rẽ của thanh ghi:

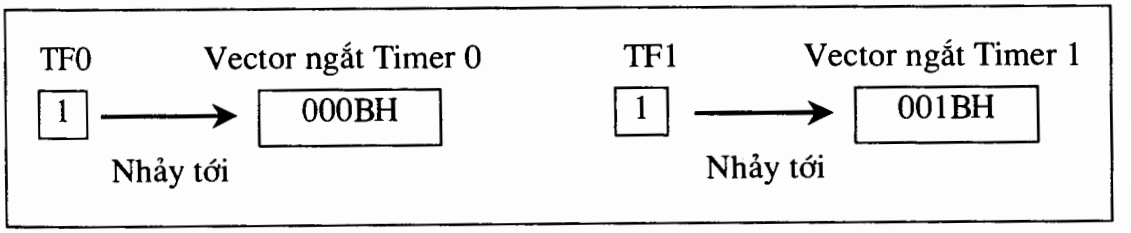
```
   SETB   IE.7      ;EA=1, cho phép tất cả mọi ngắt
   SETB   IE.4      ;Cho phép ngắt nối tiếp
   SETB   IE.1      ;Cho phép ngắt Timer 1
   SETB   IE.2      ;Cho phép ngắt phần cứng ngoài 1
```

Tất cả các lệnh này tương đương với lệnh "MOV IE, #10010110B" nêu ở trên.

```
b)CLR  IE.1       ;Xoá(che)ngắt Timer 0
c)CLR  IE.7       ;Cấm tất cả mọi ngắt.
```

11.2 LẬP TRÌNH NGẮT BỘ ĐỊNH THỜI

Trong chương 9 chúng ta đã giới thiệu cách sử dụng các bộ định thời Timer 0 và Timer 1 bằng phương pháp thăm dò. Trong phần này chúng ta sẽ sử dụng các ngắt để lập trình cho các bộ định thời của 8051.



Hình 11.3. Cờ ngắt bộ định thời TF0 và TF1

Cờ quay về 0 của bộ định thời và ngắt

Ở chương 9 chúng ta đã biết, cờ bộ định thời TF được đặt lên cao khi bộ định thời đạt giá trị cực đại và quay trở về 0 (Roll - over). Chúng ta cũng đã dùng lệnh

"JNB TF, đích" để kiểm tra trạng thái cờ TF. Phương pháp này có nhược điểm là trong quá trình kiểm tra cờ TF, bộ vi điều khiển không thể làm được việc gì khác. Để khắc phục nhược điểm này, người ta sử dụng phương pháp ngắt. Nếu bit ngắt bộ định thời ở thanh ghi IE được phép thì mỗi khi bộ định thời quay trở về 0, cờ TF được bật lên và bộ vi điều khiển ngừng mọi việc đang thực hiện và nhảy tới bảng vector ngắt để phục vụ ISR. Bằng cách đó, bộ vi điều khiển có thể làm những công việc khác cho đến khi nào được thông báo rằng bộ định thời đã quay về 0. Xem hình 11.3 và ví dụ 11.2.

Ví dụ 11.2

Viết chương trình nhận liên tục dữ liệu 8 bit ở cổng P0 và gửi đến cổng P1. Trong thời gian này cần tạo ra trên chân P2.1 một sóng vuông chu kỳ 200 µs. Sử dụng bộ Timer 0 để tạo ra sóng vuông, tần số của 8051 là XTAL = 11.0592MHz.

Giải:

Sử dụng bộ Timer 0 ở chế độ 2 (tự nạp lại), giá trị nạp cho TH0 là 100/1.085µs = 92.

```
;--Khi khởi tạo, nhảy đến chương trình main để tránh dùng
;không gian nhớ dành cho bảng vector ngắt.
      ORG   0000H
      LJMP  MAIN          ;Bỏ qua bảng vector ngắt
;--Chương trình ISR cho Timer 0 để tạo sóng vuông
      ORG   000BH         ;Bảng vector ngắt của Timer 0
      CPL   P2.1          ;Đảo mức chân P2.1
      RETI                ;Trở về chương trình
;--Chương trình chính
      ORG   0030H         ;Địa chỉ sau bảng vector ngắt
MAIN: MOV   TMOD,#02H     ;Chọn Timer 0,chế độ 2 tự nạp lại
      MOV   P0,#0FFH      ;Lấy P0 làm cổng vào
      MOV   TH0,#-92      ;Đặt TH0=A4H cho-92
      MOV   IE,#82H       ;IE=1000 0010 cho phép Timer 0
      SETB  TR0           ;Khởi động bộ Timer 0
BACK: MOV   A,P0          ;Nhận dữ liệu từ cổng P0
      MOV   P1,A          ;Chuyển dữ liệu đến cổng P1
      SJMP  BACK          ;Tiếp tục nhận và chuyển dữ liệu
                          ;chừng nào chưa bị TF0 ngắt
      END
```

Ở chương trình ví dụ 11.2 cần lưu ý những điểm sau:

1. Trình ứng dụng không được sử dụng không gian bộ nhớ dành cho bảng vector ngắt. Do vậy, mã chương trình được đặt từ địa chỉ 30H. Lệnh LJMP là lệnh đầu tiên 8051 thực hiện khi được cấp nguồn. Lệnh LJMP lái bộ điều khiển tránh khỏi bảng vector ngắt.
2. Trình phục vụ ISR của bộ Timer 0 là trình có kích thước nhỏ và được đặt gọn trong không gian nhớ dành cho ngắt này ở bảng vector ngắt bắt đầu từ địa chỉ 000BH.
3. Cũng có thể thực hiện ngắt Timer 0 bằng lệnh "MOV IE, #1000 010H" ở chương trình chính MAIN.
4. Dữ liệu được nhận vào ở cổng P0 sẽ liên tục được chuyển sang cổng P1. Trong thời gian này, mỗi khi bộ Timer 0 quay trở về 0 thì cờ TF0 được bật lên, bộ vi điều khiển thoát ra khỏi vòng lặp BACK và nhảy đến địa chỉ 000BH để thực hiện trình phục vụ ISR của bộ Timer 0.
5. Ở trình phục vụ ngắt ISR của Timer 0 không cần sử dụng lệnh "CLR TF0" trước lệnh RETI. Sở dĩ như vậy là vì 8051 xoá bên trong cờ TF khi nhảy đến bảng vector ngắt.

Ở ví dụ 11.2, trình phục vụ ngắt ISR nhỏ nên có thể đặt vừa trong không gian địa chỉ dành cho ngắt Timer 0 trong bảng vector ngắt. Tất nhiên không phải lúc nào cũng làm được như vậy. Xem ví dụ 11.3.

Ví dụ 11.3

Hãy viết lại chương trình ở ví dụ 11.2 để tạo sóng vuông với mức cao kéo dài 1085μs và mức thấp dài 15μs. Giả thiết tần số XTAL = 11.0592MHz, sử dụng bộ định thời Timer 1.

Giải:

1085μs = 1000 × 1,085 nên cần sử dụng chế độ 1 của bộ định thời Timer 1.

```
;--Khi khởi tạo tránh dùng không gian nhớ dành cho
;bảng vector ngắt.
      ORG   0000H
      LJMP  MAIN      ;Chuyển đến bảng véctơ ngắt.
;--Trình ISR cho Timer 1 để tạo ra xung vuông
      ORG   001BH     ;Địa chỉ ngắt Timer 1
      LJMP  ISR_T1    ;Nhảy đến ISR
```

```
;--Bắt đầu chương trình chính MAIN
       ORG    0030H          ;Sau bảng vector ngắt
 MAIN: MOVTMOD,#10H          ;Chọn Timer 1 chế độ 1
       MOV    P0,#0FFH       ;Chọn cổng P0 làm đầu vào
       MOV    TL1,#018H      ;Đặt TL1=18
       MOV    TH1,#0FCH      ;Đặt TH1=FC
       MOV    IE,#88H        ;IE=10001000cho phép ngắt Timer
1
       SETB   TR1            ;Khởi động Timer 1
BACK:  MOV    A,P0           ;Nhận dữ liệu đầu vào ở cổng P0
       MOV    P1,A           ;Chuyển dữ liệu đến P1
       SJMP   BACK           ;Tiếp tục nhận và chuyển dữ liệu
;--Trình ISR của Timer 1 phải được nạp lại vì ở chế độ 1
ISR-T1:CLR    TR1            ;Dừng Timer 1
       CLR    P2.1           ;P2.1=0 bắt đầu xung mức thấp
       MOV    R2,#4          ;2 chu kỳ máy MC (Machine Cycle)
HERE:  DJNZ   R2,HERE        ;4×2 MC=8 MC
       MOV    TL1,#18H       ;Nạp lại byte thấp giá trị 2 MC
       MOV    TH1,#0FCH      ;Nạp lại byte cao giá trị 2 MC
       SETB   TR1            ;Khởi động Timer 1 1 MC
       SETB   P2.1           ;P2.1=1 bật P2.1 trở lại cao
       RETI                  ;Trở về chương trình chính
       END
```

Lưu ý rằng phần xung mức thấp được tạo ra bởi 14 chu kỳ máy MC (Machine Cycle), mỗi MC = 1.085μs và 14 × 1.085μs = 15.19μs.

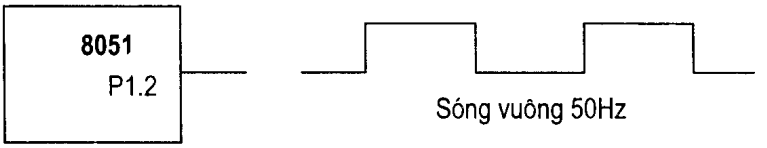
Ví dụ 11.4

Viết chương trình tạo ra sóng vuông tần số 50Hz trên chân P1.2. Ví dụ này tương tự ví dụ 9.12 ngoại trừ sử dụng ngắt Timer 0, giả thiết XTAL = 11.0592MHz.

Giải:

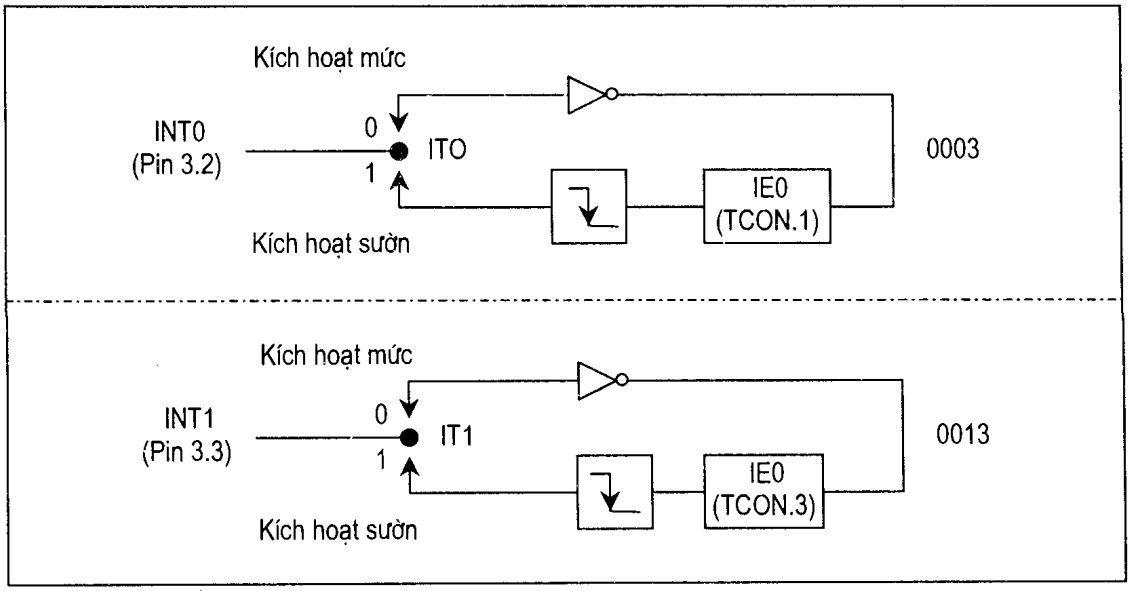
```
       ORG    0
       LJMP   MAIN
       ORG    000BH              ;ISR cho Timer 0
       CPL    P1.2
```

```
        MOV     TL0,#00
        MOV     TH0,#0DCH
        RETI
        ORG     30H
;--Chương trình chính
 MAIN:  MOV     TMOD,#00000001B     ;Chọn Timer 0 chế độ 1
        MOV     TL0,#0DCH
        MOV     IE,#82H             ;Cho phép ngắt Timer 0
        SETB    TR0
 HERE:  SJMP    HERE
        END
```



11.3 LẬP TRÌNH NGẮT PHẦN CỨNG NGOÀI

Bộ vi điều khiển 8051 có hai ngắt phần cứng bên ngoài là chân 12 (P3.2) và chân 13 (P3.3) dùng cho ngắt INT0 và INT1. Khi kích hoạt những chân này thì 8051 ngừng ngay mọi công việc đang thực hiện và nhảy đến bảng vector ngắt để chạy trình phục vụ ngắt.



Hình 14.4. Kích hoạt INT0 và INT1

Ngắt ngoài INT0 và INT1

Chỉ có hai ngắt phần cứng ngoài của 8051 là INT0 và INT1. Hai ngắt này được bố trí trên chân P3.2 và P3.3 và địa chỉ trong bảng vector ngắt là 0003H và 0013H. Như đã đề cập ở mục 11.1, các ngắt này được phép và bị cấm bởi thanh ghi IE. Vậy chúng được kích hoạt như thế nào? Có hai cách kích hoạt ngắt phần cứng ngoài, đó là theo mức và theo sườn. Dưới đây sẽ đề cập đến từng loại.

Ngắt kích phát mức

Ở chế độ ngắt theo mức, các chân INT0 và INT1 bình thường ở mức cao, giống như tất cả các chân của cổng I/O. Nếu có tín hiệu mức thấp cấp tới thì tín hiệu này kích hoạt ngắt. Khi đó, bộ vi điều khiển dừng tất cả mọi công việc đang thực hiện và nhảy đến bảng vector ngắt để phục vụ ngắt. Ngắt kích hoạt theo phương pháp này được gọi là *kích phát mức* hay *kích hoạt mức,* và là chế độ mặc định khi Reset 8051. Trước khi thực hiện lệnh cuối cùng của trình phục vụ ngắt RETI, thì mức thấp tại chân INT phải chuyển sang cao, nếu không sẽ tạo ra một ngắt khác. Nói cách khác, nếu vẫn duy trì mức thấp khi ISR kết thúc thì 8051 sẽ hiểu là có một ngắt mới và nhảy đến bảng vector ngắt để thực hiện ISR. Xem ví dụ 11.5.

Ví dụ 11.5

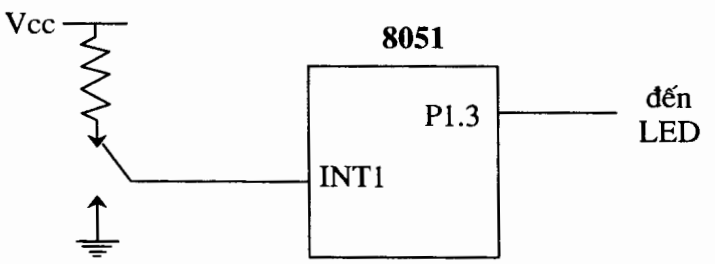
Giả sử chân INT1 được nối đến công tắc bình thường ở mức cao. Mỗi khi chân này xuống thấp thì bật đèn LED. Đèn LED được nối đến chân P1.3 và bình thường ở chế độ tắt. Nếu đèn được bật thì cần phải sáng vài phần giây. Khi công tắc được ấn xuống thấp, đèn LED phải sáng liên tục.

Giải:

```
        ORG    0000H
        LJMP   MAIN              ;Bỏ qua bảng vector ngắt
;--ISR cho ngắt cứng INT1 để bật đèn LED
        ORG    0013H             ;Trình ISR cho INT1
        SETB   P1.3              ;Bật đèn LED
        MOV    R3,#255
BACK:   DJNZ   R3,BACK           ;Giữ đèn LED sáng một lúc
        CLR    P1.3              ;Tắt đèn LED
        RETI                     ;Trở về trình chính
```

```
;--Bắt đầu chương trình chính Main.
      ORG    30H
MAIN: MOV    IE,#10000100B    ;Cho phép ngắt dài
      SJMP   HERE             ;Chờ cho đến khi được ngắt
      END
```

Ấn công tắc xuống sẽ làm cho đèn LED sáng. Nếu công tắc duy trì trạng thái ấn thì đèn LED sáng liên tục.

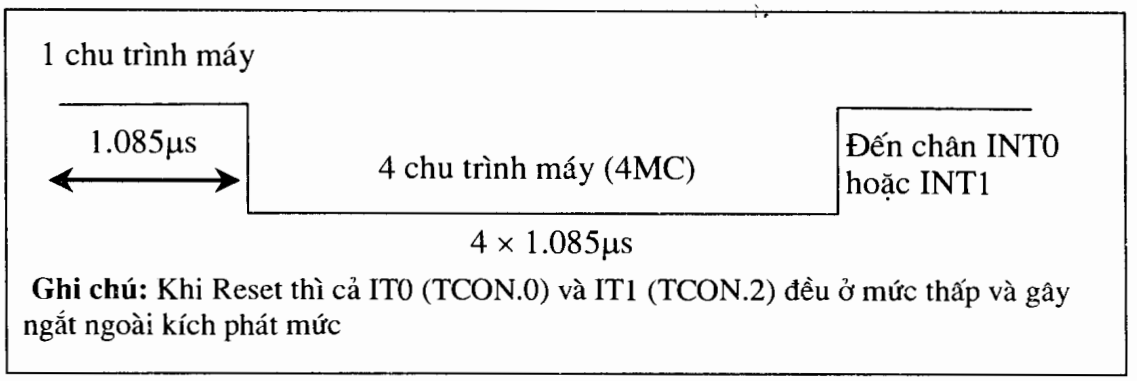


Ở chương trình này, bộ vi điều khiển quay vòng liên tục trong vòng lặp HERE. Mỗi khi công tắc trên chân P3.3 (INT1) được kích hoạt thì bộ vi điều khiển thoát khỏi vòng lặp và nhảy đến bảng vector ngắt tại địa chỉ 0013H. Trình ISR của INT1 sẽ bật đèn LED, duy trì đèn sáng một lúc rồi tắt đền trước khi thực hiện lệnh trở về RETI. Nếu trong khi thực hiện lệnh quay trở về RETI mà chân INT1 vẫn còn ở mức thấp thì bộ vi điều khiển khởi tạo lại ngắt. Do vậy, để giải quyết vấn đề này thì chân INT1 phải được đưa lên cao tại thời điểm lệnh RET1 được thực hiện.

Lấy mẫu ngắt kích phát mức thấp

Chân P3.2 và P3.3 bình thường được dùng cho vào/ra nếu các bit INT0 và INT1 ở thanh ghi IE không được kích hoạt. Sau khi các ngắt phần cứng được kích hoạt thì bộ vi điều khiển thực hiện lấy mẫu chân INTn một lần trong một chu trình máy để lấy tín hiệu mức thấp. Theo tài liệu kỹ thuật của nhà sản xuất 8051 thì "chân ngắt phải được giữ ở mức thấp cho đến khi bắt đầu thực hiện trình phục vụ ngắt ISR. Nếu chân INTn được đưa trở lại mức cao trước khi bắt đầu thực hiện ISR thì sẽ chẳng thực hiện ngắt nào". Tuy nhiên, khi kích hoạt ngắt theo mức thấp thì cần đưa trở lại mức cao trước khi thực hiện lệnh RETI. Cũng lại theo tài liệu kỹ thuật của nhà sản xuất thì "nếu chân INTn vẫn ở mức thấp cả sau lệnh RETI của trình phục vụ ngắt thì một ngắt khác lại sẽ được kích hoạt sau khi lệnh RET1

được thực hiện". Do vậy, để bảo đảm việc kích.hoạt ngắt phần cứng tại các chân INTn phải khẳng định rằng thời gian tồn tại tín hiệu mức thấp là khoảng 4 chu trình máy và không được lâu hơn. Sở dĩ như vậy vì một thực tế là ngắt theo mức không được chốt. Do vậy, chân ngắt phải được duy trì ở mức thấp cho đến khi bắt đầu thực hiện ISR.



Ghi chú: Khi Reset thì cả IT0 (TCON.0) và IT1 (TCON.2) đều ở mức thấp và gây ngắt ngoài kích phát mức

Hình 11.5. Thời gian tối thiểu của ngắt kích phát mức thấp
(XTAL = 11.0592MHz)

Ngắt kích phát sườn

Như đã nói ở trên, khi Reset thì 8051 đặt ngắt INT0 và INT1 ở chế độ kích phát mức thấp. Để đổi các ngắt thành kích phát sườn thì cần phải viết chương trình cho các bit của thanh ghi TCON. Thanh thi TCON có các bit cờ IT0 và IT1 xác định chế độ kích phát sườn hay mức của các ngắt phần cứng. IT0 và IT1 là các bit D0 và D2 tương ứng của thanh ghi TCON. Các bit này có thể viết dưới dạng TCON.0 và TCON.2 vì thanh ghi TCON có thể định địa chỉ bit. Khi Reset, TCON.0 (IT0) và TCON.2 (IT1) đều ở mức thấp (0), nghĩa là các ngắt phần cứng ngoài của các chân INT0 và INT1 là ngắt theo mức thấp. Nếu chuyển các bit TCON.0 và TCON.2 lên cao nhờ lệnh "SETB TCON.0" và "SETB TCON.2" thì các ngắt phần cứng ngoài INT0 và INT1 trở thành các ngắt theo sườn. Ví dụ, lệnh "SETB TCON.2" làm cho INT1 trở thành ngắt kích phát sườn. Khi đó, nếu có một tín hiệu chuyển từ cao xuống thấp cấp cho chân P3.3 thì bộ vi điều khiển sẽ bị ngắt và buộc nhảy đến bảng vector ngắt tại địa chỉ 0013H để thực hiện trình phục vụ ngắt. Tuy nhiên, tất cả quá trình trên đều với giải thiết rằng bit ngắt đã được cho phép trong thanh ghi IE.

D7							D0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1	TCON.7	Cờ tràn của Timer 1, được thiết lập bởi phần cứng khi bộ đếm/bộ định thời 1 tràn, và được xoá bởi phần cứng khi bộ xử lý nhảy đến trình phục vụ ngắt.
TR1	TCON.6	Bit điều khiển hoạt động của Timer 1, được thiết lập và xoá bởi phần mềm để bật/tắt bộ đếm/bộ định thời 1.
TF0	TCON.5	Tương tự như TF1 nhưng là cho Timer 0.
TR0	TCON.4	Tương tự như TR1 nhưng là cho Timer 0.
IE1	TCON.3	Cờ ngắt ngoài 1 kích phát sườn, được CPU thiết lập khi phát hiện có sườn xuống ngắt ngoài và được CPU xóa khi ngắt được xử lý. Lưu ý: Cờ này không chốt ngắt kích phát mức thấp.
IT1	TCON.2	Bit điều khiển kiểu ngắt 1 (Interrrupt 1 Type Control Bit) được thiết lập và xoá bởi phần mềm để xác định kiểu ngắt ngoài kích phát sườn xuống hay mức thấp.
IE0	TCON.1	Tương tự như IE1 nhưng là cho ngắt ngoài 0.
IT0	TCON.0	Tương tự như bit IT1 nhưng là cho ngắt ngoài 0.

Hình 11.6. Thanh ghi TCON (Timer/Counter)

Xem ví dụ 11.6, chú ý rằng sự khác nhau duy nhất giữa chương trình này với ví dụ 11.5 là dòng đầu tiên của MAIN dùng lệnh "SETB TCON.2" chuyển ngắt INT1 về dạng kích phát sườn. Khi sườn xuống của tín hiệu cấp đến chân INT1 thì đèn LED sẽ được bật lên một lúc. Đèn LED có thời gian sáng phụ thuộc vào độ trễ bên trong ISR của INT1. Để bật lại đèn LED thì cần có một sườn xung xuống khác cấp đến chân P3.3. Đây là điểm khác với ví dụ 11.5. Ở ví dụ 11.5, ngắt được kích phát theo mức, nên đèn LED còn sáng chừng nào tín hiệu ở chân INT1 vẫn còn ở mức thấp. Song, ở ví dụ 11.6, để bật lại đèn LED thì xung ở chân INT1 phải được đưa lên cao rồi sau đó hạ xuống thấp để tạo ra một sườn xuống để kích hoạt ngắt.

Ví dụ 11.6

Giả thiết chân P3.3 (INT1) được nối với một máy tạo xung. Hãy viết chương trình trong đó sườn xuống của xung sẽ chuyển chân P1.3 lên cao, chân này được nối tới đèn LED (hoặc một còi báo). Như vậy, đèn LED được bật và tắt cùng tần số với xung cấp tới chân INT1. Đây là phiên bản kích phát sườn của ví dụ 11.5 đã nêu ở trên.

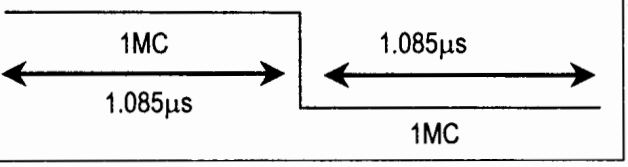
Giải:

```
        ORG    0000H
        LJMP   MAIN
;--Trình ISR dành cho ngắt INT1 để bật đèn LED
        ORG    0013H          ;Nhảy đến ISR phục vụ ngắt INT1
        SETB   P1.3           ;Bật đèn LED(hoặc còi)
        MOV    R3,#225
BACK:   DJNZ   R3,HERE        ;Giữ đèn LED(hoặc còi)một lúc
        CLR    P1.3           ;Tắt đèn LED(hoặc còi)
        RETI                  ;Quay trở về từ ngắt
;--Bắt đầu chương trình chính
        ORG    30H
        SETB   TCON.2         ;Chuyển về kiểu kích phát sườn
        MOV    IE,#10001B     ;Cho phép ngắt ngoài INT1
HERE:   SJMP   HERE           ;Chờ cho đến khi bị ngắt
        END
```

Lấy mẫu ngắt kích phát sườn

Trước khi kết thúc phần này, chúng ta cần trả lời câu hỏi ngắt kích phát sườn được trích mẫu như thế nào? Đối với trường hợp kích phát sườn, nguồn ngắt ngoài phải giữ ở mức cao tối thiểu là một chu trình máy, và sau đó duy trì mức thấp cũng tối thiểu một chu kỳ máy để đảm bảo bộ vi điều khiển nhận biết được quá trình chuyển sườn xung từ cao xuống thấp.

Thời gian xung tối thiểu để phát hiện ra ngắt kích phát sườn với tần số XTAL = 11.0592MHz.



Sườn xuống của xung được 8051 chốt lại và được lưu ở thanh ghi TCON. Các bit TCON.1 và TCON.3 giữ các sườn được chốt của chân INT0 và INT1 tương ứng. TCON.1 và TCON.3 cũng còn được gọi là các bit IE0 và IE1 như chỉ ra trên hình 11.6. Các bit này hoạt động như các cờ "ngắt đang được phục vụ" (Interrupt-in-server). Khi một cờ "ngắt đang được phục vụ" bật lên thì nó báo cho thiết bị bên ngoài biết rằng ngắt hiện nay đang được xử lý và trên chân INTn sẽ không có ngắt nào được đáp ứng chừng nào ngắt này chưa được phục vụ xong. Có thể hiểu tín hiệu này giống như tín hiệu báo bận của máy điện thoại. Cần phải nhấn mạnh đến hai điểm dưới đây khi làm việc với các bit IT0 và IT1 của thanh ghi TCON.

1. Khi các trình phục vụ ngắt ISR kết thúc, nghĩa là khi thực hiện lệnh RETI, các bit TCON.1 và TCON.3 được xoá để báo rằng phục vụ ngắt đã được hoàn tất và 8051 sẵn sàng đáp ứng ngắt khác trên chân đó. Để có thể nhận được ngắt khác thì tín hiệu trên chân đó phải trở lại mức cao và sau đó xuống thấp để tạo nên một ngắt kích phát sườn.
2. Trong thời gian trình phục vụ ngắt đang được thực hiện thì trạng thái chân chân INTn bị bỏ qua, kể cả có bao nhiêu sườn cao xuống thấp tác động cũng không ảnh hưởng. Trong thực tế, chính là một trong các chức năng của lệnh RETI được dùng để xoá bit tương ứng (TCON.1 và TCON.3) trong thanh ghi TCON và như vậy báo rằng, trình phục vụ ngắt sắp kết thúc. Vì lý do đó mà các bit TCON.1 và TCON.3 được gọi là cờ báo "ngắt đang được phục vụ", cờ này sẽ lên cao khi phát hiện có một sườn xung xuống thấp trên chân INT và duy trì mức cao trong toàn bộ quá trình thực hiện ISR. Trạng thái cao này sẽ bị xoá bởi lệnh RETI là lệnh cuối cùng của ISR. Do vậy, sẽ không bao giờ cần đến lệnh xoá bit, như lệnh "CLR TCON.1" hay "CLR TCON.3" trước lệnh RETI trong trình phục vụ các ngắt cứng INT0 và INT1. Đây là điểm khác với trường hợp của ngắt nối tiếp.

Ví dụ 11.7

Sự khác nhau giữa lệnh RET và RETI là gì? Giải thích tại sao ta không thể dùng lệnh RET thay cho lệnh RETI trong trình phục vụ ngắt.

Giải:

Hai lệnh RET và RETI đều giống nhau về nội dung thực hiện là đều lấy hai byte trên đỉnh ngăn xếp nạp vào bộ đếm chương trình và đưa 8051 trở về thực hiện lệnh ở địa chỉ mới đó. Tuy nhiên, lệnh RETI còn thực hiện một nhiệm vụ

khác nữa là xoá cờ "ngắt đang được phục vụ" để báo rằng ngắt đã kết thúc và 8051 có thể nhập một ngắt mới ở chân này. Nếu ta dùng lệnh RET thay cho RETI để làm lệnh cuối cùng của trình phục vụ ngắt, thì chúng ta đã vô tình khoá mọi ngắt mới trên chân này sau ngắt đầu tiên vì trạng thái của chân vẫn ở mức cao nên báo rằng ngắt vẫn đang được phục vụ. Do vậy cần thiết phải dùng lện RETI để xoá các cờ TF0, TF1, TCON.1 và TCON.3.

Thanh ghi TCON

Bây giờ ta xét kỹ hơn về các bit của thanh ghi TCON để hiểu vai trò của thanh ghi này trong việc duy trì các ngắt.

IT0 và IT1

Các bit IT0 và IT1 còn được gọi là TCON.0 và TCON.2 tương ứng. Đây là các bit xác định kiểu ngắt kích phát sườn hay mức của ngắt phần cứng trên chân INT.0 và INT.1. Khi Reset, cả hai bit này đều có mức 0, nghĩa là chúng được thiết lập cho ngắt kích phát mức thấp. Lập trình viên có thể lập các bit này lên cao để chuyển ngắt cứng bên ngoài thành ngắt kích phát sườn. Ở hệ thống xây dựng trên 8051 thì mỗi khi các bit này đã được đặt về 0 hoặc 1 thì trạng thái này sẽ không thay đổi vì đó là chủ ý của nhà thiết kế.

IE0 và IE1

Các bit IE0 và IE1 còn được gọi là TCON.1 và TCON.3 tương ứng. Các bit này được 8051 dùng để xác định kiểu ngắt kích phát sườn xung. Nói các khác, nếu IT0 và IT1 bằng 0 thì có nghĩa ngắt phần cứng là kích phát mức thấp, còn các bit IE0 và IE1 không được sử dụng. Các bit IE0 và IE1 chỉ được 8051 dùng để chốt sườn xung từ cao xuống thấp trên các chân INT0 và INT1. Khi có chuyển trạng thái sườn xung trên chân INT0 (hay INT1) thì 8051 chuyển lên mức cao các bit IEx ở thanh ghi TCON rồi nhảy đến bảng vector ngắt và bắt đầu thực hiện trình phục vụ ngắt ISR. Trong khi 8051 thực hiện ISR thì không có một sườn xung nào được tiếp nhận trên chân INT0 (hay INT1) để ngăn ngừa ngắt mới xuất hiện trong khi đang phục vụ một ngắt. Chỉ sau khi thực hiện lệnh RETI ở cuối trình phục vụ ngắt ISR thì các bit IEx mới bị xoá và báo rằng một sườn xung cao xuống thấp mới trên chân INT0 (hay INT1) sẽ kích hoạt ngắt trở lại. Từ phần trình bày trên có thể thấy rằng, các bit IE0 và IE1 được 8051 sử dụng bên trong để thông báo có một ngắt đang được xử lý hay không. Như vậy, lập trình viên không cần quan tâm đến các bit này.

TR0 và TR1

Đây là các bit D4 và D6 (hay TCON.4 và TCON.6) của thanh ghi TCON. Các bit này đã được giới thiệu ở chương 9. Chúng được dùng để khởi động và dừng các bộ định thời Timer 0 và Timer 1 tương ứng. Vì thanh ghi TCON có thể định địa chỉ bit, nên có thể sử dụng các lệnh "SETB TRx" và "CLR TRx" cũng như các lệnh "SETB TCON.4" và "CLR TCON.4".

TF0 và TF1

TF0 và TF1 là bit D5 (TCON.5) và D7 (TCON.7) của thanh ghi TCON và đã được đề cập ở chương 9. Các bit này được Timer 0 và Timer 1 sử dụng để báo rằng bộ định thời bị tràn hay quay về không. Đối với các bit TF0, TF1, Bạn có thể sử dụng lệnh như "JNB TFx, đích" và "CLR TFx", tuy nhiên, vì TCON là thanh ghi có thể định địa chỉ bit nên Bạn cũng có thể viết dưới dạng "SETB TCON.5, đích" và "CLR TCON.5".

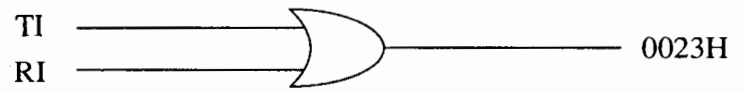
11.4 LẬP TRÌNH NGẮT TRUYỀN THÔNG NỐI TIẾP

Ở chương 10, chúng ta đã nghiên cứu về truyền thông nối tiếp của 8051 sử dụng phương pháp thăm dò (polling). Ở chương này, chúng ta tìm hiểu truyền thông dựa trên ngắt.

Cờ RI và TI và ngắt

Như đã nêu ở chương 10, cờ ngắt phát TI (Transfer interrupt) được bật lên 1 khi bit cuối cùng của khung dữ liệu, bit Stop được phát đi báo rằng thanh ghi SBUF sẵn sàng phát byte kế tiếp. Trái lại, cờ ngắt thu RI (Receive Interrupt) được bật lên 1 khi toàn bộ khung dữ liệu kể cả bit Stop đã được nhận. Nói cách khác, khi thanh ghi SBUF đang lưu một byte, thì cờ RI bật lên báo rằng byte dữ liệu thu được cần cất đi vào nơi an toàn trước khi bị mất (bị ghi đè) bởi dữ liệu tiếp theo sẽ được nhận. Cần nhấn mạnh rằng, đối với truyền thông nối tiếp thì tất cả các bước thực hiện trên đây đều được áp dụng cho cả phương pháp thăm dò lẫn phương pháp ngắt. Sự khác nhau duy nhất giữa hai phương pháp này là ở cách phục vụ quá trình truyền thông nối tiếp. Đối với phương pháp thăm dò, chúng ta phải đợi cho cờ (TI hay RI) bật lên và trong lúc chờ đợi thì bộ vi điều khiển không thể làm được việc gì khác. Còn đối với phương pháp ngắt, thì mỗi khi 8051 vừa nhận được một byte hoặc đã sẵn sàng để gửi byte tiếp theo thì đều được thông báo, do vậy chúng ta có thể làm được các công việc khác trong thời gian chờ truyền thông nối tiếp phục vụ.

Ở 8051 chỉ có một ngắt dành riêng cho truyền thông nối tiếp. Ngắt này được dùng cho cả phát và thu dữ liệu. Nếu bit ngắt trong thanh ghi IE (là bit IE.4) được phép khi RI và TI bật lên, thì 8051 nhận được ngắt và nhảy đến địa chỉ trình phục vụ ngắt dành cho truyền thông nối tiếp 0023H trong bảng vector ngắt và thực hiện nó. Lúc đó, chúng ta cần kiểm tra cờ TI và RI để xem cờ nào gây ngắt để có đáp ứng phù hợp. Xem ví dụ 11.8.



Ngắt truyền thông nối tiếp được kích hoạt bởi các cờ TI và RI

Hình 11.7. Thu và phát dùng một ngắt

Sử dụng cổng COM ở 8051

Ở phần lớn các ứng dụng, ngắt nối tiếp chủ yếu được sử dụng để nhận dữ liệu và không dùng để phát dữ liệu. Vấn đề này cũng tương tự như chuông báo nhận điện thoại. Để báo có cuộc gọi đến, điện thoại đổ chuông, còn để gọi điện thoại thì có nhiều cách khác chứ không cần đến đổ chuông. Tuy nhiên, khi điện thoại đã đổ chuông thì dù Bạn đang bận hay rỗi đều cần phải nhấc máy ngay nếu không sẽ mất cuộc gọi. Tương tự như vậy, ngắt nối tiếp được sử dụng để nhận dữ liệu. Chúng ta xem xét ví dụ 11.8 dưới đây.

Ví dụ 11.8

Hãy viết chương trình, trong đó 8051 đọc dữ liệu từ cổng P1 và ghi liên tục tới cổng P2 đồng thời đưa một bản sao dữ liệu tới cổng COM nối tiếp để thực hiện truyền nối tiếp. Giả thiết tần số XTAL là 11.0592MHz và tốc độ truyền là 9600 baud.

Giải:

```
        ORG   0
        LJMP  MAIN
        ORG   23H
        LJMP  SERIAL        ;Nhảy đến ISR nối tiếp
MAIN:   MOV   P1,#0FFH      ;Lấy cổng P1 làm cổng đầu vào
        MOV   TMOD,#20h     ;Timer 1,chế độ 2 tự nạp lại
```

```
        MOV   TH1,#0FDH     ;Chọn tốc độ baud=9600
        MOV   SCON,#50H     ;Khung 8 bit,1 stop,cho phép REN
        MOV   IE,#10010000B ;Cho phép ngắt nối tiếp
        SETB  TR1           ;Khởi động Timer 1
BACK:   MOV   A,P1          ;Đọc dữ liệu từ cổng P1
        MOV   SBUF,A        ;Lấy một bản sao tới SBUF
        MOV   P2,A          ;Gửi nó đến cổng P2
        SJMP  BACK          ;Ở lại trong vòng lặp
 ;--Trình phục vụ ngắt cổng nối tiếp
        ORG   100H
SERIAL:JB TI,TRANS          ;Nhảy nếu cờ TI cao
        MOV   A,SBUF        ;Nếu không thì nhận dữ liệu
        CLR   RI            ;Xoá cờ RI
        RETI                ;Trở về chương trình chính
TRANS:  CLR   TI            ;Xoá cờ TI
        RETI                ;Trở về chương trình chính
        END
```

Trong chương trình trên, cần lưu ý đến cờ TI và RI. Tại thời điểm một byte được ghi vào SBUF thì byte này được định khung và truyền đi nối tiếp. Kết quả là khi bit cuối cùng (bit Stop) được truyền đi thì cờ TI bật lên cao và gây ra ngắt nối tiếp nếu bit tương ứng trong thanh ghi IE được đưa lên cao. Trong trình phục vụ ngắt nối tiếp, cần kiểm tra cả cờ TI và cờ RI vì cả hai đều có thể gọi ngắt mà chỉ có một ngắt chung cho cả phát và thu dữ liệu.

Ví dụ 11.9

Hãy viết chương trình trong đó 8051 nhận dữ liệu ở cổng P1 và gửi liên tục đến cổng P2, trong khi đó dữ liệu nhận được ở cổng nối tiếp COM thì gửi đến cổng P0. Giả thiết tần số XTAL là 11.0592MHz và tốc độ baud 9600.

Giải:

```
        ORG   0
        LJMP  MAIN
        ORG   23H
        LJMP  SERIAL        ;Lấy cổng P1 là cổng đầu vào
        ORG   03H
```

```
MAIN:   MOV   P1,# FFH
        MOV   TMOD,#20H      ;Chọn Timer chế độ 2
        MOV   TH1,#0FDH      ;Khung 8 bit,1stop,cho phép REN
        MOV   SCON,# 50H     ;Cho phép ngắt nối tiếp
        MOV   IE,#10010000B ;Khởi động Timer 1
        SETB  TR1            ;Đọc dữ liệu từ cổng P1
BACK:   MOV   A,P1           ;Gửi dữ liệu đến cổng P2
        MOV   P2,A           ;Ở lại trong vòng lặp
        SJMP  BACK
;-- Trình phục vụ ngắt cổng nối tiếp.
        ORG   100H
SERIAL:JB     TI,TRANS       ;Nhảy nếu TI cao
        MOV   A,SBUF         ;Nếu không thì nhận dữ liệu
        MOV   P0,A           ;Gửi dữ liệu đến cổng P0
        CLR   RI             ;Xoá cờ RI vì CPU không xoá
        RETI                 ;Trở về trình chính vì CPU
TRANS:  CLS   TI             ;Xoá cờ TI
        RETI                 ;Trở về trình chính
        END
```

Xoá cờ RI và TI trước khi thực hiện lệnh RETI

Có thể thấy ở ví dụ 11.9, trước lệnh trở về chương trình RETI của ISR là các lệnh xoá các cờ RI và TI. Việc xoá cờ là không thể thiếu, bởi vì ngắt dùng để 8051 thu và phát không biết được nguồn gây ra ngắt. Do vậy, trình phục vụ ngắt phải xoá các cờ này để cho phép ngắt tiếp theo được đáp ứng sau khi kết thúc ngắt. Đây là điểm khác với ngắt ngoài và ngắt bộ định thời, ở đó 8051 đều thực hiện xoá các cờ. Xoá cờ ngắt bằng phần mềm thực hiện nhờ lệnh "CLR TI" và "CLR RI". Xem ví dụ 11.10 và để ý đến lệnh xoá cờ ngắt trước lệnh RETI.

Ví dụ 11.10

Hãy viết chương trình sử dụng các ngắt để thực hiện các công việc sau:
a) Thu dữ liệu nối tiếp và gửi đến cổng P0.
b) Từ cổng P1 đọc và truyền nối tiếp và sao vào cổng P2.
c) Sử dụng Timer 0 tạo sóng vuông tần số 5 kHz trên chân P0.1

Giả thiết tần số XTAL = 11.0592MHz và tốc độ baud 4800.

Giải:

```
        ORG    0
        LJMP   MAIN
        ORG    000BH          ;ISR cho Timer 0
        CPL    P0.1           ;Tạo xung ở chân P0.1
        RETI                  ;Trở về từ ISR
        ORG    23H
        LJMP   SERIAL         ;Nhảy đến ISR ngatư nối tiếp
        ORG    30H
MAIN :  MOV    P1,#0FFH       ;Đặt P1 làm cổng vào
        MOV    TMOD,#22H      ;Timer 0&1 mode 2 tự nạp lại
        MOV    TH1,#0F6H      ;Chọn tốc độ baud 4800
        MOV    SCON,#50H      ;Khung 8 bit,1 stop,cho phép REN
        MOV    TH0,#-92       ;Tạo tần số 5kHz
        MOV    IE,#10010010B  ;Cho phép ngắt nối tiếp
        SETB   TR1            ;Khởi động Timer 1
        SETB   TR0            ;Khởi động Timer 0
BACK:   MOV    A,P1           ;Đọc dữ liệu từ cổng P1
        MOV    SBUF,A         ;Lấy bản sao dữ liệu
        MOV    P2,A           ;Ghi vào cổng P2
        SJMP   BACK           ;Ở lại trong vòng lặp

;--Trình phục vụ ngắt cổng nối tiếp.
        ORG    100H
SERIAL:JB      TI,TRANS       ;Nhảy nếu TI cao
        MOV    A,SBUF         ;Nếu không thì nhận dữ liệu
        MOV    P0,A           ;Gửi dữ liệu nối tiếp đến P0
        CLR    RI             ;Xoá cờ RI vì 8051 không xoá
        RETI                  ;Trở về từ ISR
TRANS: CLRTI                  ;Xoá cờ TI vì 8051 không xoá
        RETI                  ;Trở về từ ISR
        END
```

Trước khi kết thúc phần này, hãy lưu ý đến danh sách tất cả các cờ ngắt được liệt kê ở bảng 11.2. Thanh ghi TCON có 4 cờ ngắt, còn thanh ghi SCON có 2 cờ TI và RI.

Bảng 11.2. Danh sách cờ ngắt của 8051

Ngắt	Cờ	Bit thanh ghi SFR
Ngắt ngoài 0	IE0	TCON.1
Ngắt ngoài 1	IE1	TCON.3
Ngắt Timer 0	TF0	TCON.5
Ngắt Timer 1	TF1	TCON.7
Ngắt cổng nối tiếp	T1	SCON.1
Ngắt Timer 2	TF2	T2CON.7 (TA89C52)
Ngắt Timer 2	EXF2	T2CON.6 (TA89C52)

11.5 MỨC ƯU TIÊN NGẮT CỦA 8051

Mức ưu tiên ngắt khi Reset

Khi 8051 được cấp nguồn thì các mức ưu tiên ngắt được gán theo bảng 11.3. Từ bảng này chúng ta thấy, ví dụ, nếu ngắt phần cứng ngoài 0 và 1 được kích hoạt cùng một lúc thì ngắt ngoài 0 sẽ được đáp ứng trước. Chỉ sau khi ngắt INT0 đã được phục vụ xong thì INT1 mới được phục vụ, vì INT1 có mức ưu tiên thấp hơn. Tuy nhiên, trong thực tế, sơ đồ mức ưu tiên ngắt nêu ở bảng 11.3 chưa phải là yếu tố quyết định. Cái chính là 8051 thực hiện trình tự thăm dò bên trong các ngắt theo bảng 11.3 và đáp ứng một cách phù hợp.

Bảng 11.3. Mức ưu tiên ngắt khi Reset

Mức ưu tiên từ cao xuống thấp	
Ngắt ngoài 0	INT0
Ngắt bộ định thời 0	TF0
Ngắt ngoài 1	INT1
Ngắt bộ định thời 1	TF1
Ngắt truyền thông nối tiếp	(RI + TI)

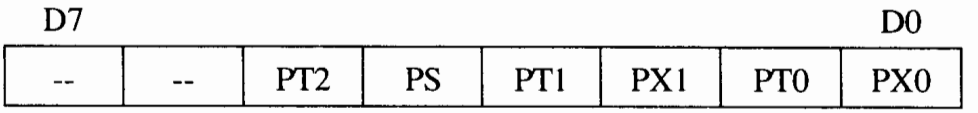
Ví dụ 11.11

Hãy xác định xem nếu các ngắt INT0, TF0 và INT1 được kích hoạt cùng một lúc thì sẽ xảy ra điều gì? Giả thiết rằng các mức ưu tiên được thiết lập như khi Reset và các ngắt ngoài là ngắt kích phát sườn.

Giải:

Nếu ba ngắt này được kích hoạt cùng một thời điểm thì chúng được chốt và được giữ ở bên trong. Sau đó 8051 kiểm tra tất cả năm ngắt theo trình tự cho

trong bảng 11.3. Nếu một ngắt bất kỳ được kích hoạt thì nó được phục vụ tuần tự. Do vậy, khi cả ba ngắt trên đây cùng được kích hoạt một lúc thì ngắt ngoài 0 (IE0) được phục vụ đầu tiên, sau đó đến ngắt Timer 0 (TF0), và cuối cùng là ngắt ngoài 1 (IE1).



Bit ưu tiên = 1 là mức ưu tiên cao, bit ưu tiên = 0 là mức ưu tiên thấp.

-	IP.7	Dự trữ
-	IP.6	Dự trữ
PT2	IP.5	Bit ưu tiên ngắt Timer 2 (chỉ dùng cho 8052)
PS	IP.4	Bit ưu tiên ngắt cổng nối tiếp
PT1	IP.3	Bit ưu tiên ngắt Timer 1
PX1	IP.2	Bit ưu tiên ngắt ngoài 1
PT0	IP.1	Bit ưu tiên ngắt Timer 0
PX0	IP.0	Bit ưu tiên ngắt ngoài 0

Người dùng không được viết phần mềm ghi 1 vào các bit chưa dùng vì chúng dành cho các ứng dụng tương lai.

Hình 11.8. Thanh ghi mức ưu tiên ngắt IP

Thiết lập mức ưu tiên ngắt dùng thanh ghi IP

Chúng ta có thể thay đổi trình tự ưu tiên cho ở bảng 11.3 bằng cách gán mức ưu tiên cao hơn cho bất kỳ ngắt nào. Điều này được thực hiện bằng cách lập trình một thanh ghi gọi là thanh ghi mức ưu tiên ngắt IP (Interrupt Priority). Trên hình 11.8 là các bit của thanh ghi này. Khi Reset, thanh ghi IP chứa hoàn toàn các số 0 để tạo ra trình tự ưu tiên ngắt theo bảng 11.3. Để một ngắt nào đó có mức ưu tiên cao hơn chúng ta thực hiện đưa bit tương ứng lên cao. Xem ví dụ 11.12.

Một điểm nữa cần được lưu ý là trường hợp khi có nhiều bit ngắt trong thanh ghi IP cùng được đặt lên cao. Khi đó, nếu các ngắt này cùng có mức ưu tiên cao hơn các ngắt khác thì chúng sẽ được phục vụ theo trình tự cho trong bảng 11.3. Hãy xem ví dụ 11.13.

Ví dụ 11.12

a) Hãy lập trình thanh ghi IP để gán mức ưu tiên cao nhất cho ngắt INT1 (ngắt ngoài 1).

b) Hãy phân tích điều gì xảy ra khi INT0, INT1 và TF0 được kích hoạt cùng một lúc. Giả thiết tất cả các ngắt dạng kích phát sườn.

Giải:

a)
```
MOV IP,#00000100B    ;Đặt IP.2=1 để INT1 có mức
                     ;ưu tiên cao hơn
```

Lệnh "SETB IP.2" có tác động tương tự vì IP là thanh ghi định địa chỉ bit.

b) Lệnh ở bước a) gán INT1 có mức ưu tiên cao hơn so với các ngắt khác, do vậy khi INT0, INT1 và TF0 cùng được kích hoạt thì INT1 được phục vụ đầu tiên, sau đó đến INT0 và cuối cùng là TF0. Sở dĩ INT0 thực hiện trước TF0 là vì, ở bước a) đã gán các bit này về 0, do vậy dựa vào bảng 11.3 INT0 có mức ưu tiên cao hơn TF0.

Ví dụ 11.13

Giả thiết rằng, sau khi Reset thì mức ưu tiên ngắt được thiết lập bởi lệnh "MOV IP, #0000 1100B". Hãy xác định trình tự phục vụ các ngắt?

Giải:

Lệnh "MOV IP, #0000 1100B" (chữ B để chỉ hệ nhị phân) thiết lập ngắt ngoài (INT1) và ngắt bộ Timer 1 (TF1) có mức ưu tiên cao hơn các ngắt khác. Tuy nhiên, vì các ngắt được thăm dò theo thứ tự nêu ở bảng 11.3, nên chúng sẽ được phục vụ theo trình tự sau:

Mức ưu tiên cao nhất:	Ngắt ngoài 1 (INT1)
	Ngắt Timer 1 (TF1)
	Ngắt ngoài 0 (INT0)
	Ngắt Timer 0 (TF0)
Mức ưu tiên thấp nhất:	Ngắt cổng truyền thông nối tiếp (RI + RT).

Ngắt một ngắt

Điều gì xảy ra nếu 8051 đang thực hiện chương trình phục vụ một ngắt nào đó thì lại có một ngắt khác được kích hoạt? Trong những trường hợp như vậy thì một ngắt có mức ưu tiên cao hơn có thể ngắt một ngắt có mức ưu tiên thấp hơn.

Đây gọi là *ngắt một ngắt* hay còn một thuật ngữ khác là *ngắt trong ngắt*. Ở 8051, một ngắt ưu tiên thấp có thể bị ngắt bởi một ngắt có mức ưu tiên cao hơn chứ không bị ngắt bởi ngắt có mức ưu tiên thấp hơn. Tất cả mọi ngắt đều được chốt và lưu bên trong, tuy nhiên không có ngắt mức thấp nào được bộ vi điều khiển quan tâm ngay nếu nó chưa kết thúc phục vụ các ngắt mức cao.

Kích hoạt ngắt bằng phần mềm

Có nhiều trường hợp cần kiểm tra một trình phục vụ ngắt bằng mô phỏng. Có thể thực hiện điều này bằng các lệnh đơn giản để thiết lập các ngắt lên cao và như vậy buộc 8051 nhảy đến bảng vector ngắt. Ví dụ, nếu bit IE dành cho bộ Timer 1 được bật lên 1, thì lệnh dạng "SETB TF1" sẽ buộc 8051 ngừng thực hiện mọi công việc hiện tại và nhảy đến bảng vector ngắt. Như vậy, không cần đợi cho Timer 1 quay trở về 0 mới tạo ra ngắt. Chúng ta có thể gây ra một ngắt bằng lệnh đưa bit của ngắt tương ứng lên cao.

Như vậy, chương này đã làm rõ, ngắt là một sự kiện bên trong hoặc bên ngoài gây ra ngắt bộ vi điều khiển để báo cho bộ vi điều khiển biết rằng thiết bị cần được phục vụ. Mỗi một ngắt có một chương trình tương ứng được gọi là trình phục vụ ngắt ISR. Bộ vi điều khiển 8051 có 6 ngắt, trong đó 5 ngắt người dùng có thể truy cập được. Đó là 2 ngắt cho các thiết bị phần cứng bên ngoài INT0 và INT1, 2 ngắt cho các bộ định thời là TF0 và TF1 và 1 ngắt dành cho truyền thông nối tiếp.

Có thể lập trình cho phép hoặc cấm một ngắt bất kỳ cũng như thiết lập mức ưu tiên cho các ngắt của 8051.

Chương 12
NỐI GHÉP VỚI THẾ GIỚI THỰC I - LCD, ADC VÀ CẢM BIẾN

12.1 NỐI GHÉP LCD VỚI 8051

Hoạt động của LCD

Trong những năm gần đây, màn hình tinh thể lỏng LCD (Liquid Crystal Display) ngày càng được sử dụng rộng rãi và đang dần thay thế các đèn LED (7 đoạn và nhiều đoạn). Đó là vì các nguyên nhân sau:

- Màn hình LCD có giá thành hạ.
- Khả năng hiển thị số, ký tự và đồ hoạ tốt hơn nhiều so với đèn LED (đèn LED chỉ hiển thị được số và một số ký tự).
- Sử dụng thêm một bộ điều khiển làm tươi LCD và như vậy giải phóng CPU khỏi công việc này. Còn đối với đèn LED luôn cần CPU (hoặc bằng cách nào đó) để duy trì việc hiển thị dữ liệu.
- Dễ dàng lập trình các ký tự và đồ hoạ.

Mô tả chân của LCD

LCD giới thiệu ở đây có 14 chân. Chức năng của các chân được cho trong bảng 12.1. Bố trí chân được mô tả ở hình 12.1 cho các loại LCD khác nhau.

V_{CC}, V_{SS} và V_{EE}

V_{CC} và V_{SS} là chân nguồn +5V và chân đất, còn V_{EE} được dùng để điều khiển độ tương phản của LCD.

Bảng 12.1. Mô tả chân của LCD

Chân	Ký hiệu	I/O	Mô tả
1	VSS	-	Đất
2	VCC	-	Dương nguồn +5V
3	VEE	-	Nguồn điều khiển tương phản
4	RS	I	RS = 0 chọn thanh ghi lệnh. RS = 1 chọn thanh ghi dữ liệu
5	R/W	I	R/W = 1 đọc dữ liệu. R/W = 0 ghi
6	E	I/O	Cho phép
7	DB0	I/O	Bus dữ liệu 8 bit
8	DB1	I/O	Bus dữ liệu 8 bit
9	DB2	I/O	Bus dữ liệu 8 bit
10	DB3	I/O	Bus dữ liệu 8 bit
11	DB4	I/O	Bus dữ liệu 8 bit
12	DB5	I/O	Bus dữ liệu 8 bit
13	DB6	I/O	Bus dữ liệu 8 bit
14	DB7	I/O	Bus dữ liệu 8 bit

RS (Register Select) - chọn thanh ghi

Có hai thanh ghi rất quan trọng bên trong LCD. Chân RS được dùng để chọn các thanh ghi này. Nếu RS = 0 thì thanh ghi mã lệnh được chọn, cho phép người dùng gửi một lệnh chẳng hạn như xoá màn hình, đưa con trỏ về đầu dòng v.v. Nếu RS = 1 thì thanh ghi dữ liệu được chọn và cho phép người dùng gửi dữ liệu cần hiển thị lên LCD.

R/W (Read/Write) - Chân đọc/ghi

Chân vào đọc/ghi cho phép người dùng đọc/ghi thông tin từ/lên LCD. R/W = 0 thì đọc, còn R/W = 1 thì ghi.

E (Enable) - Chân cho phép

Chân cho phép E được LCD sử dụng để chốt thông tin hiện có trên chân dữ liệu. Khi dữ liệu được cấp đến chân dữ liệu thì một xung mức Cao-xuống-Thấp được áp đến chân E để LCD chốt dữ liệu trên chân dữ liệu. Xung này phải rộng tối thiểu là 450ns.

D0 - D7

Đây là 8 chân dữ liệu 8 bit, được dùng để gửi thông tin lên LCD hoặc đọc nội dung của các thanh ghi trong LCD.

Để hiển thị chữ cái và con số, mã ASCII của các chữ cái từ A đến Z, a đến z và các con số từ 0 - 9 được gửi đến các chân này khi bật RS = 1.

Cũng có các mã lệnh được gửi đến LCD để xoá màn hình hoặc đưa con trỏ về đầu dòng hoặc nhấp nháy con trỏ. Bảng 12.2 liệt kê các mã lệnh này.

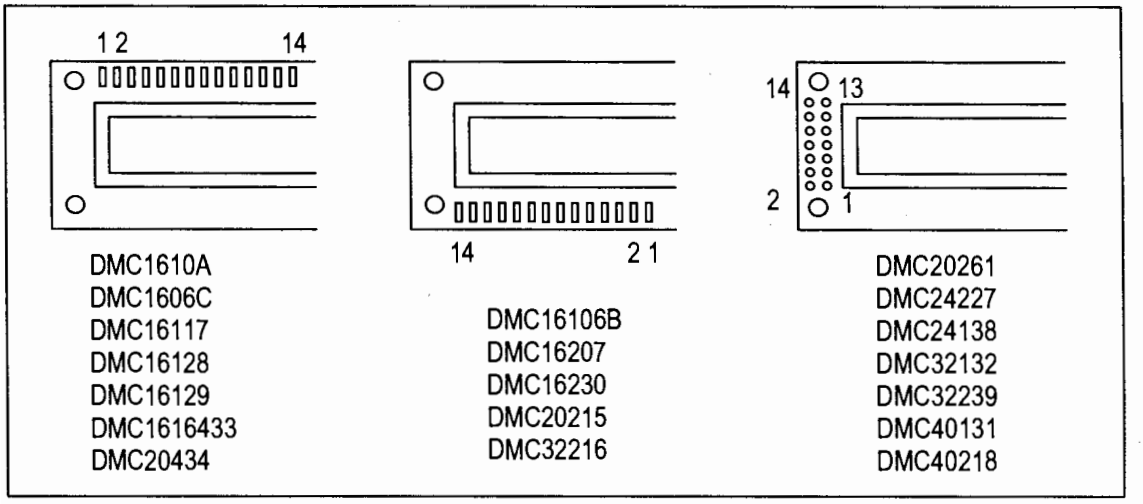
Cũng có thể sử dụng RS = 0 để kiểm tra bit cờ bận xem LCD đã sẵn sàng nhận thông tin chưa. Khi

Bảng 12.2. Mã lệnh LCD

Mã (Hexa)	Lệnh đến thanh ghi của LCD
1	Xoá màn hình hiển thị
2	Trở về đầu dòng
4	Dịch con trỏ sang trái
6	Dịch con trỏ sang phải
5	Dịch hiển thị sang phải
7	Dịch hiển thị sang trái
8	Tắt con trỏ, tắt hiển thị
A	Tắt hiển thị, bật con trỏ
C	Bật hiển thị, tắt con trỏ
E	Bật hiển thị, nhấp nháy con trỏ
F	Tắt con trỏ, nhấp nháy con trỏ
10	Dịch vị trí con trỏ sang trái
14	Dịch vị trí con trỏ sang phải
18	Dịch toàn bộ hiển thị sang trái
1C	Dịch toàn bộ hiển thị sang phải
80	Đưa con trỏ về đầu dòng thứ nhất
C0	Đưa con trỏ về đầu dòng thứ hai
38	Hai dòng và ma trận 5 × 7

Ghi chú: Bảng này mở rộng từ bảng 12.4.

R/W = 1 và RS = 0 thì cờ bận D7 thực hiện các chức năng như sau: Nếu D7 = 1 (cờ bận bằng 1) có nghĩa LCD đang bận các công việc bên trong và sẽ không nhận bất kỳ thông tin mới nào, còn nếu D7 = 0 thì LCD sẵn sàng nhận thông tin mới. Trong mọi trường hợp cần kiểm tra cờ bận trước khi ghi bất kỳ dữ liệu nào lên LCD.



Hình 12.1. Bố trí chân của các phiên bản LCD của hãng Optrex

Gửi có trễ lệnh và dữ liệu đến LCD

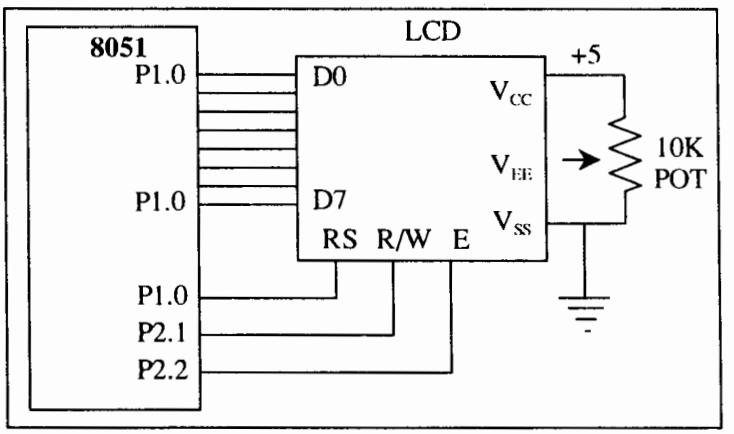
Để gửi một lệnh bất kỳ nêu ở bảng 12.2 đến LCD, cần đưa chân RS = 0, còn để gửi dữ liệu thì bật RS = 1. Sau đó, gửi một sườn xung cao xuống thấp đến chân E để cho phép chốt dữ liệu trong LCD. Chúng ta xem trong đoạn chương trình minh hoạ sau đây, xem hình 12.2.

```
;Gọi thời gian trễ trước khi gửi dữ liệu/lệnh kế tiếp.
;ChânP1.0-P1.7được nối tới chân dữ liệu D0-D7 của LCD.
;Chân P2.0 được nối tới chân RS của LCD.
;Chân P2.1 được nối tới chân R/W của LCD.
;Chân P2.2 được nối đến chân E của LCD.
        ORG
        MOV     A,#38H      ;LCD 2 dòng, ma trận 5×7
        ACALL   COMNWRT     ;Gọi chương trình con
        ACALL   DELAY       ;Tạo độ trễ cho LCD
        MOV     A,#0EH      ;Hiển thị màn hình và con trỏ
        ACALL   COMNWRT     ;Gọi chương trình con
```

```
        ACALL   DELAY      ;Tạo độ trễ cho LCD
        MOV     AM,#01     ;Xoá LCD
        ACALL   COMNWRT    ;Gọi chương trình con
        ACALL   DELAY      ;Tạo độ trễ cho LCD
        MOV     A,#06H     ;Dịch con trỏ sang phải
        ACALL   COMNWRT    ;Gọi chương trình con
        ACALL   DELAY      ;Tạo độ trễ cho LCD
        MOV     AM,#48H    ;Đưa con trỏ về dòng 1 cột 4
        ACALL   COMNWRT    ;Gọi chương trình con
        ACALL   DELAY      ;Tạo độ trễ cho LCD
        MOV     A,#"N"     ;Hiển thị chữ N
        ACALL   DATAWRT    ;Gọi trình hiển thị DISPLAY
        ACALL   DELAY      ;Tạo độ trễ cho LCD
        MOV     AM,#"0"    ;Hiển thị chữ O
        ACALL   DATAWRT    ;Gọi trình DISPLAY
AGAIN:  SJMP    AGAIN      ;Chờ
COMNWRT: ACALL  READY      ;Gửi lệnh đến LCD
        MOV     P1,A       ;Sao thanh ghi A đến cổng P1
        CLR     P2.0       ;Đặt RS=0 để gửi lệnh
        CLR     P2.1       ;Đặt R/W=0 để ghi dữ liệu
        SETB    P2.2       ;Đặt E=1 cho xung cao
        CLR     P2.2       ;Đặt E=0 xung cao xuống thấp
        RET
DATAWRT:                   ;Ghi dữ liệu ra LCD
        MOV     P1,A       ;Sao thanh ghi A đến cổng P1
        SETB    P2.0       ;Đặt RS=1 để gửi dữ liệu
        CLR     P2.1       ;Đặt R/W=0 để ghi
        SETB    P2.2       ;Đặt E=1 cho xung cao
        CLR     P2.2       ;Đặt E=0 xung cao xuống thấp
        RET
DELAY:  MOV     R3,#50     ;Đặt trễ 50µs
HERE2:  MOV     R4,#255    ;Đặt R4=255
HERE:   DJNZ    R4,HERE    ;Đợi cho đến khi R4=0
        DJNZ    R3,HERE2
        RET
        END
```

Gửi mã lệnh hoặc dữ liệu đến LCD có kiểm tra cờ bận

Đoạn chương trình trên trình bày cách gửi lệnh đến LCD mà không có kiểm tra cờ bận (Busy Flag). Lưu ý rằng, chúng ta phải đặt một độ trễ lớn trong quá trình xuất dữ liệu hoặc lệnh ra LCD. Tuy nhiên, một cách tốt hơn nhiều là hiển thị cờ bận trước khi xuất một lệnh hoặc dữ liệu tới LCD. Dưới đây là chương trình thực hiện ý tưởng đó.



Hình 12.2. Nối ghép LCD

```
;Kiểm tra cờ bận trước khi gửi dữ liệu và lệnh ra LCD
;P1 làm cổng dữ liệu
;P2.0 nối tới chân RS
;P2.1 nối tới chân R/W
;P2.2 nối tới chân E
          ORG
          MOV     A,#38H        ;LCD 2 dòng, ma trận 5×7
          ACALL   COMMAND       ;Xuất lệnh
          MOV     A,#0EH        ;Dịch con trỏ sang phải
          ACALL   COMMAND       ;Xuất lệnh
          MOV     A,#01H        ;Xoálệnh LCD
          ACALL   COMMAND       ;Xuất lệnh
          MOV     A,#86H        ;Dịch con trỏ sang phải
          ACALL   COMMAND       ;Đưa con trỏ về dòng 1 lệnh 6
          MOV     A,#"N"        ;Hiển thị chữ N
          ACALL   DATA-DISPLAY
          MOV     A,#"O"        ;Hiển thị chữ O
          ACALL   DATA-DISPLAY
HERE:     SJMP    HERE          ;Chờ
COMMAND:  ACALL   READY         ;LCD đã sẵn sàng chưa?
```

```
          MOV     P1,A       ;Xuất mã lệnh
          CLR     P2.0       ;Đặt RS=0 cho xuất lệnh
          CLR     P2.1       ;R/W=0 để ghi dữ liệu tới LCD
          SETB    P2.2       ;E=1 nhận xung cao-xuống-thấp
          CLR     P2.2       ;Đặt E=0 chốt dữ liệu
          RET
DATA-DISPLAY:
          ACALL   READY      ;LCD đã sẵn sàng chưa?
          MOV     P1, A      ;Xuất dữ liệu
          SETB    P2.0       ;Đặt RS=1 cho xuất dữ liệu
          CLR     P2.1       ;R/W=0 để ghi dữ liệu ra LCD
          SETB    P2.2       ;E=1 nhận xung cao xuống thấp
          CLR     P2.2       ;Đặt E=0 chốt dữ liệu
          RET
DELAY:    SETB    P1.7       ;Lấy P1.7 làm cổng vào
          CLR     P2.0       ;RS=0 để truy cập TG lệnh
          SETB    P2.1       ;R/W=1 đọc thanh ghi lệnh
;Đọc thanh ghi lệnh và kiểm tra cờ lệnh
BACK:     CLR     P2.2       ;E=1 nhận xung cao xuống thấp
          SETB    P2.2       ;E=0 chốt dữ liệu vào
          JB      P1.7,BACK ;Đợi cho đến khi cờ bận=0
          RET
          END
```

Lưu ý cờ bận D7 của thanh ghi lệnh ở chương trình. Để đọc thanh ghi lệnh cần đặt RS = 0, R/W = 1 và xung Cao-Xuống-Thấp đến bit E. Sau khi đọc thanh ghi lệnh, nếu bit D7 (cờ bận) ở mức cao thì LCD bận và không có thông tin (lệnh) nào được xuất đến. Chỉ khi D7 = 0 mới có thể gửi dữ liệu hoặc lệnh đến LCD. Lưu ý trong phương pháp này không sử dụng trễ thời gian vì chúng ta đã kiểm tra cờ bận trước khi xuất lệnh hoặc dữ liệu lên LCD.

Bảng dữ liệu của LCD

Ở LCD chúng ta có thể đặt dữ liệu vào bất cứ chỗ nào. Dưới đây là các địa chỉ và cách chúng được truy cập.

RS	E/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
0	0	1	A	A	A	A	A	A	A

Trong đó, AAAAAAA = từ 0000000 đến 0100111 ở dòng lệnh 1 và AAAAAAA = 1000000 đến 1100111 ở dòng lệnh 2. Xem bảng 12.3.

Bảng 12.3. Định địa chỉ cho LCD

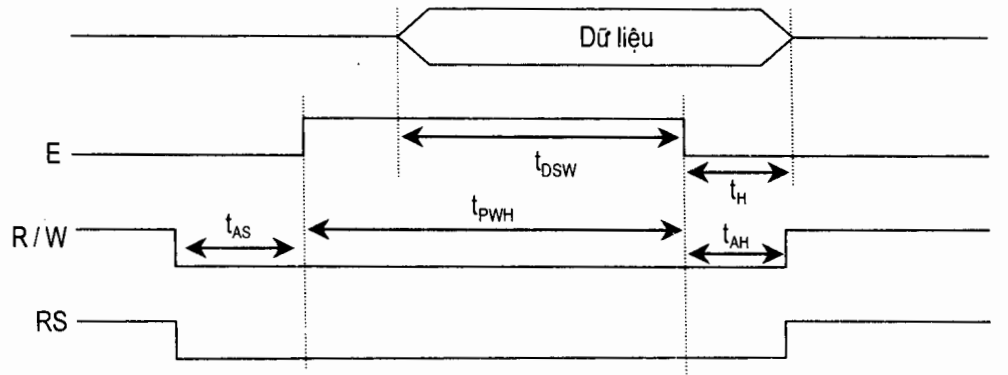
	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Dòng 1 (min)	1	0	0	0	0	0	0	0
Dòng 1 (max)	1	0	1	0	0	1	1	1
Dòng 2 (min)	1	1	0	0	0	0	0	0
Dòng 2 (max)	1	1	1	0	0	1	1	1

Vùng địa chỉ cao có thể lên tới 0100111 đối với LCD 40 ký tự (0100111B=39D), còn đối với LCD 20 ký tự thì chỉ đến 010011 (10011B= 19D). Như vậy, đối với LCD 40 × 2 ta có địa chỉ theo thập phân là từ 0 đến 39.

Từ những trình bày trên đây, chúng ta có thể xác định được địa chỉ của vị trí con trỏ đối với LCD có các kích thước khác nhau. Xem hình 12.3. Chú ý rằng, tất cả địa chỉ đều cho ở dạng Hexa. Hình 12.4 giới thiệu phân bố thời gian của LCD. Bảng 12.4 liệt kê chi tiết các lệnh của LCD.

16 × 2 LCD	80	81	82	83	84	85	86	.....	8F
	C0	C0	C2	C3	C4	C5	C6	.....	CF
20 × 1 LCD	80	81	82	83	.....	93			
20 × 2 LCD	80	81	82	83	.....	93			
	C0	C0	C2	C3	.....	D3			
20 × 4 LCD	80	81	82	83	.....	93			
	C0	C0	C2	C3	.....	D3			
	94	95	96	97	.....	A7			
	D4	D5	D6	D7	.....	E7			
20 × 2 LCD	80	81	82	83	.....	A7			
	C0	C0	C2	C3	.....	E7			
Ghi chú: Tất cả dữ liệu đều ở dạng hexa.									

Hình 12.3. Địa chỉ con trỏ của một số LED



t_{Pwh} = Độ rộng xung cho phép = 450 ns (tối thiểu) (Enable pulse width)
t_{DSW} = Thời gian thiết lập dữ liệu (Data set up time) = 195 ns (tối thiểu)
t_H = Thời gian duy trì dữ liệu 10 ns (tối thiểu)
t_{AS}= Thời gian thiết lập trước khi E lên cao đối với RS và R/W = 140 ns (tối thiểu)
t_{AH}= Thời gian duy trì sau khi E chuyển xuống thấp đối với RS và R/W = 10 ns (tối thiểu) (Hold time)

Hình 12.4. Phân chia thời gian của LCD

Bảng 12.4. Danh sách lệnh của LCD

Lệnh	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Mô tả	Thời gian thực hiện
Xoá màn hình	0	0	0	0	0	0	0	0	0	1	Xoá toàn màn hình và đặt địa chỉ 0 của DD RAM vào bộ đếm địa chỉ	1.64 μs
Trở về đầu dòng	0	0	0	0	0	0	0	0	1	-	Đặt địa chỉ 0 của DD RAM như bộ đếm địa chỉ. Trả hiển thị dịch về vị trí gốc DD RAM không thay đổi	1.64 μs
Đặt chế độ truy nhập	0	0	0	0	0	0	0	1	1/D	S	Đặt hướng dịch con trỏ và dịch hiển thị. Thao tác này được thực hiện khi đọc và ghi dữ liệu	40 μs
Điều khiển Bật/tắt hiển thị	0	0	0	0	0	0	1	D	C	B	Đặt Bật/ tắt màn hình (D) Bật/ tắt con trỏ (C) và nhấp nháy ký tự ở vị trí con trỏ (B)	40 μs
Dịch hiển thị hoặc con trỏ	0	0	0	0	0	1	S/C	R/I	-	-	Dịch con trỏ và dịch hiển thị mà không thay đổi DD RAM	40 μs

Đặt chức năng	0 0 0 0 1 DL N F - - -	Thiết lập độ dài dữ liệu (DL) số dòng hiển thị (L) và phông ký tự (F)	40 µs
Đặt địa chỉ CGRAM	0 0 0 1 AGC	Thiết lập địa chỉ C6 RAM dữ liệu CG RAM được gửi đi và nhận sau thiết lập này	40 µs
Thiết lập địa chỉ DD RAM	0 0 1 ADD	Thiết lập địa chỉ DD RAM dữ liệu DD RAM được gửi và nhận sau thiết lập này	40 µs
Cờ bận đọc và địa chỉ	0 1 BF ADD	Cờ bận đọc (BF) báo hoạt động bên trong đang được thực hiện và đọc nội dung bộ đếm địa chỉ	40 µs
Ghi dữ liệu CG hoặc DD RAM	1 0 Ghi dữ liệu	Ghi dữ liệu vào DD RAM hoặc CG RAM	40 µs
Đọc dữ liệu CG hoặc DD RAM	1 1 Đọc dữ liệu	Đọc dữ liệu từ DD RAM hoặc CG RAM	40 µs

Ghi chú:

1. Thời gian thực hiện là thời gian cực đại khi tần số f_{CP} hoặc f_{osc} là 250KHz
2. Thời gian thực hiện thay đổi khi tần số thay đổi. Ví dụ, nếu f_{EP} hay f_{osc} là 270kHz thì thời gian thực hiện được tính 250/270 × 40 = 35µs v.v.
3. Các ký hiệu viết tắt trong bảng là:

DD RAM RAM dữ liệu hiển thị (Display Data RAM)
CG RAM RAM máy phát ký tự (character Generator)
ACC Địa chỉa của RAM máy phát ký tự
ADD Địa chỉ của RAM dữ liệu hiển thị phù hợp với địa chỉ con trỏ.
AC Bộ đếm địa chỉ (Address Counter) dùng cho DD RAM và CG RAM.

1/D = 1	Tăng	1/D = 0	Giảm
S = 1	Kết hợp dịch hiển thị		
S/C = 1	Dịch hiển thị	S/C = 0	Dịch con trỏ
R/L = 1	Dịch sang phải	R/L = 0	Dịch trái
DL = 1	8 bit	DL = 04 bit	
N = 1	2 dòng	N = 1	1 dòng
F = 1	Ma trận điểm 5 × 10	F = 0	Ma trận điểm 5 × 7
BF = 1	Bận	BF = 0	Có thể nhận lênh

12.2 NỐI GHÉP 8051 VỚI ADC VÀ CÁC BỘ CẢM BIẾN

Bộ chuyển đổi ADC

Các bộ chuyển đổi ADC được sử dụng hết sức rộng rãi. Máy tính số làm việc trên các giá trị nhị phân, tuy nhiên, trong thực tế, các đại lượng vật lý đều ở dạng tương tự (liên tục). Nhiệt độ, áp suất, độ ẩm, tốc độ... là một trong những đại lượng vật lý của thế giới thực mà ta thường gặp hàng ngày. Một đại lượng vật lý được chuyển về dòng điện hoặc điện áp qua một thiết bị được gọi là bộ biến đổi. Bộ biến đổi cũng có thể được xem là bộ cảm biến. Mặc dù chỉ có các bộ cảm biến nhiệt, tốc độ, áp suất, ánh sáng và nhiều đại lượng tự nhiên khác, nhưng chúng đều có một điểm chung là cho ra các tín hiệu dòng điện hoặc điện áp ở dạng liên tục. Do vậy, cần một bộ chuyển đổi tương tự số để bộ vi điều khiển có thể đọc được chúng. Chip ADC được sử dụng rộng rãi hiện nay đó là ADC804.

Chip ADC804

Chip ADC804 là bộ chuyển đổi tương tự số thuộc họ ADC800 của hãng National Semiconductor. Chip này cũng được nhiều hãng khác sản xuất. Chip có điện áp nuôi +5V và độ phân giải 8 bit. Ngoài độ phân giải thì thời gian chuyển đổi cũng là một tham số quan trọng khi đánh giá bộ ADC. Thời gian chuyển đổi được định nghĩa là thời gian mà bộ ADC cần để chuyển một đầu vào tương tự thành một số nhị phân. Đối với ADC804 thời gian chuyển đổi phụ thuộc vào tần số đồng hồ được cấp tới chân CLK và CLK IN và không bé hơn 110μs. Các chân của ADC804 có chức năng như sau:

CS (Chip Select) - chọn chip

Là chân chọn chip, đầu vào tích cực mức thấp được sử dụng để kích hoạt chip ADC804. Để truy cập ADC804 thì chân này phải ở mức thấp.

RD (Read) - đọc

Đây là một tín hiệu vào, tích cực mức thấp. Các bộ ADC chuyển đổi đầu vào tương tự thành số nhị phân và giữ nó ở một thanh ghi trong. RD được sử dụng để có dữ liệu được đã chuyển đổi tới đầu ra của ADC804. Khi CS = 0 nếu có một xung cao xuống thấp áp đến chân RD thì dữ liệu ra dạng số 8 bit được đưa tới các chân dữ liệu D0 - D7. Chân RD còn được coi là cho phép đầu ra.

WR (Wtite) - ghi

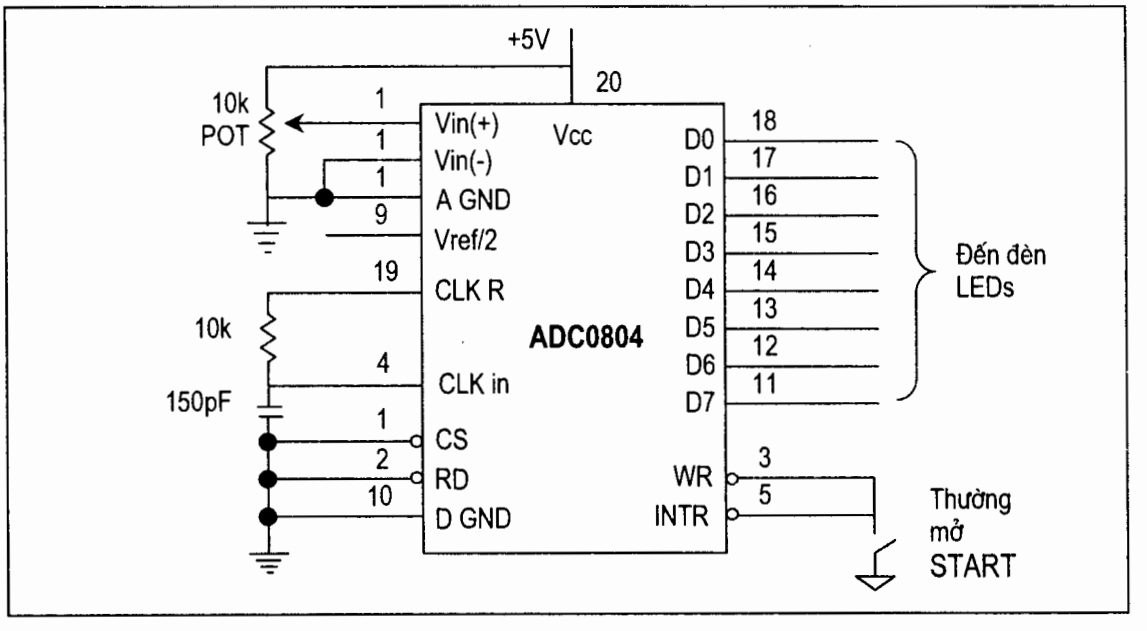
Thực ra, tên chính xác là "Bắt đầu chuyển đổi". Đây là chân vào tích cực mức thấp được dùng để báo cho ADC804 bắt đầu quá trình chuyển đổi. Nếu CS = 0 khi WR tạo ra xung cao xuống thấp thì bộ ADC804 bắt đầu tiến hành chuyển đổi giá trị đầu vào tương tự V_{in} về số nhị phân 8 bit. Lượng thời gian cần thiết để chuyển đổi thay đổi phụ thuộc vào tần số đưa đến chân CLK IN và CLK R. Khi việc chuyển đổi dữ liệu được hoàn tất thì chân INTR được ADC804 hạ xuống thấp.

CLK IN và CLK R

CLK IN là chân vào, nối tới đồng hồ ngoài khi đồng hồ ngoài được sử dụng để tạo thời gian. Tuy nhiên, 804 cũng có một bộ tạo xung đồng hồ trên chip. Để dùng đồng hồ trong (cũng còn được gọi là đồng hồ riêng) của 804 thì các chân CLK IN và CLK R được nối tới một tụ điện và một điện trở như chỉ ra ở hình 12.5. Trong trường hợp này tần số đồng hồ được xác định bằng biểu thức:

$$f = \frac{1}{1{,}1RC}$$

Giá trị thông thường của các đại lượng trên là R = 10kΩ, C= 150pF và tần số nhận được là f = 606kHz, còn thời gian chuyển đổi sẽ là 110μs.



Hình 12.5. Kiểm tra ADC804 ở chế độ chạy tự do

Ngắt INTR (Interrupt)

Ngắt hay còn gọi là "kết thúc việc chuyển đổi'. Đây là chân ra tích cực mức thấp. Bình thường, chân này ở trạng thái cao và khi việc chuyển đổi hoàn tất thì nó xuống thấp để báo cho CPU biết là dữ liệu được chuyển đổi đã sẵn sàng để lấy đi. Sau khi INTR xuống thấp, cần đặt CS = 0 và gửi một xung cao xuống thấp tới chân RD để đưa dữ liệu ra.

Vin (+) và Vin (-)

Đây là hai đầu vào tương tự vi sai, trong đó $V_{in} = V_{in}(+) - V_{in}(-)$. Thông thường $V_{in}(-)$ được nối xuống đất và $V_{in}(+)$ được dùng làm đầu vào tương tự và sẽ được chuyển đổi về dạng số.

Vcc

Là chân nguồn nuôi +5V. Chân này còn được dùng làm điện áp tham chiếu khi đầu vào $V_{ref/2}$ (chân 9) để hở.

Vref/2

Chân 9 là điện áp đầu vào được dùng làm điện áp tham chiếu. Nếu chân này hở (không được nối) thì điện áp đầu vào tương tự cho ADC804 nằm trong dải 0 đến +5V (giống như chân V_{CC}). Tuy nhiên, có nhiều ứng dụng mà đầu vào tương tự áp đến V_{in} khác với dải 0 đến 5V. Chân $V_{ref/2}$ được dùng để thực hiện các điện áp đầu vào có dải khác với 0 - 5V. Ví dụ, nếu dải đầu vào tương tự cần biến đổi từ 0 đến 4V thì $V_{ref/2}$ được nối với +2V.

Bảng 12.5 biểu diễn dải điện áp V_{in} đối với các đầu vào $V_{ref/2}$ khác nhau.

Bảng 12.5. Quan hệ điện áp $V_{ref/2}$ với V_{in}

Vref/ 2(V)	Vin(V)	Kích thước bước (mV)
Hở *	0 đến 5	5/256 = 19.53
2.0	0 đến 4	4/255 = 15.62
1.5	0 đến 3	3/256 = 11.71
1.28	0 đến 2.56	2.56/256 = 10
1.0	0 đến 2	2/256 = 7.81
0.5	0 đến 1	1/256 = 3.90

Ghi chú: - $V_{CC} = 5V$.

- Khi $V_{ref}/2$ hở thì đo được ở đó khoảng 2,5V.
- Kích thước bước (độ phân dải) là sự thay đổi nhỏ nhất mà ADC có thể phân biệt được.

D0 - D7

D0 - D7 là các chân ra dữ liệu số (D7 là bit cao nhất MSB và D0 là bit thấp nhất LSB). Các chân này được đệm ba trạng thái và dữ liệu đã được chuyển đổi chỉ được truy cập khi chân CS = 0 và chân RD đưa xuống mức thấp. Để tính điện áp đầu ra ta có thể sử dụng công thức sau:

$$D_{out} = \frac{Vin}{Kích\ thước\ bước}$$

ở đây:

- D_{out} là đầu ra dữ liệu số (dạng thập phân).;
- V_{in} là điện áp đầu vào tương tự;
- Kích thước bước hay độ phân dải là sự thay đổi nhỏ nhất được tính bằng $(2\times V_{ref}/2)$ chia cho 256 đối với ADC 8 bit.

Chân đất tương tự và chân đất số

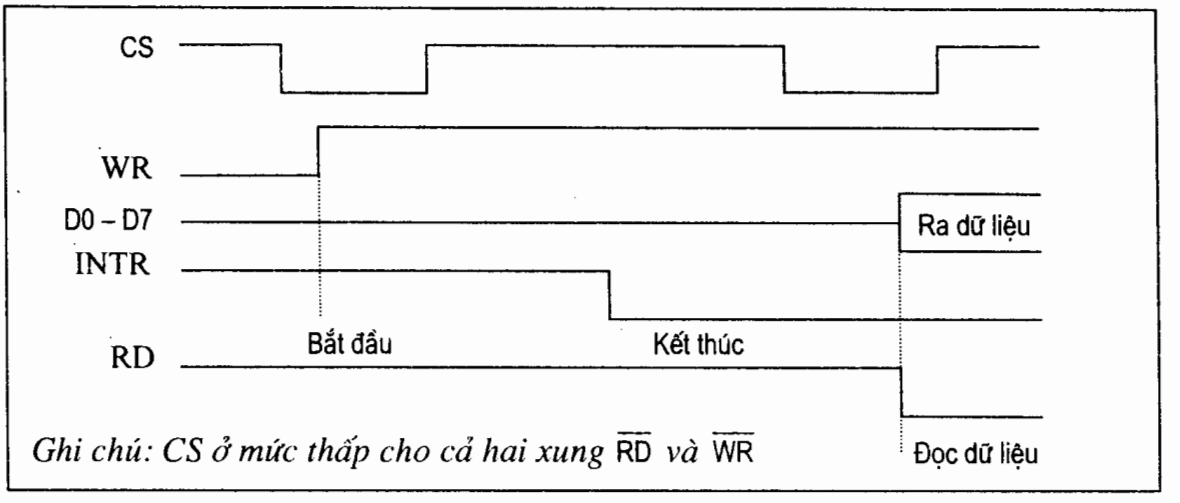
Đây là những chân đầu vào cấp đất chung cho cả tín hiệu số và tương tự. Đất tương tự được nối tới đất của chân V_{in} tương tự, còn đất số được nối tới đất của chân V_{cc}. Lý do có hai đất là để cách ly tín hiệu tương tự V_{in} khỏi các điện áp ký sinh gây ra do các chuyển mạch số đầu ra D0-D7. Việc cách ly này nhằm tăng độ chính xác của dữ liệu ra số. Để đơn giản trong trình bày, hai đất này được nối chung. Tuy nhiên, trong thực tế nhận dữ liệu, các chân đất này là riêng biệt.

Từ những trình bày trên, có thể tóm tắt các bước khi ADC804 thực hiện chuyển đổi dữ liệu là:

1. Bật CS = 0 và gửi một xung thấp lên cao tới chân WR để bắt đầu chuyển đổi.

2. Duy trì kiểm tra chân INTR. Nếu INTR xuống thấp thì việc chuyển đổi được hoàn tất và có thể chuyển sang bước tiếp theo. Nếu INTR còn ở mức cao thì tiếp tục thăm dò cho đến khi nó xuống thấp.

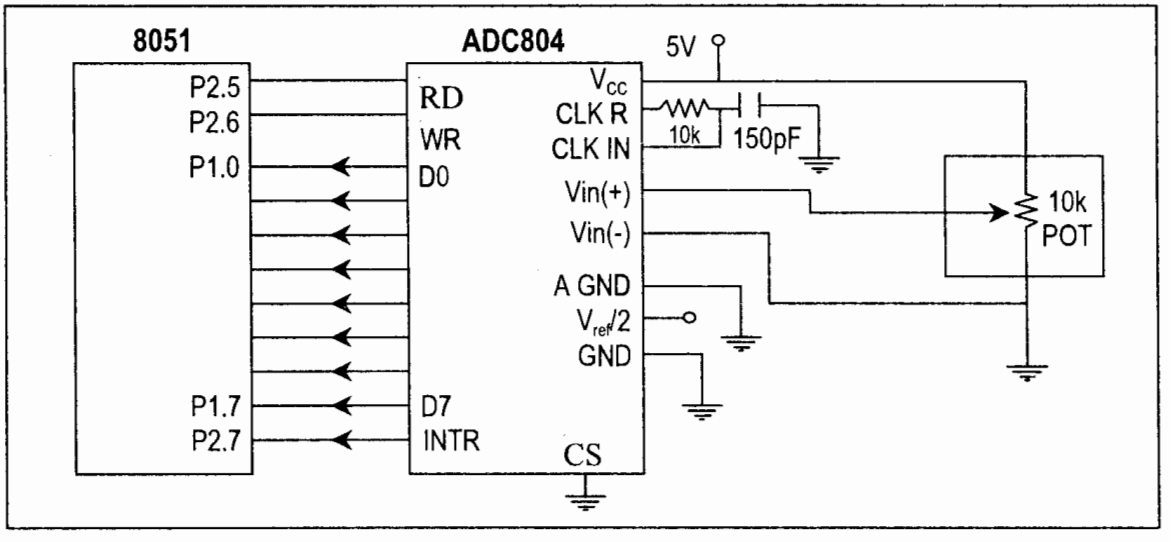
3. Sau khi chân INTR xuống thấp, bật CS = 0 và gửi một xung cao xuống thấp đến chân RD để nhận dữ liệu từ chip ADC804. Phân chia thời gian cho quá trình này được trình bày trên hình 12.6.



Hình 12.6. Phân chia thời gian đọc và ghi của ADC804

Kiểm tra ADC804

Có thể tiến hành kiểm tra ADC804 sử dụng sơ đồ mạch ở hình 12.5. Thiết lập theo sơ đồ này được gọi là chế độ kiểm tra chạy tự do và được nhà sản xuất khuyến cáo nên sử dụng. Ở hình 12.5 có dùng một chiết áp để cấp điện áp tương tự từ 0 đến 5V tới chân đầu vào $V_{in}(+)$ của ADC804. Các đầu ra nhị phân được đưa đến hiển thị trên đèn LED. Cần lưu ý rằng, ở chế độ kiểm tra chạy tự do thì đầu vào CS được nối tới đất và đầu vào WR được nối tới đầu ra INTR. Tuy nhiên,



Hình 12.7. Nối ghép ADC804 với nguồn đồng hồ riêng

theo tài liệu của hãng National Semiconductor "WR và INTR phải được tạm thời đưa xuống thấp ngay sau chu trình cấp nguồn để bảo đảm sự hoạt động bình thường".

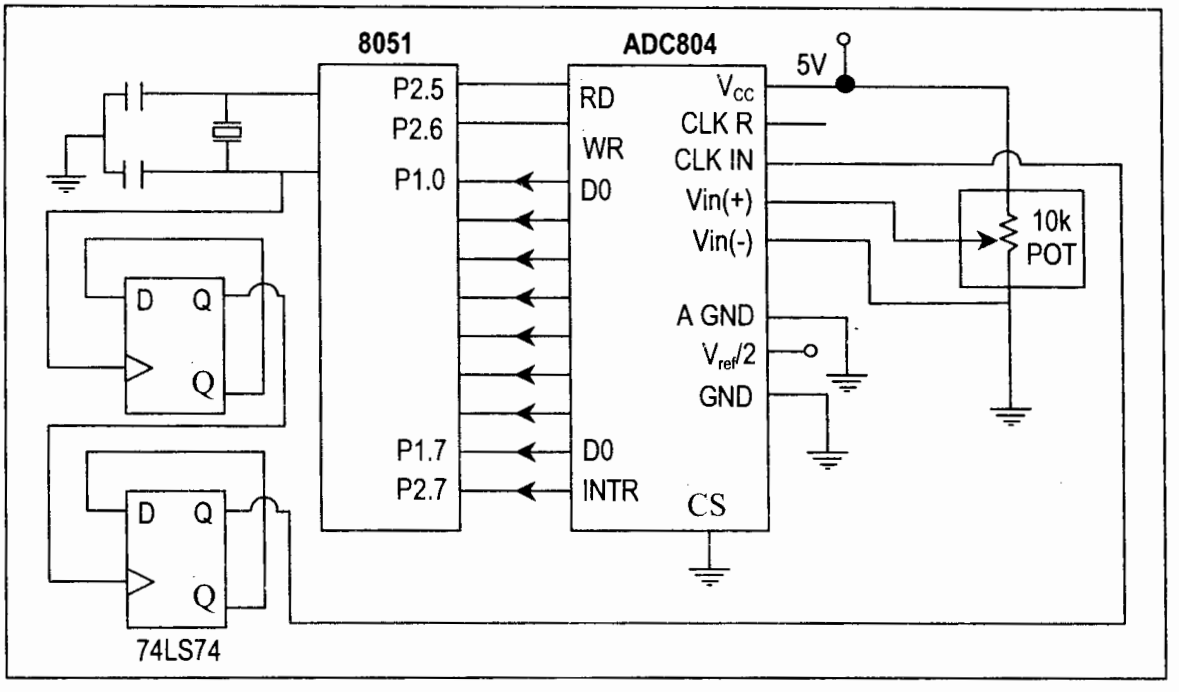
Ví dụ 12.1

Nối ghép ADC804 với 8051 theo sơ đồ 12.7.Viết chương trình kiểm tra chân INTR và chuyển tín hiệu đầu vào tương tự vào thanh ghi A. Sau đó gọi chương trình con chuyển mã Hexa sang ASCII và hiển thị dữ liệu. Thực hiện liên tục công việc này.

Giải:

```
;Đặt P2.6=WR (bắt đầu chuyển đổi cần 1 xung thấp lên cao)
;Đặt chân P2.7=0 khi kết thúc chuyển đổi
;Đặt P2.5=RD(xung Cao-Xuống-Thấp sẽ đọc dữ liệu từ ADC)
;P1.0-P1.7 = D0-D7 của ADC804
   MOV    P1,#0FFH        ;Chọn P1 là cổng đầu vào
BACK:
   CLR    P2.6            ;Đặt WR=0
   SETB   P2.6            ;Đặt WR=1 để bắt đầu chuyển đổi
HERE:
   JB     P2.7,HERE       ;Chờ kết thúc chuyển đổi
   CLR    P2.5            ;Kết thúc chuyển đổi,cho phép RD
   MOV    A,P1            ;Đọc dữ liệu vào thanh ghi A
   ACALL  CONVERSION      ;Chuyển đổi số Hexa ra mã ASCII
   ACALL  DATA-DISPLAY    ;Hiển thị dữ liệu
   SETB   P2.5            ;Đưa RD=1 để cho lần đọc sau.
   SJMP   BACK
```

Trên hình 12.8 ta có thể thấy rằng, tín hiệu đồng hồ vào ADC804 là từ tần số thạch anh của 8051. Vì tần số này quá cao nên chúng ta cần sử dụng hai mạch lật Flip - Flop kiểu D (74LS74) để chia tần số này cho 4. Mỗi mạch lật chia tần số cho 2 nếu nối $\overline{Q}$ tới đầu vào D. Trường hợp tần số còn cao hơn nữa thì cần sử dụng nhiều mạch Flip - Flop hơn.



Hình 12.8. Nối ghép ADC804 với chân đồng hồ XTAL2 của 8051

Nối ghép 8051 với cảm biến nhiệt

Bộ cảm biến (Transducer) chuyển đổi các đại lượng vật lý, ví dụ như nhiệt độ, cường độ ánh sáng, lưu tốc và tốc độ thành các tín hiệu điện. Phụ thuộc vào bộ cảm biến mà đầu ra có thể là tín hiệu dạng điện áp, dòng, trở kháng hay dung kháng. Ví dụ, nhiệt độ được biến đổi thành tín hiệu điện sử dụng một bộ cảm biến nhiệt gọi là Thermistor. Bộ cảm biến nhiệt đáp ứng sự thay đổi nhiệt độ bằng cách thay đổi trở kháng, song đáp ứng này không tuyến tính, xem bảng 12.6.

Bảng 12.6. Trở kháng của bộ cảm biến nhiệt theo nhiệt độ

Nhiệt độ(0C)	Trở kháng của cảm biến (kΩ)
0	29.490
25	10.000
50	3.893
75	1.700
100	0.817

Bảng 12.7. Thông số kỹ thuật chính của cảm biến nhiệt họ LM34

Mã ký hiệu	Dải nhiệt độ	Độ chính xác	Đầu ra
LM34A	-55 F to + 300 C	+ 2.0 F	10mV/F
LM34	-55 F to + 300 C	+ 3.0 F	10mV/F

LM34CA	-40 F to + 230 C	+ 2.0 F	10mV/F
LM34C	-40 F to + 230 C	+ 3.0 F	10mV/F
LM34D	-32 F to + 212 C	+ 4.0 F	10mV/F

Bảng 12.8. Thông số kỹ thuật chính của cảm biến nhiệt họ LM35

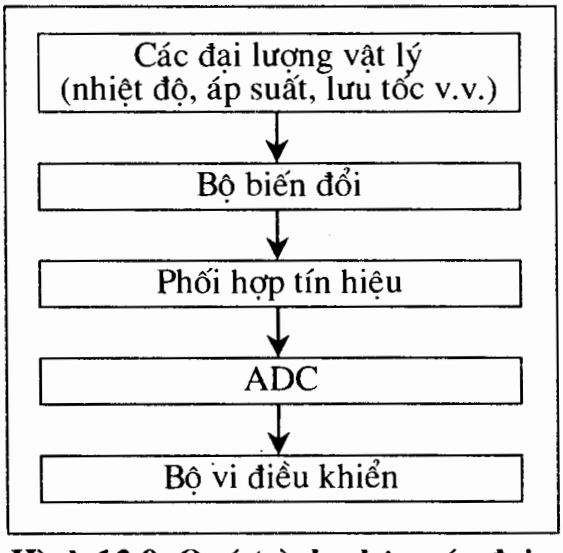
Mã sản phẩm	Dải nhiệt độ	Độ chính xác	Đầu ra
LM35A	-55 C to + 150 C	+ 1.0 C	10 mV/F
LM35	-55 C to + 150 C	+ 1.5 C	10 mV/F
LM35CA	-40 C to + 110 C	+ 1.0 C	10 mV/F
LM35C	-40 C to + 110 C	+ 1.5 C	10 mV/F
LM35D	0 C to + 100 C	+ 2.0 C	10 mV/F

Đối với các bộ cảm biến phi tuyến, việc viết phần mềm tương đối phức tạp, do vậy, nhiều nhà sản xuất đã cho ra thị trường các loạt bộ cảm biến nhiệt tuyến tính. Các bộ cảm biến nhiệt đơn giản và được sử dụng rộng rãi bao gồm họ LM34 và LM35 của hãng National Semiconductor Corp.

Cảm biến nhiệt họ LM34 và LM35

LM34 là họ cảm biến nhiệt, mạch tích hợp, chính xác cao có điện áp đầu ra tỷ lệ tuyến tính với nhiệt độ Fahrenheit, xem hình 12.7. Họ LM34 không yêu cầu căn chỉnh bên ngoài vì vốn nó đã được căn chỉnh rồi. Họ này cho điện áp ra thay đổi 10mV ứng với thay đổi nhiệt độ 1^0F. Bảng 12.7 giới thiệu một số thông số kỹ thuật chính của họ LM34.

LM35 cũng là họ cảm biến nhiệt mạch tích hợp chính xác cao có điện áp đầu ra tỷ lệ tuyến tính với nhiệt độ theo thang độ Celsius. Họ cảm biến này cũng không yêu cầu căn chỉnh ngoài vì vốn nó đã được căn chỉnh. Họ này cho ra điện áp 10mV ứng với thay đổi nhiệt độ là 1^0C. Bảng 12.7 giới thiệu một số thông số kỹ thuật chính của họ LM35.



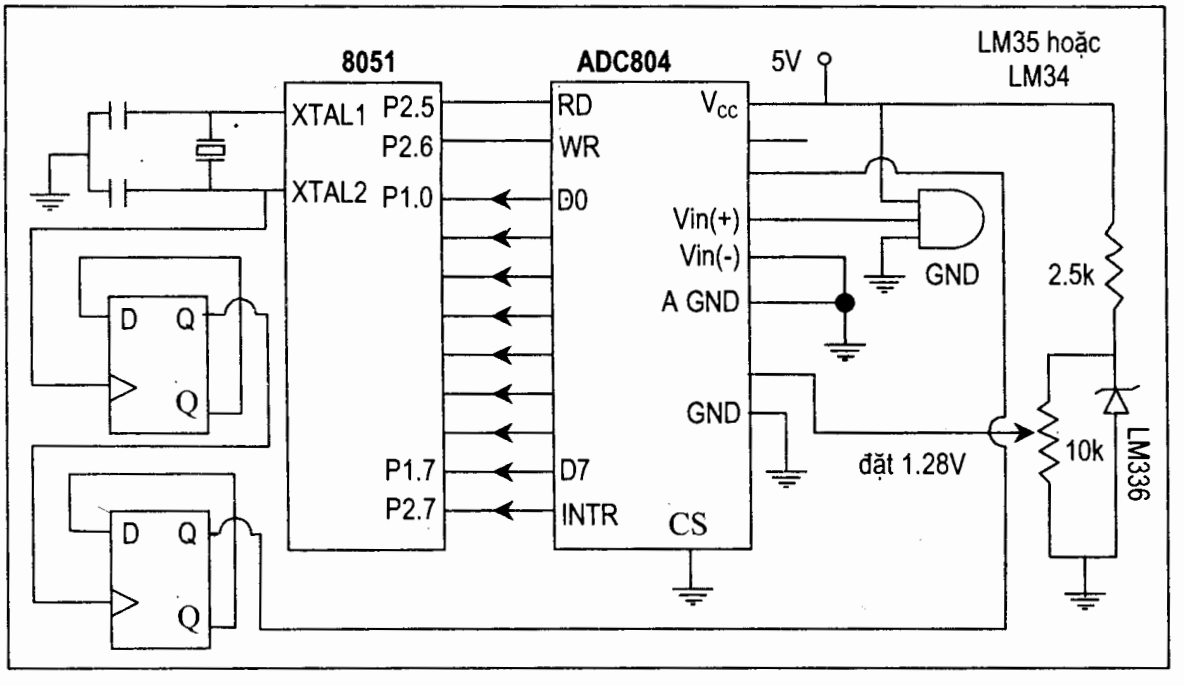
Hình 12.9. Quá trình nhận các đại lượng vật lý

Phối hợp tín hiệu và nối ghép LM35 với 8051

Phối hợp tín hiệu là một thuật ngữ được sử dụng rộng rãi trong lĩnh vực thu nhận dữ liệu. Hầu hết các bộ cảm biến đều đưa ra tín hiệu dạng điện áp, dòng điện, dung kháng hoặc trở kháng. Tuy nhiên, chúng ta cần chuyển đổi các tín hiệu này về điện áp để đưa đến đầu vào của bộ chuyển đổi ADC. Sự chuyển đổi (biến đổi) này được gọi chung là *phối hợp tín hiệu*. Phối hợp tín hiệu có thể là chuyển dòng điện thành điện áp hoặc khuyếch đại tín hiệu. Ví dụ, bộ cảm biến nhiệt thay đổi trở kháng theo nhiệt độ. Sự thay đổi trở kháng cần được chuyển thành điện áp để các bộ ADC có thể sử dụng được. Xét trường hợp nối LM35 tới ADC804. Vì ADC804 có độ phân dải 8 bit với tối đa có 256 mức (2^8), và LM35 (hoặc ML34) tạo ra điện áp 10mV

Bảng 12.9. Nhiệt độ và Vout của ADC804

Nhiệt độ (0C)	Vin (mV)	Vout (D7 – D0)
0	0	0000 0000
1	10	0000 0001
2	20	0000 0010
3	30	0000 0011
10	100	0000 1010
30	300	0001 1110



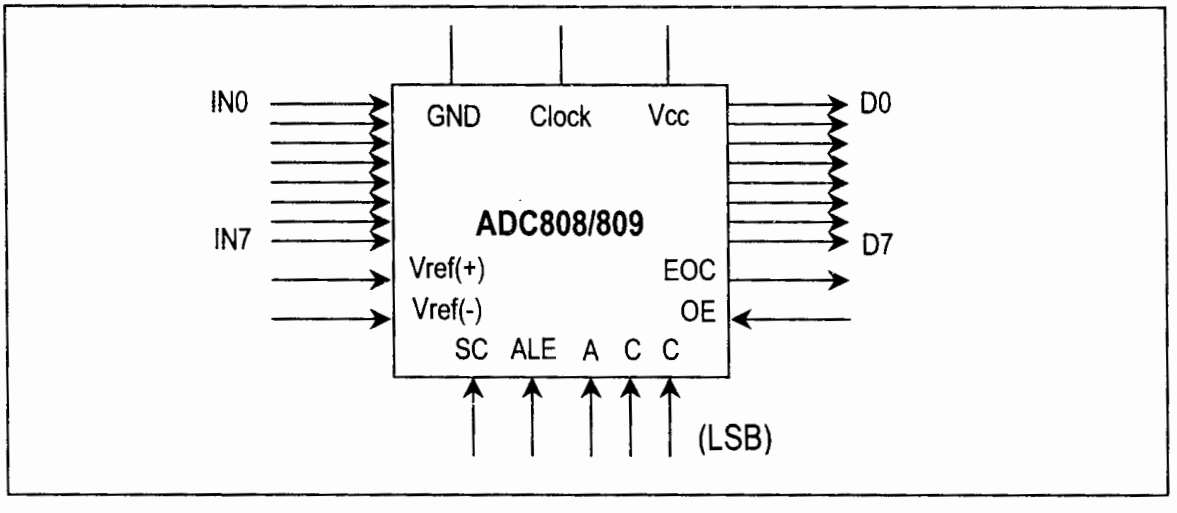
Hình 12.10. Nối gép 8051 với ADC804 và bộ cảm biến nhiệt

ứng với sự thay đổi nhiệt độ 1^0C, nên ta có thể phối hợp V_{in} của ADC804 để tạo ra V_{out} = 2560mV (2,56V) ở đầu ra đầy thang đo. Để tạo ra V_{out} đầy thang 2,56V của ADC804 ta cần đặt điện áp $V_{ref}/2$ = 1,28V. Như vậy, V_{out} của ADC804 đáp ứng trực tiếp nhiệt độ được giám sát trên LM35 (xem bảng 12.9). Các giá trị của $V_{ref}/2$ được cho ở bảng 12.5.

Hình 12.10 trình bày cách nối ghép bộ cảm biến nhiệt với ADC804. Ở đây dùng diode zener LM336 - 2.5 để cố định điện áp 2,5V trên chiết áp 10kΩ. Sử dụng LM336 - 2.5 có thể khắc phục được sự thăng giáng của nguồn nuôi.

Chip ADC808/809 8 kênh tương tự

Một chip rất hữu ích khác của National Semiconductor là ADC808/809, xem hình 12.11. ADC804 chỉ có một đầu vào tương tự, còn chip này có 8 kênh đầu vào. Như vậy, chip này cho phép giám sát đồng thời 8 bộ cảm biến. Lưu ý rằng, ADC808/809 cũng có đầu ra dữ liệu 8 bit như ADC804. 8 kênh đầu vào tương tự được dồn kênh và được chọn như cho ở bảng 12.10 nhờ ba chân địa chỉ A, B và C.



Hình 12.11. Bộ biến đổi ADC 808/809

Bảng 12.10. Chọn kênh tương tự của ADC808

Chọn kênh tương tự	C	B	A
IN0	0	0	0
IN1	0	0	1
IN2	0	1	0
IN3	0	1	1
IN4	1	0	0

IN5	1	0	1
IN6	1	1	0
IN7	1	1	1

Đối với ADC808/809, các điện áp $V_{rer}(+)$ và $V_{ref}(-)$ thiết lập điện áp tham chiếu. Nếu $V_{ref}(-1)$ = Gnd và $V_{ref}(+)$ = 5V thì độ phân dải là 5V/256 = 19,53 mV. Do vậy, để có độ phân dải 10 mV ta cần đặt $V_{ref}(+)$ = 2,56 V và $V_{ref}(-)$ = Gnd. Lưu ý đến chân ALE ở hình 12.11. Các chân địa chỉ A, B và C được dùng để chọn kênh đầu vào IN0 – IN7 và kích hoạt chân ALE chốt địa chỉ. Chân SC dùng để bắt đầu chuyển đổi (Start Conversion). Chân EOC dùng để kết thúc chuyển đổi (End - Of - Conversion) và chân OE cho phép đọc đầu ra (Out put Enable).

Các bước lập trình cho ADC 808/809

Các bước chuyển dữ liệu từ đầu vào của ADC808/809 vào bộ vi điều khiển như sau:

1. Chọn một kênh tương tự bằng cách dùng các chân địa chỉ A, B và C theo bảng 12.10.
2. Kích hoạt chân cho phép chốt địa chỉ ALE (Address Latch Enable) bằng cách đưa một xung thấp lên cao để chốt địa chỉ.
3. Kích hoạt chân SC bằng xung cao xuống thấp để bắt đầu chuyển đổi.
4. Giám sát EOC để báo kết thúc chuyển đổi. Đầu ra cao xuống thấp báo rằng dữ liệu đã được chuyển đổi và cần phải được lấy đi.
5. Kích hoạt chân đọc dữ liệu ra OE của ADC bằng xung cao xuống thấp.

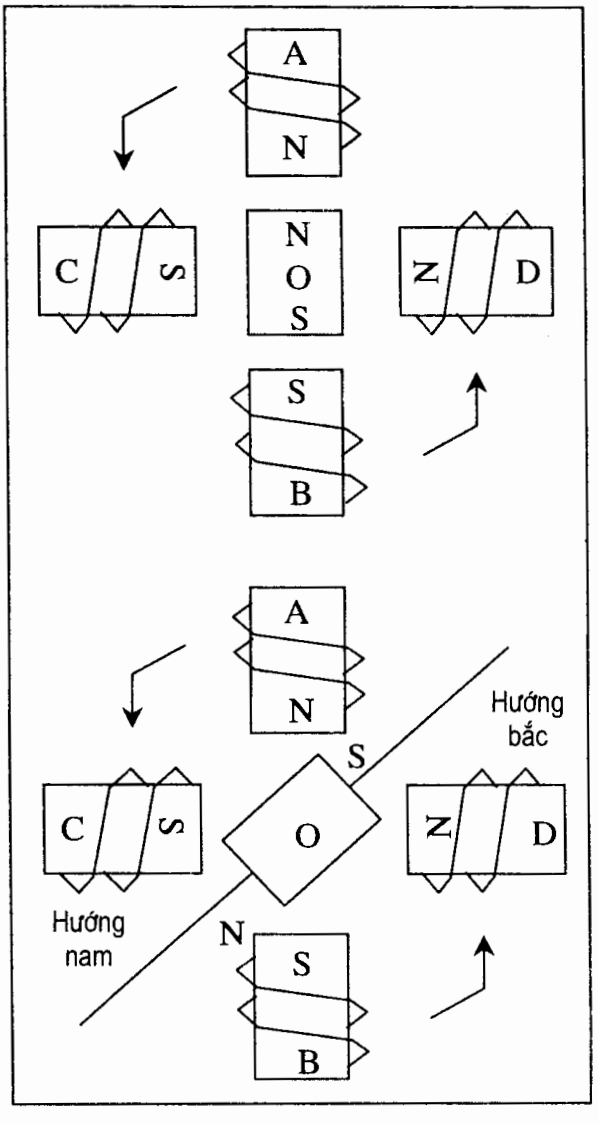
Lưu ý rằng ADC 808/809 không có đồng hồ riêng và do vậy phải cấp xung đồng bộ ngoài đến chân CLK. Mặc dù tốc độ chuyển đổi phụ thuộc vào tần số đồng hồ được nối đến CLK nhưng tốc độ này không thể vượt quá 100ms.

Chương 13
NỐI GHÉP VỚI THẾ GIỚI THỰC II - ĐỘNG CƠ BƯỚC, BÀN PHÍM VÀ BỘ BIẾN ĐỔI DAC

13.1 NỐI GHÉP VỚI ĐỘNG CƠ BƯỚC

Động cơ bước

Động cơ bước là một thiết bị được sử dụng rộng rãi dùng để chuyển các xung điện thành chuyển động cơ học. Ở một số ứng dụng, chẳng hạn như bộ điều khiển đĩa, máy in kim ma trận và robot, thì động cơ bước được dùng để điều khiển chuyển động. Mỗi động cơ bước đều có phần quay rotor là nam châm vĩnh cửu, được bao xung quanh là một phần tĩnh, gọi là stator, xem hình 13.1. Động cơ bước có 4 cuộn dây stator được sắp xếp theo cặp đối xứng qua tâm, xem hình 13.2. Động cơ bước dạng này còn được gọi là động cơ bước 4 pha. Điểm giữa cho phép thay đổi chiều dòng điện của một trong hai lõi khi một cuộn dây được nối đất, do đó đổi cực của stator. Lưu ý rằng, trục của động cơ thông thường thì quay tự do, còn trục của động cơ bước thì quay theo từng bước cố định, lặp lại và đến từng vị trí cụ thể. Động cơ bước quay được như vậy là từ cơ sở của lý thuyết từ



Hình 13.1. Bố trí rotor

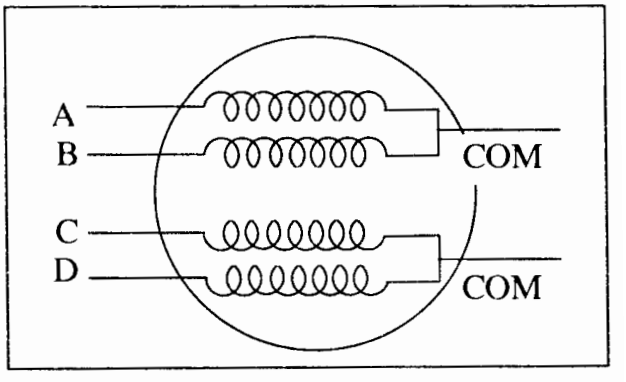
trường: các cực cùng dấu đẩy nhau và các cực khác dấu hút nhau. Chiều quay được xác định bởi từ trường của stator, mà từ trường này thì do dòng điện chạy qua lõi cuộn dây gây nên. Khi hướng của dòng thay đổi thì cực từ trường cũng thay đổi theo, gây ra chuyển động ngược lại của động cơ (đảo chiều). Động cơ bước ở đây có 6 đầu dây: 4 đầu của cuộn dây stator và 2 đầu dây chung điểm giữa của các cặp dây. Khi chuỗi xung nguồn được cấp đến từng cuộn dây stator thì động cơ sẽ quay. Mỗi chuỗi xung có thể có cấp độ chính xác khác nhau. Bảng 13.1 giới thiệu chuỗi 4 bước thông thường.

Bảng 13.1. Chuỗi xung 4 bước thông thường

Chiều kim đồng hồ	Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D	Chiều quay bộ đếm
↓	1	1	0	0	1	↑
	2	1	1	0	0	
	3	0	1	1	0	
	4	0	0	1	1	

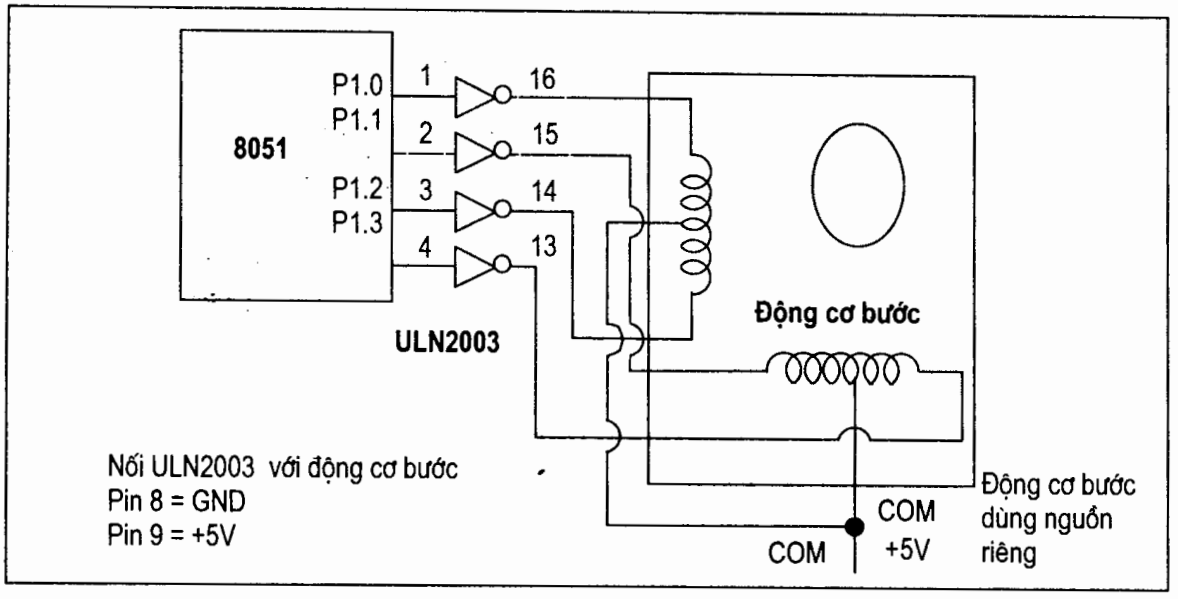
Bảng13.2. Góc bước của động cơ bước

Góc bước	Số bước/vòng
0.72	500
1.8	200
2.0	180
2.5	144
5.0	72
7.5	48
15	24



Hình 13.2. Bố trí cuộn dây stator

Cần nhớ rằng, chúng ta có thể bắt đầu với chuỗi xung nào đó trong bảng 13.1, song khi đã bắt đầu với chuỗi xung nào thì cần phải tiếp tục theo đúng thứ tự của chuỗi xung đó. Ví dụ, nếu bắt đầu bước thứ ba là chuỗi (0110) thì cần tiếp tục với chuỗi của bước 4 rồi sau đó lặp lại 1, 2, 3 v.v.



Hình 13.3. Nối ghép 8051 với động cơ bước

Ví dụ 13.1

Hãy trình bày việc nối ghép 8051 với động cơ bước ở hình 13.3 và viết chương trình điều khiển động cơ quay liên tục.

Giải:

Các bước nối ghép 8051 với động cơ bước như sau:

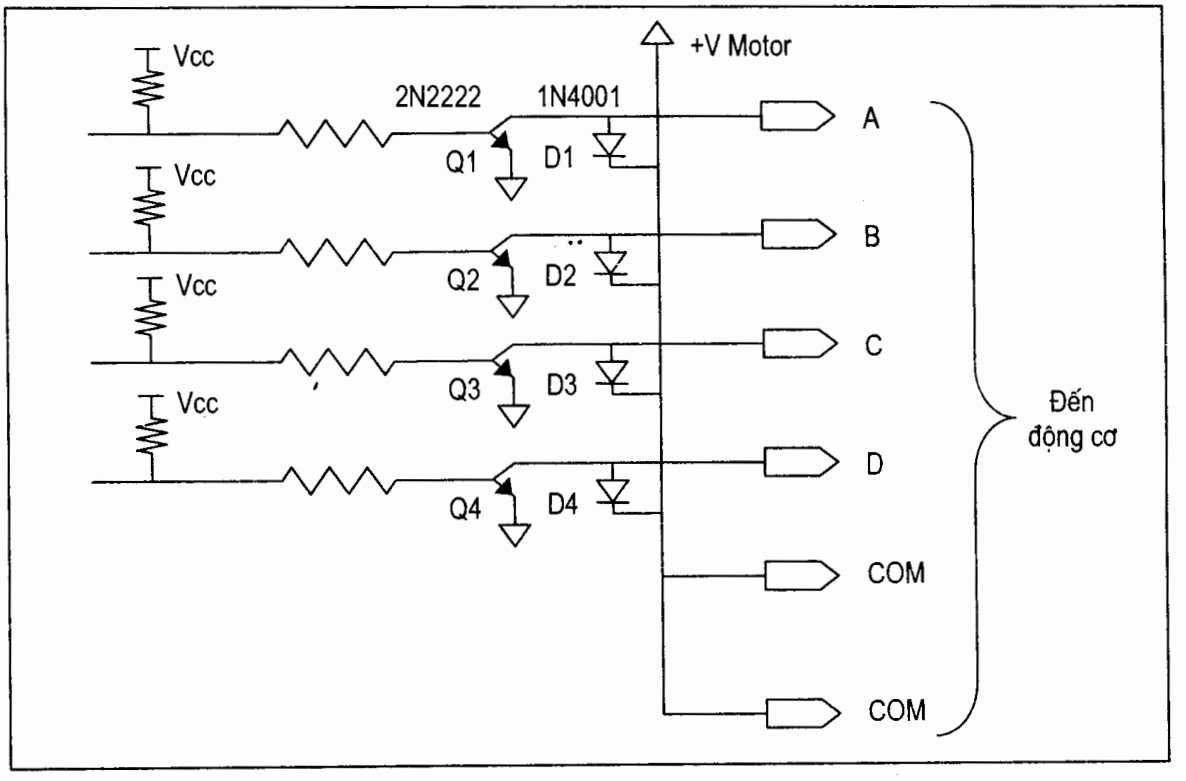
1. Dùng ôm kế đo trở kháng của các đầu dây nhằm xác định đầu dây chung COM để nối với nhau.

2. Các đầu dây chung được nối tới điện áp dương của nguồn cấp cho động cơ. Nhiều động cơ thường dùng + 5V.

3. Bốn đầu của cuộn dây stator được 4 bit cổng P1 (P1.0 - P1.3) của 8051 điều khiển. Tuy nhiên, vì 8051 không đủ dòng để điều khiển các cuộn dây động cơ bước nên cần sử dụng một bộ điều khiển chẳng hạn như ULN2003 để cấp đủ dòng cho stator. Có thể thay ULN2003 bằng các bóng bán dẫn như giới thiệu ở hình 13.4. Dĩ nhiên, nếu dùng bóng bán dẫn làm bộ điều khiển thì cần sử dụng các diode để ngăn dòng cảm ứng ngược tạo ra khi tắt cuộn dây. Vì thế, một lý do mà ULN2003 được ưa chuộng chính là vì nó đã có sẵn diode bên trong để ngăn dòng cảm ứng ngược.

```
     MOV  A,#66H          ;Nạp chuỗi xung bước
BACK:MOV  P1,A            ;Xuất chuỗi xung đến động cơ
     RR   A               ;Quay theo chiều kim đồng hồ
     ACALL DELAY          ;Chờ
     SJMP BACK            ;Tiếp tục chạy
     ---
DELAY
     MOV  R2,#100
H1:  MOV  R3,#255
H2:  DJNZ R3,H2
     DJNZ R2,H1
     RET
```

Bạn thử thay đổi giá trị của DELAY để đặt lại tốc độ quay. Ngoài ra, Bạn cũng có thể sử dụng lệnh xử lý bit SETB và CLR thay cho lệnh RRA để tạo ra chuỗi xung.



Hình 13.4. Sử dụng bóng bán dẫn điều khiển động cơ bước

Một số thông số và khái niệm

Góc bước (Step Angle)

Câu hỏi đặt ra là mỗi bước có độ dịch chuyển là bao nhiêu? Điều này phụ thuộc vào cấu trúc bên trong của động cơ, đặc biệt là số răng của stator và rotor. Góc bước là độ quay nhỏ nhất của một bước. Các động cơ khác nhau có góc bước khác nhau. Bảng 13.2 giới thiệu góc bước của một số chủng loại động cơ, trong đó có dùng thuật ngữ số bước trong một vòng (Steps per revolution). Đây là tổng số bước cần để quay hết một vòng 360^0 (chẳng hạn 180 bước $\times 2^0 = 360^0$).

Dường như trái với ấn tượng ban đầu, động cơ bước không cần nhiều đầu dây ở cuộn stator để có góc bước nhỏ hơn. Tất cả động cơ bước được nói ở phần này chỉ dùng 4 đầu ở cuộn dây stator và 2 đầu dây chung ở nút giữa. Mặc dù nhiều hãng sản xuất chỉ dùng một đầu chung, song vẫn phải có 4 đầu dây stator.

Quan hệ số bước/giây và số vòng quay/phút RPM

Quan hệ giữa số vòng quay/phút RPM (Revolutions Per Minute) với số bước của một vòng quay và số bước/giây là quan hệ trực quan và được biểu diễn như sau:

$$\text{Số bước trong giây} = \frac{RPM \times \text{số bước trong vòng quay}}{60}$$

Chuỗi xung bốn bước và số răng trên rotor

Chuỗi xung chuyển mạch trình bày ở bảng 13.1 được gọi là chuỗi chuyển mạch 4 bước, bởi vì sau 4 bước thì hai cuộn dây giống nhau sẽ được bật "ON". Vậy sau 4 bước này động cơ quay được bao nhiêu? Sau khi thực hiện xong 4 bước thì rotor chỉ quay được một bước răng. Do vậy, ở động cơ 200 bước/vòng thì rotor có 50 răng vì $50 \times 4 = 200$ bước cần để quay hết một vòng. Như vậy, có thể kết luận là góc bước tối thiểu luôn là hàm của số răng trên rotor. Nói cách khác, góc bước càng nhỏ thì rotor quay được càng nhiều răng. Hãy xét ví dụ 13.2:

Ví dụ 13.2

Hãy tính số lần của chuỗi 4 bước nêu ở bảng 13.1 cấp cho một động cơ bước để quay được 80^0 nếu động cơ có góc bước là 2^0.

Giải:

Động cơ có góc bước là 2^0 thì có những thông số sau: góc bước 2^0, số bước/vòng là 180, số răng của rotor là 45, góc quay sau mỗi chuỗi 4 bước là 8^0. Vậy để quay 80^0 thì cần 10 chuỗi 4 bước vì $10 \times 4 \times 2 = 80^0$.

Từ ví dụ 13.2 sẽ có người hỏi, vậy muốn quay 45^0 thì làm thế nào khi góc bước là 2^0. Muốn có độ phân giải nhỏ hơn thì động cơ bước cần dùng chuỗi chuyển mạch 8 bước. Chuỗi 8 bước cũng còn được gọi chuỗi nửa bước (half - stepping), vì ở chuỗi 8 bước, thì mỗi bước là một nửa của góc bước bình thường. Ví dụ, một động cơ có góc bước là 2^0 có thể sử dụng góc bước 1^0 nếu áp dụng chuỗi ở bảng 13.3.

Bảng 13.3. Chuỗi xung 8 bước

Chiều kim đồng hồ	Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D	Chiều quay bộ đếm
↓	1	1	0	0	1	↑
	2	1	0	0	0	
	3	1	1	0	0	
	4	0	1	0	0	
	5	0	1	1	0	
	6	0	0	1	0	
	7	0	0	1	1	
	8	0	0	0	1	

Tốc độ động cơ

Tốc độ động cơ được đo bằng số bước trong một giây (bước/giây) là một hàm của tốc độ chuyển mạch. Từ ví dụ 13.1 có thể thấy rằng, bằng việc thay đổi thời gian trễ ta có thể đạt được các tốc độ quay khác nhau.

Moment giữ

Moment giữ được định nghĩa là lượng moment ngoài cần thiết để làm quay trục động cơ từ vị trí giữ của nó với điều kiện trục động cơ đang đứng yên hoặc đang quay với tốc độ RPM = 0. Đại lượng này được đo bằng tỷ lệ điện áp và dòng cấp đến động cơ. Đơn vị của moment giữ là kilôgam - centimet.

Chuỗi 4 bước điều khiển dạng sóng

Ngoài chuỗi 4 bước và 8 bước được trình bày trên đây, còn có một chuỗi khác được gọi là chuỗi 4 bước dạng sóng. Chuỗi này được giới thiệu ở bảng 13.4.

Để ý 8 bước trong bảng 13.3 là sự kết hợp đơn giản của các chuỗi 4 bước thường và chuỗi 4 bước điều khiển dạng sóng được cho ở bảng 13.1 và 13.4.

Bảng 13.4. Chuỗi xung 4 bước

Chiều kim đồng hồ	Bước	Cuộn dây A	Cuộn dây B	Cuộn dây C	Cuộn dây D	Chiều quay bộ đếm
↓	1	1	0	0	0	↑
	2	0	1	0	0	
	3	0	0	1	0	
	4	0	0	0	1	

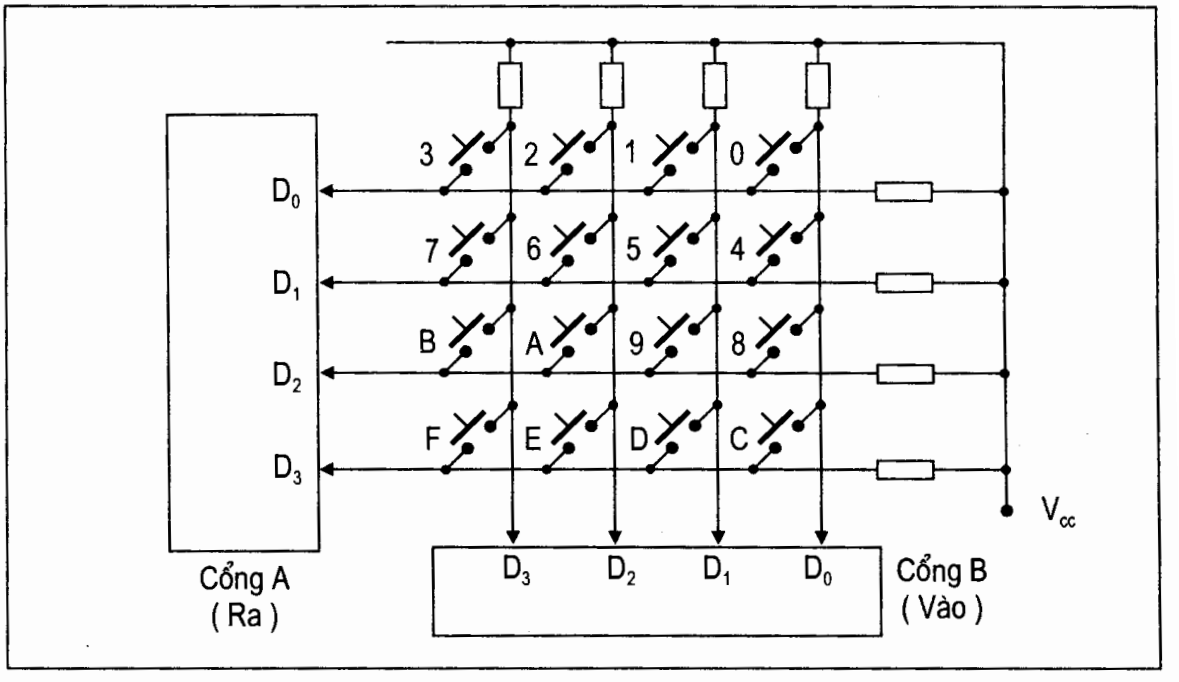
13.2 NỐI GHÉP 8051 VỚI BÀN PHÍM

Nối ghép bàn phím với 8051

Ở dạng đơn giản, bàn phím được tổ chức theo kiểu ma trận các hàng và các cột. CPU truy cập cả hàng lẫn cột thông qua các cổng. Do vậy, với hai cổng 8 bit thì có thể nối một bàn phím 64 phím (8 × 8) tới bộ vi xử lý. Khi một phím được nhấn thì một hàng và một cột được tiếp xúc, các hàng và cột còn lại không có sự tiếp xúc nào. Trong các bàn phím máy tính IBM PC có một bộ vi điều khiển (bao gồm một bộ vi xử lý, bộ nhớ RAM, EPROM và một số cổng tất cả được bố trí trên một chip) chịu trách nhiệm nối ghép phần cứng và phần mềm của bàn phím. Ở những hệ như vậy, chương trình được lưu trong EPROM của bộ vi điều khiển đảm nhiệm quét liên tục các phím, xác định xem phím nào đã được kích hoạt và gửi thông tin đến bo mạch chính. Trong phần này, chúng ta nghiên cứu về cơ cấu 8051 quét và xác định phím nhấn.

Quét và xác định phím

Hình 13.5 trình bày một ma trận 4 × 4 được nối tới hai cổng. Các hàng được nối tới một cổng ra, còn các cột được nối tới một cổng vào. Nếu không có phím nào được nhấn thì đọc cổng vào sẽ được toàn là 1 vì tất cả các cột đều được nối tới nguồn V_{CC}. Nếu tất cả các hàng được nối đất và một phím được nhấn thì một trong các cột sẽ có giá trị 0 vì phím được nhấn nối cột xuống đất. Chức năng của bộ vi điều khiển là quét liên tục để phát hiện và xác định phím được nhấn.



Hình 13.5. Nối ghép ma trận bàn phím tới các cổng

Nối đất các hàng và đọc các cột

Để phát hiện phím nhấn, bộ vi xử lý nối đất tất cả các hàng bằng cách đặt giá trị 0 lên các chốt ra, sau đó đọc các cột. Nếu dữ liệu đọc được ở các cột có giá trị D3-D0 = 1111 tức là không có phím nào được nhấn và quá trình này cứ tiếp tục cho đến khi phát hiện ra phím được nhấn. Nếu một trong các bit của cột có giá trị bằng 0, điều đó xác nhận có phím được nhấn. Ví dụ, nếu D3 - D0 = 1101 có nghĩa là phím ở cột D1 được nhấn. Sau khi một phím nhấn đã được phát hiện, bộ vi xử lý sẽ chuyển qua quá trình xác định phím nhấn đó. Bắt đầu từ hàng trên cùng, bộ vi xử lý sẽ nối đất hàng đó bằng cách đưa vào một điện áp thấp cho hàng D0, sau đó nó tiến hành đọc các cột. Nếu dữ liệu đọc được có giá trị toàn là 1 tức là không phím nào ở hàng này được kích hoạt cả thì quá trình sẽ chuyển sang hàng tiếp theo. Bộ vi xử lý lại nối đất hàng tiếp theo, đọc giá trị ở các cột và kiểm tra xem có giá trị nào bằng 0 không. Quá trình này tiếp tục cho đến khi có hàng được xác định. Sau khi xác định xong hàng có phím nhấn nhiệm vụ tiếp theo là tìm xem cột nào có phím nhấn. Việc này cũng khá đơn giản bởi vì CPU biết được bất cứ thời điểm nào hàng nào và cột nào được truy nhập. Xem ví dụ 13.3.

Ví dụ 13.3

Từ hình 13.3 hãy xác định hàng và cột của phím được nhấn cho các trường hợp sau:

a) Ở hàng: D3 - D0 = 1110
ở cột: D3 - D0 = 1011

b) Ở hàng: D3 - D0 = 1101
ở cột D3 - DU = 0111

Giải:

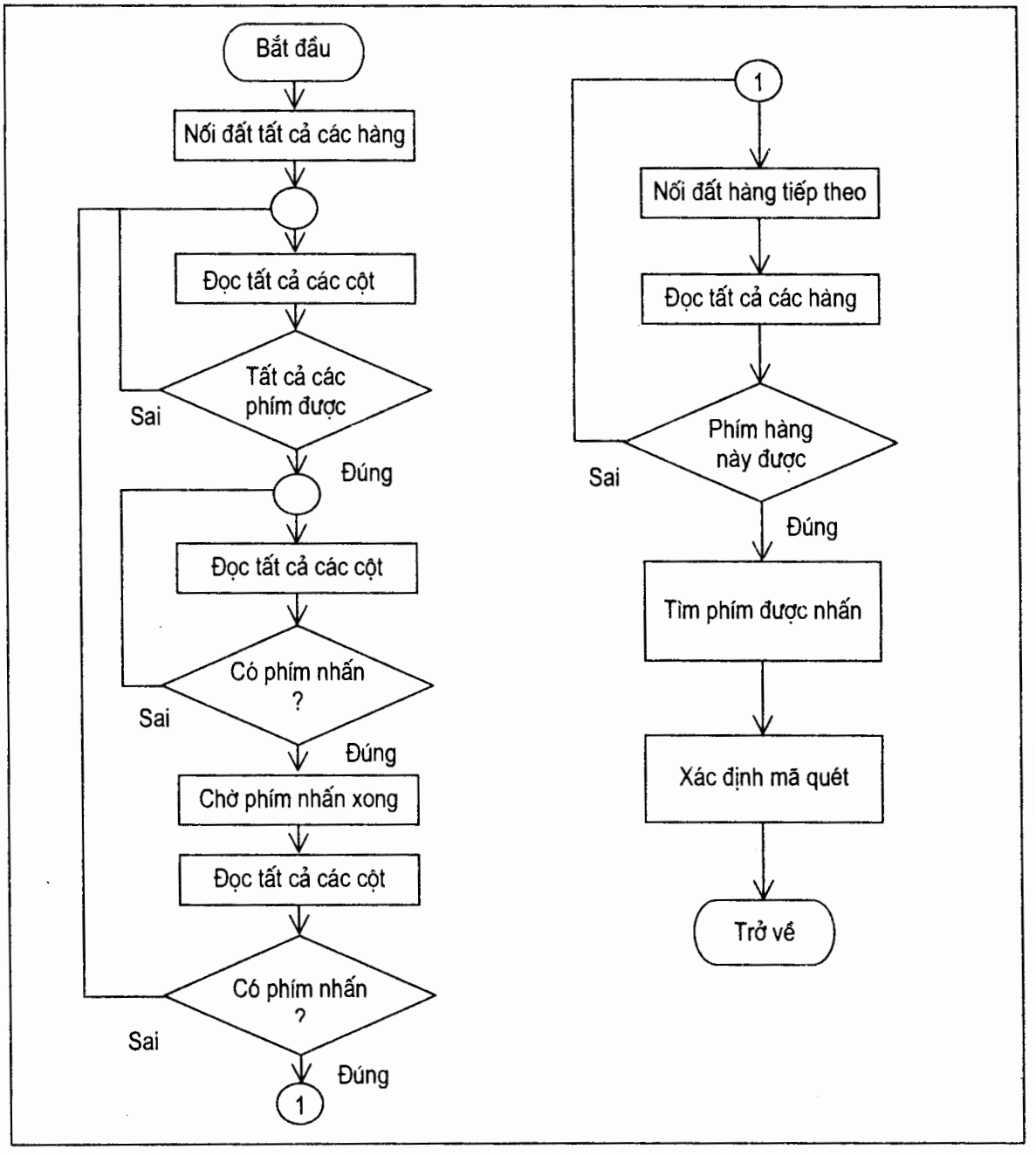
Từ hình 13.3 có thể xác định được phím nhấn:

a) Hàng D0 và cột D2 được nhấn - vì vậy phím nhấn là phím số 2.

b) Hàng D1 và cột D3 được nhấn, vì vậy phím số 7 là phím được nhấn.

Chương trình 13.1 là chương trình hợp ngữ dùng để phát hiện và xác định phím được kích hoạt. Ở chương trình này giả sử rằng P1 và P2 được khởi tạo tương ứng là lối ra và lối vào. Chương trình thực hiện qua 4 giai đoạn chính như sau:

1. Để bảo đảm rằng phím nhấn trước đó đã được nhả ra, các giá trị 0 cùng một lúc áp đến tất cả các hàng, và tiến hành đọc nhiều lần giá trị các cột cho đến khi tất cả các cột có giá trị cao. Khi tất cả các cột đã có giá trị cao, chương trình sẽ đợi trong một lúc trước khi chuyển sang bước tiếp theo là chờ một phím được nhấn.
2. Để kiểm tra xem có phím nào được nhấn, các cột được quét theo một chu trình lặp cho đến khi một trong các cột có giá trị 0. Cần nhớ rằng các chốt lối ra nối với các hàng vẫn còn có các giá trị 0 ban đầu (do thực hiện từ bước 1), có nghĩa là chúng được nối đất. Sau khi phát hiện được phím nhấn, chương trình sẽ chờ thêm 20ms rồi mới quét các cột một lần nữa. Làm như vậy với 2 mục đích: (a) bảo đảm việc phát hiện ra phím nhấn không phải do lỗi xung nhiễu gây ra và (b) trễ 20ms ngăn ngừa việc coi phím đó được nhấn nhiều lần. Nếu sau khi giữ chậm 20ms phím đó vẫn còn bị nhấn thì chương trình sẽ chuyển sang bước tiếp theo là xác định phím đó thuộc về hàng nào, hay nói cách khác chương trình trở về vòng lặp để xác định phím được nhấn thực sự.



Hình 13.6 Lược đồ chương trình 13.1

3. Để phát hiện phím nhấn thuộc về hàng nào, cần nối đất từng hàng một và mỗi lần như vậy phải đọc giá trị các cột. Nếu tất cả các cột đều ở mức cao có nghĩa là phím được nhấn không thuộc hàng đó; vì vậy cần nối đất hàng tiếp theo và

cứ thế tiếp tục cho đến khi tìm được hàng có phím được nhấn. Khi tìm ra hàng có phím nhấn có nghĩa chúng ta có được thông tin cần thiết để xác định vị trí phím thông qua việc đối chiếu bảng chứa mã quét bàn phím.

4. Để xác định phím nhấn cần quay từng bit cột vào cờ nhớ và kiểm tra xem liệu có giá trị thấp hay không. Khi tìm thấy giá trị 0, có nghĩa là đã xác định được mã quét cho phím nhấn; nếu không thì con trỏ sẽ tiếp tục tăng thêm 1 để trỏ đến phần tử tiếp theo của bảng mã quét.

Việc phát hiện phím nhấn là chuẩn chung cho mọi bàn phím, song việc xác định phím nào được nhấn ở các bàn phím khác nhau lại khác nhau. Phương pháp dùng bảng mã quét được chỉ ở chương trình 13.1 có thể cải tiến để làm việc với bất kỳ ma trận 8 × 8 nào. Hình 13.6 giới thiệu sơ đồ của chương trình 13.1 thực hiện việc quét và xác định phím nhấn.

Có những chip IC như MM74C923 của National Semiconductor có thể tích hợp quá trình quét và giải mã bàn phím trên một chip. Các chip như vậy (không phải là bộ vi xử lý) sử dụng tổ hợp các bộ đếm và cổng logic để thực hiện các ý tưởng được nêu ở chương trình 13.1.

```
; Chương trình con bàn phím gửi mã ASCII đến chân P0.1 khi nhấn 1 phím
; Các chân P1.0 - P1.3 được nối tới các hàng còn P2.0 - P2.3 tới các cột.
     MOV   P2,#0FFH            ;Đặt P2 làm cổng vào
K1:  MOV   P1,#0               ;Tiếp đất tất cả các hàng
     MOV   A,P2                ;Đọc các cột
     ANL   A,00001111B         ;Che các bit không dùng
     CJNE  A,#00001111B,K1     ;Kiểm tra nhả phím
K2:  ACALL DELAY               ;Tạo trễ 20 ms
     MOV   A,P2                ;Kiểm tra phím nhấn
     ANL   A,#00001111B        ;Che các bit không dùng
     CJNE  A,#00001111B,OVER   ;Nếu phím nhấn, chờ xong
     SJMP  K2                  ;Nếu không,kiểmtra lại
OVER:ACALL DELAY               ;Chờ 20 giây
     MOV   A,P2                ;Kiểm tra nhấn xong
     ANL   A,#00001111B        ;Che các bit không dùng
     CJNE  A,#00001111B,OVER1  ;Nếu có phím nhấn, tìm hàng
     SJMP  K2                  ;Nếu không thì giám sát
OVER1:MOV  P1,#11111110B       ;Tiếp đất các hàng
```

```
     MOV   A,P2                  ;Đọc tất cả các cột
     ANL   A,#00001111B          ;Che các bit không dùng
     CJNE  A,#00001111B,ROW-0 ;Hàng 0, tìm cột
     MOV   P1,#11111101B         ;Tiếp đất hàng 1
     MOV   A,P2                  ;Đọc tất cả các cột
     ANL   A,#00001111B          ;Che các bit không dùng
     CJNE  A,#00001111B,ROW-1 ;Hàng 1, tìm cột
     MOV   P1,#11111011B         ;Tiếp đất hàng 2
     MOV   A,P2                  ;Đọc tất cả các cột
     ANL   A,#00001111B          ;Che các bit không dùng
     CJNE  A,#00001111B,ROW-2 ;Hàng 2, tìm cột
     MOV   P1,#111110111B        ;Tiếp đất hàng 3
     MOV   A,P2                  ;Đọc tất cả các cột
     ANL   A,#00001111B          ;Che các bit không dùng
     CJNE  A,#00001111B,ROW-3 ;Hàng 3, tìm cột
     LJMP  K2              ;Nếu không,vào sai, lặp lại
ROW-0:MOV  DPTR,#KCODE0  ;Đặt DPTR khởi động từ hàng 0
     SJMP  FIND            ;Tìm cột có phím nhấn
ROW-1:MOV  DPTR,#KCODE1  ;Đặt DPTR=khởi động từ hàng 1
     SJMP  FIND            ;Tìm cột có phím nhấn
ROW-2:MOV  DPTR,#KCODE2  ;Đặt DPTR=khởi động từ hàng 2
     SJMP  FIND            ;Tìm cột có phím nhấn
ROW-3:MOV  DPTR,#KCODE3  ;Đặt DPTR=khởi động từ hàng 3
FIND :RRC  A               ;Kiểm tra cờ CY có ở mức thấp
     JNC   MATCH           ;Nếu bằng 0, lấy mã ASCII
     INC   DPTR            ;Trỏ đến cột tiếp theo
     SJMP  FIND            ;Tiếp tục tìm
MATCH:CLR  A               ;Xoá A=0 (Tìm đươc)
     MOVC  A,@A+DPTR       ;Xác định mã ASCII từ bảng
     MOV   P0,A            ;Hiển thị phím nhấn
     LJMP  K1
;Bảng tham chiếu mã ASCII theo các hàng
     ORG   300H
KCODE0:    DB'0','1','2','3'  ;Hàng 0
```

```
KCODE1:     DB'4','5','6','7'   ;Hàng 1
KCODE2:     DB'8','9','A','B'   ;Hàng 2
KCODE3:     DB'C','D','E','F'   ;Hàng 3
     END
```

13.3 NỐI GHÉP DAC VỚI 8051

Bộ biến đổi số - tương tự DAC

Bộ biến đổi số - tương tự DAC là thiết bị được sử dụng rộng rãi để chuyển đổi xung số về tín hiệu tương tự. Trong phần này chúng ta làm quen với cách nối ghép bộ DAC với 8051.

Có hai phương pháp thực hiện chuyển đổi DAC: Phương phát trọng số nhị phân và phương pháp bậc thang R/2R. Rất nhiều mạch tích hợp DAC, trong đó có MC1408 (DAC808) được sử dụng trong phần này, đều sử dụng phương pháp hình thang R/2R vì phương pháp này cho phép đạt được độ chính xác cao hơn. Tiêu chuẩn để đánh giá một bộ DAC trước hết là độ phân giải. Độ phân giải là hàm của số đầu vào nhị phân. Độ phân giải chung thường 8, 10 và 12 bit. Số bit dữ liệu đầu vào quyết định độ phân giải của bộ DAC, vì số mức đầu ra tương tự bằng 2^n với n là số bit dữ liệu đầu vào. Do vậy, một bộ DAC 8 bit như DAC808 chẳng hạn có 256 mức điện áp (dòng điện) rời rạc ở đầu ra. Tương tự như vậy, một bộ DAC 12 bit cho 4096 mức điện áp rời rạc. Cũng có các bộ DAC 16 bit song chúng rất đắt.

Bộ biến đổi DAC MC1408 (hay DAC808)

Ở DAC808 tín hiệu đầu vào số được chuyển đổi thành dòng (I_{out}) và nếu nối điện trở tới chân I_{out} thì kết quả được chuyển thành điện áp. Dòng tổng được cấp bởi chân I_{out} là một hàm số nhị phân của các đầu vào D0 - D7 của DAC808 và được tính theo I_{ref} như sau:

$$I_{out} = I_{ref}(\frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256})$$

trong đó D0 là bit thấp LSB và D7 là bit cao MSB, dòng đầu vào I_{ref} phải được áp vào chân 14. Dòng I_{ref} thường đặt giá trị 2.0 mA. Hình 13.7 giới thiệu mạch tạo tham chiếu dòng (thiết lập I_{ref}= 2 mA) bằng cách sử dụng điện áp nuôi 5 V và các điện trở 1KΩ, 1,5 KΩ. Nếu I_{ref}=2 mA, còn tất cả đầu vào nối tới DAC ở mức cao, thì dòng điện cực đại ở đầu ra là 1,99 mA (mời Bạn tự kiểm tra).

Chuyển I_{out} sang điện áp ở DAC808

Nếu nối điện trở tới chân I_{out} thì dòng được chuyển thành điện áp và có thể kiểm tra đầu ra bằng máy hiện sóng. Tuy nhiên, như vậy sẽ làm giảm độ chính xác do bị thay đổi trở kháng vào của tải. Vì vậy, dòng ra I_{ref} cần được cách ly bằng cách dùng khuyếch đại thuật toán, ví dụ như 741 với điện trở hồi tiếp Rf=5 kΩ. Nếu R=5 kΩ thì khi thay đổi đầu vào nhị phân, điện áp đầu ra sẽ thay đổi như ở ví dụ 13.4.

Ví dụ 13.4

Cho R=5KΩ , I = 2mA, hãy tính V_{OUT} khi đầu vào là các mã nhị phân sau:

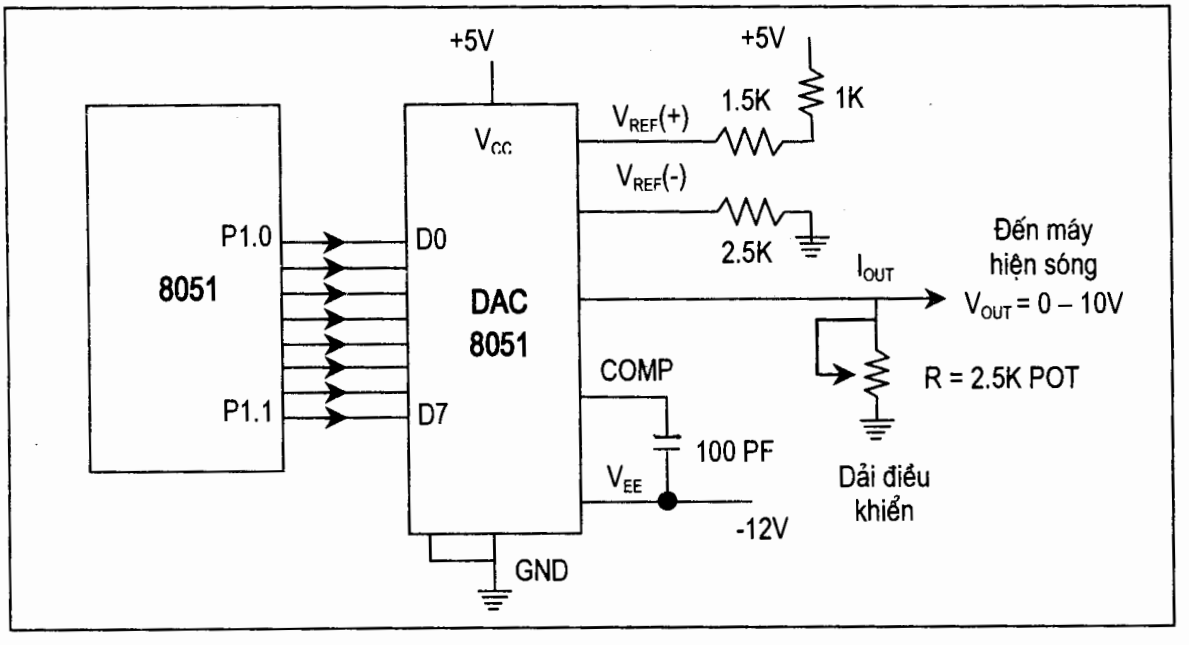
a) 10011001 (99H)

b) 11001000 (C8H)

Giải:

a) I_{OUT} = 2mA(153/225) = 1.195mA và V_{OUT} = 1.195mA × 5K = 5.975V

b) I_{OUT} = 2mA(200/256) = 1.562mA và V_{OUT} = 1.562mA × 5K = 7.81255V



Hình 13.7. Nối ghép 8051 với DAC808

Ví dụ 13.5

Để tạo ra một điện áp dạng bậc thang, thử nối mạch như ở hình 13.7 và nối máy hiện sóng tới đầu ra. Hãy viết chương trình gửi dữ liệu tới bộ DAC để tạo ra sóng hình bậc thang.

Giải:

```
        CLR     A
AGAIN:  MOV     P1, A           ;Gửi dữ liệu đến DAC
        INC     A
        ACALL   DELAY           ;Khôi phục lại tín hiệu
        SJMP    AGAIN
```

Ví dụ 13.6

Hãy kiểm tra các giá trị cho bởi các góc sau: a) 30; b) 60

Giải:

a) VOUT = 5V + (5V × sinθ) = 5V + 5 × 0.5 = 7.5V

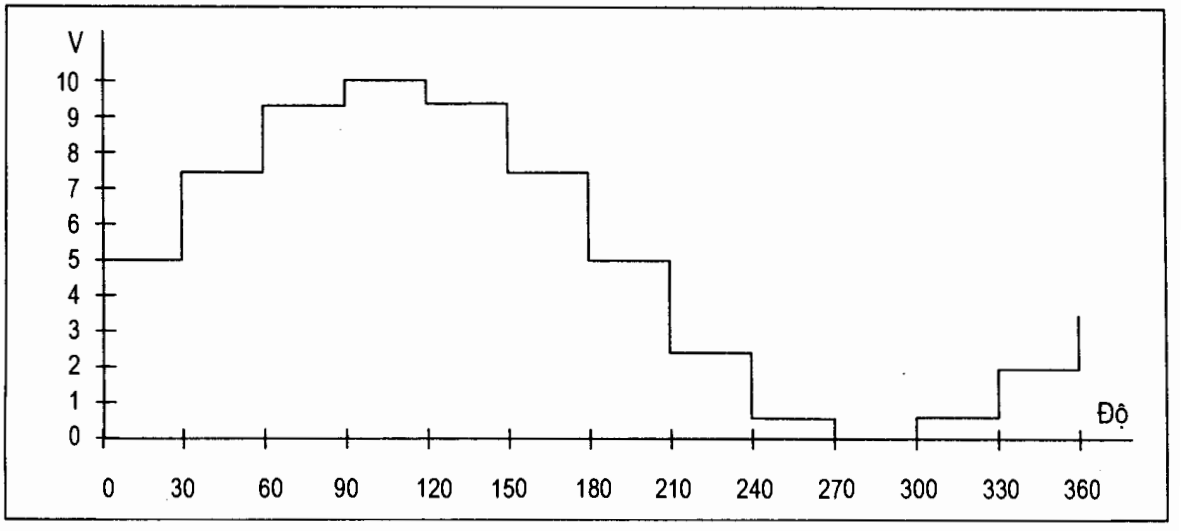
DAC=7.5V×25.6=192.

b) VOUT = 5V+5V×sinθ) = 5V+5×sin60 = 5V+5×0.866 = 9.33V

DAC = 9.33V × 25.6 = 238.

Bảng 13-5. Quan hệ góc và điện áp của sóng hình sin

Góc θ (độ)	Sin θ	Biên độ điện áp Vout 5V + (5V × sinθ)	Giá trị gửi tới DAC (biên độ điện áp ×25.6)
0	0	5	128
30	0.5	7.5	192
60	0.866	9.33	238
90	1.0	10	255
120	0.866	9.33	238
150	0.5	7.5	192
180	0	5	128
210	-0.5	2.5	64
240	-0.866	0.669	17
270	-1.0	0	0
300	-0.866	0.669	17
330	-0.5	2.5	64
360	0	5	128



Hình 13.8. Quan hệ điện áp-góc sóng hình sin

Tạo sóng hình sin

Để tạo sóng hình sin, đầu tiên chúng ta thiết lập bảng giá trị biên độ hình sin theo góc từ 0 đến 360^0. Trong khoảng này, biên độ hình sin thay đổi trong khoảng -1, +1. Vì vậy, bảng giá trị là các số nguyên biểu diễn biên độ điện áp theo góc theta. Phương pháp này bảo đảm rằng, chỉ có các số nguyên là tín hiệu ra của DAC được nối tới 8051. Bảng 13.5 biểu diến quan hệ góc với giá trị biên độ hình sin, biên độ điện áp và các số nguyên tương ứng. Để tạo ra bảng 13.5, chúng ta giả thiết rằng điện áp đầy thang là 10 V ở đầu ra DAC. Tín hiệu ra của DAC sẽ lớn nhất (hết thang) nếu tất cả dữ liệu đầu vào của DAC ở mức cao. Vì vậy, để được tín hiệu ra đầy thang, chúng ta sử dụng phương trình sau:

$$V_{out} = 5\ V + (5x \sin \theta)$$

Để tìm được giá trị gửi đến DAC với các giá trị góc khác nhau, chúng ta chỉ cần đơn giản là nhân điện áp ra V_{out} với 256. Để rõ hơn, hãy xem đoạn chương trình sau. Chương trình này gửi các giá trị liên tục tới DAC để tạo ra sóng hình sin dưới dạng thô. Xem hình 13.8.

```
AGAIN: MOV   DPTR,#TABLE
       MOV   R2,#COUNT
BACK:  CLR   A
```

```
          MOVC A,@A+DPTR
          MOV  P1,A
          INC  DPTR
          DJNZ R2,BACK
          SJMP AGAIN
          ORG  300
   TABLE: DB   128,192,238,255,238,192
          DB   128,64,17,0,17,64,128
;Bạn có thể tham khảo thêm ở bảng 13.5 để biết giá trị cụ
;thể của sóng hình sin
```

Chương 14
NỐI PHÉP 8031/51 VỚI BỘ NHỚ NGOÀI

14.1 BỘ NHỚ BÁN DẪN

Trước hết chúng ta nhắc lại đôi điều về bộ nhớ bán dẫn và một số đặc tính của chúng.

Bộ nhớ bán dẫn dùng để lưu giữ mã lệnh và dữ liệu. Bộ nhớ được ghép nối trực tiếp với CPU và là nơi đầu tiên được CPU lấy thông tin. Yêu cầu đặt ra cho bộ nhớ là phải đủ nhanh để đáp ứng kịp thời các đòi hỏi của CPU (và chỉ có các bộ nhớ bán dẫn mới đáp ứng được). Các bộ nhớ bán dẫn thông dụng là ROM và RAM. Sau đây chúng ta sẽ tìm hiểu một số khái niệm quan trọng nhất của các bộ nhớ bán dẫn.

Dung lượng nhớ

Số lượng bit mà chip nhớ bán dẫn có thể dùng để lưu dữ liệu, được gọi là *dung lượng chip nhớ*. Đơn vị dung lượng chip nhớ thường dùng là kbit (kilobit), Mbit (megabit),... Cần phân biệt khái niệm này với dung lượng bộ nhớ của máy tính. Dung lượng chip nhớ luôn được tính bằng **bit,** còn dung lượng bộ nhớ của máy tính được tính bằng **byte**. Chẳng hạn, trong các tạp chí kỹ thuật nói về một vi mạch nhớ có dung lượng là 16M thì cần hiểu đó là dung lượng 16M bit, còn nói máy tính có bộ nhớ 16M thì phải hiểu là 16M byte.

Tổ chức bộ nhớ

Bộ nhớ được tổ chức theo các ô nhớ trong vi mạch nhớ. Mỗi ô nhớ có thể chứa 4 bit, 8 bit hoặc 16 bit tuỳ theo thiết kế. Số lượng bit trong mỗi ô nhớ của chip luôn bằng số chân dữ liệu của chip nhớ đó, còn số ô nhớ lại phụ thuộc số chân địa chỉ và bằng 2 luỹ thừa của số chân địa chỉ. Do đó, tổng các bit mà IC nhớ có thể lưu giữ sẽ bằng số các ô nhớ nhân với số bit trong mỗi ô. Tóm lại là:

1. Mỗi chip nhớ có 2^x ô nhớ, trong đó x là số chân địa chỉ của chip.
2. Mỗi ô nhớ có y bit, trong đó y là số chân dữ liệu trên chip.

Toàn bộ chip sẽ chứa $2^x \times y$ bit, trong đó x là số chân địa chỉ và y là số chân dữ liệu trên chip.

Tốc độ

Một trong những thông số quan trọng nhất của vi mạch nhớ là tốc độ truy nhập dữ liệu. Để truy nhập dữ liệu, trước tiên địa chỉ cần được chuyển tới các chân địa chỉ và sau một khoảng thời gian dữ liệu mới được áp đến các chân dữ liệu. Khoảng thời gian này càng ngắn càng tốt và tất nhiên chip nhớ sẽ càng đắt. Tốc độ của chip nhớ thường được gọi là *thời gian truy nhập*. Thời gian truy nhập thường từ vài ns đến hàng trăm ns phụ thuộc vào công nghệ chế tạo IC.

Cả ba thông số của bộ nhớ là dung lượng, tổ chức bộ nhớ và thời gian truy cập đều là những thông số quan trọng và được dùng nhiều trong thực tế. Bảng 14.1 trình bày mối quan hệ giữa số chân địa chỉ và dung lượng nhớ. Ví dụ 14.1 và 14.2 sẽ làm rõ thêm một số khái niệm vừa nêu.

Bảng 14.1. Xác định dung lượng nhớ

Số địa chỉ : x	Dung lượng 2^x
10	1K
11	2K
12	4K
13	8K
14	16K
15	32K
16	64K
17	128K
18	256K
19	512K
20	1M
21	2M
22	4M
23	8M
24	16M

Ví dụ 14.1

Một chip nhớ có 12 chân địa chỉ và 4 chân dữ liệu. Hãy xác định:
a) Tổ chức bộ nhớ; b) Dung lượng

Giải:

a) Chip nhớ trên có $2^{12} = 4096$ ô nhớ và mỗi ô nhớ có 4 bit dữ liệu. Như vậy, tổ chức bộ nhớ là 4096 x4, thường được ghi là 4Kx4.
b) Dung lượng nhớ là 16 K bit vì có tổng cộng có 4K ô nhớ, mỗi ô nhớ gồm 4 bit dữ liệu.

Ví dụ 14.2

Vi mạch nhớ 512 K có 8 chân dữ liệu. Hãy xác định:
a) Tổ chức bộ nhớ;
b Số chân địa chỉ

Giải:

a) Vi mạch nhớ có 8 chân dữ liệu nghĩa là mỗi ô nhớ có 8 bit. Để tìm số ô nhớ, ta lấy dung lượng nhớ chia cho số chân dữ liệu: 512K/8 = 64K; như vậy tổ chức bộ nhớ này là 64Kx8.
b) Do 64 K=2^{16} nên vi mạch có 16 chân địa chỉ.

Bộ nhớ ROM

ROM là một kiểu bộ nhớ không bị mất nội dung khi tắt nguồn. Vì lý do đó, ROM còn được gọi là *bộ nhớ không thay đổi* (nonvolatile memory). Có nhiều kiểu ROM khác nhau như: PROM, EPROM, EEPROM, Flash EPROM và ROM che mặt nạ.

Bộ nhớ PROM

PROM là bộ nhớ mà người sử dụng có thể lập trình được. Mỗi bit của PROM có một cầu chì. PROM được lập trình bằng cách đốt cháy các cầu chì. Nếu thông tin đốt cầu chì của PROM bị sai thì PROM đó không dùng được nữa vì cầu chì đã bị đứt không thể khôi phục lại được. Do đó, PROM còn được gọi là bộ nhớ lập trình một lần OTP (one-time programmable). Quá trình lập trình cho ROM còn được gọi là đốt (hay nạp) ROM và cần có một thiết bị chuyên dùng gọi là bộ nạp ROM hay bộ lập trình ROM.

Bộ nhớ EPROM và UV - EPROM

EPROM là loại PROM có thể thay đổi nội dung bằng cách xoá đi và nạp lại có thể tới hàng ngàn lần. Thời gian xoá EPROM cần khoảng 20 phút. Tất cả các EPROM đều có cửa sổ xoá để chiếu tia cực tím. Vì lý do này nên EPROM còn được gọi là UV-EPROM (Ultraviolet). Hình 14.1 trình bày các chân của một chip UV - EPROM.

Để lập trình cho một UV - EPROM cần thực hiện các bước sau:

1. Xoá nội dung cũ. Cần lấy EPROM ra khỏi đế cắm trên bảng mạch và đặt vào thiết bị xoá, chiếu tia cực tím qua cửa sổ của EPROM trong khoảng 15-20 phút.
2. Lập trình chip nhớ. Để lập trình, trước hết cần đặt chip UV-EPROM vào bộ nạp (hay bộ lập trình) ROM. Tuỳ theo loại EPROM, cần sử dụng điện áp 12,5V hoặc cao hơn để đốt EPROM. Điện áp này được đặt vào chân VPP.
3. Sau khi nạp EPROM xong, cắm trả chip nhớ vào đế cắm trên bảng mạch.

Từ những bước trên có thể thấy, để lập trình cho EPROM cần phải có thiết bị xoá và thiết bị nạp riêng biệt. Nhược điểm chủ yếu của tất cả các kiểu bộ nhớ UV-EPROM là không thể lập trình trực tiếp khi chúng đang nằm trên bảng mạch chủ. Để khắc phục nhược điểm này, người ta chế tạo bộ nhớ khác, gọi là EEPROM.

Cần lưu ý về ký hiệu của IC trong bảng 14.2 như sau: Ký hiệu phần mã số 27128-25 để chỉ UV - EPROM có dung lượng 128 K bit và thời gian truy cập là

250ns. Dung lượng của chip nhớ được ký hiệu trên phần mã số, còn thời gian truy cập được bỏ đi một số 0. Ở phần mã số thì chữ C để chỉ công nghệ CMOS, còn 27xx dùng để chỉ chip nhớ EPROM. Bạn có thể tham khảo thêm catalog của các hãng như JAMECO (jameco.com) hay JDR (jdr.com) có trên internet.

Bảng 14.2. Một số chip nhớ UV-EPROM

Phần số	Dung lượng	Tổ chức	Thời gian	Số chân	Vpp
2716	16K	2K × 8	450 ns	24	25 V
2732	32K	4K × 8	450 ns	24	25 V
2732A-20	32K	4K × 8	200 ns	24	21 V
27C32-1	32K	4K × 8	450 ns	24	12.5 V CMOS
2764-20	64K	8K × 8	200 ns	28	21 V
2764A-20	64K	8K × 8	200 ns	28	12.5 V
27C64-12	64K	8K × 8	120 ns	28	12.5 V CMOS
27128-25	128K	16K × 8	250 ns	28	21 V
27C128-12	128K	16K × 8	150 ns	28	12.5 V CMOS
27256-25	256K	32K × 8	250 ns	28	12.5 V
27C256-15	256K	32K × 8	150 ns	28	12.5 V CMOS
27512-25	512K	64K × 8	250 ns	28	12.5 V
27C512-15	512K	64K × 8	150 ns	28	12.5 V CMOS
27C010-15	1024K	128K × 8	150 ns	32	12.5 V CMOS
27C020-15	2048K	256K × 8	150 ns	32	12.5 V CMOS
27C040-15	4096K	512K × 8	150 ns	32	12.5 V CMOS

Ví dụ 14.3

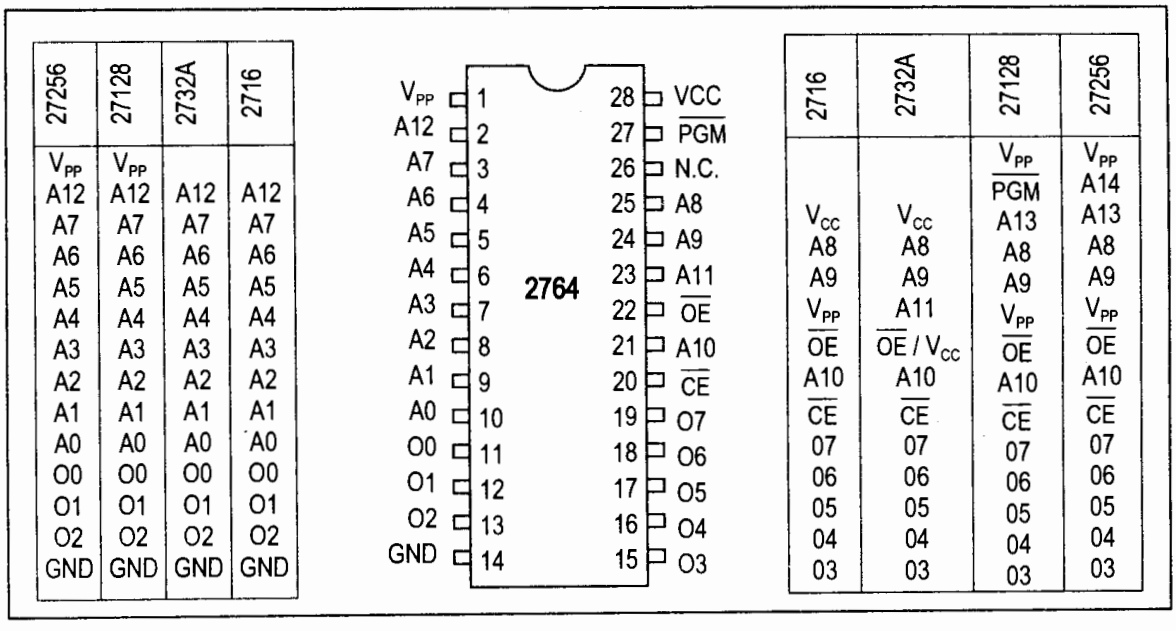
Hãy xác định số chân địa chỉ và số chân dữ liệu của ROM có mã 27128.

Giải:

ROM 27128 có dung lượng nhớ là 128k bit. Tra bảng ta thấy, chip có tổ chức 16k × 8 (tất cả mọi ROM đều có 8 chân dữ liệu), như vậy có 8 chân dữ liệu và số chân địa chỉ là 14 vì 2^{14} = 16k.

Bộ nhớ PROM có thể xoá được bằng điện EEPROM

So với EPROM, EEPROM có một số ưu điểm như nhờ sử dụng phương pháp xoá điện nên việc xoá được thực hiện gần như tức thì, mà không phải mất đến 20 phút. Ngoài ra, có thể chọn riêng từng byte trong EEPROM để xoá, chứ không phải xoá toàn bộ nội dung như ở EPROM. Tuy nhiên, ưu điểm chính của EEPROM là có thể lập trình và xoá nội dung cho EEPROM ngay trên bảng mạch chính mà không cần phải gỡ vi mạch ra khỏi đế. Như vậy, khác với EPROM, EEPROM không cần thiết bị nạp và xoá bên ngoài. Để sử dụng EEPROM, cần có mạch lập trình cho EEPROM ngay trên bảng mạch hệ thống với điện áp VPP 12,5 V. Tuy nhiên có thể dùng VPP 5-7 V, song giá thành tính cho mỗi bit lúc đó thường tăng đáng kể so với UV-EPROM.



Hình 14.1. Tổ chức chân của bộ nhớ ROM họ 27XX

Bảng 14.3. Một số chip EEPROM và Flash

EEPROMs					
Ký hiệu	**Dung lượng**	**Tổ chức**	**Tốc độ**	**Số chân**	**Vpp**
2816A-25	16K	2K × 8	250 ns	24	5 V
2864A	64K	8K × 8	250 ns	28	5 V
28C64A-25	64K	8K × 8	250 ns	28	5 V CMOS

28C256-15	256K	32K × 8	150 ns	28	5 V
28C256-25	256K	32K × 8	250 ns	28	5 V CMOS
Flash					
Ký hiệu	**Dung lượng**	**Tổ chức**	**Tốc độ**	**Số chân**	**Vpp**
28F256-20	256K	32K × 8	200 ns	32	12 V CMOS
28F010-15	1024K	128K × 8	150 ns	32	12 V CMOS
28F020-15	2048K	256K × 8	150 ns	32	12 V CMOS

Bộ nhớ FLASH EPROM

Từ đầu những năm 90 của thế kỷ XX, flash EPROM đã trở thành vi mạch nhớ lập trình được sử dụng khá phổ biến với một số ưu điểm: trước hết, thời gian xoá toàn bộ nội dung chưa đến một giây, do đó nó được gọi là bộ nhớ cực nhanh hay bộ nhớ chớp nhoáng (flash memory). Ngoài ra, đây là phương pháp xoá bằng điện nên đôi khi còn được gọi là flash EEPROM. Để tránh nhầm lẫn, người ta thường gọi đó là bộ nhớ flash. Điểm khá c biệt chính giữa EEPROM và bộ nhớ flash là khi xoá bộ nhớ flash thì toàn bộ nội dung sẽ bị xoá, còn đối với EEPROM có thể xoá từng byte hoặc từng phần. Tuy vậy, một số bộ nhớ flash được chế tạo gần đây có tổ chức bộ nhớ theo khối, lúc đó có thể tiến hành xoá theo từng khối, song vẫn không thể xoá được theo từng byte như ở EEPROM. Do các bộ nhớ flash có thể được lập trình ngay trên bảng mạch hệ thống nên nó được dùng rộng rãi khi nâng cấp các ROM BIOS của PC. Một số nhà thiết kế còn tin tưởng rằng bộ nhớ flash sẽ thay thế cả đĩa cứng - hiện đang là công cụ lưu trữ chính của PC. Nếu được như vậy, hiệu năng của máy tính sẽ được tăng lên rất lớn do bộ nhớ flash là bộ nhớ bán dẫn có thời gian truy nhập chỉ cỡ 100 ns trong khi thời gian truy nhập của đĩa cứng là hàng chục ms. Để khả năng này có thể trở thành hiện thực, trước hết chu kỳ lập/xoá của bộ nhớ flash - cũng giống như đĩa cứng - phải tiến tới vô cực. *Chu kỳ lập/xoá* là số lần chip nhớ có thể lập trình và xoá trước khi bị hỏng. Hiện nay, chu kỳ lập trình/xoá của bộ nhớ flash mới khoảng 10.000, của UV-EPROM là 1000 còn của RAM và đĩa là vô tận.

ROM mặt nạ (Mask ROM)

ROM mặt nạ là dạng ROM được lập trình không phải do người dùng mà do nhà sản xuất IC. Thuật ngữ *mặt nạ* được sử dụng trong quá trình sản xuất IC. Đây là một quá trình tốn kém, vì vậy ROM mặt nạ chỉ được sử dụng khi cần số lượng

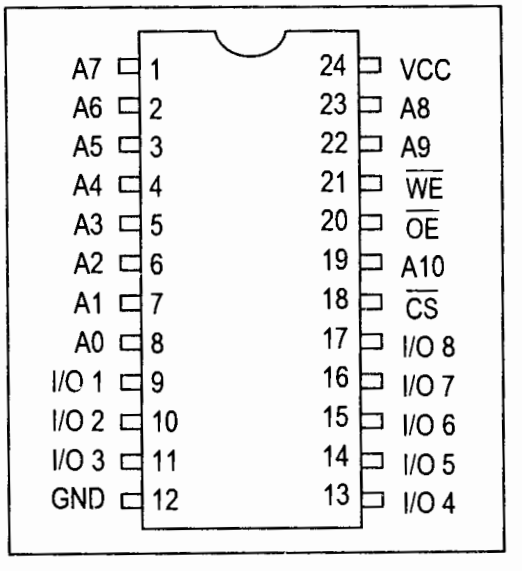
lớn và khi đã chắc chắn rằng nội dung của ROM sẽ không thay đổi nữa. Thông thường, UV-PROM được sử dụng trong giai đoạn đầu phát triển chương trình, còn sau khi mã lệnh/dữ liệu đã được hoàn thiện, lúc đó mới đặt hàng ROM mặt nạ. Ưu điểm cơ bản của ROM mặt nạ là có tổng chi phí thấp hơn các loại ROM khác, song nếu có lỗi trong chương trình thì toàn bộ lô ROM đó phải huỷ bỏ. Nhiều nhà sản xuất 8051 hỗ trợ phiên bản ROM che mặt nạ của 8051. Lưu ý là tất cả bộ nhớ ROM đều 8 bit chân dữ liệu, vì thế nó có tổ chức x8.

Bộ nhớ truy nhập ngẫu nhiên RAM (random access memory)

Bộ nhớ RAM còn được gọi là *bộ nhớ thay đổi* (volatile memory) vì nếu tắt nguồn nuôi thì dữ liệu cũng bị mất. Đôi khi, RAM còn được gọi là bộ nhớ ghi đọc RAWM (read and write memory), khác với ROM chỉ đọc mà không ghi được. Có ba kiểu RAM: RAM tĩnh (SRAM), RAM động (DRAM) và NV-RAM (RAM không thay đổi).

RAM tĩnh (SRAM - static RAM)

Các ô nhớ SRAM được chế tạo dưới dạng mạch lật (flip-flop), do đó không cần phải làm tươi dữ liệu. Đây là điểm khác với DRAM (sẽ đề cập ở phần sau). Một vấn đề khi dùng mạch lật xây dựng bộ nhớ là, để có 1 bit nhớ có khi cần dùng đến 6 transistor. Những năm sau này, số transistor/bit nhớ đã giảm xuống đến 4, nhưng như thế vẫn là quá nhiều. Với khả năng sử dụng 4 transistor/bit nhớ và với công nghệ CMOS đã cho ra đời bộ nhớ SRAM dung lượng cao, tuy vậy vẫn còn thấp hơn nhiều so với DRAM. Bảng 14.4 giới thiệu một số SRAM. Trong các máy tính PC, SRAM thường được dùng làm bộ nhớ cache. Hình 14.2 mô tả sơ đồ chân của chip SRAM.



Hình 14.2. Bố trí chân SRAM 2Kx8

Bảng 14.4. Một số chip SRAM và NV-RAM

SRAM					
Ký hiệu	**Dung lượng**	**Tổ chức**	**Tốc độ**	**Số chân**	**Vpp**
6116P-1	16K	2K × 8	100 ns	24	CMOS
6116P-2	16K	2K × 8	120 ns	24	CMOS
6116P-3	16K	2K × 8	150 ns	24	CMOS
6116LP-1	16K	2K × 8	100 ns	24	CMOS công suất nhỏ
6116LP-2	16K	2K × 8	120 ns	24	CMOS công suất nhỏ
6116LP-3	16K	2K × 8	150 ns	24	CMOS công suất nhỏ
6264P-10	64K	8K × 8	100 ns	28	CMOS
6264P-70	64K	8K × 8	70 ns	28	CMOS công suất nhỏ
6264P-12	64K	8K × 8	120 ns	28	CMOS công suất nhỏ
62256LP-10	256K	32K×8	100 ns	28	CMOS công suất nhỏ
62256LP-12	256K	32K×8	120 ns	28	CMOS công suất nhỏ
NV-RAM của hãng Dallas Semiconductor					
Ký hiệu	**Dung lượng**	**Tổ chức**	**Tốc độ**	**Số chân**	**Vpp**
DS1220Y-150	16K	2K × 8	150 ns	24	
DS1225AB-150	64K	8K × 8	150 ns	28	
DS1230Y-85	256K	32K×8	85 ns	28	

Bộ nhớ RAM không thay đổi NV-RAM (Nonvolatile RAM)

Cả hai loại DRAM và SRAM đều là bộ nhớ có nội dung thay đổi được, song vẫn có một loại RAM khác gọi là bộ nhớ RAM không thay đổi - viết tắt là NV-RAM. Cũng giống như RAM, bạn có thể thực hiện ghi/đọc lên NV-RAM, song điểm trội hơn là nội dung không bị mất khi mất nguồn - tương tự như ROM. NV-RAM đã kết hợp những ưu điểm quan trọng của RAM và ROM là: khả năng đọc/ghi của RAM và không bị mất nội dung của ROM. Để giữ được nội dung không bị mất, bên trong NV-RAM có cấu trúc như sau:

1. Sử dụng công nghệ CMOS để bộ nhớ đạt được hiệu suất sử dụng năng lượng cực cao (tiêu thụ năng lượng cực thấp).

2. Cài bên trong nguồn pin litium.

3. Sử dụng mạch điều khiển thông minh. Công việc chính của mạch này là thường xuyên kiểm soát điện áp chân V_{cc} để phát hiện trường hợp mất nguồn bên ngoài. Nếu nguồn ngoài giảm xuống dưới mức cho phép, mạch điều khiển tự động chuyển sang dùng pin lithium bên trong. Như vậy, chỉ khi mất điện ngoài, nguồn pin lithium mới được sử dụng và duy trì nội dung của NV-RAM.

Tất nhiên, cả 3 thành phần trên đều được tích hợp vào trong một chip, vì vậy NV-RAM có giá thành cao hơn nhiều so với các RAM thông thường. Bù lại, nó có khả năng lưu giữ được nội dung đến mười năm mà không cần nguồn điện ngoài và cho phép đọc, ghi hệt như SRAM. Bảng14.4 giới thiệu một số NV-RAM của hãng Dallas Semiconductor.

Ví dụ 14.4

Giả sử có 4 byte dữ liệu dạng hexa như sau: 25H, 62H, 3FH, 52H.

a) Hãy tìm byte kiểm tổng.

b) Thực hiện kiểm tổng để khẳng định tính toàn vẹn của dữ liệu.

c) Nếu byte thứ hai 62H bị đổi thành 22H thì hãy trình bày cách phát hiện ra lỗi này.

Giải:

a) Trình bày byte kiểm tổng

```
   25H
+  62H
+  3FH
+  52H
  118H
```

Byte kiểm tổng được tìm bằng cách trước hết cộng tất cả các byte lại với nhau. Tổng là 118H và bỏ phần nhớ ta còn lại 18H. Lấy bù 2 của 18H là E8H. Đây chính là byte kiểm tổng.

b) Thực hiện thao tác kiểm tổng để khẳng định tính toàn vẹn của dữ liệu:

```
   25H
+  62H
+  3FH
+  52H
+  E8H
  200H
```

Cộng các byte dữ liệu và byte kiểm tổng lại với nhau và kết quả phải bằng 0. Điều này chứng tỏ rằng các byte dữ liệu không bị thay đổi và tính toàn vẹn của dữ liệu được bảo đảm (bỏ phần nhớ, còn lại là 00H).

c) Nếu byte thứ hai 62H bị đổi thành 22H thì cách phát hiện dữ liệu bị thay đổi như sau:

```
    25H
+   22H
+   3FH
+   52H
+   E8H
  1C0H
```

Cộng các byte dữ liệu và byte kiểm tổng lại với nhau và bỏ phần nhớ, còn lại C0H khác 0. Điều đó chứng tỏ một hoặc nhiều byte dữ liệu đã bị thay đổi (sau khi bỏ phần nhớ còn C0H).

RAM động (DRAM - dynamic RAM)

Ở các máy tính thời kỳ đầu, các nhà thiết kế chủ yếu quan tâm đến bộ nhớ đọc/ghi giá thành thấp nhưng dung lượng lớn. Năm 1970, hãng Intel sản xuất bộ nhớ DRAM đầu tiên. Mật độ (dung lượng) ban đầu là 1024 bit và sử dụng tụ điện để nhớ từng bit. Sự có mặt của tụ điện đồng nghĩa với việc giảm số lượng transistor làm mạch lật của các ô nhớ. Tuy nhiên lại nảy sinh nhu cầu phải làm tươi thường xuyên để khắc phục hiện tượng dò điện của tụ điện. Đây là điểm khác với SRAM, với các ô nhớ là các mạch flip-flop. Do mỗi bit của SRAM là một flip-flop và mỗi flip-flop có 6 transistor nên SRAM phải có nhiều transistor hơn và do vậy có dung lượng nhớ thấp hơn. Dùng tụ điện làm ô nhớ trong DRAM làm giảm đáng kể kích thước ô nhớ.

Có thể tóm tắt một số nét đặc trưng chính của DRAM như sau. Ưu điểm chính là dung lượng (mật độ) cao hơn, chi phí trên mỗi bit rẻ hơn và tiêu thụ điện năng trên mỗi bit thấp hơn. Nhược điểm chính là phải thực hiện làm tươi do hiện tượng tự phóng điện của tụ điện; hơn nữa, trong khi đang làm tươi sẽ không thể truy nhập dữ liệu. Đây là những điểm khác với SRAM. SRAM không cần phải làm tươi mà dữ liệu vẫn tồn tại nếu được cấp nguồn và nội dung của SRAM có thể được truy nhập bất cứ lúc nào. Từ năm 1970 đến nay, dung lượng nhớ của DRAM ngày càng tăng nhanh chóng. Sau chip 1K bit (1024) là chip 4K vào năm 1973, rồi 16K năm 1976. Vào những năm 1980 có các chip nhớ 64K, 256K, 1M và cuối cùng là chip 4M. Trong những năm 1990 có các chip nhớ 16M, 64M, 256M và khả năng có chip DRAM 1G bit. Vào thời gian IBM tung ra thị trường máy tính PC, các chip 16K bit được sử dụng rộng rãi và đó là lý do tại sao một số máy tính IBM PC và tương thích vẫn sử dụng các chip 16 K trên bảng mạch chủ. Ngày nay, các bảng mạch chủ sử dụng các chip 1M, 4M, 16M, ... Nên lưu ý là khi nói đến chip nhớ thì dung lượng luôn được tính theo bit. Chip nhớ 1M nghĩa

là 1 megabit còn chip nhớ 4M nghĩa là 4 megabit. Tuy nhiên, khi nói về bộ nhớ hệ thống máy tính thì lại dùng đơn vị byte.

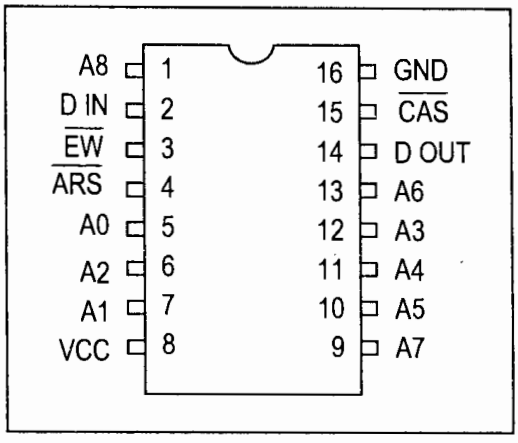
Đóng vỏ DRAM

Các chip DRAM ngày nay có số ô nhớ ngày càng lớn, do đó cần giải quyết vấn đề đóng vỏ và bố trí chân một cách hợp lý. Ví dụ, chip 64K bit (64K×1), theo phương pháp thông thường phải có 16 đường địa chỉ và một đường dữ liệu. Ngoài ra còn thêm chân nguồn, chân đất và chân điều khiển đọc/ghi. Như vậy, là cần có một lượng chân khá lớn, điều này mâu thuẫn với mong muốn IC có dung lượng lớn với kích thước nhỏ. Do đó, để giảm số chân địa chỉ, có thể dùng phương án dồn kênh (multiplexing/demultiplexing). Cách thực hiện là tách địa chỉ làm hai nửa và gửi từng nửa địa chỉ đến các chân tương ứng và như vậy sẽ giảm được số chân đáng kể. Về cấu trúc, DRAM được chia thành ô vuông các hàng và cột. Nửa địa chỉ đầu gọi là *hàng* và nửa sau gọi là *cột*. Ví dụ, với DRAM 64K×1, nửa địa chỉ đầu được gửi qua các chân A0-A7 bằng cách kích hoạt chân RAS (row address strobe), các mạch chốt bên trong DRAM sẽ chốt nửa địa chỉ đầu. Sau đó, nửa địa chỉ thứ hai được gửi đến cũng ở trên các chân đó nhưng lần này kích hoạt chân CAS (column address strobe), các mạch chốt bên trong DRAM sẽ sẽ chốt nửa địa chỉ sau. Như vậy, chỉ cần dùng 8 chân địa chỉ và 2 chân RAS và CAS, tổng cộng là 10 chân thay cho 16 chân ở phương pháp thông thường (không dồn kênh). Để truy nhập 1 bit dữ liệu DRAM, cần phải có địa chỉ về hàng và cột. Để thực hiện theo phương pháp này, cần có bộ dồn kênh địa chỉ bên ngoài DRAM và bộ tách kênh ở mỗi chip DRAM. Do khá phức tạp trong tổ chức truy nhập DRAM (RAS, CAS, mạch dồn kênh và mạch làm tươi) nên người ta sử dụng thêm mạch làm chức năng điều khiển DRAM để việc ghép nối DRAM được dễ dàng hơn. Tuy nhiên, với những hệ thống không đòi hỏi nhiều RAM (thường nhỏ hơn 64 kbyte) thì người ta thường dùng SRAM thay cho DRAM.

Tổ chức DRAM

Chúng ta đã biết, EPROM có 8 chân dữ liệu, còn đối với RAM thì khác. SRAM thường có tổ chức ×4 hoặc ×8, còn DRAM là ×1, ×4, ×8 hoặc ×16. Tuy nhiên, hầu hết DRAM có tổ chức ×1 và ×4. Xem ví dụ 14.5.

Ở một số chip nhớ (thường là ROM), các chân dữ liệu được gọi là I/O. Một số DRAM có các chân D_{in} và D_{out} riêng biệt. Hình 14.3 giới thiệu bố trí chân của chip RAM 256Kx1, trong đó có các chân dữ liệu A0-A8, RAS, CAS, WE (write enable), dữ liệu vào, ra, chân nguồn và chân đất. Hiện nay, công nghệ chế tạo vi mạch phát triển rất mạnh, các chip nhớ thường có mật độ rất cao để tiết kiệm diện tích mạch in. Cũng vì lý do đó, bộ nhớ của các dòng máy tính PC rất khác nhau tuỳ thuộc vào thời điểm sản xuất.



Hình 14.3. DAM 256Kx1

Bảng 14.5. Một số chip DRAM thông dụng

Ký hiệu	Tốc độ	Dung lượng	Tổ chức	Số chân
4164-15	150 ns	64K	64K×1	16
4164-8	80 ns	256K	64K×4	18
41256-15	150 ns	256K	256K×1	16
41256-6	60 ns	256K	256K×1	16
41256-10	100 ns	1 M	256K×1	20
511000P-8	80 ns	1 M	1M×1	18
514100-7	70 ns	4 M	4M×1	20

Ví dụ 14.5

Xác định số chân địa chỉ ở các vi mạch nhớ sau:

a) DRAM 16K × 4; b) SRAM 16K × 4

Giải:

Vì 2^{14} = 16K, nên:

a) Đối với DRAM ta có 7 chân địa chỉ (A0-A6) và 2 chân RAS, CAS.

b) Đối với SRAM ta có 14 chân địa chỉ và không có các chân cho RAS và CAS (vì chúng chỉ có ở DRAM). Trong cả 2 trường hợp đều có 4 chân dữ liệu.

14.2 GIẢI MÃ ĐỊA CHỈ

Như đã biết, CPU cung cấp các địa chỉ dữ liệu, song phần việc tiếp theo lại thuộc về mạch giải mã địa chỉ. Để hiểu rõ hơn về mạch giải mã, trước hết cần nắm được các phương pháp giải mã địa chỉ. Cho đơn giản, trong phần này chúng ta sử dụng SRAM hoặc ROM.

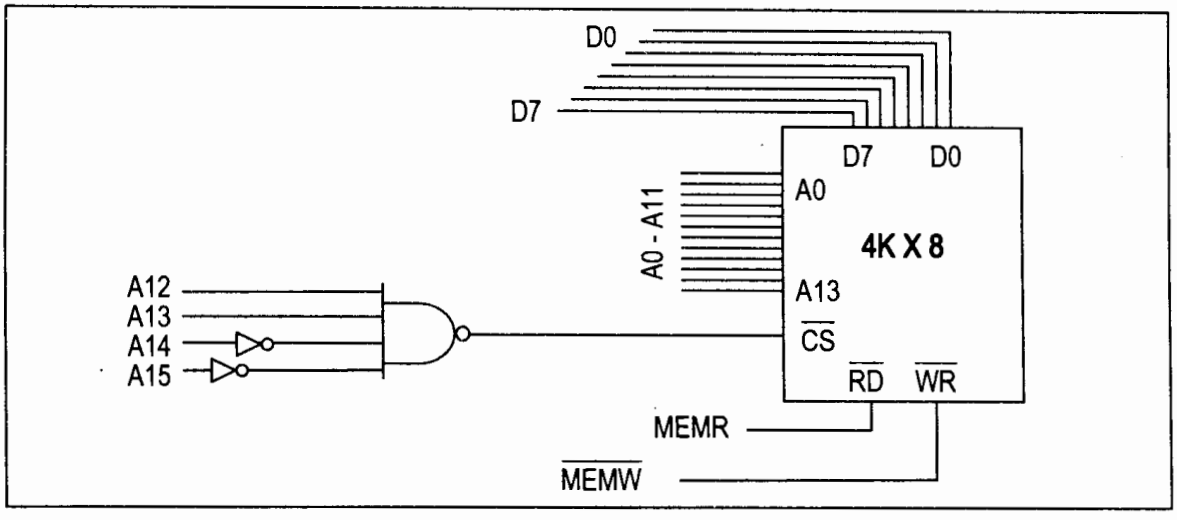
Các chip nhớ luôn có một hoặc nhiều chân gọi là *chọn chip* CS (chip select). Chân này cần được kích hoạt khi truy nhập bộ nhớ. CS còn được gọi là *cho phép mở* CE (chip enable). Khi tổ chức nối ghép, bus dữ liệu của CPU có thể được nối trực tiếp đến các chân dữ liệu của bộ nhớ. Các tín hiệu điều khiển MEMR và MEMW từ CPU được nối tới các chân RD và WR của vi mạch nhớ tương ứng. Đối với bus địa chỉ, các bit địa chỉ thấp được nối đến các chân địa chỉ của chip nhớ, còn các bit địa chỉ cao dùng để kích hoạt chân CS của chip nhớ. Chân CS cùng với RD/WR dùng để cho phép luồng dữ liệu đi vào hay ra khỏi chip nhớ. Nói cách khác, sẽ không có dữ liệu nào có thể ghi/đọc chip nhớ nếu chân CS chưa được kích hoạt. Chân CS thường có mức tích cực thấp và được kích hoạt theo một trong các phương pháp giải mã địa chỉ sau.

Dùng cổng logic đơn giản làm mạch giải mã

Phương pháp đơn giản nhất là dùng cổng NAND làm mạch giải mã. Đầu ra của cổng NAND thường có mức tích cực thấp và đầu vào $\overline{CS}$ cũng mức tích cực thấp, vì vậy chúng phối hợp tốt với nhau. Trong trường hợp đầu vào CS có mức tích cực cao thì cần sử dụng cổng AND. Sử dụng tổ hợp các mạch logic NAND và mạch đảo, ta có thể giải mã bất cứ vùng địa chỉ nào. Như giới thiệu ở hình 14.4, các bit được dùng để chọn chip là A15-A12 = 0011. Như vậy, chip nhớ này được định địa chỉ cho vùng 3000H - 3FFFFH.

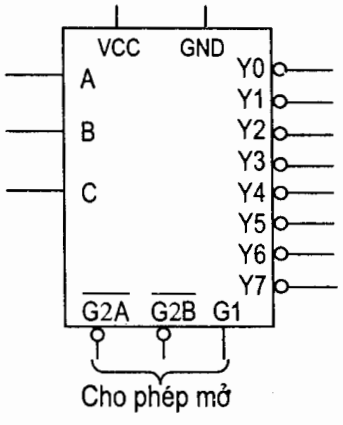
Dùng bộ giải mã 74xx138

74xx138 là một trong những bộ giải mã địa chỉ được sử dụng rộng rãi nhất. Ba chân vào A, B và C dùng để tạo ra 8 tín hiệu ra mức tích cực thấp Y0-Y7. Mỗi đầu ra Y được nối đến chân CS của một chip nhớ, cho phép điều khiển 8 khối nhớ (chỉ cần một chip 74138). Ở 74138, ngoài các chân vào A, B và C để chọn đầu ra thích hợp, còn có 3 thêm đầu vào là $\overline{G2A}$, $\overline{G2B}$ và G1. Trong đó, $\overline{G2A}$ và $\overline{G2B}$ đều là tích cực mức thấp, còn G1 là tích cực mức cao. Nếu một trong ba đầu vào này



Hình 14.4. Cổng logic làm mạch giải mã

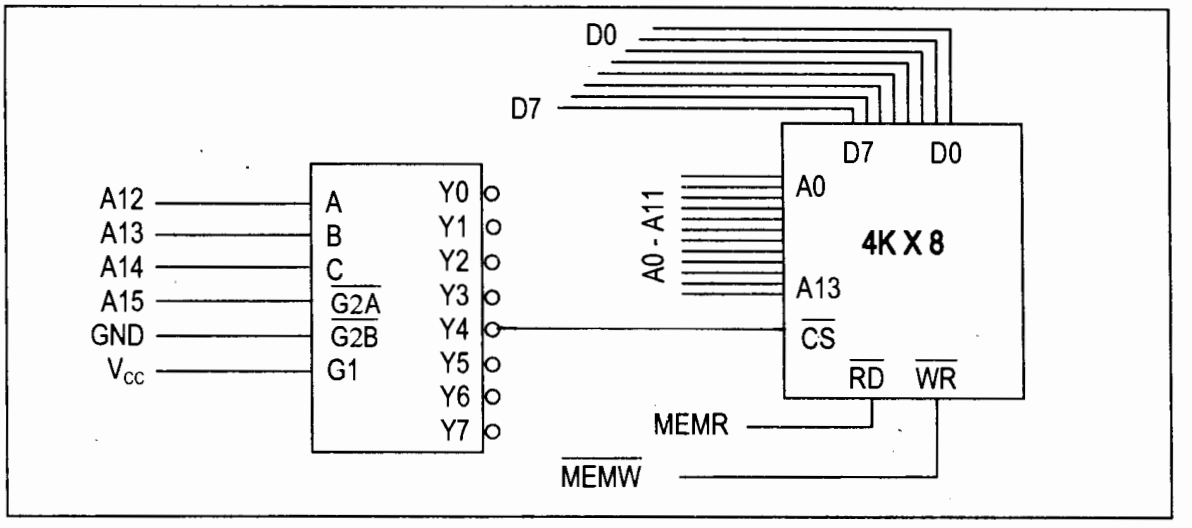
không được nối đến tín hiệu địa chỉ (đôi khi được nối đến tín hiệu điều khiển) thì chúng phải được kích hoạt cố định bằng cách hoặc nối đất hoặc nối nguồn V_{cc} tuỳ theo mức kích hoạt của từng chân. Ví dụ 3.6 trình bày việc thiết kế cũng như cách xác định vùng địa chỉ của bộ giải mã địa chỉ 74138.



Bảng chức năng

Input				Output							
Enable		Select									
G1	G2	C	B	A	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
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

Hình 14.5. Bộ giải mã địa chỉ 74LS138



Hình 14.6. Mạch 74LS138 làm mạch giải mã

Ví dụ 14.6

Dựa theo sơ đồ hình 14.6, hãy xác định vùng địa chỉ cho:
a) Y4; b) Y2; c) Y7

Giải:

a) Vùng địa chỉ cho Y4 được tính như sau:

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Có thể thấy vùng địa chỉ Y4 là F0000H - F3FFFH. Chú ý A19, A18 và A17 trên hình 14.6 phải bằng 1 để bộ giải mã được kích hoạt. Y4 được chọn khi A16 A15 A14 = 100. A13-A0=0 cho địa chỉ thấp nhất và bằng 1 cho địa chỉ cao nhất.

b) Tương tự, vùng địa chỉ cho Y2 là E8000H đến EBFFFH

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

c) Vùng địa chỉ cho Y7 là từ FC000H đến FFFFFH

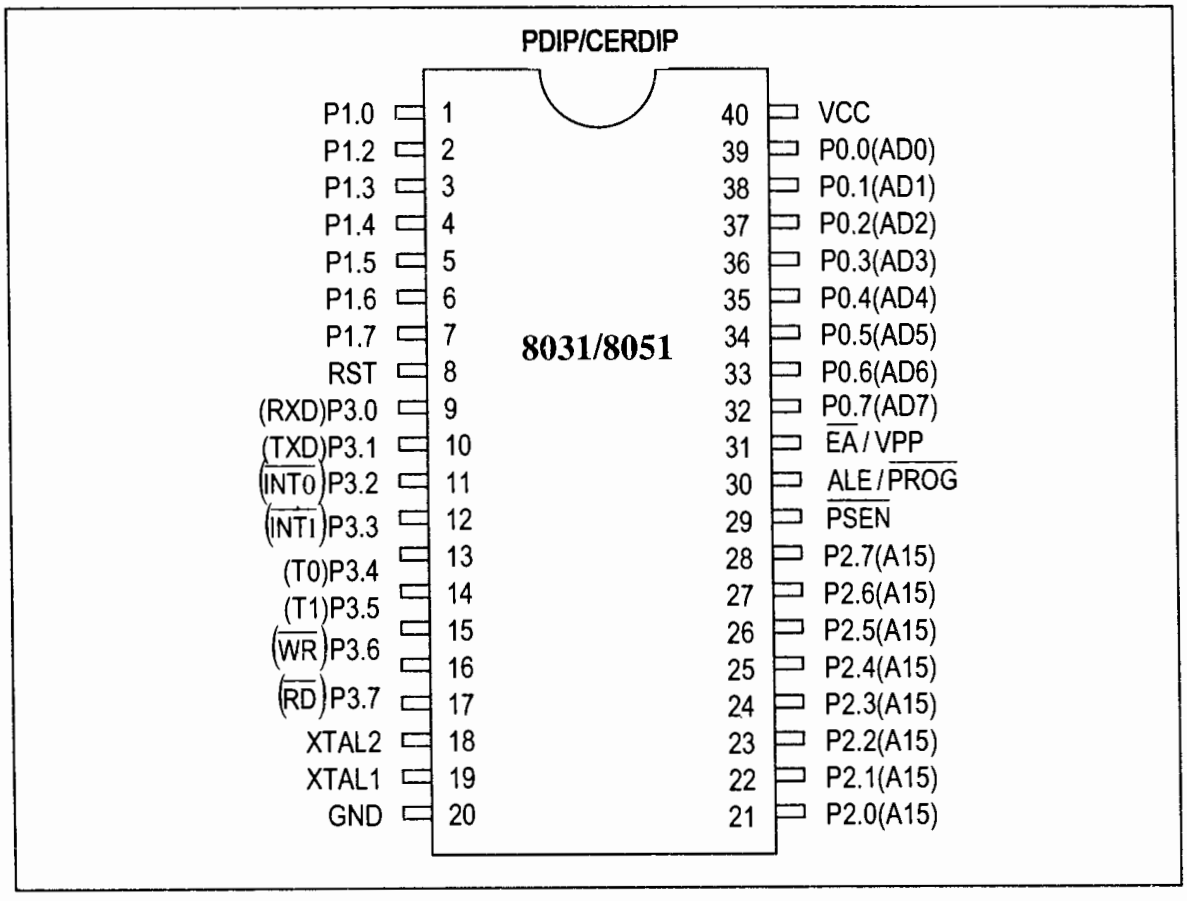
A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Dùng PAL16L8 làm mạch giải mã

Một dạng mạch giải mã được sử dụng rộng rãi khác là chip PAL16L8. Nhược điểm của PAL16L8 là cần phải có thiết bị lập trình PAL và phần mềm PALASM, còn đối với 74138 thì không cần. Ưu điểm của 16L8 là linh hoạt hơn vì nó có thể lập trình để giải mã cho bất kỳ vùng địa chỉ nào. Do 16L8 có 10 chân vào cho bit địa chỉ (74138 chỉ có 6 chân) nên có khả năng giải mã cho nhiều chân địa chỉ hơn.

4.3 GIAO TIẾP CỦA 8031/51 VỚI BỘ NHỚ ROM NGOÀI

Như đã trình bày ở chương I, chip 8031 là phiên bản không có ROM của 8051. Nói cách khác, về mặt tổ chức và thực hiện lệnh, 8031 cũng tương tự với các thành viên của họ 8051, như 8751 hay 89C51, nhưng có điểm khác là không có bộ nhớ ROM.



Hình 14.7. Bố trí chân của 8051

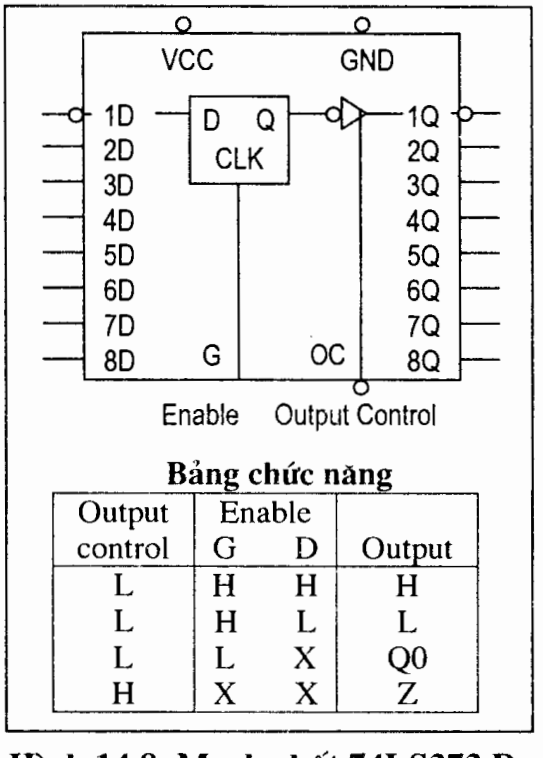
Để 8031 thực hiện lệnh của 8051 thì cần được nối với bộ nhớ ROM ngoài có chứa mã chương trình. Ở mục này, chúng ta sẽ tìm hiểu về nối ghép 8031 với bộ nhớ ROM ngoài. Có thể sẽ có người đặt câu hỏi, tại sao lại phải dùng 8031 như vậy trong khi dễ dàng mua được 8751, 89C51 hay DS5000. Lý do đơn giản là vì những chip này có dung lượng ROM trên chip nhỏ. Do vậy, ở nhiều hệ thống, khi bộ nhớ trên chip của 8051 không đủ thì 8031 là rất thích hợp vì nó cho phép mở rộng kích thước chương trình đến 64 kbyte. Thực tế là chip 8031 rẻ hơn nhiều so với các thành viên khác của họ 8051, song một hệ thống xây dựng trên 8031 thì lại đắt hơn nhiều vì bộ nhớ ROM ngoài nằm ở bên ngoài chip và cần có nhiều mạch điện hỗ trợ để ghép nối như ta sẽ được bàn đến ở dưới đây.

Bố trí một số chân

Chúng ta sẽ tìm hiểu một số chân của 8031/51 dùng để ghép nối với bộ nhớ ngoài, xem hình 14.7.

EA

Như đã trình bày ở chương 4, đối với các hệ thống xây dựng trên 8751/89C51/DS5000 thì chân EA được nối với dương nguồn V_{CC} để báo rằng mã chương trình nằm ở bộ nhớ ROM trên chip. Còn để báo mã chương trình được lưu ở trên bộ nhớ ROM ngoài thì chân này phải được nối xuống đất GND. Đây là trường hợp đối với các hệ thống 8031. Thực tế, nếu số lần đốt và xoá bộ nhớ ROM trên chip quá nhiều thì UV-EPROM không thể hoạt động được nữa. Trong những trường hợp như vậy người ta cũng có thể sử dụng 8751 (hoặc 89C51 hay 8051 bất kỳ) như là 8031 và công việc chúng ta phải làm là nối chân EA xuống đất và nối chip với bộ nhớ ROM ngoài có chứa mã chương trình.



Bảng chức năng

Output control	Enable G	Enable D	Output
L	H	H	H
L	H	L	L
L	L	X	Q0
H	X	X	Z

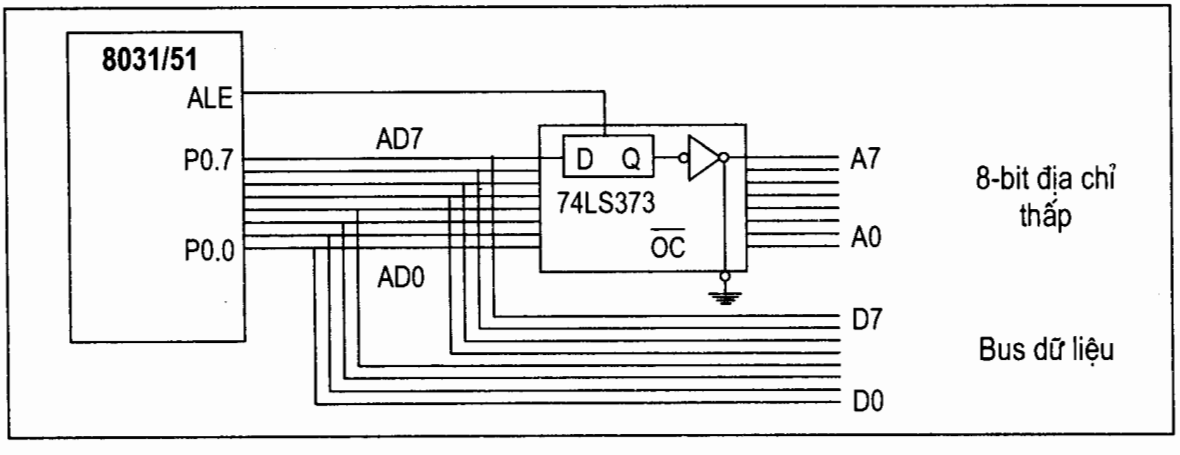
Hình 14.8. Mạch chốt 74LS373 D

Cổng P0 và P2 dùng để cung cấp địa chỉ

Do bộ đếm PC của 8031/51 là 16 bit

nên nó có thể truy cập được 64 kbyte mã chương trình. Ở 8031/51, cổng P0 và P2 cung cấp địa chỉ 16 bit để truy cập tới bộ nhớ ngoài, trong đó, P0 cấp 8 bit địa chỉ thấp là A0-A7, còn P2 thì cấp 8 bit địa chỉ cao từ A8-A15. Một nhiệm vụ quan trọng nữa đó là P0 còn được dùng để cấp bus dữ liệu 8 bit D0-D7. Như vậy, các chân P0.0-P0.7 vừa được dùng làm bus địa chỉ vừa làm bus dữ liệu. Cách tổ chức như vậy được gọi là dồn kênh dữ liệu/địa chỉ. Lý do mà hãng Intel sử dụng dồn kênh dữ liệu/địa chỉ là để tiết kiệm số chân của vi mạch 8031/51. Vậy làm thế nào để biết được khi nào thì P0 được dùng làm bus dữ liệu và khi nào làm bus địa chỉ. Đó là nhiệm vụ của chân cho phép chốt địa chỉ ALE (Address Latch Enable). Đây là một chân ra. Khi ALE = 0 thì P0 làm bus dữ liệu, còn khi ALE = 1 thì làm bus địa chỉ. Để mở rộng địa chỉ, cần nối các chân của P0 tới mạch chốt địa chỉ 74LS373 (xem hình 14.8) và dùng chân ALE để chốt địa chỉ như trình bày ở hình 14.9. Phương pháp mở rộng địa chỉ như vậy được gọi là phân kênh dữ liệu/địa chỉ.

Từ hình 14.9 cần lưu ý rằng, bình thường thì ALE = 0 và cổng P0 được dùng làm bus dữ liệu. Nhưng nếu 8031/51 muốn dùng P0 làm bus địa chỉ thì cần nối bus địa chỉ A0-A7 vào các chân của P0 và kích hoạt ALE = 1 để báo trên P0 có các địa chỉ.



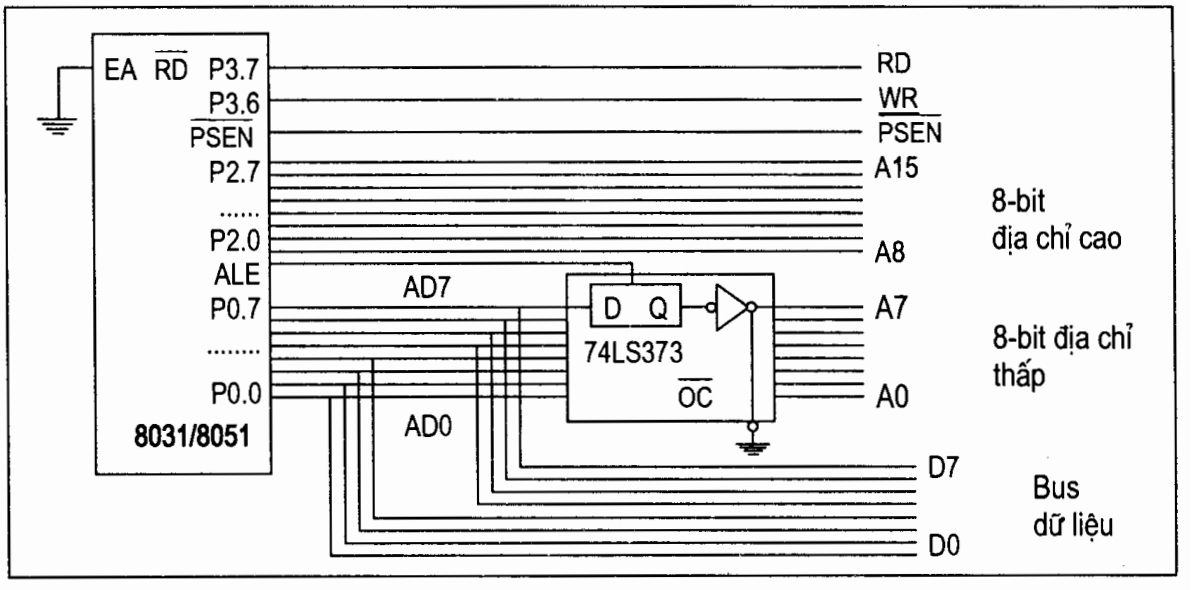
Hình 14.9. Dồn kênh dữ liệu/địa chỉ

PSEN

Một tín hiệu quan trọng khác của 8031/51 là cho phép cất chương trình PSEN (Program Store Enable). Đây là tín hiệu ra và được nối với chân OE của bộ nhớ ROM. Như vậy, để truy cập bộ nhớ ROM bên ngoài thì 8031/51 sử dụng tín hiệu PSEN. Cần nhấn mạnh đến vai trò của EA và PSEN khi ghép nối 8031/51

với ROM ngoài. Khi chân EA được nối đất thì 8031/51 nạp mã lệnh từ ROM ngoài vào thông qua chân PSEN. Để ý trên hình 14.1 ta thấy chân PSEN được nối với chân OE của ROM. Trong các hệ thống 8751/89C51/DS5000, chân EA được nối với V_{cc} và các chip này không kích hoạt chân PSEN. Điều này báo rằng mã lệnh đặt ở ROM trên chip.

Để ghi chương trình vào ROM ngoài, người ta thường dùng cổng nối tiếp hoặc song song của máy tính PC. Đây là phương pháp được ưa chuộng và được sử dụng khá phổ biến khi phát triển phần mềm, cũng như khi xây dựng các hệ thống huấn luyện và mô phỏng dựa trên 8051.

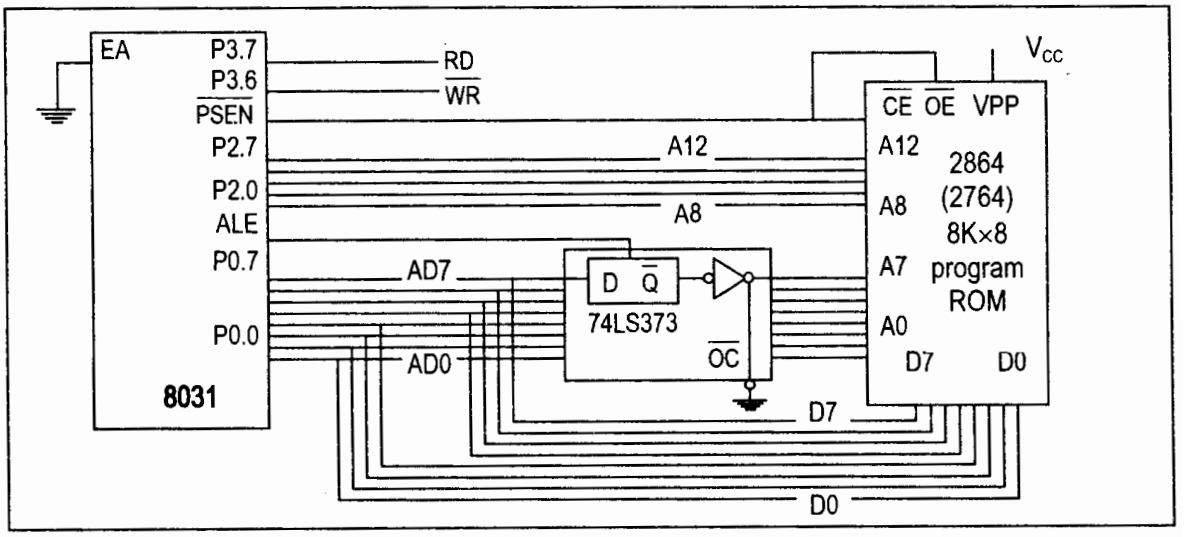


Hình 14.10. Bus điều khiển, địa chỉ và dữ liệu của 8031

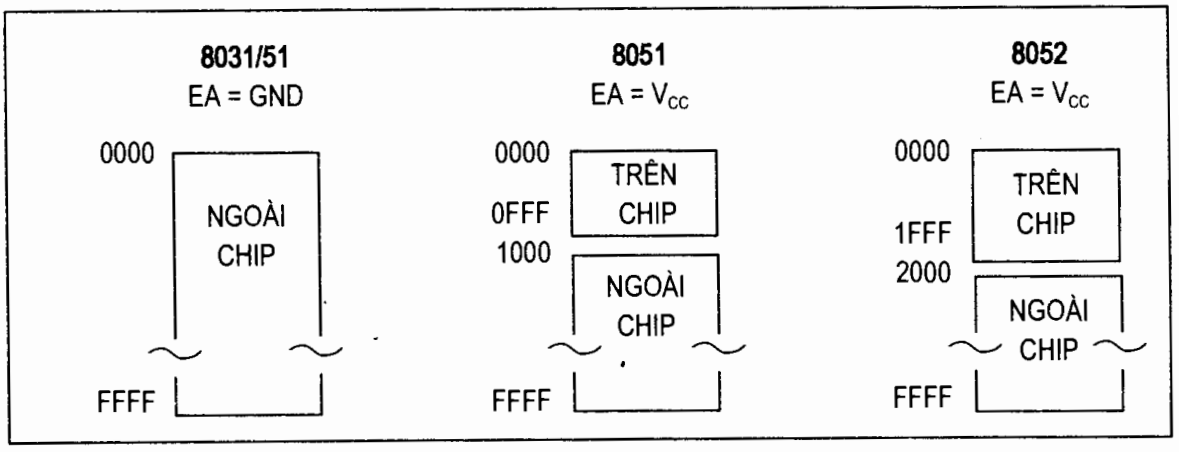
Bộ nhớ ROM chứa mã chương trình trên chip và ngoài chip

Ở các ví dụ cho đến nay, chúng ta dùng hoặc ROM trên chip hoặc ROM ngoài chip để lưu mã chương trình. Nếu muốn sử dụng đồng thời cả hai bộ nhớ ROM thì thế nào? Liệu có thực hiện được không? Câu trả lời là có. Ví dụ trong hệ 8751 hoặc 89C51 có thể dùng ROM trên chip để lưu mã chương trình khởi động, còn ROM ngoài (thường dùng là NV-RAM) để lưu mã chương trình người dùng. Theo cách này, trình khởi động hệ thống nằm ở trên chip, còn trình ứng dụng được nạp vào NV-RAM ngoài chip. Ở các hệ thống này, EA vẫn được nối vào V_{CC}. Như vậy, khi Reset thì 8051 thực hiện chương trình ở trên chip trước và sau

khi chạy xong thì mới chuyển sang chạy chương trình trên ROM ngoài. Nhiều thiết bị huấn luyện được thiết kế theo cách này. Cần lưu ý rằng, tất cả trình tự trên đều do 8051 tự động thực hiện. Ví dụ, trong một hệ 8751 (hoặc 89C51) khi có sử dụng cả ROM trên chip, ROM ngoài và chân EA nối với V_{CC} thì bộ vi điều khiển nạp các mã lệnh bắt đầu từ địa chỉ 0000 rồi đến địa chỉ 0FFF (địa chỉ cuối của ROM trên chip). Sau đó bộ đếm chương trình tạo địa chỉ 1000H và tự động chuyển hướng ra ROM ngoài có chứa mã chương trình. Chúng ta xem ví dụ 14.7 và 14.8. Hình 14.12 giới thiệu cấu hình bộ nhớ.



Hình 14.11. Nối ghép 8031 với bộ nhớ ROM ngoài



Hình 14.12. Bộ nhớ ROM trên chip và ngoài chip

Ví dụ 14.7

Hãy xem xét việc sử dụng không gian bộ nhớ ROM chương trình cho các trường hợp dưới đây:

a) EA = 0 đối với chip 8751 (89C51).

b) EA = V_{CC} đối với 8751 có sử dụng ROM trên chip và ROM ngoài.

c) EA = V_{CC} đối với 8752 có sử dụng ROM trên chip và ROM ngoài.

Giải:

a) Khi EA = 0 thì chân EA được nối đất và tất cả các chốt chương trình đều được chuyển hướng ra bộ nhớ ngoài bất kể 8751 có ROM trên chip chứa mã chương trình hay không. Bộ nhớ ROM ngoài này có thể lên đến 64 kbyte với không gian nhớ từ 0000H đến FFFFH. Trong trường hợp này thì 8751 (89C51) hoàn toàn giống như 8031.

b) Khi EA được nối với V_{cc} thì 8751 (89C51) nhận mã chương trình ở địa chỉ 0000H đến 0FFFH từ bộ nhớ ROM trên chip (có 4 kbyte ROM trên chip), còn chương trình từ 1000H đến FFFFH đều được nhận từ ROM ngoài.

c) Đối với hệ 8752 (89C52) có chân EA nối với V_{cc} thì bộ vi điều khiển nhận mã chương trình từ địa chỉ 0000H đến 1FFFH từ bộ nhớ ROM trên chip, còn chương trình có địa chỉ từ 2000H đến FFFFH đều nhận từ ROM ngoài.

Ví dụ 14.8

Xem xét vai trò của chân PSEN trong việc truy cập các mã chương trình trên chip và ngoài chip.

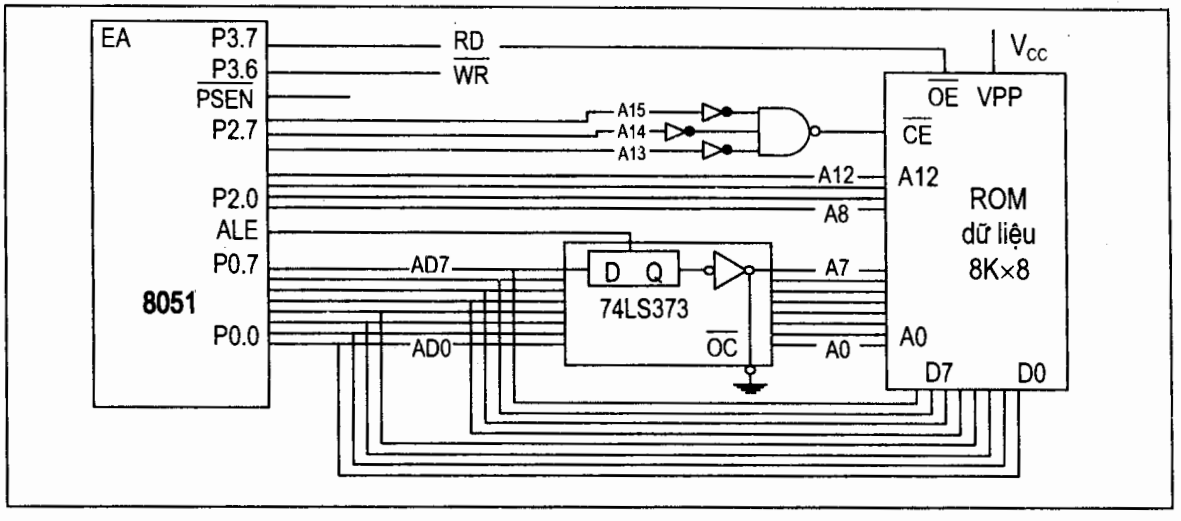
Giải:

Trong quá trình nạp mã chương trình từ bộ nhớ ROM trên chip thì chân PSEN không được sử dụng và không bao giờ được kích hoạt. Tuy nhiên, PSEN lại được dùng để nạp phần chương trình từ bộ nhớ ROM bên ngoài. Trên hình 14.11 có thể thấy chân PSEN cũng được dùng để kích hoạt chân CE của ROM chương trình.

14.4 KHÔNG GIAN BỘ NHỚ DỮ LIỆU CỦA 8051

Từ đầu đến giờ tài liệu này chỉ mới đề cập đến không gian bộ nhớ chương trình. Chúng ta cũng biết được rằng, bộ đếm chương trình của 8051 là 16 bit và

do vậy có thể truy cập đến 64 kbyte mã chương trình. Ở nhiều ví dụ trước, dữ liệu được đặt trong không gian mã chương trình và dùng lệnh "MOVEC A, @A+DPTR" để lấy dữ liệu. Chữ C trong lệnh MOVEC là từ chữ mã lệnh (Code) để báo rằng dữ liệu được đặt ở vùng không gian mã lệnh của 8051. Tuy nhiên, họ 8051 còn có không gian dữ liệu riêng biệt. Ở mục này chúng ta sẽ bàn kỹ hơn về không gian bộ nhớ dữ liệu của 8051.



Hình 14.13. Nối ghép 8051 với bộ nhớ ROM ngoài

Không gian bộ nhớ dữ liệu

Ngoài vùng không gian bộ nhớ chương trình, họ 8051 còn có 64 kbyte không-gian bộ nhớ dữ liệu. Như vậy, 8051 có tổng cộng 128 kbyte không gian địa chỉ, trong đó 64 kbyte dành cho mã chương trình và 64 kbyte dành cho dữ liệu. Không gian chương trình được truy cập nhờ bộ đếm chương trình PC, còn không gian dữ liệu được truy cập bởi thanh ghi con trỏ dữ liệu DPTR và một lệnh có tên là MOVX (chữ X là từ External để chỉ không gian bộ nhớ dữ liệu được thực hiện từ bộ nhớ ngoài).

ROM ngoài dành cho dữ liệu

Để nối ghép 8031/51 với bộ nhớ ROM dữ liệu ngoài, cần sử dụng chân RD (chân số P3.7) xem hình 14.13. Lưu ý vai trò của tín hiệu PSEN được dùng để nạp mã chương trình, còn đối với ROM dữ liệu thì tín hiệu RD dùng để đọc dữ liệu. Để truy cập vào vùng không gian bộ nhớ dữ liệu ngoài ta phải sử dụng lệnh MOVX như mô tả dưới đây.

Lệnh MOVX

Lệnh MOVX thường được sử dụng để truy cập đến không gian bộ nhớ dữ liệu ngoài. Để chuyển dữ liệu lưu trữ ở bộ nhớ ngoài vào CPU lệnh có dạng "MOVX A,@DPTR". Lệnh sẽ đọc byte dữ liệu do thanh ghi DPTR trỏ đến. Đối với các ứng dụng có không gian dữ liệu lớn thì sử dụng phương pháp bảng tra cứu. Ví dụ 14.9 và 14.10 dưới đây sẽ minh hoạ cách sử dụng lệnh MOVX.

Ví dụ 14.9

Bộ nhớ ROM ngoài được dùng để cất bảng dữ liệu DAC (bắt đầu từ địa chỉ 1000H). Hãy viết chương trình đọc 30 byte dữ liệu này và gửi tới cổng P1.

Giải:

```
MYXDATA EQU     1000H
COUNT   EQU     30
        ...
        MOV     DPTR,#MYXDATA   ;Trỏ đến dữ liệu ngoài
        MOV     R2,#COUNT       ;Bộ đếm
AGAIN:  MOVX    AM@DPTR         ;Chuyển từ bộ nhớ ngoài
        MOV     P1,A            ;Chuyển đến P1
        INC     DPTR            ;Trỏ đến dữ liệu tiếp theo
        DJNZ    R2,AGAIN        ;Cho đến khi đọc hết
```

Ví dụ 14.10

ROM dữ liệu ngoài chứa bảng giá trị bình phương các số từ 0 đến 9. Vì RAM của 8031/51 có thời gian truy cập ngắn hơn nên bạn hãy viết chương trình để sao chép các phần tử của bảng này vào RAM trong bắt đầu từ địa chỉ 30H. Bảng dữ liệu bắt đầu từ địa chỉ 0 của ROM ngoài.

Giải:

```
TABLE   EQU     1000H
RAMTBLE EQU     30H
COUNT   EQU     10
        ...
        MOV     DPTR,#TABLE   ;Trỏ đến dữ liệu ngoài
        MOV     R5,#COUNT     ;Nạp giá trị đếm
        MOV     R0,#RAMTBLE   ;Trỏ đến bộ nhớ RAM trong
```

```
BACK:   MOVX    A,@DPTR      ;Lấy dữ liệu từ bộ nhớ ngoài
        MOV     @R0,A        ;Lưu vào RAM trong
        INC     DPTR         ;Trỏ đến dữ liệu tiếp theo
        INC     R0           ;Trỏ đến vị trí RAM tiếp theo
        DJNZ    R5,BACK      ;Cho đến khi đọc xong
```

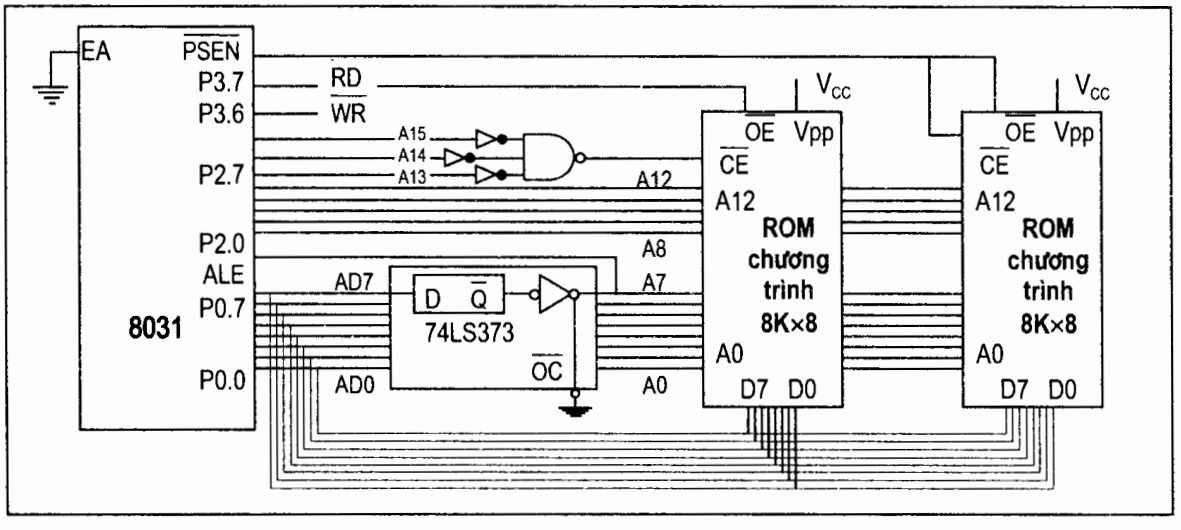
So sánh ví dụ 14.10 với ví dụ 5.8 ta thấy rằng, ở ví dụ 5.8 thì các phần tử bảng được lưu cất trong vùng mã chương trình của 8051 và cần dùng lệnh MOVC để truy cập vào bảng này. Mặc dù cả hai lệnh "MOVC A,@A+DPTR" và "MOVX A, @DPTR" khá giống nhau về hình thức, nhưng điểm khác nhau là một lệnh thực hiện nhận dữ liệu từ vùng nhớ chương trình, còn lệnh kia thì nhận dữ liệu từ vùng dữ liệu của bộ vi điều khiển.

Ví dụ 14.11

Hãy thiết kế hệ xây dựng trên 8031 gồm có 8 kbyte ROM chương trình và 8 kbyte ROM dữ liệu.

Giải:

Hình 14.14 là thiết kế theo yêu cầu đã cho. Cần lưu ý đến chân PSEN và RD cho từng loại ROM. Đối với ROM chương trình, PSEN được dùng để kích hoạt cả OE và CE, còn đối với ROM dữ liệu thì dùng RD để kích hoạt OE, còn CE thì được kích hoạt nhờ một bộ giải mã đơn giản.

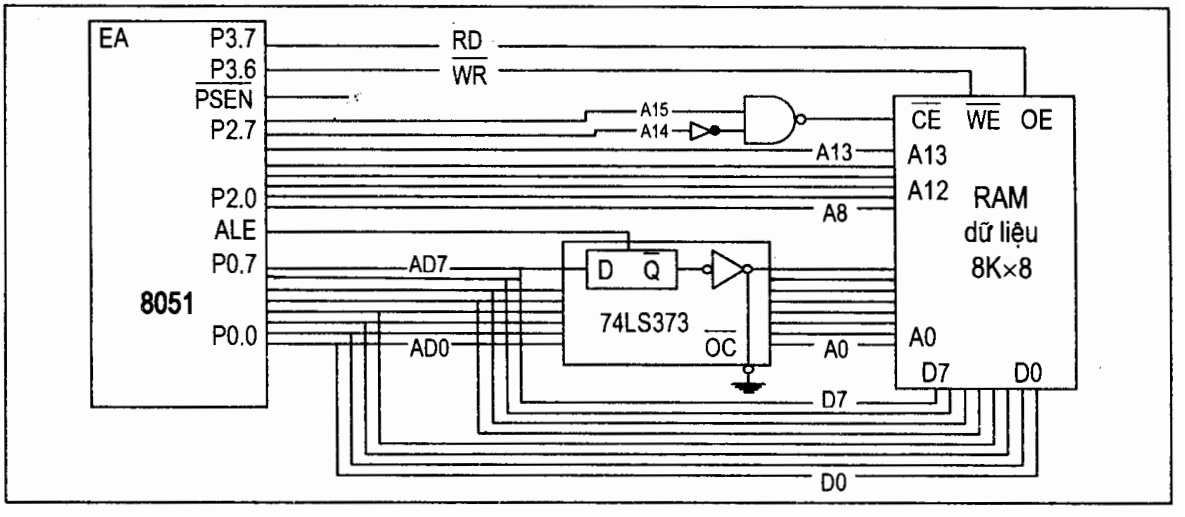


Hình 14.14. Nối ghép 8031 với ROM dữ liệu và ROM chương trình ngoài

Từ những trình bày trên cho thấy, có thể dùng RAM và các thanh ghi trong CPU để lưu dữ liệu, còn mọi không gian nhớ bổ xung cho dữ liệu đọc/ghi thì phải thực thi từ bên ngoài. Dưới đây ta tiếp tục bàn về vấn đề này.

RAM dữ liệu ngoài

Để ghép nối 8051 với RAM ngoài thì cần phải dùng cả chân RD (P3.7) và chân WR (P3.6). Xem hình 14.15.



Hình 14.15. Nối ghép 8051 với bộ nhớ RAM dữ liệu ngoài

Lệnh MOVX nhận dữ liệu RAM ngoài

Để ghi dữ liệu ra RAM dữ liệu ngoài có thể dùng lệnh "MOVX @DPTR, A", trong đó thanh ghi A chứa dữ liệu, còn địa chỉ của RAM ngoài do thanh ghi DPTR trỏ đến. Lệnh này rất hay được sử dụng, kể cả khi cần chuyển một lượng dữ liệu lớn. Dữ liệu thường được cất trong NV-RAM để nếu mất nguồn, dữ liệu vẫn không bị mất. Chúng ta xem ví dụ 14.12.

Ví dụ 14.12

a) Viết chương trình đọc 200 byte dữ liệu từ cổng P1 và lưu kết quả đọc được vào RAM ngoài bắt đầu từ địa chỉ 5000H.

b) Không gian địa chỉ được cấp cho RAM dữ liệu ở hình 14.15 là gì?

Giải:

a)

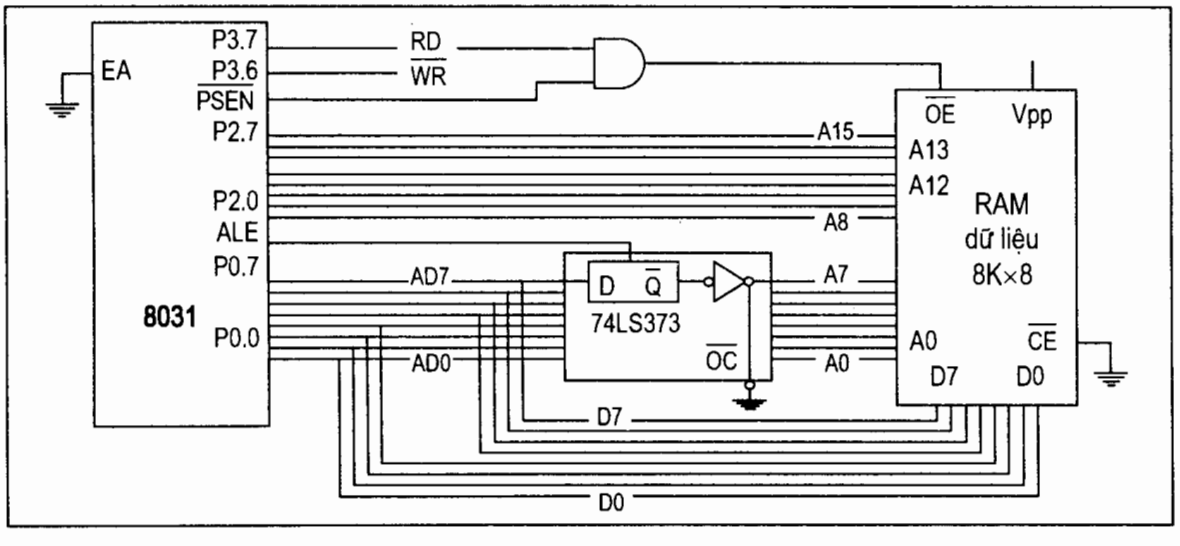
```
RAMDATA EQU     500H
```

```
COUNT   EQU     200
        MOV     DPTR,#RAMDTA    ;Trỏ đến NV-RAM
        MOV     R3,#COUNT       ;Nạp giá trị đếm
AGAIN:  MOV     A,P1            ;Đọc dữ liệu từ P1
        MOVX    @DPTR,A         ;Lưu vào NV-RAM
        ACALL   DELAY           ;Chờ trước khi đọc tiếp
        INC     DPTR            ;Trỏ đến dữ liệu tiếp
        DJNZ    R3,AGAIN        ;Cho đến khi đọc hết
HERE:   SJMP    HERE            ;Dừng tại đây khi kết thúc
```

b) Không gian địa chỉ dữ liệu là 8000H đến BFFFH.

ROM ngoài chung cho cả mã chương trình và dữ liệu

Giả sử hệ thống 8031 được nối với một chip ROM ngoài (ROM ngoài duy nhất) có dung lượng 64K× 8 (27512). Chip ROM này được dùng để lưu cả mã chương trình và dữ liệu. Ví dụ, không gian địa chỉ từ 0000H đến 7FFFH được cấp cho mã chương trình, còn không gian từ D000H đến FFFFH dành cho dữ liệu. Dùng lệnh MOVX để truy cập dữ liệu. Vậy cách nối chân PSEN và RD như thế nào. Hãy nhớ lại phần trước rằng, tín hiệu PSEN được dùng để truy cập tới không gian mã ngoài và tín hiệu RD được dùng để truy cập không gian dữ liệu ngoài. Để chỉ một chip ROM này có thể lưu cả mã chương trình và cả dữ liệu, cần nối cổng AND đến chân OE của chip ROM như trên hình 14.16.



Hình 14.16. Một ROM dùng đồng thời cho dữ liệu và chương trình

Hệ 8031 với bộ nhớ ROM và RAM

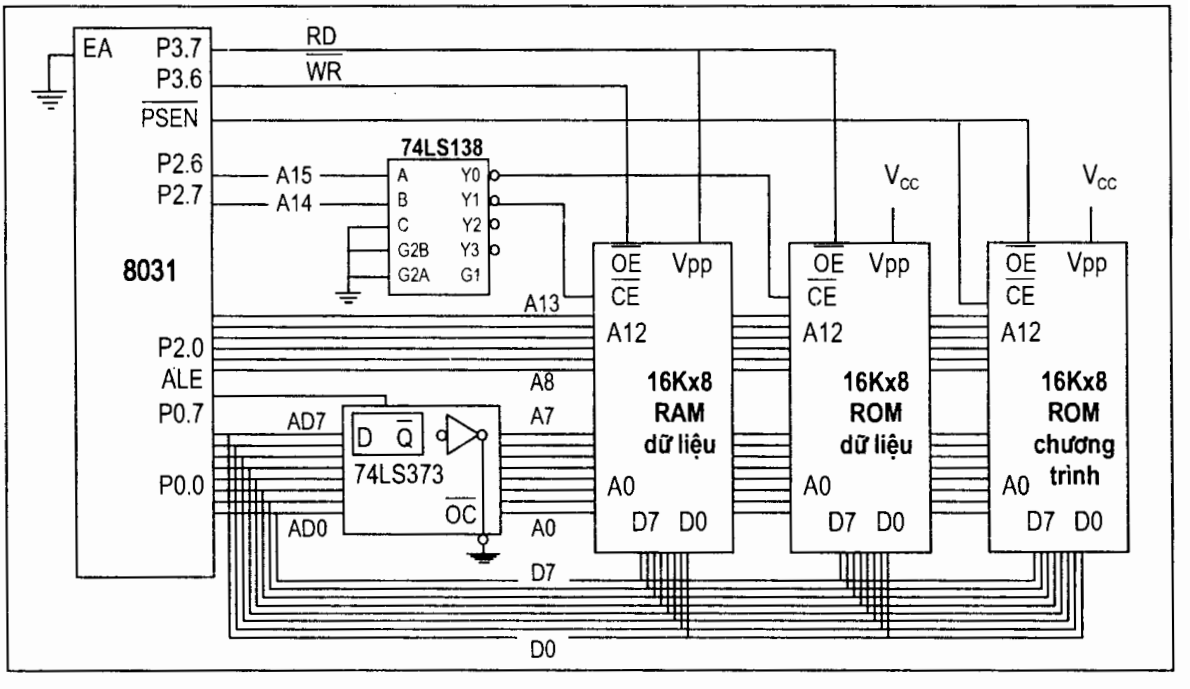
Nhiều ứng dụng cần có đồng thời ROM chương trình, ROM dữ liệu và RAM dữ liệu trong một hệ thống. Chúng ta xem ví dụ minh hoạ 14.13.

Ví dụ 14.13

Giả sử cần có một hệ 8031 với 16 kbyte ROM mã chương trình, 16 kbyte ROM dữ liệu bắt đầu tại địa chỉ 0000H và 16 kbyte NV-RAM bắt đầu từ địa chỉ 8000H. Hãy thiết kế một hệ thống có sử dụng một vi mạch 74LS138 làm giải mã địa chỉ.

Giải:

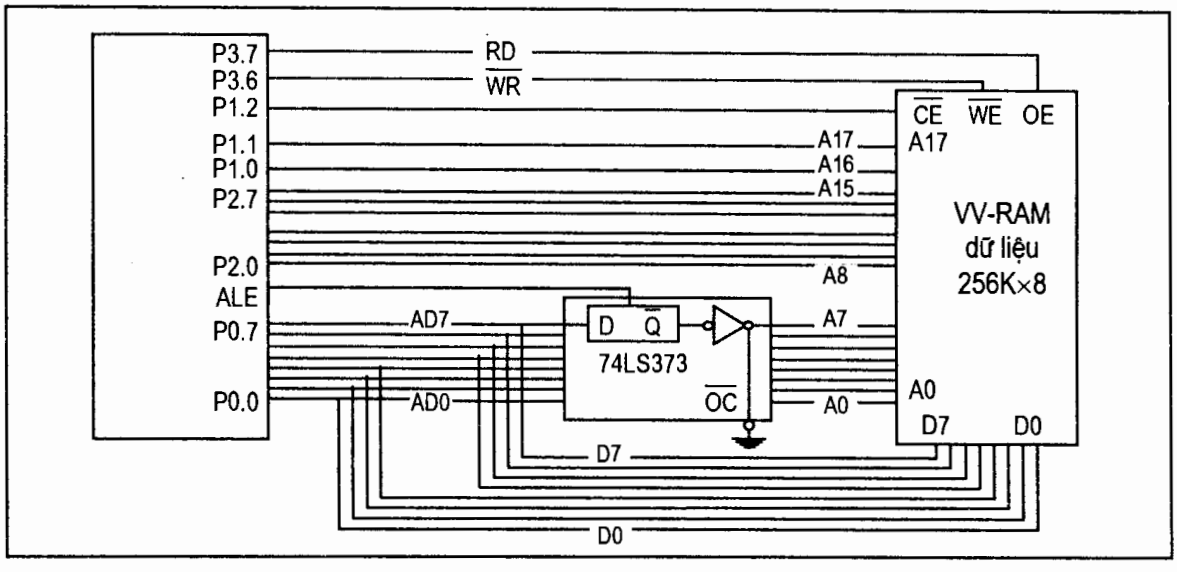
Sơ đồ thiết kế được trình bày trên hình 14.17. Để ý ta thấy rằng đối với ROM chương trình thì không cần bộ giải mã, song đối với ROM và RAM dữ liệu thì cần một bộ giải mã địa chỉ 74LS138. Cũng có thể nhận thấy rằng G1 = V_{CC}, G2A = GND, G2B = GND và đầu vào C của 74LS138 được nối đất vì chỉ cần dùng Y0 – Y3.



Hình 14.17. Ghép nối 8031 với bộ nhớ ngoài: ROM chương trình, ROM dữ liệu, RAM dữ liệu

Nối ghép với bộ nhớ ngoài lớn

Nhiều ứng dụng cần có một bộ nhớ ngoài lớn để lưu dữ liệu (chẳng hạn 256 Kb). Tuy nhiên 8051 chỉ có thể hỗ trợ bộ nhớ dữ liệu ngoài với kích thước 64 K vì DPTR là thanh ghi 16 bit. Để giải quyết vấn đề này, có thể nối trực tiếp chân A0 - A15 của 8051 với chân A0 - A15 của bộ nhớ ngoài, và dùng một số chân của cổng P1 để truy cập khối 64 kbyte khác của chip nhớ 256k×8 byte. Chúng ta xem xét ví dụ 14.14 và hình 14.18.



Hình 14.18. Nối ghép 8051 với bộ nhớ NV-RAM ngoài

Ví dụ 14.14

Một ứng dụng 8051 cần có 256 kbyte NV-RAM để lưu dữ liệu.

a) Trình bày sơ đồ ghép nối 8051 với NV-RAM 256k×8.

b) Trình bày cách truy cập từng khối dữ liệu của chip này.

Giải:

a) NV-RAM 256k×8 có 18 chân địa chỉ (A0-A17) và 8 đường dữ liệu như trình bày ở hình 14.18. Các chân A0 - A15 được nối trực tiếp với chip nhớ, còn A16 và A17 được nối với các chân P1.0 và P1.1. Cũng để ý thấy rằng chân $\overline{CS}$ của chip nhớ được nối với P1.2 của 8051.

b) 256 kbyte bộ nhớ được chia thành 4 khối, từng khối được truy cập như sau:

Chọn chip	***A17***	***A16***	
P1.2	***P1.1***	***P1.0***	***Địa chỉ***
0	0	0	00000H - 0FFFFH
0	0	1	10000H - 1FFFFH
0	1	0	20000H - 2FFFFH
0	1	1	30000H - 3FFFFH
1	x	x	External RAM disabled

Ví dụ, để truy cập địa chỉ 2000H – 2FFFFH chương trình được viết như sau:

```
CLR  P1.2     ;Cho phép mở RAM ngoài
MOV  DPTR,#0  ;Bắt đầu từ khối nhớ 64 K
CLR  P1.0     ;A16=0
SETB P1.1     ;A17=1 truy cập khối nhớ có địa chỉ 20000H
MOV  A,SBUF   ;Lấy dữ liệu qua cổng nối tiếp
MOVX @DPTR    ;Cất dữ liệu vào khối có địa chỉ 20000H
INC  DPTR     ;Trỏ đến ô nhớ tiếp theo
```

Chương 15
NỐI GHÉP 8031/51 VỚI 8255

Như đề cập ở chương 14, khi nối ghép 8031/51 với bộ nhớ ngoài thì hai cổng P0 và P2 được sử dụng. Trong chương này chúng ta sẽ trình bày phương pháp mở rộng các cổng I/O của 8031/51 thông qua chip 8255.

15.1 LẬP TRÌNH 8255

Trong mục này chúng ta tìm hiểu 8255, một trong những chip vào/ra được sử dụng rộng rãi nhất.

Đặc điểm của 8255

8255 là chip dạng DIP 40 chân (hình 15.1). Chip có 3 cổng 8 bit là A, B và C được truy cập riêng biệt. Các cổng này đều có thể lập trình làm cổng vào hoặc ra độc lập. Ngoài ra, các cổng của 8255 có khả năng bắt tay, do vậy, cho phép giao diện với các thiết bị khác cũng có tín hiệu bắt tay, ví dụ như máy in. Khả năng bắt tay của 8255 sẽ được bàn tới ở mục sau.

PA0 - PA7

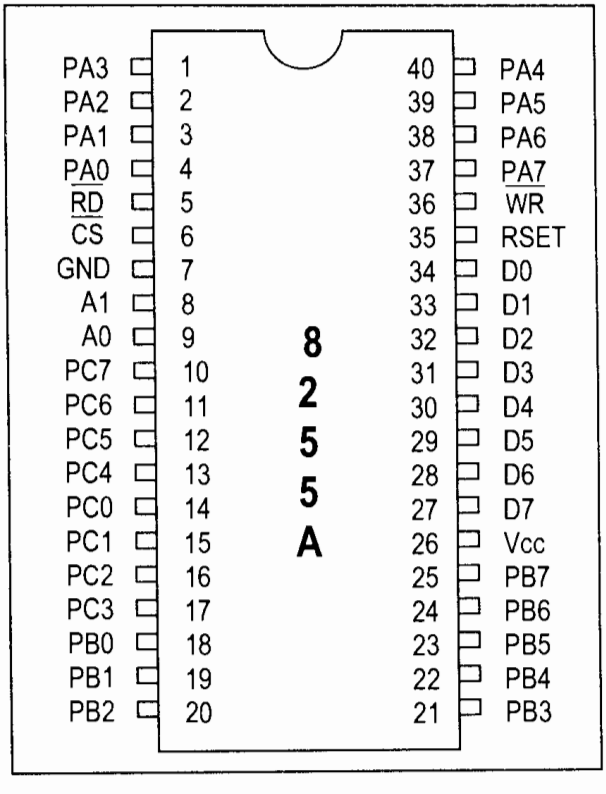
Cả 8 bit của cổng A PA0 - PA7 có thể được lập trình thành 8 bit vào, 8 bit ra, hoặc cả 8 bit hai chiều vào/ra.

PB0 - PB7

Cả 8 bit của cổng B (PB0-PB7) cũng có thể lập trình thành 8 bit vào, 8 bit ra, hoặc cả 8 bit hai chiều vào/ra.

PC0 - PC7

Tất cả 8 bit của cổng C (PC0 - PC7) đều có thể được lập trình thành các bit vào hoặc các bit ra. 8 bit này cũng có thể được chia làm hai phần:

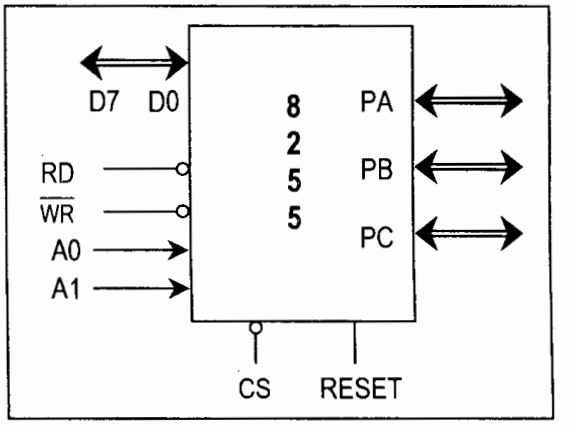


Hình 15.1. Chip 8255

Phần cao (PC4 - PC7) là CU (Upper Bits) và phần thấp (PC0 - PC3) là CL (Lower Bits). Mỗi phần có thể được sử dụng độc lập làm đầu vào hoặc làm đầu ra. Ngoài ra, từng bit của cổng C từ PC0 - PC7 cũng có thể được lập trình riêng rẽ.

$\overline{RD}$ và $\overline{WR}$

Là hai tín hiệu điều khiển tích cực mức thấp và là các chân vào của 8255. Các chân tín hiệu $\overline{RD}$ và $\overline{WR}$ của 8031/51 được nối tới các chân này.



Hình 15.2. Sơ đồ khối của 8255

D0 - D7

Các chân dữ liệu D0 - D7 của 8255 được nối tới các chân dữ liệu của bộ vi điều khiển để cho phép trao đổi dữ liệu giữa chúng.

RESET

Là tín hiệu vào tích cực mức cao, được dùng để xoá thanh ghi điều khiển. Khi chân RESET được kích hoạt thì tất cả các cổng được khởi tạo lại làm các cổng vào. Trong nhiều thiết kế thì chân này được nối tới đầu ra RESET của bus hệ thống hoặc được nối tới đất để không bị kích hoạt. Cũng như tất cả các chân vào của IC, chân này cũng có thể để hở.

A0, A1 và $\overline{CS}$

$\overline{CS}$ là chân chọn chip, còn A0 và A1 cho phép chọn cổng. Các chân này được dùng để truy cập các cổng A, B, C hoặc thanh ghi điều khiển như trình bày ở bảng 15.1. Chân $\overline{CS}$ có mức tích cực thấp.

Bảng 15.1. Chọn cổng của 8255

$\overline{CS}$	A1	A0	Chọn cổng
0	0	0	Cổng A
0	0	1	Cổng B
0	1	0	Cổng C
0	1	1	Thanh ghi điều khiển
1	x	X	8255 không được

Chọn chế độ của 8255

Các cổng A, B và C của 8255 được dùng để nhập và xuất dữ liệu, còn thanh ghi điều khiển thì cần được lập trình để chọn chế độ làm việc cho các cổng này. Các cổng của 8255 có thể được lập trình theo các chế độ sau:

1. Chế độ 0 (Mode 0): Đây là chế độ vào/ra đơn giản. Ở chế độ này, các cổng A,

B, CL và CU có thể được lập trình để làm đầu vào hoặc đầu ra. Nên nhấn mạnh rằng, ở chế độ 0 thì tất cả các bit hoặc làm đầu vào hoặc làm đầu ra mà không thể điều khiển riêng rẽ từng bit như ở các cổng P0 - P3 của 8051. Vì các ứng dụng liên quan đến 8255 chủ yếu sử dụng chế độ này nên chúng ta sẽ tìm hiểu kỹ hơn.

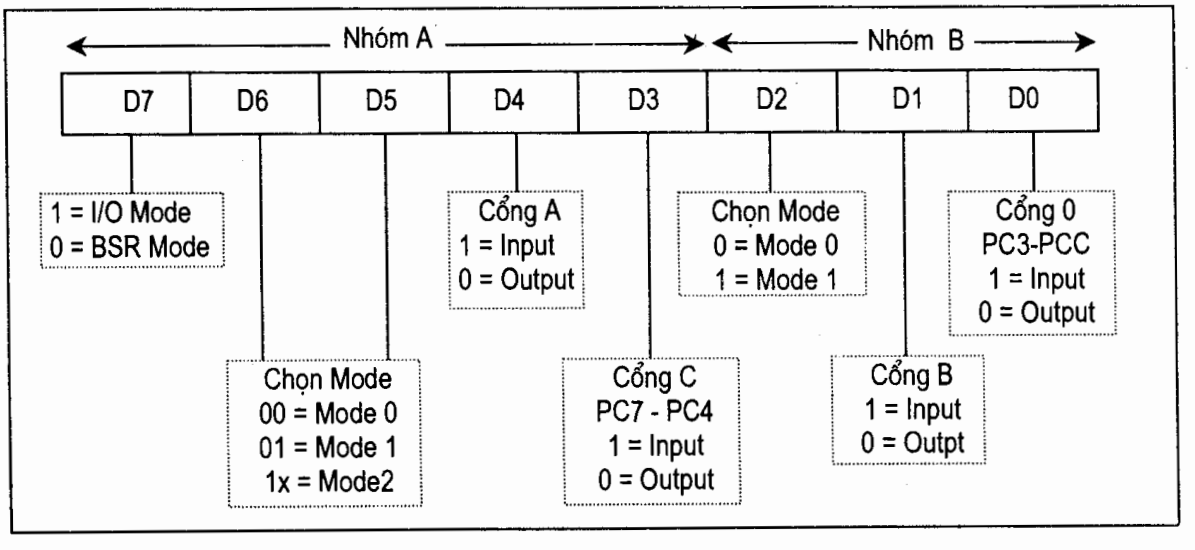
2. Chế độ 1 (Mode 1): Ở chế độ này, các cổng A và B có thể được dùng làm đầu vào hoặc đầu ra có khả năng bắt tay. Tín hiệu bắt tay được các bit của cổng C cấp (sẽ được đề cập ở mục 15.3).

3. Chế độ 2 (Mode 2): Ở chế độ này, cổng A có thể được dùng làm cổng vào/ra hai chiều với khả năng bắt tay. Các tín hiệu bắt tay được cấp bởi các bit cổng C. Cổng B có thể được dùng như ở chế độ vào/ra đơn giản hoặc ở chế độ bắt tay Mode1. Chế độ này sẽ không được trình bày trong tài liệu này.

4. Chế độ BSR: Đây là chế độ thiết lập/xoá bit (Bit Set/Reset). Ở chế độ này chỉ có những bit riêng rẽ của cổng C có thể được lập trình (sẽ được trình bày ở mục sau).

Lập trình chế độ vào/ra đơn giản

Hãng Intel gọi chế độ 0 là chế độ vào/ra cơ sở. Một thuật ngữ khác được sử dụng phổ biến đó là vào/ra đơn giản. Ở chế độ này thì cổng bất kỳ A, B hay C đều có thể được lập trình làm cổng vào hoặc ra riêng rẽ. Tuy nhiên một cổng không thể đồng thời vừa là đầu vào lại vừa là đầu ra.



Hình 15.3. Khuôn dạng từ điều khiển của 8255 (chế độ vào/ra)

Ví dụ 15.1

Hãy xác định từ điều khiển của 8255 cho các cấu hình sau:

a) Tất cả các cổng A, B và C đều là các cổng đầu ra (chế độ 0).

b) PA là đầu vào, PB - đầu ra, PCL - đầu vào và PCH - đầu ra.

Giải:

Từ hình 15.3 ta tìm được:

```
a) 1000 0000 = 80H
b) 1001 000 = 90H
```

Nối ghép 8031/51 với 8255

Chip 8255 được lập trình theo một trong bốn chế độ vừa trình bày ở trên bằng cách gửi một byte (hãng Intel gọi là từ điều khiển) tới thanh ghi điều khiển của 8255. Trước hết, chúng ta phải xác định được địa chỉ cổng được gán cho mỗi cổng A, B, C và thanh ghi điều khiển. Đây được gọi là ánh xạ cổng vào/ra (mapping).

Như có thể thấy từ hình 15.4, 8255 nối tới 8031/51 cũng giống như một bộ nhớ RAM. Nên lưu ý đến cách sử dụng tín hiệu $\overline{RD}$ và $\overline{WR}$. Phương pháp nối chip vào/ra tới CPU như vậy gọi là *vào/ra được ánh xạ vào bộ nhớ*. Hay nói cách khác, chúng ta sử dụng không gian bộ nhớ để truy cập các thiết bị vào/ra. Vì vậy mà chúng ta có thể dùng lệnh MOVX để truy cập 8255. Ở chương 14 chúng ta đã sử dụng lệnh MOVX để truy nhập RAM và ROM. Trường hợp 8255 được gép nối với 8031/51, chúng ta cũng dùng lệnh MOVX để thực hiện truyền thông giữa chúng. Hãy xem ví dụ 15.2.

Ví dụ 15.2

Dựa vào hình 15.4:

a) Hãy tìm địa chỉ vào/ra gán cho cổng A, B, C và thanh ghi điều khiển.

b) Hãy lập trình để các cổng A, B và C của 8255 thành các cổng ra.

c) Viết một chương trình để gửi liên tục 55H và AAH đến tất cả các cổng.

Giải:

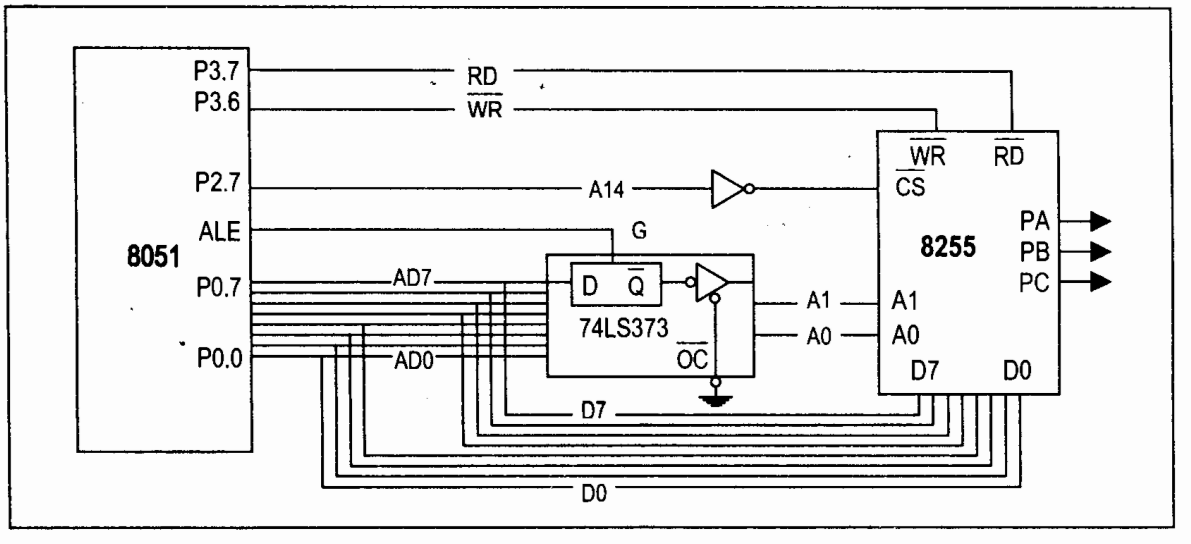
a) Địa chỉ cơ sở của 8255 như sau:

A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	0	=4000HPA
x	1	x	x	x	x	x	x	x	x	x	x	x	X	0	1	=4000HPB
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	0	=4000HPC
x	1	x	x	x	x	x	x	x	x	x	x	x	X	1	1	=4000HCR

b) Byte (từ) điều khiển cho tất cả các cổng làm đầu ra là 80H như được tính ở Ví dụ 15.1.

c)

```
      MOV  A,#80H          ;Từ điều khiển (cổng ra)
      MOV  DPTR,#4003H     ;Nạp ĐC cổng TG điều khiển
      MOVX @DPTR,A         ;Xuất từ điển khiển
      MOV  A,#55H          ;Gán A=55
AGAIN:
      MOV  DPTR,#4000H     ;Địa chỉ cổng PA
      MOVX @DPTR,A         ;Đan xen trị 0,1 các bit cổng PA
      INC  DPTR            ;Địa chỉ cổng PB
      MOVX @DPTR,A         ;Đan xen trị 0,1 các bit cổng PB
      INC  DPTR            ;Địa chỉ cổng PC
      MOVX @DPTR,A         ;Đan xen trị 0,1 các bit cổng PC
      CPL  A               ;Đảo các bit thanh ghi A
      ACALL DELAY          ;Chờ
      SJMP AGAIN           ;Tiếp tục
```



Hình 15.4. Nối ghép 8051 với 8255 ở ví dụ 15.2

Ví dụ 15.3

Dựa trên hình 15.5:

a) Tìm địa chỉ cổng vào/ra gán cho các cổng A, B, C và thanh ghi điều khiển.

b) Tìm byte điều khiển để PA là cổng vào, PB là cổng ra và PC là cổng ra.

c) Viết một chương trình để lấy dữ liệu ở PA và gửi đến cổng B và C.

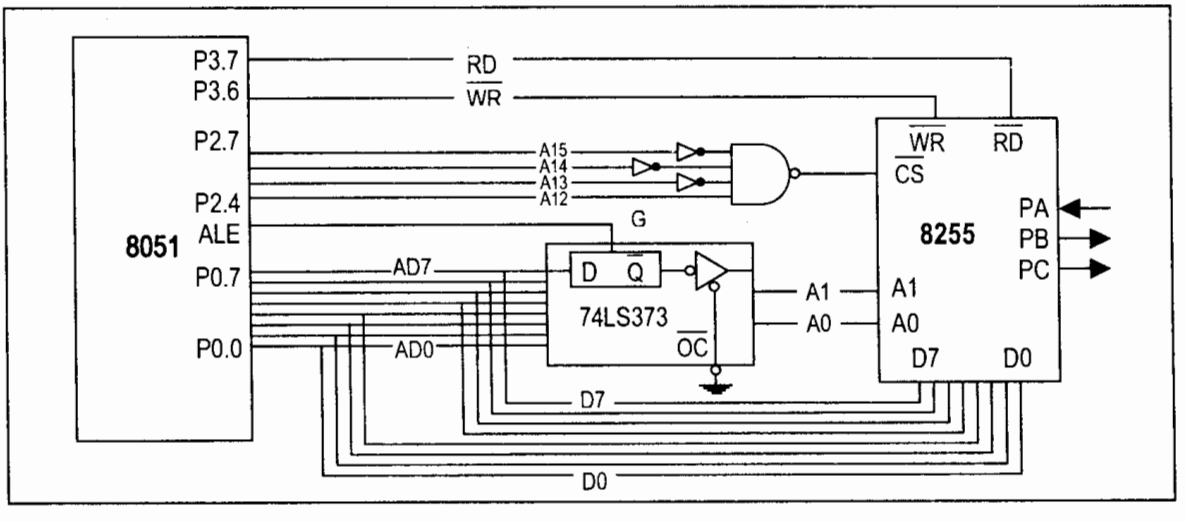
Giải:

a) Giả sử tất các các bit không dùng đều được gán giá trị 0, thì địa chỉ cổng cơ sở của 8255 là 1000H. Do vậy ta có:

PA - 1000H, PB - 1001H, PC - 1002H

và thanh ghi điều khiển giá trị 1003H.

c)
```
    MOV     A,#90H          ;PA đầu vào,PB-đầu ra,PC-đầu ra
    MOV     DPTR,#1003H     ;Nạp địa chỉ cổng thanh ghi ĐK
    MOVX    @DPTR,A         ;Xuất từ điều khiển
AGAIN:
    MOV     DPTR,#4000H     ;Địa chỉ PA
    MOVX    A,@DPTR         ;Nhận dữ liệu từ PA
    INC     DPTR            ;Địa chỉ PB
    MOVX    @DPTR,A         ;Gửi dữ liệu ra PB
    INC     DPTR            ;Địa chỉ PC
    MOVX    @DPTR,A         ;Gửi dữ liệu ra PC
```



Hình 15.5. Nối ghép 8051 với 8255 ở ví dụ 15.3

Ở ví dụ 15.3 ta nên dùng chỉ dẫn EQU để định nghĩa địa chỉ các cổng A, B, C và thanh ghi điều khiển CNTPORT như sau:

```
APORT       EQU  1000H
BPORT       EQU  1001H
CPORT       EQU  1002H
CNTPORT     EQU  1003H

MOV    A,#90H          ;PA là đầu vào,PB-đầu ra,PC-đầu ra
MOV    DPTR,#CNTPORT   ;Nạp địa chỉ cổng TG điều khiển
MOVX   @DPTR,A         ;Xuất từ điều khiển
MOV    DPTR,#CNTPORT   ;Địa chỉ PA
MOVX   DPTR,APORT      ;Nhận dữ liệu PA
INC    A,@DPTR         ;Địa chỉ PB
MOVX   DPTR            ;Gửi dữ liệu ra PB
INC    DPTR            ;Địa chỉ PC
MOVX   DPTR,A          ;Gửi dữ liệu ra PC
```

hoặc có thể viết lại như sau:

```
CONTRBYT EQU  90H              ;PA=vào, PB=PC=ra
BAS8255P EQU  1000H            ;Địa chỉ cơ sở của 8255
  MOV    A,#CONTRBYT
  MOV    DPTR,#BAS8255P+3      ;Nạp địa chỉ cổng C
  MOVX   @DPTR,A               ;Xuất từ điều khiển
  MOV    DPTR,#BASS8255P       ;Địa chỉ cổng A
  ...
```

Để ý trong ví dụ 15.2 và 15.3 chúng ta sử dụng thanh ghi DPTR vì địa chỉ cơ sở mà 8255 dùng là 16 bit. Nếu địa chỉ cơ sở là 8 bit, thì có thể sử dụng lệnh "MOVX A, @R0" và "MOVX @R0, A", trong đó R0 (hoặc R1) giữ địa chỉ cổng 8 bit. Ở ví dụ 15.4 có sử dụng một cổng lôgíc đơn giản để giải mã địa chỉ cho 8255. Trường hợp hệ thống có nhiều chip 8255 thì ta có thể sử dụng mạch 74LS138 để giải mã như sẽ trình bày ở ví dụ 15.5.

Ví dụ 15.4

Dựa trên hình 15.6:

a) Hãy tìm địa chỉ cổng vào/ra của các cổng A, B, C và thanh ghi điều khiển.

b) Tìm từ điều khiển cho trường hợp PA là đầu ra, PB là đầu vào, PC - PC3 là đầu vào và CP4 - CP7 là đầu ra.

c) Viết chương trình để lấy dữ liệu từ PB và gửi ra PA. Ngoài ra, dữ liệu từ PC1 được gửi đến CPU.

Giải:

a) Các địa chỉ cổng tìm được như sau:

BB	$\overline{CS}$	A1	A0	Địa chỉ	Cổng
0010	00	0	0	20H	Cổng A
0010	00	0	1	21H	Cổng B
0010	00	1	0	22H	Cổng C
0010	00	1	1	23H	Thanh ghi điều khiển

b) Từ điều khiển là 10000011 hay 83H.

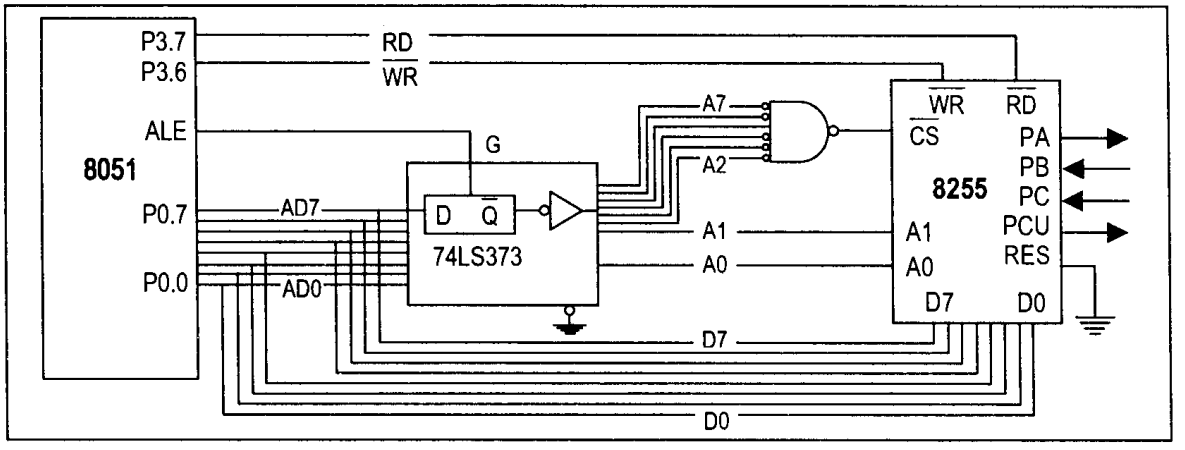
c)
```
CONTRBYT    EQU  83H   ;PA=ra,PB=vào,PCL=vào,PCU=ra
APORT  EQU  20H
BPORT  EQU  21H
CPORT  EQU  22H
CNTPORT     EQU  23H
       ...
  MOV  A,#CONTRBYT     ;PA=ra,PB=vào,PCL=vào,PCU=ra
  MOV  R0,#CNTPORT     ;Nạp địa chỉ cổng TG điều khiển
  MOVX @R0,A           ;Xuất từ điều khiển
  MOV  R0,#BPORT       ;Nạp địa chỉ PB
  MOVX A,@R0           ;Đọc PB
  DEC  R0              ;Chỉ đến PA (20H)
  MOVX @R0,A           ;Gửi nó đến PA
  MOV  R0,#CPORT       ;Nạp địa chỉ PC
  MOVX A,@R0           ;Đọc PCL
  ANL  A, #0FH         ;Che phần cao
  SWAP A               ;Tráo đổi phần cao và thấp
  MOVX @R0,A           ;Gửi đến PCU
```

Ví dụ 15.5

Dựa vào hình 15.7 hãy xác định địa chỉ cơ sở cho 8255.

Giải:

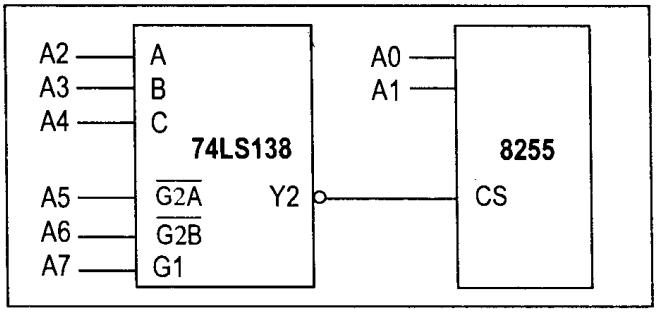
GA	$\overline{G2B}$	$\overline{G2A}$	C	B	A			Địa chỉ
A7	A6	A5	A4	A3	A2	A1	A0	
1	0	0	0	1	0	0	0	88H



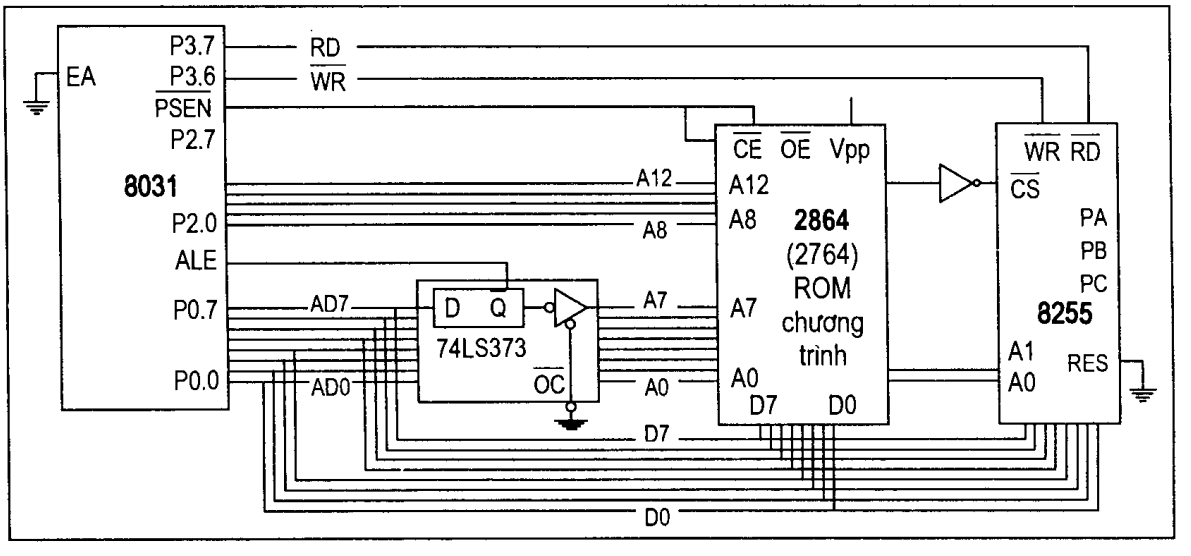
Hình 15.6. Nối ghép 8051 với 8255 ở ví dụ 15.4

Nối ghép 8031 với 8255

Nếu hệ 8031 có bộ nhớ ROM ngoài, thì sử dụng 8255 là giải pháp tốt. Lý do, là nếu không dùng 8255 thì 2 cổng P0 và P2 của 8031 sẽ bị chiếm và chỉ còn lại duy nhất cổng P1. Như vậy, nối ghép 8255 cho phép có thêm một số cổng. Xem hình 15.8.



Hình 15.7. Dùng mạch 74LS138



Hình 15.8. Nối ghép 8031 với 8255 và ROM chương trình ngoài

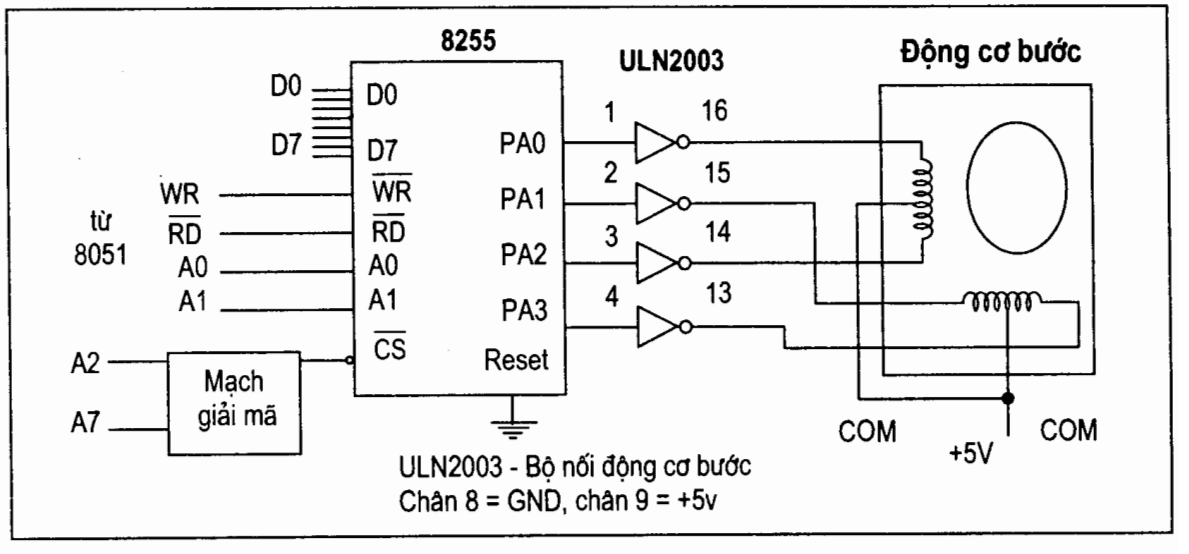
15.2 NỐI GHÉP VỚI 8255

Nối ghép 8255 với động cơ bước

Chương 13 đã đề cập phối ghép động cơ bước với 8051. Ở đây chúng ta tìm hiểu cách nối ghép động cơ bước với 8255 và lập trình, xem hình 15.9.

Chương trình viết cho sơ đồ nối ghép này như sau:

```
      MOV   A,#80H          ;Đặt từ điều khiển để PA là đầu ra
      MOV   R1,#CRPORT      ;Địa chỉ cổng thanh ghi điều khiển
      MOVX  @R1,A           ;Cấu hình cho PA đầu ra
      MOV   R1,#APORT       ;Nạp địa chỉ cổng PA
      MOV   A,#66H          ;A=66H, chuyển xung động cơ bước
AGAIN:
      MOVX  @R1,A           ;Xuất chuỗi ĐK động cơ đến PA
      RR    A               ;Quay theo chiều kim đồng hồ
      ACALL DELAY           ;Chờ
      SJMP  AGAIN
```

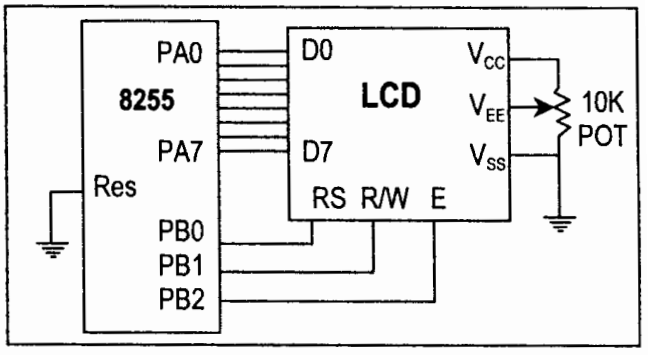


Hình 15.9. Nối ghép 8255 với động cơ bước

Nối ghép 8255 với LCD

Chương trình 15.1 giới thiệu phương pháp xuất lệnh và dữ liệu tới LCD sử dụng 8255 theo sơ đồ hình 15.10. Ở chương trình 15.1 có đặt trễ thời gian trước

mỗi lần xuất thông tin bất kỳ (lệnh hoặc dữ liệu) tới LCD. Cách tốt hơn nếu kiểm tra cờ bận trước khi xuất lện/dữ liệu tới LCD như đã trình bày ở chương 12. Chương trình 15.2 lặp lại chương trình 15.1 với một điểm khác đó là thực hiện kiểm tra cờ bận. Trong trường hợp này không cần thời gian giữ chậm như ở ví dụ 15.1.



Hình 15.10. Nối ghép 8255 với LCD

```
;Ghi lệnh và dữ liệu tới LCD không có kiểm tra cờ bận.
;Giả sử PA của 8255 được nối tới D0-D7 của LCD và
;PB0=RS, PB1=R/W, PB2=E - nối các chân điều khiển LCD
   MOV    A,#80H          ;Đặt tất cả cổng 8255 là đầu ra
   MOV    R0,#CNTPORT     ;Nạp ĐC thanh ghi điều khiển
   MOVX   @R0,A           ;Xuất từ điều khiển
   MOV    A,#38H          ;LCD hai dòng và ma trận 5x7
   ACALL  CMDWRT          ;Gửi lệnh ra LCD
   ACALL  DELAY           ;Chờ đến lần xuất kế tiếp (2ms)
   MOV    A,#0EH          ;Bật con trỏ của LCD
   ACALL  CMDWRT          ;Ghi lệnh ra LCD
   ACALL  DELAY           ;Chờ lần xuất kế tiếp
   MOV    A,#01H          ;Xoá LCD
   ACALL  CMDWRT          ;Gửi lệnh ra LCD
   ACALL  DELAY           ;Chờ lệnh tiếp theo
   MOV    A,#06           ;Dịch con trỏ sang phải
   ACALL  CMDWRT          ;Gửi lệnh này ra LCD
   ACALL  DELAY           ;Chờ lần xuất sau
   ...
   MOV    A,#'N'          ;Hiển thị dữ liệu (chữ N)
   ACALL  DATAWRT         ;Gửi dữ liệu ra LCD để hiển thị
   ACALL  DELAY           ;Chờ lần xuất sau
   MOV    A,#'0'          ;Hiển thị chữ "0"
   ACALL  DATAWRT         ;Gửi ra LCD để hiển thị
```

```
    ACALL  DELAY            ;Chờ lần xuất sau
    ...
;Chương trình con ghi lệnh CMDWRT ra LCD
CMDWRT:
    MOV    R0,#APORT        ;Nạp địa chỉ cổng A
    MOVX   @R0,A            ;Xuất dữ liệu tới LCD
    MOV    R0,# BPORT       ;Nạp địa chỉ cổng B
    MOV    A,# 00000100B    ;RS=0,R/W=1,E=1-tạo  xung  H-TO-L
    MOVX   @R0,A            ;Kích hoạt chân RS,R/W,E của LCD
    NOP                     ;Tạo độ rộng xung ở chân E
    NOP
    MOV    A,# 00000000B    ;RS=0,R/W=1,E=1 tạo xung H-TO-L
    MOVX   @R0,A            ;Chốt thông tin chân dữ liệu LCD
    RET
;Chương trình con hiển thị dữ liệu ra LCD.
CMDWRT:
    MOV    R0,#APORT        ;Nạp địa chỉ cổng A
    MOVX   @R0,A            ;Xuất dữ liệu tới LCD
    MOV    R0,# BPORT       ;RS=1,R/W=0,E=0 tạo xung H-TO-L
    MOV    A,# 00000101B    ;Kích hoạt các chân RS, R/W, E
    MOVX   @R0,A            ;Tạo độ rộng xung ở chân E
    NOP
    NOP
    MOV    A,# 00000001B    ;RS=1,R/W=0,E=0 tạo xung H-TO-L
    MOVX   @RC,A            ;Chốt thông tin của LCD
    RET
```

Chương trình 15.1

```
;Ghi lệnh và dữ liệu tới LCD có sử dụng kiểm tra cờ bận.
;Giả sử PA của 8255 được nối tới D0-D7 của LCD và
;PB0=RS,PB1=R/W,PB2=E nối 8255 tới chân điều khiển LCD
    MOV    A,#80H           ;Đặt các cổng 8255 là đầu ra
    MOV    R0,#CNTPORT      ;Nạp địa chỉ TG điều khiển
    MOVX   @R0,A            ;Xuất từ điều khiển
    MOV    A,#38H           ;LCD có hai dòng và ma trận 5×7
```

```
    ACALL   NMDWRT          ;Ghi lệnh ra LCD
    MOV     A,# 0EH         ;Bật con trỏ LCD
    ACALL   NMDWRT          ;Ghi lệnh ra LCD
    MOV     A,# 01H         ;Xoá LCD
    ACALL   NMDWRT          ;Ghi lệnh ra LCD
    MOV     A,#06           ;Dịch con trỏ sang phải
    ACALL   CMDWRT          ;Ghi lệnh ra LCD
    ...
    MOV     A,#'N'          ;Hiển thị dữ liệu (chữ N)
    ACALL   NCMDWRT         ;Gửi dữ liệu ra LCD để hiển thị
    MOV     A,#'0'          ;Hiển thị chữ "0"
    ACALL   NDADWRT         ;Gửi ra LCD để hiển thị
    ...
;Chương trình con ghi lệnh NCMDWRT có kiểm tra cờ bận
 NCMDWRT:
    MOV     R2, A           ;Lưu giá trị thanh ghi A
    MOV     A,#90H          ;PA=cổng vào
    MOV     R0,# CNTPORT    ;Nạp địa chỉ TG điều khiển
    MOVX    @R0,A           ;Đặt PA đầu vào, PB đầu ra
    MOV     A,#00000110B    ;RS=0, R/W=1, E=1 đọc lệnh
    MOV     @R0,BPORT       ;Nạp địa chỉ cổng B
    MOVX    R0,A            ;RS=0,R/W=1 cho chân RD và RS
    MOV     R0, APORT       ;Nạp địa chỉ cổng A
 READY:
    MOVX    @R0             ;Đọc thanh ghi lệnh
    RLC     A               ;Chuyển cờ bận D7 vào bit nhớ
    JC      READY           ;Chờ cho đến khi LCD sẵn sàng
    MOV     A,#80H          ;Đặt lại PA, PB thành đầu ra
    MOV     R0,#CNTPORT     ;Nạp địa chỉ cổng điều khiển
    MOVX    @R0,A           ;Xuất từ điều khiển tới 8255
    MOV     A,R2            ;Nhận giá trị trả lại  tới LCD
    MOV     R0,#APORT       ;Nạp địa chỉ cổng A
    MOVX    @R0,A           ;Xuất thông tin tới LCD
    MOV     R0,#BPORT       ;Nạp địa chỉ cổng B
    MOV     A,#00000100B    ;RS=0,R/W=0,E=1 tạo xung H-TO-L
    MOVX    @R0, A          ;Kích hoạt RS, R/W, E của LCD
```

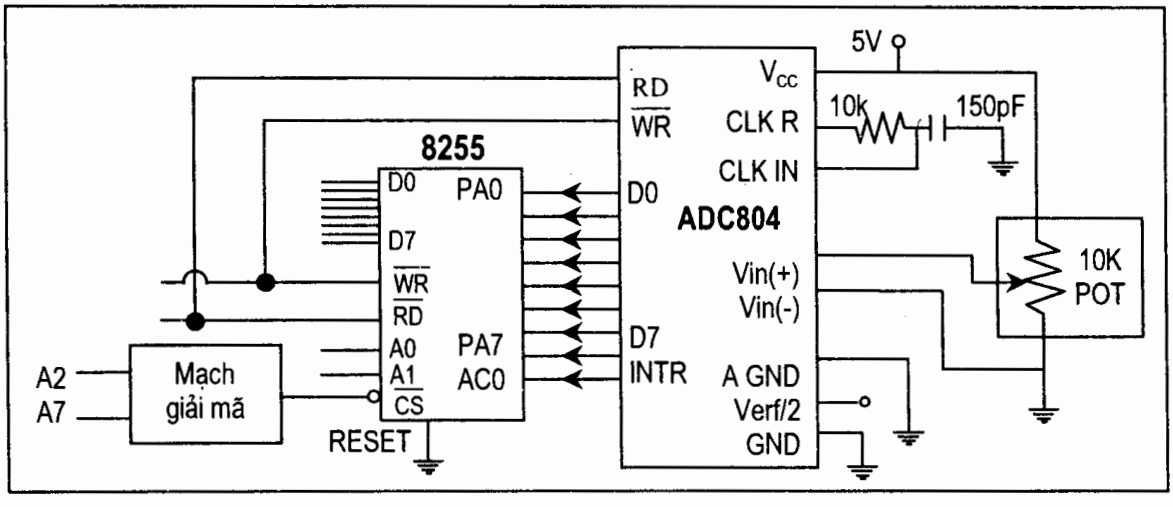
```
    NOP                     ;Tạo độ rộng xung của chân E
    NOP
    MOV     A,#00000000B    ;RS=0,R/W=0,E=0 tạo xung H-TO-L
    MOVX    @R0,A           ;Chốt dữ liệu LCD
    RET
 ;Chương trình con ghi dữ liệu mới có kiểm tra cờ bận
NCMDWRT:
    MOV     R2, A           ;Lưu giá trị thanh ghi A
    MOV     A,#90H          ;Đặt PA là cổng đầu vào
    MOV     R0,#CNTPORT     ;Nạp địa chỉ TG điều khiển
    MOVX    @R0,A           ;Đặt PA đầu vào, PB đầu ra
    MOV     A,#00000110B    ;RS=0, R/W=1, E=1 đọc lệnh
    MOV     @R0,BPORT       ;Nạp địa chỉ cổng B
    MOVX    R0,A            ;RS=0,R/W=1 cho chân RD và RS
    MOV     R0,APORT        ;Nạp địa chỉ cổng A
READY:
    MOVX    A,@R0           ;Đọc thanh ghi lệnh
    RLC     A               ;Chuyển cờ bận D7 vào bit nhớ
    JC      READY           ;Chờ cho đến khi LCD sẵn sàng
    MOV     A,#80H          ;Đặt lại PA, PB thành đầu ra
    MOV     R0,#CNTPORT     ;Nạp địa chỉ cổng điều khiển
    MOVX    @R0,A           ;Xuất từ điều khiển tới 8255
    MOV     A,R2            ;Nhận giá trị trả lại  tới LCD
    MOV     R0,#APORT       ;Nạp địa chỉ cổng A
    MOVX    @R0,A           ;Xuất dữ liệu đến LCD
    MOV     R0,#BPORT       ;Nạp địa chỉ cổng B
    MOV     A,#00000101B    ;RS=1,R/W=0,E=1 tạo xung H-TO-L
    MOVX    @R0,A           ;Kích hoạt RS, R/W, E của LCD
    NOP                     ;Tạo độ rộng xung ở chân E
    NOP
    MOV     A,#00000001B    ;RS=1,R/W=0,E=1 tạo xung H-TO-L
    MOVX    @R0,A           ;Chốt dữ liệu của LCD
    RET
```

Chương trình 15.2

Nối ghép ADC với 8255

Bộ biến đổi ADC đã được trình bày ở chương 12. Dưới đây là chương trình viết cho trường hợp nối ghép ADC với 8255 theo sơ đồ hình 15.11.

```
   MOV     A,#80H          ;PA = đầu ra và PC  = đầu vào
   MOV     R1,#CRPORT      ;Nạp địa chỉ cổng điều khiển
   MOVX    @R1, A          ;Đặt PA = đầu ra và PC = đầu vào
 BACK:
   MOV     R1,#CPORT       ;Nạp địa chỉ cổng C
   MOVX    A,@R1           ;Đọc cổng C xem ADC đã sẵn sàng?
   ANL     A,#00000001B    ;Che tất cả các bit trừ PC0
   JNZ     BACK            ;Kiểm tra PC0 để tìm EOC
;Kết thúc chuyển đổi, bây giờ nhận dữ liệu của ADC
   MOV     R1,#APORT       ;Nạp địa chỉ PA
   MOVX    A,@R1           ;A=đầu vào dữ liệu tương tự
```



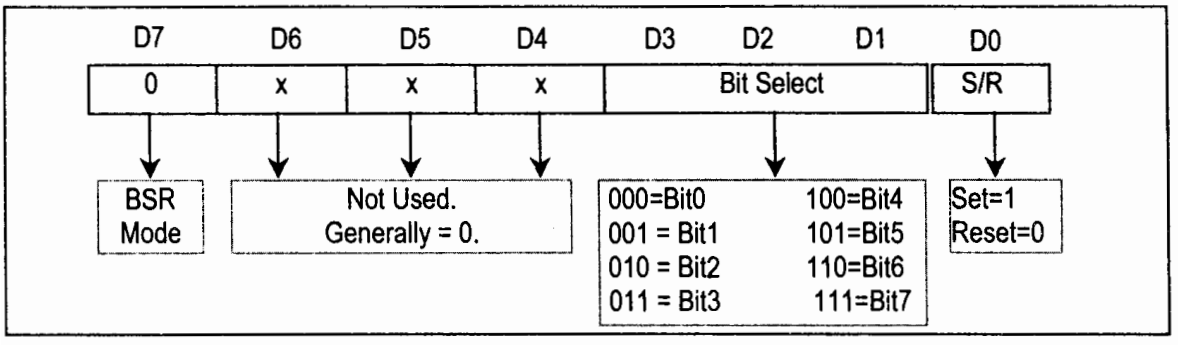
Hình 15.11. Nối ghép ADC804 với 8255

Như vậy, chúng ta đã tìm hiểu chế độ vào/ra đơn giản của 8255. Bây giờ chúng ta sẽ nghiên cứu thêm một số chế độ khác.

15.3 CÁC CHẾ ĐỘ KHÁC CỦA 8255

Chế độ thiết lập/xoá bit BSR

Cổng C có một đặc điểm khác với cổng A và B là có thể điều khiển riêng rẽ các bit. Chế độ BSR cho phép thiết lập từng bit PC0 - PC7. Xem hình 15.12. Hai ví dụ 15.6 và 15.7 giới thiệu cách sử dụng chế độ này.



Hình 15.12. Từ điều khiển chế độ BSR

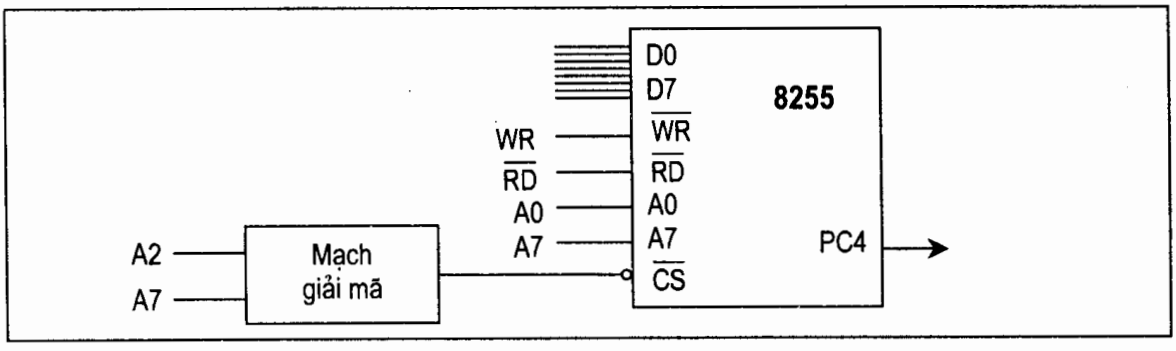
Ví dụ 15.6

Hãy viết chương trình để PC4 của 8255 tạo ra một xung 50 ms và độ đầy xung 50%.

Giải:

Để lập trình PC4 của 8255 ở chế độ BSR thì bit D7 của từ điều khiển phải ở mức thấp. Để PC4 ở mức cao, từ điều khiển phải là "0xxx1001", còn ở mức thấp thì "0xxx1000". Các bit đánh dấu x là không quan tâm, và thường được đặt về 0.

```
MOV     A,00001001B     ;Đặt byte điều khiển cho PC4 =1
MOV     R1,#CNTPORT     ;Nạp cổng thanh ghi điều khiển
MOVX    @R1,A           ;Tạo PC4=1
ACALL   DELAY           ;Thời gian giữ chậm cho xung cao
MOV     A,#00001000B    ;Đặt byte điều khiển cho PC4 = 0
MOVX    @R1,A           ;Tạo PC4=0
ACALL   DELAY
```



Hình 15.13. Sơ đồ mạch cho ví dụ 15.6 và 15.7

Ví dụ 15.7

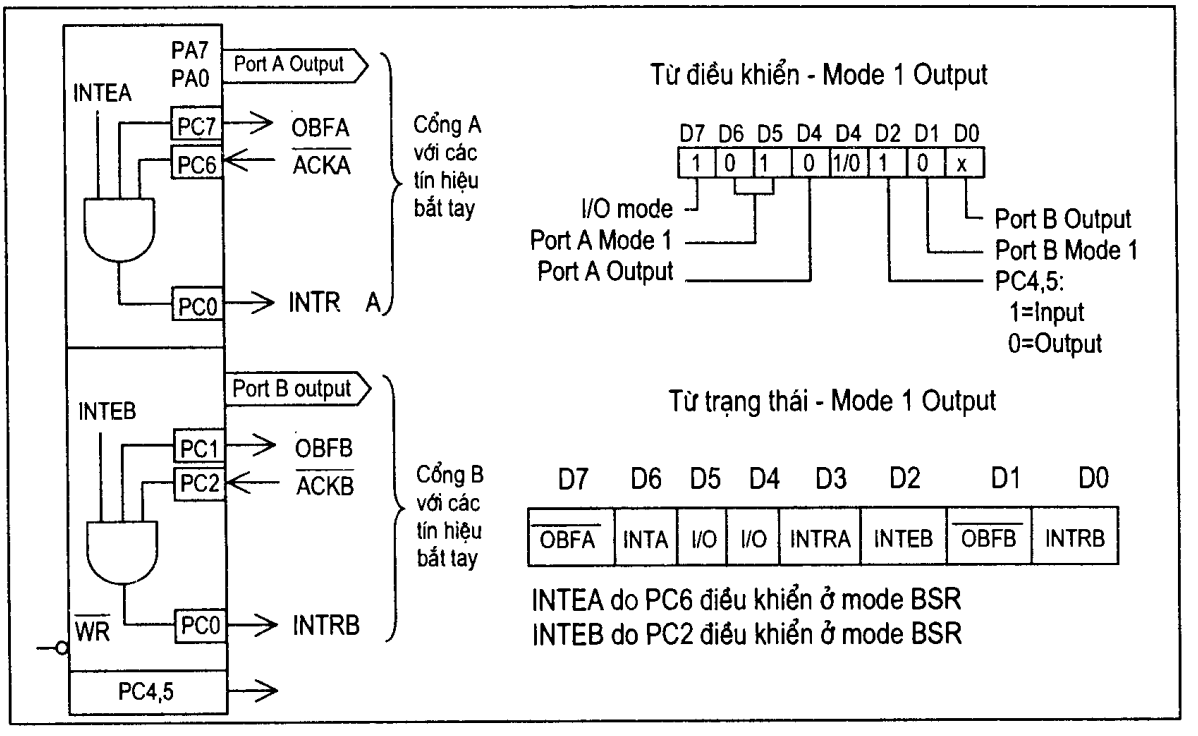
Dựa trên sơ đồ hình 15.13, hãy lập trình 8255 để:

a) Đặt PC2 lên cao.

b) Sử dụng PC6 để tạo xung vuông liên tục với độ đầy xung 66%.

Giải:

```
a)        MOV   R0,#CNTPORT
          MOV   A,#0XXX0101    ;Byte điều khiển
          MOV   @R0,A
b) AGAIN:
          MOV   A,#00001101B   ;Chọn PC6=1
          MOV   R0,#CNTPORT    ;Nạp địa chỉ TG điều khiển
          MOVX  @R0,A          ;Tạo PC6=1
          ACALL DELAY
          ACALL DELAY
          MOV   A,#00001100B   ;PC6=0
          ACALL DELAY          ;Thời gian giữ chậm
          SJMP  AGAIN
```



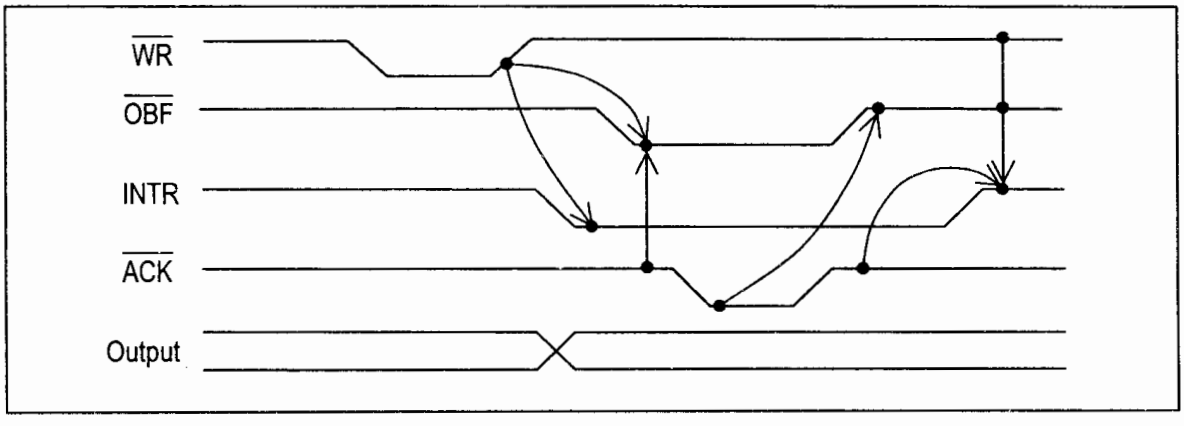
Hình 15.14. Sơ đồ đầu ra chế độ 1 của 8255A

8255 ở chế độ 1: Vào/ra với khả năng này bắt tay

Một trong những điểm mạnh của 8255 là khả năng bắt tay với các thiết bị khác. Khả năng bắt tay là quá trình trao đổi thông tin giữa hai thiết bị thông minh. Ví dụ về thiết bị có sử dụng tín hiệu bắt tay là máy in. Dưới đây chúng ta nghiên cứu các tín hiệu bắt tay của 8255 với máy in.

Chế độ 1: Xuất dữ liệu với tín hiệu bắt tay

Như thấy ở hình 15.14, cổng A và B có thể được dùng làm cổng ra để gửi dữ liệu tới một thiết bị với tín hiệu bắt tay. Tín hiệu bắt tay cho cả hai cổng A và B được các bit của cổng C cấp. Hình 15.15 trình bày biểu đồ định thời của 8255.



Hình 15.15. Biểu đồ thời gian tín hiệu ra chế độ 1

Các tín hiệu bắt tay của cổng A (đối với B hoàn toàn tương tự) như sau:

$\overline{\text{OBFa}}$ (Output Buffer Full for Port A) -đầy bộ đệm ra của cổng A

Là tín hiệu ra mức tích cực thấp của PC7 để báo rằng CPU đã ghi 1 byte dữ liệu tới cổng A. Tín hiệu này phải được nối tới chân STROBE của thiết bị nhận dữ liệu (chẳng hạn như máy in) để báo rằng thiết bị đang sẵn sàng để đọc một byte dữ liệu từ chốt cổng A.

$\overline{\text{ACKa}}$ (Acknowledge for Port A) -báo nhận ở cổng A

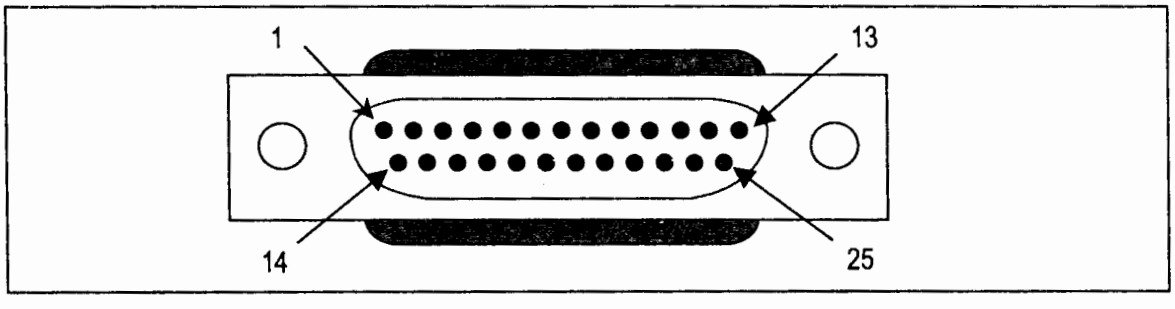
Là tín vào tích cực thấp nhận được ở chân PC6 của 8255. Nhờ tín hiệu $\overline{\text{ACKa}}$ mà 8255 biết rằng tín hiệu tại cổng A đã được thiết bị thu tiếp nhận. Khi đó, 8255 bật $\overline{\text{OBFa}}$ lên cao để báo rằng dữ liệu tại cổng A bây giờ là dữ liệu cũ và khi CPU đã ghi một byte dữ liệu mới tới cổng A thì $\overline{\text{OBFa}}$ lại xuống thấp v.v.

INTRa (Interupt Request For Port A) - yêu cầu ngắt của cổng A

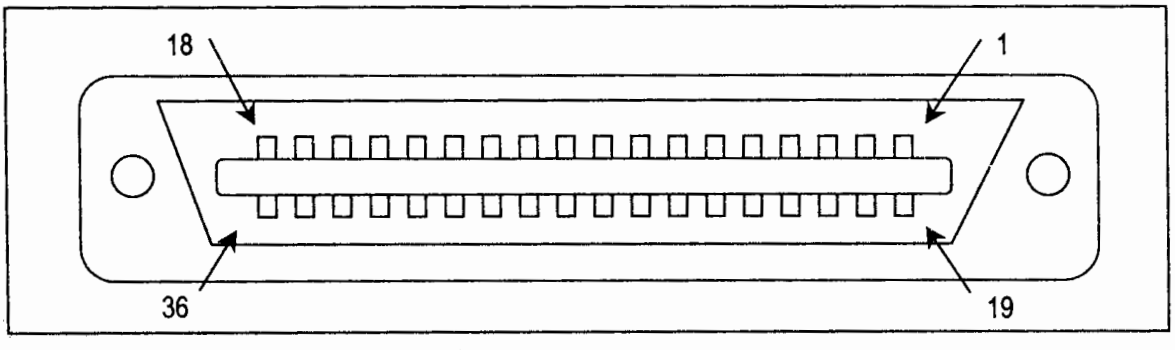
Là tín hiệu yêu cầu ngắt của cổng A có mức tích cực cao đi ra từ chân PC3 của 8255. Tín hiệu $\overline{ACK}$ là tín hiệu có độ dài hạn chế. Khi tín hiệu này xuống thấp (tích cực) thì làm $\overline{OBFa}$ chuyển sang tích cực, duy trì ở mức thấp một thời gian ngắn và sau đó trở nên cao (thôi tích cực). Sườn lên của $\overline{ACK}$ kích hoạt INTRa lên cao và thông qua đó, báo cho CPU biết rằng máy in đã nhận xong một byte và sẵn sàng để nhận byte dữ liệu tiếp theo. INTRa buộc CPU ngừng mọi công việc đang thực hiện để gửi byte tiếp theo tới cổng A để in. Cần chú ý rằng INTRa được bật lên 1 chỉ khi đồng thời INTRa, $\overline{OBFa}$ và $\overline{ACKa}$ đều ở mức cao và chân này bị xoá về 0 khi CPU vừa ghi một byte tới cổng A.

INTEa (Interrup Enable for Port A) - cho phép ngắt cổng A

8255 có thể cấm INTRa để ngăn không cho nó ngắt CPU. Đây là chức năng của tín hiệu INTEa. INTEa là một mạch lật Flip - Flop bên trong, được thiết kế để che (cấm) INTRa. Tín hiệu INTRa có thể được bật lên hoặc bị xoá qua cổng C ở chế độ BSR vì INTEa là Flip - Plop được điều khiển bởi PC6.



Hình 15.16. Ổ cắm DB-25



Hình 15.17. Đầu cáp máy in Centronics 36 chân

Từ trạng thái

8255 cho phép giám sát trạng thái của các tín hiệu INTR, OBF và INTE ở cả 2 cổng A và B. Cách thực hiện là đọc cổng C vào thanh ghi tích luỹ và kiểm tra các bit, và như vậy cho phép thực hiện giải pháp thăm dò thay cho giải pháp ngắt phần cứng.

Tín hiệu máy in

Để hiểu được quá trình bắt tay với 8255, chúng ta xem lại các tín hiệu bắt tay. Các bước truyền thông có bắt tay giữa máy in và 8255 như sau:

1. Một byte dữ liệu được gửi đến bus dữ liệu máy in.
2. Máy in được báo có 1 byte dữ liệu cần được in bằng cách kích hoạt tín hiệu đầu vào STROBE của nó.
3. Khi máy in nhận được dữ liệu thì nó báo bên gửi bằng cách kích hoạt tín hiệu đầu ra $\overline{\text{ACK}}$ (báo đã nhận).
4. Tín hiệu $\overline{\text{ACK}}$ khởi tạo quá trình cấp byte tiếp theo đến máy in.

Từ các bước trên có thể thấy rằng, nếu chỉ riêng dữ liệu tới máy in thôi là không đủ. Trước hết, máy in cần được thông báo về sự hiện diện của dữ liệu. Ngoài ra, khi dữ liệu đã được gửi đi nhưng máy in có thể đang bận hoặc hết giấy, do vậy máy in phải báo cho bên gửi biết trạng thái sẵn sàng nhận dữ liệu. Hình 15.16 và 15.17 trình bày ổ cắm và đầu cáp Centronic DB25 của máy in.

Bảng 15.2. Bố trí chân đầu cắm máy in DB25

Chân	Mô tả	Chân	Mô tả
1	STROBE	11	Bận
2	Dữ liệu D0	12	Hết giấy
3	Dữ liệu D1	13	Chọn
4	Dữ liệu D2	14	Tự nạp
5	Dữ liệu D3	15	
6	Dữ liệu D4	16	
7	Dữ liệu D5	17	
8	Dữ liệu D6	18	
9	Dữ liệu D7	19	
10	$\overline{\text{ACK}}$ (chấp nhận)	18-25	Đất (ground)

PHỤ LỤC

PHỤ LỤC A. TẬP LỆNH CỦA 8051

Bảng A.1. Tổng hợp tập lệnh của 8051

Mã lệnh		Byte	MC
Lệnh số học			
ADD	A,Rn	1	1
ADD	A,trtiếp	2	1
ADD	A,@Ri	1	1
ADD	A,#dliệu	2	1
ADDC	A,Rn	1	1
ADDC	A,trtiếp	2	1
ADDC	A,@Ri	1	1
ADDC	A,#dliệu	2	1
SUBB	A,Rn	1	1
SUBB	A,trtiếp	2	1
SUBB	A,@Ri	1	1
SUBB	A,#dliệu	2	1
INC	A	1	1
INC	Rn	1	1
INC	Trtiếp	2	1
INC	@Ri	1	1
DEC	A	1	1
DEC	Rn	1	1
DEC	Trtiếp	2	1
DEC	@Ri	1	1
INC	DPTR	1	2
MUL	AB	1	4
DIV	AB	1	4
DA	A	1	1

Mã lệnh		Byte	MC
Truyền dữ liệu			
MOV	A,Rn	1	1
MOV	A,trtiếp	2	1
MOV	A,@Ri	1	1
MOV	A,#dliệu	2	1
MOV	Rn,A	1	1
MOV	Rn,trtiếp	2	2
MOV	Rn,#dliệu	2	1
MOV	Trtiếp,A	2	1
MOV	trtiếp,Rn	2	2
MOV	trtiếp,trt	2	2
MOV	trtiếp,@Ri	2	2
MOV	trtiếp,#dl	3	2
MOV	@Ri,A	1	1
MOV	@Ri,trtiếp	2	2
MOV	@Ri,#dliệu	2	1
MOV	PDTR,#dliệu	3	2
MOVX	A,@Ri	1	2
MOVX	A,@DPTR	1	2
MOVX	@Ri,A	1	2
MOV	@DPTR,A	1	2
PUSH	trtiếp	2	2
POP	trtiếp	2	2
XCH	A,Rn	1	1
XCH	A,trtiếp	2	1
XCH	A,@Ri	1	1
XCHD	A,@Ri	1	1

Lệnh logic			
ANL	A,Rn	1	1
ANL	A,trtiếp	2	1
ANL	A,@Ri	1	1
ANL	A,#dữliệu	2	1
ANL	trtiếp,A	2	1
ANL	trtiếp,dl	3	2
ORL	A,Rn	1	1
ORL	A,trtiếp	2	1
ORL	A,@Ri	1	1
ORL	A,#dữliệu	2	1
ORL	trtiếp,A	2	1
ORL	trtiếp,dl	3	2
XRL	A,Rn	1	1
XRL	A,trtiếp	2	1
XRL	A,@Ri	1	1
XRL	A,#dữliệu	2	1
XRL	trtiếp,A	2	1
XRL	trtiếp,dl	3	2
CLR	A	1	1
CPL	A	1	1
RL	A	1	1
RLC	A	1	1
RR	A	1	1
RRC	A	1	1
SWAP	A	1	1

Xử lý biến Boolean			
CLR	C	1	1
CLR	bit	2	1
SETB	C	1	1
SETB	bit	2	1
CPL	C	1	1
CPL	bit	2	1
ANL	C,bit	2	2
ANL	C,/bit	2	2
ORL	C,bit	2	2
ORL	C,/bit	2	2
MOV	C.bit	2	1
MOV	bit,C	2	2
JC	rel	2	2
JNC	rel	2	2
JB	bit,rel	3	2
JNB	bit,rel	3	2
JBC	bit,rel	3	2

Lệnh rẽ nhánh			
ACALL	đchỉ11	2	2
LCALL	đchỉ16	3	2
RET		1	2
RETI		1	2
AJMP	đchỉ11	2	2
LJMP	đchỉ16	3	2
SJMP	rel	2	2
JMP	@A+DPTR	1	2
JZ	rel	2	2

Lệnh rẽ nhánh			
JNZ	rel	2	2
CJNE	A,trtiếp,rel	3	2
CJNE	A,#dliệu,rel	3	2
CJNE	Rn,dliệu,rel	3	2
DJNZ	Rn,rel	3	2
DJNZ	Rn,rel	2	2
DJNZ	trtiếp,rel	3	2
NOP		1	2

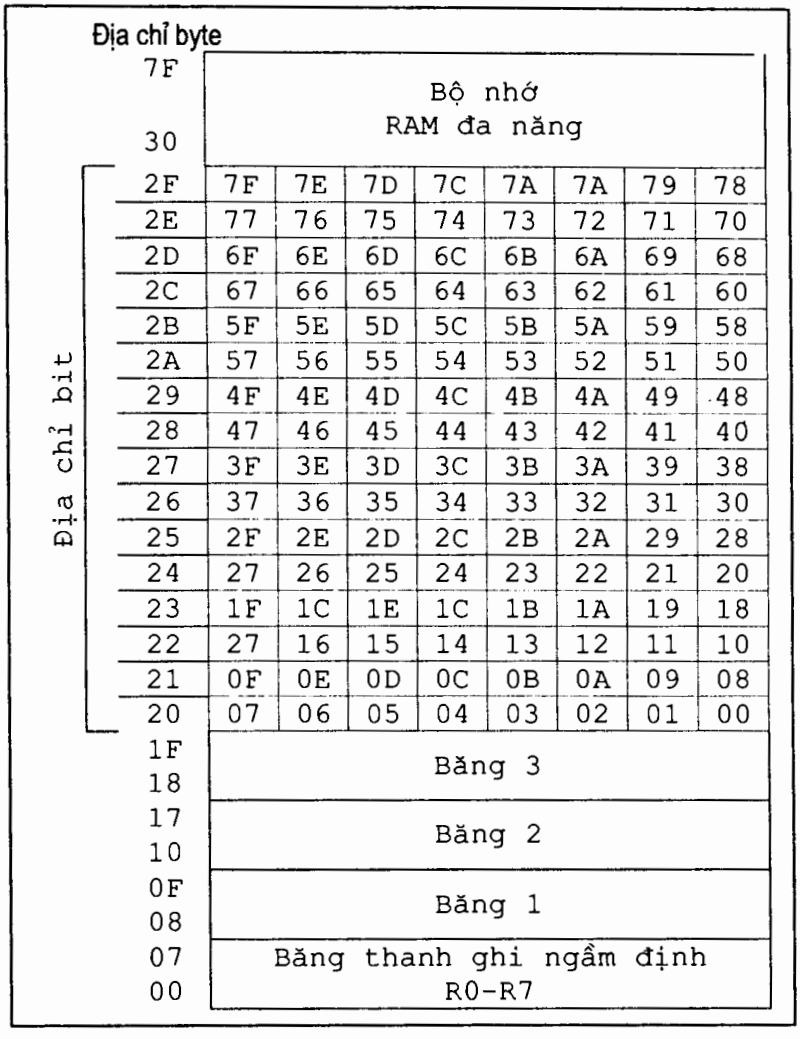
Bảng A.2. Địa chỉ các thanh ghi chức năng đặc biệt SFR

Lệnh	Tên	Địa chỉ
ACC*	Thanh ghi tích luỹ (thanh ghi tổng) A	0E0H
B*	Thanh ghi B	0F0H
PSW*	Từ trạng thái chương trình	0D0H
SP	Con trỏ ngăn xếp	81H
DPTR	Con trỏ dữ liệu hai byte	
DPL	Byte thấp của DPTR	82H
DPH	Byte cao của DPTR	83H
P0*	Cổng 0	80H
P1*	Cổng 1	90H
P2*	Cổng 2	0A0H
P3*	Cổng 3	0B0H
IP*	Điều khiển ưu tiên ngắt	0B8H
IE*	Điều khiển cho phép ngắt	0A8H
TMOD	Điều khiển chế độ bộ đếm/ Bộ định thời	89H
TCON*	Điều khiển bộ đếm/ Bộ định thời	88H
T2CON*	Điều khiển bộ đếm/ Bộ định thời 2	0C8H
T2MOD	Điều khiển chế độ bộ đếm/ Bộ định thời 2	0C9H
TH0	Byte cao của bộ đếm/ Bộ định thời 0	8CH
TL0	Byte thấp của bộ đếm/ Bộ định thời 0	8AH
TH1	Byte cao của bộ đếm/ Bộ định thời 1	8DH
TL1	Byte thấp của bộ đếm/ Bộ định thời 1	8BH
TH2	Byte cao của bộ đếm/ Bộ định thời 2	0CDH
TL2	Byte thấp của bộ đếm/ Bộ định thời 2	0CCH
RCAP2H	Byte cao của thanh ghi bộ đếm/ Bộ định thời 2	0CBH
RCAP2L	Byte thấp của thanh ghi bộ đếm/ Bộ định thời 2	0CAH
SCON*	Điều khiển nối tiếp	98H
SBUF	Bộ đệm dữ liệu nối tiếp	99H
PCON	Điều khiển công suất	87H

Địa chỉ byte	Địa chỉ bit								
FF									
F0	E7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	D1	D0	PSW
B8	--	--	--	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	F3
A8	AF	--	--	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Không định địa chỉ bit								SBUF
98	9F	9E	9D	9C	9B	9A	99	99	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Không định địa chỉ bit								TH1
8C	Không định địa chỉ bit								TH0
8B	Không định địa chỉ bit								TL1
8A	Không định địa chỉ bit								TL0
89	Không định địa chỉ bit								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Không định địa chỉ bit								PCON
83	Không định địa chỉ bit								DPH
82	Không định địa chỉ bit								DPL
81	Không định địa chỉ bit								SP
80	87	86	85	84	83	82	81	80	P0

Các thanh ghi chức năng đặc biệt

Hình A.1. Địa chỉ byte và bit bộ nhớ RAM của các thanh ghi chức năng đặc biệt



Hình A.2. 128 byte của RAM trong

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

EA	IE.7	Nếu EA = 0 thì mọi ngắt bị cấm Nếu EA = 1 thì từng nguồn ngắt sẽ được phép hoặc bị cấm bằng cách bật hoặc xoá bit cho phép tương ứng.
- -	IE.6	Dự phòng cho tương lai
ET2	IE.5	Cho phép hoặc cấm ngắt tràn hoặc thu của Timer 2 (8952)
ES	IE.4	Cho phép hoặc cấm ngắt cổng nối tiếp
ET1	IE.3	Cho phép hoặc cấm ngắt tràn của Timer 1
EX1	IE.2	Cho phép hoặc cấm ngắt ngoài 1
ET0	IE.1	Cho phép hoặc cấm ngắt tràn của Timer 0
EX0	IE.0	Cho phép hoặc cấm ngắt ngoài 0

* Người dùng không phải ghi 1 vào các bit dự phòng. Các bit này có thể dùng cho các bộ vi điều khiển nhanh có đặc tính mới trong tương lai.

Hình A.3. Thanh ghi cho phép ngắt IE

D7							D0
--	--	PT2	PS	PT1	PX1	PT0	PX0

Bit ưu tiên = 1 là mức ưu tiên cao, bit ưu tiên = 0 là mức ưu tiên thấp.

-	IP.7	Dự trữ
-	IP.6	Dự trữ
PT2	IP.5	Bit ưu tiên ngắt Timer 2 (chỉ dùng cho 8052)
PS	IP.4	Bit ưu tiên ngắt cổng nối tiếp
PT1	IP.3	Bit ưu tiên ngắt Timer 1
PX1	IP.2	Bit ưu tiên ngắt ngoài 1
PT0	IP.1	Bit ưu tiên ngắt Timer 0
PX0	IP.0	Bit ưu tiên ngắt ngoài 0

Người dùng không được viết phần mềm ghi 1 vào các bit chưa dùng vì chúng dành cho các ứng dụng tương lai.

Hình A.4. Thanh ghi mức ưu tiên ngắt IP

SMOD	--	--	--	GF1	GF0	PD	IDL

Hình A.5. Thanh ghi BCON (không định địa chỉ bit)

Cách tính giá trị TH1 cho các tốc độ baud khác nhau:

SMOD=0 (ngầm định khi Reset)

TH1=256 - Tần số thạch anh/(384 x tốc độ baud)

SMOD = 1

TH1=256 - Tần số thạch anh/(192 x tốc độ baud)

CY	AC	F0	RS1	RS0	OV	–	P

CY	PSW.7	; Cờ nhớ
AC	PSW.6	; Cờ phụ
--	PSW.5	; Dành cho người dùng sử dụng mục đích chung
RS1	PSW.4	; Bit = 1 chọn băng thanh ghi
RS0	PSW.3	; Bit = 0 chọn băng thanh ghi
OV	PSW.2	; Cờ tràn
--	PSW.1	; Bit dành cho người dùng định nghĩa
P	PSW.0	; Cờ chẵn, lẻ. Thiết lập/xoá bằng phần cứng

RS1	RS0	Băng thanh ghi	Địa chỉ
0	0	0	00H - 07H
0	1	1	08H - 0FH
1	0	2	10H - 17H
1	1	3	18H - 1FH

Hình A.6. Thanh ghi từ trạng thái chương trình PSW (định địa chỉ bit)

SM0	SM1	SM2	REN	TB8	RB8	T1	R1

SM0	SCON.7	Số xác định chế độ làm việc cổng nối tiếp
SM1	SCON.6	Số xác định chế độ làm việc cổng nối tiếp
SM2	SCON.5	Dùng cho truyền thông giữa các bộ vi xử lý (SM2 = 0)
REN	SCON.4	Bật/xoá bằng phần mềm để cho phép/ không cho thu
TB8	SCON.3	Không sử dụng rộng rãi
RB8	SCON.2	Không sử dụng rộng rãi
T1	SCON.1	Cờ ngắt truyền. Đặt bằng phần cứng khi bắt đầu bit Stop ở Mode 1. Xoá bằng phần mềm
R1	SCON.0	Cờ ngắt thu. Đặt bằng phần cứng khi bắt đầu bit Stop ở Mode 1. Xoá bằng phần mềm.

Hình A.7. Thanh ghi điều khiển cổng nối tiếp SCON (định địa chỉ bit)

(MSB) (LSB)

GATE	C/T	M1	M0	GATE	C/T	M1	M0
Timer 1				Timer 0			

GATE	Điều khiển cổng khi được thiết lập. Bộ định thời/bộ đếm được mở chỉ khi chân INTx cao và chân điều khiển TRx được lập. Nếu GATE được xoá, bộ định thời được mở khi TRx được lập
C/T	Bộ định thời hoặc bộ đếm bị xoá khi hoạt động (đầu vào từ đồng hồ hệ thống trong). Thiết lập hoạt động bộ đếm (đầu vào từ chân vào Tx)
M1	Chế độ bit 1
M0	Chế độ bit 0

M1	M0	Mode	Chế độ hoạt động
0	0	0	Chế độ bộ định thời 13 bit Bộ định thời/bộ đếm 8 bit, định tỷ lệ trước 5 bit
0	1	1	Chế độ bộ định thời 16 bit, không định tỷ lệ trước
1	0	2	Chế độ 8 bit tự nạp lại Bộ đếm/bộ định thời 8 bit tự nạp lại. THx lưu giá trị sẽ tự nạp vào TLx mỗi khi tràn
1	1	3	Chế độ bộ định thời chia tách

Hình A.8. Thanh ghi TMOD

PHỤ LỤC C. BỘ MÃ ASCII

CTR	Dec	Hex	Ch	Code
^@	0	00		NUL
^A	1	01	☺	SOH
^B	2	02	☻	STX
^C	3	03	♥	ēT
^D	4	04	♦	EOT
^E	5	05	♣	ENQ
^F	6	06	♠	ACK
^G	7	07	•	BEL
^H	8	08	◘	BS
^I	9	09	○	HT
^J	10	0A	◙	LF
^K	11	0B	♂	VT
^L	12	0C	♀	FF
^M	13	0D	♪	CR
^N	14	0E	♫	SO
^O	15	0F	☼	SI
^P	16	10	►	DLE
^Q	17	11	◄	DC1
^R	18	12	↕	DC2
^S	19	13	!!	DC3
^T	20	14	¶	DC4
^U	21	15	§	NAK
^V	22	16	▬	SYN
^W	23	17	↨	ETB
^X	24	18	↑	CAN
^Y	25	19	↓	EM
^Z	26	1A	→	SUB
^[	27	1B	←	ES
^\	28	1C	∟	FS
^]	29	1D	↔	GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Ch
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	`
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Ch
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	'
95	5F	_

Dec	Hex	Ch
96	60	'
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	-
127	7F	^

TÀI LIỆU THAM KHẢO

1. Nguyễn Tăng Cường, Phan Quốc Thắng, Vũ Hữu Nghị. Máy tính - Cấu trúc và lập trình. Nhà xuất bản Khoa học và Kỹ thuật, 2003.
2. Myke Predko. Programming & Customizing the 8051 Microcontroller. McGraw - Hill, 1999.
3. Michael Predko, Mike Predko. Handbook of Microcontrollers. McGraw-Hill, 1998.
4. James W. Stewart. 8051 Microcontroller. Hardware, Software, & Interfacing. Prentice Hall, 1998.
5. Kenneth J. Ayala. 8051 Microcontroller: Architecture, Programming & Applications. Delmar Learning, 1996.
6. Scott MacKenzie. The 8051 Microcontroller. Prentice Hall, 1998.
8. Thomas W. Schultz. C and the 8051: Building Efficient Applications. Prentice Hall, 1999.
9. William Kleitz. Microprocessor and Microcontroller Fundamentals: The 8085 and 8051 Hardware and Software. Pearson Education, 1997.
10. Barry B. Brey. Intel Microprocessors 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, and Pentium Pro Processor: Architecture, Programming, and Interfacing. Prentice Hall, 1999.
11. Sencer Yeralan, Helen Emery. Programming and Interfacing the 8051 Microcontroller in C and Assembly. Rigel Corporation, 2000.

NGUYỄN TĂNG CƯỜNG, PHAN QUỐC THẮNG

CẤU TRÚC VÀ LẬP TRÌNH
HỌ VI ĐIỀU KHIỂN 8051

Chịu trách nhiệm xuất bản: PGS. TS. TÔ ĐĂNG HẢI
Biên tập: ĐỖ TIẾN KHANG, NGUYỄN KIM ANH
Vẽ bìa: HƯƠNG LAN

NHÀ XUẤT BẢN KHOA HỌC VÀ KỸ THUẬT
70 Trần Hưng Đạo, Hà Nội

In 1800 cuốn, khổ 19 x 26,5 cm, tại Công ty In và Văn hóa phẩm.
Số giấy phép 113-266, cấp ngày 27/1/2003.
In xong và nộp lưu chiểu Quý I/2004.